baseline_model

June 28, 2025

1 Baseline Model Notebook

1.0.1 In this notebook, we will go through the process of machine learning model training using the different python libraries.

1.0.2 Project Definition: Stock Price Movement Prediction

Goal: Develop a supervised machine learning model that predicts whether the closing price of the AAPL stock will increase on the next trading day, based on daily historical market data.

Dataset: A CSV file containing the following columns for each trading day: • Open (opening price) • High (highest price) • Low (lowest price) • Close (closing price) • Volume (number of shares traded)

Task Specification: • Target variable: Price_Up_Tomorrow (binary; 1 if Close_tomorrow > Close_today, otherwise 0) • Features: All columns above, as well as any engineered features you design (e.g., daily return, volatility, moving averages). • Prediction task: For each day, use available information to predict whether the closing price on the following day will be higher than the current day's close.

Project Steps: 1. EDA: Explore and visualize the data, identify trends, missing values, and correlations. 2. Preprocessing: Handle missing values, scale/normalize features, and create the target variable. 3. Feature Engineering: Create additional features such as daily returns, volatility, and rolling averages. 4. Model Training: Split the data into train/test sets; train and evaluate multiple classifiers (e.g., Random Forest, Logistic Regression, XGBoost). 5. Evaluation: Assess model performance using classification metrics (accuracy, precision, recall, ROC-AUC, confusion matrix). 6. Explainability: Use feature importance and SHAP values to interpret model predictions. 7. Discussion: Summarize findings, limitations, and possible next steps.

Constraints: • The model should use only features available at prediction time (no "peeking" into future data). • The solution must be reproducible and clearly documented.

[ ]:
```python
# Load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns # Data visualization library
import calendar
import pandas_datareader.data as web
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder # Data␣
 ↪preprocessing libraries
from sklearn.ensemble import RandomForestClassifier # Machine learning libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪auc
from sklearn.feature_selection import mutual_info_regression
import shap # SHAP (SHapley Additive exPlanations) is a unified approach to␣
 ↪explain the output of any machine learning model.

import xgboost as xgb # XGBoost is an optimized distributed gradient boosting␣
 ↪library designed to be highly efficient, flexible and portable.
import yfinance as yf # Yahoo Finance is a media property that is part of␣
 ↪Yahoo's network, providing financial news, data and commentary including␣
 ↪stock quotes, press releases, financial reports, and original content.
```

1.0.3 load data

[47]:
```
df = yf.download("AAPL", start="2020-01-01", end="2024-12-31")
df.to_csv("stock_data.csv")
```

```
/var/folders/x0/x2vffgpn4mv88fvwb47hvlmc0000gn/T/ipykernel_61651/3440584184.py:1
: FutureWarning: YF.download() has changed argument auto_adjust default to True
  df = yf.download("AAPL", start="2020-01-01", end="2024-12-31")
[*********************100%***********************]  1 of 1 completed
```

1.0.4 1. Exploratory Data Analysis (EDA)

[48]:
```
# explore the data
df.info()
print(df.describe())
print(df.isnull().sum())
```

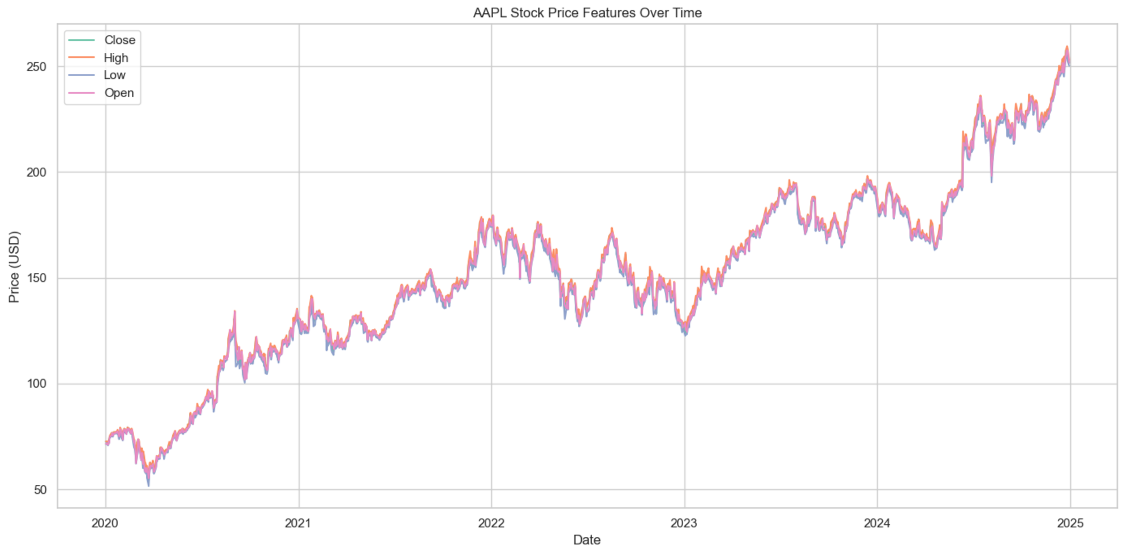
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1257 entries, 2020-01-02 to 2024-12-30
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   (Close, AAPL)   1257 non-null   float64
 1   (High, AAPL)    1257 non-null   float64
 2   (Low, AAPL)     1257 non-null   float64
 3   (Open, AAPL)    1257 non-null   float64
 4   (Volume, AAPL)  1257 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 58.9 KB
Price          Close         High          Low         Open        Volume
Ticker          AAPL         AAPL         AAPL         AAPL          AAPL
```

