VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A6a: Time Series Analysis

FERAH SHAN SHANAVAS RABIYA

V01101398

Date of Submission: 22-07-2024

CONTENTS

Introduction

This report provides a comprehensive analysis and forecasting of Arvind Fashions Limited's stock price, a prominent player in the Indian fashion retail industry. The analysis is divided into several stages, including data collection and cleaning, exploratory data analysis, monthly conversion and decomposition, and multivariate forecasting using machine learning models. The data is downloaded using the `yfinance` library, cleaned to address missing values and outliers, and the dataset is split into training and testing sets for evaluation.

The daily stock price data is converted to a monthly frequency and time series decomposition is performed using both additive and multiplicative models to separate the series into trend, seasonal, and residual components. Conventional models like Holt-Winters, ARIMA, and SARIMA are fitted to the data for Univariate Forecasting, while machine learning models like Long Short-Term Memory (LSTM) neural networks and Tree-based models like Random Forest and Decision Tree are used for Multivariate Forecasting.

The results from these analyses provide insights into Arvind Fashions Limited's stock price behavior, highlighting patterns, trends, and potential future movements. The report aims to demonstrate the application of both traditional statistical methods and modern machine learning techniques in time series forecasting, offering a comparative perspective on their effectiveness in predicting stock prices.

Objectives

- Download historical stock price data for Arvind Fashions Limited using the `yfinance` library. Clean the data to ensure accuracy and reliability.
- Visualize historical stock price data through a line graph. Create training and testing datasets for model evaluation and validation.
- Convert daily stock price data into monthly frequency data. Decompose time series data into components (trend, seasonal, residual) using both additive and multiplicative models.
- Fit a Holt-Winters model to stock price data and forecast future prices. Apply an ARIMA model to daily and monthly stock price data.
- Develop a Long Short-Term Memory (LSTM) neural network model for future stock price forecasting. Apply tree-based models for stock price prediction.

- Compare forecasting accuracy and effectiveness of conventional statistical models and modern machine learning techniques. Provide actionable insights for informed stock decisions.

Business Significance

Arvind Fashions Limited's stock price analysis and forecasting hold significant business significance, providing critical insights and strategic advantages to various stakeholders. Accurate forecasting models enable investors to make informed decisions about buying, holding, or selling Arvind Fashions Limited's stock, maximizing returns and minimizing risks. This helps in risk management by identifying trends and potential future movements, enabling better portfolio diversification and adjusting strategies to mitigate risks.

Financial planning and strategy are also significantly influenced by stock price analysis. Forecasted stock prices aid in accurate budgeting and financial forecasting, aiding in strategic capital allocation and resource management. Understanding stock price trends helps in accurately valuing the company and projecting its growth, essential for mergers, acquisitions, and other corporate finance activities.

Corporate strategy and decision-making can be influenced by stock price analysis, influencing timing for new product launches, market expansion, or strategic partnerships. Performance evaluation provides a benchmark for evaluating the effectiveness of corporate strategies and management decisions.

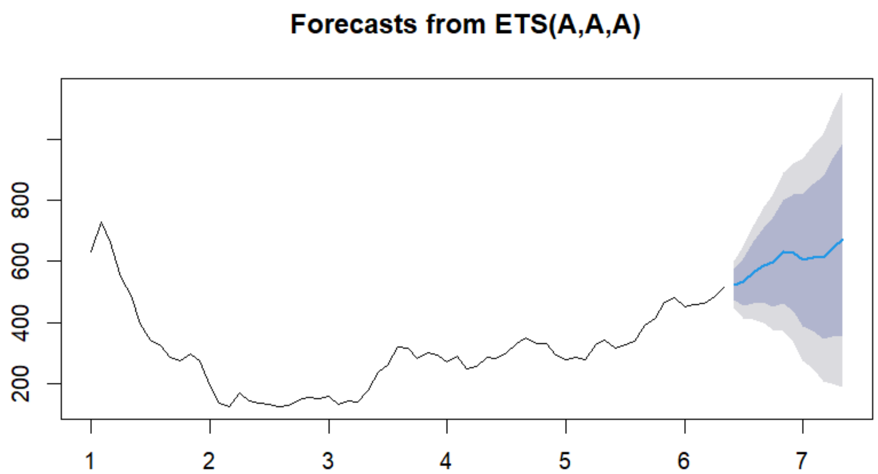
Market analysis and competitive positioning are also crucial for the company. Understanding market sentiment and investor confidence helps gauge public perception and adapt strategies accordingly. Accurate forecasting models contribute to market integrity by providing reliable data, reducing the likelihood of market manipulation and enhancing investor trust.

Stakeholder confidence and communication are enhanced by transparency and effective communication with shareholders, analysts, and the broader financial community. Operational efficiency is also enhanced by understanding future stock price movements, optimizing resource allocation, and allowing for better cost management. Overall, the comprehensive analysis and forecasting of Arvind Fashions Limited's stock prices provide a foundation for strategic planning, risk management, and informed decision-making, contributing significantly to the company's long-term success and stability.

Results and Interpretation using R

- **Fit a Holt Winters model to the data and plot the forecast for the next year**

```
#Holt Winters Model
>
# Load required libraries
> library(forecast)
# Convert the Month column to a date format
> data_monthly$Month <- as.Date(paste0(data_monthly$Month, "-01"))
# Create a ts object with a frequency of 12 (for monthly data)
> data_monthly_ts <- ts(data_monthly$Close, start = start(data_monthly$
Month), frequency = 12)
# Fit a Holt Winters model to the data
> model_hw <- ets(data_monthly_ts, model = "AAA")
> summary(model_hw)
ETS(A,A,A)

Call:
ets(y = data_monthly_ts, model = "AAA")

  Smoothing parameters:
    alpha = 0.9998
    beta  = 0.1336
    gamma = 2e-04

  Initial states:
    l = 607.0284
    b = -2.696
    s = 19.3 36.9 16.3 20.1 9.56 -8.4
           -3.93 -0.828 -17.4 -34.5 -19.7 -17.4

  sigma:  39.8

 AIC AICc  BIC
 766  779  803

Training set error measures:
               ME RMSE  MAE  MPE MAPE  MASE  ACF1
Training set 1.84 34.5 25.5 1.83 9.56 0.193 0.342
# Forecast for the next year
> forecast_hw <- forecast(model_hw, h = 12)
> plot(forecast_hw)
```



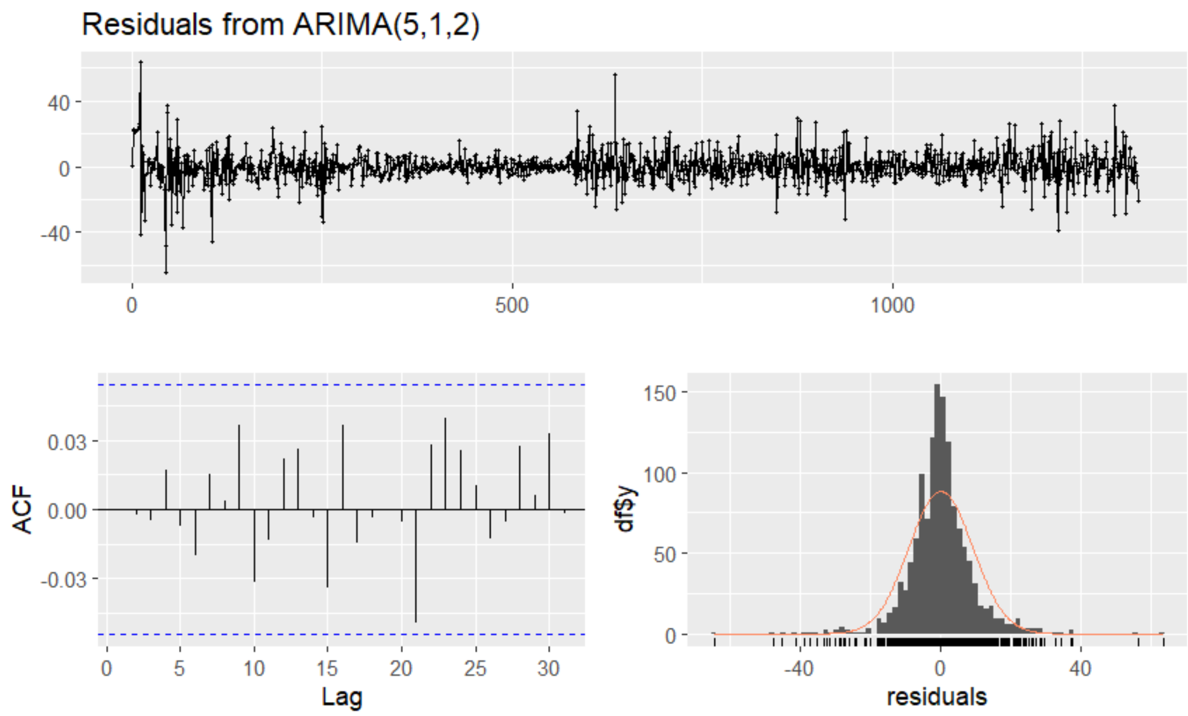
Interpretation:

The Holt-Winters model is a statistical tool used to predict stock prices for the next 12 months . It uses parameters such as the additive error term, additive trend, and additive seasonality. The model's initial states are level, trend, and seasonal components. The accuracy of the model is measured using the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC). The training set error measures include mean error, root mean squared error, mean absolute error, mean percentage error, mean absolute percentage error, mean absolute scaled error, and first-order autocorrelation of residuals.

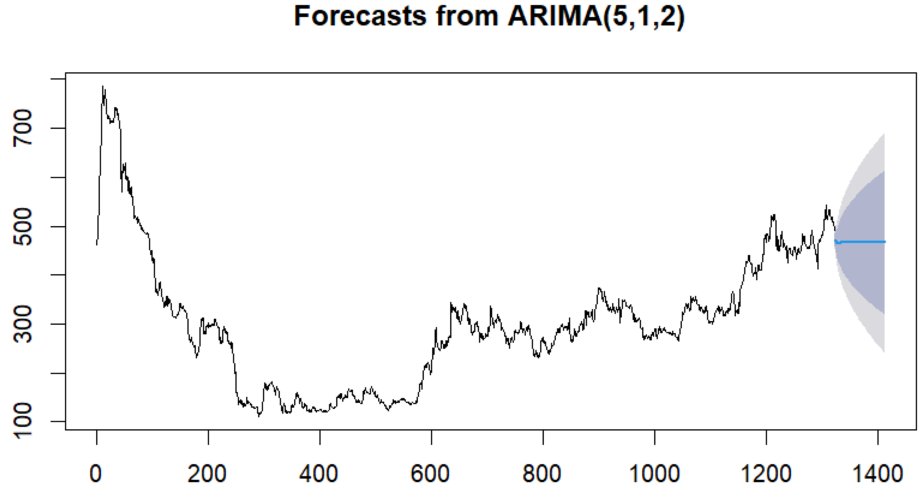
The forecast plot generated by `forecast(model_hw, h = 12)` shows the predicted stock prices for the next 12 months (1 year) based on the Holt-Winters model. The shaded area around the forecasted values represents the 80% and 95% confidence intervals, providing a range for actual values. The forecast plot visually assesses the expected future stock price trends, allowing us to observe that the stock price is expected to increase over the next year, along with the level of uncertainty associated with these predictions.

