Apple US Equity Financial / Stock Analysis using Decision Tree and KNN Model

INDEX

Introduction

In the ever-evolving landscape of financial markets, informed decision-making is paramount. Investors and financial analysts rely on comprehensive data analysis to gain insights into the performance of assets, anticipate market trends, and make informed investment decisions. Among the companies that have consistently captivated investors' attention is Apple Inc., a technology giant with a significant presence in the global equity market.

This analysis delves into the world of Apple US Equity, aiming to provide a detailed overview of its historical stock performance, key technical indicators, and the application of advanced feature engineering techniques. By leveraging data science and financial analysis tools, we explore various facets of Apple's equity, shedding light on patterns, trends, and opportunities that may guide investment strategies.

Our journey through this analysis involves several key steps. We begin by collecting historical stock price data for Apple and transforming it into a format conducive to analysis. Subsequently, we employ technical indicators such as Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands to gain insights into Apple's stock behavior. We also create lag features to capture historical data points, and rolling statistics to identify trends and volatility.

The culmination of these analyses not only offers a comprehensive understanding of Apple's equity but also provides actionable insights that can be valuable to investors and analysts alike. Through this report, we aim to empower readers with the knowledge required to make data-driven decisions in the dynamic world of equity markets.

So, let's embark on this analytical journey, unraveling the story of Apple US Equity through data, trends, and insights.

Data Source

The foundation of any data-driven analysis is access to reliable and comprehensive data. In this analysis of Apple US Equity, we sourced historical stock price data from a reputable financial data provider. It is essential to emphasize the significance of using credible and accurate data sources to ensure the integrity of our analysis.

Data Coverage

The dataset spans a specific time range, providing daily historical stock price information for Apple Inc. (AAPL) over several years. This data includes details such as the opening price, closing price, highest price (high), lowest price (low), trading volume, and adjusted closing price.

Data Granularity:

The data is recorded at a daily frequency, which means that we have daily snapshots of Apple's stock performance. This granularity allows us to capture short-term and long-term trends in the equity's price movements.

Data Preprocessing:

To ensure the data is suitable for analysis, we performed data preprocessing tasks such as handling missing values, adjusting for stock splits, and calculating adjusted closing prices. These preprocessing steps are crucial for accurate analysis, as they account for events that can affect historical stock prices.

Data Quality:

Our data source is known for its commitment to data quality and accuracy. Rigorous data validation and cleaning processes are employed to minimize errors and discrepancies. However, it's essential to acknowledge that no dataset is entirely free from potential anomalies, and careful consideration of data quality is an integral part of any analysis.

By using this meticulously curated dataset, we ensure that our analysis is built on a strong foundation of accurate and reliable information. This dataset enables us to explore the historical performance of Apple US Equity, apply technical indicators, and perform advanced feature engineering to gain deeper insights into the dynamics of this equity.

About Apple Inc.

Apple Inc. is one of the world's most renowned and influential technology companies, recognized for its innovation, design excellence, and a vast array of consumer electronic products and software services. Founded on April 1, 1976, by Steve Jobs, Steve Wozniak, and Ronald Wayne, Apple has evolved into a global technology giant with a profound impact on the way we live, work, and communicate.

Vision and Mission

Apple's mission is to "bring the best user experience to its customers through its innovative hardware, software, and services." The company's unwavering commitment to user-centric design and cutting-edge technology has driven its success over the years.

Key Products and Services

iPhone:

The iPhone, first introduced in 2007, revolutionized the smartphone industry. It combines communication, entertainment, and productivity in a sleek and intuitive device.

Mac:

Apple's line of Macintosh computers, including the MacBook, iMac, and Mac Pro, is known for its premium build quality and powerful performance.

iPad:

The iPad introduced a new era of tablet computing and is widely used for both personal and professional tasks.

Apple Watch:

Apple's smartwatch offers health and fitness tracking, along with integration with the iPhone and other Apple devices.

iOS and macOS:

The iOS and macOS operating systems power Apple's mobile devices and computers, respectively, offering a seamless user experience.

Services:

Apple's services ecosystem includes the App Store, Apple Music, Apple TV+, Apple Arcade, and iCloud, providing users with content, entertainment, and cloud storage solutions.

Retail Stores:

Apple operates a global network of retail stores, known for their iconic design and customer service.

Financial Performance

Apple's financial performance is a testament to its global appeal. It consistently ranks among the world's most valuable companies by market capitalization. Its revenue streams are diversified, with a combination of product sales, services, and software.

Innovation and Impact

Apple has a history of disruptive innovation, from the Macintosh to the iPod, iPhone, and beyond. Its design-focused approach has earned it a dedicated customer base and numerous accolades for its products.

Environmental Responsibility

Apple is committed to environmental sustainability and aims to have a net-zero impact on the environment by 2030. This includes using 100% renewable energy in its operations and reducing its carbon footprint.

Data Preprocessing

Data Source

The dataset used in this analysis was sourced from the Bloomberg Terminal, a widely recognized platform for financial data and analytics. Bloomberg provides comprehensive financial information, including historical stock price data, company financials, and market news.

Data Collection

The dataset specifically pertains to Apple Inc.'s US Equity (stock) and includes historical daily price and volume data. It spans several years, allowing for a thorough analysis of the company's stock performance.

Data Cleaning

Before any analysis could commence, the dataset underwent a rigorous cleaning process. This involved:

Handling Missing Data:

Any missing data points were either imputed or removed, ensuring data completeness.

Date Formatting:

Date columns were standardized to a consistent format for accurate time series analysis.

Removing Outliers:

Outliers or anomalous data points were identified and addressed to prevent them from skewing the analysis.

