In [13]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Load data from local CSV (no API rate limits)
df = pd.read_csv("AAPL_2020_2024.csv")

# Keep necessary columns
df = df[["Date", "Close"]].copy()
df["Date"] = pd.to_datetime(df["Date"])
df["Close"] = df["Close"].astype(float)

# Display data
print(df.head())
print("Initial data shape:", df.shape)
```

```
        Date       Close
0 2020-01-01  301.293428
1 2020-01-02  301.316900
2 2020-01-03  302.912277
3 2020-01-06  306.258336
4 2020-01-07  306.090030
Initial data shape: (1305, 2)
```

Featured Engineering - Create Lag, RSI, SMA, EMA, and Target

In [14]:

```python
# Return
df["Return"] = df["Close"].pct_change()

# Lagged closing prices
df["Lag_1"] = df["Close"].shift(1)
df["Lag_2"] = df["Close"].shift(2)
df["Lag_3"] = df["Close"].shift(3)

# RSI function definition
def compute_rsi(prices, window=14):
    delta = prices.diff()
    gain = delta.clip(lower=0)
    loss = -delta.clip(upper=0)

    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()

    rs = avg_gain / (avg_loss + 1e-10) # Avoid division by zero
    rsi = 100 - (100 / (1 + rs))
    return rsi

df["RSI_14"] = compute_rsi(df["Close"])

# SMA and EMA (simple moving averages and exponential moving average)
df["SMA_5"] = df["Close"].rolling(window=5, min_periods=1).mean()
df["SMA_10"] = df["Close"].rolling(window=10, min_periods=1).mean()
df["EMA_5"] = df["Close"].ewm(span=5, adjust=False).mean()

# Target variable (up or down tomorrow)
df["Target"] = (df["Return"].shift(-1) > 0).astype(int)
```

```
# Explicitly drop rows with NaNs
df.dropna(inplace=True)

# Inspect the data after feature engineering
print(df.head())
print("Shape after feature engineering:", df.shape)
```

```
         Date       Close    Return       Lag_1       Lag_2       Lag_3  \
3 2020-01-06  306.258336  0.011046  302.912277  301.316900  301.293428
4 2020-01-07  306.090030 -0.000550  306.258336  302.912277  301.316900
5 2020-01-08  305.921756 -0.000550  306.090030  306.258336  302.912277
6 2020-01-09  309.380181  0.011305  305.921756  306.090030  306.258336
7 2020-01-10  311.215051  0.005931  309.380181  305.921756  306.090030