```
count   1257.000000  1257.000000  1257.000000  1257.000000  1.257000e+03
mean     151.900499   153.427262   150.196283   151.743844  9.061449e+07
std       41.943932    42.082741    41.718433    41.883286  5.325627e+07
min       54.378582    55.379535    51.528416    55.277744  2.323470e+07
25%      126.750748   127.839523   124.727659   126.223154  5.546960e+07
50%      150.537064   152.292443   148.535311   150.362277  7.629970e+07
75%      176.443726   178.148983   175.007757   176.290017  1.077601e+08
max      258.396667   259.474086   257.010028   257.568678  4.265100e+08
Price   Ticker
Close   AAPL      0
High    AAPL      0
Low     AAPL      0
Open    AAPL      0
Volume  AAPL      0
dtype: int64
```

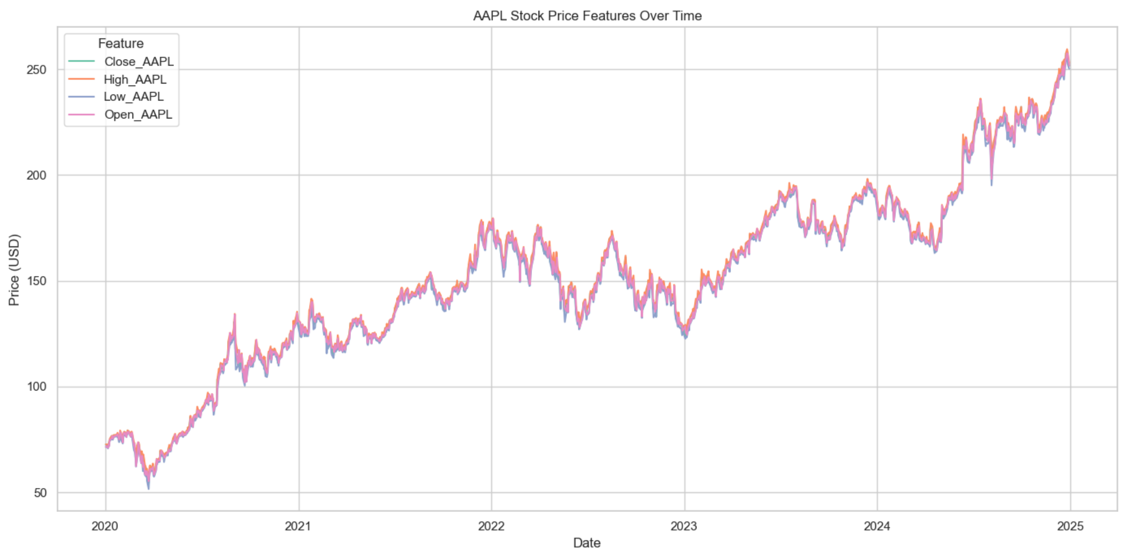
1.0.5 simple data visualizations

Stock prices over time

[49]:
```python
features = ['Close', 'High', 'Low', 'Open']
plt.figure(figsize= (14,7))

for feat in features:
    plt.plot(df.index, df[feat], label=feat)
    plt.plot()

plt.title('AAPL Stock Price Features Over Time')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

