Import the necessary libraries

In [1]:
```
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.stats.diagnostic import acorr_ljungbox
from sklearn.metrics import mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# AAPL Stock data from 2011 to 2025
Stock_data = yf.download("AAPL", start="2015-01-01", end="2025-01-01")
```

```
YF.download() has changed argument auto_adjust default to True
[*********************100%***********************]  1 of 1 completed
```

In [3]:
```
# Display first 5 rows
Stock_data.head()
```

Out[3]:

Price	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL
Date					
2015-01-02	24.288586	24.757340	23.848711	24.746232	212818400
2015-01-05	23.604334	24.137514	23.417722	24.057537	257142000
2015-01-06	23.606550	23.866475	23.244431	23.668754	263188400
2015-01-07	23.937571	24.037541	23.704304	23.815383	160423600
2015-01-08	24.857302	24.915063	24.148616	24.266361	237458000

Data Processessing

In [4]:
```
Stock_close = Stock_data[['Close']].copy()

 ## the index changed to datetime
Stock_close.index = pd.to_datetime(Stock_close.index)

# Sort by date
Stock_close.sort_index(inplace=True)
```

In [5]:
```
# Check for missing values
print("Missing values:", Stock_close.isna().sum())

#  Drop or fill missing values
Stock_close.dropna(inplace=True)


# final structure
print(Stock_close.head())
print(Stock_close.info())
```

```
Missing values: Price  Ticker
Close  AAPL      0
dtype: int64
Price           Close
Ticker           AAPL
Date
2015-01-02  24.288586
2015-01-05  23.604334
2015-01-06  23.606550
2015-01-07  23.937571
2015-01-08  24.857302
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2516 entries, 2015-01-02 to 2024-12-31
Data columns (total 1 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   (Close, AAPL)  2516 non-null   float64
dtypes: float64(1)
memory usage: 39.3 KB
None
```

Exploratory Data Analysis(EDA)

