

CSCI 6515 - Machine Learning for Big Data (Fall 2023)

Final Project

Group_ID: Group 10

Group Members:

1. Aamir_B00924776

Dataset Information

1. Apple Stock Prices

- **Dataset Name:** Apple Stock Prices. We will refer this as dataset_1 throughout the code. Anything with suffix _1 refers to Apple Stock.
- **Dataset Link:** *Not Applicable (Generated in Code)*
- **Dataset Description:** This dataset contains historical daily closing prices of Apple (AAPL) stock from 1981-01-01 to 2022-12-31. The data was obtained using the Yahoo Finance API.

2. Microsoft Stock Prices

- **Dataset Name:** Microsoft Stock Prices. We will refer this as dataset_2 throughout the code. Anything with suffix _2 refers to Microsoft Stock.
- **Dataset Link:** *Not Applicable (Generated in Code)*
- **Dataset Description:** This dataset includes historical daily closing prices of Microsoft (MSFT) stock from 1981-01-01 to 2022-12-31. The data was obtained using the Yahoo Finance API.

3. Intel Stock Prices

- **Dataset Name:** Intel Stock Prices. We will refer this as dataset_3 throughout the code. Anything with suffix _3 refers to Intel Corporation Stock.
- **Dataset Link:** *Not Applicable (Generated in Code)*
- **Dataset Description:** This dataset comprises historical daily closing prices of Intel (INTC) stock from 1981-01-01 to 2022-12-31. The data was obtained using the Yahoo Finance API.

2. Task Information

Task Goal: Predict Future Close Prices for the Next Day using Yahoo Finance API and Multiple Regression Algorithms .

Task Description:

This project aims to build predictive models for stock prices using historical data obtained from the Yahoo Finance API. The goal is to predict the closing prices of a selected stock for the

next day based on the available historical features. The project involves data collection, exploratory data analysis, feature engineering, and the implementation of multiple regression algorithms for predicting stock prices.

The key steps include:

1. **Data Collection:** Fetch historical stock prices using the Yahoo Finance API.
2. **Data Preprocessing:** Clean and preprocess the data, handling missing values and visualizing data for better understanding.
3. **Model Training:** Implement various regression algorithms, including:
 - Linear Regression
 - Random Forest Regression
 - Support Vector Machine (SVM) Regression
 - Gradient Boosting Regression
 - XGBoost Regression
 - K-Nearest Neighbors (KNN) Regression
4. **Model Evaluation:** Assess the performance of each model using appropriate metrics, such as mean squared error, R squared scores, mean expected variance or mean absolute error.
5. **Model Comparison:** Compare the performance of different regression algorithms to identify the most effective model for predicting stock prices. We use learning curves and residual plots in this section.
6. **Prediction:** Utilize the best-performing model to predict close prices for the next day.

Through this project, we aim to explore the effectiveness of various regression algorithms in predicting stock prices, providing valuable insights for investors and financial analysts.

Project Goal

Task Goal: Predict Future Close Prices for the Next Day using Yahoo Finance API and Multiple Regression Algorithms.

Project Description

Task Description: This project aims to build predictive models for stock prices using historical data obtained from the Yahoo Finance API. The goal is to predict the closing prices of a selected stock for the next day based on the available historical features. The project involves data collection, exploratory data analysis, feature engineering, and the implementation of multiple regression algorithms for predicting stock prices.

The key steps include:

1. **Data Collection:** Fetch historical stock prices using the Yahoo Finance API.
2. **Data Preprocessing:** Clean and preprocess the data, handling missing values and outliers.
3. **Feature Engineering:** Create relevant features that can aid in predicting future stock prices.
4. **Model Training:** Implement various regression algorithms, including:
 - Linear Regression
 - Random Forest Regression
 - Support Vector Machine (SVM) Regression
 - Gradient Boosting Regression
 - XGBoost Regression
 - K-Nearest Neighbors (KNN) Regression
5. **Model Evaluation:** Assess the performance of each model using appropriate metrics, such as mean squared error or mean absolute error.
6. **Model Comparison:** Compare the performance of different regression algorithms to identify the most effective model for predicting stock prices.
7. **Prediction:** Utilize the best-performing model to predict close prices for the next day.

Through this project, we aim to explore the effectiveness of various regression algorithms in predicting stock prices, providing valuable insights for investors and financial analysts.

3. Task Implementation: Coding

3.1 Preprocessing

In [1]:

```
1 # Importing necessary libraries
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 import yfinance as yf
```

In [2]:

```
1 # Define the stock symbol and date range
2 stock_symbol = "AAPL"
3 start_date = "1981-01-01"
4 end_date = "2022-12-31"
5
6 # Fetch historical data
7 dataset_1 = yf.download(stock_symbol, start=start_date, end=end_date)
8
9 # Display the data
10 print(dataset_1.head())
```

[*****100%*****] 1 of 1 completed

Volume	Open	High	Low	Close	Adj Close	V
Date						
1981-01-02	0.154018	0.155134	0.154018	0.154018	0.119183	216
60800						
1981-01-05	0.151228	0.151228	0.150670	0.150670	0.116592	357
28000						
1981-01-06	0.144531	0.144531	0.143973	0.143973	0.111410	451
58400						
1981-01-07	0.138393	0.138393	0.137835	0.137835	0.106660	556
86400						
1981-01-08	0.135603	0.135603	0.135045	0.135045	0.104501	398
27200						

In [3]:

```
1 # Define the stock symbol and date range
2 stock_symbol = "MSFT"
3 start_date = "1981-01-01"
4 end_date = "2022-12-31"
5
6 # Fetch historical data
7 dataset_2 = yf.download(stock_symbol, start=start_date, end=end_date)
8
9 # Display the data
10 print(dataset_2.head())
```

[*****100%*****] 1 of 1 completed

	Open	High	Low	Close	Adj Close	Volume
Date						
1986-03-13	0.088542	0.101563	0.088542	0.097222	0.060274	1031788800
1986-03-14	0.097222	0.102431	0.097222	0.100694	0.062427	308160000
1986-03-17	0.100694	0.103299	0.100694	0.102431	0.063503	133171200
1986-03-18	0.102431	0.103299	0.098958	0.099826	0.061888	67766400
1986-03-19	0.099826	0.100694	0.097222	0.098090	0.060812	47894400

In [4]:

```
1 # Define the stock symbol and date range
2 stock_symbol = "INTC"
3 start_date = "1981-01-01"
4 end_date = "2022-12-31"
5
6 # Fetch historical data
7 dataset_3 = yf.download(stock_symbol, start=start_date, end=end_date)
8
9 # Display the data
10 print(dataset_3.head())
```

[*****100%*****] 1 of 1 completed

	Open	High	Low	Close	Adj Close	V
Volume						
Date						
1981-01-02	0.419271	0.424479	0.419271	0.419271	0.236902	7795200
1981-01-05	0.434896	0.440104	0.434896	0.434896	0.245731	9187200
1981-01-06	0.427083	0.427083	0.424479	0.424479	0.239845	16099200
1981-01-07	0.414063	0.414063	0.411458	0.411458	0.232488	16857600
1981-01-08	0.406250	0.406250	0.403646	0.403646	0.228074	12556800

Data exploration

Data statistics

In [5]:

```
1 # Data dimension
2 print(dataset_1.shape)
```

(10590, 6)

```
In [6]: 1 #Check the data types and summary statistics of the variables
2 print(dataset_1.dtypes)
3 print(dataset_1.describe())
```

```
Open          float64
High          float64
Low           float64
Close          float64
Adj Close      float64
Volume         int64
dtype: object
      Open        High        Low       Close
Adj Close \
count  10590.000000  10590.000000  10590.000000  10590.000000  105
90.000000
mean    16.489507   16.676773   16.303221   16.496443
15.737323
std     35.147292   35.572207   34.730471   35.166443
34.646507
min    0.049665    0.049665    0.049107    0.049107
0.038000
25%    0.287946    0.296875    0.281295    0.287946
0.237027
50%    0.486607    0.494811    0.478125    0.486607
0.401790
75%    16.203304   16.326340   16.004911   16.141250
13.880048
max    182.630005  182.940002  179.119995  182.009995  1
79.953888

      Volume
count  1.059000e+04
mean    3.279425e+08
std     3.379436e+08
min    0.000000e+00
25%    1.215374e+08
50%    2.150960e+08
75%    4.069191e+08
max    7.421641e+09
```

```
In [7]: 1 # Check for missing values
2 print(dataset_1.isnull().sum())
3 print(dataset_2.isnull().sum())
4 print(dataset_3.isnull().sum())
```

```
Open      0
High      0
Low       0
Close     0
Adj Close 0
Volume    0
dtype: int64
Open      0
High      0
Low       0
Close     0
Adj Close 0
Volume    0
dtype: int64
Open      0
High      0
Low       0
Close     0
Adj Close 0
Volume    0
dtype: int64
```

No missing values

```
In [8]: 1 # Predicting stock price 1 day into the future
2 dataset_1['Target'] = dataset_1['Close'].shift(-1)
3 dataset_2['Target'] = dataset_2['Close'].shift(-1)
4 dataset_3['Target'] = dataset_3['Close'].shift(-1)
```

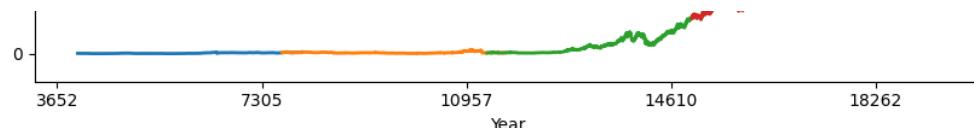
Data visualization

Visualize the target variable over time

```
In [9]:
```

```
1 from matplotlib.dates import YearLocator
2 from matplotlib.ticker import ScalarFormatter
3
4 # Generate 10-year intervals
5 year_interval = 10
6 years = range(dataset_1.index.year.min(), dataset_1.index.year.max(), year_interval)
7
8 # Create a plot with different colors for each 10-year interval
9 fig, ax = plt.subplots(figsize=(10, 6))
10
11 for i in range(len(years) - 1):
12     start_year = years[i]
13     end_year = years[i + 1]
14
15     # Ensure data exists for the specified time range
16     subset = dataset_1[(dataset_1.index.year >= start_year) & (dataset_1.index.year < end_year)]
17     if subset.empty:
18         print(f"Warning: No data for the time range {start_year} to {end_year}")
19     else:
20         ax.plot(subset.index, subset['Target'], label=f'{start_year}-{end_year}')
21
22 # Include the complete year 2022
23 subset_2022 = dataset_1[dataset_1.index.year == 2022]
24 if not subset_2022.empty:
25     ax.plot(subset_2022.index, subset_2022['Target'], label='2022')
26
27 # Set plot properties
28 ax.set_title('Stock Target Price Over Time (Apple)')
29 ax.set_xlabel('Year')
30 ax.set_ylabel('Close Price (Target)')
31 ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
32 ax.xaxis.set_major_locator(YearLocator(10))
33 ax.xaxis.set_major_formatter(ScalarFormatter())
34
35 # Show the plot
36 plt.tight_layout()
37 plt.show()
```

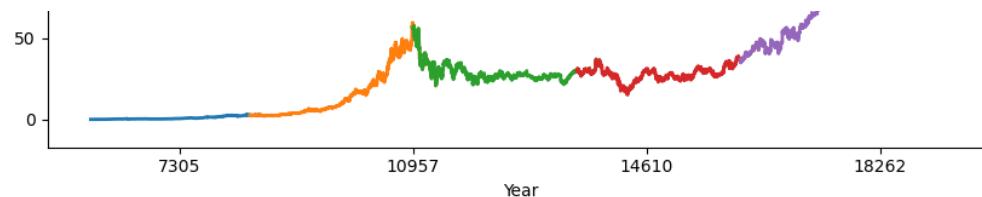




In [10]:

```
1 # Generate 7-year intervals
2 year_interval = 7
3 years = range(dataset_2.index.year.min(), dataset_2.index.year.max(), year_interval)
4
5 # Create a plot with different colors for each 10-year interval
6 fig, ax = plt.subplots(figsize=(10, 6))
7
8 for i in range(len(years) - 1):
9     start_year = years[i]
10    end_year = years[i + 1]
11
12    # Ensure data exists for the specified time range
13    subset = dataset_2[(dataset_2.index.year >= start_year) & (dataset_2.index.year < end_year)]
14    if subset.empty:
15        print(f"Warning: No data for the time range {start_year} to {end_year}")
16    else:
17        ax.plot(subset.index, subset['Target'], label=f'{start_year}-{end_year}')
18
19 # Include the complete year 2022
20 subset_2022 = dataset_2[dataset_2.index.year == 2022]
21 if not subset_2022.empty:
22     ax.plot(subset_2022.index, subset_2022['Target'], label='2022')
23
24 # Set plot properties
25 ax.set_title('Stock Close Price Over Time (Microsoft)')
26 ax.set_xlabel('Year')
27 ax.set_ylabel('Close Price (Target)')
28 ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
29 ax.xaxis.set_major_locator(YearLocator(10))
30 ax.xaxis.set_major_formatter(ScalarFormatter())
31
32 # Show the plot
33 plt.tight_layout()
34 plt.show()
```





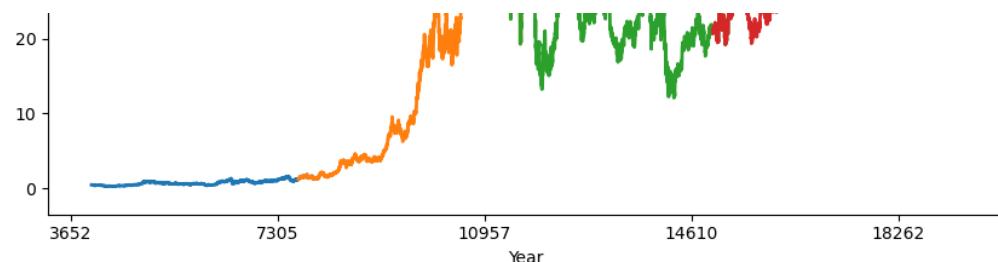
In [11]:

```

1 # Generate 10-year intervals
2 year_interval = 10
3 years = range(dataset_3.index.year.min(), dataset_3.index.year.max(), year_interval)
4
5 # Create a plot with different colors for each 10-year interval
6 fig, ax = plt.subplots(figsize=(10, 6))
7
8 for i in range(len(years) - 1):
9     start_year = years[i]
10    end_year = years[i + 1]
11
12    # Ensure data exists for the specified time range
13    subset = dataset_3[(dataset_3.index.year >= start_year) & (dataset_3.index.year < end_year)]
14    if subset.empty:
15        print(f"Warning: No data for the time range {start_year}-{end_year}")
16    else:
17        ax.plot(subset.index, subset['Target'], label=f'{start_year}-{end_year}')
18
19 # Include the complete year 2022
20 subset_2022 = dataset_3[dataset_3.index.year == 2022]
21 if not subset_2022.empty:
22     ax.plot(subset_2022.index, subset_2022['Target'], label='2022')
23
24 # Set plot properties
25 ax.set_title('Stock Close Price Over Time')
26 ax.set_xlabel('Year')
27 ax.set_ylabel('Close Price (Target)')
28 ax.legend(loc='upper left', bbox_to_anchor=(1, 1))
29 ax.xaxis.set_major_locator(YearLocator(10))
30 ax.xaxis.set_major_formatter(ScalarFormatter())
31
32 # Show the plot
33 plt.tight_layout()
34 plt.show()

```

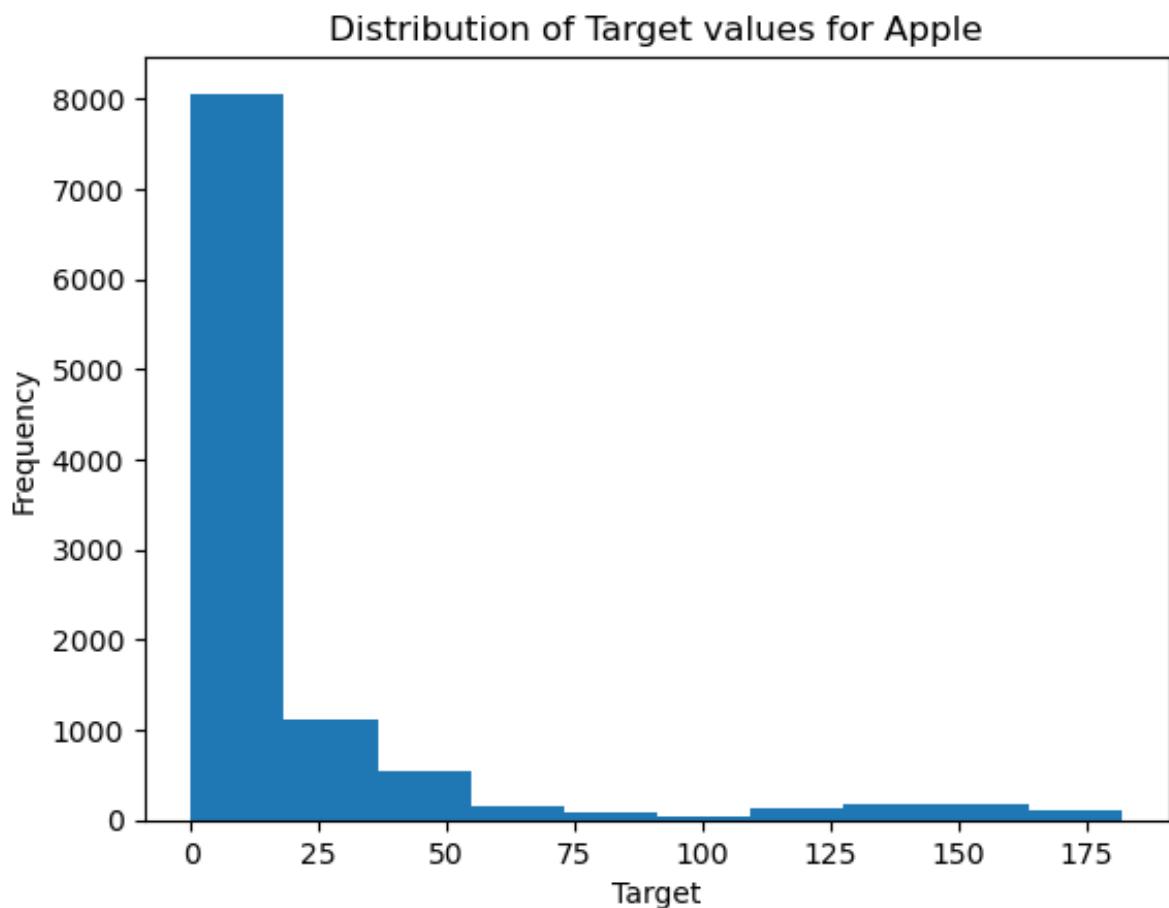




Distribution of 'Output'