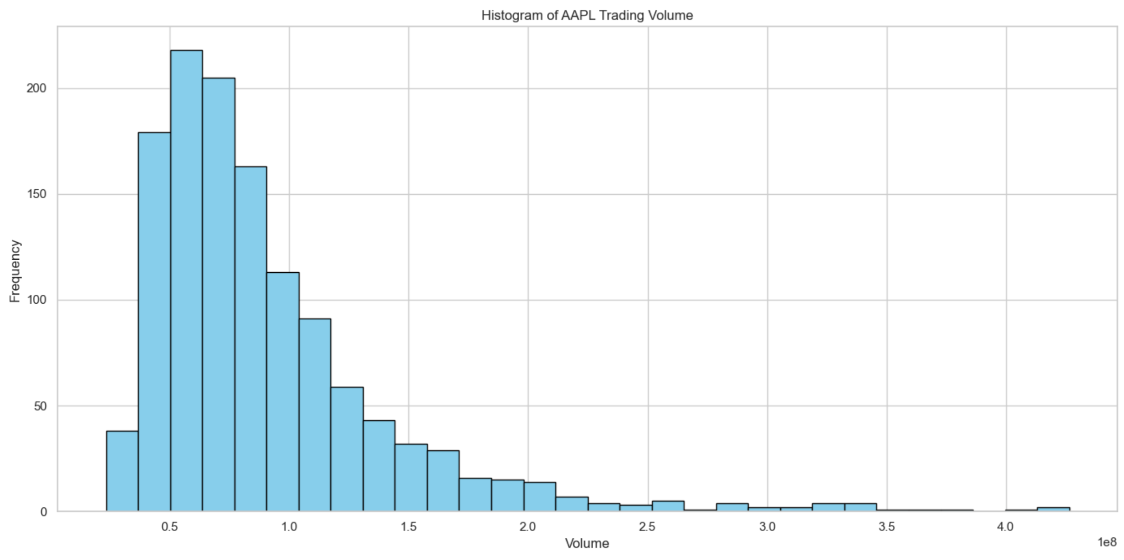


[50]:
```
df_for_sns = df.copy()
df_for_sns.columns = ['_'.join(col).strip() for col in df_for_sns.columns.
 ↪values]

features_for_sns = ['Close_AAPL', 'High_AAPL', 'Low_AAPL', 'Open_AAPL']

# df long convertion
df_long = df_for_sns[features_for_sns].reset_index().rename(columns={'index':␣
 ↪'Date'}).melt(
    id_vars='Date',
    var_name='Feature',
    value_name='Price'
)

sns.set_theme(style="whitegrid", palette="Set2")

plt.figure(figsize=(14,7))
sns.lineplot(data=df_long, x='Date', y='Price', hue='Feature')

plt.title('AAPL Stock Price Features Over Time')
plt.xlabel('Date')
plt.ylabel('Price (USD)')
plt.grid(True)
plt.tight_layout()
plt.show()
```



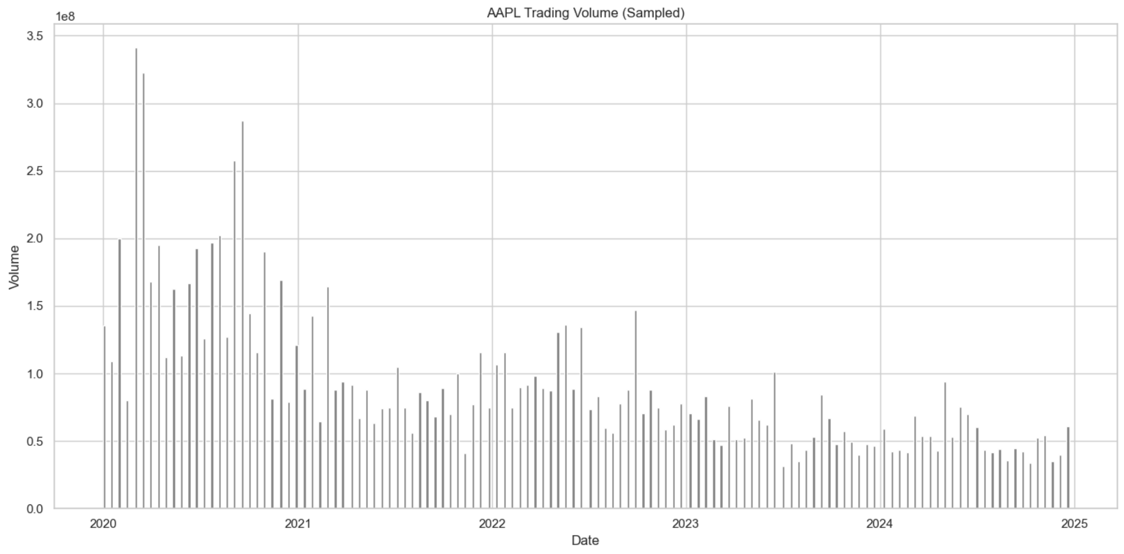
Volume histogram

```
[51]: plt.figure(figsize=(14, 7))
      plt.hist(df[('Volume', 'AAPL')], bins=30, color='skyblue', edgecolor='black')
      plt.title('Histogram of AAPL Trading Volume')
      plt.xlabel('Volume')
      plt.ylabel('Frequency')
      plt.grid(True)
      plt.tight_layout()
      plt.show()
```



