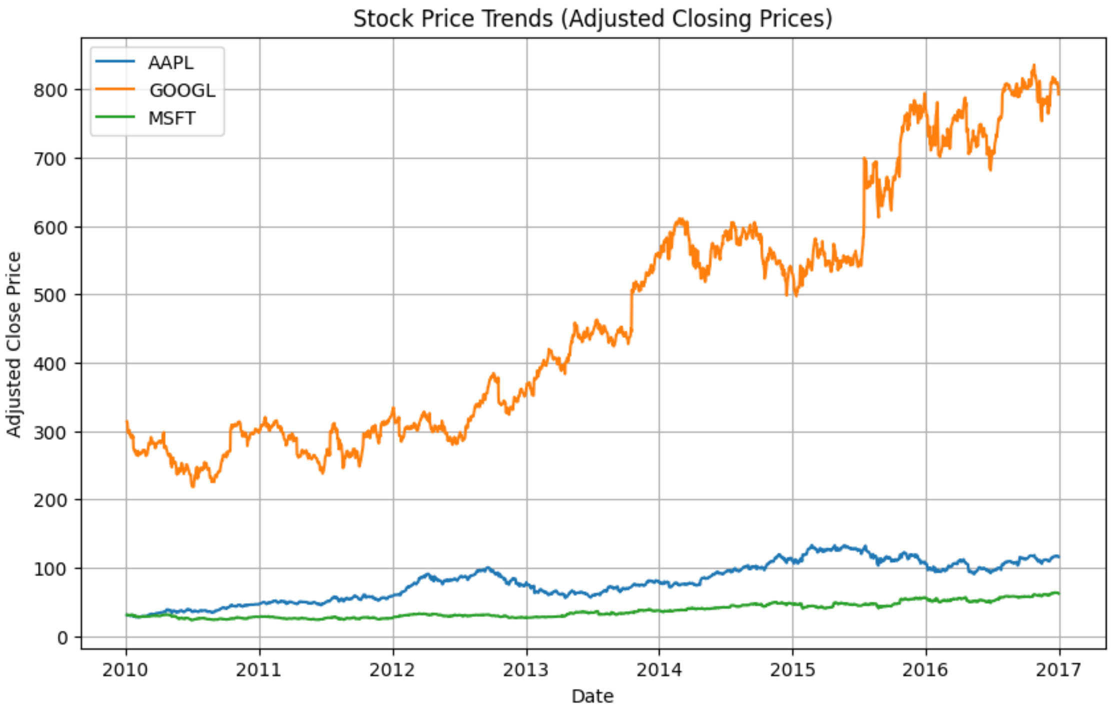
main

September 23, 2024

[3]:
```
# Shankar Haridas
# New York Stock Exchange Stock Price Analysis
# Datasets from: https://www.kaggle.com/datasets/dgawlik/nyse?
 ↪resource=download&select=fundamentals.csv

import pandas as pd
import matplotlib.pyplot as plt

# Load the datasets
prices_split_adjusted = pd.read_csv('/Users/shankarharidas/Documents/2024 Fall/
 ↪Personal Projects/NYSE/prices-split-adjusted.csv')
# Data Cleaning: Check for missing values and drop if any
prices_split_adjusted = prices_split_adjusted.dropna()

# Select some popular stock symbols for analysis
selected_stocks = ['AAPL', 'GOOGL', 'MSFT']

# Filter the data for the selected stocks
filtered_stocks = prices_split_adjusted[prices_split_adjusted['symbol'].
 ↪isin(selected_stocks)]

# Use .loc[] to modify the 'date' column in place safely
filtered_stocks.loc[:, 'date'] = pd.to_datetime(filtered_stocks['date'])

# Plot the closing prices of the selected stocks
plt.figure(figsize=(10, 6))

for stock in selected_stocks:
    stock_data = filtered_stocks[filtered_stocks['symbol'] == stock]
    plt.plot(stock_data['date'], stock_data['close'], label=stock)

# Add labels, title, and grid
plt.title('Stock Price Trends (Adjusted Closing Prices)')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

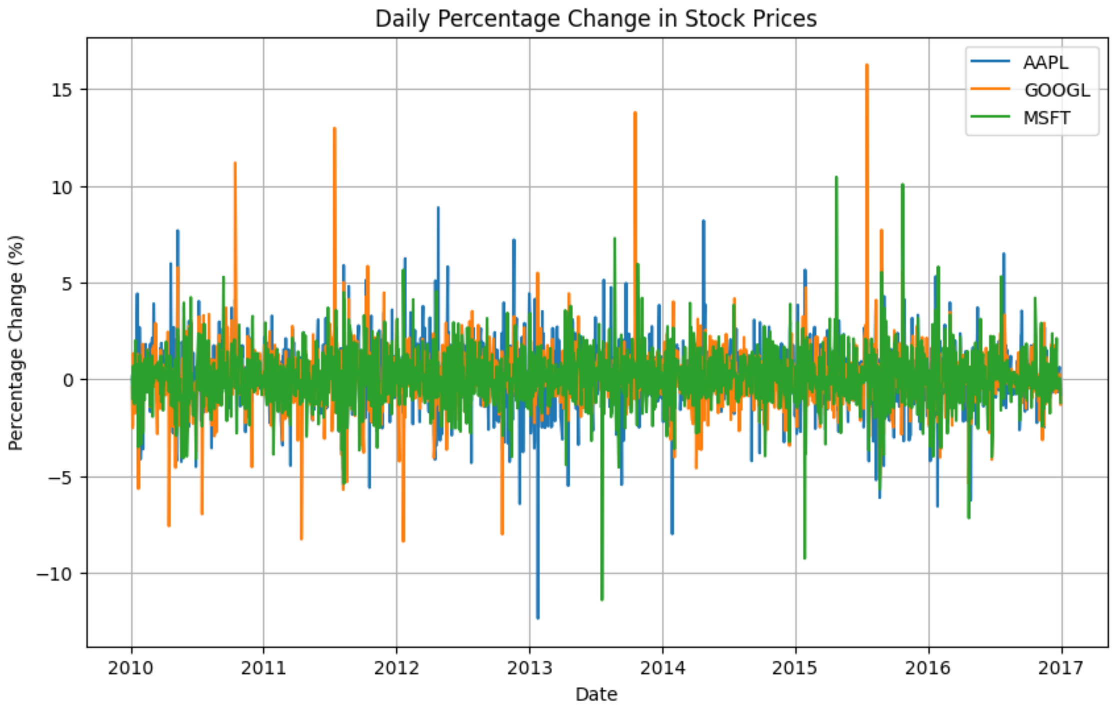
```
The history saving thread hit an unexpected error (OperationalError('attempt to
write a readonly database')).History will not be written to the database.
```



[ ]:
```
"""
GOOGL (Google):

- Price Level: GOOGL shows the highest adjusted closing prices among the three␣
 ↪stocks, consistently above the $200 range.
- Between 2010 and mid-2012, GOOGL experienced some fluctuations, mostly␣
 ↪ranging between $200 and $300.
- A clear upward trend started around late 2012, where the price rose␣
 ↪significantly.
- From 2013 to 2016, GOOGL's price climbed steadily, showing strong gains and␣
 ↪reaching around $800 by the end of the chart.