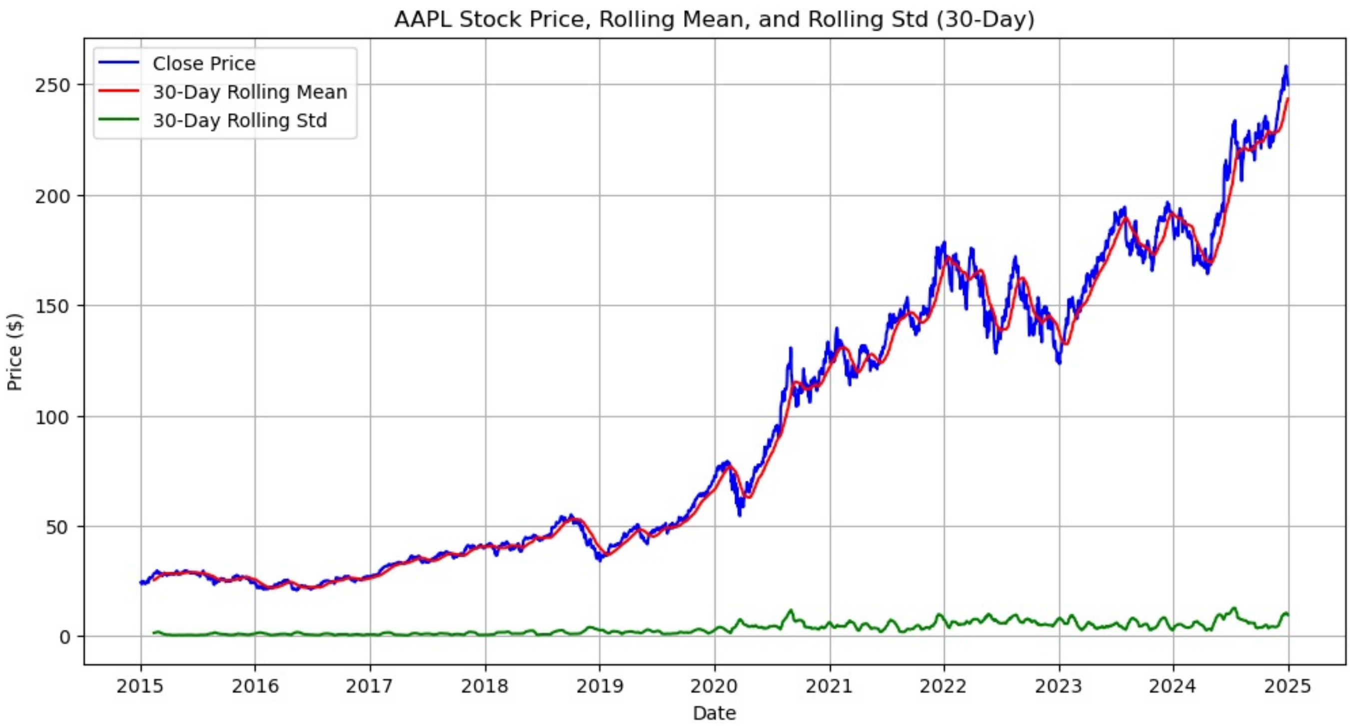
Line Plot of Closing Prices

In [6]:
```
plt.figure(figsize=(12, 6))
plt.plot(Stock_close, label='AAPL Close Price')
plt.title('AAPL Stock Price Over Time')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.grid(True)
plt.show()
```

Rolling Mean and Standard Deviation

In [7]:
```
# 30-day rolling mean and std deviation
rolling_mean = Stock_close['Close'].rolling(window=30).mean()
rolling_std = Stock_close['Close'].rolling(window=30).std()

plt.figure(figsize=(12, 6))
plt.plot(Stock_close['Close'], label='Close Price', color='blue')
plt.plot(rolling_mean, label='30-Day Rolling Mean', color='red')
plt.plot(rolling_std, label='30-Day Rolling Std', color='green')
plt.title('AAPL Stock Price, Rolling Mean, and Rolling Std (30-Day)')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.grid(True)
plt.show()
```



Augmented Dickey-Fuller (ADF) Test for Stationarity

In [8]:
```
# Perform ADF test on the Close prices
result = adfuller(Stock_close['Close'])
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')

# Interpretation:
if result[1] < 0.05:
    print("The series is likely stationary.")
else:
    print("The series is likely non-stationary.")
```

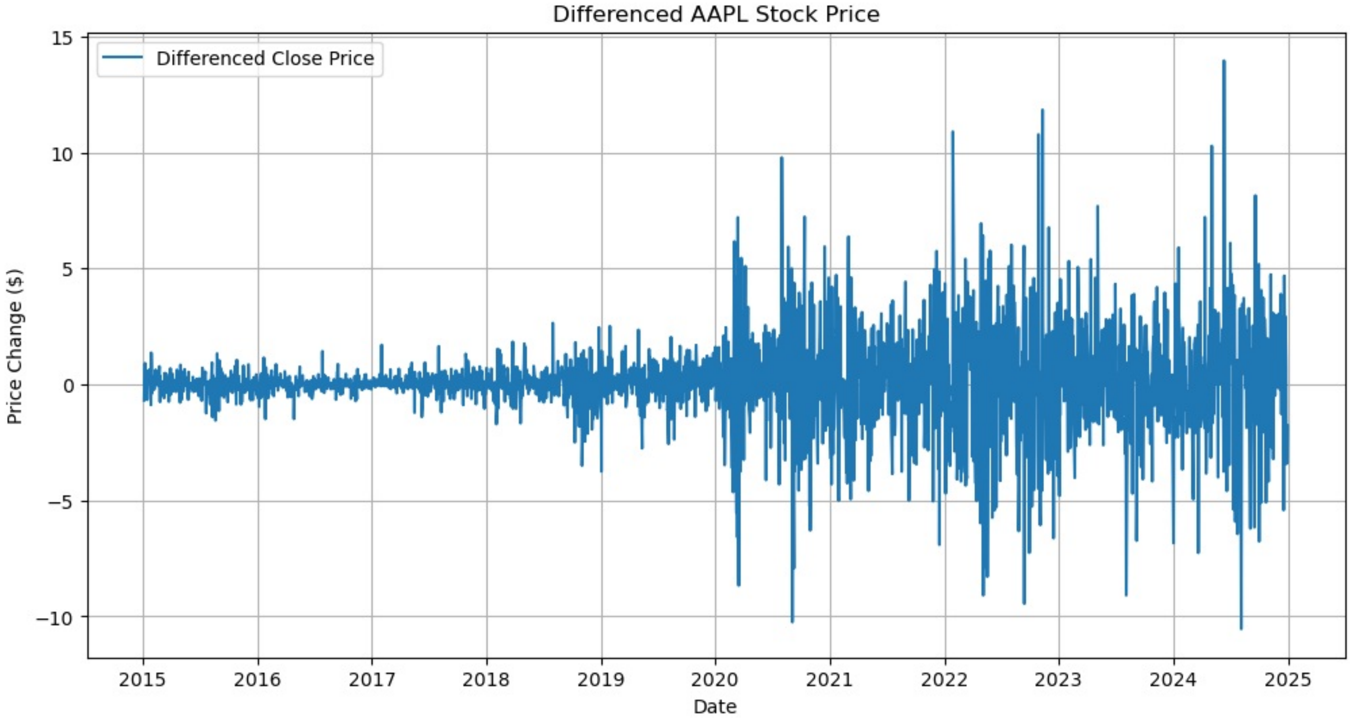
```
ADF Statistic: 0.7973433031053031
p-value: 0.991593162707525
The series is likely non-stationary.
```