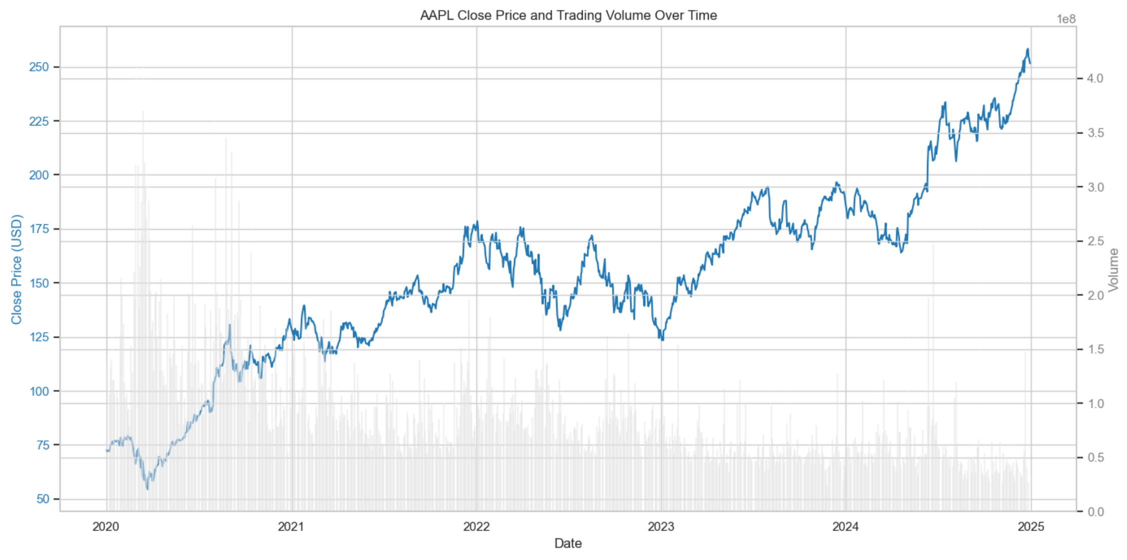
time series of volume

[52]:
```
df_downsampled = df[::10]

plt.figure(figsize=(14, 7))
plt.bar(df_downsampled.index, df_downsampled[('Volume', 'AAPL')], width=5,␣
  ↪color='gray')
plt.title('AAPL Trading Volume (Sampled)')
plt.xlabel('Date')
plt.ylabel('Volume')
plt.tight_layout()
plt.show()
```



[53]:
```
fig, ax1 = plt.subplots(figsize=(14, 7))

color_price = 'tab:blue'
ax1.set_xlabel('Date')
ax1.set_ylabel('Close Price (USD)', color=color_price)
ax1.plot(df.index, df[('Close', 'AAPL')], color=color_price, label='Close␣
  ↪Price')
ax1.tick_params(axis='y', labelcolor=color_price)

ax2 = ax1.twinx()
color_volume = 'tab:gray'
ax2.set_ylabel('Volume', color=color_volume)
ax2.bar(df.index, df[('Volume', 'AAPL')], color=color_volume, alpha=0.3,␣
  ↪width=1.0)
ax2.tick_params(axis='y', labelcolor=color_volume)
```