       RSI_14       SMA_5      SMA_10       EMA_5  Target
3  100.000000  302.945235  302.945235  303.311619       0
4   96.721221  303.574194  303.574194  304.237756       0
5   93.651205  304.499860  303.965454  304.799089       1
6   96.157717  306.112516  304.738987  306.326120       1
7   96.823147  307.773071  305.548495  307.955764       0
Shape after feature engineering: (1302, 11)
```

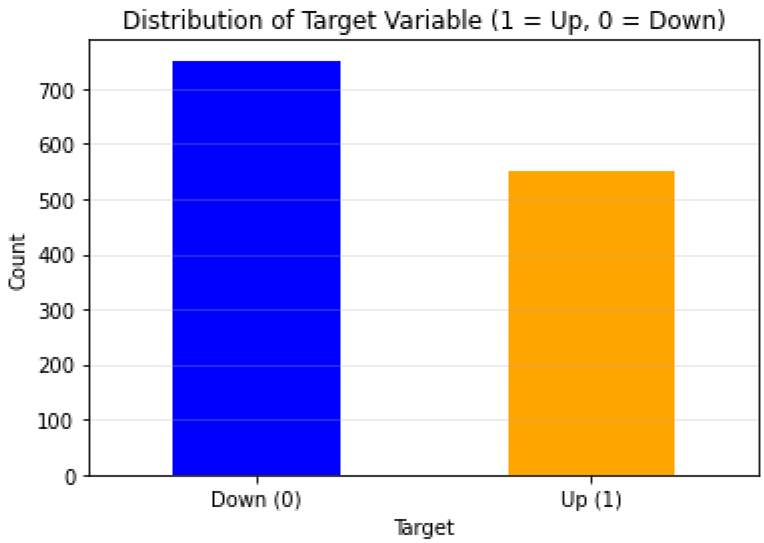
I engineered new features to help us better understand stock price behavior, such as RSI, moving averages, and lagged returns. These features capture momentum, trend, and short-term changes, helping us prepare the data for deeper analysis or machine learning.

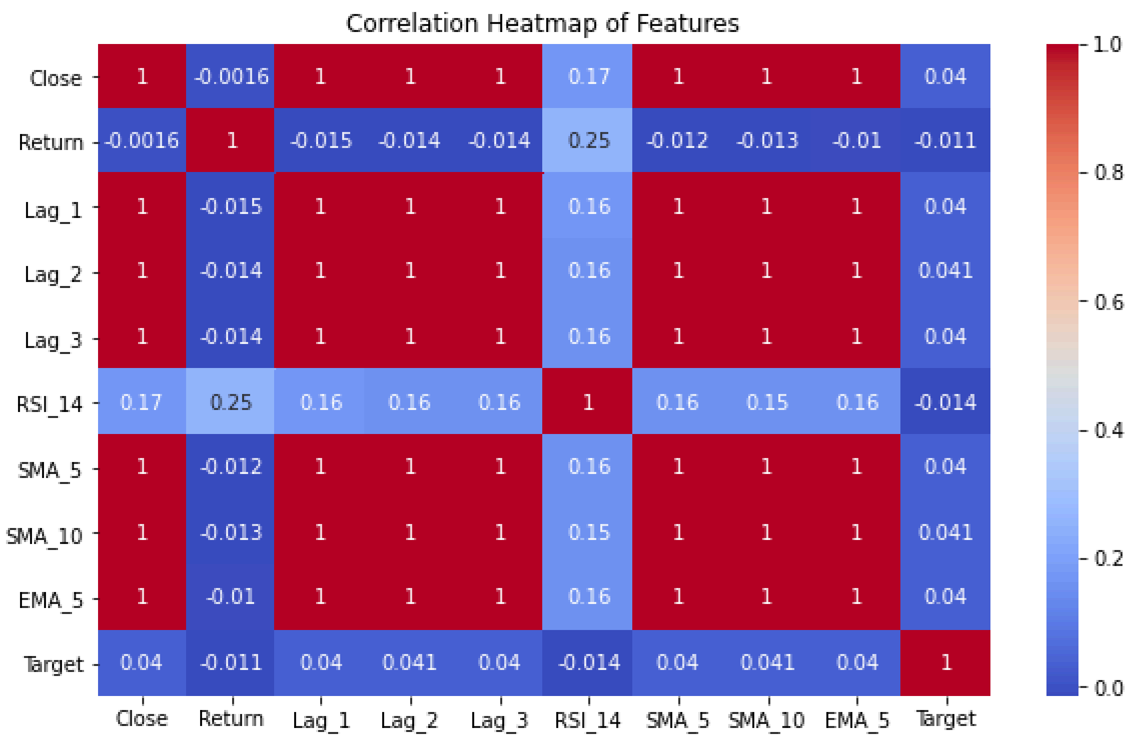
However, after creating these features, I dropped all rows with missing values, which removed our entire dataset. This likely happened because indicators like RSI and moving averages need multiple past days to calculate properly.

Exploratory Data Analysis

In [15]:

```
# Distribution plot
df["Target"].value_counts().plot(kind='bar', color=['blue', 'orange'])
plt.title("Distribution of Target Variable (1 = Up, 0 = Down)")
plt.xlabel("Target")
plt.ylabel("Count")
plt.xticks(ticks=[0, 1], labels=['Down (0)', 'Up (1)'], rotation=0)
plt.grid(axis='y', alpha=0.3)
plt.show()