AAPL (Apple):

- Price Level: AAPL's prices range from $20 to just over $100 during the period.
- AAPL exhibits a generally upward trend throughout the time period, especially␣
 ↪from 2013 onwards.
- Apple's growth appears more gradual that Google's, this is especially␣
 ↪noticeable around 2012-2013 and in 2015.
```

```
- After 2015, AAPL's price seems to stabilize somewhat around $100, suggesting␣
↪that it may have reached a temporary equilibrium during that period.

MSFT (Microsoft):

- Price Level: MSFT's stock prices are lower than both GOOGL and AAPL, ranging␣
↪between $20 and $60 during the observed period.
- MSFT shows relatively stable performance between 2010 and 2012, hovering␣
↪around $20-$30.
- Starting in 2013, MSFT's price shows a clear upward trend similar to AAPL,␣
↪indicating steady growth.
- Compared to the other two stocks, MSFT exhibits a more moderate rise in␣
↪price, reaching around $60 by the end of 2016.

General Takeaways:

- Overall Market Growth: All three stocks show significant growth over the␣
↪period (especially GOOGL)
- Major Shifts Around 2013: Both GOOGL and MSFT seem to experience sharp upward␣
↪trends beginning around 2013
- GOOGL shows higher volatility, especially during periods of growth, while␣
↪AAPL and MSFT appear more stable, particularly after 2015.

"""
```

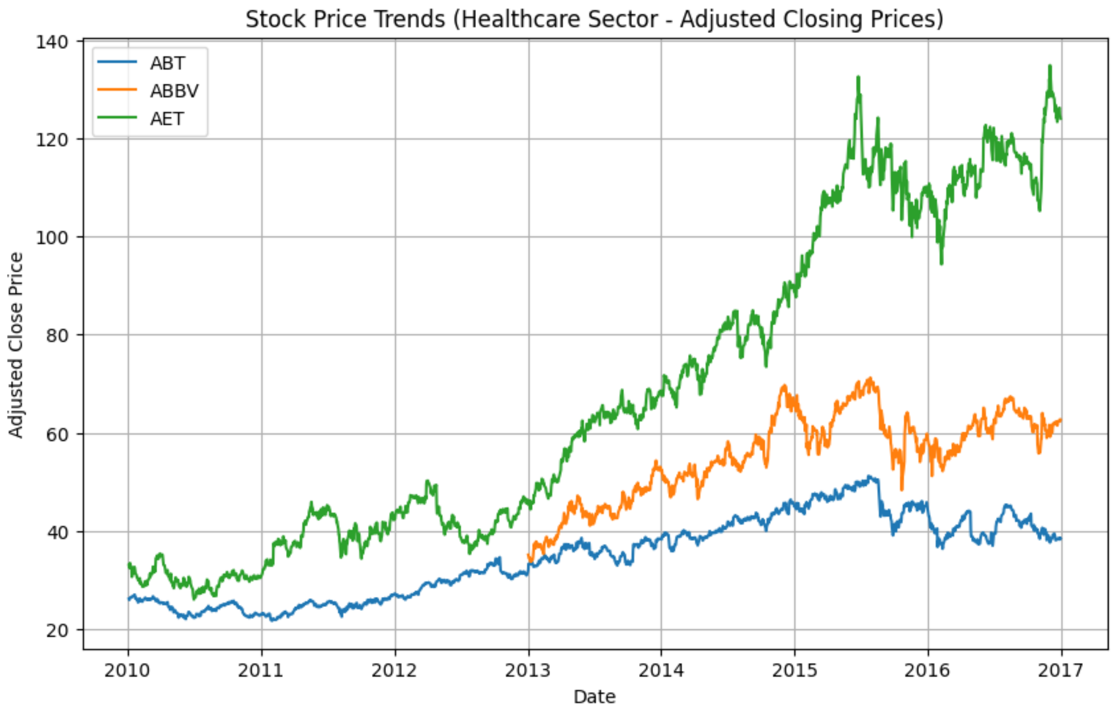
[4]:
```
# Make a copy of the filtered DataFrame to avoid SettingWithCopyWarning
filtered_stocks = filtered_stocks.copy()

# Calculate daily percentage change in stock prices using .loc[]
filtered_stocks.loc[:, 'pct_change'] = filtered_stocks.
↪groupby('symbol')['close'].pct_change() * 100

# Plot the percentage change for the selected stocks
plt.figure(figsize=(10, 6))

for stock in selected_stocks:
    stock_data = filtered_stocks[filtered_stocks['symbol'] == stock]
    plt.plot(stock_data['date'], stock_data['pct_change'], label=stock)