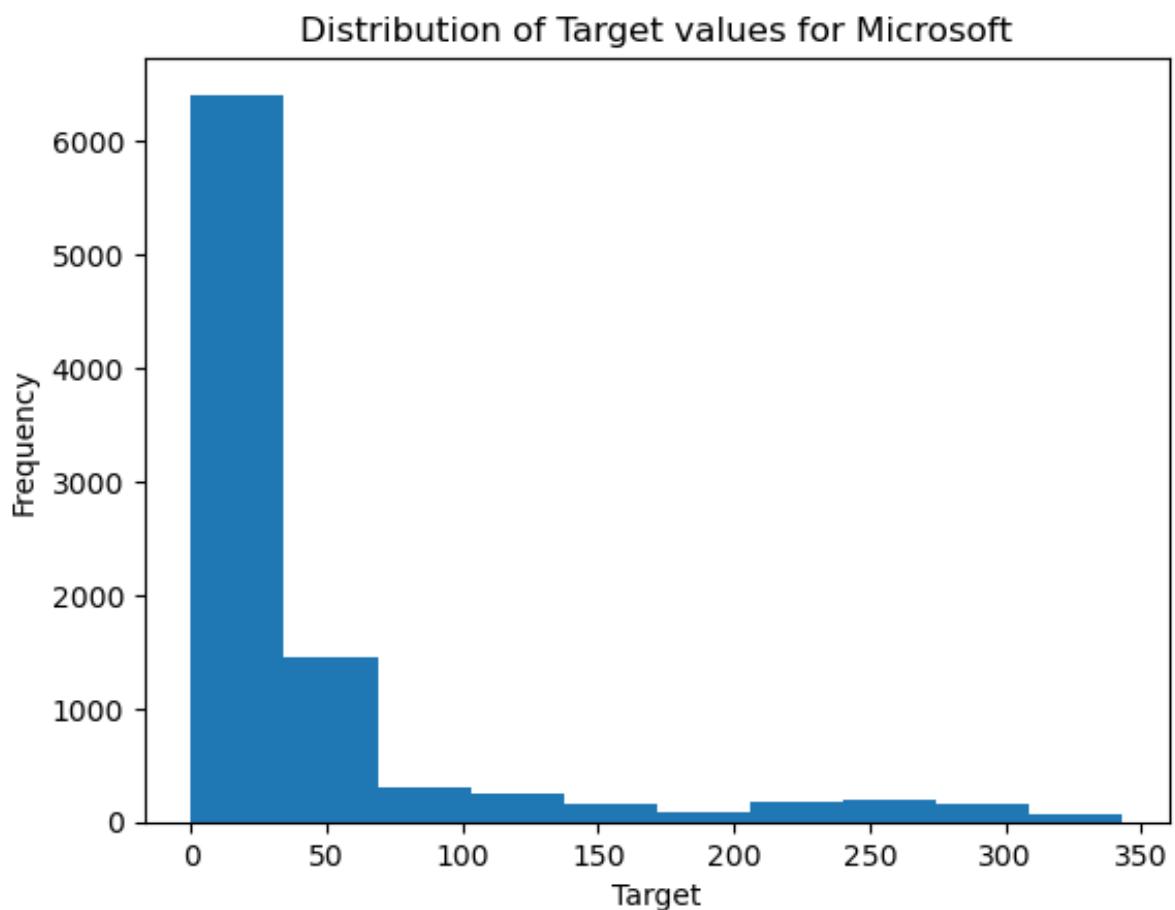
In [12]:

```
1 # Create a histogram of the output variable
2 plt.hist(dataset_1['Target'])
3
4 # Add labels and title
5 plt.xlabel('Target')
6 plt.ylabel('Frequency')
7 plt.title('Distribution of Target values for Apple')
8 plt.show()
```



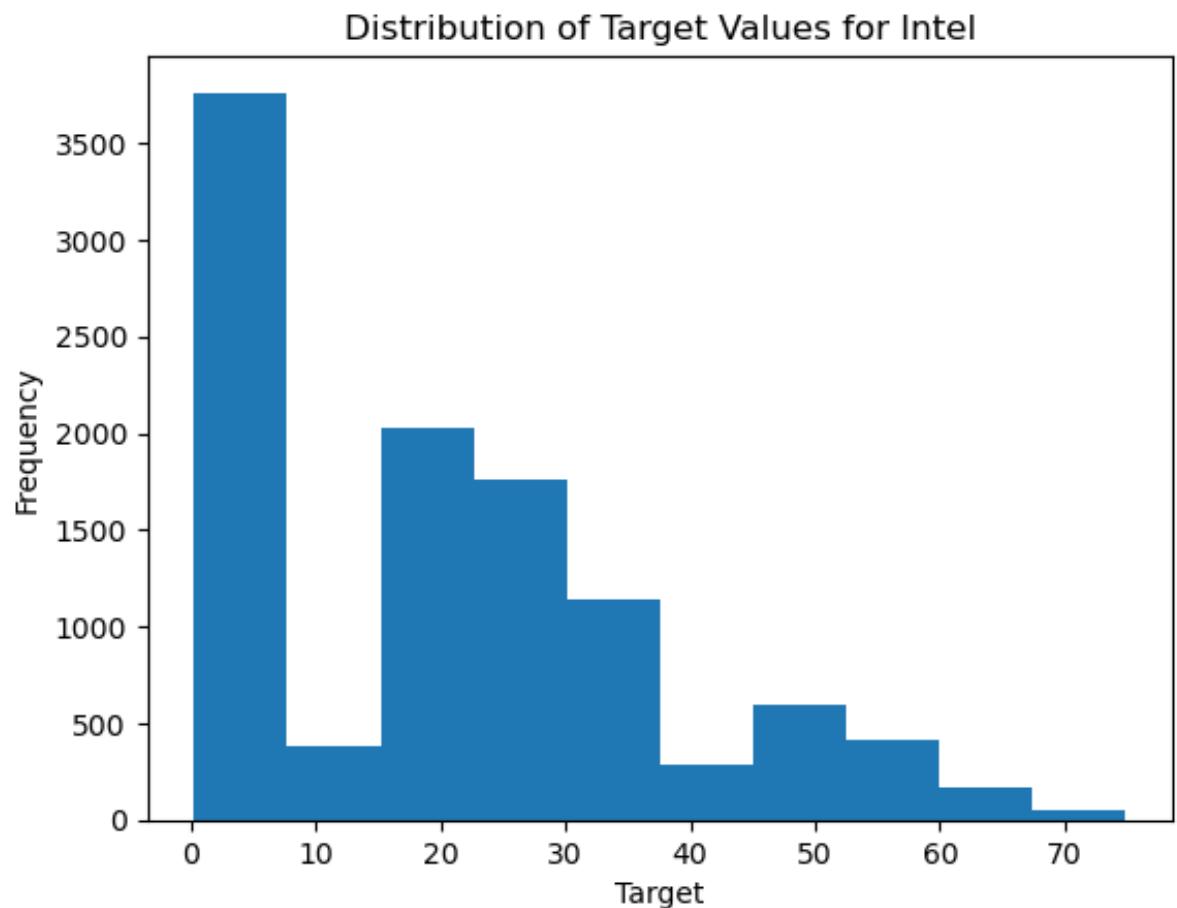
In [13]:

```
1 # Create a histogram of the output variable
2 plt.hist(dataset_2['Target'])
3
4 # Add labels and title
5 plt.xlabel('Target')
6 plt.ylabel('Frequency')
7 plt.title('Distribution of Target values for Microsoft')
8 plt.show()
```



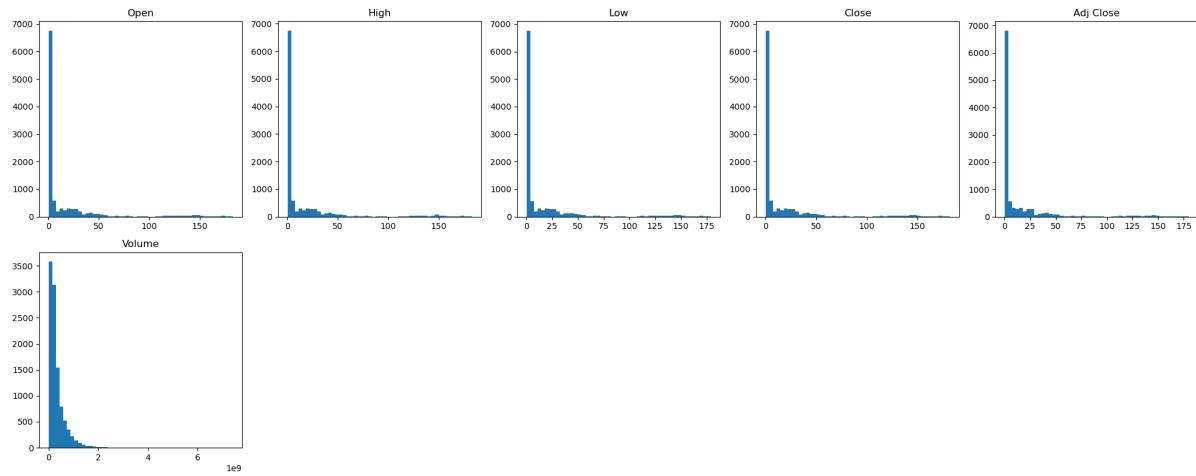
In [14]:

```
1 # Create a histogram of the output variable
2 plt.hist(dataset_3['Target'])
3
4 # Add labels and title
5 plt.xlabel('Target')
6 plt.ylabel('Frequency')
7 plt.title('Distribution of Target Values for Intel')
8 plt.show()
```



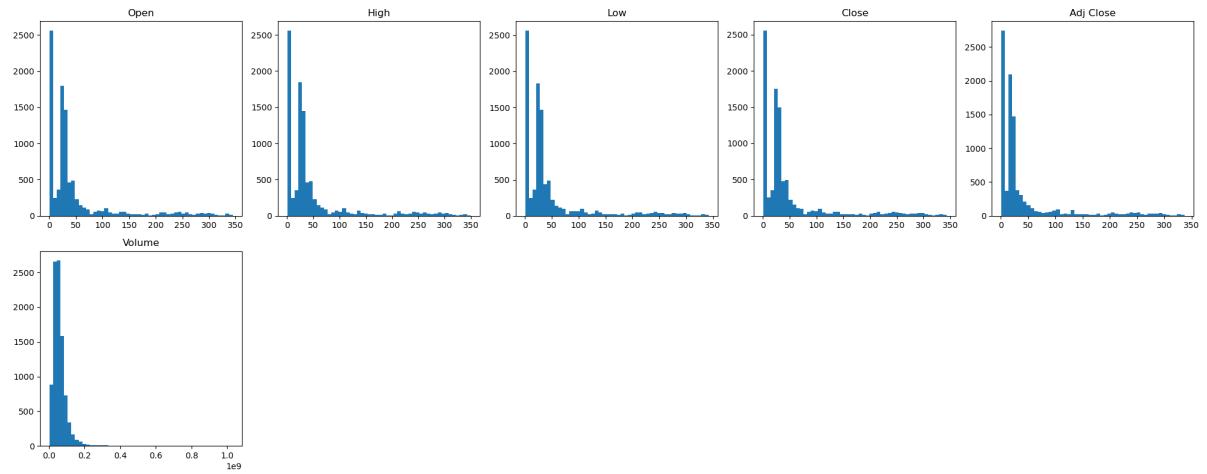
In [15]:

```
1 #Plot the distribution of all predictor variables for Apple
2
3 # create a list of predictor variable names
4 predictor_vars = dataset_1.columns[:-1].tolist()
5
6 # define the number of subplots per row
7 num_cols = 5
8
9 # calculate the number of rows needed
10 num_rows = (len(predictor_vars) - 1) // num_cols + 1
11
12 # create a new figure with the appropriate size
13 fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 4*num_
14
15 # flatten the axs array so we can index it with a single number
16 axs = axs.flatten()
17
18 # loop over each predictor variable and plot its distribution
19 for i, var in enumerate(predictor_vars):
20     axs[i].hist(dataset_1[var], bins=50)
21     axs[i].set_title(var)
22
23 # remove any unused subplots
24 for i in range(len(predictor_vars), num_rows*num_cols):
25     fig.delaxes(axs[i])
26
27 # adjust the spacing between subplots
28 fig.tight_layout()
29
30 # show the plot
31 plt.show()
```



In [16]:

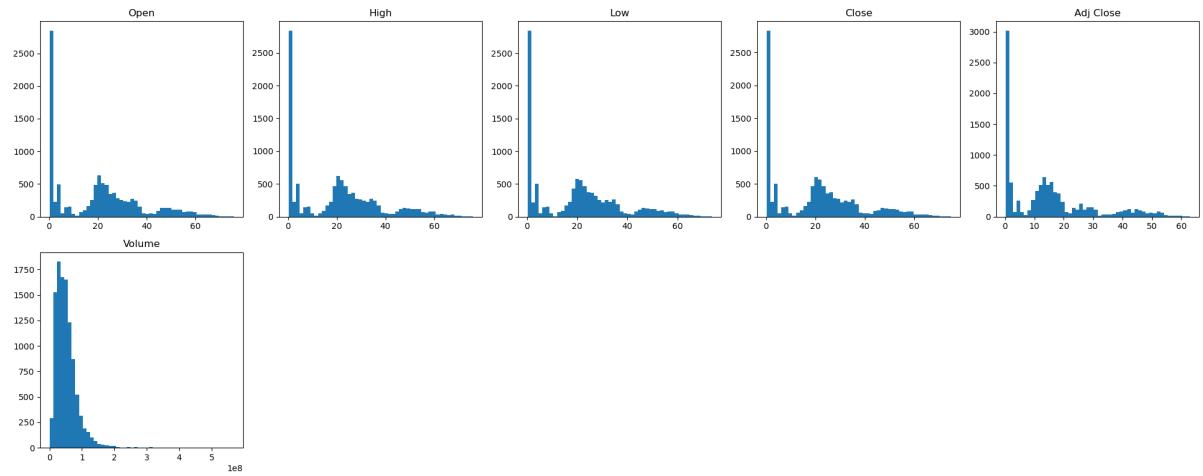
```
1 #Plot the distribution of all predictor variables for Microsoft
2
3 # create a list of predictor variable names
4 predictor_vars = dataset_2.columns[:-1].tolist()
5
6 # define the number of subplots per row
7 num_cols = 5
8
9 # calculate the number of rows needed
10 num_rows = (len(predictor_vars) - 1) // num_cols + 1
11
12 # create a new figure with the appropriate size
13 fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 4*num_
14
15 # flatten the axs array so we can index it with a single number
16 axs = axs.flatten()
17
18 # loop over each predictor variable and plot its distribution
19 for i, var in enumerate(predictor_vars):
20     axs[i].hist(dataset_2[var], bins=50)
21     axs[i].set_title(var)
22
23 # remove any unused subplots
24 for i in range(len(predictor_vars), num_rows*num_cols):
25     fig.delaxes(axs[i])
26
27 # adjust the spacing between subplots
28 fig.tight_layout()
29
30 # show the plot
31 plt.show()
```



```

In [17]: 1 #Plot the distribution of all predictor variables for Intel
2
3 # create a list of predictor variable names
4 predictor_vars = dataset_3.columns[:-1].tolist()
5
6 # define the number of subplots per row
7 num_cols = 5
8
9 # calculate the number of rows needed
10 num_rows = (len(predictor_vars) - 1) // num_cols + 1
11
12 # create a new figure with the appropriate size
13 fig, axs = plt.subplots(num_rows, num_cols, figsize=(20, 4*num_
14
15 # flatten the axs array so we can index it with a single number
16 axs = axs.flatten()
17
18 # loop over each predictor variable and plot its distribution
19 for i, var in enumerate(predictor_vars):
20     axs[i].hist(dataset_3[var], bins=50)
21     axs[i].set_title(var)
22
23 # remove any unused subplots
24 for i in range(len(predictor_vars), num_rows*num_cols):
25     fig.delaxes(axs[i])
26
27 # adjust the spacing between subplots
28 fig.tight_layout()
29
30 # show the plot
31 plt.show()

```

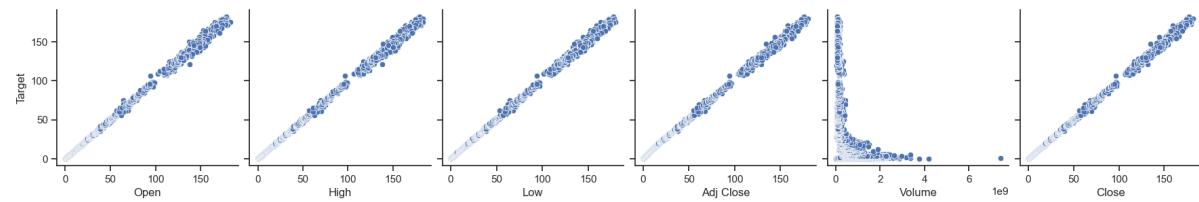


Relationship between predictor variable vs response variable

Scatterplots

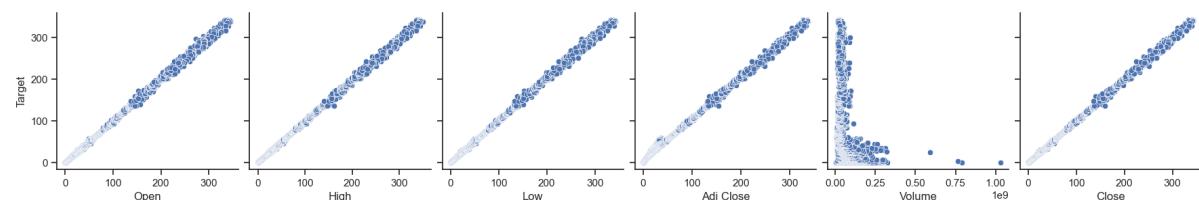
```
In [18]: 1 # Create scatterplots for all predictor variables and the output
2
3 # Use seaborn to create scatterplots for all predictor variables
4 sns.set(style="ticks", color_codes=True)
5
6 # Pairplot with scatterplots
7 sns.pairplot(dataset_1, x_vars=['Open', 'High', 'Low', 'Adj Close'],
8               y_vars='Target')
9 # Show the plots
10 plt.show()
```

/Users/aamir/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)