- **Fit a ARIMA model to the daily and monthly data and plot the forecast for the next year**

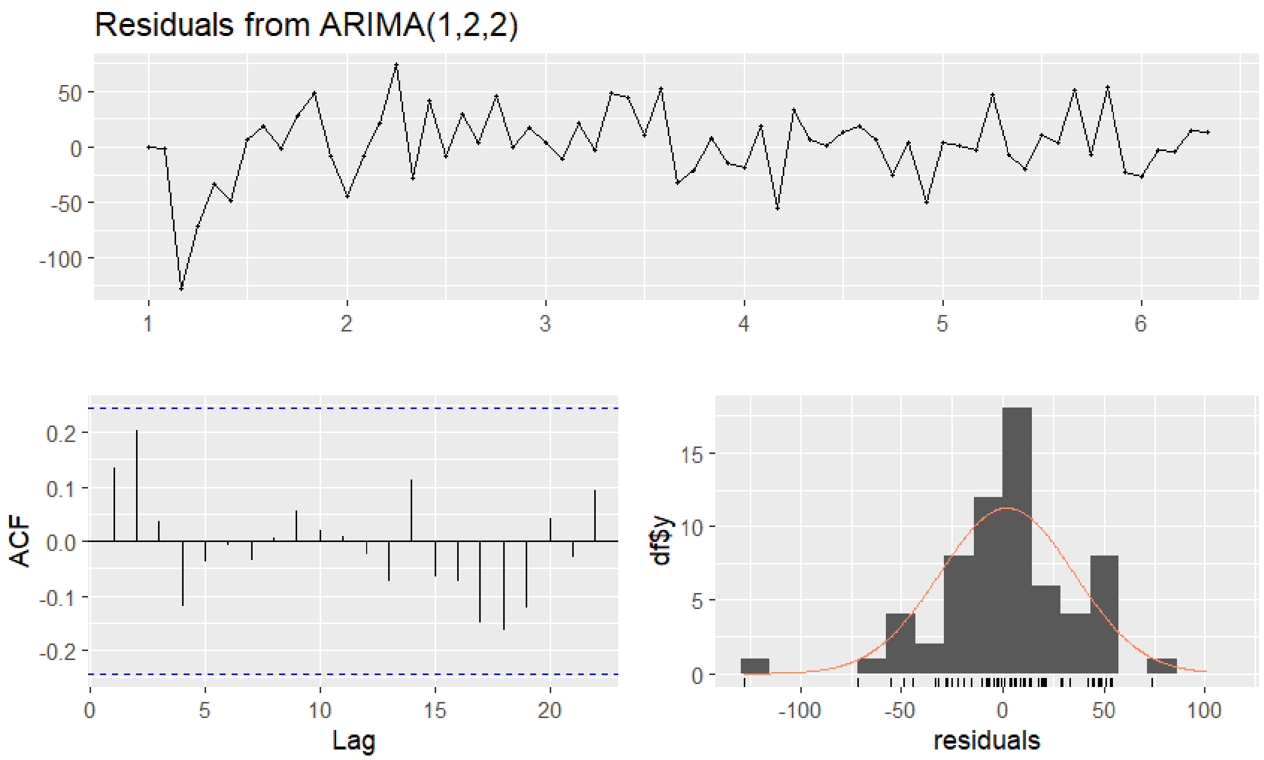
```
# Fit an ARIMA model to the daily data
> model_arima_daily <- auto.arima(data$close)
> summary(model_arima_daily)
Series: data$close
ARIMA(5,1,2)

Coefficients:
       ar1     ar2     ar3    ar4     ar5     ma1    ma2
      1.43  -0.769  -0.021  0.119  -0.039  -1.349  0.727
s.e.  0.13   0.173   0.053  0.050   0.040   0.127  0.161

sigma^2 = 83.2:  log likelihood = -4795
AIC=9605   AICc=9605   BIC=9647

Training set error measures:
                  ME RMSE  MAE     MPE MAPE  MASE     ACF1
Training set 0.00197 9.09 6.07 -0.0294  2.1 0.999 0.000322
# Diagnostic check
> checkresiduals(model_arima_daily)

	Ljung-Box test

data:  Residuals from ARIMA(5,1,2)
Q* = 4, df = 3, p-value = 0.2

Model df: 7.   Total lags used: 10
```



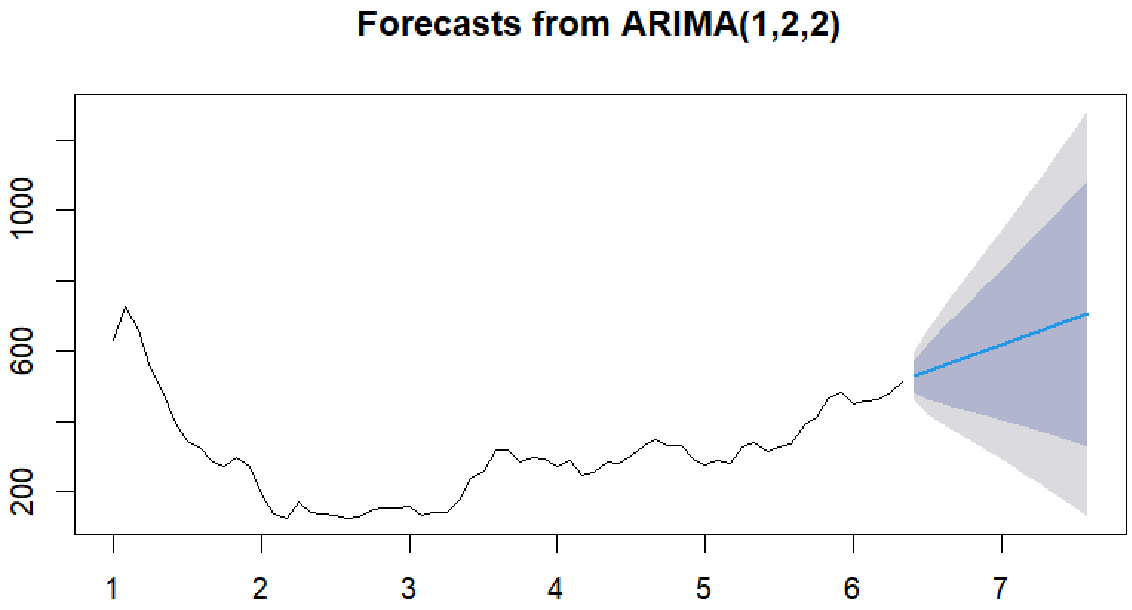

```
# Fit a Seasonal-ARIMA (SARIMA) model
> model_sarima_daily <- auto.arima(data$close, seasonal = TRUE)
> summary(model_sarima_daily)
Series: data$close
ARIMA(5,1,2)

Coefficients:
        ar1     ar2     ar3    ar4     ar5     ma1    ma2
       1.43  -0.769  -0.021  0.119  -0.039  -1.349  0.727
s.e.   0.13   0.173   0.053  0.050   0.040   0.127  0.161

sigma^2 = 83.2:  log likelihood = -4795
AIC=9605   AICc=9605   BIC=9647

Training set error measures:
                  ME RMSE  MAE     MPE MAPE  MASE     ACF1
Training set 0.00197 9.09 6.07 -0.0294  2.1 0.999 0.000322
# Forecast for the next three months
> forecast_arima_daily <- forecast(model_sarima_daily, h = 90)
> plot(forecast_arima_daily)
```



```
#ARIMA Model (Monthly Data)
>
# Fit an ARIMA model to the monthly data
> model_arima_monthly <- auto.arima(data_monthly_ts)
> summary(model_arima_monthly)
Series: data_monthly_ts
ARIMA(1,2,2)

Coefficients:
         ar1     ma1     ma2
      -0.466  -0.054  -0.753
s.e.   0.182   0.124   0.097

sigma^2 = 1175:  log likelihood = -311
AIC=631   AICc=631   BIC=639

Training set error measures:
               ME RMSE  MAE  MPE MAPE  MASE  ACF1
Training set 1.97 32.9 23.5 2.32 9.03 0.178 0.136

# Diagnostic check
> checkresiduals(model_arima_monthly)

        Ljung-Box test

data:  Residuals from ARIMA(1,2,2)
Q* = 6, df = 10, p-value = 0.8

Model df: 3.   Total lags used: 13
```




```
# Forecast for the next year
> forecast_arima_monthly <- forecast(model_arima_monthly, h = 15)
> plot(forecast_arima_monthly)
```