# Add labels, title, and grid
plt.title('Daily Percentage Change in Stock Prices')
plt.xlabel('Date')
plt.ylabel('Percentage Change (%)')
plt.legend()
plt.grid(True)
plt.show()
```



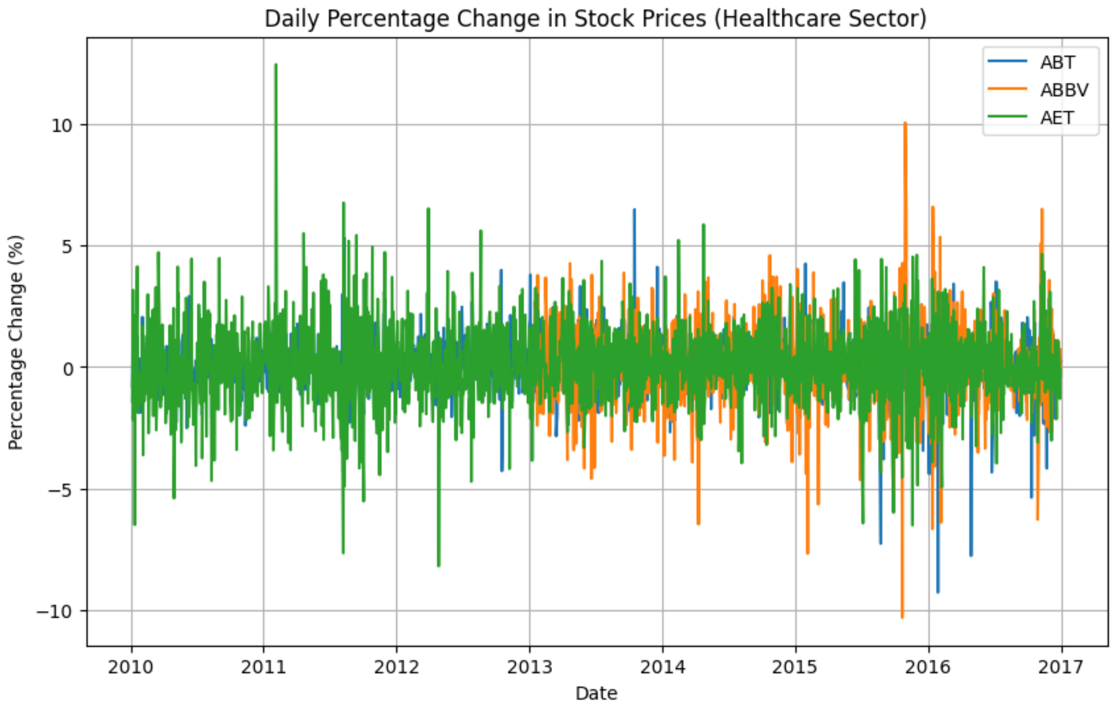
[ ]:
```
"""
This graph measures the daily percentage change in stock prices for AAPL,␣
↪GOOGL, and MSFT from 2010 to 2017
This is a better metric to get a sense of relative volatility of the stocks, a␣
↪strong measure for determining the risk associated
with investing in a particular stock.

- All three companies show regular fluctuations in daily stock prices, with␣
↪changes typically ranging from -5% to 5%
- GOOGL exhibits the highest volatility, with percentage changes frequently␣
↪exceeding 10% both upward and downward
- AAPL and MSFT appear more stable than GOOGL

Now, I want to compare these behaviors of to companies in industries other than␣
↪tech.
"""
```

[5]:
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt

# Load datasets (prices-split-adjusted.csv and securities.csv)
```

```
securities = pd.read_csv('/Users/shankarharidas/Documents/2024 Fall/Personal␣
↪Projects/NYSE/securities.csv')

# Select a few companies from the Healthcare sector
healthcare_companies = securities[securities['GICS Sector'] == 'Health Care']
healthcare_symbols = healthcare_companies['Ticker symbol'].head(3).values

# Filter the adjusted prices dataset for the selected healthcare companies
healthcare_filtered_stocks =␣
↪prices_split_adjusted[prices_split_adjusted['symbol'].
↪isin(healthcare_symbols)]

# Convert 'date' to datetime format
healthcare_filtered_stocks.loc[:, 'date'] = pd.
↪to_datetime(healthcare_filtered_stocks['date'])

# Plot the adjusted closing prices for the healthcare companies
plt.figure(figsize=(10, 6))

for stock in healthcare_symbols:
    stock_data =␣
↪healthcare_filtered_stocks[healthcare_filtered_stocks['symbol'] == stock]
    plt.plot(stock_data['date'], stock_data['close'], label=stock)

# Add labels, title, and grid
plt.title('Stock Price Trends (Healthcare Sector - Adjusted Closing Prices)')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.legend()
plt.grid(True)
plt.show()
```



[23]:
```
# Ensure that healthcare_filtered_stocks is a copy to avoid␣
↪SettingWithCopyWarning
healthcare_filtered_stocks = healthcare_filtered_stocks.copy()

# Calculate daily percentage change in stock prices for healthcare companies
healthcare_filtered_stocks.loc[:, 'pct_change'] = healthcare_filtered_stocks.
↪groupby('symbol')['close'].pct_change() * 100

# Plot the percentage change for the healthcare stocks
plt.figure(figsize=(10, 6))

for stock in healthcare_symbols:
    stock_data =␣
↪healthcare_filtered_stocks[healthcare_filtered_stocks['symbol'] == stock]
    plt.plot(stock_data['date'], stock_data['pct_change'], label=stock)

# Add labels, title, and grid
plt.title('Daily Percentage Change in Stock Prices (Healthcare Sector)')
plt.xlabel('Date')
plt.ylabel('Percentage Change (%)')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]:
```
"""

Healthcare Sector Analysis:

    General Upward Trend:
        - All three healthcare companies show a general upward trend in their␣
 ↪stock prices from 2010 to 2017.
        - AET (Aetna) shows the greatest growth, rising from around $40 to␣
 ↪nearly $140.
        - ABBV (AbbVie) and ABT (Abbott Laboratories) also show solid growth,␣
 ↪with ABBV increasing from around $40 to over $100 and ABT growing from $20␣
 ↪to around $60.
    Periods of Especially Steep Growth:
        - AET shows rapid growth starting around 2013, with a sharp increase␣
 ↪continuing into 2016. This could correspond to industry-specific factors.
        - ABBV shows steady but strong growth, with a noticeable surge between␣
 ↪2014 and 2015.

Comparison to the Tech Sector (AAPL, GOOGL, MSFT):