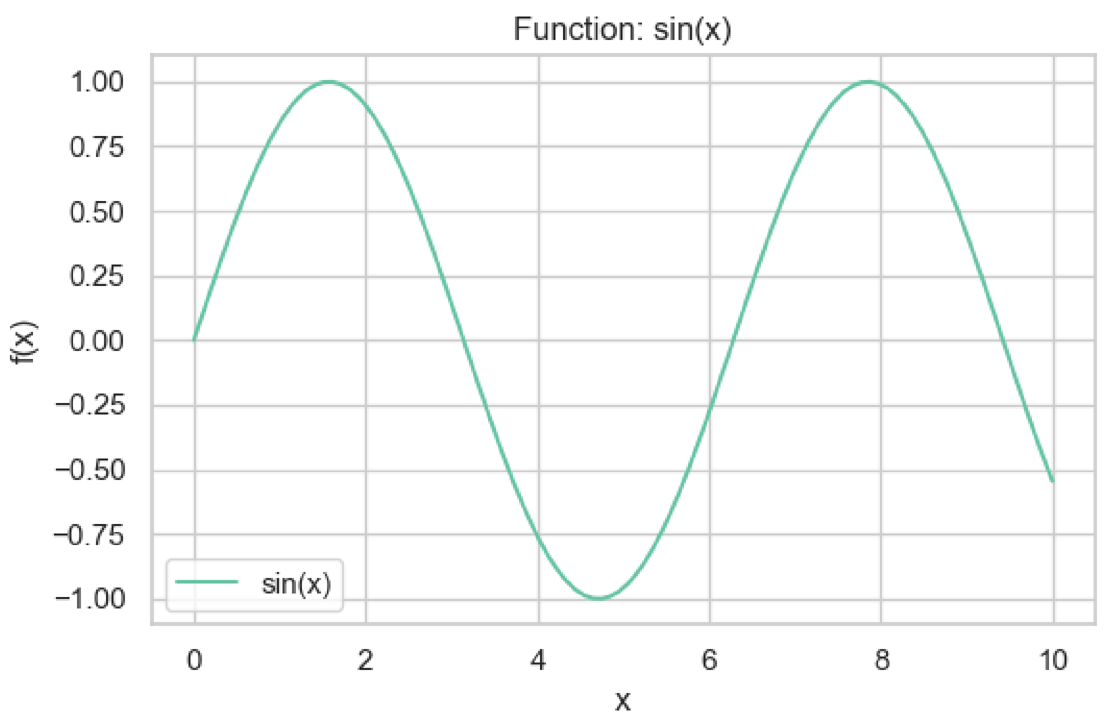
```
#
plt.title('AAPL Close Price and Trading Volume Over Time')
fig.tight_layout()
plt.show()
```