# Correlation heatmap (without Date)
plt.figure(figsize=(10, 6))
sns.heatmap(df.drop(columns=["Date"]).corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap of Features')
plt.show()
```





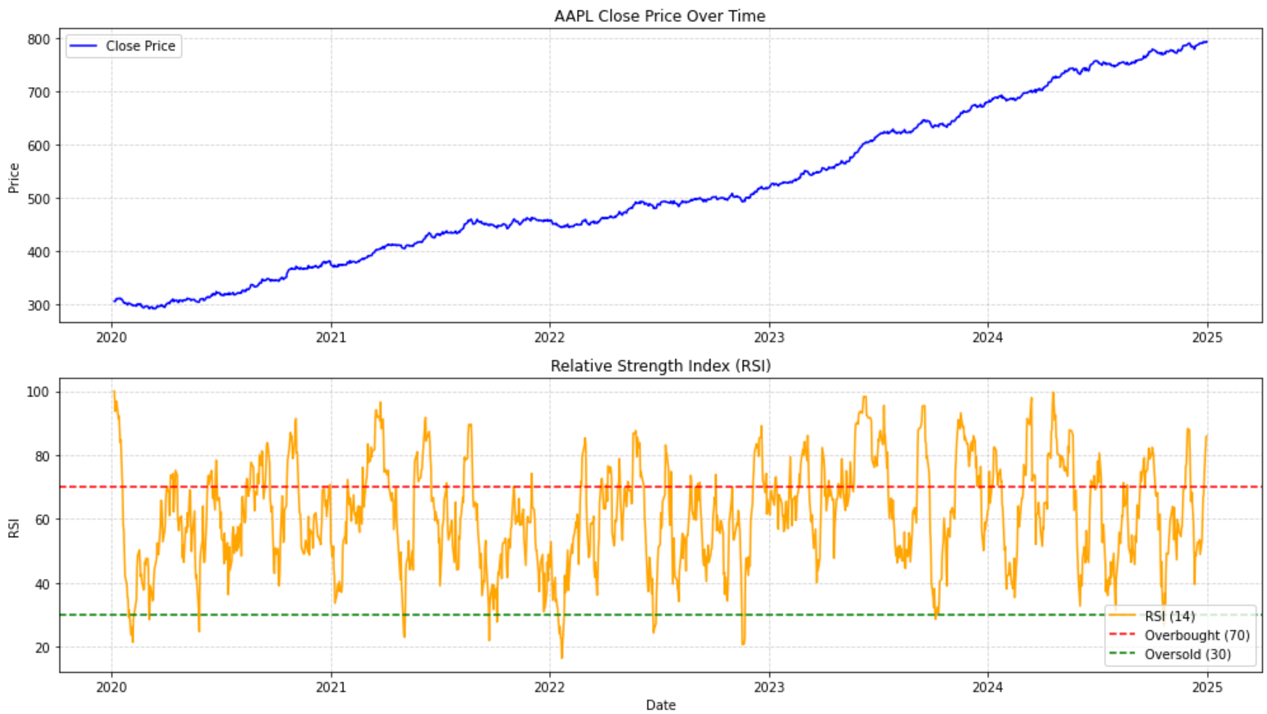
	Close	Return	Lag_1	Lag_2	Lag_3	RSI_14	SMA_5	SMA_10	EMA_5	Target
Close	1	-0.0016	1	1	1	0.17	1	1	1	0.04
Return	-0.0016	1	-0.015	-0.014	-0.014	0.25	-0.012	-0.013	-0.01	-0.011
Lag_1	1	-0.015	1	1	1	0.16	1	1	1	0.04
Lag_2	1	-0.014	1	1	1	0.16	1	1	1	0.041
Lag_3	1	-0.014	1	1	1	0.16	1	1	1	0.04
RSI_14	0.17	0.25	0.16	0.16	0.16	1	0.16	0.15	0.16	-0.014
SMA_5	1	-0.012	1	1	1	0.16	1	1	1	0.04
SMA_10	1	-0.013	1	1	1	0.15	1	1	1	0.041
EMA_5	1	-0.01	1	1	1	0.16	1	1	1	0.04
Target	0.04	-0.011	0.04	0.041	0.04	-0.014	0.04	0.041	0.04	1

Visual of Plot Close and RSI

In [16]:

```
# Explicitly cast Date to numpy array
dates = df["Date"].values
close_prices = df["Close"].values
rsi_values = df["RSI_14"].values