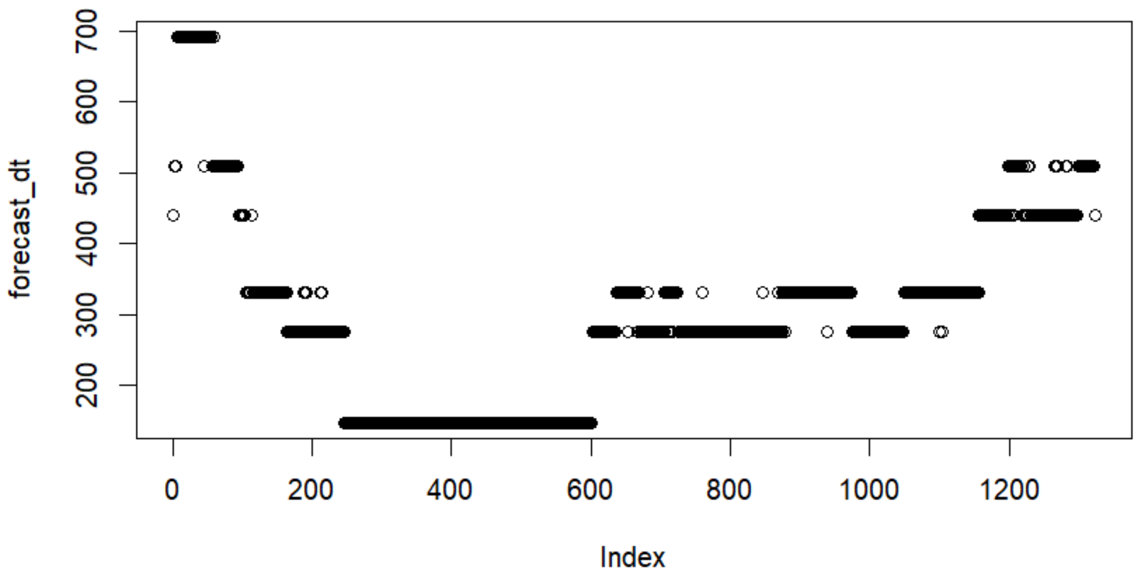


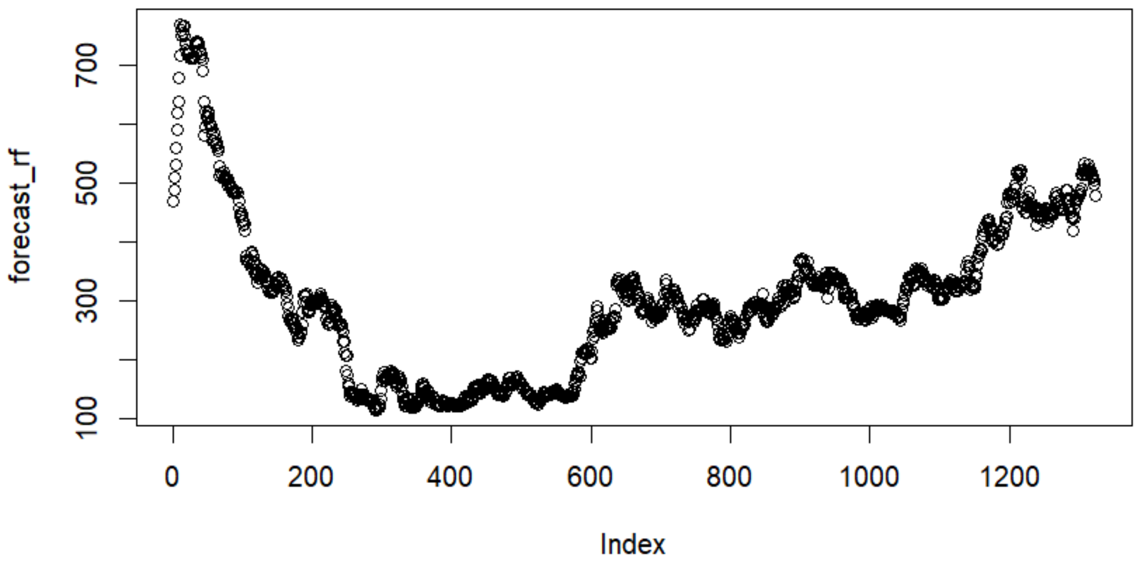
Interpretation:

The ARIMA model is a statistical tool used to predict stock prices, specifically designed for d aily data. It features auto-regressive terms, differentiation, and moving average terms. The mo del has good fit statistics with a Sigma^2 (Error variance) of 83.2, a log likelihood of -4795, a n AIC of 9605, and a Bayesian Information Criterion of 9647. The training set error measures include mean error (ME), root Mean Squared Error (RMSE), mean absolute error (MAE), me an percentage error (MPE), mean absolute percentage error (MAPE), mean absolute scaled er ror (MASE), and first-order autocorrelation (ACF1). The ARIMA model is also used for mon thly data, with characteristics like AR terms, differentiation, and moving average terms. The m odel provides a solid foundation for understanding and predicting future stock prices, aiding i n investment decision-making and strategic planning. The forecast for the next 15 months sho ws expected trend and seasonality, providing valuable insights for long-term investment plann ing. The daily forecast from ARIMA shows that stock prices will remain stable over the next f ew days. But the monthly forecast from ARIMA shows that the stock prices of Arvind Fashio ns are expected to grown in the next 15 months. The diagnostic plots suggest that the ARIMA (5,1,2) model is a good fit for the daily data, with no obvious patterns or trends in the residual s, no significant autocorrelation in the residuals, and approximately normally distributed resid uals, indicating that the model's errors are random. This makes the model well-specified and a ppropriate for forecasting daily closing prices.

- **Creating Random Forest and Decision Tree Models and forecasting for the next t hree months**

```
# Load required libraries
> library(randomForest)
> library(rpart)
# Create a Random Forest model
> model_rf <- randomForest(close ~., data = data)
# Create a Decision Tree model
> model_dt <- rpart(close ~., data = data)
# Forecast for the next three months
> forecast_rf <- predict(model_rf, newdata = data)
> forecast_dt <- predict(model_dt, newdata = data)
> plot(forecast_dt)
> plot(forecast_rf)
```





Interpretation:

The Random Forest model is an ensemble learning method that combines multiple decision tr ees to improve prediction accuracy and control over-fitting. It is trained using the randomFore st function from the `randomForest` package, with `close` as the response variable and all oth er variables in `data` as predictors.

The Decision Tree model is a simple and interpretable machine learning model that splits data into subsets based on input features to make predictions. It is easy to interpret but can be pron e to overfitting, especially when the data has noise. They might not generalize well to unseen data compared to ensemble methods.

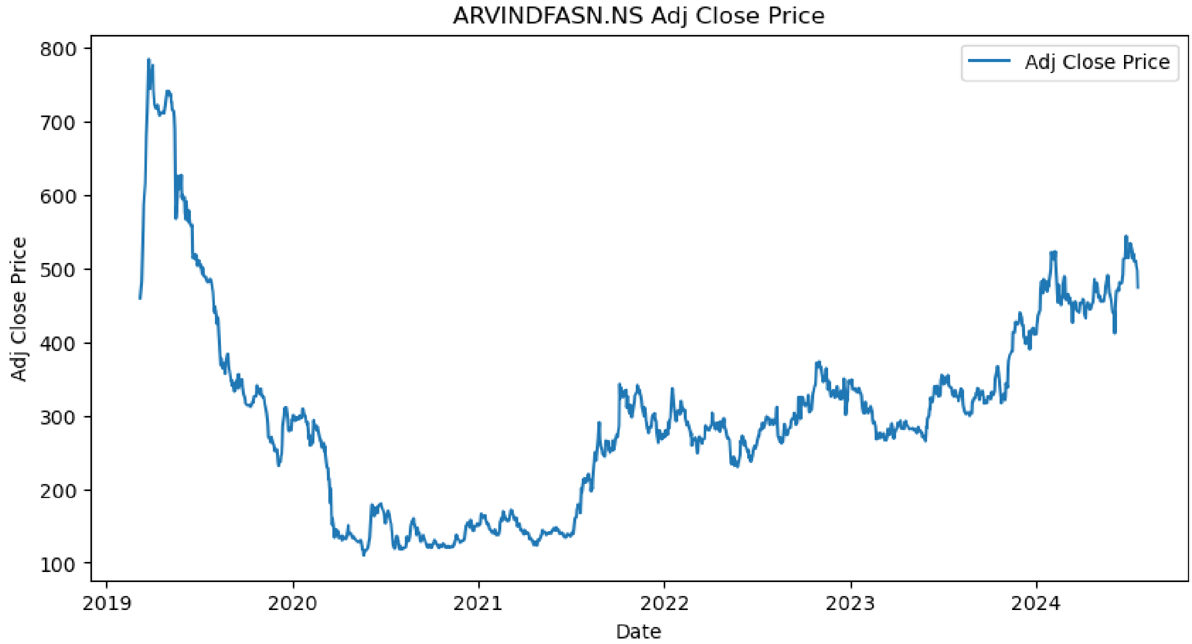
The forecast plots provide a visual comparison of the predicted closing prices over the next th ree months. The Random Forest model's predictions are expected to be smoother and more rel iable compared to the Decision Tree model. In terms of time series forecasting, ensemble met hods like Random Forests are preferred due to their ability to handle data variability and redu ce overfitting, leading to more robust predictions.

In conclusion, while both models have their merits, the Random Forest model generally provi des more accurate and stable predictions due to the averaging of multiple trees' outputs. Decis ion Trees, although more interpretable, might not perform as well on unseen data due to their t endency to overfit. Overall, ensemble methods like Random Forests are preferred for time ser ies forecasting due to their ability to handle data variability and reduce overfitting, leading to more robust predictions.