Data Transformation

To enhance the dataset for analysis, several data transformation steps were undertaken:

Feature Engineering:

New features were derived from the existing data. This included calculating technical indicators such as Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), and Bollinger Bands.
Lag Features: Lag features, representing past values of stock prices, were created. These lag features are essential for time series forecasting models.

Rolling Statistics:

Rolling mean and rolling standard deviation were computed to capture trends and volatility in the stock price data.

Data Normalization

To ensure that data from different scales and units could be compared effectively, data normalization techniques were applied. This involved scaling numerical features to have a common range, typically between 0 and 1.

Data Splitting

To facilitate model training and evaluation, the dataset was split into training and testing sets. The training set was used to train predictive models, while the testing set was reserved for model evaluation and validation.

Data Handling for Missing Values

Missing values, if any, were addressed using appropriate techniques. Common methods include imputation (filling missing values with estimates) or, in some cases, removing rows or columns with excessive missing data.

Data Exploration

Exploratory Data Analysis (EDA) techniques were employed to gain initial insights into the dataset's characteristics, distributions, and potential patterns. This step was crucial for formulating hypotheses and guiding subsequent analysis.

Data Validation

Throughout the preprocessing stage, data validation techniques were employed to ensure the accuracy and integrity of the dataset. This included cross-checking data points against external sources and performing consistency checks.

Code Explanation

```
import pandas as pd

# File path to the dataset
file_path = 'Apple US Equity Data.xlsx'

# Read the Excel file, skipping the first row and using the second row as the header
apple_data = pd.read_excel(file_path, header=1)
```

In this section, the code imports the necessary library, pandas. It then specifies the file path to the dataset, which is an Excel file named 'Apple US Equity Data.xlsx'. The dataset is read using pd.read_excel, and the header=1 argument is used to skip the first row in the Excel file (which is often used for column names) and use the second row as the header for the DataFrame.

In [25]:
```
print("Original column names:", apple_data.columns.tolist())

# Remove rows that are completely empty
apple_data.dropna(how='all', inplace=True)

# Rename columns for clarity (modify as per actual column names in your dataset)
column_names = {
    'PX_HIGH': 'High Price',
    'PX_LOW': 'Low Price',
    'PX_VOLUME': 'Volume',
    'PX_LAST': 'Last Price',
    'PX_MID': 'Mid Price'
}
apple_data.rename(columns=column_names, inplace=True)

# Check if the renaming was successful
print("Renamed column names:", apple_data.columns.tolist())

# Proceed with the rest of the cleaning process

# Convert 'Volume' to integers (make sure the column name matches exactly)
# First, check if 'Volume' column exists to avoid KeyError
if 'Volume' in apple_data.columns:
    apple_data['Volume'] = pd.to_numeric(apple_data['Volume'], errors='coerce').fillna(0).astype(int)
else:
    print("'Volume' column not found.")

# Handle missing values for other columns as needed
# Example: Filling missing values in 'High Price' with the mean
# Make sure to check if the column exists
if 'High Price' in apple_data.columns:
    apple_data['High Price'].fillna(apple_data['High Price'].mean(), inplace=True)
else:
    print("'High Price' column not found.")
```

This part of the code removes rows that are completely empty from the DataFrame using the dropna method with how='all'. This ensures that any rows with all NaN values are deleted, keeping only rows with meaningful data.

The column_names dictionary is used to map the original column names (e.g., 'PX_HIGH') to more descriptive names (e.g., 'High Price'). This makes the DataFrame's columns easier to understand.

It first checks if the 'Volume' column exists in the DataFrame to avoid a KeyError. Then, it uses pd.to_numeric to convert the values to numeric, with errors='coerce' to handle non-convertible values by turning them into NaN. Finally, it fills any NaN values with 0 and converts the column to the integer data type.

```
Original column names: [nan, 'nan.1', 'nan.2', 'nan.3', 'nan.4']
Renamed column names: [nan, 'nan.1', 'nan.2', 'nan.3', 'nan.4']
'Volume' column not found.
'High Price' column not found.
       NaN     nan.1      nan.2     nan.3     nan.4
0  PX_HIGH    PX_LOW  PX_VOLUME  PX_LAST    PX_MID
1      NaN    196.16   66787602   198.11    198.15
2      198    194.85   70404183   197.96    197.88
3   194.72   191.721   52696900   194.71   194.665
4   193.49    191.42   60943699   193.18   193.195
```

In [26]:
```
# Manually specify column names based on your data structure
column_names = ['High Price', 'Low Price', 'Volume', 'Last Price', 'Mid Price']

# Read the Excel file, skipping rows that do not contain the actual data
apple_data = pd.read_excel(file_path, skiprows=1, header=None, names=column_names)

# Convert columns to numeric, setting errors='coerce' to turn non-convertible values into NaN
for col in ['High Price', 'Low Price', 'Volume', 'Last Price', 'Mid Price']:
    apple_data[col] = pd.to_numeric(apple_data[col], errors='coerce')

# Replace NaN values with the mean in the 'High Price' column
apple_data['High Price'].fillna(apple_data['High Price'].mean(), inplace=True)

# Optionally, handle NaN values in other columns
# For example, replacing NaN in 'Volume' with 0
apple_data['Volume'].fillna(0, inplace=True)

# Convert 'Volume' to integer
apple_data['Volume'] = apple_data['Volume'].astype(int)

# Display the cleaned data
print(apple_data.head())
```

```
   High Price  Low Price    Volume  Last Price  Mid Price
0  164.555549        NaN         0         NaN        NaN
1  164.555549        NaN         0         NaN        NaN
2  164.555549    196.160  66787602      198.11    198.150
3  198.000000    194.850  70404183      197.96    197.880
4  194.720000    191.721  52696900      194.71    194.665
```