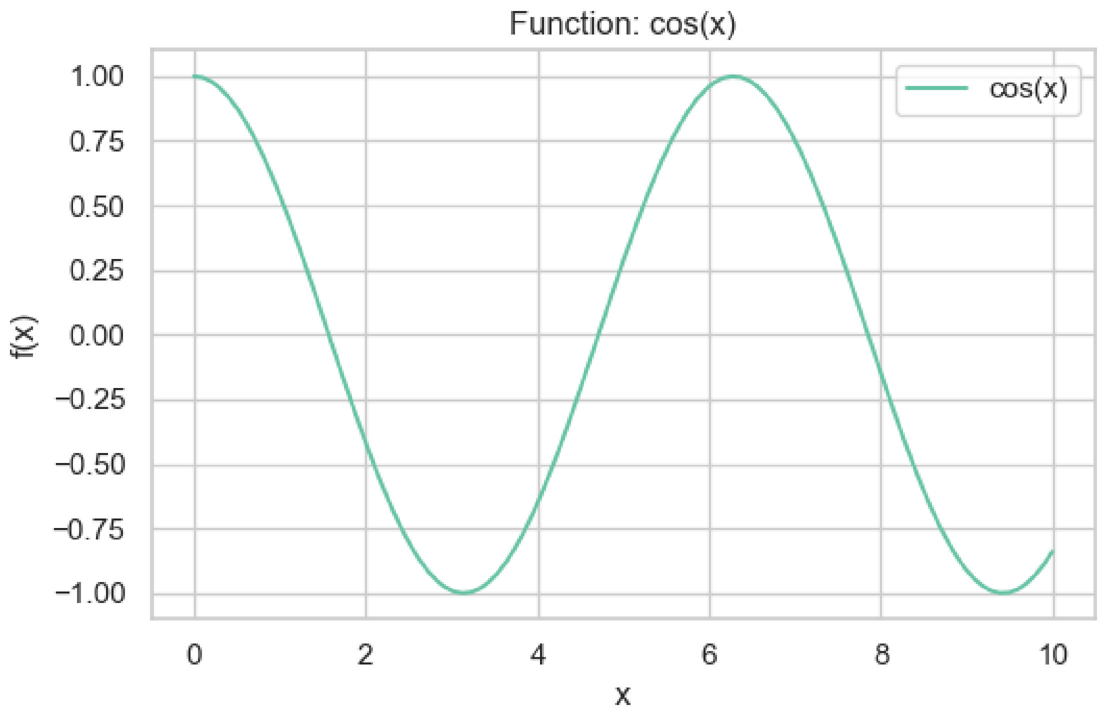


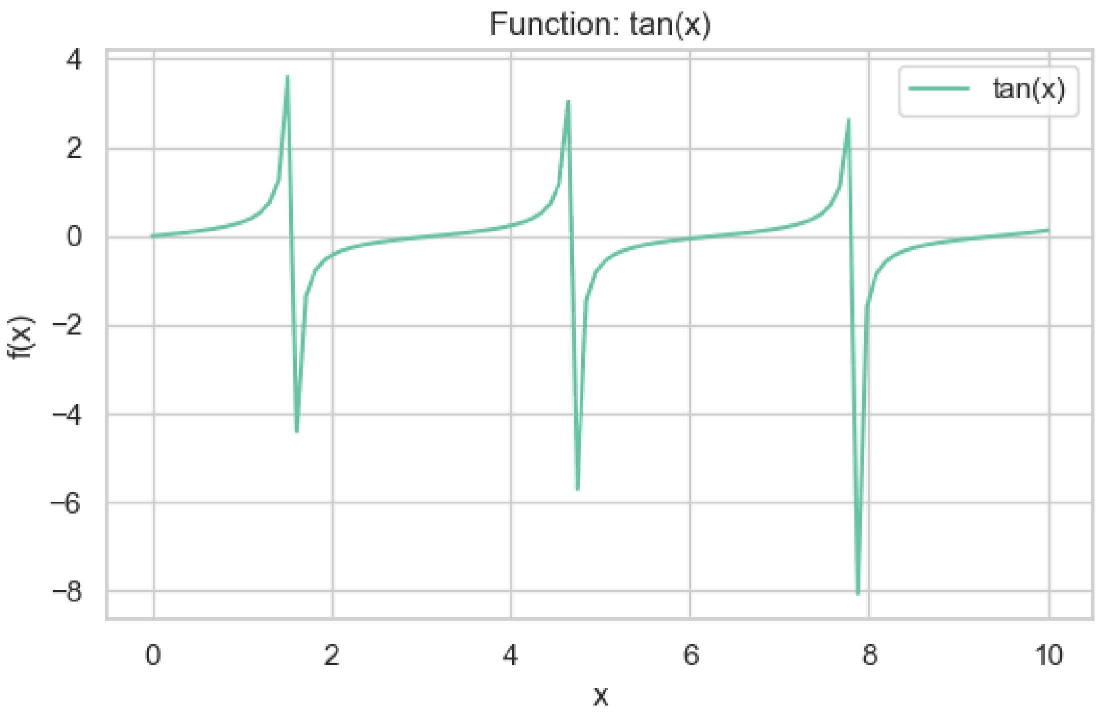
```
[ ]:
```

some general examples

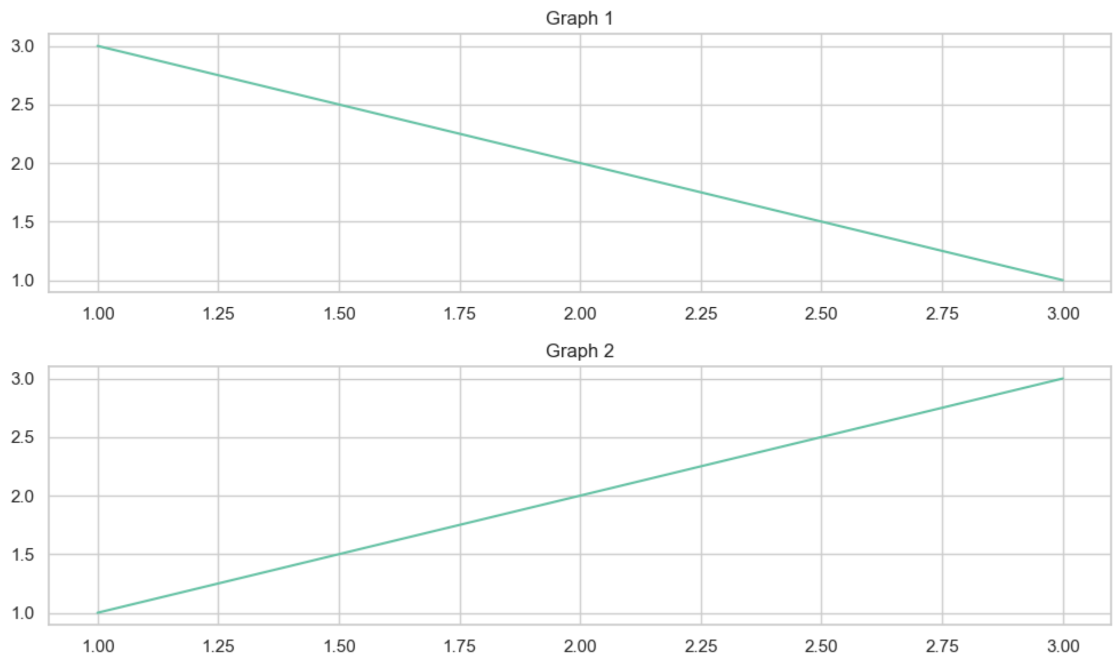
```
[54]: x = np.linspace(0, 10, 100)
functions = {
    'sin(x)': np.sin(x),
    'cos(x)': np.cos(x),
    'tan(x)': np.tan(x) / 5  #    5
}

for name, y in functions.items():
    plt.figure(figsize=(6, 4))        #
    plt.plot(x, y, label=name)