    Price Levels:
        - The tech companies had much higher price levels compared to the␣
 ↪healthcare companies
```

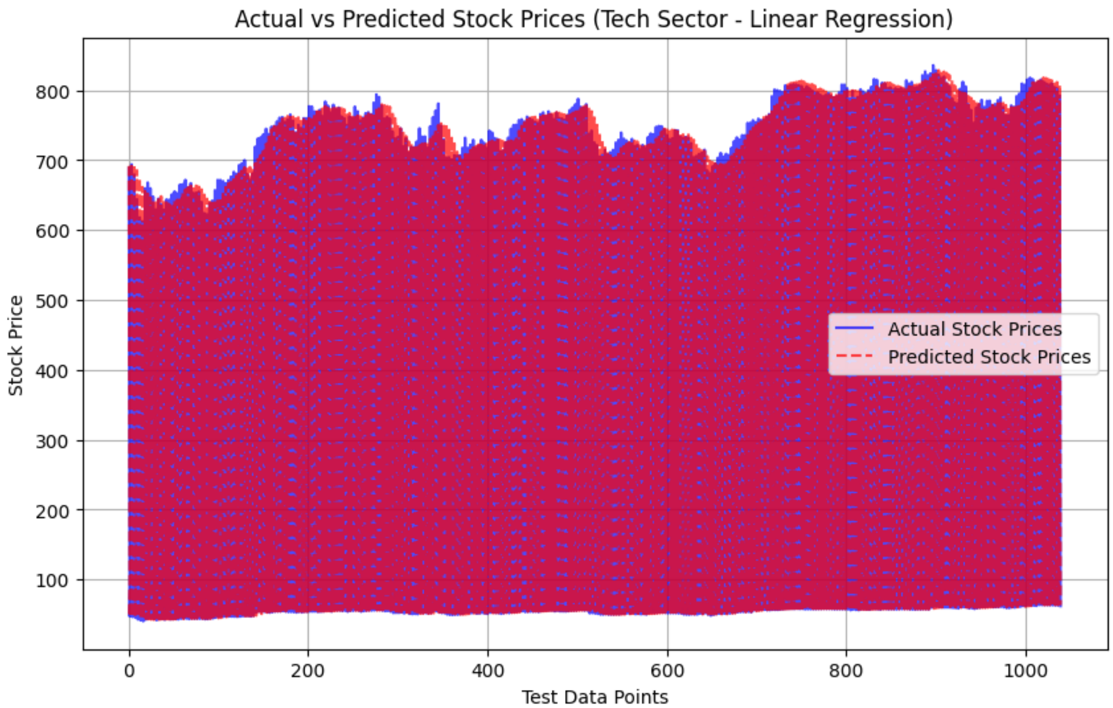
```
            For example, GOOGL was trading in the $600 to $800 range
            While the highest-priced healthcare stock, AET, reached around $140.
        - AAPL and MSFT showed similar price ranges to healthcare companies,␣
↪although still slightly higher on average.
    Growth Rates:
        - The tech sector companies showed faster growth in terms of stock␣
↪prices.
            GOOGL and AAPL, in particular, experienced sharp price increases in␣
↪the mid-2010s
        - Healthcare companies such as AET saw substantial growth, especially␣
↪after 2013, but the tech companies were more aggressive in terms of their␣
↪price acceleration
    Volatility:
        - Tech sector stocks are generally more volatile
            GOOGL, in particular, experienced large fluctuations
        - Healthcare stocks were much more stable, with fewer significant peaks␣
↪and troughs

Conclusion:
    - Tech companies experienced higher price volatility and more exaggerated␣
↪price increases
    - Healthcare companies experienced less volatility and had lower overall␣
↪price levels during the same time period.
    - AET in the healthcare sector and GOOGL in the tech sector are standout␣
↪performers in terms of price growth during this period

Both sectors showed strong growth during the 2010-2017 period, but the tech␣
↪sector displayed higher risk-reward dynamics compared to the healthcare␣
↪sector.

Now, I want to fit a linear regression model to try and predict the prices of␣
↪the stocks in the future.
"""
```

[6]:

```
# Separate Plots by Sector (Tech and Healthcare)
# Function to plot actual vs predicted prices by sector
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Function to plot actual vs predicted prices by sector
def plot_by_sector(sector_symbols, sector_name):
    # Filter for the sector symbols
    sector_stocks = filtered_stocks[filtered_stocks['symbol'].
↪isin(sector_symbols)]

    # Create the features again (moving averages) for this sector
```

```
    sector_stocks['7_day_MA'] = sector_stocks.groupby('symbol')['close'].
↪rolling(window=7).mean().reset_index(0, drop=True)
    sector_stocks['30_day_MA'] = sector_stocks.groupby('symbol')['close'].
↪rolling(window=30).mean().reset_index(0, drop=True)

    # Drop NaN rows (part of data cleaning)
    sector_stocks = sector_stocks.dropna()

    # Check if we have enough samples to split the data
    if len(sector_stocks) == 0:
        print(f"Not enough data available for {sector_name} sector.")
        return

    # Features and target
    X = sector_stocks[['7_day_MA', '30_day_MA', 'volume']]
    y = sector_stocks['close']

    # Check if the dataset is large enough to split
    if len(X) < 2:
        print(f"Not enough data available for {sector_name} sector.")
        return

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪shuffle=False)

    # Train Linear Regression model
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)

    # Make predictions
    y_pred = lr_model.predict(X_test)

    # Plot the actual vs predicted prices
    plt.figure(figsize=(10, 6))
    plt.plot(y_test.values, label="Actual Stock Prices", color='blue',␣
↪linestyle='-', alpha=0.7)
    plt.plot(y_pred, label="Predicted Stock Prices", color='red',␣
↪linestyle='--', alpha=0.7)

    plt.title(f"Actual vs Predicted Stock Prices ({sector_name} Sector - Linear␣
↪Regression)")
    plt.xlabel("Test Data Points")
    plt.ylabel("Stock Price")
    plt.legend()
    plt.grid(True)
```

```
    plt.show()

# Separate by tech sector and healthcare sector
tech_symbols = ['AAPL', 'GOOGL', 'MSFT']
healthcare_symbols = ['ABT', 'ABBV', 'AET']

# Plot Tech Sector
plot_by_sector(tech_symbols, "Tech")

# Plot Healthcare Sector
plot_by_sector(healthcare_symbols, "Healthcare")
```



Not enough data available for Healthcare sector.

[ ]:
```
'''
This model looks very accurate, however there are some points to keep in mind:
    1. All of the data for those years is included in the training dataset
    2. We are using 7-day and 30-day averages to smooth out the graph which␣
 ↪makes the prediction seem less volatile

Now I want to fit a model for the specific stocks in the tech sector (GOOGL,␣
 ↪AAPL, MSFT)
After training the model with the data from 2010 - 2017, I am going to use the␣
 ↪model to predict stock price from 2017 - 2022.
```