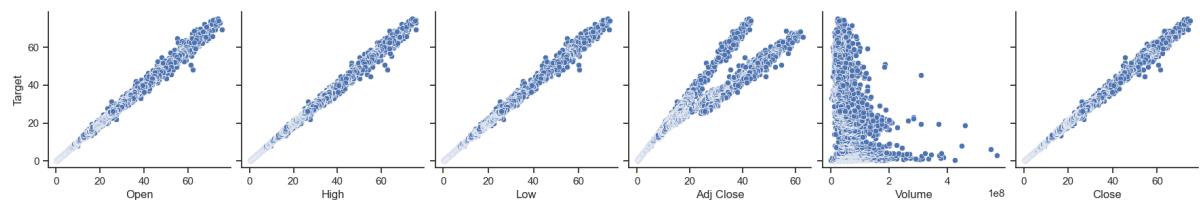
```
In [19]: 1 # Create scatterplots for all predictor variables and the output
2
3 # Use seaborn to create scatterplots for all predictor variables
4 sns.set(style="ticks", color_codes=True)
5
6 # Pairplot with scatterplots
7 sns.pairplot(dataset_2, x_vars=['Open', 'High', 'Low', 'Adj Close'],
8               y_vars='Target')
9 # Show the plots
10 plt.show()
```

/Users/aamir/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
self._figure.tight_layout(*args, **kwargs)



```
In [20]: 1 # Create scatterplots for all predictor variables and the output
2
3 # Use seaborn to create scatterplots for all predictor variables
4 sns.set(style="ticks", color_codes=True)
5
6 # Pairplot with scatterplots
7 sns.pairplot(dataset_3, x_vars=['Open', 'High', 'Low', 'Adj Close'],
8               y_vars='Target')
9
10 # Show the plots
11 plt.show()
```

/Users/aamir/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
 self._figure.tight_layout(*args, **kwargs)



Correlation coefficients

FOR APPLE

```
In [21]: 1 # Calculate the correlation coefficients between each predictor
2 correlations = dataset_1.corr()['Target'].sort_values(ascending=True)
3
4 # Print the correlation coefficients in descending order
5 print(correlations)
```

Target	1.000000
Close	0.999752
Low	0.999703
High	0.999684
Open	0.999593
Adj Close	0.999460
Volume	-0.212677
Name:	Target, dtype: float64

FOR MICROSOFT

In [22]:

```

1 # Calculate the correlation coefficients between each predictor
2 correlations = dataset_2.corr()['Target'].sort_values(ascending=True)
3
4 # Print the correlation coefficients in descending order
5 print(correlations)

```

```

Target      1.000000
Close       0.999740
Low         0.999698
High        0.999697
Open         0.999626
Adj Close    0.997898
Volume      -0.317650
Name: Target, dtype: float64

```

FOR INTEL

In [23]:

```

1 # Calculate the correlation coefficients between each predictor
2 correlations = dataset_3.corr()['Target'].sort_values(ascending=True)
3
4 # Print the correlation coefficients in descending order
5 print(correlations)

```

```

Target      1.000000
Close       0.999272
Low         0.999145
High        0.999143
Open         0.998966
Adj Close    0.967135
Volume      -0.190078
Name: Target, dtype: float64

```

FOR APPLE

In [24]:

```

1 # Calculate basic statistics
2 mean = dataset_1.mean()
3 median = dataset_1.median()
4 std_dev = dataset_1.std()
5 skewness = dataset_1.skew()
6 kurtosis = dataset_1.kurtosis()
7
8 # Print statistics
9 print('Mean:', mean)
10 print('Median:', median)
11 print('Standard Deviation:', std_dev)
12 print('Skewness:', skewness)
13 print('Kurtosis:', kurtosis)

```

```

Mean: Open      1.648951e+01
High        1.667677e+01
Low         1.630322e+01
Close       1.610611e+01

```

```
          1.079044e+01
Adj Close    1.573732e+01
Volume      3.279425e+08
Target       1.649799e+01
dtype: float64
Median: Open      4.866070e-01
High         4.948105e-01
Low          4.781250e-01
Close        4.866070e-01
Adj Close    4.017901e-01
Volume      2.150960e+08
Target       4.866070e-01
dtype: float64
Standard Deviation: Open      3.514729e+01
High         3.557221e+01
Low          3.473047e+01
Close        3.516644e+01
Adj Close    3.464651e+01
Volume      3.379436e+08
Target       3.516775e+01
dtype: float64
Skewness: Open      2.906946
High         2.909595
Low          2.903898
Close        2.906756
Adj Close    2.973560
Volume      3.550393
Target       2.906585
dtype: float64
Kurtosis: Open      8.027738
High         8.038691
Low          8.015121
Close        8.026841
Adj Close    8.369000
Volume      30.136917
Target       8.025735
dtype: float64
```

FOR MICROSOFT

In [25]:

```
1 # Calculate basic statistics
2 mean = dataset_2.mean()
3 median = dataset_2.median()
4 std_dev = dataset_2.std()
5 skewness = dataset_2.skew()
6 kurtosis = dataset_2.kurtosis()
7
8 # Print statistics
9 print('Mean:', mean)
10 print('Median:', median)
11 print('Standard Deviation:', std_dev)
12 print('Skewness:', skewness)
13 print('Kurtosis:', kurtosis)
```

```
Mean: Open      4.589458e+01
High       4.638958e+01
Low        4.538523e+01
Close      4.590371e+01
Adj Close   4.028181e+01
Volume     5.812576e+07
Target     4.590865e+01
dtype: float64
Median: Open      2.700000e+01
High       2.725000e+01
Low        2.675000e+01
Close      2.700500e+01
Adj Close   1.885390e+01
Volume     5.143360e+07
Target     2.700500e+01
dtype: float64
Standard Deviation: Open      6.690583e+01
High       6.759592e+01
Low        6.616577e+01
Close      6.691558e+01
Adj Close   6.647166e+01
Volume     3.831354e+07
Target     6.691749e+01
dtype: float64
Skewness: Open      2.527182
High       2.526299
Low        2.526516
Close      2.526559
Adj Close   2.613491
Volume     5.087524
Target     2.526427
dtype: float64
Kurtosis: Open      5.900486
High       5.888960
Low        5.899761
Close      5.895114
Adj Close   6.180203
Volume     80.668586
Target     5.894284
dtype: float64
```

FOR INTEL

In [26]:

```
1 # Calculate basic statistics
2 mean = dataset_3.mean()
3 median = dataset_3.median()
4 std_dev = dataset_3.std()
5 skewness = dataset_3.skew()
6 kurtosis = dataset_3.kurtosis()
7
8 # Print statistics
9 print('Mean:', mean)
10 print('Median:', median)
11 print('Standard Deviation:', std_dev)
12 print('Skewness:', skewness)
13 print('Kurtosis:', kurtosis)
```

```
Mean: Open      2.014376e+01
High       2.041796e+01
Low        1.987084e+01
Close      2.014220e+01
Adj Close   1.465558e+01
Volume     5.133172e+07
Target     2.014407e+01
dtype: float64
Median: Open      2.044000e+01
High       2.075000e+01
Low        2.018875e+01
Close      2.047363e+01
Adj Close   1.265825e+01
Volume     4.528320e+07
Target     2.047656e+01
dtype: float64
Standard Deviation: Open      1.751598e+01
High       1.775754e+01
Low        1.728002e+01
Close      1.751487e+01
Adj Close   1.467904e+01
Volume     3.484645e+07
Target     1.751465e+01
dtype: float64
Skewness: Open      0.627319
High       0.631932
Low        0.624933
Close      0.628053
Adj Close   1.087890
Volume     3.040134
Target     0.627952
dtype: float64
Kurtosis: Open      -0.355804
High       -0.339596
Low        -0.365128
```

```
Close      -0.353449
Adj Close   0.452695
Volume     22.401223
Target      -0.353484
dtype: float64
```

Split the 'data' into training, testing and validation set

Before training the models, we split 'data' into training (60%), testing (20%) and validation sets (20%).

- The trainng set will be used to train our models.
- The validation set will be used to compare models performance to choose the best models and hyperparamter tuning.
- The testing set will be used to calculate best models' performance metricst to see how well the models will perform on new data in the future.

In [27]:

```
from sklearn.model_selection import train_test_split
# Separate predictor variables (X) from the response variable (y)
dataset_1.drop(dataset_1.index[-1], inplace=True)
X1 = dataset_1.drop(['Target'], axis=1)
y1 = dataset_1['Target']

# Split the data into train (60%) and combined test & validation (40%)
X_train_1, X_test_val_1, y_train_1, y_test_val_1 = train_test_split(
    dataset_1, test_size=0.4, random_state=42)

# Split the combined test & validation set further into test (50%) and validation (50%)
X_test_1, X_val_1, y_test_1, y_val_1 = train_test_split(X_test_val_1, test_size=0.5)
```

In [28]:

```
# Separate predictor variables (X) from the response variable (y)
dataset_2.drop(dataset_2.index[-1], inplace=True)
X2 = dataset_2.drop(['Target'], axis=1)
y2 = dataset_2['Target']

# Split the data into train (60%) and combined test & validation (40%)
X_train_2, X_test_val_2, y_train_2, y_test_val_2 = train_test_split(
    dataset_2, test_size=0.4, random_state=42)

# Split the combined test & validation set further into test (50%) and validation (50%)
X_test_2, X_val_2, y_test_2, y_val_2 = train_test_split(X_test_val_2, test_size=0.5)
```

In [29]:

```
1 # Separate predictor variables (X) from the response variable (y)
2 dataset_3.drop(dataset_3.index[-1], inplace=True)
3 X3 = dataset_3.drop(['Target'], axis=1)
4 y3 = dataset_3['Target']
5
6 # Split the data into train (60%) and combined test & validation (40%)
7 X_train_3, X_test_val_3, y_train_3, y_test_val_3 = train_test_split(X3, y3, test_size=0.4, random_state=42)
8
9 # Split the combined test & validation set further into test (50%) and validation (50%)
10 X_test_3, X_val_3, y_test_3, y_val_3 = train_test_split(X_test_val_3, y_test_val_3, test_size=0.5, random_state=42)
```

3.2 Model development and training

This is a regression problem so we will train various types of suitable models using the train set: Linear Regression, Random Forest, Gradient Boosting, SVR, XG Boost and KNN

Then we will examine our models' R squared and calculate RMSE based on validation set.

Train Linear Regression, Random Forest, Gradient Boosting, XG Boost, Ada Boost

In [30]:

```
1 from sklearn.linear_model import LinearRegression, Ridge, Lasso
2 from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
3 import xgboost as xgb
4 from sklearn.ensemble import AdaBoostRegressor
5 from sklearn.metrics import r2_score, mean_squared_error
6
7 # Define the models
8 lr_model = LinearRegression()
9 rf_model = RandomForestRegressor(random_state=18)
10 gb_model = GradientBoostingRegressor(random_state=18)
11 xgb_model = xgb.XGBRegressor(random_state=18)
12
13 # Train the models on the training set
14 lr_model.fit(X_train_1, y_train_1)
15 rf_model.fit(X_train_1, y_train_1)
16 gb_model.fit(X_train_1, y_train_1)
17 xgb_model.fit(X_train_1, y_train_1)
```

Out [30]:

```
▼ XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_cat_to_onehot=None,
             max_cat_threshold=None, max_depth=None, max_leaves=None,
             max_bin=None,
```

Train SVR model

```
In [31]: 1 #SVR model  
2  
3 from sklearn.preprocessing import StandardScaler  
4 from sklearn.svm import SVR  
5 from sklearn import metrics  
6  
7 # Normalize the data  
8 scaler = StandardScaler()  
9 X_train_norm_1 = scaler.fit_transform(X_train_1)  
10 X_val_norm_1 = scaler.transform(X_val_1)  
11  
12 # Train the SVR model  
13 svr_model = SVR(kernel='linear')  
14 svr_model.fit(X_train_norm_1, y_train_1)
```

Out [31]:

▼ SVR
SVR(kernel='linear')

Train KNN

```
In [32]: 1 from sklearn.neighbors import KNeighborsRegressor  
2 from sklearn.metrics import r2_score, mean_squared_error  
3  
4 # Create a KNN regressor object  
5 knn_model = KNeighborsRegressor()  
6  
7 # Train the KNN model on the training set  
8 knn_model.fit(X_train_1, y_train_1)
```

Out [32]:

▼ KNeighborsRegressor
KNeighborsRegressor()

3.3 Model evaluation

K Fold Cross-Validation

In [33]:

```
1 from sklearn.metrics import mean_squared_error
2 from sklearn.model_selection import cross_val_score
3 from sklearn.model_selection import KFold
4
5 # List of regression models
6 models = {
7     'Linear Regression': LinearRegression(),
8     'Random Forest': RandomForestRegressor(),
9     'Gradient Boosting Regressor': GradientBoostingRegressor(),
10    'Support Vector Machine': SVR(),
11    'K-Nearest Neighbors': KNeighborsRegressor(),
12    'XGBoost': xgb.XGBRegressor(objective='reg:squarederror')
13}
14
15 # Number of folds for cross-validation
16 k_folds = 5
17 results = {}
18 for model_name, model in models.items():
19     # R-squared
20     scores_r2 = cross_val_score(model, X_train_1, y_train_1, sc
21
22     # RMSE
23     scores_rmse = np.sqrt(-cross_val_score(model, X_train_1, y_
24
25     # MAE
26     scores_mae = -cross_val_score(model, X_train_1, y_train_1, s
27
28     # Explained Variance
29     scores_explained_variance = cross_val_score(model, X_train_1
30
31     results[model_name] = {
32         'R-squared': scores_r2,
33         'RMSE': scores_rmse,
34         'MAE': scores_mae,
35         'Explained Variance': scores_explained_variance
36     }
37
38 # Display results
39 for model_name, scores in results.items():
40     print(f'{model_name}: Mean R-squared - {np.mean(scores["R-sq
41
```

Linear Regression: Mean R-squared - 0.9996, Mean RMSE - 0.7464, Mean MAE - 0.2387, Mean Explained Variance - 0.9996
Random Forest: Mean R-squared - 0.9994, Mean RMSE - 0.8600, Mean MAE - 0.2739, Mean Explained Variance - 0.9994
Gradient Boosting Regressor: Mean R-squared - 0.9994, Mean RMSE - 0.8533, Mean MAE - 0.2974, Mean Explained Variance - 0.9994
Support Vector Machine: Mean R-squared - -0.2118, Mean RMSE - 39.4950, Mean MAE - 16.6236, Mean Explained Variance - -0.0112
K-Nearest Neighbors: Mean R-squared - 0.0243, Mean RMSE - 35.4218, Mean MAE - 20.5415, Mean Explained Variance - 0.0247
XGBoost: Mean R-squared - 0.9990, Mean RMSE - 1.1356, Mean MAE - 0.3525, Mean Explained Variance - 0.9990

The results show the average performance of each model using the different evaluation metrics across 5 folds of cross-validation.

- Mean R²: The mean coefficient of determination or R-squared metric which indicates the proportion of the variance in the target variable that is predictable from the independent variables.
- Mean RMSE: The mean root-mean-square-error or RMSE metric which measures the average difference between the predicted values and the actual values.
- Mean MAE: The mean absolute error or MAE metric which measures the absolute difference between the predicted values and the actual values.
- Mean Explained Variance: The mean explained variance score which measures the proportion of the variance in the target variable that is explained by the independent variables.

In general, the higher the R² and Explained Variance, and the lower the RMSE and MAE, the better the model. Based on these metrics:

KNN and SVR do not perform well. We will not examine these models further.

These following models seem to perform pretty well and needs to be examined further: Random Forest, Gradient Boosting, XGBoost and Linear Regression.

In [34]:

```

1 # After selecting the best model based on cross-validation, eva
2 best_model = LinearRegression() # Based on the best model base
3 best_model.fit(X_train_1, y_train_1)
4
5 # Evaluate on the validation set
6 y_val_pred_1 = best_model.predict(X_val_1)
7 rmse_val_1 = np.sqrt(mean_squared_error(y_val_1, y_val_pred_1))
8 print(f'Validation set RMSE for the best model: {rmse_val_1:.4f}
9
10 # Evaluate on the test set
11 y_test_pred_1 = best_model.predict(X_test_1)
12 rmse_test_1 = np.sqrt(mean_squared_error(y_test_1, y_test_pred_
13 print(f'Test set RMSE for the best model: {rmse_test_1:.4f}')')

```

Validation set RMSE for the best model: 0.8418
Test set RMSE for the best model: 0.7985

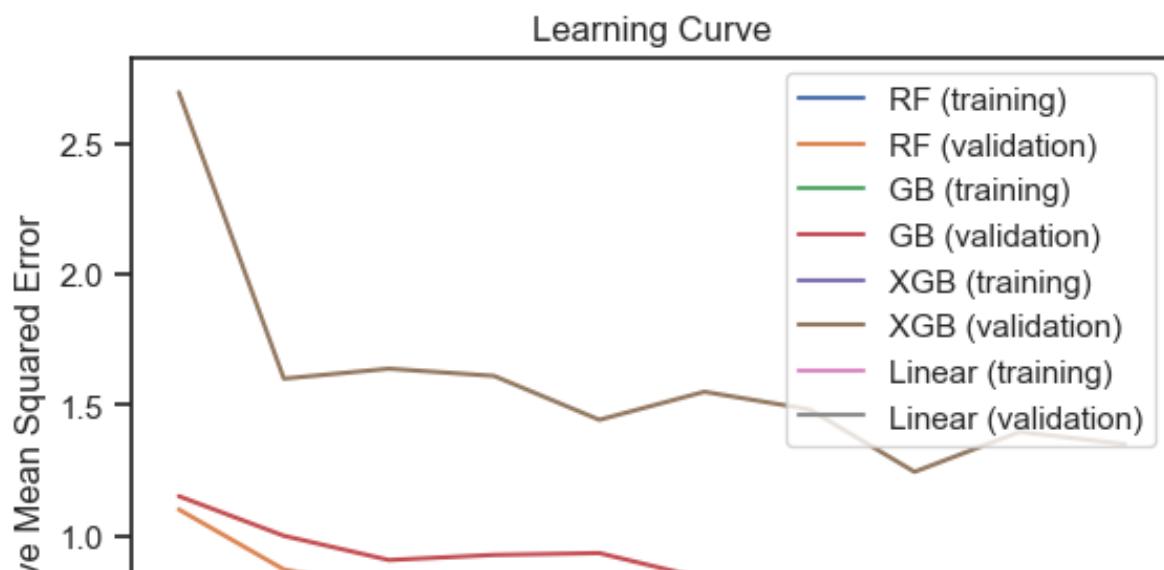
Learning curve

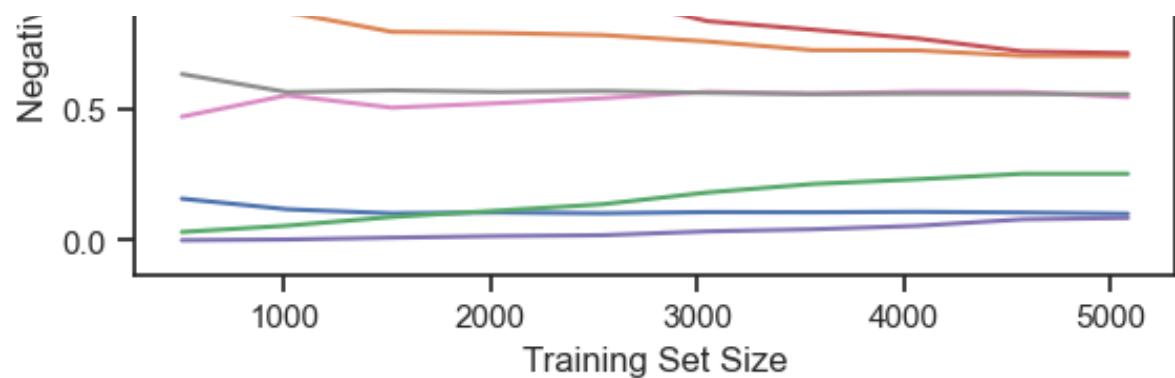
We will plot learning curves of Linear Regression, Random Forest, Gradient Boosting and XGBoost to compare their performance