    plt.title(f'Function: {name}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```


Function: sin(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
f(x)
0
2
4
6
8
10
x
sin(x)


Function: cos(x)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
f(x)
0
2
4
6
8
10
x
cos(x)



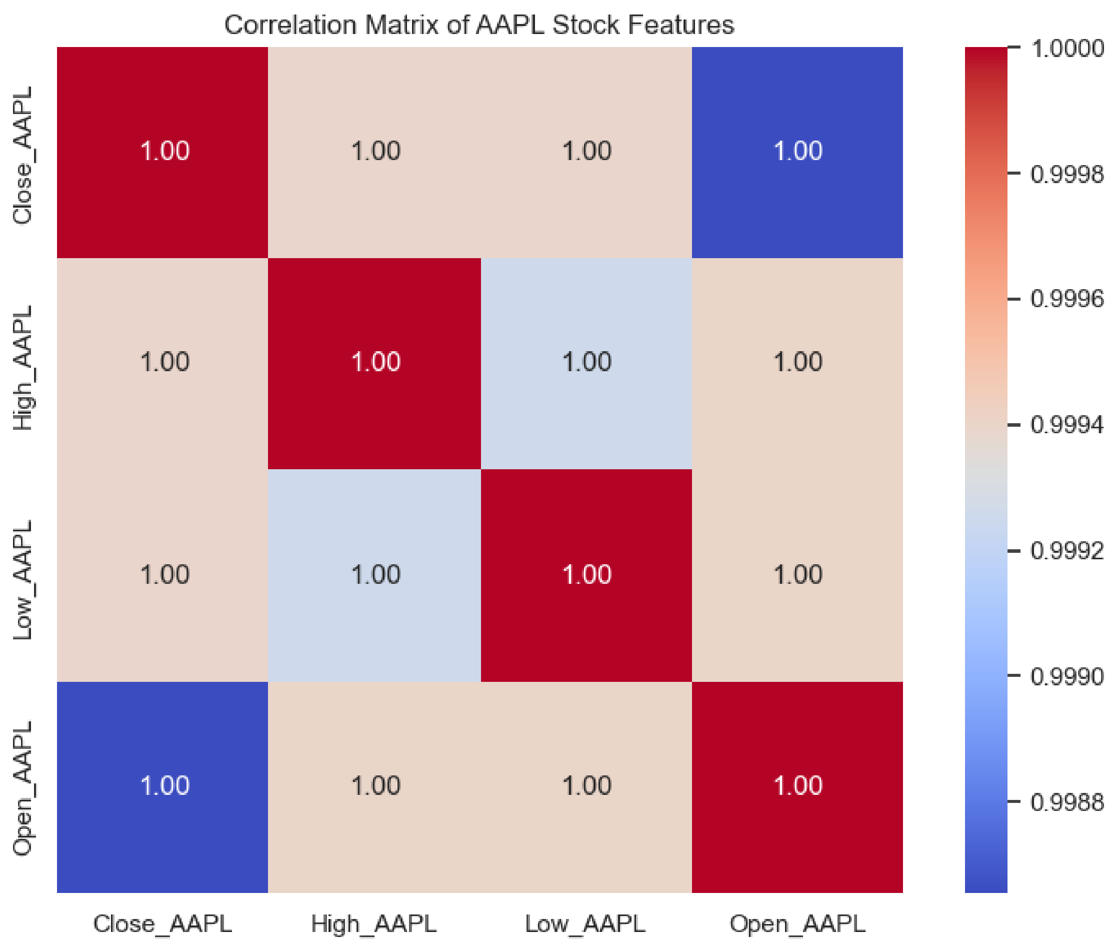
```
[55]: fig, axs = plt.subplots(2, 1, figsize=(10, 6))