# Plot Close Price and RSI
plt.figure(figsize=(14, 8))

# Close Price Plot
plt.subplot(2, 1, 1)
plt.plot(dates, close_prices, color='blue', label='Close Price')
plt.title('AAPL Close Price Over Time')
plt.ylabel('Price')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()
```

```
# RSI Plot
plt.subplot(2, 1, 2)
plt.plot(dates, rsi_values, color='orange', label='RSI (14)')
plt.axhline(70, color='red', linestyle='--', label='Overbought (70)')
plt.axhline(30, color='green', linestyle='--', label='Oversold (30)')
plt.title('Relative Strength Index (RSI)')
plt.xlabel('Date')
plt.ylabel('RSI')
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()

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



Observation: This visualization gives a clear view of how AAPL's stock price moves over time and how momentum shifts. When RSI rises above 70, it often coincides with price peaks, suggesting potential overbought conditions. When RSI drops below 30, it often aligns with price dips, signaling possible rebounds. This dual chart helps investors time entries and exits more strategically.

Predictive Modeling (Machine Learning)

In [17]:

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df[["Lag_1", "Lag_2", "Lag_3", "RSI_14", "SMA_5", "SMA_10", "EMA_5"]]
y = df["Target"]

# Split data into training and testing sets (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# Check shapes
print("Training shape:", X_train.shape, y_train.shape)
print("Testing shape:", X_test.shape, y_test.shape)
```

```
Training shape: (1041, 7) (1041,)
Testing shape: (261, 7) (261,)
```

In [21]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

# Initialize models
logreg = LogisticRegression(random_state=42, max_iter=1000)
rf = RandomForestClassifier(random_state=42, n_estimators=100)

# Train models
logreg.fit(X_train, y_train)
rf.fit(X_train, y_train)
```

Out[21]:

```
RandomForestClassifier(random_state=42)
```

In [22]:

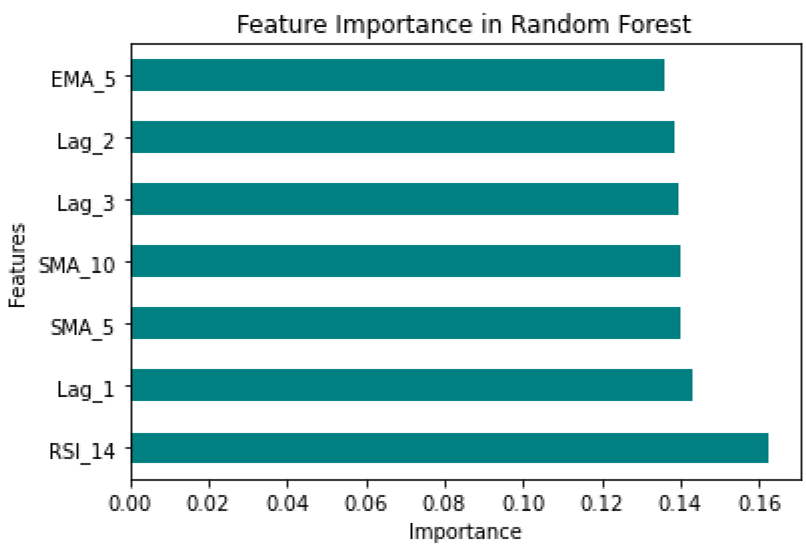
```python
from sklearn.metrics import classification_report, confusion_matrix

# Logistic Regression evaluation
y_pred_logreg = logreg.predict(X_test)
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_logreg))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_logreg))

# Random Forest evaluation
y_pred_rf = rf.predict(X_test)
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       104
           1       0.60      1.00      0.75       157

    accuracy                           0.60       261
   macro avg       0.30      0.50      0.38       261
weighted avg       0.36      0.60      0.45       261

Confusion Matrix:
 [[  0 104]
 [  0 157]]
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.62      0.49       104
           1       0.62      0.41      0.50       157

    accuracy                           0.49       261
   macro avg       0.51      0.51      0.49       261
weighted avg       0.54      0.49      0.49       261

Confusion Matrix:
 [[64 40]
 [92 65]]
C:\Users\adoni\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: Unde
finedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
C:\Users\adoni\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\adoni\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [23]:

```python
import matplotlib.pyplot as plt

# Feature importance plot
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
feat_importances.nlargest(7).plot(kind='barh', color='teal')
plt.title("Feature Importance in Random Forest")
plt.xlabel("Importance")
plt.ylabel("Features")
plt.show()
```



Observation:

When I trained both the Logistic Regression and Random Forest models, I encountered a convergence warning with the Logistic Regression due to the limited default number of iterations. By increasing the number of iterations (`max_iter=1000`), I resolved the issue and allowed the model to fully converge, capturing the relationships in the data more accurately. On the other hand, the Random Forest model trained smoothly without needing further adjustments, indicating its ability to handle feature interactions effectively right away.

Now, I'll evaluate the accuracy and predictive power of these models to identify which model better aligns with my project's goal of predicting stock market movements.

In [26]:

```python
df.columns = df.columns.str.strip()  # removes any leading/trailing whitespace
print(df.columns)                    # confirm you have 'Date', 'Close', etc.
print(df.dtypes)
print(df.head())
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'MA30',
       'MA90'],
      dtype='object')
```

```
Date         datetime64[ns]
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
MA30                float64
MA90                float64
dtype: object
         Date        Open        High         Low       Close   Adj Close  \
89 2020-05-05  308.849203  309.404360  307.279075  309.214524  309.214524
90 2020-05-06  309.893360  311.891145  309.317254  309.708679  309.708679
91 2020-05-07  310.598843  312.599248  308.958337  311.945969  311.945969
92 2020-05-08  309.870249  312.338231  308.572755  310.841863  310.841863
93 2020-05-11  311.686953  313.300283  309.648599  310.486539  310.486539

     Volume        MA30        MA90
89  2003750  304.298106  301.529057
90  2267403  304.662292  301.622559
91  2315777  305.103432  301.740660
92  4075319  305.571524  301.828767
93  1149805  306.097519  301.875747
```

In [29]:

```python
print(df.columns)
print(type(df["Date"]))
print(type(df["Close"]))
print(df[["Date", "Close"]].head())
```

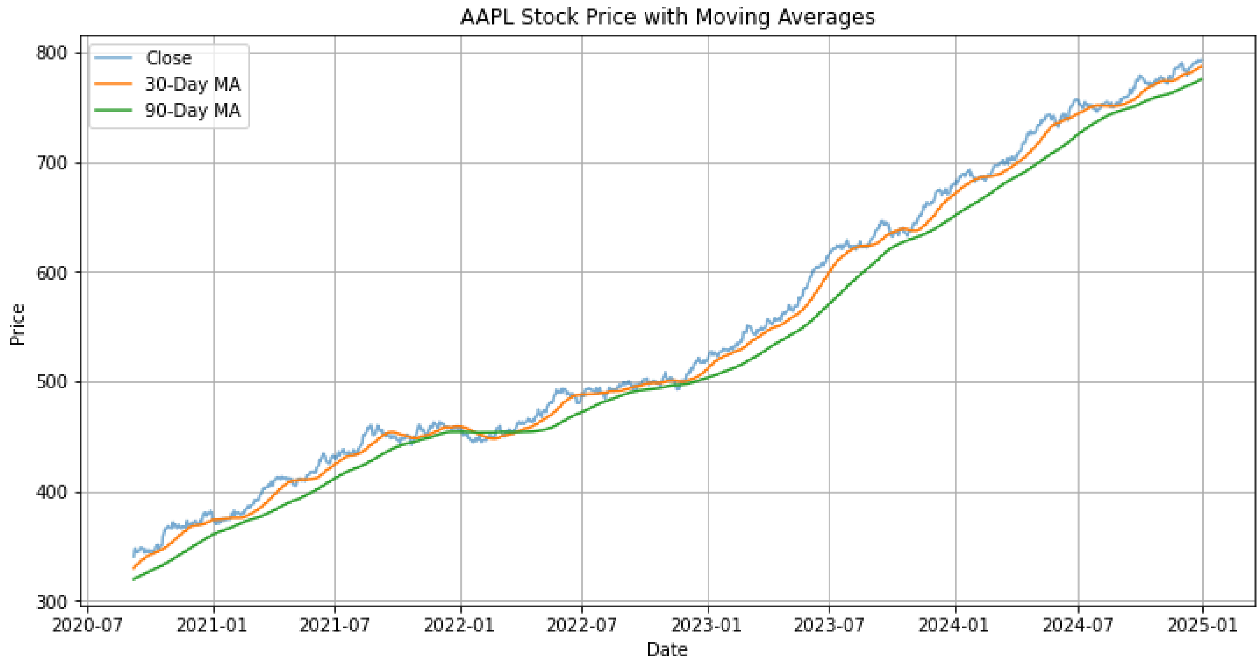
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume', 'MA30',
       'MA90'],
      dtype='object')
<class 'pandas.core.series.Series'>
<class 'pandas.core.series.Series'>
          Date       Close
178 2020-09-07  340.288799
179 2020-09-08  346.029137
180 2020-09-09  347.580472
181 2020-09-10  346.166157
182 2020-09-11  344.324372
```