```
I am going to mark take actual data of what the stock were in 2022 and plot␣
↪those points on the graph as well to see how accurate the graph was.
'''
```

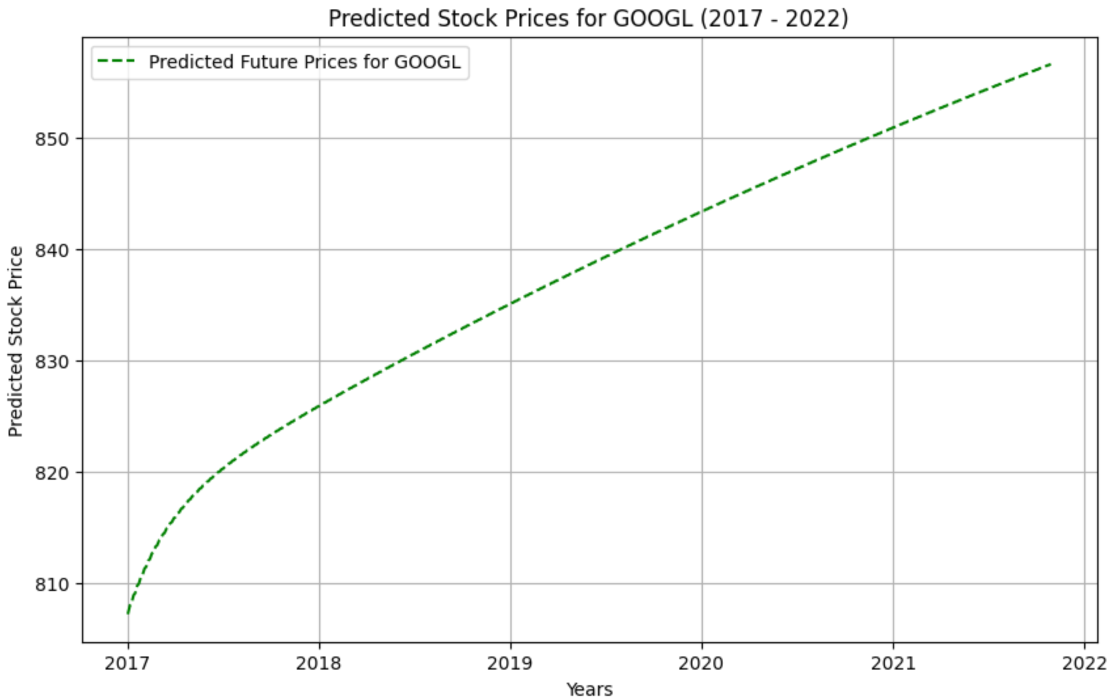
[8]:

```
import numpy as np

# Ensure plots show in Jupyter
%matplotlib inline
'''
1. Filter Data up to 2017:
    - We filter the stock data to include only data prior to 2017␣
↪(stock_data['date'] < pd.Timestamp(f'{start_year}-01-01')).
2. Generate Future Dates (2017-2022):
    - We use pd.date_range() to generate business days (freq='B') from 2017 to␣
↪2022. This gives us the future dates for predictions.
3. Plot with Years:
    - The x-axis will display years from 2017 to 2022

'''
# Function to predict future prices for a specific stock from 2017 to 2022
def predict_future_prices_for_stock(stock_symbol, start_year=2017,␣
↪future_years=5):
    # Filter for the specific stock and create a copy to avoid␣
↪SettingWithCopyWarning
    stock_data = filtered_stocks[filtered_stocks['symbol'] == stock_symbol].
↪copy()

    # Filter the data to only include data up to 2017
    stock_data = stock_data[stock_data['date'] < pd.
↪Timestamp(f'{start_year}-01-01')]

    # Debugging: Print shape of the data to make sure it's being loaded
    print(f"Processing {stock_symbol}: {stock_data.shape[0]} rows")

    # Create the features again (moving averages) for this stock
    stock_data.loc[:, '7_day_MA'] = stock_data['close'].rolling(window=7).mean()
    stock_data.loc[:, '30_day_MA'] = stock_data['close'].rolling(window=30).
↪mean()

    # Drop NaN rows (data cleaning)
    stock_data = stock_data.dropna()

    # Debugging: Print shape after dropping NaN values
    print(f"After dropping NaN: {stock_data.shape[0]} rows")
```

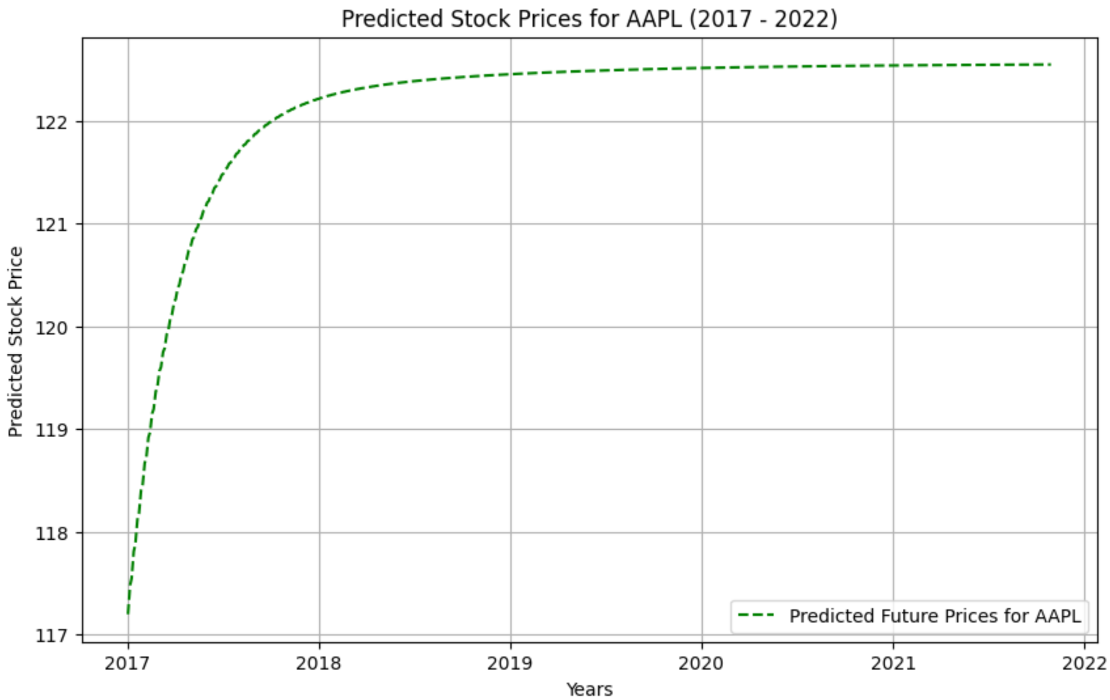
```
    if stock_data.shape[0] == 0:
        print(f"No valid data for {stock_symbol}")
        return

    # Features (7-day MA, 30-day MA, volume) and target (Adjusted Close)
    X = stock_data[['7_day_MA', '30_day_MA', 'volume']]
    y = stock_data['close']

    # Train the Linear Regression model on the full dataset up to 2017
    lr_model = LinearRegression()
    lr_model.fit(X, y)