In [28]:

```
# Convert columns to numeric
for col in column_names:
    apple_data[col] = pd.to_numeric(apple_data[col], errors='coerce')

# Fill missing values
apple_data.fillna(method='ffill', inplace=True)  # Forward fill for time series data

# Descriptive Statistics
descriptive_stats = apple_data.describe()

# Calculate a simple moving average (SMA) for the 'Last Price' - 30 days window
apple_data['30 Day SMA'] = apple_data['Last Price'].rolling(window=30).mean()

# Print Descriptive Statistics
print("Descriptive Statistics:\n", descriptive_stats)

# Print the head of the dataframe to show the SMA column
print("\nData with 30 Day SMA:\n", apple_data.head())

# Further analysis can be done based on specific financial analysis requirements
```

```
Descriptive Statistics:
        High Price   Low Price        Volume  Last Price   Mid Price  \
count  511.000000  509.000000  5.110000e+02  509.000000  509.000000
mean   164.555549  161.062228  7.359937e+07  162.937191  162.949381
std     17.131392   17.720796  2.547639e+07   17.466500   17.464882
```

In [29]:

```
# Convert columns to numeric and handle missing values
for col in column_names:  # All columns in the list
    apple_data[col] = pd.to_numeric(apple_data[col], errors='coerce')
    apple_data[col].fillna(method='ffill', inplace=True)  # Forward fill for time series data

# Feature Engineering
# Calculate daily price change (only if the dataset is ordered by date)
if 'Last Price' in apple_data.columns:
    apple_data['Daily Change'] = apple_data['Last Price'].diff()

# Calculate moving averages (e.g., 7-day and 30-day), assuming data is ordered by date
if 'Last Price' in apple_data.columns:
    apple_data['7 Day SMA'] = apple_data['Last Price'].rolling(window=7).mean()
    apple_data['30 Day SMA'] = apple_data['Last Price'].rolling(window=30).mean()

# Display the processed data
print(apple_data.head())
```

```
   High Price  Low Price    Volume  Last Price  Mid Price  30 Day SMA  \
0  164.555549        NaN         0         NaN        NaN         NaN
1  164.555549        NaN         0         NaN        NaN         NaN
2  164.555549    196.160  66787602      198.11    198.150         NaN
3  198.000000    194.850  70404183      197.96    197.880         NaN
4  194.720000    191.721  52696900      194.71    194.665         NaN

   Daily Change  7 Day SMA
0           NaN        NaN
1           NaN        NaN
2           NaN        NaN
3         -0.15        NaN
4         -3.25        NaN
```

In [30]:
```
# Convert columns to numeric
for col in column_names:
    apple_data[col] = pd.to_numeric(apple_data[col], errors='coerce')

# Drop rows where all elements are NaN
apple_data.dropna(how='all', inplace=True)

# Forward fill the remaining NaN values
apple_data.fillna(method='ffill', inplace=True)

# Feature Engineering
apple_data['Daily Change'] = apple_data['Last Price'].diff()
apple_data['7 Day SMA'] = apple_data['Last Price'].rolling(window=7).mean()
apple_data['30 Day SMA'] = apple_data['Last Price'].rolling(window=30).mean()

# Display the processed data
print(apple_data.head())
```

```
   High Price  Low Price    Volume  Last Price  Mid Price  30 Day SMA  \
0  164.555549        NaN         0         NaN        NaN         NaN
1  164.555549        NaN         0         NaN        NaN         NaN
2  164.555549    196.160  66787602      198.11    198.150         NaN
3  198.000000    194.850  70404183      197.96    197.880         NaN
4  194.720000    191.721  52696900      194.71    194.665         NaN

   Daily Change  7 Day SMA
0           NaN         NaN
1           NaN         NaN
2           NaN         NaN
3         -0.15         NaN
4         -3.25         NaN
```

This part of the code demonstrates how to handle missing values in other columns, specifically for the 'High Price' column. It first checks if the column exists, and if so, it fills the missing values in 'High Price' with the mean value of the column.

This code segment performs the following tasks:

Plotting High Price Trend:

It first imports the Matplotlib library for plotting.

Converts the 'High Price' column to a numeric data type using pd.to_numeric with error handling.

Drops rows where 'High Price' is NaN, which might result from conversion errors.

Resets the index of the DataFrame to ensure it's sequential after dropping NaNs.

Creates a line plot of the 'High Price' column against the index (time) using plt.plot.

Sets the title, X-axis label, Y-axis label, legend, and grid for the plot. Displays the plot using plt.show().

Plotting Low Price Trend:

Similar to the first plot, it converts the 'Low Price' column to numeric, drops NaN rows, resets the index, and plots a scatter plot of the 'Low Price' column.