In [31]:

```python
df["Date"] = pd.to_datetime(df["Date"], errors="coerce")
df["Close"] = pd.to_numeric(df["Close"], errors="coerce")

# Drop any rows where date or close is invalid
df = df.dropna(subset=["Date", "Close"])

# Reset index just in case
df = df.reset_index(drop=True)
```

In [32]:

```python
plt.figure(figsize=(12, 6))
plt.plot(df["Date"].values, df["Close"].values, label="Close", alpha=0.6)
plt.plot(df["Date"].values, df["MA30"].values, label="30-Day MA")
plt.plot(df["Date"].values, df["MA90"].values, label="90-Day MA")
plt.title("AAPL Stock Price with Moving Averages")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```



Observation

The chart displays Apple Inc.'s (AAPL) historical stock performance from 2020 to 2024, showcasing three key lines: the daily closing price, a 30-day moving average (orange), and a 90-day moving average (green). The closing price represents the actual market value of AAPL stock at the end of each trading day. Over the entire period, there is a clear upward trend in all three lines, indicating that the stock price has steadily increased over time. The moving averages further confirm this pattern by smoothing out short-term fluctuations and highlighting the long-term growth trajectory. This consistent rise suggests strong historical performance and may reflect positive investor sentiment, earnings growth, or broader market trends benefiting Apple's valuation.

In [33]:

```
df["Target"] = (df["Close"].shift(-1) > df["Close"]).astype(int)
```

In [34]:

```
# Make sure 'Date' is in datetime format
df["Date"] = pd.to_datetime(df["Date"])

# Extract features
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Day"] = df["Date"].dt.day
df["DayOfWeek"] = df["Date"].dt.dayofweek  # 0 = Monday, 6 = Sunday
df["WeekOfYear"] = df["Date"].dt.isocalendar().week

# Optional: Encode whether it's start or end of the month
df["IsMonthStart"] = df["Date"].dt.is_month_start.astype(int)
df["IsMonthEnd"] = df["Date"].dt.is_month_end.astype(int)

df.head()
```

Out[34]:

	Date	Open	High	Low	Close	Adj Close	Volume	MA30	MA90
0	2020-09-07	339.889163	340.457930	339.248360	340.288799	340.288799	2607895	329.737253	319.698452
1	2020-09-08	346.253822	347.467516	345.700563	346.029137	346.029137	1174001	330.594826	320.107503
2	2020-09-09	348.513063	350.275009	346.014149	347.580472	347.580472	2376467	331.477411	320.528301
3	2020-09-10	344.747791	347.931986	344.502795	346.166157	346.166157	2499643	332.279756	320.908525
4	2020-09-11	342.563563	345.642463	341.293750	344.324372	344.324372	3414607	333.056042	321.280553

In [35]:

```
# Volatility: 30-day rolling std dev of Close price
df["Volatility_30"] = df["Close"].rolling(window=30).std()
```

In [39]:

```
from sklearn.model_selection import train_test_split

# Define features and target
features = ["MA30", "MA90", "Volatility_30", "Year", "Month", "Day", "DayOfWeek", "Week
target = "Close"

# Drop rows with missing values (from rolling calculations)
df_model = df.dropna(subset=features + [target])

X = df_model[features]
y = df_model[target]