Stationarizing the Series

In [9]:
```
#  Differencing
# First difference (remove trend)
Stock_close['Differenced'] = Stock_close['Close'] - Stock_close['Close'].shift(1)

# Drop NaN created by differencing
Stock_close.dropna(inplace=True)

# Plot the differenced series
plt.figure(figsize=(12, 6))
plt.plot(Stock_close['Differenced'], label='Differenced Close Price')
plt.title('Differenced AAPL Stock Price')
plt.xlabel('Date')
plt.ylabel('Price Change ($)')
plt.legend()
plt.grid(True)
plt.show()
```



ADF Test After Differencing

In [10]:
```
# Perform ADF test again on the differenced series
result_diff = adfuller(Stock_close['Differenced'])
print(f'ADF Statistic (Differenced): {result_diff[0]}')
print(f'p-value (Differenced): {result_diff[1]}')

# Interpretation
if result_diff[1] < 0.05:
    print("The differenced series is stationary.")
else:
    print("The differenced series is still non-stationary.")
```

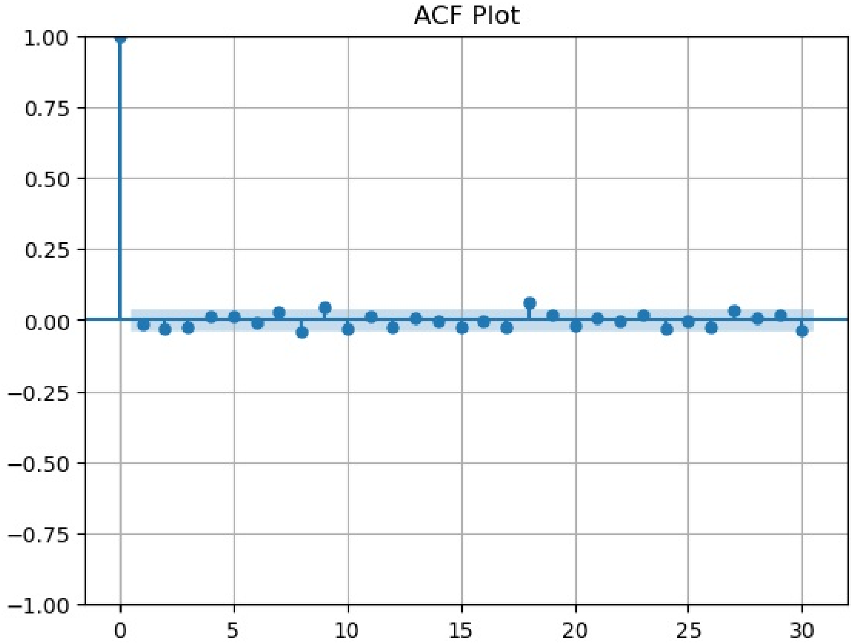
```
ADF Statistic (Differenced): -15.833236355123667
p-value (Differenced): 9.886975652882048e-29
The differenced series is stationary.
```

Time Series Modelling (ARIMA)