    # Predict future prices from 2017 to 2022
    last_row = X.iloc[-1].copy()  # Start with the last row in the dataset

    future_prices = []
    future_dates = pd.date_range(start=f'{start_year}-01-01',␣
↪periods=future_years*252, freq='B')  # Business days

    for date in future_dates:
        # Reshape last row and retain feature names using a DataFrame
        future_features = pd.DataFrame([last_row], columns=['7_day_MA',␣
↪'30_day_MA', 'volume'])
        next_price = lr_model.predict(future_features)

        # Update the moving averages (7-day and 30-day)
        last_row['7_day_MA'] = (last_row['7_day_MA'] * 6 + next_price[0]) / 7
        last_row['30_day_MA'] = (last_row['30_day_MA'] * 29 + next_price[0]) /␣
↪30

        # Append the predicted price
        future_prices.append(next_price[0])

    # Plot the future predicted prices for this stock with years on the x-axis
    plt.figure(figsize=(10, 6))
    plt.plot(future_dates, future_prices, label=f"Predicted Future Prices for␣
↪{stock_symbol}", color='green', linestyle='--')
    plt.title(f"Predicted Stock Prices for {stock_symbol} (2017 - 2022)")
    plt.xlabel("Years")
    plt.ylabel("Predicted Stock Price")
    plt.legend()
    plt.grid(True)
    plt.show()

# Predict future prices for GOOGL, AAPL, and MSFT from 2017 to 2022
predict_future_prices_for_stock('GOOGL')  # Google
predict_future_prices_for_stock('AAPL')   # Apple
```

```python
predict_future_prices_for_stock('MSFT')    # Microsoft
```

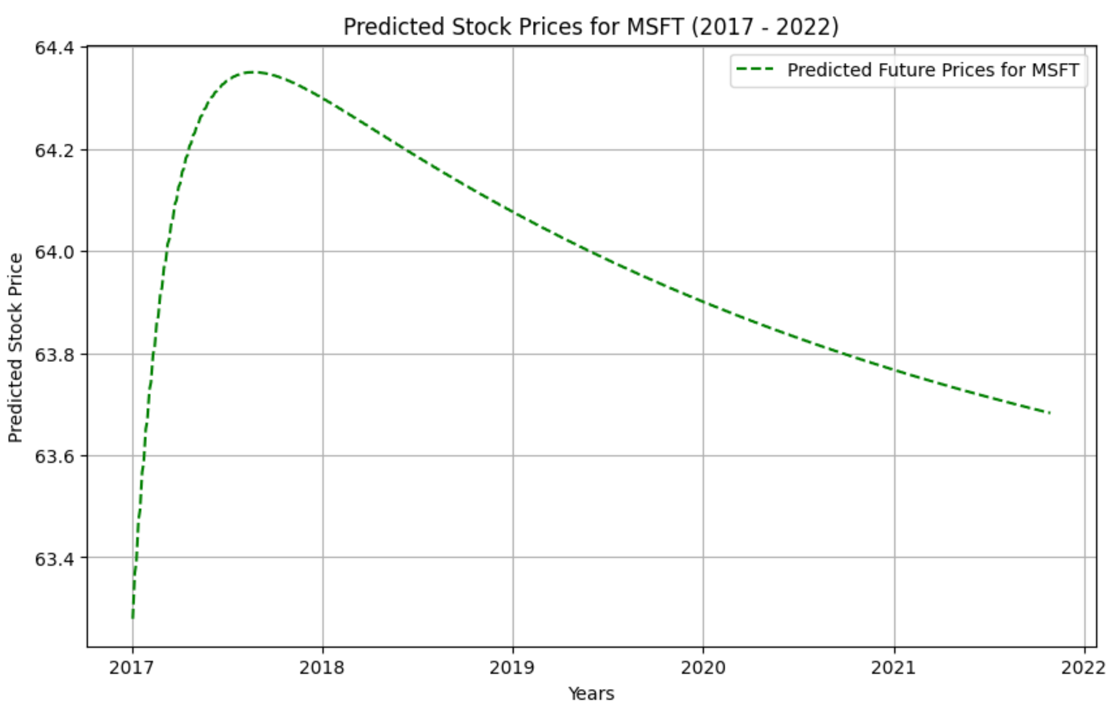
```
Processing GOOGL: 1762 rows
After dropping NaN: 1733 rows
```




```
Processing AAPL: 1762 rows
After dropping NaN: 1733 rows
```




```
Processing MSFT: 1762 rows
After dropping NaN: 1733 rows
```



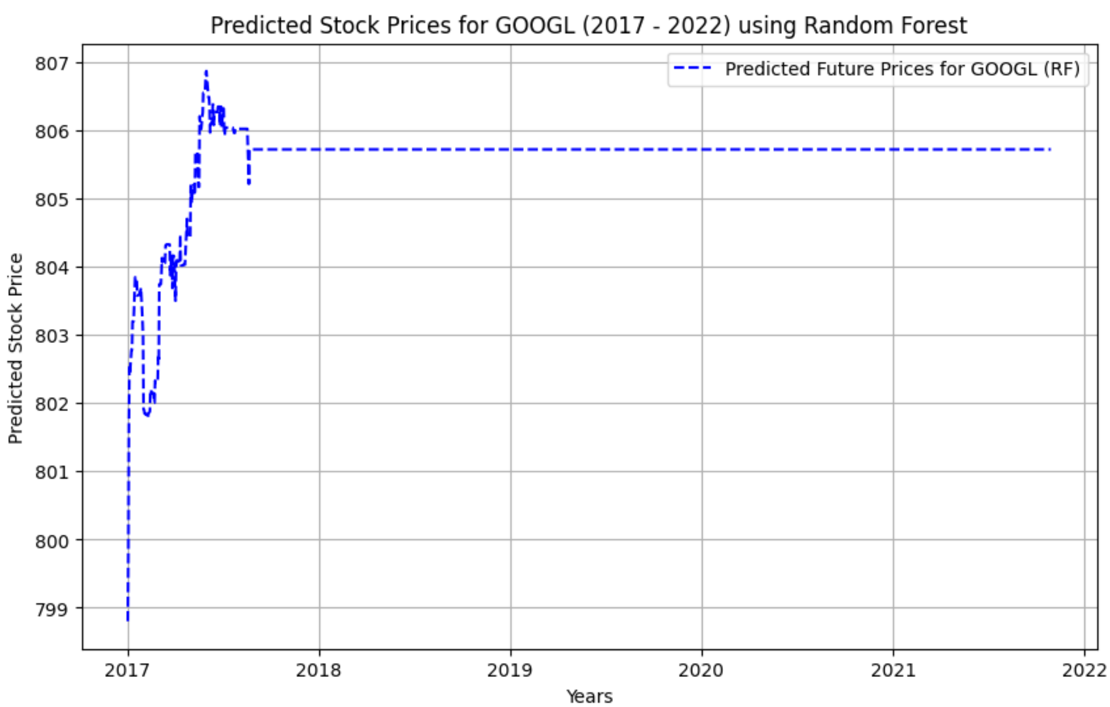
[ ]:
```
'''
Now I am going to try to learn a new method of prediction. I have learned about␣
↪Linear Regression Models in the
classroom but I am going to try to implement a method of prediction I have not␣
↪yet learned in the classroom:
Random Forest