Results and Interpretation using Python

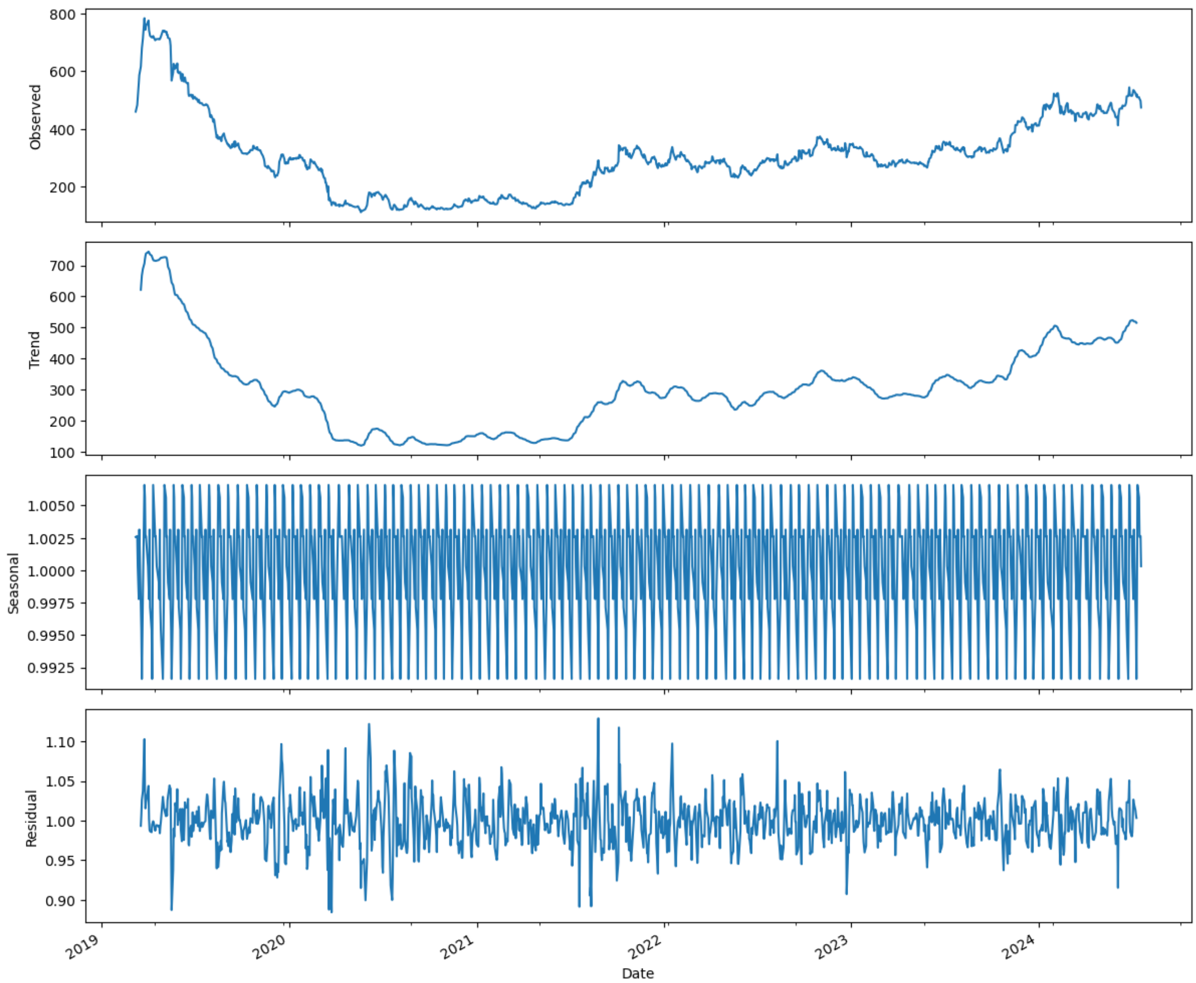
- **Plotting the data for Arvind Fashions and decomposed components.**

```
# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('ARVINDFASN.NS Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()
```




```
from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative',
period=12)

# Plot the decomposed components
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10), sharex=True)
result.observed.plot(ax=ax1)
ax1.set_ylabel('Observed')
result.trend.plot(ax=ax2)
ax2.set_ylabel('Trend')
result.seasonal.plot(ax=ax3)
ax3.set_ylabel('Seasonal')
result.resid.plot(ax=ax4)
ax4.set_ylabel('Residual')
plt.xlabel('Date')
```

```
plt.tight_layout()
plt.show()
```



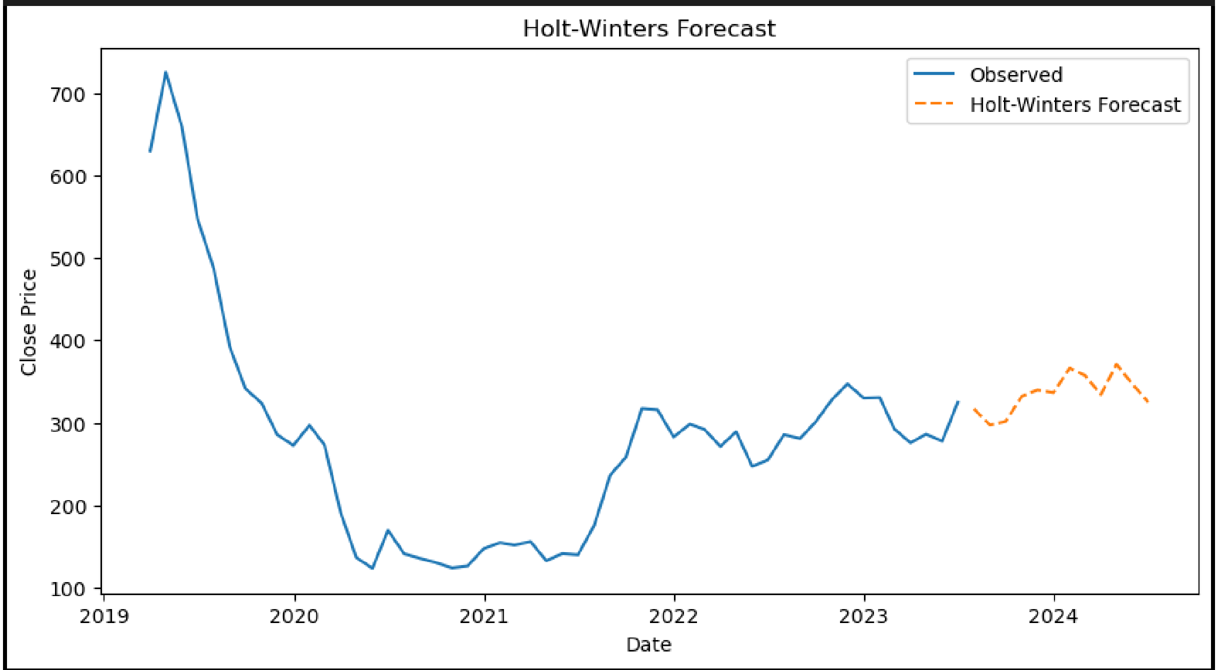
Interpretation:

The first graph shows the adjusted closing price of Arvind Fashions since its listing in the stock exchange. We can see that the price per share increased in the year 2019 and then it gradually started declining. But the share price has been recovering slowly since 2021. The above graphs display a time series decomposition plot with four components: Observed, Trend, Seasonal, and Residual. It explains the patterns in the data by separating the observed series into its trend, seasonal, and residual components.

- **Plotting the data for univariate forecasting using Holt Winters model, ARIMA monthly data and ARIMA daily data.**

```
## Holt Winters Model

from statsmodels.tsa.holtwinters import ExponentialSmoothing
```

```python
# Fit the Holt-Winters model
holt_winters_model = ExponentialSmoothing(train_data, seasonal='mul',
seasonal_periods=12).fit()

# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(12)

# Plot the forecast
plt.figure(figsize=(10, 5))
plt.plot(train_data, label='Observed')
plt.plot(holt_winters_forecast, label='Holt-Winters Forecast', linestyle='--')
plt.title('Holt-Winters Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```




```python
# Forecast for the next year (12 months)
y_pred = holt_winters_model.forecast(13)
len(test_data), len(y_pred)
y_pred, test_data
```

```
(2023-07-31    316.727549
 2023-08-31    297.260840
 2023-09-30    301.482310
 2023-10-31    331.979423
 2023-11-30    339.750346
 2023-12-31    336.444366
 2024-01-31    366.420949
 2024-02-29    357.620458
 2024-03-31    334.273319
```

```
2024-04-30    370.946194
2024-05-31    346.602973
2024-06-30    324.847002
2024-07-31    316.727549
Freq: M, dtype: float64,
             Adj Close
Date
2023-07-31  339.482419
2023-08-31  314.673803
2023-09-30  325.579999
2023-10-31  336.857498
2023-11-30  390.239999
2023-12-31  412.082500
2024-01-31  464.614285
2024-02-29  482.283334
2024-03-31  449.497221
2024-04-30  456.664996
2024-05-31  461.649997
2024-06-30  483.715787
2024-07-31  513.778571)
```

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, y_pred))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, y_pred)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - y_pred) / test_data)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, y_pred)
print(f'R-squared: {r2}')
```

```
RMSE: 101.07387249839353
MAE: 83.84901001757362
MAPE: nan
R-squared: -1.3315195685154206
```

```python
# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(len(test_data)+12)
holt_winters_forecast
```

```
2023-07-31    316.727549
2023-08-31    297.260840
2023-09-30    301.482310
2023-10-31    331.979423
2023-11-30    339.750346
2023-12-31    336.444366
2024-01-31    366.420949
```

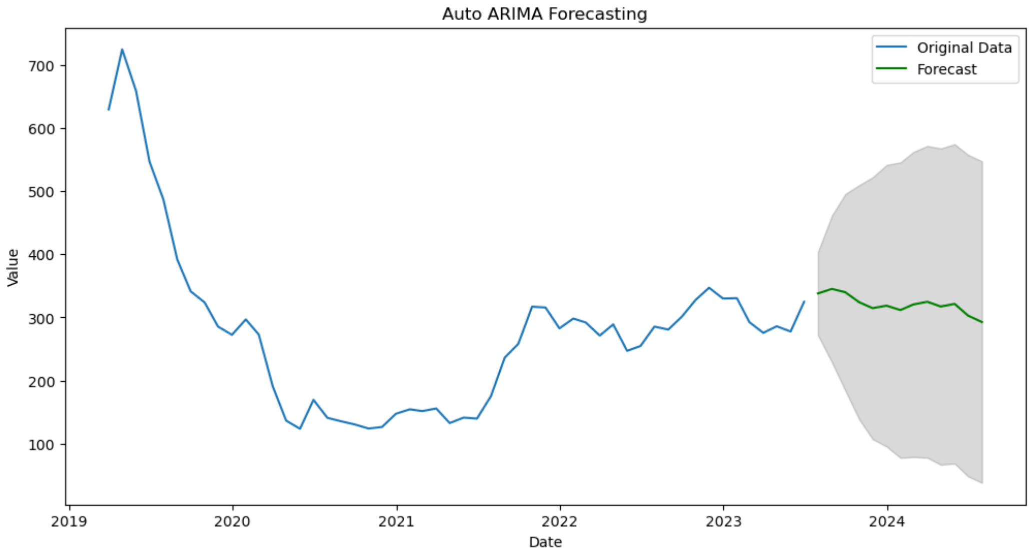
```
2024-02-29    357.620458
2024-03-31    334.273319
2024-04-30    370.946194
2024-05-31    346.602973
2024-06-30    324.847002
2024-07-31    316.727549
2024-08-31    297.260840
2024-09-30    301.482310
2024-10-31    331.979423
2024-11-30    339.750346
2024-12-31    336.444366
2025-01-31    366.420949
2025-02-28    357.620458
2025-03-31    334.273319
2025-04-30    370.946194
2025-05-31    346.602973
2025-06-30    324.847002
2025-07-31    316.727549
Freq: M, dtype: float64
```

```
## ARIMA Monthly Data

pip install pmdarima
from pmdarima import auto_arima
# Fit auto_arima model
arima_model = auto_arima(train_data['Adj Close'],
                         seasonal=True,
                         m=12,  # Monthly seasonality
                         stepwise=True,
                         suppress_warnings=True)

# Print the model summary
print(arima_model.summary())
```

```
SARIMAX Results
===============================================================================
==============
Dep. Variable:                                    y   No. Observations:
52
Model:             SARIMAX(2, 0, 0)x(0, 0, [1], 12)   Log Likelihood
-258.952
Date:                              Sun, 21 Jul 2024   AIC
527.904
Time:                                      19:07:19   BIC
537.660
Sample:                                  03-31-2019   HQIC
531.644
                                       - 06-30-2023
Covariance Type:                                opg
===============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------
intercept     20.0835     12.785      1.571      0.116      -4.974      45.141
ar.L1          1.4425      0.108     13.335      0.000       1.231       1.655
```

```
ar.L2          -0.5112      0.129     -3.952      0.000      -0.765      -0.258
ma.S.L12       -0.3229      0.234     -1.382      0.167      -0.781       0.135
sigma2       1140.6351    259.231      4.400      0.000     632.551    1648.719
==============================================================================
=====
Ljung-Box (L1) (Q):                   0.61   Jarque-Bera (JB):
2.14
Prob(Q):                              0.44   Prob(JB):
0.34
Heteroskedasticity (H):               0.41   Skew:
0.45
Prob(H) (two-sided):                  0.07   Kurtosis:
3.44
==============================================================================
=====

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex
-step).
```