Here XGBoost =>(Extreme Gradient Boosting)

In [35]:

```
1 from sklearn.model_selection import learning_curve
2
3 train_sizes, train_scores_rf, val_scores_rf = learning_curve(RandomForestClassifier(), X, y, cv=5, n_jobs=-1)
4 train_sizes, train_scores_gb, val_scores_gb = learning_curve(GradientBoostingClassifier(), X, y, cv=5, n_jobs=-1)
5 train_sizes, train_scores_xgb, val_scores_xgb = learning_curve(XGBClassifier(), X, y, cv=5, n_jobs=-1)
6 train_sizes, train_scores_lin, val_scores_lin = learning_curve(LogisticRegression(), X, y, cv=5, n_jobs=-1)
7
8
9 train_scores_rf = -train_scores_rf
10 val_scores_rf = -val_scores_rf
11 train_scores_gb = -train_scores_gb
12 val_scores_gb = -val_scores_gb
13 train_scores_xgb = -train_scores_xgb
14 val_scores_xgb = -val_scores_xgb
15 train_scores_lin = -train_scores_lin
16 val_scores_lin = -val_scores_lin
17
18
19 plt.plot(train_sizes, np.mean(train_scores_rf, axis=1), label='RF (training)')
20 plt.plot(train_sizes, np.mean(val_scores_rf, axis=1), label='RF (validation)')
21 plt.plot(train_sizes, np.mean(train_scores_gb, axis=1), label='GB (training)')
22 plt.plot(train_sizes, np.mean(val_scores_gb, axis=1), label='GB (validation)')
23 plt.plot(train_sizes, np.mean(train_scores_xgb, axis=1), label='XGB (training)')
24 plt.plot(train_sizes, np.mean(val_scores_xgb, axis=1), label='XGB (validation)')
25 plt.plot(train_sizes, np.mean(train_scores_lin, axis=1), label='Linear (training)')
26 plt.plot(train_sizes, np.mean(val_scores_lin, axis=1), label='Linear (validation)')
27
28
29 plt.title('Learning Curve')
30 plt.xlabel('Training Set Size')
31 plt.ylabel('Negative Mean Squared Error')
32 plt.legend()
33 plt.show()
```





Compare residual plots of trained models on training and validation set

We will compare the performance of regression models by analyzing the residuals, which are the differences between the predicted and actual target values.

- Scatter plot of residuals vs. predicted values: This plot helps to identify whether there is a systematic pattern in the residuals. If there is, it suggests that the model is making consistent errors in its predictions. Ideally, there should be no pattern and the residuals should be randomly scattered around the zero line.
- Scatter plot of absolute residuals vs. predicted values: This plot helps to identify whether the variance of the residuals is consistent across all levels of predicted values. Ideally, the variance of the residuals should be constant across all levels of predicted values.
- Scatter plot of predicted values vs. actual values: This plot helps to examine how well the model predicts the target variable. Ideally, the predicted values should be very close to the actual values, and the points should be aligned along the diagonal line.
- Normal probability plot of residuals: This plot helps to identify whether the residuals are normally distributed. Ideally, the residuals should follow a straight line, indicating a normal distribution.
- Scatter plot of standardized residuals vs. predicted values: This plot helps to identify whether there are any outliers or influential points in the data. Outliers are points that have a much higher or lower value than the rest of the data, while influential points are points that have a large effect on the regression line. Ideally, there should be no outliers or influential points.
- Histogram of residuals: This plot helps to examine the distribution of the residuals. Ideally, the residuals should be normally distributed around the zero line.
- Scale-location plot: This plot helps to identify whether there is a pattern in the variance of the residuals. Ideally, the variance of the residuals should be constant across all levels of predicted values.

Examine the residuals on the training set is to ensure that the model is not overfitting the data.

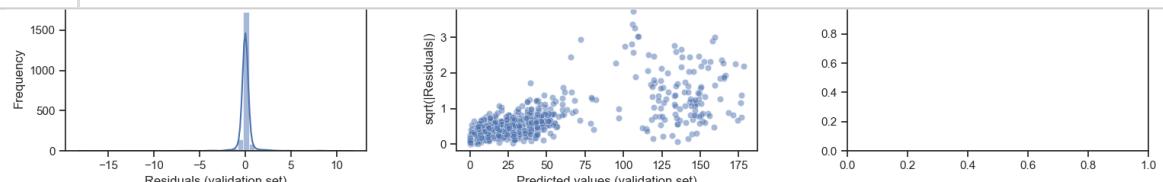
Examine the residuals on the validation set can help us choose the model that is best suited to the data.

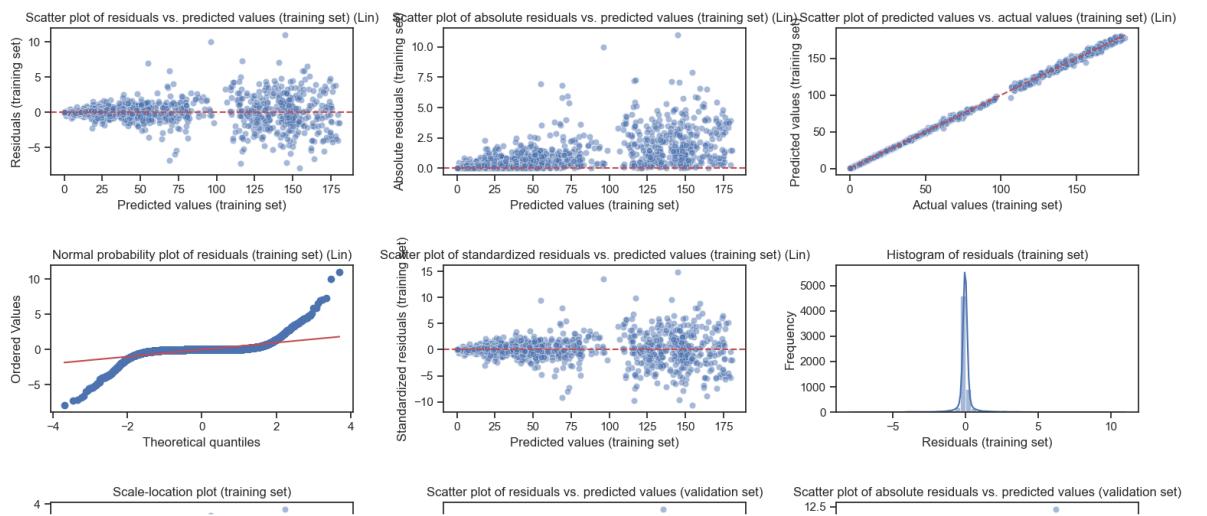
In [36]:

```
1 from scipy import stats
2
3 def plot_residuals(model, X_train, y_train, X_val, y_val, model_
# Use the trained models to make predictions on the trainin_
5 train_preds_1 = model.predict(X_train_1)
6 val_preds_1 = model.predict(X_val_1)
7
8 # Calculate the residuals on the training and validation se
9 train_residuals_1 = y_train_1 - train_preds_1
10 val_residuals_1 = y_val_1 - val_preds_1
11
12 # Calculate the standardized residuals on the training set
13 train_std_resid_1 = train_residuals_1 / np.sqrt(np.mean(squar
```

```
14
15 # Create a 4x4 grid of plots
16 fig, axs = plt.subplots(nrows=5, ncols=3, figsize=(16, 16))
17 fig.subplots_adjust(hspace=0.5)
18 axs = axs.flatten()
19
20 # Plot the residual analysis plots
21 sns.scatterplot(x=train_preds_1, y=train_residuals_1, ax=axs[0])
22 axs[0].axhline(y=0, color='r', linestyle='--')
23 axs[0].set_xlabel('Predicted values (training set)')
24 axs[0].set_ylabel('Residuals (training set)')
25 axs[0].set_title('Scatter plot of residuals vs. predicted v')
26
27 sns.scatterplot(x=train_preds_1, y=np.abs(train_residuals_1), ax=axs[1])
28 axs[1].axhline(y=0, color='r', linestyle='--')
29 axs[1].set_xlabel('Predicted values (training set)')
30 axs[1].set_ylabel('Absolute residuals (training set)')
31 axs[1].set_title('Scatter plot of absolute residuals vs. pr')
32
33 sns.scatterplot(x=y_train_1, y=train_preds_1, ax=axs[2], al)
34 axs[2].plot([y_train.min(), y_train.max()], [y_train.min(), y_train.max()])
35 axs[2].set_xlabel('Actual values (training set)')
36 axs[2].set_ylabel('Predicted values (training set)')
37 axs[2].set_title('Scatter plot of predicted values vs. actu')
38
39 stats.probplot(train_residuals_1, dist="norm", plot=axs[3])
40 axs[3].set_title('Normal probability plot of residuals (tra')
41
42 sns.scatterplot(x=train_preds_1, y=train_std_resid_1, ax=axs[4])
43 axs[4].axhline(y=0, color='r', linestyle='--')
44 axs[4].set_xlabel('Predicted values (training set)')
45 axs[4].set_ylabel('Standardized residuals (training set)')
46 axs[4].set_title('Scatter plot of standardized residuals vs')
47
48 sns.histplot(train_residuals_1, ax=axs[5], bins=50, kde=True)
49 axs[5].set_xlabel('Residuals (training set)')
50 axs[5].set_ylabel('Frequency')
51 axs[5].set_title('Histogram of residuals (training set)')
52
53 sns.scatterplot(x=train_preds_1, y=np.abs(train_std_resid_1), ax=axs[6])
54 axs[6].set_xlabel('Predicted values (training set)')
55 axs[6].set_ylabel('sqrt(|Residuals|)')
56 axs[6].set_title('Scale-location plot (training set)')
57
58 # Plot the residual analysis plots for the validation set
59 val_preds_1 = model.predict(X_val_1)
60 val_residuals_1 = y_val_1 - val_preds_1
61 val_std_resid_1 = val_residuals_1 / np.sqrt(mean_squared_error(val_preds_1, val_residuals_1))
62
63 sns.scatterplot(x=val_preds_1, y=val_residuals_1, ax=axs[7])
64 axs[7].axhline(y=0, color='r', linestyle='--')
65 axs[7].set_xlabel('Predicted values (validation set)')
66 axs[7].set_ylabel('Residuals')
67 axs[7].set_title('Scatter plot of residuals vs. predicted v')
```

```
68
69     sns.scatterplot(x=val_preds_1, y=np.abs(val_residuals_1), ax= axs[8].axhline(y=0, color='r', linestyle='--')
70     axs[8].set_xlabel('Predicted values (validation set)')
71     axs[8].set_ylabel('Absolute residuals')
72     axs[8].set_title('Scatter plot of absolute residuals vs. pr
73
74
75     stats.probplot(val_residuals_1, dist="norm", plot=axs[9])
76     axs[9].set_title('Normal probability plot of residuals (val
77
78
79     sns.scatterplot(x=val_preds_1, y=val_std_resid_1, ax= axs[10].axhline(y=0, color='r', linestyle='--')
80     axs[10].set_xlabel('Predicted values (validation set)')
81     axs[10].set_ylabel('Standardized residuals')
82     axs[10].set_title('Scatter plot of standardized residuals v
83
84
85     sns.scatterplot(x=y_val_1, y=val_preds_1, ax= axs[11], alpha=
86     axs[11].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.
87     axs[11].set_xlabel('Actual values (validation set)')
88     axs[11].set_ylabel('Predicted values (validation set)')
89     axs[11].set_title('Scatter plot of predicted values vs. act
90
91
92     sns.histplot(val_residuals_1, ax= axs[12], bins=50, kde=True)
93     axs[12].set_xlabel('Residuals (validation set)')
94     axs[12].set_ylabel('Frequency')
95     axs[12].set_title('Histogram of residuals (validation set)')
96
97
98     sns.scatterplot(x=val_preds_1, y=np.abs(val_std_resid_1) ** 0.5, ax= axs[13].set_xlabel('Predicted values (validation set)')
99     axs[13].set_ylabel('sqrt(|Residuals|)')
100    axs[13].set_title('Scale-location plot (validation set)')
101
102    # Adjust the spacing between subplots for a better display
103    fig.tight_layout(pad=3.0)
104
105    # Plot the residual analysis plots for the Random Forest model
106    plot_residuals(rf_model, X_train_1, y_train_1, X_val_1, y_val_1)
107
108    # Plot the residual analysis plots for the Gradient Boosting model
109    plot_residuals(gb_model, X_train_1, y_train_1, X_val_1, y_val_1)
110
111    # Plot the residual analysis plots for the XGBoost model on the validation set
112    plot_residuals(xgb_model, X_train_1, y_train_1, X_val_1, y_val_1)
113
114    # Plot the residual analysis plots for the Linear Regression model
115    plot_residuals(lr_model, X_train_1, y_train_1, X_val_1, y_val_1)
```





From the above plots, it seems that all 4 models fit the data well.

7. Compare RF, GB, XGBoost and Linear Regression model's performance on unseen data by using testing set

In [37]:

```
1 # Use the trained models to make predictions on the testing set
2 rf_preds = rf_model.predict(X_test_1)
3 gb_preds = gb_model.predict(X_test_1)
4 xgb_preds = xgb_model.predict(X_test_1)
5 lin_preds = lr_model.predict(X_test_1)
6
7 rf_r2 = r2_score(y_test_1, rf_preds)
8 rf_rmse = mean_squared_error(y_test_1, rf_preds, squared=False)
9 print('Random Forest R-squared score on testing set:', rf_r2)
10 print('Random Forest RMSE on testing set:', rf_rmse)
11
12 gb_r2 = r2_score(y_test_1, gb_preds)
13 gb_rmse = mean_squared_error(y_test_1, gb_preds, squared=False)
14 print('Gradient Boosting R-squared score on testing set:', gb_r2)
15 print('Gradient Boosting RMSE on testing set:', gb_rmse)
16
17 xgb_r2 = r2_score(y_test_1, xgb_preds)
18 xgb_rmse = mean_squared_error(y_test_1, xgb_preds, squared=False)
19 print('XGBoost R-squared score on testing set:', xgb_r2)
20 print('XGBoost RMSE on testing set:', xgb_rmse)
21
22 lin_r2 = r2_score(y_test_1, lin_preds)
23 lin_rmse = mean_squared_error(y_test_1, lin_preds, squared=False)
24 print('Linear Regression score on testing set:', lin_r2)
25 print('Linear Regression RMSE on testing set:', lin_rmse)
```

```
Random Forest R-squared score on testing set: 0.9993747470233457
Random Forest RMSE on testing set: 0.8439820207724127
Gradient Boosting R-squared score on testing set: 0.99935004841639
01
Gradient Boosting RMSE on testing set: 0.8604899732984866
XGBoost R-squared score on testing set: 0.9987854146733343
XGBoost RMSE on testing set: 1.1763033289964542
Linear Regression score on testing set: 0.9994403631102378
Linear Regression RMSE on testing set: 0.7984697695280717
```

It seems that Linear Regression > RF > Gradient Boosting > XGBoost but the performance difference is not that significant!