From my readings and learning online, my understanding of Random Forest is as␣
↪follows:
    - Random Forest is a method that builds multiple decision trees and␣
↪averages their predictions to improve accuracy and reduce overfitting
        - Overfitting is when a model is overly trained on the training dataset␣
↪and becomes inaccurate when predicting actual information
    - It's well-suited for non-linear relationships and can handle more complex␣
↪patterns in the data
        - this makes it particularly useful when stock prices have volatility␣
↪and non-linear trends

Plan for model:
    1. Train a Random Forest Regressor model using historical stock data (same␣
↪data we have been using)
        2. Predict future stock prices for the next 5 years (2017-2022)
        3. Plot the results to compare predictions from Random Forest with␣
↪those from Linear Regression

'''
```

[12]:
```
from sklearn.ensemble import RandomForestRegressor

# Ensure plots show in Jupyter
%matplotlib inline

# Function to predict future prices for a specific stock using Random Forest␣
↪from 2017 to 2022
def predict_future_prices_with_rf(stock_symbol, start_year=2017,␣
↪future_years=5):
    # Filter for the specific stock and create a copy to avoid␣
↪SettingWithCopyWarning
    stock_data = filtered_stocks[filtered_stocks['symbol'] == stock_symbol].
↪copy()

    # Filter the data to only include data up to 2017
    stock_data = stock_data[stock_data['date'] < pd.
↪Timestamp(f'{start_year}-01-01')]
```

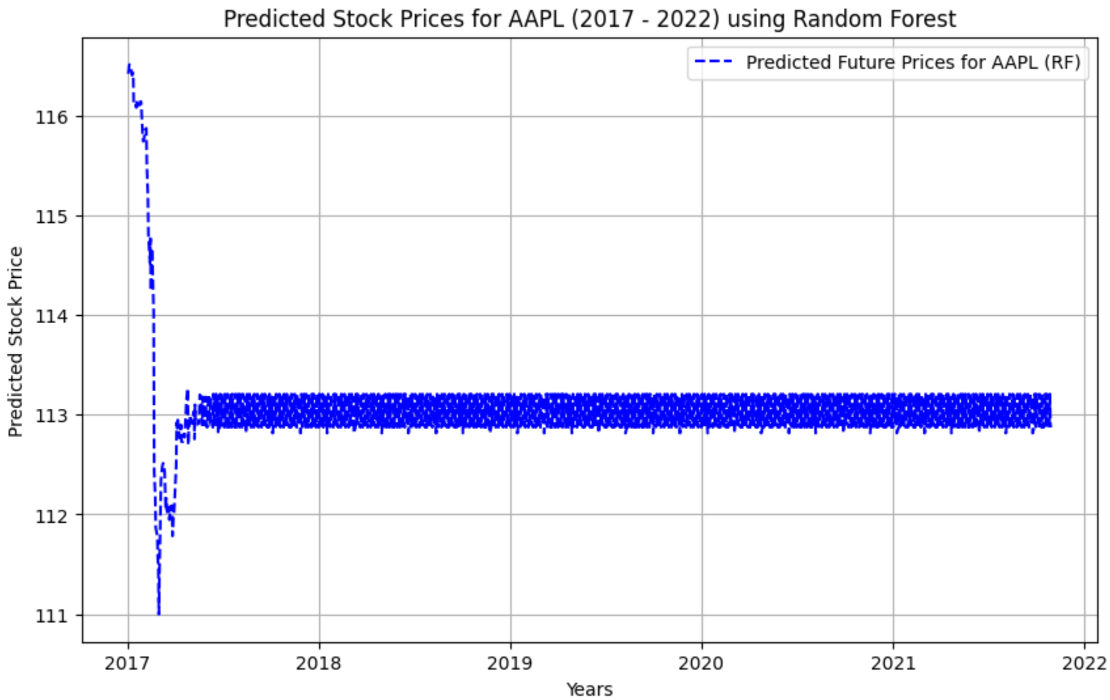
```
    # Debugging: Print shape of the data to make sure it's being loaded
    print(f"Processing {stock_symbol}: {stock_data.shape[0]} rows")

    # Create the features again (moving averages) for this stock
    stock_data.loc[:, '7_day_MA'] = stock_data['close'].rolling(window=7).mean()
    stock_data.loc[:, '30_day_MA'] = stock_data['close'].rolling(window=30).
↪mean()

    # Drop NaN rows
    stock_data = stock_data.dropna()

    # Debugging: Print shape after dropping NaN values
    print(f"After dropping NaN: {stock_data.shape[0]} rows")

    if stock_data.shape[0] == 0:
        print(f"No valid data for {stock_symbol}")
        return

    # Features (7-day MA, 30-day MA, volume) and target (Adjusted Close)
    X = stock_data[['7_day_MA', '30_day_MA', 'volume']]
    y = stock_data['close']

    # Train the Random Forest Regressor on the full dataset up to 2017
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X, y)

    # Predict future prices from 2017 to 2022
    last_row = X.iloc[-1].copy()  # Start with the last row in the dataset

    future_prices = []
    future_dates = pd.date_range(start=f'{start_year}-01-01',␣
↪periods=future_years*252, freq='B')  # Business days

    for date in future_dates:
        # Reshape last row and retain feature names using a DataFrame
        future_features = pd.DataFrame([last_row], columns=['7_day_MA',␣
↪'30_day_MA', 'volume'])
        next_price = rf_model.predict(future_features)

        # Update the moving averages (7-day and 30-day)
        last_row['7_day_MA'] = (last_row['7_day_MA'] * 6 + next_price[0]) / 7
        last_row['30_day_MA'] = (last_row['30_day_MA'] * 29 + next_price[0]) /␣
↪30