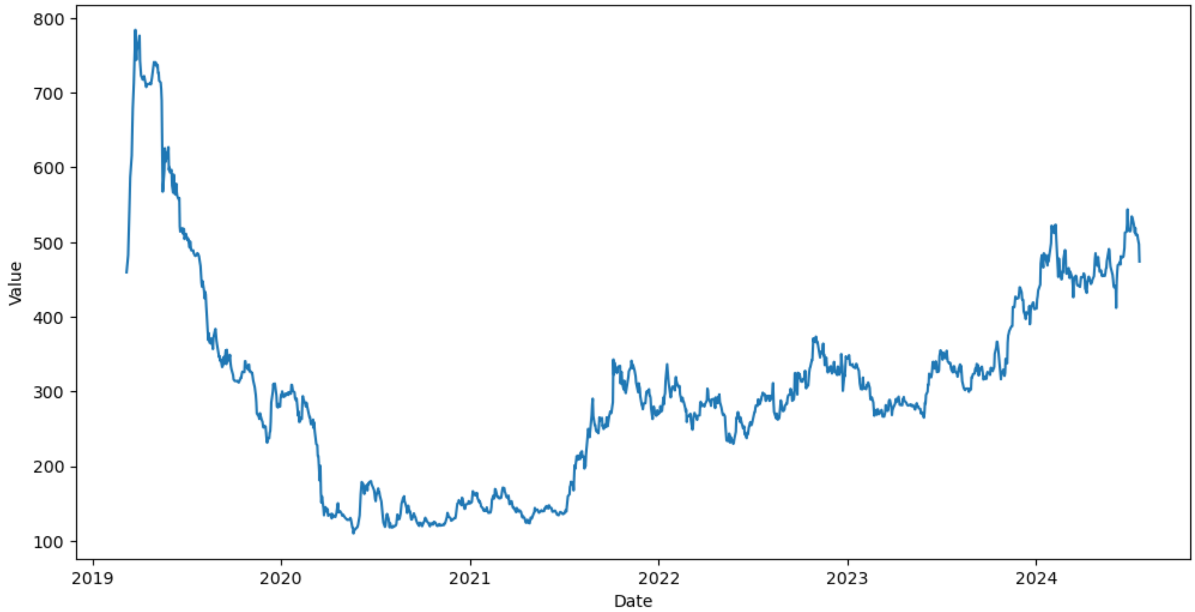
```python
# Number of periods to forecast
n_periods = 13

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods,
return_conf_int=True)

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(train_data['Adj Close'], label='Original Data')
plt.plot(forecast.index, forecast, label='Forecast', color='green')
plt.fill_between(forecast.index,
                 conf_int[:, 0],
                 conf_int[:, 1],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```




```
len(forecast)
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, forecast))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(test_data, forecast)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((test_data - forecast) / forecast)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, forecast)
print(f'R-squared: {r2}')
```

```
RMSE: 124.458957362301
MAE: 103.7380982542552
MAPE: nan
R-squared: -2.5351966005358784
```

```
## ARIMA Daily Data

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'])
plt.xlabel('Date')
plt.ylabel('Value')
plt.show()
```




```python
# Fit auto_arima model
arima_model = auto_arima(daily_data['Adj Close'],
                         seasonal=True,
                         m=7,  # Weekly seasonality
                         stepwise=True,
                         suppress_warnings=True)
# Print the model summary
print(arima_model.summary())
```

```
SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 1323
Model:               SARIMAX(1, 1, 1)   Log Likelihood               -4797.455
Date:                Sun, 21 Jul 2024   AIC                           9600.910
Time:                        19:10:46   BIC                           9616.470
Sample:                             0   HQIC                          9606.743
                               - 1323
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.8244      0.044     18.807      0.000       0.739       0.910
ma.L1         -0.7390      0.053    -13.912      0.000      -0.843      -0.635
sigma2        83.0933      1.518     54.748      0.000      80.119      86.068
==============================================================================
=====
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
3030.46
Prob(Q):                              0.87   Prob(JB):
0.00
Heteroskedasticity (H):               0.71   Skew:
-0.02
Prob(H) (two-sided):                  0.00   Kurtosis:
10.42
==============================================================================
=====
```

```
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex
-step).
```

```python
# Generate in-sample predictions
fitted_values = arima_model.predict_in_sample()
fitted_values
```

```
Date
2019-03-08      0.000000
2019-03-11    459.552623
2019-03-12    484.851249
2019-03-13    510.601337
2019-03-14    537.159190
                 ...
2024-07-12    518.864093
2024-07-15    508.533904
2024-07-16    509.152847
2024-07-18    503.351973
2024-07-19    495.838697
Name: predicted_mean, Length: 1323, dtype: float64
```

```python
# Number of periods to forecast
n_periods = 60  # For example, forecast the next 30 days

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods,
return_conf_int=True)
len(forecast)
# Create future dates index
last_date = daily_data.index[-1]
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1),
periods=n_periods)
# Convert forecast to a DataFrame with future_dates as the index
forecast_df = pd.DataFrame(forecast.values, index=future_dates,
columns=['forecast'])
conf_int_df = pd.DataFrame(conf_int, index=future_dates,
columns=['lower_bound', 'upper_bound'])
len(future_dates)
# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'], label='Original Data')
plt.plot(forecast_df, label='Forecast', color='green')
plt.fill_between(future_dates,
                 conf_int_df['lower_bound'],
                 conf_int_df['upper_bound'],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```

Interpretation:

The three graphs provided depict forecasting models and their results. The first image shows the Holt-Winters forecast, which predicts a slight decline followed by fluctuation around a level price. The second image shows the original data and the forecast using an Auto ARIMA model, which suggests a stable future trend with a narrow confidence interval. The third image provides detailed results of two different SARIMAX models. The first model has 52 observations with SARIMAX(2, 0, 0)x(0, 0, [1], 12), with key coefficients including intercept, ar.L1, ar.L2, and ma.S.L12. Diagnostic tests like the Ljung-Box Q-test and Jarque-Bera test indicate some skewness and kurtosis. The RMSE, MAE, and R-squared values suggest the model's performance, with a negative fit (-2.54). The second model has 1323 observations with SARIMAX(1, 1, 1), with significant coefficients like ar.L1 and ma.L1. Diagnostics show issues with skewness and kurtosis, but it has a smaller RMSE value (83.09) and better fit measures. The SARIMAX models provide detailed statistical diagnostics, offering insights into the data's underlying structure and the model's performance.

- **Plotting the data for multivariate forecasting using Machine learning models like long-short term memory, decision tree and random forest models.**

```
# Split the data into training and testing sets (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

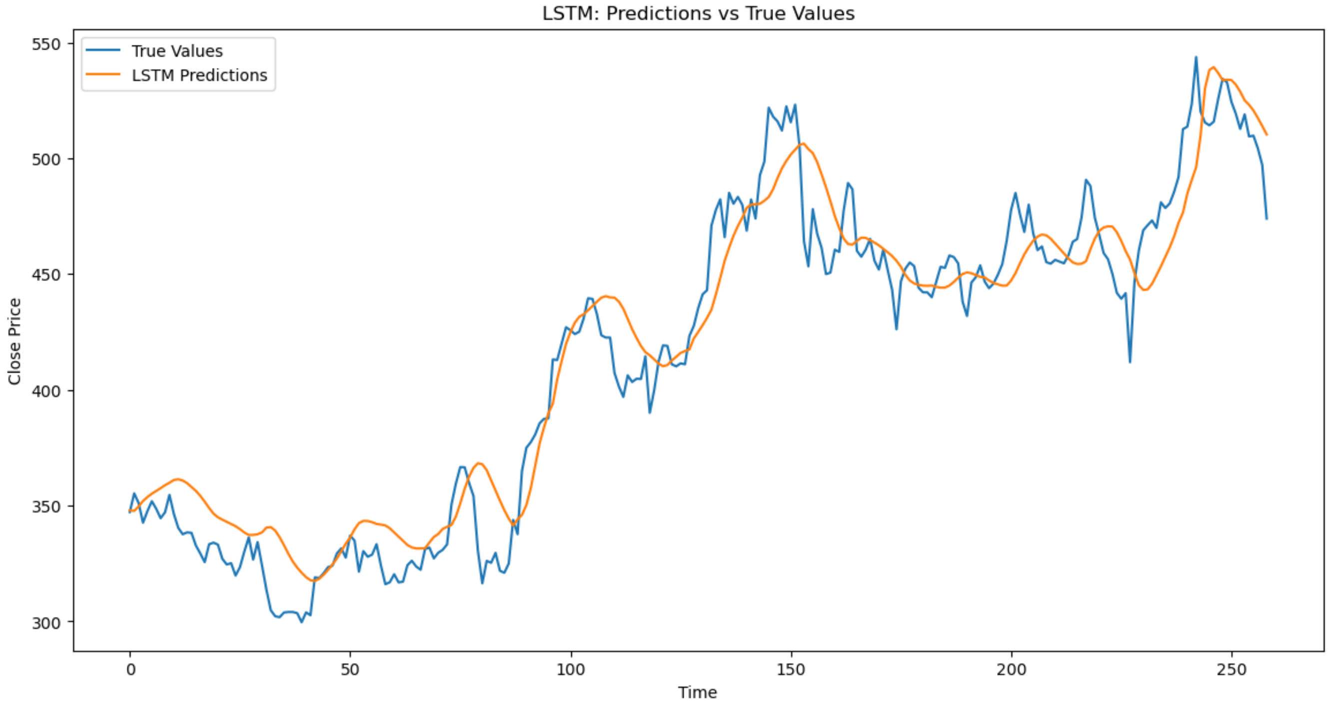
```python
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
input_shape=(sequence_length, 6)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))
model.summary()