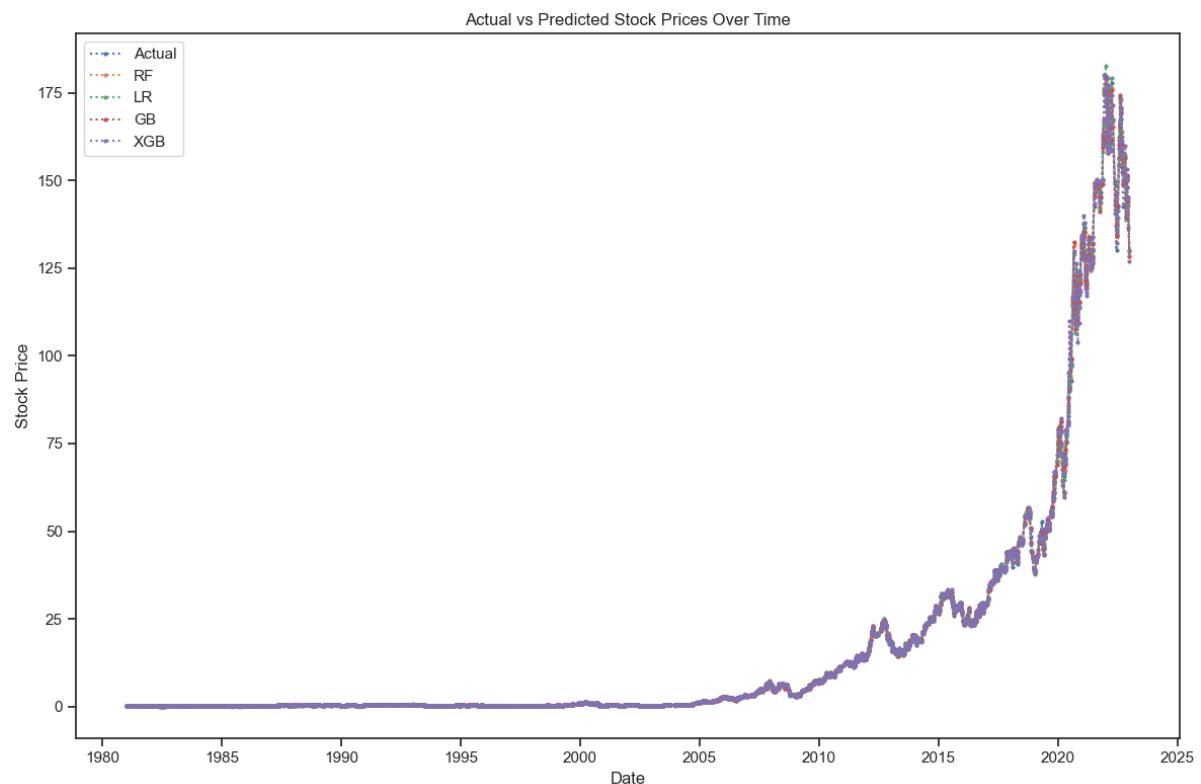
Plot predicted values on testing set vs actual values

In [38]:

```

1 # Create a DataFrame with actual and predicted values
2 result_df = pd.DataFrame({'Actual': y_test_1, 'Random Forest':
3
4     result_df['date_column'] = pd.to_datetime(result_df.index)
5     result_df = result_df.sort_values(by='date_column') # Sort by date
6
7 # Plotting
8 plt.figure(figsize=(12, 8))
9 plt.plot(result_df['date_column'], result_df['Actual'], label='Actual')
10 plt.plot(result_df['date_column'], result_df['Random Forest'], label='RF')
11 plt.plot(result_df['date_column'], result_df['Linear Regression'], label='LR')
12 plt.plot(result_df['date_column'], result_df['Gradient Boosting'], label='GB')
13 plt.plot(result_df['date_column'], result_df['XGBoost'], label='XGB')
14
15 # Set plot properties
16 plt.title('Actual vs Predicted Stock Prices Over Time')
17 plt.xlabel('Date')
18 plt.ylabel('Stock Price')
19 plt.legend()
20
21 # Show the plot
22 plt.tight_layout()
23 plt.show()
24

```



Compare residual plots for testing set

In [39]:

```

1 def plot_residuals(model, X_train, y_train, X_test, y_test, model_name):
2     # Use the trained models to make predictions on the testing set

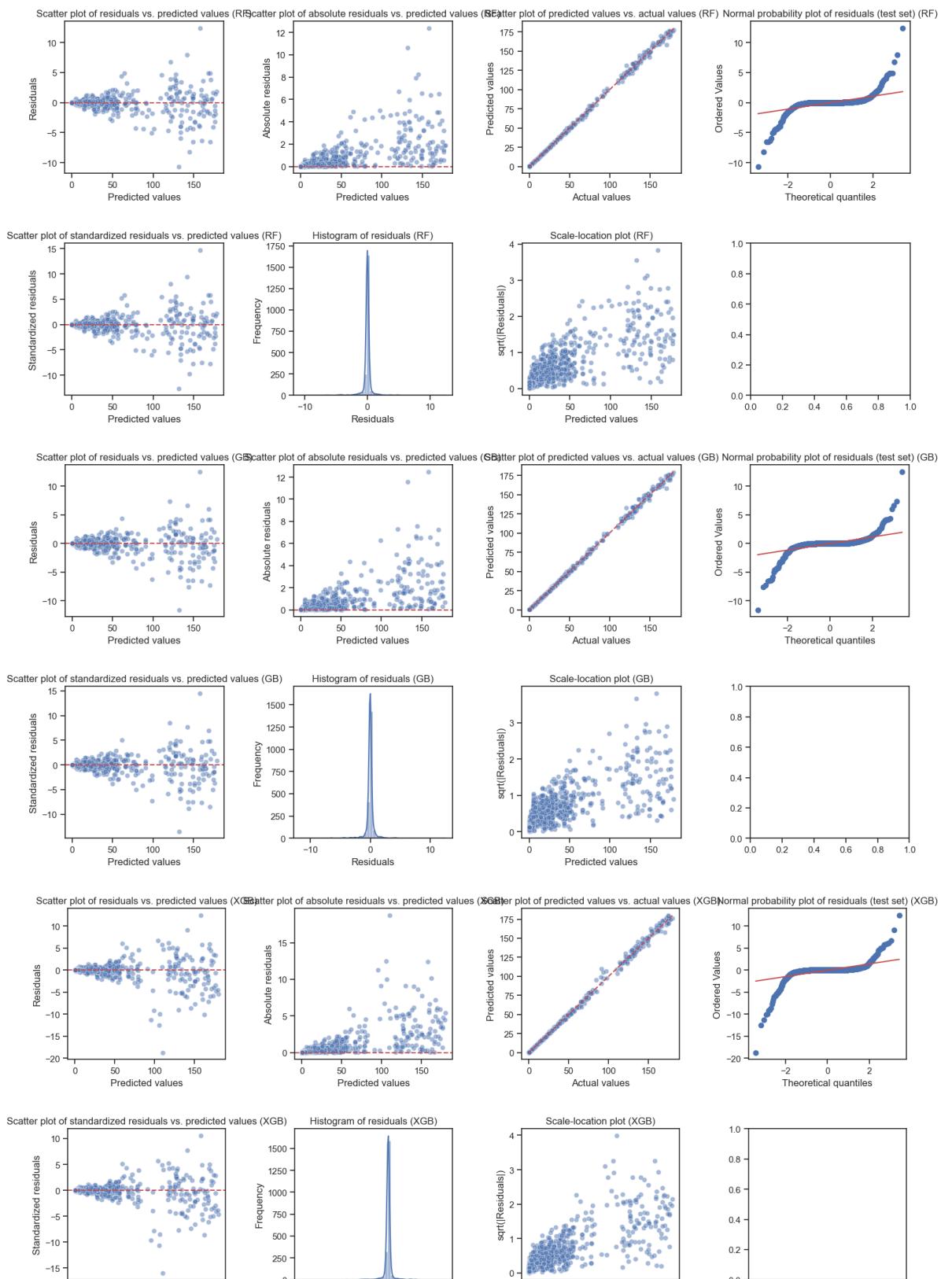
```

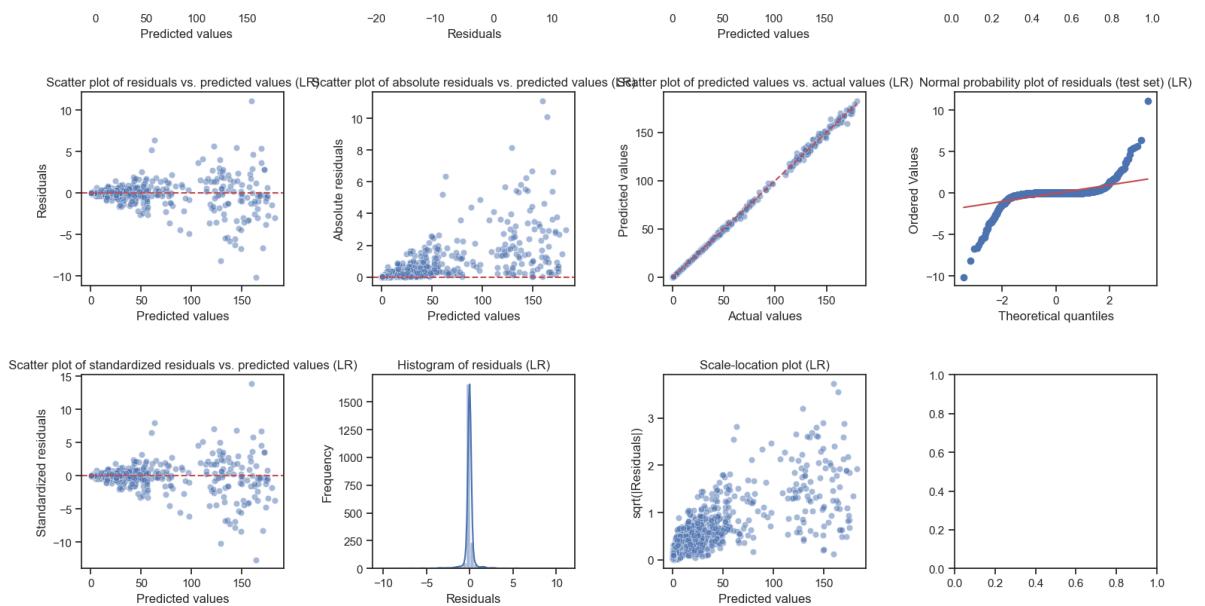
```
2      # Use the trained models to make predictions on the testing
3      preds = model.predict(X_test_1)
4
5      # Calculate the residuals
6      residuals = y_test_1 - preds
7      std_resid = residuals / np.sqrt(mean_squared_error(y_test_1))
8
9      # Create a 2x4 grid of plots
10     fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))
11     axs = axs.flatten()
12
13     # Plot the residual analysis plots
14     sns.scatterplot(x=preds, y=residuals, ax=axs[0], alpha=0.5)
15     axs[0].axhline(y=0, color='r', linestyle='--')
16     axs[0].set_xlabel('Predicted values')
17     axs[0].set_ylabel('Residuals')
18     axs[0].set_title('Scatter plot of residuals vs. predicted v')
19
20     sns.scatterplot(x=preds, y=np.abs(residuals), ax=axs[1], al
21     axs[1].axhline(y=0, color='r', linestyle='--')
22     axs[1].set_xlabel('Predicted values')
23     axs[1].set_ylabel('Absolute residuals')
24     axs[1].set_title('Scatter plot of absolute residuals vs. pr
25
26     sns.scatterplot(x=y_test_1, y=preds, ax=axs[2], alpha=0.5)
27     axs[2].plot([y_test_1.min(), y_test_1.max()], [y_test_1.m
28     axs[2].set_xlabel('Actual values')
29     axs[2].set_ylabel('Predicted values')
30     axs[2].set_title('Scatter plot of predicted values vs. actu
31
32     stats.probplot(residuals, dist="norm", plot=axs[3])
33     axs[3].set_title('Normal probability plot of residuals (tes
34
35     sns.scatterplot(x=preds, y=std_resid, ax=axs[4], alpha=0.5)
36     axs[4].axhline(y=0, color='r', linestyle='--')
37     axs[4].set_xlabel('Predicted values')
38     axs[4].set_ylabel('Standardized residuals')
39     axs[4].set_title('Scatter plot of standardized residuals vs
40
41     sns.histplot(residuals, ax=axs[5], bins=50, kde=True)
42     axs[5].set_xlabel('Residuals')
43     axs[5].set_ylabel('Frequency')
44     axs[5].set_title('Histogram of residuals (' + model_label +
45
46     sns.scatterplot(x=preds, y=np.abs(std_resid) ** 0.5, ax=axs
47     axs[6].set_xlabel('Predicted values')
48     axs[6].set_ylabel('sqrt(|Residuals|)')
49     axs[6].set_title('Scale-location plot (' + model_label + ')')
50
51     # Adjust the spacing between subplots for a better display
52     fig.tight_layout(pad=3.0)
53
54     # Plot the residual analysis plots for the Random Forest model
55     plot_residuals(rf_model, X_train_1, y_train_1, X_test_1, y_te
56
```

```

57 # Plot the residual analysis plots for the Gradient Boosting model
58 plot_residuals(gb_model, X_train_1, y_train_1, X_test_1, y_test_1)
59
60 # Plot the residual analysis plots for the XGBoost model on the test set
61 plot_residuals(xgb_model, X_train_1, y_train_1, X_test_1, y_test_1)
62
63 # Plot the residual analysis plots for the AdaBoost model on the test set
64 plot_residuals(lr_model, X_train_1, y_train_1, X_test_1, y_test_1)

```





3.4(a) Analysis

The performance of all the models is very similar and they perform well on unseen data as well. The fact that our actual and predicted prices are nearly identical on plots does not necessarily indicate overfitting. In the context of stock prices, there are a few factors to consider:

- Smoothness of Stock Prices:** Stock prices often exhibit smooth trends over time. If the model captures these trends accurately, the actual and predicted prices may indeed align closely.
- Model Performance Metrics:** It's important to assess the model's performance using appropriate metrics. Evaluate metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), or others to quantitatively measure the performance of our model. If the model performs well on both the training and testing sets, it's a positive sign. That's what we did throughout this whole project.

It's necessary to thoroughly evaluate the model's performance, understand the nature of the stock data, and be cautious about overfitting.

So the best performing model?: It seems to be Linear Regression but not by much. This could open possibilities for further experimentation in the future where one idea is to use more derived features such as moving averages, percentage changes in the stock price which could lead to more ambiguity in the dataset and probably the machine learning algorithms can capture deeply hidden patterns.

In [40]: 1 result_df

Out [40]:

	Actual	Random Forest	Linear Regression	Gradient Boosting	XGBoost	date_column
Date						
1981-01-02	0.150670	0.151328	0.158054	0.166234	0.157164	1981-01-02
1981-01-07	0.135045	0.137921	0.141551	0.150402	0.141186	1981-01-07
1981-01-14	0.139509	0.137662	0.140477	0.150402	0.141186	1981-01-14
1981-01-16	0.146763	0.134772	0.141891	0.150402	0.141186	1981-01-16
1981-01-20	0.145089	0.141099	0.145894	0.158011	0.141186	1981-01-20
...
2022-11-22	151.070007	149.656001	150.056880	150.218474	153.172256	2022-11-22
2022-11-23	148.110001	151.331300	151.630292	151.600470	150.562469	2022-11-23
2022-11-28	141.169998	145.396101	144.560607	145.501791	144.811020	2022-11-28
2022-12-14	136.500000	144.740301	142.771921	144.091055	145.103409	2022-12-14
2022-12-29	129.929993	128.534801	130.090189	128.390596	126.929932	2022-12-29

2118 rows × 6 columns

So we did train all the models for dataset_1 which is for APPLE TICKER. Now we repeat the same for 'MICROSOFT' and 'INTEL'

For Microsoft

In [41]:

```

1 # Define the models
2 lr_model = LinearRegression()
3 rf_model = RandomForestRegressor(random_state=18)
4 gb_model = GradientBoostingRegressor(random_state=18)
5 xgb_model = xgb.XGBRegressor(random_state=18)
6
7 # Train the models on the training set
8 lr_model.fit(X_train_2, y_train_2)
9 rf_model.fit(X_train_2, y_train_2)
10 gb_model.fit(X_train_2, y_train_2)
11 xgb_model.fit(X_train_2, y_train_2)
12
13 # List of regression models
14 models = {
15     'Linear Regression': lr_model,
16     'Random Forest': rf_model,
17     'Gradient Boosting Regressor': gb_model,
18     'Support Vector Machine': SVR(),
19     'K-Nearest Neighbors': KNeighborsRegressor(),
20     'XGBoost': xgb.XGBRegressor(objective='reg:squarederror')
21 }
22
23 # Number of folds for cross-validation
24 k_folds = 5
25 results = {}
26 for model_name, model in models.items():
27     # R-squared
28     scores_r2 = cross_val_score(model, X_train_2, y_train_2, sc
29
30     # RMSE
31     scores_rmse = np.sqrt(-cross_val_score(model, X_train_2, y_
32
33     # MAE
34     scores_mae = -cross_val_score(model, X_train_2, y_train_2,
35
36     # Explained Variance
37     scores_explained_variance = cross_val_score(model, X_train_
38
39     results[model_name] = {
40         'R-squared': scores_r2,
41         'RMSE': scores_rmse,
42         'MAE': scores_mae,
43         'Explained Variance': scores_explained_variance
44     }
45
46     # Display results
47     for model_name, scores in results.items():
48         print(f'{model_name}: Mean R-squared - {np.mean(scores["R-s
49

```

Linear Regression: Mean R-squared - 0.9995, Mean RMSE - 1.5105, Mean MAE - 0.6294, Mean Explained Variance - 0.9995
 Random Forest: Mean R-squared - 0.9994, Mean RMSE - 1.6566, Mean MAE - 0.6877, Mean Explained Variance - 0.9994

Gradient Boosting Regressor: Mean R-squared – 0.9994, Mean RMSE – 1.6949, Mean MAE – 0.7588, Mean Explained Variance – 0.9994
Support Vector Machine: Mean R-squared – 0.0864, Mean RMSE – 65.2435, Mean MAE – 33.2044, Mean Explained Variance – 0.1321
K-Nearest Neighbors: Mean R-squared – 0.1211, Mean RMSE – 63.9758, Mean MAE – 38.8960, Mean Explained Variance – 0.1219
XGBoost: Mean R-squared – 0.9991, Mean RMSE – 2.0805, Mean MAE – 0.8554, Mean Explained Variance – 0.9991

In [42]:

```
1 # After selecting the best model based on cross-validation, evaluate
2 best_model = LinearRegression() # Based on the best model based
3 best_model.fit(X_train_2, y_train_2)
4
5 # Evaluate on the validation set
6 y_val_pred_2 = best_model.predict(X_val_2)
7 rmse_val_2 = np.sqrt(mean_squared_error(y_val_2, y_val_pred_2))
8 print(f'Validation set RMSE for the best model: {rmse_val_2:.4f}')
9
10 # Evaluate on the test set
11 y_test_pred_2 = best_model.predict(X_test_2)
12 rmse_test_2 = np.sqrt(mean_squared_error(y_test_2, y_test_pred_2))
13 print(f'Test set RMSE for the best model: {rmse_test_2:.4f}')
```

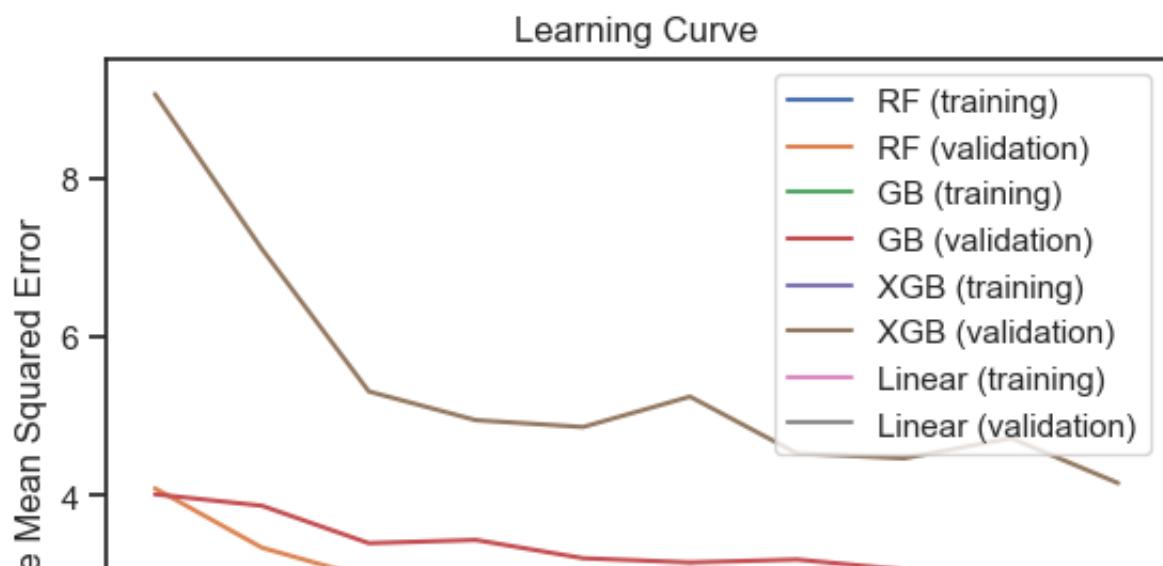
Validation set RMSE for the best model: 1.6670