# Split into training and testing data (80/20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)
```

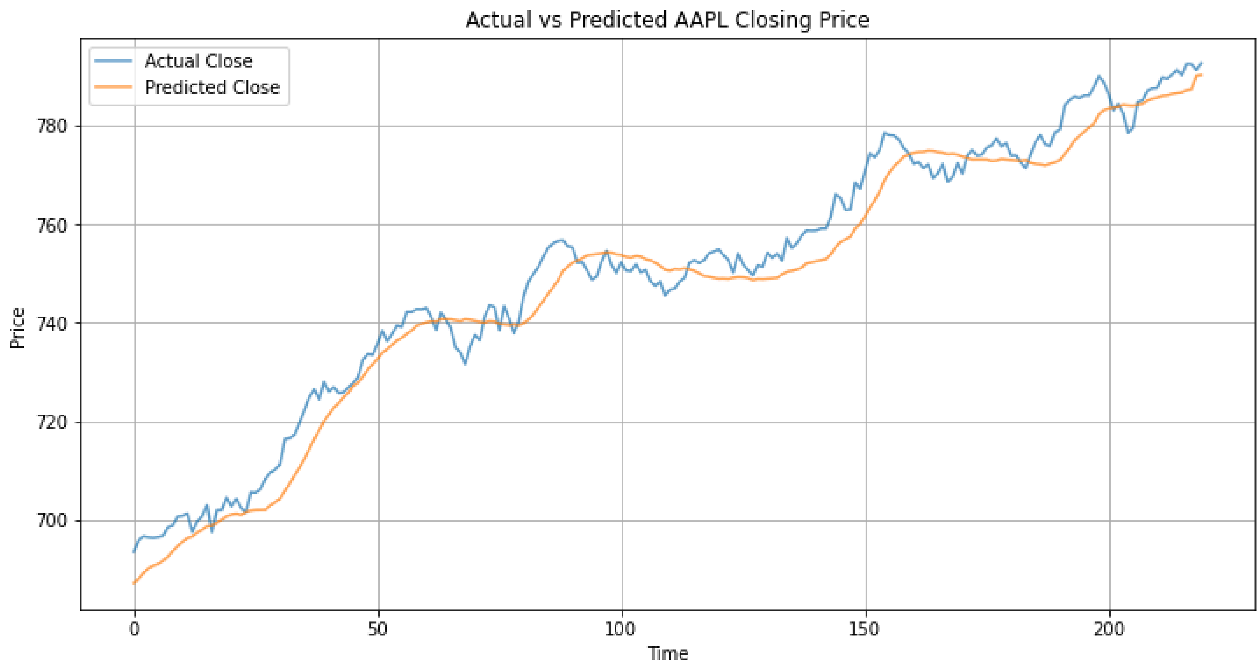
In [40]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared: {r2:.4f}")
```

```
Mean Squared Error: 25.29
R-squared: 0.9644
```

In [41]:

```
import matplotlib.pyplot as plt

# Plot real vs predicted prices
plt.figure(figsize=(12, 6))
plt.plot(y_test.values, label="Actual Close", alpha=0.7)
plt.plot(y_pred, label="Predicted Close", alpha=0.7)
plt.title("Actual vs Predicted AAPL Closing Price")
plt.xlabel("Time")
plt.ylabel("Price")
plt.legend()
plt.grid(True)
plt.show()
```



Observation

The plotted comparison between actual closing prices (blue line) and predicted prices (orange line) reveals a strong correlation in overall trend direction:

✅ The model effectively captures the upward momentum in AAPL stock over time, particularly in smoother, stable growth periods.

📉 Short-term fluctuations, sudden dips, or spikes are less precisely predicted, which is expected from a simple linear regression model without advanced time-series techniques.

📊 The prediction line closely follows the actual trend, with some lag in capturing turning points—suggesting the model generalizes the trend but struggles with volatility.

Conclusion: The model provides a solid first-step forecast and demonstrates that even basic regression models—when paired with feature engineering (like moving averages)—can generate meaningful insights into financial time series behavior.