        # Append the predicted price
        future_prices.append(next_price[0])
```

```
    # Plot the future predicted prices for this stock with years on the x-axis
    plt.figure(figsize=(10, 6))
    plt.plot(future_dates, future_prices, label=f"Predicted Future Prices for␣
↪{stock_symbol} (RF)", color='blue', linestyle='--')
    plt.title(f"Predicted Stock Prices for {stock_symbol} (2017 - 2022) using␣
↪Random Forest")
    plt.xlabel("Years")
    plt.ylabel("Predicted Stock Price")
    plt.legend()
    plt.grid(True)
    plt.show()

# Predict future prices for GOOGL, AAPL, and MSFT using Random Forest from 2017␣
↪to 2022
predict_future_prices_with_rf('GOOGL')  # Google
predict_future_prices_with_rf('AAPL')   # Apple
predict_future_prices_with_rf('MSFT')   # Microsoft
```

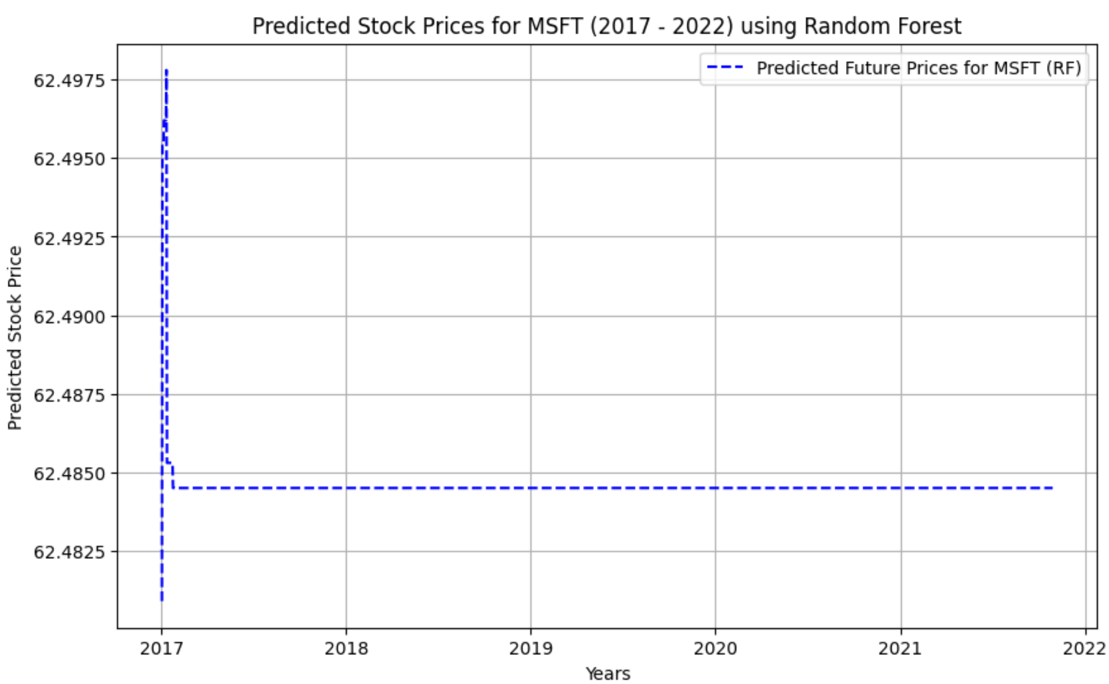
```
Processing GOOGL: 1762 rows
After dropping NaN: 1733 rows
```




```
Processing AAPL: 1762 rows
After dropping NaN: 1733 rows
```




```
Processing MSFT: 1762 rows
After dropping NaN: 1733 rows
```



```
[ ]: '''
Comparison of Predictions Using Linear Regression and Random Forest

1. Google (GOOGL)
    Linear Regression:
        The linear regression model predicted a steady upward trend for Google␣
 ↪stock, starting from about $810 and rising steadily to $870 by 2022.
        The trend appears smooth, indicating the model assumes a constant␣
 ↪growth pattern without accounting for fluctuations or volatility.
    Random Forest:
        The random forest model, however, shows significant initial fluctuations
            The price drops from about $806 to $800, then oscilates before␣
 ↪stabilizing at about $806.
        Once the initial fluctuations are over, the price remains mostly flat,␣
 ↪not showing much variation for the rest of the period.

2. Apple (AAPL):
    Linear Regression:
        The linear regression model forecasts that Apple's stock price will␣
 ↪grow steadily from $117 in early 2017 to around $122 by 2022.
        The increase is smooth and gradual, similar to the pattern in Google's␣
 ↪linear regression model
    Random Forest:
        The random forest model shows much greater fluctuations early on
            The stock price dips below $112 before oscillating around $113 and␣
 ↪$114.
        Similar to the GOOGL prediction, after the initial volatility, the␣
 ↪price stabilizes and becomes flat, hovering around $114.

3. Microsoft (MSFT):
    Linear Regression:
        The linear regression model predicted Microsoft's stock price to rise␣
 ↪slightly from $63.4 to around $64.4 before declining by 2022 to around $63.8.
        This model shows an increase, then a steady decline, indicating more␣
 ↪subtle stock price changes compared to Apple and Google.
    Random Forest:
        The random forest model predicts a very different behavior for␣
 ↪Microsoft's stock, showing a sharp drop from $62.5 to $62.48 right at the␣
 ↪beginning, followed by an almost completely flat line.
        After the sharp initial drop, the random forest model doesn't show much␣
 ↪movement, and the price remains flat around $62.48.

Overall Comparison:
        Linear Regression:
        Predicts steady, continuous growth for all three stocks.
```

```
      Does not account for stock price fluctuations or volatility, resulting␣
↪in smooth upward or slightly downward trends over time.
      It works well when stocks follow relatively linear growth patterns over␣
↪long periods time but may miss short-term volatility or sudden changes.

   Random Forest:
      Shows early volatility in all three stocks, particularly in the first␣
↪year (2017).
      Predictions stabilize quickly after initial fluctuations, leading to␣
↪flat trends for most of the future period (2018-2022)
          This is likely due to overfitting the short-term patterns and␣
↪struggling to predict long-term trends
      The model may be better at capturing short-term noise or fluctuations␣
↪but does not generalize well over longer-term trends.

Conclusion:

      •        Linear Regression is better suited for modeling long-term␣
↪trends, showing smooth and predictable growth for all stocks.
      •        Random Forest, while capturing some short-term volatility,␣
↪struggles with making meaningful long-term predictions.
```