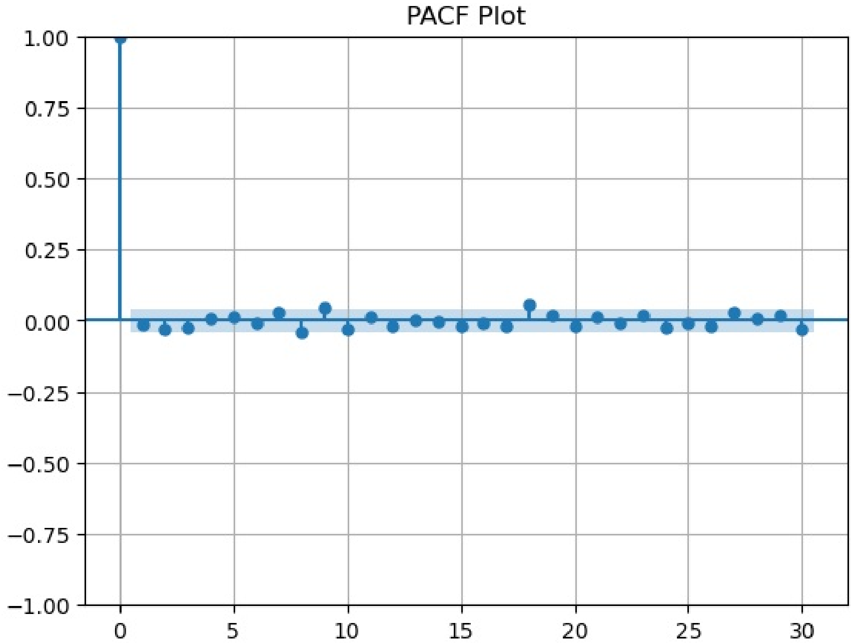
In [11]:
```
plt.figure(figsize=(12, 5))
plot_acf(Stock_close['Differenced'], lags=30)
plt.title("ACF Plot")
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 5))
plot_pacf(Stock_close['Differenced'], lags=30)
plt.title("PACF Plot")
plt.grid(True)
plt.show()
```

```
<Figure size 1200x500 with 0 Axes>
```



<Figure size 1200x500 with 0 Axes>



Fit an ARIMA Model

In [13]:
```
# Fit the model
model = ARIMA(Stock_close['Close'], order=(5, 1, 2))
model_fit = model.fit()

# Summary of the model
```

```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                   AAPL   No. Observations:                 2515
Model:                 ARIMA(5, 1, 2)   Log Likelihood               -5214.138
Date:                Thu, 15 May 2025   AIC                          10444.277
Time:                        10:16:54   BIC                          10490.914
Sample:                             0   HQIC                         10461.203
                               - 2515
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.9139      4.497      0.203      0.839      -7.901       9.728
ar.L2         -0.1956      3.272     -0.060      0.952      -6.609       6.217
ar.L3         -0.0031      0.077     -0.040      0.968      -0.154       0.148
ar.L4          0.0307      0.032      0.964      0.335      -0.032       0.093
ar.L5         -0.0040      0.147     -0.027      0.978      -0.291       0.283
ma.L1         -0.9258      4.498     -0.206      0.837      -9.741       7.889
ma.L2          0.1816      3.325      0.055      0.956      -6.336       6.699
sigma2         3.7069      0.054     69.265      0.000       3.602       3.812
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):              3926.31
Prob(Q):                              0.91   Prob(JB):                         0.00
Heteroskedasticity (H):              41.96   Skew:                             0.05
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.12
===================================================================================

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

In [15]:

```
# Fit ARIMA(1,1,1) model
model = ARIMA(Stock_close['Close'], order=(1, 1, 1))
model_fit = model.fit()