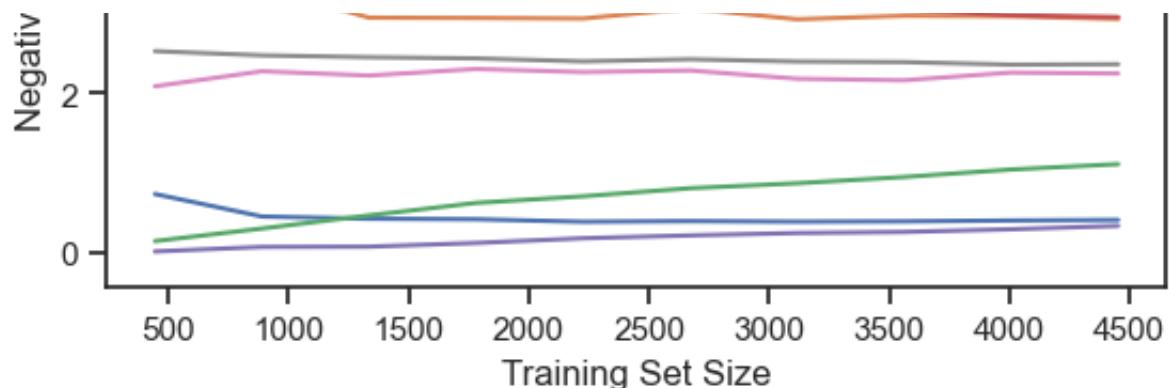
Test set RMSE for the best model: 1.3278

Plot Learning Curves

In [43]:

```
1 from sklearn.model_selection import learning_curve
2
3 train_sizes, train_scores_rf, val_scores_rf = learning_curve(RandomForestClassifier(), X, y, cv=5, n_jobs=-1)
4 train_sizes, train_scores_gb, val_scores_gb = learning_curve(GradientBoostingClassifier(), X, y, cv=5, n_jobs=-1)
5 train_sizes, train_scores_xgb, val_scores_xgb = learning_curve(XGBClassifier(), X, y, cv=5, n_jobs=-1)
6 train_sizes, train_scores_lin, val_scores_lin = learning_curve(LogisticRegression(), X, y, cv=5, n_jobs=-1)
7
8
9 train_scores_rf = -train_scores_rf
10 val_scores_rf = -val_scores_rf
11 train_scores_gb = -train_scores_gb
12 val_scores_gb = -val_scores_gb
13 train_scores_xgb = -train_scores_xgb
14 val_scores_xgb = -val_scores_xgb
15 train_scores_lin = -train_scores_lin
16 val_scores_lin = -val_scores_lin
17
18
19 plt.plot(train_sizes, np.mean(train_scores_rf, axis=1), label='RF (training)')
20 plt.plot(train_sizes, np.mean(val_scores_rf, axis=1), label='RF (validation)')
21 plt.plot(train_sizes, np.mean(train_scores_gb, axis=1), label='GB (training)')
22 plt.plot(train_sizes, np.mean(val_scores_gb, axis=1), label='GB (validation)')
23 plt.plot(train_sizes, np.mean(train_scores_xgb, axis=1), label='XGB (training)')
24 plt.plot(train_sizes, np.mean(val_scores_xgb, axis=1), label='XGB (validation)')
25 plt.plot(train_sizes, np.mean(train_scores_lin, axis=1), label='Linear (training)')
26 plt.plot(train_sizes, np.mean(val_scores_lin, axis=1), label='Linear (validation)')
27
28
29 plt.title('Learning Curve')
30 plt.xlabel('Training Set Size')
31 plt.ylabel('Negative Mean Squared Error')
32 plt.legend()
33 plt.show()
```





Compare residual plots for training and validation sets

In [44]:

```

1  from scipy import stats
2
3  def plot_residuals(model, X_train, y_train, X_val, y_val, model_
4      # Use the trained models to make predictions on the training set
5      train_preds_2 = model.predict(X_train_2)
6      val_preds_2 = model.predict(X_val_2)
7
8      # Calculate the residuals on the training and validation sets
9      train_residuals_2 = y_train_2 - train_preds_2
10     val_residuals_2 = y_val_2 - val_preds_2
11
12     # Calculate the standardized residuals on the training set
13     train_std_resid_2 = train_residuals_2 / np.sqrt(mean_squared_error(
14         train_residuals_2, np.zeros(len(train_residuals_2))))
15
16     # Create a 4x4 grid of plots
17     fig, axs = plt.subplots(nrows=5, ncols=3, figsize=(16, 16))
18     fig.subplots_adjust(hspace=0.5)
19     axs = axs.flatten()
20
21     # Plot the residual analysis plots
22     sns.scatterplot(x=train_preds_2, y=train_residuals_2, ax=axs[0])
23     axs[0].axhline(y=0, color='r', linestyle='--')
24     axs[0].set_xlabel('Predicted values (training set)')
25     axs[0].set_ylabel('Residuals (training set)')
26     axs[0].set_title('Scatter plot of residuals vs. predicted values')
27
28     sns.scatterplot(x=train_preds_2, y=np.abs(train_residuals_2), ax=axs[1])
29     axs[1].axhline(y=0, color='r', linestyle='--')
30     axs[1].set_xlabel('Predicted values (training set)')
31     axs[1].set_ylabel('Absolute residuals (training set)')
32     axs[1].set_title('Scatter plot of absolute residuals vs. predicted values')
33
34     sns.scatterplot(x=y_train_2, y=train_preds_2, ax=axs[2], alpha=0.5)
35     axs[2].plot([y_train_2.min(), y_train_2.max()], [y_train_2.min(), y_train_2.max()])
36     axs[2].set_xlabel('Actual values (training set)')
37     axs[2].set_ylabel('Predicted values (training set)')
38     axs[2].set_title('Scatter plot of predicted values vs. actual values')

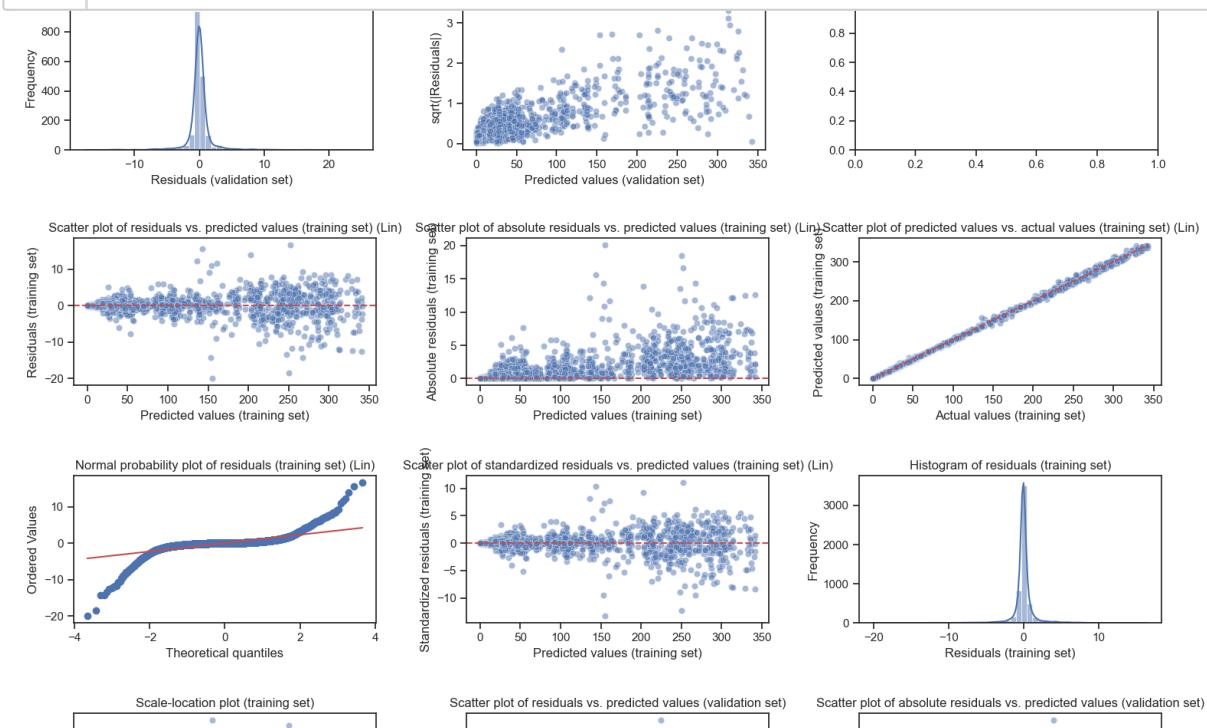
```

```
37
38
39 stats.probplot(train_residuals_2, dist="norm", plot= axs[3])
40 axs[3].set_title('Normal probability plot of residuals (train
41
42 sns.scatterplot(x=train_preds_2, y=train_std_resid_2, ax=axs[4])
43 axs[4].axhline(y=0, color='r', linestyle='--')
44 axs[4].set_xlabel('Predicted values (training set)')
45 axs[4].set_ylabel('Standardized residuals (training set)')
46 axs[4].set_title('Scatter plot of standardized residuals vs.
47
48 sns.histplot(train_residuals_2, ax=axs[5], bins=50, kde=True)
49 axs[5].set_xlabel('Residuals (training set)')
50 axs[5].set_ylabel('Frequency')
51 axs[5].set_title('Histogram of residuals (training set)')
52
53 sns.scatterplot(x=train_preds_2, y=np.abs(train_std_resid_2),
54 axs[6].set_xlabel('Predicted values (training set)')
55 axs[6].set_ylabel('sqrt(|Residuals|)')
56 axs[6].set_title('Scale-location plot (training set)')
57
58 # Plot the residual analysis plots for the validation set
59 val_preds_2 = model.predict(X_val_2)
60 val_residuals_2 = y_val_2 - val_preds_2
61 val_std_resid_2 = val_residuals_2 / np.sqrt(mean_squared_error(val_residuals_2))
62
63 sns.scatterplot(x=val_preds_2, y=val_residuals_2, ax=axs[7])
64 axs[7].axhline(y=0, color='r', linestyle='--')
65 axs[7].set_xlabel('Predicted values (validation set)')
66 axs[7].set_ylabel('Residuals')
67 axs[7].set_title('Scatter plot of residuals vs. predicted values (validation set)')
68
69 sns.scatterplot(x=val_preds_2, y=np.abs(val_residuals_2), ax=axs[8])
70 axs[8].axhline(y=0, color='r', linestyle='--')
71 axs[8].set_xlabel('Predicted values (validation set)')
72 axs[8].set_ylabel('Absolute residuals')
73 axs[8].set_title('Scatter plot of absolute residuals vs. predicted values (validation set)')
74
75 stats.probplot(val_residuals_2, dist="norm", plot=axs[9])
76 axs[9].set_title('Normal probability plot of residuals (validation set)')
77
78
79 sns.scatterplot(x=val_preds_2, y=val_std_resid_2, ax=axs[10])
80 axs[10].axhline(y=0, color='r', linestyle='--')
81 axs[10].set_xlabel('Predicted values (validation set)')
82 axs[10].set_ylabel('Standardized residuals')
83 axs[10].set_title('Scatter plot of standardized residuals vs. predicted values (validation set)')
84
85 sns.scatterplot(x=y_val_2, y=val_preds_2, ax=axs[11], alpha=0.5)
86 axs[11].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()])
87 axs[11].set_xlabel('Actual values (validation set)')
88 axs[11].set_ylabel('Predicted values (validation set)')
89 axs[11].set_title('Scatter plot of predicted values vs. actual values (validation set)')
90
91 sns.histplot(val_residuals_2, ax=axs[12], bins=50, kde=True)
```

```

92     axs[12].set_xlabel('Residuals (validation set)')
93     axs[12].set_ylabel('Frequency')
94     axs[12].set_title('Histogram of residuals (validation set)')
95
96     sns.scatterplot(x=val_preds_2, y=np.abs(val_std_resid_2) ** 0.5)
97     axs[13].set_xlabel('Predicted values (validation set)')
98     axs[13].set_ylabel('sqrt(|Residuals|)')
99     axs[13].set_title('Scale-location plot (validation set)')
100
101    # Adjust the spacing between subplots for a better display
102    fig.tight_layout(pad=3.0)
103
104    # Plot the residual analysis plots for the Random Forest model
105    plot_residuals(rf_model, X_train_2, y_train_2, X_val_2, y_val_2)
106
107    # Plot the residual analysis plots for the Gradient Boosting model
108    plot_residuals(gb_model, X_train_2, y_train_2, X_val_2, y_val_2)
109
110    # Plot the residual analysis plots for the XGBoost model on the training set
111    plot_residuals(xgb_model, X_train_2, y_train_2, X_val_2, y_val_2)
112
113    # Plot the residual analysis plots for the Linear Regression model on the validation set
114    plot_residuals(lr_model, X_train_2, y_train_2, X_val_2, y_val_2)
115

```



In [45]:

```
1 # Use the trained models to make predictions on the testing set
2 rf_preds = rf_model.predict(X_test_2)
3 gb_preds = gb_model.predict(X_test_2)
4 xgb_preds = xgb_model.predict(X_test_2)
5 lin_preds = lr_model.predict(X_test_2)
6
7 rf_r2 = r2_score(y_test_2, rf_preds)
8 rf_rmse = mean_squared_error(y_test_2, rf_preds, squared=False)
9 print('Random Forest R-squared score on testing set:', rf_r2)
10 print('Random Forest RMSE on testing set:', rf_rmse)
11
12 gb_r2 = r2_score(y_test_2, gb_preds)
13 gb_rmse = mean_squared_error(y_test_2, gb_preds, squared=False)
14 print('Gradient Boosting R-squared score on testing set:', gb_r2)
15 print('Gradient Boosting RMSE on testing set:', gb_rmse)
16
17 xgb_r2 = r2_score(y_test_2, xgb_preds)
18 xgb_rmse = mean_squared_error(y_test_2, xgb_preds, squared=False)
19 print('XGBoost R-squared score on testing set:', xgb_r2)
20 print('XGBoost RMSE on testing set:', xgb_rmse)
21
22 lin_r2 = r2_score(y_test_2, lin_preds)
23 lin_rmse = mean_squared_error(y_test_2, lin_preds, squared=False)
24 print('Linear Regression score on testing set:', lin_r2)
25 print('Linear Regression RMSE on testing set:', lin_rmse)
```

Random Forest R-squared score on testing set: 0.9994559630212702
Random Forest RMSE on testing set: 1.4430379533391477
Gradient Boosting R-squared score on testing set: 0.99947270108434
54
Gradient Boosting RMSE on testing set: 1.4206659863289954
XGBoost R-squared score on testing set: 0.9990538331967833
XGBoost RMSE on testing set: 1.9030377214060559
Linear Regression score on testing set: 0.9995393800255813
Linear Regression RMSE on testing set: 1.327806917886345

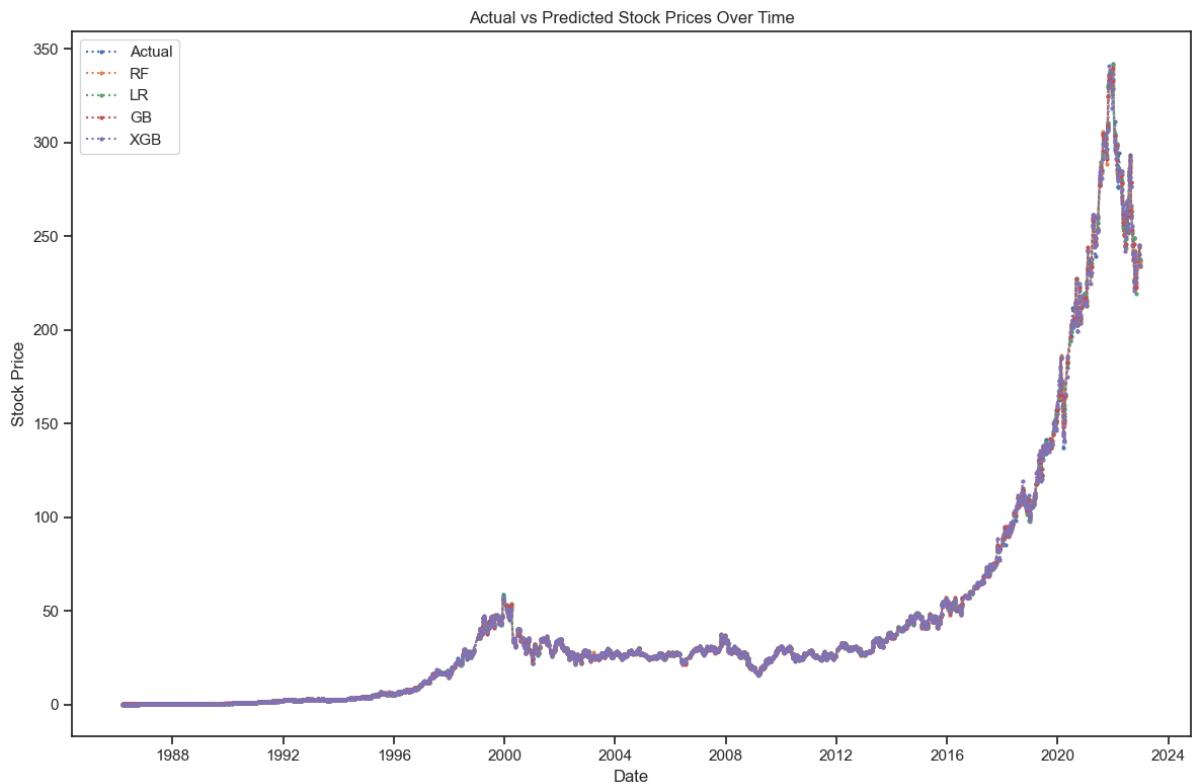
Visualize the predicted and actual target values