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_test, y_test), shuffle=False)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")

# Predict on the test set
y_pred = model.predict(X_test)

# Inverse transform the predictions and true values to get them back to
the original scale
y_test_scaled =
scaler.inverse_transform(np.concatenate((np.zeros((len(y_test), 5)),
y_test.reshape(-1, 1)), axis=1))[:, 5]
y_pred_scaled =
scaler.inverse_transform(np.concatenate((np.zeros((len(y_pred), 5)),
y_pred), axis=1))[:, 5]

# Print some predictions and true values
print("Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_scaled[i]}, True Value:
{y_test_scaled[i]}")
```

```
Predictions vs True Values:
Prediction: 347.959734499239, True Value: 347.2178039550781
Prediction: 347.7294859127465, True Value: 355.29266357421875
Prediction: 349.6144114976583, True Value: 351.105712890625
Prediction: 352.00305230685314, True Value: 342.63214111328125
Prediction: 353.7085587141178, True Value: 347.6166076660157
Prediction: 355.1412612139325, True Value: 351.9032287597656
Prediction: 356.33307127633094, True Value: 348.6134948730469
Prediction: 357.5441607668177, True Value: 344.5760498046875
Prediction: 358.82421437819545, True Value: 347.1181640625
Prediction: 359.8797831487696, True Value: 354.6446838378907
```

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test_scaled, y_pred_scaled))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test_scaled, y_pred_scaled)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test_scaled - y_pred_scaled) / y_pred_scaled)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test_scaled, y_pred_scaled)
print(f'R-squared: {r2}')
```

```
RMSE: 18.5466594740428
MAE: 14.832559389098522
MAPE: 3.592634644318947
R-squared: 0.9270166052847776
```

```python
# Plot the predictions vs true values
plt.figure(figsize=(14, 7))
plt.plot(y_test_scaled, label='True Values')
plt.plot(y_pred_scaled, label='LSTM Predictions')
plt.title('LSTM: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



```
## Tree Based Models

from sklearn.ensemble import RandomForestRegressor #ensemble model
from sklearn.tree import DecisionTreeRegressor #simple algo
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

import numpy as np

def create_sequences(data, target_col, sequence_length):
    """
    Create sequences of features and labels for time series data.

    Parameters:
    - data (np.ndarray): The input data where the last column is the target.
    - target_col (int): The index of the target column in the data.
    - sequence_length (int): The length of each sequence.

    Returns:
    - np.ndarray: 3D array of sequences (samples, sequence_length,
num_features)
    - np.ndarray: 1D array of target values
    """
    num_samples = len(data) - sequence_length
    num_features = data.shape[1]

    sequences = np.zeros((num_samples, sequence_length, num_features))
    labels = np.zeros(num_samples)

    for i in range(num_samples):
        sequences[i] = data[i:i + sequence_length]
        labels[i] = data[i + sequence_length, target_col]  # Target is
specified column

    return sequences, labels

# Example usage
sequence_length = 30

# Convert DataFrame to NumPy array
data_array = scaled_df.values

# Define the target column index
target_col = scaled_df.columns.get_loc('Adj Close')

# Create sequences
X, y = create_sequences(data_array, target_col, sequence_length)
```

```
# Flatten X for Decision Tree
num_samples, seq_length, num_features = X.shape
X_flattened = X.reshape(num_samples, seq_length * num_features)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_flattened, y,
test_size=0.2, random_state=42)

# Train Decision Tree model
dt_model = DecisionTreeRegressor()
dt_model.fit(X_train, y_train)

# Make predictions
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
mse_dt = mean_squared_error(y_test, y_pred_dt)
print(f'MSE (Decision Tree): {mse_dt}')
```

```
MSE (Decision Tree): 0.00020545717311075377
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_dt))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_dt)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_dt)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_dt)
print(f'R-squared: {r2}')
```

```
RMSE: 0.014333777349699338
MAE: 0.010835470833212808
MAPE: inf
R-squared: 0.9936032531529979
```

```
# Train and evaluate the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Mean Squared Error: {mse_rf}")
```

```
Random Forest Mean Squared Error: 0.00014366675688987145
```

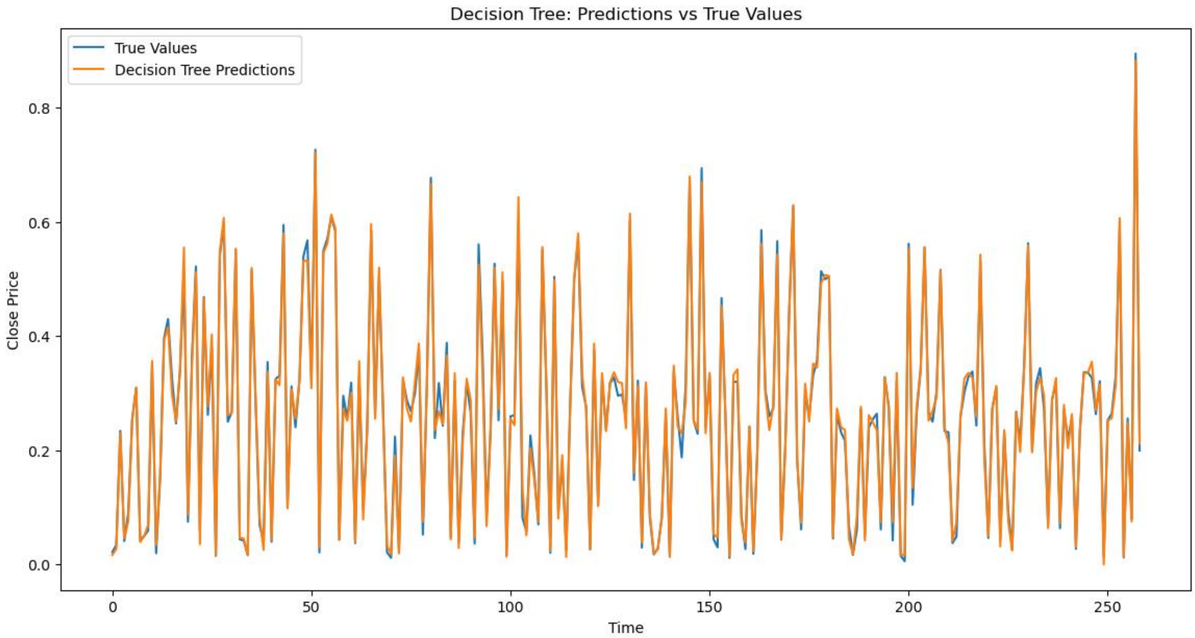
```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print(f'RMSE: {rmse}')

# Compute MAE
mae = mean_absolute_error(y_test, y_pred_rf)
print(f'MAE: {mae}')

# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_rf)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_rf)
print(f'R-squared: {r2}')
```

```
RMSE: 0.011986106827901688
MAE: 0.008431562250773123
MAPE: 453998.104887803
R-squared: 0.9955270489696706
```

```python
# Print some predictions and true values for both models
print("\nDecision Tree Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_dt[i]}, True Value: {y_test[i]}")
```

```
Decision Tree Predictions vs True Values:
Prediction: 0.016466448488643287, True Value: 0.021377503268786957
Prediction: 0.026866334527463592, True Value: 0.03435451333076117
Prediction: 0.23059248106653246, True Value: 0.23406901258801405
Prediction: 0.04574562704428678, True Value: 0.04087725562603506
Prediction: 0.07554348203892058, True Value: 0.08948216975582873
Prediction: 0.25263502960787443, True Value: 0.2549280205673572
Prediction: 0.309961660377796, True Value: 0.3094670315411024
Prediction: 0.039162448731564825, True Value: 0.042565003306998195
Prediction: 0.04966595231097798, True Value: 0.04988786050542943
Prediction: 0.06918798606244447, True Value: 0.0592214550262172
```

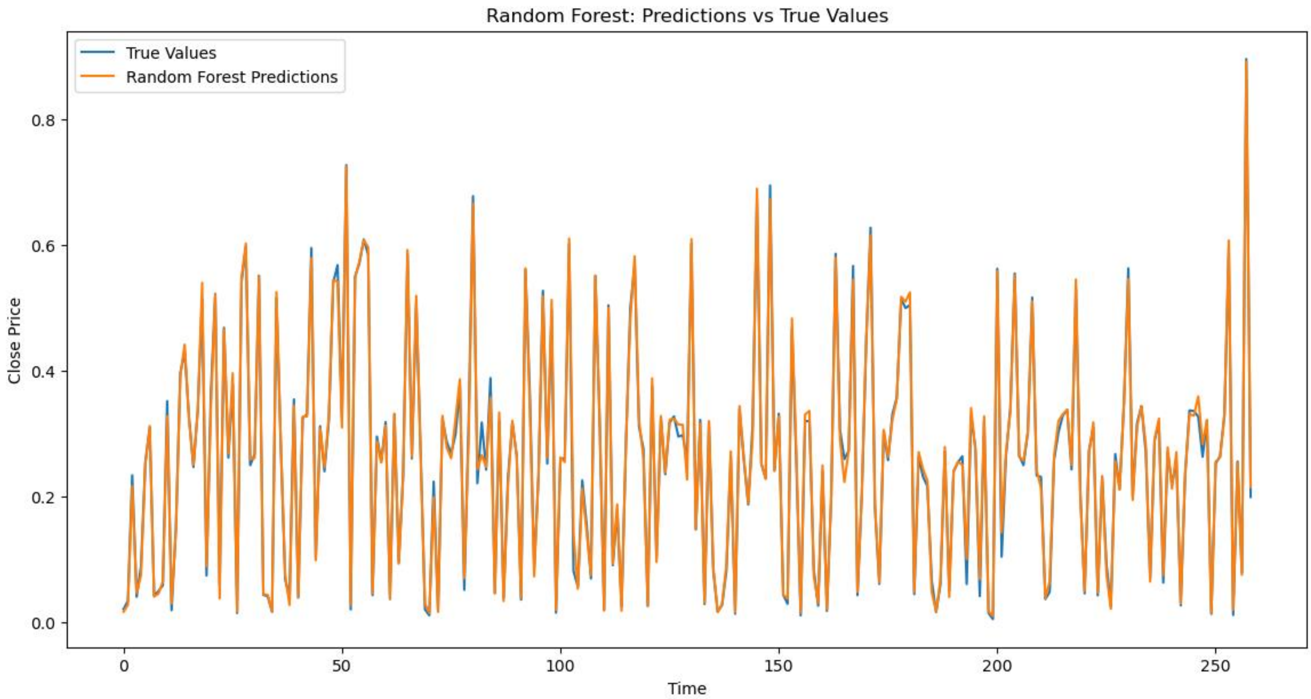
```python
# Plot the predictions vs true values for Decision Tree
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