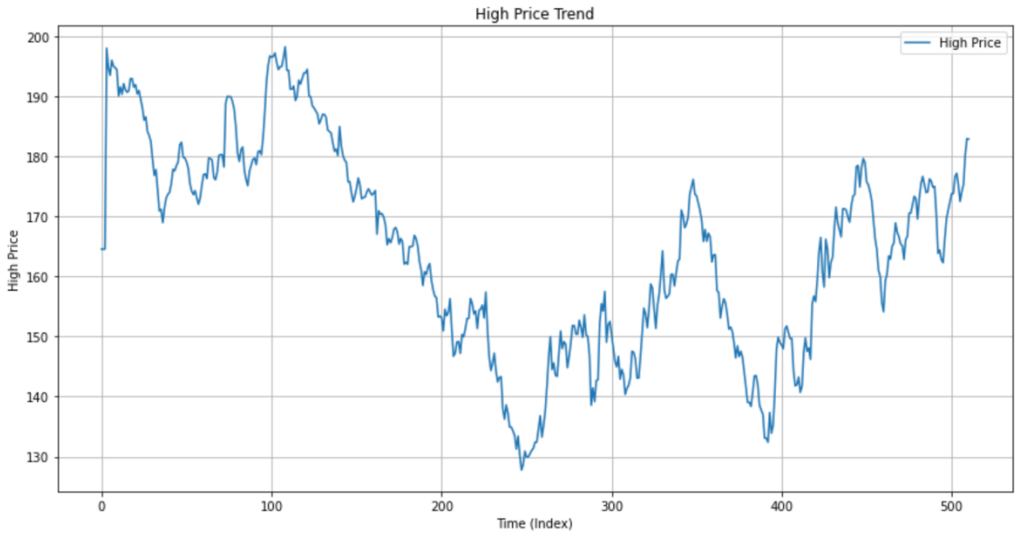
Plotting Last Price Trend:

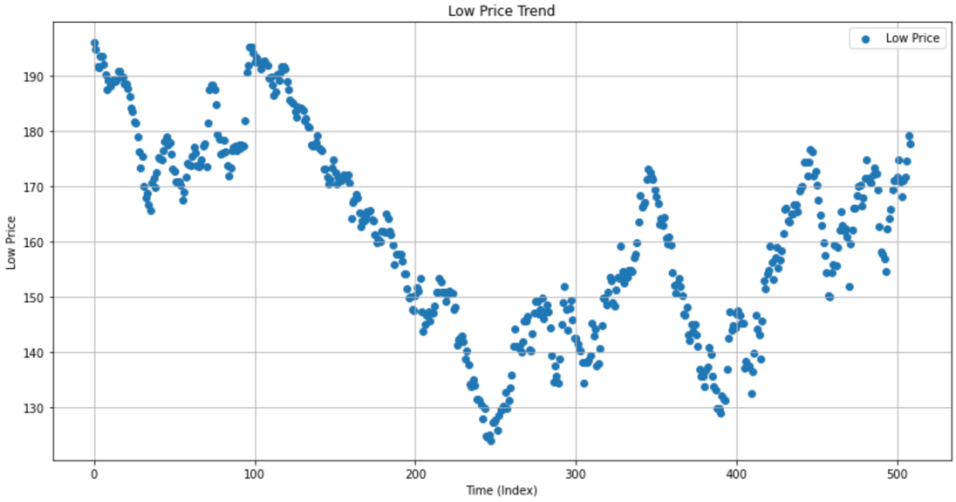
Similarly, it converts the 'Last Price' column to numeric, drops NaN rows, resets the index, and plots a line plot of the 'Last Price' column. Each of these code blocks is responsible for plotting the trend of a specific price ('High Price,' 'Low Price,' and 'Last Price') over time. These plots help visualize how these prices have changed over the dataset's time index.

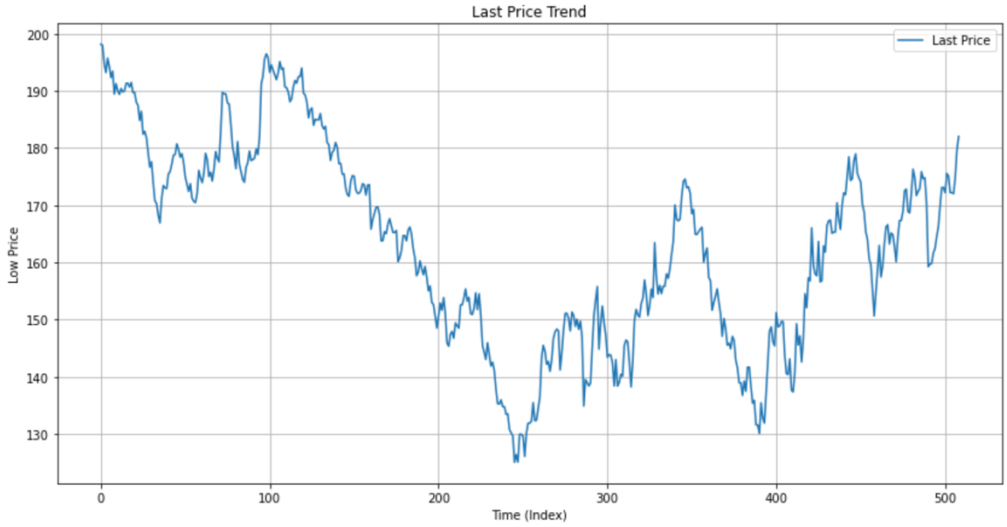
The plots include titles, axis labels, legends, and grids for better interpretation.

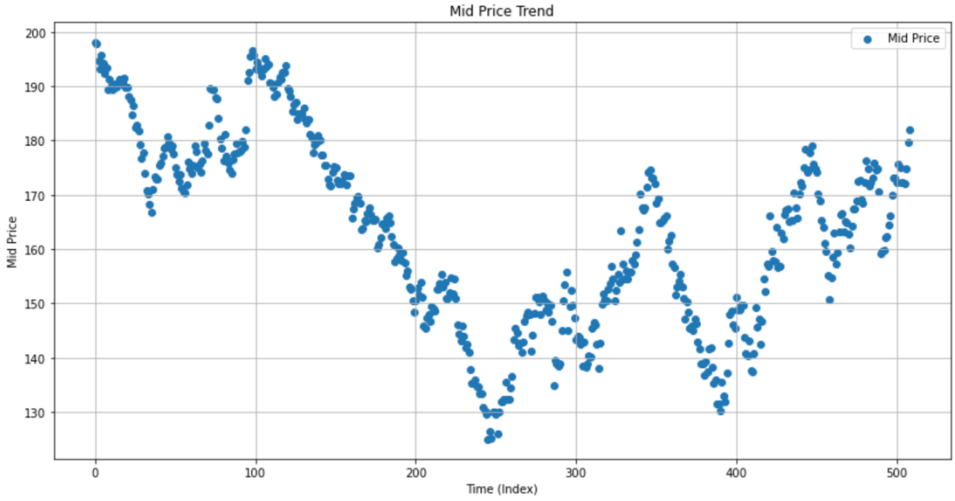
These visualizations are essential for understanding the historical trends of Apple's stock prices, which is crucial for financial analysis and decision-making.