In [46]:

```

1 # Create a DataFrame with actual and predicted values
2 result_df = pd.DataFrame({'Actual': y_test_2, 'Random Forest':
3
4     result_df['date_column'] = pd.to_datetime(result_df.index)
5     result_df = result_df.sort_values(by='date_column') # Sort by date
6
7 # Plotting
8 plt.figure(figsize=(12, 8))
9 plt.plot(result_df['date_column'], result_df['Actual'], label='Actual')
10 plt.plot(result_df['date_column'], result_df['Random Forest'], label='RF')
11 plt.plot(result_df['date_column'], result_df['Linear Regression'], label='LR')
12 plt.plot(result_df['date_column'], result_df['Gradient Boosting'], label='GB')
13 plt.plot(result_df['date_column'], result_df['XGBoost'], label='XGB')
14
15 # Set plot properties
16 plt.title('Actual vs Predicted Stock Prices Over Time')
17 plt.xlabel('Date')
18 plt.ylabel('Stock Price')
19 plt.legend()
20
21 # Show the plot
22 plt.tight_layout()
23 plt.show()

```



Compare residual plots on testing set

In [47]:

```

1 def plot_residuals(model, X_train, y_train, X_test, y_test, model_name):
2     # Use the trained models to make predictions on the testing set
3     preds = model.predict(X_test)

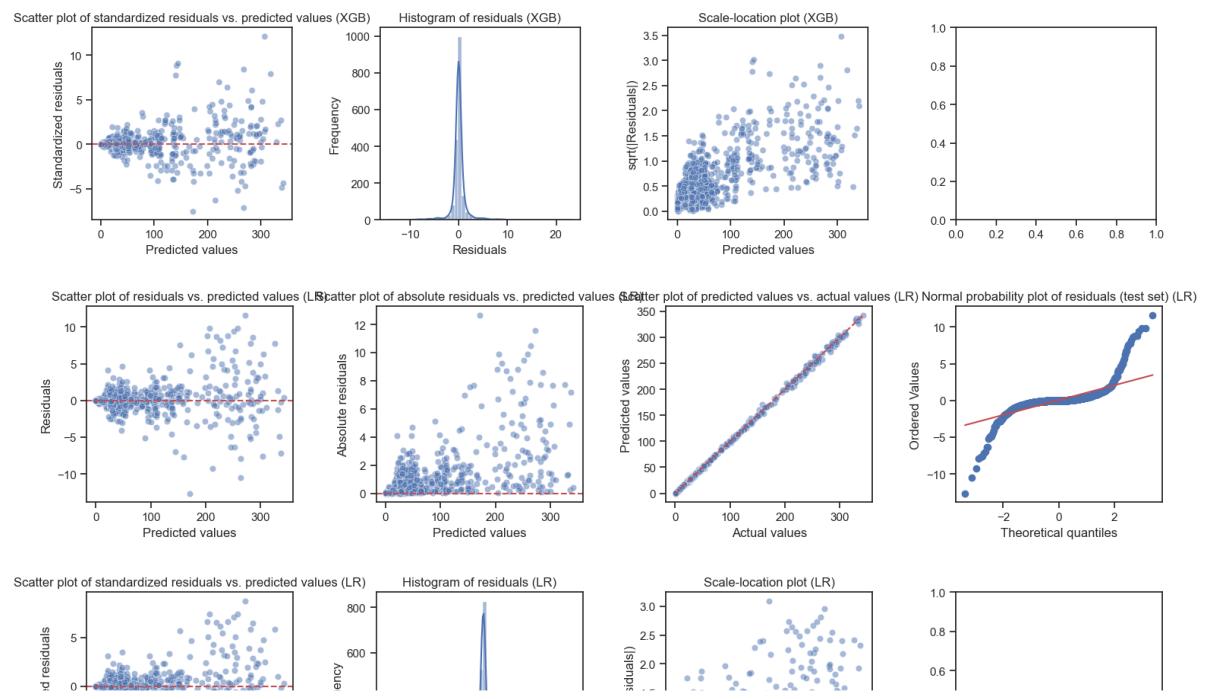
```

```
3     preds = model.predict(X_test_2)
4
5     # Calculate the residuals
6     residuals = y_test_2 - preds
7     std_resid = residuals / np.sqrt(mean_squared_error(y_test_2))
8
9     # Create a 2x4 grid of plots
10    fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))
11    axs = axs.flatten()
12
13    # Plot the residual analysis plots
14    sns.scatterplot(x=preds, y=residuals, ax=axs[0], alpha=0.5)
15    axs[0].axhline(y=0, color='r', linestyle='--')
16    axs[0].set_xlabel('Predicted values')
17    axs[0].set_ylabel('Residuals')
18    axs[0].set_title('Scatter plot of residuals vs. predicted values')
19
20    sns.scatterplot(x=preds, y=np.abs(residuals), ax=axs[1], alpha=0.5)
21    axs[1].axhline(y=0, color='r', linestyle='--')
22    axs[1].set_xlabel('Predicted values')
23    axs[1].set_ylabel('Absolute residuals')
24    axs[1].set_title('Scatter plot of absolute residuals vs. predicted values')
25
26    sns.scatterplot(x=y_test_2, y=preds, ax=axs[2], alpha=0.5)
27    axs[2].plot([y_test_2.min(), y_test_2.max()], [y_test_2.min(), y_test_2.max()])
28    axs[2].set_xlabel('Actual values')
29    axs[2].set_ylabel('Predicted values')
30    axs[2].set_title('Scatter plot of predicted values vs. actual values')
31
32    stats.probplot(residuals, dist="norm", plot=axs[3])
33    axs[3].set_title('Normal probability plot of residuals (test statistic: ' + str(statistic) + ', p-value: ' + str(p_value) + ')')
34
35    sns.scatterplot(x=preds, y=std_resid, ax=axs[4], alpha=0.5)
36    axs[4].axhline(y=0, color='r', linestyle='--')
37    axs[4].set_xlabel('Predicted values')
38    axs[4].set_ylabel('Standardized residuals')
39    axs[4].set_title('Scatter plot of standardized residuals vs. predicted values')
40
41    sns.histplot(residuals, ax=axs[5], bins=50, kde=True)
42    axs[5].set_xlabel('Residuals')
43    axs[5].set_ylabel('Frequency')
44    axs[5].set_title('Histogram of residuals (' + model_label + ')')
45
46    sns.scatterplot(x=preds, y=np.abs(std_resid) ** 0.5, ax=axs[6])
47    axs[6].set_xlabel('Predicted values')
48    axs[6].set_ylabel('sqrt(|Residuals|)')
49    axs[6].set_title('Scale-location plot (' + model_label + ')')
50
51    # Adjust the spacing between subplots for a better display
52    fig.tight_layout(pad=3.0)
53
54    # Plot the residual analysis plots for the Random Forest model
55    plot_residuals(rf_model, X_train_2, y_train_2, X_test_2, y_test_2)
56
57    # Plot the residual analysis plots for the Gradient Boosting model
```

```

57   # Plot the residual analysis plots for the gradient boosting model
58   plot_residuals(gb_model, X_train_2, y_train_2, X_test_2, y_test)
59
60   # Plot the residual analysis plots for the XGBoost model on the test set
61   plot_residuals(xgb_model, X_train_2, y_train_2, X_test_2, y_test)
62
63   # Plot the residual analysis plots for the AdaBoost model on the test set
64   plot_residuals(lr_model, X_train_2, y_train_2, X_test_2, y_test)

```



3.4(b) Analysis

The performance of all the models is very similar and they perform well on unseen data as well. The fact that our actual and predicted prices are nearly identical on plots does not necessarily indicate overfitting. In the context of stock prices, there are a few factors to consider:

1. **Smoothness of Stock Prices:** Stock prices often exhibit smooth trends over time. If the model captures these trends accurately, the actual and predicted prices may indeed align closely.
2. **Model Performance Metrics:** It's important to assess the model's performance using appropriate metrics. Evaluate metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), or others to quantitatively measure the performance of our model. If the model performs well on both the training and testing sets, it's a positive sign. That's what we did throughout this whole project.

It's necessary to thoroughly evaluate the model's performance, understand the nature of the stock data, and be cautious about overfitting.

So the best performing model?: It seems to be Linear Regression again but not by much. This could open possibilities for further experimentation in the future where one idea is to use more derived features such as moving averages, percentage changes in the stock price which could lead to more ambiguity in the dataset and probably the machine learning algorithms can capture deeply hidden patterns.

In [48]: 1 result_df

Out [48]:

	Actual	Random Forest	Linear Regression	Gradient Boosting	XGBoost	date_column
Date						
1986-03-18	0.098090	0.100742	0.132289	0.232969	0.106236	1986-03-18
1986-03-25	0.094618	0.096962	0.133808	0.232969	0.091474	1986-03-25
1986-04-01	0.095486	0.096536	0.143262	0.234940	0.118910	1986-04-01
1986-04-03	0.096354	0.097730	0.142519	0.234940	0.143929	1986-04-03
1986-04-08	0.097222	0.096124	0.144852	0.234940	0.118910	1986-04-08
...
2022-10-14	237.529999	231.158101	230.285693	230.379410	231.859314	2022-10-14
2022-10-20	242.119995	237.104501	238.001480	236.366939	239.170578	2022-10-20
2022-11-04	227.869995	223.341501	219.182747	222.633565	225.120483	2022-11-04
2022-12-16	240.449997	244.780401	245.367082	243.987865	245.122025	2022-12-16
2022-12-23	236.960007	236.019399	237.548485	233.855888	234.333145	2022-12-23

1855 rows × 6 columns

For Intel Corporation

In [49]:

```

1 # Define the models
2 lr_model = LinearRegression()
3 rf_model = RandomForestRegressor(random_state=18)
4 gb_model = GradientBoostingRegressor(random_state=18)
5 xgb_model = xgb.XGBRegressor(random_state=18)
6
7 # Train the models on the training set
8 lr_model.fit(X_train_3, y_train_3)
9 rf_model.fit(X_train_3, y_train_3)
10 gb_model.fit(X_train_3, y_train_3)
11 xgb_model.fit(X_train_3, y_train_3)
12
13 # List of regression models
14 models = {
15     'Linear Regression': lr_model,
16     'Random Forest': rf_model,
17     'Gradient Boosting Regressor': gb_model,
18     'Support Vector Machine': SVR(),
19     'K-Nearest Neighbors': KNeighborsRegressor(),
20     'XGBoost': xgb.XGBRegressor(objective='reg:squarederror')
21 }
22
23 # Number of folds for cross-validation
24 k_folds = 5
25 results = {}
26 for model_name, model in models.items():
27     # R-squared
28     scores_r2 = cross_val_score(model, X_train_3, y_train_3, sc
29
30     # RMSE
31     scores_rmse = np.sqrt(-cross_val_score(model, X_train_3, y_
32
33     # MAE
34     scores_mae = -cross_val_score(model, X_train_3, y_train_3,
35
36     # Explained Variance
37     scores_explained_variance = cross_val_score(model, X_train_
38
39     results[model_name] = {
40         'R-squared': scores_r2,
41         'RMSE': scores_rmse,
42         'MAE': scores_mae,
43         'Explained Variance': scores_explained_variance
44     }
45
46     # Display results
47     for model_name, scores in results.items():
48         print(f'{model_name}: Mean R-squared - {np.mean(scores["R-s
49

```

Linear Regression: Mean R-squared - 0.9986, Mean RMSE - 0.6487, Mean MAE - 0.3411, Mean Explained Variance - 0.9986
 Random Forest: Mean R-squared - 0.9983, Mean RMSE - 0.7159, Mean MAE - 0.3727, Mean Explained Variance - 0.9983

Gradient Boosting Regressor: Mean R-squared – 0.9984, Mean RMSE – 0.7020, Mean MAE – 0.3749, Mean Explained Variance – 0.9984
Support Vector Machine: Mean R-squared – 0.0522, Mean RMSE – 17.1490, Mean MAE – 13.6852, Mean Explained Variance – 0.0580
K-Nearest Neighbors: Mean R-squared – -0.0660, Mean RMSE – 18.1806, Mean MAE – 14.3198, Mean Explained Variance – -0.0652
XGBoost: Mean R-squared – 0.9982, Mean RMSE – 0.7527, Mean MAE – 0.3974, Mean Explained Variance – 0.9982

In [50]:

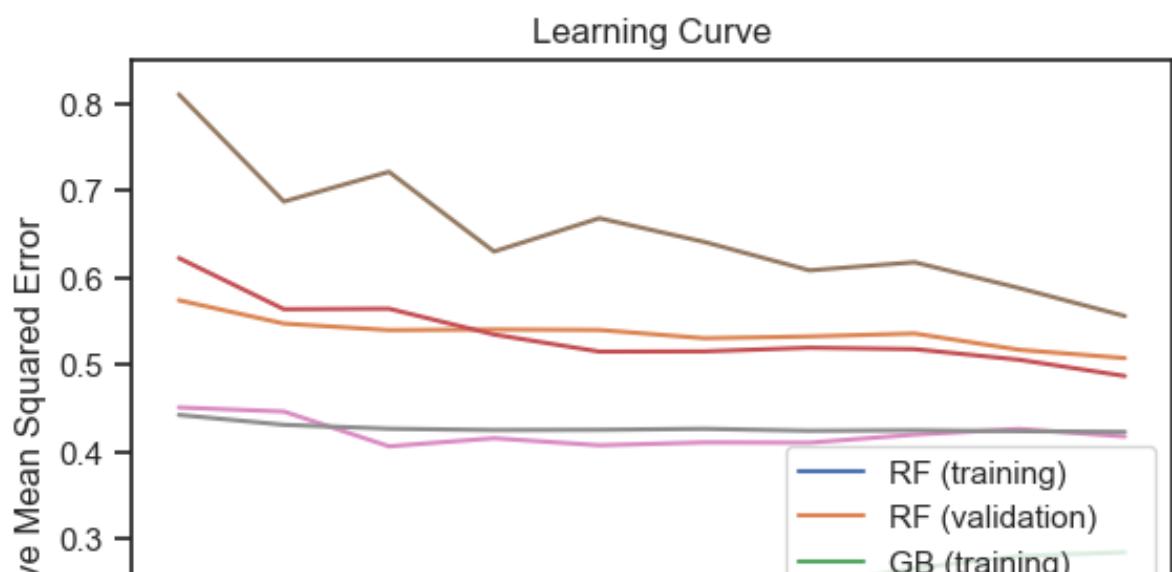
```
1 # After selecting the best model based on cross-validation, eva
2 best_model = LinearRegression() # Based on the best model base
3 best_model.fit(X_train_3, y_train_3)
4
5 # Evaluate on the validation set
6 y_val_pred_3 = best_model.predict(X_val_3)
7 rmse_val_3 = np.sqrt(mean_squared_error(y_val_3, y_val_pred_3))
8 print(f'Validation set RMSE for the best model: {rmse_val_3:.4f}
9
10 # Evaluate on the test set
11 y_test_pred_3 = best_model.predict(X_test_3)
12 rmse_test_3 = np.sqrt(mean_squared_error(y_test_3, y_test_pred_
13 print(f'Test set RMSE for the best model: {rmse_test_3:.4f}')')
```

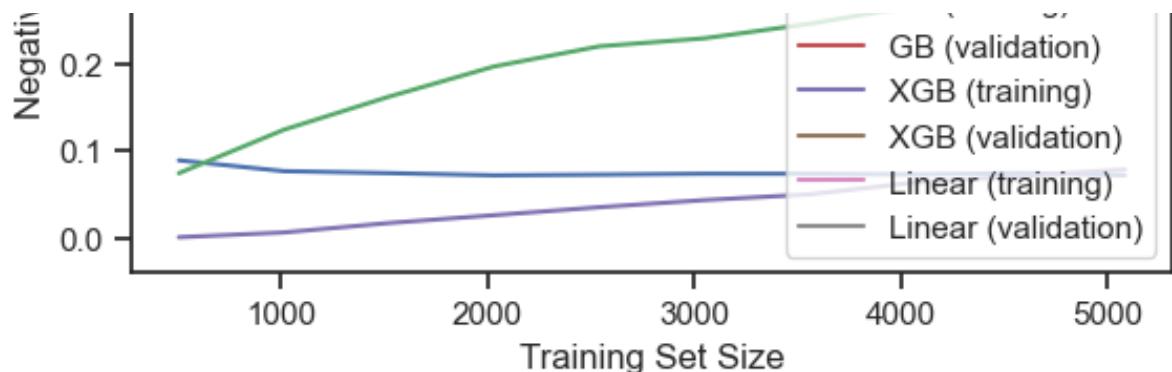
Validation set RMSE for the best model: 0.7307
Test set RMSE for the best model: 0.6284

Plot Learning Curves

In [51]:

```
1 from sklearn.model_selection import learning_curve
2
3 train_sizes, train_scores_rf, val_scores_rf = learning_curve(RandomForestClassifier(), X, y, cv=5, n_jobs=-1)
4 train_sizes, train_scores_gb, val_scores_gb = learning_curve(GradientBoostingClassifier(), X, y, cv=5, n_jobs=-1)
5 train_sizes, train_scores_xgb, val_scores_xgb = learning_curve(XGBClassifier(), X, y, cv=5, n_jobs=-1)
6 train_sizes, train_scores_lin, val_scores_lin = learning_curve(LogisticRegression(), X, y, cv=5, n_jobs=-1)
7
8
9 train_scores_rf = -train_scores_rf
10 val_scores_rf = -val_scores_rf
11 train_scores_gb = -train_scores_gb
12 val_scores_gb = -val_scores_gb
13 train_scores_xgb = -train_scores_xgb
14 val_scores_xgb = -val_scores_xgb
15 train_scores_lin = -train_scores_lin
16 val_scores_lin = -val_scores_lin
17
18
19 plt.plot(train_sizes, np.mean(train_scores_rf, axis=1), label='RF (training)')
20 plt.plot(train_sizes, np.mean(val_scores_rf, axis=1), label='RF (validation)')
21 plt.plot(train_sizes, np.mean(train_scores_gb, axis=1), label='GB (training)')
22 plt.plot(train_sizes, np.mean(val_scores_gb, axis=1), label='GB (validation)')
23 plt.plot(train_sizes, np.mean(train_scores_xgb, axis=1), label='XGB (training)')
24 plt.plot(train_sizes, np.mean(val_scores_xgb, axis=1), label='XGB (validation)')
25 plt.plot(train_sizes, np.mean(train_scores_lin, axis=1), label='Logistic Regression (training)')
26 plt.plot(train_sizes, np.mean(val_scores_lin, axis=1), label='Logistic Regression (validation)')
27
28 plt.title('Learning Curve')
29 plt.xlabel('Training Set Size')
30 plt.ylabel('Negative Mean Squared Error')
31 plt.legend()
32 plt.show()
```