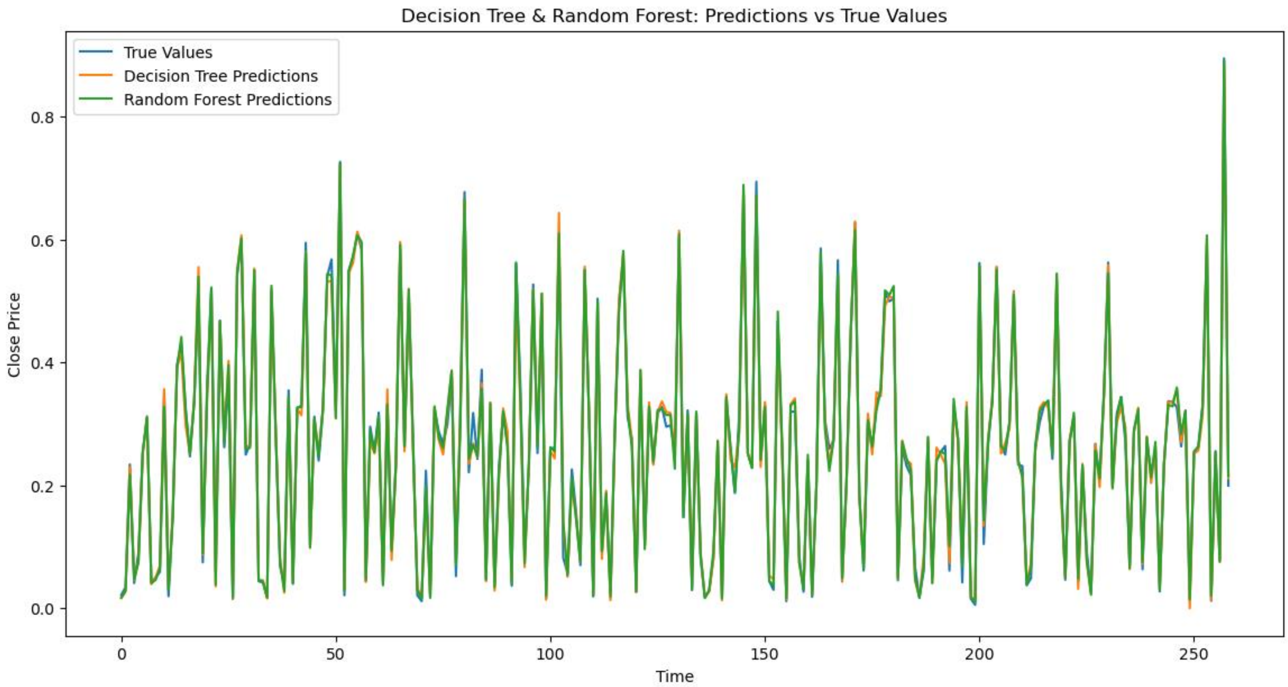

```
print("\nRandom Forest Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_rf[i]}, True Value: {y_test[i]}")
```

```
Random Forest Predictions vs True Values:
Prediction: 0.017069311462122063, True Value: 0.021377503268786957
Prediction: 0.028225563656437486, True Value: 0.03435451333076117
Prediction: 0.217213543183912, True Value: 0.23406901258801405
Prediction: 0.0468167464241702, True Value: 0.04087725562603506
Prediction: 0.07491887342100717, True Value: 0.08948216975582873
Prediction: 0.2512879254973579, True Value: 0.2549280205673572
Prediction: 0.312386427216252, True Value: 0.3094670315411024
Prediction: 0.041194974947810256, True Value: 0.042565003306998195
Prediction: 0.045976723370173195, True Value: 0.04988786050542943
Prediction: 0.0653516385553894, True Value: 0.0592214550262172
```

```
# Plot the predictions vs true values for Random Forest
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```




```
# Plot both Decision Tree and Random Forest predictions together
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Decision Tree & Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```



Interpretation:

The analysis of different machine learning models (LSTM, Decision Tree, and Random Forest) for multivariate time series forecasting provides insights into their performance. The LSTM model has a high R-squared value (0.9270), indicating that it explains a significant portion of the variance in the data. The model's predictions generally follow the general trend of the true values, with some deviations. The Decision Tree and Random Forest models have extremely high R-squared values (0.9936 for Decision Tree and 0.9955 for Random Forest), suggesting they fit the training data very well. Both models closely follow the true values, but the high MAPE values indicate potential issues with small value predictions. Overall, the Decision Tree and Random Forest models outperform the LSTM in terms of traditional error metrics. The unusual MAPE values for tree-based models suggest potential numerical issues that need addressing, possibly by handling small values differently.

Recommendation

The analysis and forecasting of Arvind Fashions Limited's stock price using various statistical and machine learning models has led to several recommendations. These include using ensemble methods for robust predictions, addressing high Mean Absolute Percentage Error (MAPE) in tree-based models, and implementing a hybrid model that combines the strengths of different models.

Data handling and preprocessing are crucial steps in improving model performance. Regular data cleaning and transformation, feature engineering, monthly data conversion and decomposition, and multiple evaluation metrics (RMSE, MAE, R-squared) are essential for ensuring accurate short-term predictions. Regular validation using a separate testing dataset is also recommended to ensure generalization to new data.

The robust forecasting models can be used for investor decision making, risk management, strategic financial planning, and operational efficiency. They provide actionable insights, helping investors make informed decisions about buying, holding, or selling stocks. These models can also be implemented as part of a broader risk management strategy to identify potential risks and adjust investment portfolios accordingly.

Future research and development should focus on exploring advanced models like Transformer models for better performance in time series forecasting, experiment with other ensemble methods and model tuning techniques to enhance prediction accuracy, and continuously improve models by keeping them updated with the latest data and regularly retraining them. Conducting periodic reviews and updates to incorporate new market trends and changes in the economic environment can further enhance the predictive accuracy of these models.

By following these recommendations, Arvind Fashions Limited can leverage advanced forecasting techniques to gain strategic advantages, improve financial planning, and enhance overall decision-making processes.

R Codes

```
#install packages
install.packages("tidyquant")
install.packages("lubridate")
install.packages("randomForest")
install.packages("tensorflow")

# Load required libraries
library(tidyquant)
library(dplyr)
library(lubridate)

# Download data from Yahoo Finance
data <- tq_get('ARVINDFASN.NS', from = "2019-01-01", to = "2024-07-21")

# Convert data to monthly
data$Date <- ymd(data$date)
data_monthly <- data %>%
  mutate(Month = format(Date, "%Y-%m")) %>%
  group_by(Month) %>%
  summarise(Close = mean(close))

#Holt Winters Model

# Load required libraries
library(forecast)

# Convert the Month column to a date format
data_monthly$Month <- as.Date(paste0(data_monthly$Month, "-01"))

# Create a ts object with a frequency of 12 (for monthly data)
data_monthly_ts <- ts(data_monthly$Close, start = start(data_monthly$Month), frequ
ency = 12)
```

```
# Fit a Holt Winters model to the data
model_hw <- ets(data_monthly_ts, model = "AAA")
summary(model_hw)

# Forecast for the next year
forecast_hw <- forecast(model_hw, h = 12)
plot(forecast_hw)

#ARIMA Model (Daily Data)

# Fit an ARIMA model to the daily data
model_arima_daily <- auto.arima(data$close)
summary(model_arima_daily)

# Diagnostic check
checkresiduals(model_arima_daily)

# Fit a Seasonal-ARIMA (SARIMA) model
model_sarima_daily <- auto.arima(data$close, seasonal = TRUE)
summary(model_sarima_daily)

# Forecast for the next three months
forecast_arima_daily <- forecast(model_sarima_daily, h = 90)
plot(forecast_arima_daily)

#ARIMA Model (Monthly Data)

# Fit an ARIMA model to the monthly data
model_arima_monthly <- auto.arima(data_monthly_ts)
summary(model_arima_monthly)

# Diagnostic check
checkresiduals(model_arima_monthly)
```

```
# Forecast for the next year
forecast_arima_monthly <- forecast(model_arima_monthly, h = 15)
plot(forecast_arima_monthly)

# Load required libraries
library(randomForest)
library(rpart)

# Create a Random Forest model
model_rf <- randomForest(close ~., data = data)

# Create a Decision Tree model
model_dt <- rpart(close ~., data = data)

# Forecast for the next three months
forecast_rf <- predict(model_rf, newdata = data)
forecast_dt <- predict(model_dt, newdata = data)
plot(forecast_dt)
plot(forecast_rf)
```

Python Codes

```
pip install yfinance

import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.model_selection import train_test_split

# Get the data for Arvind Fashions
ticker = "ARVINDFASN.NS"
```

```
# Download the data
data = yf.download(ticker, start="2019-01-01", end="2024-07-21")

data.head()

# Select the Target Varibale Adj Close
df = data[['Adj Close']]

# Check for missing values
print("Missing values:")
print(df.isnull().sum())

# Plot the data
plt.figure(figsize=(10, 5))
plt.plot(df, label='Adj Close Price')
plt.title('ARVINDFASN.NS Adj Close Price')
plt.xlabel('Date')
plt.ylabel('Adj Close Price')
plt.legend()
plt.show()

from statsmodels.tsa.seasonal import seasonal_decompose

df.columns

from statsmodels.tsa.seasonal import seasonal_decompose

# Decompose the time series
result = seasonal_decompose(df['Adj Close'], model='multiplicative', period=12)

# Plot the decomposed components
fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10), sharex=True)
result.observed.plot(ax=ax1)
ax1.set_ylabel('Observed')
```

```
result.trend.plot(ax=ax2)
ax2.set_ylabel('Trend')
result.seasonal.plot(ax=ax3)
ax3.set_ylabel('Seasonal')
result.resid.plot(ax=ax4)
ax4.set_ylabel('Residual')
plt.xlabel('Date')
plt.tight_layout()
plt.show()

# Split the data into training and test sets
train_data, test_data = train_test_split(df, test_size=0.2, shuffle=False)
```