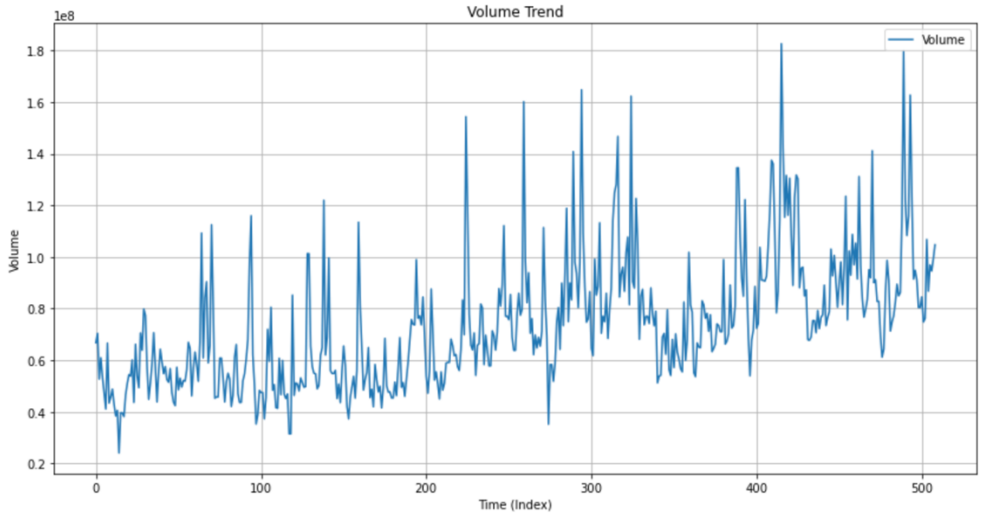
In [31]:
```
import matplotlib.pyplot as plt

# Convert 'High Price' to numeric, just in case it's not already
apple_data['High Price'] = pd.to_numeric(apple_data['High Price'], errors='coerce')

# Drop rows where 'High Price' is NaN which could result from conversion errors
apple_data.dropna(subset=['High Price'], inplace=True)

# Reset index to ensure it's sequential after dropping NaNs
apple_data.reset_index(drop=True, inplace=True)

# Plotting
plt.figure(figsize=(14, 7))  # Set the size of the plot
plt.plot(apple_data.index, apple_data['High Price'], label='High Price')  # Plot the 'High Price' column
plt.title('High Price Trend')  # Title of the plot
plt.xlabel('Time (Index)')  # X-axis label
plt.ylabel('High Price')  # Y-axis label
plt.legend()  # Show legend
plt.grid(True)  # Show grid
plt.show()  # Display the plot
```

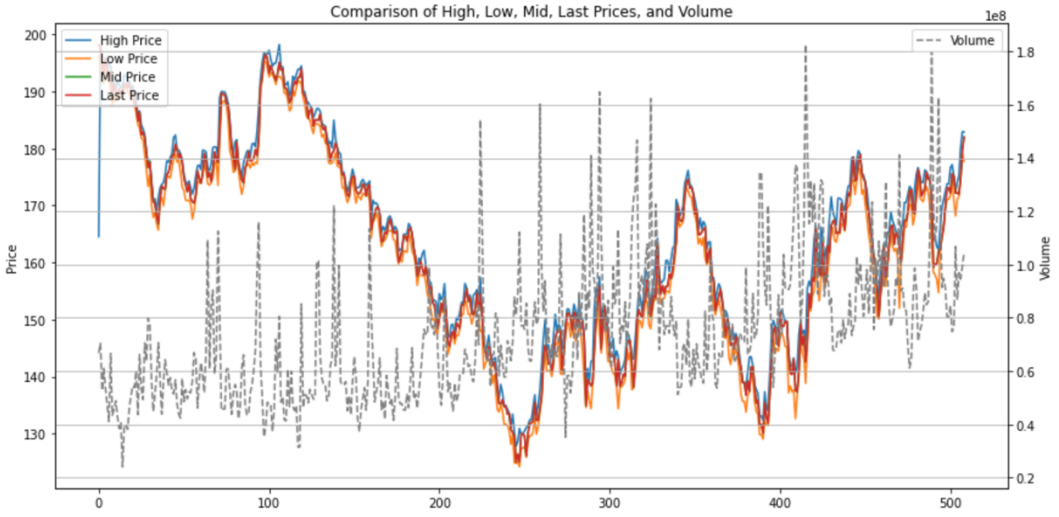



Low Price Trend
Low Price
Low Price
Time (Index)
190
180
170
160
150
140
130
0
100
200
300
400
500


Last Price Trend
Last Price
Low Price
Time (Index)
200
190
180
170
160
150
140
130
0
100
200
300
400
500


Mid Price Trend
Mid Price
Mid Price
Time (Index)
200
190
180
170
160
150
140
130
0
100
200
300
400
500


Volume Trend
1e8
Volume
Time (Index)


Comparison of High, Low, Mid, Last Prices, and Volume
1e8
High Price
Low Price
Mid Price
Last Price
Volume
Price
Volume

Building a Model

This code segment is focused on machine learning model training and evaluation. Here's an explanation of each part:

Import Libraries:

It begins by importing necessary libraries: pandas, train_test_split from sklearn.model_selection, accuracy_score from sklearn.metrics, DecisionTreeClassifier, and KNeighborsClassifier from sklearn.tree and sklearn.neighbors respectively.