# Summary
print(model_fit.summary())
```

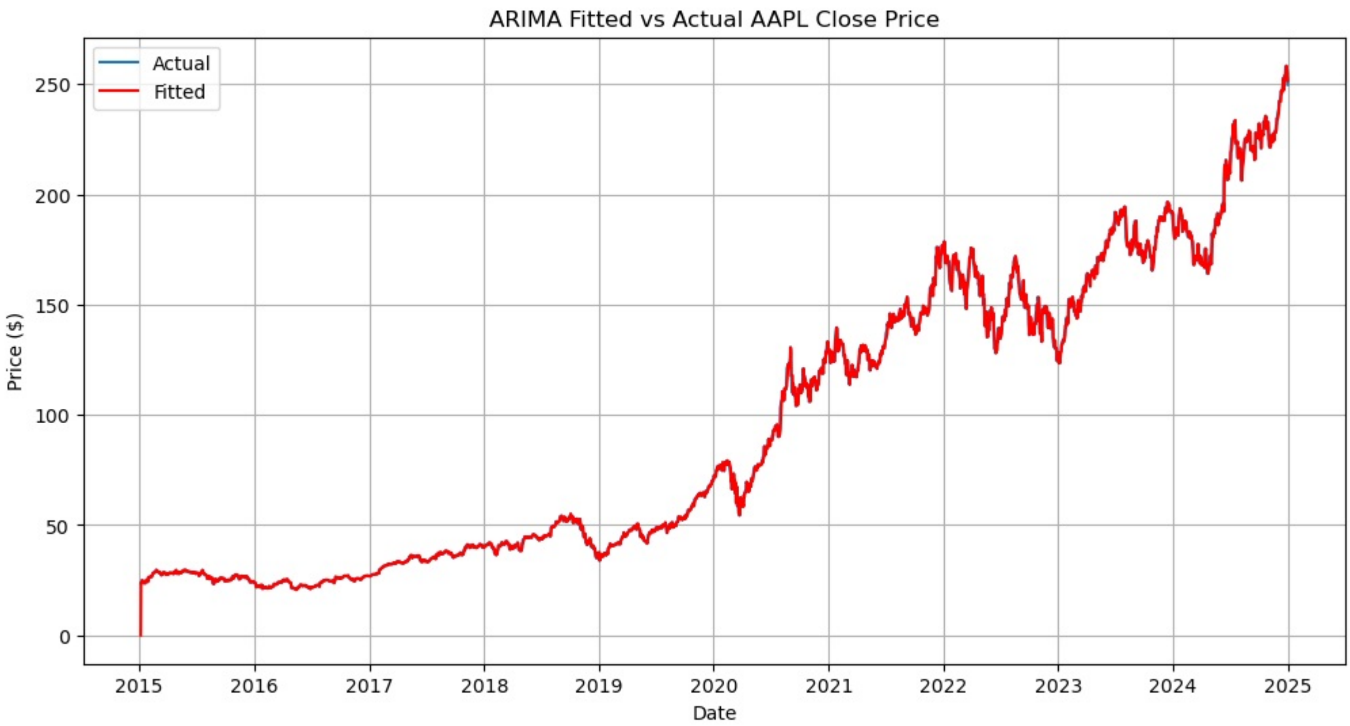
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                   AAPL   No. Observations:                 2515
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -5215.698
Date:                Thu, 15 May 2025   AIC                          10437.395
Time:                        10:23:27   BIC                          10454.884
Sample:                             0   HQIC                         10443.743
                               - 2515
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.5480      0.379      1.445      0.149      -0.195       1.292
ma.L1         -0.5678      0.375     -1.514      0.130      -1.303       0.167
sigma2         3.7115      0.053     70.204      0.000       3.608       3.815
===================================================================================
Ljung-Box (L1) (Q):                   0.08   Jarque-Bera (JB):              3902.29
Prob(Q):                              0.77   Prob(JB):                         0.00
Heteroskedasticity (H):              42.01   Skew:                             0.05
Prob(H) (two-sided):                  0.00   Kurtosis:                         9.10
===================================================================================

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

Plot Fitted vs Actual

In [16]:

```
# Plot predictions vs actual values
plt.figure(figsize=(12, 6))
plt.plot(Stock_close['Close'], label='Actual')
plt.plot(model_fit.fittedvalues, label='Fitted', color='red')
plt.title('ARIMA Fitted vs Actual AAPL Close Price')
plt.xlabel('Date')
plt.ylabel('Price ($)')
plt.legend()
plt.grid(True)
plt.show()
```



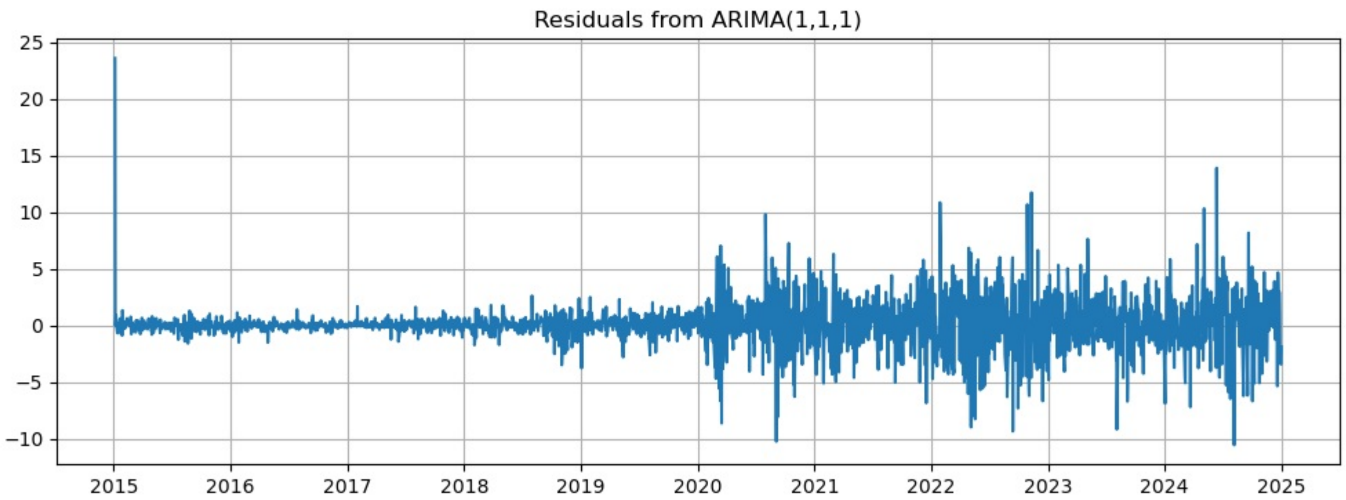
Residual Diagnostic

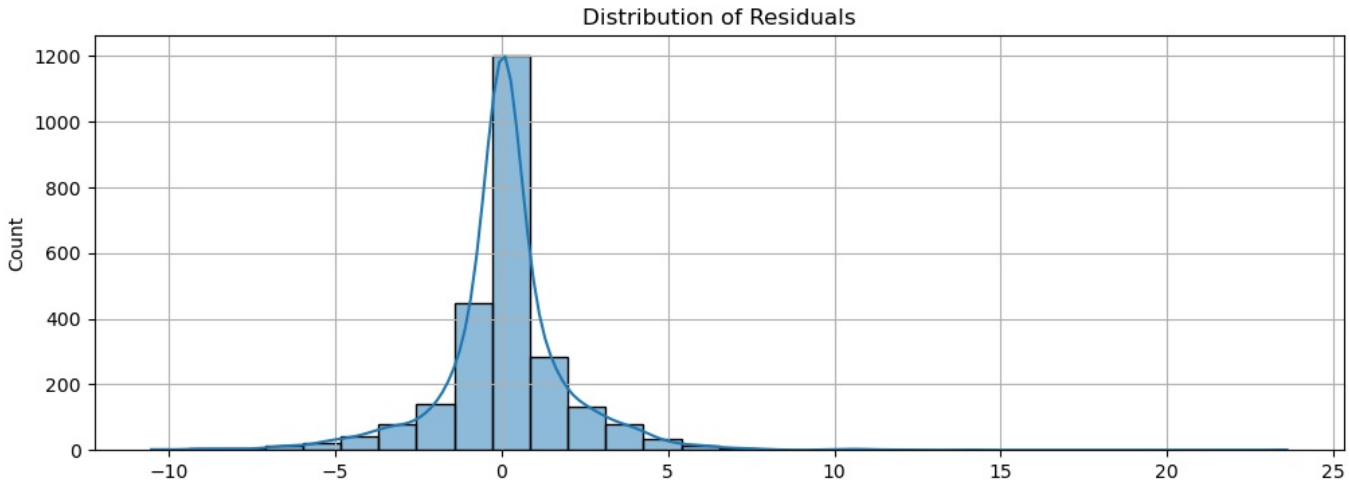
In [17]:
```
# Extract residuals
import seaborn as sns

residuals = model_fit.resid

# Plot residuals
plt.figure(figsize=(12, 4))
plt.plot(residuals)
plt.title('Residuals from ARIMA(1,1,1)')
plt.grid(True)
plt.show()