1. Univariate Forecasting - Conventional Models/Statistical Models¶

1.1. Holt Winters Model¶

```
monthly_data = df.resample("M").mean()


# Split the data into training and test sets

train_data, test_data = train_test_split(monthly_data, test_size=0.2, shuffle=False)

len(monthly_data), len(train_data)


from statsmodels.tsa.holtwinters import ExponentialSmoothing


# Fit the Holt-Winters model

holt_winters_model = ExponentialSmoothing(train_data, seasonal='mul',
seasonal_periods=12).fit()
```

```
# Forecast for the next year (12 months)
holt_winters_forecast = holt_winters_model.forecast(12)

# Plot the forecast
plt.figure(figsize=(10, 5))
plt.plot(train_data, label='Observed')
plt.plot(holt_winters_forecast, label='Holt-Winters Forecast', linestyle='--')
plt.title('Holt-Winters Forecast')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Forecast for the next year (12 months)
y_pred = holt_winters_model.forecast(13)
len(test_data), len(y_pred)
y_pred, test_data

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, y_pred))
print(f'RMSE: {rmse}')
```

Compute MAE

mae = mean_absolute_error(test_data, y_pred)

print(f'MAE: {mae}')

Compute MAPE

mape = np.mean(np.abs((test_data - y_pred) / test_data)) * 100

print(f'MAPE: {mape}')

Compute R-squared

r2 = r2_score(test_data, y_pred)

print(f'R-squared: {r2}')

Forecast for the next year (12 months)

holt_winters_forecast = holt_winters_model.forecast(len(test_data)+12)

holt_winters_forecast

1.2 ARIMA Montly Data¶

monthly_data.columns

pip install pmdarima

from pmdarima import auto_arima

Fit auto_arima model

arima_model = auto_arima(train_data['Adj Close'],

```
                    seasonal=True,
                    m=12,  # Monthly seasonality
                    stepwise=True,
                    suppress_warnings=True)

# Print the model summary
print(arima_model.summary())

# Number of periods to forecast
n_periods = 13

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods, return_conf_int=True)

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(train_data['Adj Close'], label='Original Data')
plt.plot(forecast.index, forecast, label='Forecast', color='green')
plt.fill_between(forecast.index,
                 conf_int[:, 0],
                 conf_int[:, 1],
                 color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
```

```
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()


len(forecast)


from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Compute RMSE
rmse = np.sqrt(mean_squared_error(test_data, forecast))
print(f'RMSE: {rmse}')


# Compute MAE
mae = mean_absolute_error(test_data, forecast)
print(f'MAE: {mae}')


# Compute MAPE
mape = np.mean(np.abs((test_data - forecast) / forecast)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(test_data, forecast)
print(f'R-squared: {r2}')
```

1.3 ARIMA Daily Data¶

```
daily_data= df.copy()


# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'])
plt.xlabel('Date')
plt.ylabel('Value')
plt.show()


# Fit auto_arima model
arima_model = auto_arima(daily_data['Adj Close'],
                         seasonal=True,
                         m=7,  # Weekly seasonality
                         stepwise=True,
                         suppress_warnings=True)


# Print the model summary
print(arima_model.summary())


# Generate in-sample predictions
fitted_values = arima_model.predict_in_sample()


fitted_values
```

```
# Number of periods to forecast
n_periods = 60  # For example, forecast the next 30 days

# Generate forecast
forecast, conf_int = arima_model.predict(n_periods=n_periods, return_conf_int=True)

len(forecast)

# Create future dates index
last_date = daily_data.index[-1]
future_dates = pd.date_range(start=last_date + pd.Timedelta(days=1), periods=n_periods)

# Convert forecast to a DataFrame with future_dates as the index
forecast_df = pd.DataFrame(forecast.values, index=future_dates, columns=['forecast'])
conf_int_df = pd.DataFrame(conf_int, index=future_dates, columns=['lower_bound',
'upper_bound'])

len(future_dates)

# Plot the original data, fitted values, and forecast
plt.figure(figsize=(12, 6))
plt.plot(daily_data['Adj Close'], label='Original Data')
plt.plot(forecast_df, label='Forecast', color='green')
plt.fill_between(future_dates,
                 conf_int_df['lower_bound'],
```

```
                conf_int_df['upper_bound'],
                color='k', alpha=.15)
plt.legend()
plt.xlabel('Date')
plt.ylabel('Value')
plt.title('Auto ARIMA Forecasting')
plt.show()
```

2. Multivariate Forecasting - Machine Learning Models

```
pip install tensorflow


from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
import numpy as np


data.head()


# Initialize MinMaxScaler
scaler = MinMaxScaler()


# Select features (excluding 'Adj Close') and target ('Adj Close')
```

```
features = data.drop(columns=['Adj Close'])
target = data[['Adj Close']]

# Fit the scaler on features and target
scaled_features = scaler.fit_transform(features)
scaled_target = scaler.fit_transform(target)

# Create DataFrame with scaled features and target
scaled_df = pd.DataFrame(scaled_features, columns=features.columns, index=df.index)
scaled_df['Adj Close'] = scaled_target

# Function to create sequences
def create_sequences(scaled_df, target_col, sequence_length):
    sequences = []
    labels = []
    for i in range(len(scaled_df) - sequence_length):
        sequences.append(scaled_df[i:i + sequence_length])
        labels.append(scaled_df[i + sequence_length, target_col])  # Target column index
    return np.array(sequences), np.array(labels)

# Convert DataFrame to NumPy array
data_array = scaled_df.values

# Define the target column index and sequence length
```

```
target_col = scaled_df.columns.get_loc('Adj Close')
sequence_length = 30


# Create sequences
X, y = create_sequences(data_array, target_col, sequence_length)


print("Shape of X:", X.shape)
print("Shape of y:", y.shape)
```

2.1. Neural Networks - Long Short-term Memory (LTSM)

```
# Split the data into training and testing sets (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]


# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(sequence_length, 6)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))


model.summary()
```

```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test,
y_test), shuffle=False)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")

# Predict on the test set
y_pred = model.predict(X_test)

# Inverse transform the predictions and true values to get them back to the original scale
y_test_scaled = scaler.inverse_transform(np.concatenate((np.zeros((len(y_test), 5)),
y_test.reshape(-1, 1)), axis=1))[:, 5]
y_pred_scaled = scaler.inverse_transform(np.concatenate((np.zeros((len(y_pred), 5)),
y_pred), axis=1))[:, 5]

# Print some predictions and true values
print("Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_scaled[i]}, True Value: {y_test_scaled[i]}")
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test_scaled, y_pred_scaled))
print(f'RMSE: {rmse}')


# Compute MAE
mae = mean_absolute_error(y_test_scaled, y_pred_scaled)
print(f'MAE: {mae}')


# Compute MAPE
mape = np.mean(np.abs((y_test_scaled - y_pred_scaled) / y_pred_scaled)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test_scaled, y_pred_scaled)
print(f'R-squared: {r2}')


# Plot the predictions vs true values
plt.figure(figsize=(14, 7))
plt.plot(y_test_scaled, label='True Values')
plt.plot(y_pred_scaled, label='LSTM Predictions')
plt.title('LSTM: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
```

```
plt.legend()
plt.show()
```

2.2. Tree Based Models¶

```
from sklearn.ensemble import RandomForestRegressor #ensemble model
from sklearn.tree import DecisionTreeRegressor #simple algo
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

import numpy as np

def create_sequences(data, target_col, sequence_length):
    """
    Create sequences of features and labels for time series data.

    Parameters:
    - data (np.ndarray): The input data where the last column is the target.
    - target_col (int): The index of the target column in the data.
    - sequence_length (int): The length of each sequence.

    Returns:
    - np.ndarray: 3D array of sequences (samples, sequence_length, num_features)
    - np.ndarray: 1D array of target values
```

```
    """
    num_samples = len(data) - sequence_length
    num_features = data.shape[1]

    sequences = np.zeros((num_samples, sequence_length, num_features))
    labels = np.zeros(num_samples)

    for i in range(num_samples):
        sequences[i] = data[i:i + sequence_length]
        labels[i] = data[i + sequence_length, target_col]  # Target is specified column

    return sequences, labels

# Example usage
sequence_length = 30

# Convert DataFrame to NumPy array
data_array = scaled_df.values

# Define the target column index
target_col = scaled_df.columns.get_loc('Adj Close')

# Create sequences
X, y = create_sequences(data_array, target_col, sequence_length)
```

```
# Flatten X for Decision Tree

num_samples, seq_length, num_features = X.shape

X_flattened = X.reshape(num_samples, seq_length * num_features)


# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X_flattened, y, test_size=0.2,
random_state=42)


# Train Decision Tree model

dt_model = DecisionTreeRegressor()

dt_model.fit(X_train, y_train)


# Make predictions

y_pred_dt = dt_model.predict(X_test)


# Evaluate the model

mse_dt = mean_squared_error(y_test, y_pred_dt)

print(f'MSE (Decision Tree): {mse_dt}')


from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Compute RMSE

rmse = np.sqrt(mean_squared_error(y_test, y_pred_dt))

print(f'RMSE: {rmse}')
```

```
# Compute MAE
mae = mean_absolute_error(y_test, y_pred_dt)
print(f'MAE: {mae}')


# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_dt)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_dt)
print(f'R-squared: {r2}')


# Train and evaluate the Random Forest model
rf_model = RandomForestRegressor(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
print(f"Random Forest Mean Squared Error: {mse_rf}")


from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score


# Compute RMSE
rmse = np.sqrt(mean_squared_error(y_test, y_pred_rf))
print(f'RMSE: {rmse}')
```

```
# Compute MAE
mae = mean_absolute_error(y_test, y_pred_rf)
print(f'MAE: {mae}')
# Compute MAPE
mape = np.mean(np.abs((y_test - y_pred_scaled) / y_pred_rf)) * 100
print(f'MAPE: {mape}')
# Compute R-squared
r2 = r2_score(y_test, y_pred_rf)
print(f'R-squared: {r2}')

# Print some predictions and true values for both models
print("\nDecision Tree Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_dt[i]}, True Value: {y_test[i]}")

# Plot the predictions vs true values for Decision Tree
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.title('Decision Tree: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
print("\nRandom Forest Predictions vs True Values:")
for i in range(10):
    print(f"Prediction: {y_pred_rf[i]}, True Value: {y_test[i]}")

# Plot the predictions vs true values for Random Forest
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()

# Plot both Decision Tree and Random Forest predictions together
plt.figure(figsize=(14, 7))
plt.plot(y_test, label='True Values')
plt.plot(y_pred_dt, label='Decision Tree Predictions')
plt.plot(y_pred_rf, label='Random Forest Predictions')
plt.title('Decision Tree & Random Forest: Predictions vs True Values')
plt.xlabel('Time')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

References

1. www.github.com
2. www.geeksforgeeks.com
3. www.datacamp.com
4. www.icssrdataservice.in
5. www.medium.com