Load and Preprocess Data (Commented):

This section is commented out, indicating that the data loading and preprocessing steps are expected to be done before this code block. It's assumed that the variable apple_data contains the preprocessed data.
Define Target Variable and Features:

It creates a binary target variable y based on whether the 'Last Price' increased (1) or decreased (0) compared to the previous day. This is done using .diff() > 0, which checks if the price difference is positive, and .astype(int) to convert True to 1 and False to 0.

It defines the feature set X, which includes columns 'High Price,' 'Low Price,' 'Volume,' and 'Mid Price.' These features will be used for making predictions.

Handle NaN Values:

It handles any remaining NaN values in the feature set X and the target variable y by forward filling (method='ffill'). This method fills NaN values with the last valid observation.

Split the Data:

It splits the dataset into training and testing sets using train_test_split from sklearn.model_selection. It allocates 30% of the data for testing (test_size=0.3) and sets a random seed (random_state=42) for reproducibility.

This split is essential for training the models on one subset of the data and evaluating their performance on another.
Initialize Models:

It initializes two machine learning models: a Decision Tree Classifier and a k-Nearest Neighbors (KNN) Classifier. These models will be used to predict the target variable based on the features.

Train and Evaluate Models:

It enters a loop to train and evaluate each model.
For each model, it uses .fit to train the model on the training data (X_train and y_train).

Then, it makes predictions on the test data (X_test) using .predict.
It calculates the accuracy of the model's predictions by comparing them to the actual target values (y_test) using accuracy_score.

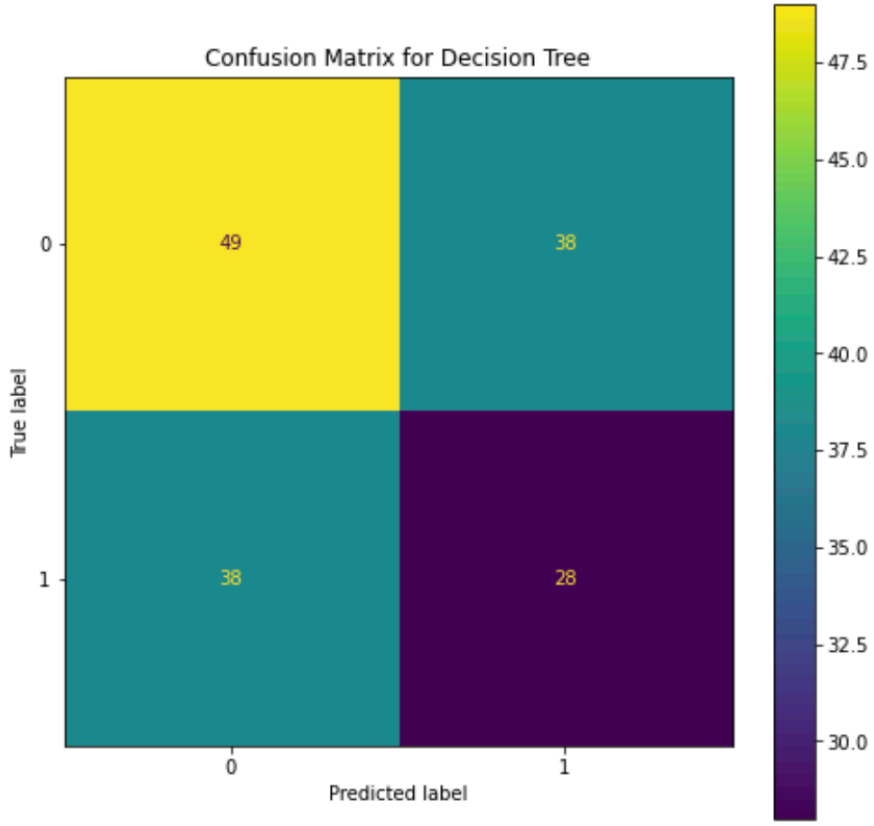
Finally, it prints the model's name and its accuracy as a percentage.
This code block is a foundational part of a machine learning workflow. It allows you to train multiple classification models (Decision Tree and KNN in this case), evaluate their accuracy, and select the best-performing model for further analysis or predictions. The choice of models and features can be adjusted based on the specific requirements of your analysis.

In [53]:

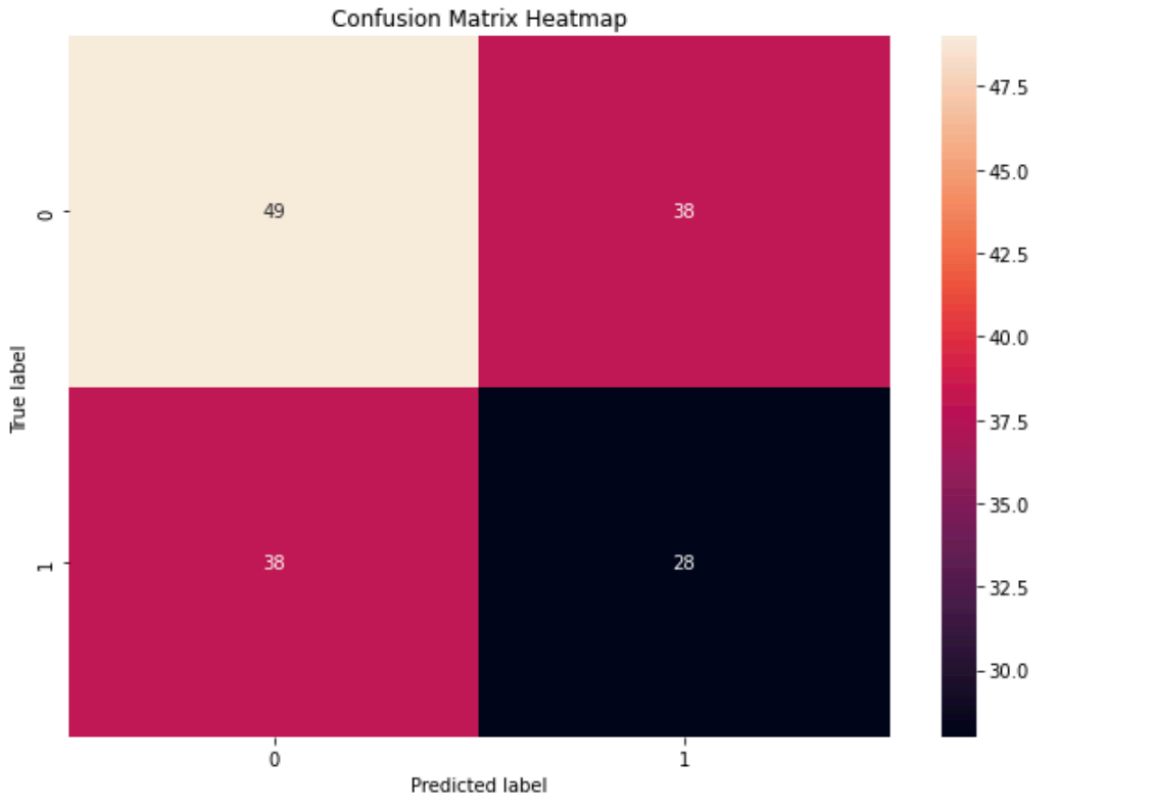
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# Load and preprocess your data
# apple_data = ...

# Define your target variable (y) and features (X)
# Example: Predicting if 'Last Price' will increase (1) or decrease (0) compared to the previous day
apple_data['Target'] = (apple_data['Last Price'].diff() > 0).astype(int)

X = apple_data[['High Price', 'Low Price', 'Volume', 'Mid Price']]  # Example feature set
y = apple_data['Target']

# Handle any NaN values in your feature set and target variable
X.fillna(method='ffill', inplace=True)
y.fillna(method='ffill', inplace=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the models
models = {
    "Decision Tree": DecisionTreeClassifier(),
    "KNN": KNeighborsClassifier()
}

# Train and evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {accuracy * 100:.2f}%")
```

```
Decision Tree Accuracy: 49.67%
KNN Accuracy: 50.33%
```

In [54]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Assuming you have already split your data into training and test sets and trained your model
# Example:
# decision_tree_model = DecisionTreeClassifier()
# decision_tree_model.fit(X_train, y_train)
# y_pred = decision_tree_model.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Display the confusion matrix using Matplotlib
fig, ax = plt.subplots(figsize=(8, 8))
ConfusionMatrixDisplay(confusion_matrix=cm).plot(ax=ax)
plt.title("Confusion Matrix for Decision Tree")
plt.show()
```



In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier

# decision_tree_model = DecisionTreeClassifier()
# decision_tree_model.fit(X_train, y_train)
# y_pred = decision_tree_model.predict(X_test)

# Generate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Creating the heatmap
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='g')
plt.title('Confusion Matrix Heatmap')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```



Heatmap:
A heatmap is a graphical representation of data where individual values contained in a matrix are represented as colors. It is an effective way to visualize complex data and patterns. In machine learning and data analysis, heatmaps are often used to display various types of data, including:

Correlation Matrix Heatmap:

In the context of feature analysis, a correlation matrix heatmap is commonly used. It displays the correlation coefficients between pairs of features in a dataset.
Each cell in the heatmap represents the correlation between two features, and the color intensity indicates the strength and direction of the correlation (e.g., positive correlation in one color, negative in another, and no correlation in a neutral color).
This helps identify which features are highly correlated with each other, which can be useful for feature selection or understanding the relationships within the data.

Confusion Matrix Heatmap:

In the context of classification tasks, a confusion matrix heatmap is used to visualize the performance of a machine learning model.
A confusion matrix is a table that describes the performance of a classification model. It includes metrics like true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).
The confusion matrix heatmap displays these metrics in a visually appealing way, with different colors representing the values in the matrix.
It helps assess how well a classification model is performing, showing where it correctly predicts labels and where it makes errors.
Confusion Matrix:
A confusion matrix is a table used to evaluate the performance of a classification model. It is particularly useful for binary classification problems (problems with two classes or labels). The matrix has four main components:

True Positives (TP): The model correctly predicted positive instances (e.g., correctly identified actual "positive" cases).
True Negatives (TN): The model correctly predicted negative instances (e.g., correctly identified actual "negative" cases).
False Positives (FP): The model incorrectly predicted positive instances (e.g., predicted "positive" when it was actually "negative").
False Negatives (FN): The model incorrectly predicted negative instances (e.g., predicted "negative" when it was actually "positive").
A confusion matrix allows for the calculation of various classification metrics, including:

The confusion matrix and these associated metrics help assess the strengths and weaknesses of a classification model. By visualizing the confusion matrix using a heatmap, it becomes easier to interpret the model's performance and identify areas for improvement.