# Plot density of residuals
plt.figure(figsize=(12, 4))
sns.histplot(residuals, kde=True, bins=30)
plt.title('Distribution of Residuals')
plt.grid(True)
plt.show()
```





In [18]:
```
# Test if residuals are white noise
ljung_box_result = acorr_ljungbox(residuals, lags=[10], return_df=True)
print(ljung_box_result)
```

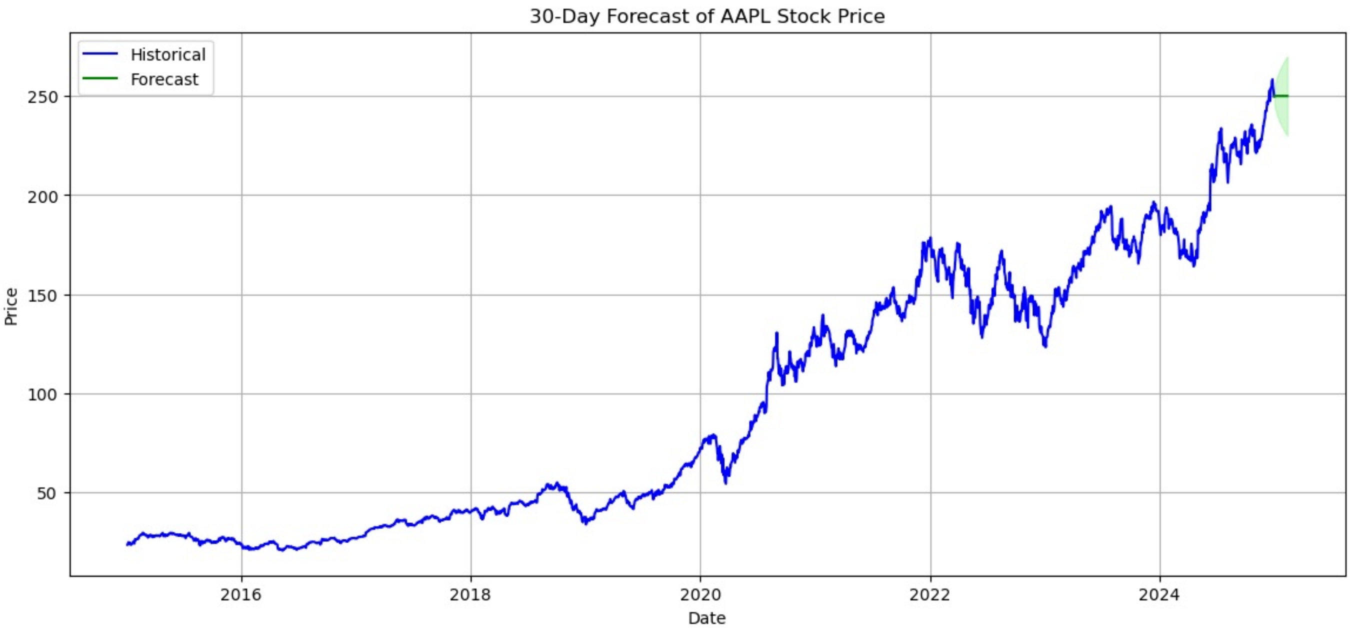
```
      lb_stat  lb_pvalue
10  14.675135   0.144362
```

Forecasting with ARIMA Model

In [19]:
```
# Forecast 30 steps into the future
forecast_steps = 30
forecast_result = model_fit.get_forecast(steps=forecast_steps)

# Get prediction and confidence intervals
forecast = forecast_result.predicted_mean
conf_int = forecast_result.conf_int()

# Plot forecast with confidence intervals
plt.figure(figsize=(14, 6))
plt.plot(Stock_close.index, Stock_close['Close'], label='Historical', color='blue')
forecast_index = pd.date_range(start=Stock_close.index[-1], periods=forecast_steps + 1, freq='B')[1:]
plt.plot(forecast_index, forecast, label='Forecast', color='green')
plt.fill_between(forecast_index, conf_int.iloc[:, 0], conf_int.iloc[:, 1], color='lightgreen', alpha=0.4)
plt.title('30-Day Forecast of AAPL Stock Price')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