Compare Residual plots for training and validation sets

In [52]:

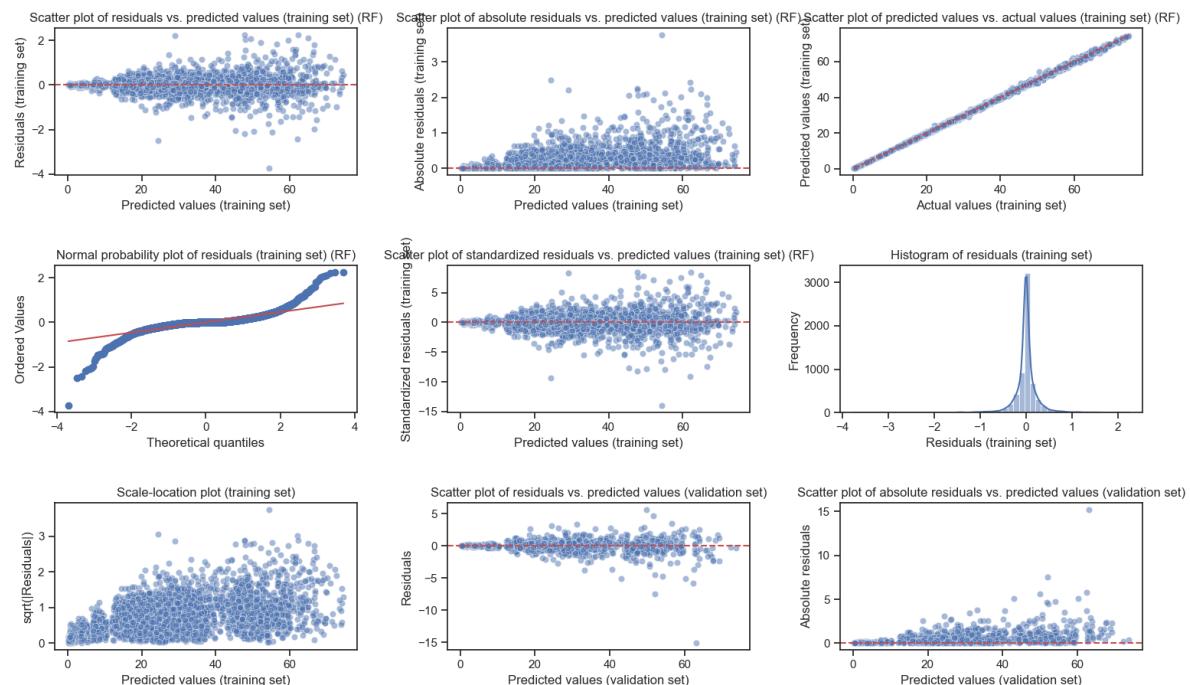
```
1 from scipy import stats
2
3 def plot_residuals(model, X_train, y_train, X_val, y_val, model_1
4     # Use the trained models to make predictions on the training
5     train_preds_3 = model.predict(X_train_3)
6     val_preds_3 = model.predict(X_val_3)
7
8     # Calculate the residuals on the training and validation sets
9     train_residuals_3 = y_train_3 - train_preds_3
10    val_residuals_3 = y_val_3 - val_preds_3
11
12    # Calculate the standardized residuals on the training set
13    train_std_resid_3 = train_residuals_3 / np.sqrt(mean_squared_
14
15    # Create a 4x4 grid of plots
16    fig, axs = plt.subplots(nrows=5, ncols=3, figsize=(16, 16))
17    fig.subplots_adjust(hspace=0.5)
18    axs = axs.flatten()
19
20    # Plot the residual analysis plots
21    sns.scatterplot(x=train_preds_3, y=train_residuals_3, ax=axs[0])
22    axs[0].axhline(y=0, color='r', linestyle='--')
23    axs[0].set_xlabel('Predicted values (training set)')
24    axs[0].set_ylabel('Residuals (training set)')
25    axs[0].set_title('Scatter plot of residuals vs. predicted val
26
27    sns.scatterplot(x=train_preds_3, y=np.abs(train_residuals_3),
28                    ax=axs[1])
29    axs[1].axhline(y=0, color='r', linestyle='--')
30    axs[1].set_xlabel('Predicted values (training set)')
31    axs[1].set_ylabel('Absolute residuals (training set)')
32    axs[1].set_title('Scatter plot of absolute residuals vs. pred
33
34    sns.scatterplot(x=y_train_3, y=train_preds_3, ax=axs[2], alpha=
35    axs[2].plot([y_train.min(), y_train.max()], [y_train.min(), y_
36    axs[2].set_xlabel('Actual values (training set)')
37    axs[2].set_ylabel('Predicted values (training set)')
38    axs[2].set_title('Scatter plot of predicted values vs. actual
```

```
38
39     stats.probplot(train_residuals_3, dist="norm", plot= axs[3])
40     axs[3].set_title('Normal probability plot of residuals (train
41
42     sns.scatterplot(x=train_preds_3, y=train_std_resid_3, ax=axs[4])
43     axs[4].axhline(y=0, color='r', linestyle='--')
44     axs[4].set_xlabel('Predicted values (training set)')
45     axs[4].set_ylabel('Standardized residuals (training set)')
46     axs[4].set_title('Scatter plot of standardized residuals vs.
47
48     sns.histplot(train_residuals_3, ax=axs[5], bins=50, kde=True)
49     axs[5].set_xlabel('Residuals (training set)')
50     axs[5].set_ylabel('Frequency')
51     axs[5].set_title('Histogram of residuals (training set)')
52
53     sns.scatterplot(x=train_preds_3, y=np.abs(train_std_resid_3)
54     axs[6].set_xlabel('Predicted values (training set)')
55     axs[6].set_ylabel('sqrt(|Residuals|)')
56     axs[6].set_title('Scale-location plot (training set)')
57
58 # Plot the residual analysis plots for the validation set
59 val_preds_3 = model.predict(X_val_3)
60 val_residuals_3 = y_val_3 - val_preds_3
61 val_std_resid_3 = val_residuals_3 / np.sqrt(mean_squared_error(
62
63     sns.scatterplot(x=val_preds_3, y=val_residuals_3, ax=axs[7])
64     axs[7].axhline(y=0, color='r', linestyle='--')
65     axs[7].set_xlabel('Predicted values (validation set)')
66     axs[7].set_ylabel('Residuals')
67     axs[7].set_title('Scatter plot of residuals vs. predicted val
68
69     sns.scatterplot(x=val_preds_3, y=np.abs(val_residuals_3), ax=
70     axs[8].axhline(y=0, color='r', linestyle='--')
71     axs[8].set_xlabel('Predicted values (validation set)')
72     axs[8].set_ylabel('Absolute residuals')
73     axs[8].set_title('Scatter plot of absolute residuals vs. pred
74
75     stats.probplot(val_residuals_3, dist="norm", plot=axs[9])
76     axs[9].set_title('Normal probability plot of residuals (valid
77
78
79     sns.scatterplot(x=val_preds_3, y=val_std_resid_3, ax=axs[10])
80     axs[10].axhline(y=0, color='r', linestyle='--')
81     axs[10].set_xlabel('Predicted values (validation set)')
82     axs[10].set_ylabel('Standardized residuals')
83     axs[10].set_title('Scatter plot of standardized residuals vs.
84
85     sns.scatterplot(x=y_val_3, y=val_preds_3, ax=axs[11], alpha=0.5)
86     axs[11].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.
87     axs[11].set_xlabel('Actual values (validation set)')
88     axs[11].set_ylabel('Predicted values (validation set)')
89     axs[11].set_title('Scatter plot of predicted values vs. actual
90
91     sns.histplot(val_residuals_3, ax=axs[12], bins=50, kde=True)
```

```

92     axs[12].set_xlabel('Residuals (validation set)')
93     axs[12].set_ylabel('Frequency')
94     axs[12].set_title('Histogram of residuals (validation set)')
95
96     sns.scatterplot(x=val_preds_3, y=np.abs(val_std_resid_3) ** 0.5)
97     axs[13].set_xlabel('Predicted values (validation set)')
98     axs[13].set_ylabel('sqrt(|Residuals|)')
99     axs[13].set_title('Scale-location plot (validation set)')
100
101 # Adjust the spacing between subplots for a better display
102 fig.tight_layout(pad=3.0)
103
104 # Plot the residual analysis plots for the Random Forest model or
105 plot_residuals(rf_model, X_train_3, y_train_3, X_val_3, y_val_3,
106
107 # Plot the residual analysis plots for the Gradient Boosting mode
108 plot_residuals(gb_model, X_train_3, y_train_3, X_val_3, y_val_3,
109
110 # Plot the residual analysis plots for the XGBoost model on the t
111 plot_residuals(xgb_model, X_train_3, y_train_3, X_val_3, y_val_3,
112
113 # Plot the residual analysis plots for the Linear Regression mode
114 plot_residuals(lr_model, X_train_3, y_train_3, X_val_3, y_val_3,
115

```



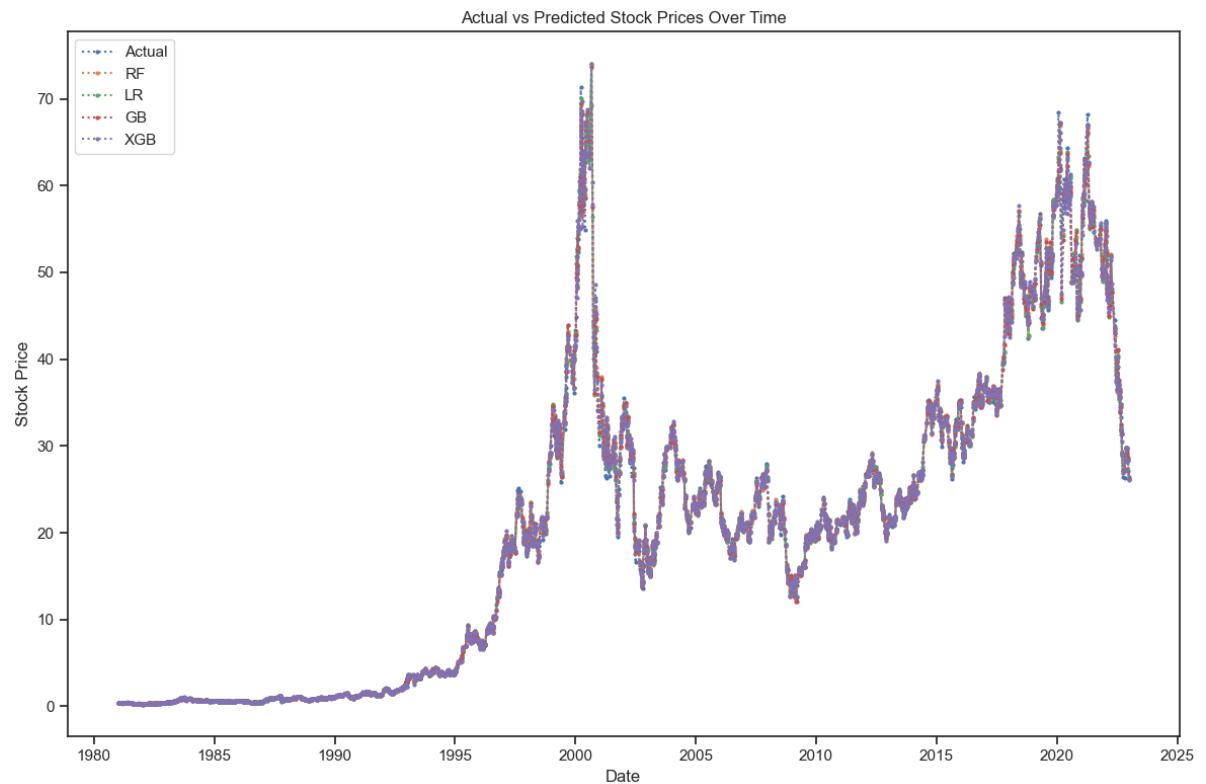
In [53]:

```
1 # Use the trained models to make predictions on the testing set
2 rf_preds = rf_model.predict(X_test_3)
3 gb_preds = gb_model.predict(X_test_3)
4 xgb_preds = xgb_model.predict(X_test_3)
5 lin_preds = lr_model.predict(X_test_3)
6
7 rf_r2 = r2_score(y_test_3, rf_preds)
8 rf_rmse = mean_squared_error(y_test_3, rf_preds, squared=False)
9 print('Random Forest R-squared score on testing set:', rf_r2)
10 print('Random Forest RMSE on testing set:', rf_rmse)
11
12 gb_r2 = r2_score(y_test_3, gb_preds)
13 gb_rmse = mean_squared_error(y_test_3, gb_preds, squared=False)
14 print('Gradient Boosting R-squared score on testing set:', gb_r2)
15 print('Gradient Boosting RMSE on testing set:', gb_rmse)
16
17 xgb_r2 = r2_score(y_test_3, xgb_preds)
18 xgb_rmse = mean_squared_error(y_test_3, xgb_preds, squared=False)
19 print('XGBoost R-squared score on testing set:', xgb_r2)
20 print('XGBoost RMSE on testing set:', xgb_rmse)
21
22 lin_r2 = r2_score(y_test_3, lin_preds)
23 lin_rmse = mean_squared_error(y_test_3, lin_preds, squared=False)
24 print('Linear Regression score on testing set:', lin_r2)
25 print('Linear Regression RMSE on testing set:', lin_rmse)
```

Random Forest R-squared score on testing set: 0.9984824715596768
Random Forest RMSE on testing set: 0.6745659488825815
Gradient Boosting R-squared score on testing set: 0.99852735944608
Gradient Boosting RMSE on testing set: 0.6645143645199214
XGBoost R-squared score on testing set: 0.9980068784019159
XGBoost RMSE on testing set: 0.7730772497980173
Linear Regression score on testing set: 0.9986829962272291
Linear Regression RMSE on testing set: 0.6284192858946899

Visualize predicted and actual values

```
In [54]: # Create a DataFrame with actual and predicted values
1 result_df = pd.DataFrame({'Actual': y_test_3, 'Random Forest': rf_
2
3
4 result_df['date_column'] = pd.to_datetime(result_df.index)
5 result_df = result_df.sort_values(by='date_column') # Sort by date
6
7
# Plotting
8 plt.figure(figsize=(12, 8))
9 plt.plot(result_df['date_column'], result_df['Actual'], label='Actual')
10 plt.plot(result_df['date_column'], result_df['Random Forest'], label='RF')
11 plt.plot(result_df['date_column'], result_df['Linear Regression'], label='LR')
12 plt.plot(result_df['date_column'], result_df['Gradient Boosting'], label='GB')
13 plt.plot(result_df['date_column'], result_df['XGBoost'], label='XGB')
14
15 # Set plot properties
16 plt.title('Actual vs Predicted Stock Prices Over Time')
17 plt.xlabel('Date')
18 plt.ylabel('Stock Price')
19 plt.legend()
20
21 # Show the plot
22 plt.tight_layout()
23 plt.show()
```



Compare residual plots for test data

```
In [55]: def plot_residuals(model, X_train, y_train, X_test, y_test, model_name):
    # Use the trained models to make predictions on the testing data
    preds = model.predict(X_test_3)
```

```
3     preds = model.predict(X_test_3)
4
5     # Calculate the residuals
6     residuals = y_test_3 - preds
7     std_resid = residuals / np.sqrt(mean_squared_error(y_test_3))
8
9     # Create a 2x4 grid of plots
10    fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(16, 8))
11    axs = axs.flatten()
12
13    # Plot the residual analysis plots
14    sns.scatterplot(x=preds, y=residuals, ax=axs[0], alpha=0.5)
15    axs[0].axhline(y=0, color='r', linestyle='--')
16    axs[0].set_xlabel('Predicted values')
17    axs[0].set_ylabel('Residuals')
18    axs[0].set_title('Scatter plot of residuals vs. predicted values')
19
20    sns.scatterplot(x=preds, y=np.abs(residuals), ax=axs[1], alpha=0.5)
21    axs[1].axhline(y=0, color='r', linestyle='--')
22    axs[1].set_xlabel('Predicted values')
23    axs[1].set_ylabel('Absolute residuals')
24    axs[1].set_title('Scatter plot of absolute residuals vs. predicted values')
25
26    sns.scatterplot(x=y_test_3, y=preds, ax=axs[2], alpha=0.5)
27    axs[2].plot([y_test_3.min(), y_test_3.max()], [y_test_3.min(), y_test_3.max()])
28    axs[2].set_xlabel('Actual values')
29    axs[2].set_ylabel('Predicted values')
30    axs[2].set_title('Scatter plot of predicted values vs. actual values')
31
32    stats.probplot(residuals, dist="norm", plot=axs[3])
33    axs[3].set_title('Normal probability plot of residuals (test statistic: ' + str(statistic) + ', p-value: ' + str(p_value) + ')')
34
35    sns.scatterplot(x=preds, y=std_resid, ax=axs[4], alpha=0.5)
36    axs[4].axhline(y=0, color='r', linestyle='--')
37    axs[4].set_xlabel('Predicted values')
38    axs[4].set_ylabel('Standardized residuals')
39    axs[4].set_title('Scatter plot of standardized residuals vs. predicted values')
40
41    sns.histplot(residuals, ax=axs[5], bins=50, kde=True)
42    axs[5].set_xlabel('Residuals')
43    axs[5].set_ylabel('Frequency')
44    axs[5].set_title('Histogram of residuals (' + model_label + ')')
45
46    sns.scatterplot(x=preds, y=np.abs(std_resid) ** 0.5, ax=axs[6])
47    axs[6].set_xlabel('Predicted values')
48    axs[6].set_ylabel('sqrt(|Residuals|)')
49    axs[6].set_title('Scale-location plot (' + model_label + ')')
50
51    # Adjust the spacing between subplots for a better display
52    fig.tight_layout(pad=3.0)
53
54    # Plot the residual analysis plots for the Random Forest model
55    plot_residuals(rf_model, X_train_3, y_train_3, X_test_3, y_test_3)
56
57    # Plot the residual analysis plots for the Gradient Boosting model
```