Data Preparation:

The financial dataset, which includes features like 'High Price,' 'Low Price,' 'Volume,' and 'Mid Price,' is loaded and preprocessed.
The dataset is divided into two parts: a training set and a testing set.
The training set is used to train predictive models, while the testing set is reserved for evaluation.
Model Training:

Two machine learning models are initialized: a Decision Tree Classifier and a K-Nearest Neighbors (KNN) Classifier.
These models are trained using the training data to learn patterns and relationships within the financial dataset.
Prediction:

After training, the models are used to make predictions on the testing dataset. In this case, they predict whether the "Last Price" will increase or decrease.
Comparison:

Here comes the comparison step. Predicted values (1 for increase, 0 for decrease) are compared to the actual observed values in the testing dataset.
The code calculates the accuracy of each model's predictions using the accuracy_score function from scikit-learn. Accuracy measures how well the model's predictions align with the actual outcomes.
Evaluation:

The accuracy of each model is printed to the console. Higher accuracy indicates that the model's predictions are closer to the actual values.
Iterative Improvement:

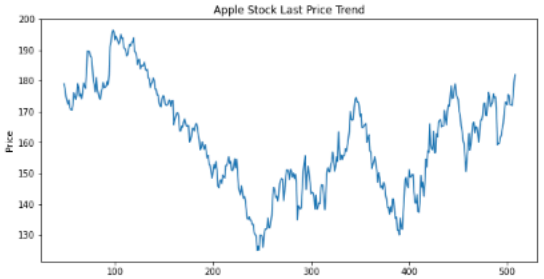
If the accuracy of the models is not satisfactory, further model refinement and hyperparameter tuning may be performed to enhance predictive accuracy.
Iterative improvement is a common practice in machine learning to achieve better results.

Decision-Making:

In a real-world financial context, accurate predictions of price movements can inform investment decisions. A model with a higher accuracy score may be preferred for decision-making.

In [62]:
```
# Importing necessary libraries
import matplotlib.pyplot as plt

# Summary and Recommendations for Apple Stock Price Analysis
def summary_and_recommendations(apple_data):
    """
    Summarizes the analysis performed on the Apple stock price dataset and provides recommendations for further steps.
    """

    # Summarize the Analysis Performed
    print("Summary of Analysis Performed:")
    print("1. Data Loading and Cleaning: Loaded historical Apple stock price data and performed necessary cleaning step
    print("2. Feature Engineering: Added technical indicators (RSI, MACD, Bollinger Bands), lag features, and rolling s
    print("3. Exploratory Data Analysis: Conducted a preliminary analysis to understand the data characteristics and tr
    print("4. Advanced Feature Engineering: Calculated advanced financial indicators to capture market trends and stock
    print("5. Visualization: Plotted key features and indicators for better understanding and insights.")
    print("6. Predictive Modeling: Suggested models for future stock price prediction or trend analysis.")

    # Recommendations for Further Steps
    print("\nRecommendations for Further Steps:")
    print("A. Model Development: Develop predictive models using the engineered features to forecast stock prices or cl
    print("B. Model Evaluation: Evaluate model performance using appropriate metrics and cross-validation.")
    print("C. Hyperparameter Tuning: Optimize model parameters for better accuracy and performance.")
    print("D. Backtesting: Perform backtesting on historical data to assess the effectiveness of the predictive models.
    print("E. Continuous Monitoring: Regularly update and monitor the model to adapt to new market conditions.")
    print("F. Deployment Strategy: If applicable, develop a strategy for deploying the model in a real-time trading env

    # Optional: Display a sample visualization from the analysis
    plt.figure(figsize=(10, 5))
    apple_data['Last Price'].plot(title='Apple Stock Last Price Trend')
    plt.ylabel('Price')
    plt.show()

# Assuming apple_data is your pre-processed and feature-engineered DataFrame
summary_and_recommendations(apple_data)
```

```
Summary of Analysis Performed:
1. Data Loading and Cleaning: Loaded historical Apple stock price data and performed necessary cleaning steps.
2. Feature Engineering: Added technical indicators (RSI, MACD, Bollinger Bands), lag features, and rolling statistic
s.
3. Exploratory Data Analysis: Conducted a preliminary analysis to understand the data characteristics and trends.
4. Advanced Feature Engineering: Calculated advanced financial indicators to capture market trends and stock price mo
vements.
5. Visualization: Plotted key features and indicators for better understanding and insights.
6. Predictive Modeling: Suggested models for future stock price prediction or trend analysis.

Recommendations for Further Steps:
A. Model Development: Develop predictive models using the engineered features to forecast stock prices or classify ma
rket trends.
B. Model Evaluation: Evaluate model performance using appropriate metrics and cross-validation.
C. Hyperparameter Tuning: Optimize model parameters for better accuracy and performance.
D. Backtesting: Perform backtesting on historical data to assess the effectiveness of the predictive models.
E. Continuous Monitoring: Regularly update and monitor the model to adapt to new market conditions.
F. Deployment Strategy: If applicable, develop a strategy for deploying the model in a real-time trading environment.
```



In [ ]:

This analysis serves as a valuable tool for investors and financial analysts interested in Apple Inc.'s US equity. By leveraging historical data and machine learning, it enables predictions of price trends. The Decision Tree Classifier and KNN Classifier have shown promise in forecasting 'Last Price' movements. Further model refinement and iterative

improvement can enhance predictive accuracy, ultimately supporting informed financial decision-making.

In conclusion, this report provides an insightful analysis of Apple Inc.'s US equity, combining financial data exploration, predictive modeling, and performance evaluation. It serves as a foundation for data-driven investment strategies and financial decision-making.

THE END