Model Evaluation: Train-Test Split & Forecast Accuracy

Split the Data

In [20]:
```
# Using the last 30 business days as test data
train_data = Stock_close['Close'][:-30]
test_data = Stock_close['Close'][-30:]
```

Fit ARIMA on Training Data

In [21]:
```
model = ARIMA(train_data, order=(1,1,1))
model_fit = model.fit()
```

Forecast

In [22]:
```
forecast_result = model_fit.forecast(steps=30)
```

Evaluate Performance

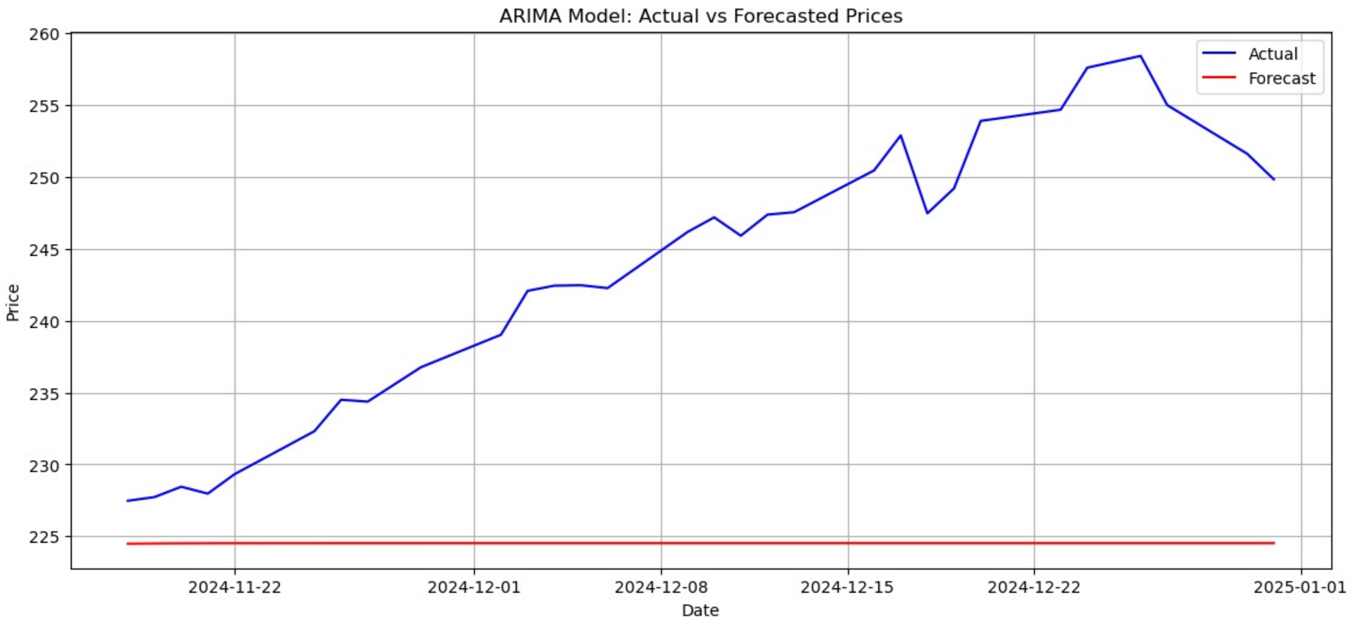
In [23]:
```
rmse = mean_squared_error(test_data, forecast_result, squared=False)
mae = mean_absolute_error(test_data, forecast_result)
mape = np.mean(np.abs((test_data - forecast_result) / test_data.replace(0, np.nan))) * 100

print(f'RMSE: {rmse:.2f}')
print(f'MAE: {mae:.2f}')
print(f'MAPE: {mape:.2f}%')
```

```
RMSE: 21.10
MAE: 18.88
MAPE: nan%
```

Visualize Actual vs Forecast

In [24]:
```
plt.figure(figsize=(14, 6))
plt.plot(test_data.index, test_data, label='Actual', color='blue')
plt.plot(test_data.index, forecast_result, label='Forecast', color='red')
plt.title('ARIMA Model: Actual vs Forecasted Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js