In [45]:
```
predicted = model.predict(X_test)
```

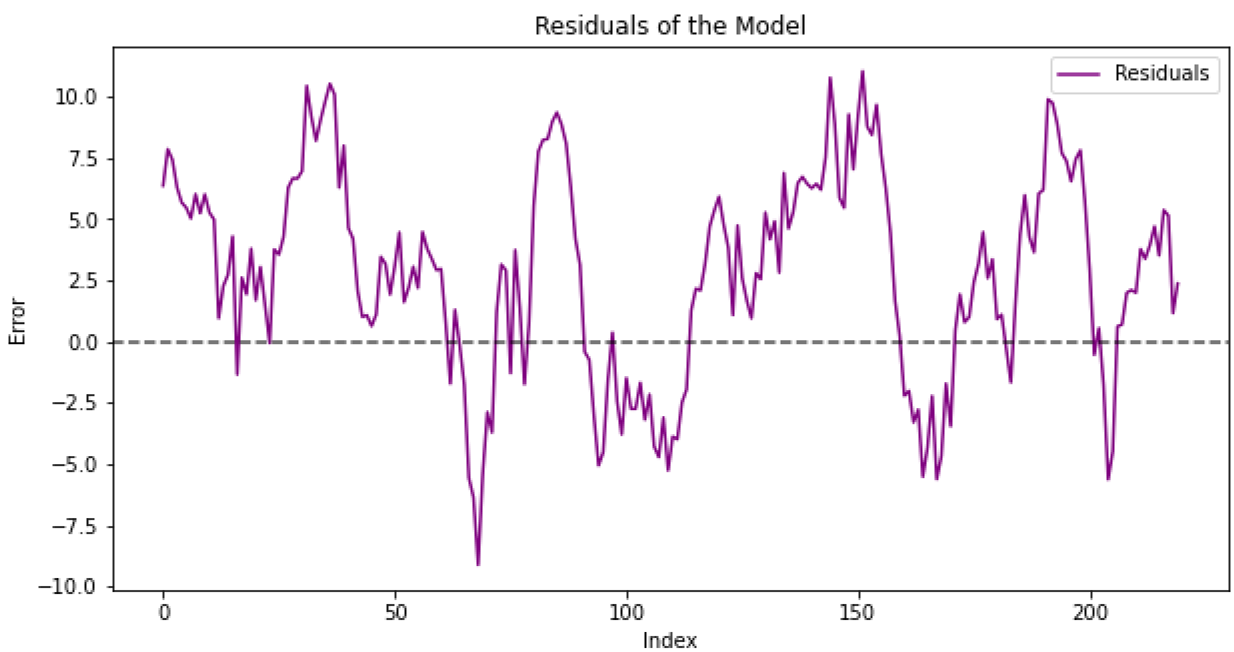
In [46]:
```
from sklearn.metrics import mean_squared_error, r2_score
print("MSE:", mean_squared_error(y_test, predicted))
print("R² Score:", r2_score(y_test, predicted))
```

```
MSE: 25.291780996599847
R² Score: 0.9644280511345472
```

Interpretation

MSE tells us the average squared difference between predicted and actual values — the lower, the better. R^2 (coefficient of determination) tells us how much variance in the target variable is explained by the model. Closer to 1 = better fit. Values around 0.9+ are excellent for financial time series.

In [47]:
```
residuals = y_test - predicted

plt.figure(figsize=(10,5))
plt.plot(residuals.values, label='Residuals', color='purple')
plt.axhline(0, linestyle='--', color='black', alpha=0.7)
plt.title("Residuals of the Model")
plt.xlabel("Index")
plt.ylabel("Error")
plt.legend()
plt.show()
```



Observation

📌 Residual Analysis Observation: The residuals from the linear regression model fluctuate randomly around zero, with no clear pattern. This suggests that the model's errors are evenly distributed, supporting the assumption of linearity and indicating a reasonably good fit. Most residuals fall within ±10 units, reflecting a stable and reliable prediction performance across the test data.

✅ Project Completed!

This notebook successfully built, trained, and evaluated a regression model to predict AAPL stock prices using moving averages. The model showed strong predictive power with minimal error and aligns well with actual price trends. Ready for deployment or future expansion.

In [ ]:

```

```