      axs[0].plot([1, 2, 3], [3, 2, 1])
      axs[0].set_title("Graph 1")

      axs[1].plot([1, 2, 3], [1, 2, 3])
      axs[1].set_title("Graph 2")

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



Correlation matrix

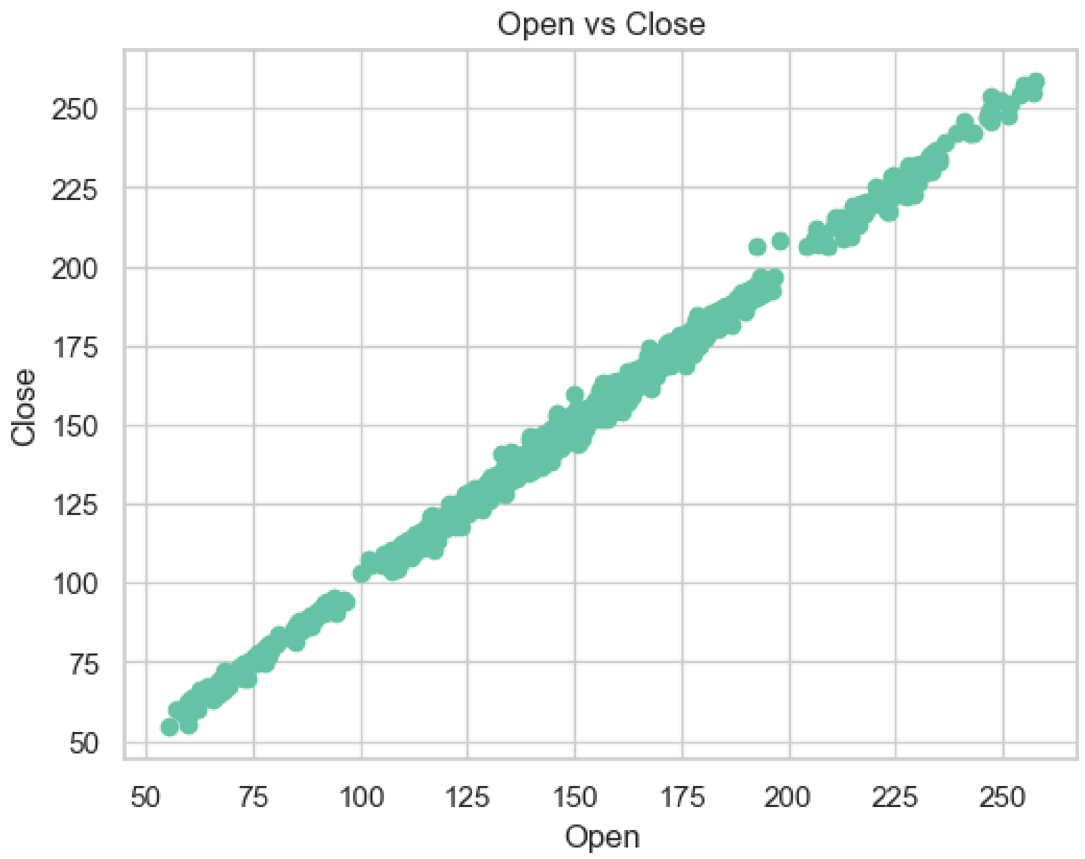
[56]:
```
# Compute and plot the correlation matrix for the features
corr_matrix = df_for_sns[features_for_sns].corr()

print(corr_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Matrix of AAPL Stock Features')
plt.tight_layout()
plt.show()
```

```
            Close_AAPL  High_AAPL  Low_AAPL  Open_AAPL
Close_AAPL    1.000000   0.999396  0.999392   0.998656
High_AAPL     0.999396   1.000000  0.999256   0.999398
Low_AAPL      0.999392   0.999256  1.000000   0.999399
Open_AAPL     0.998656   0.999398  0.999399   1.000000
```



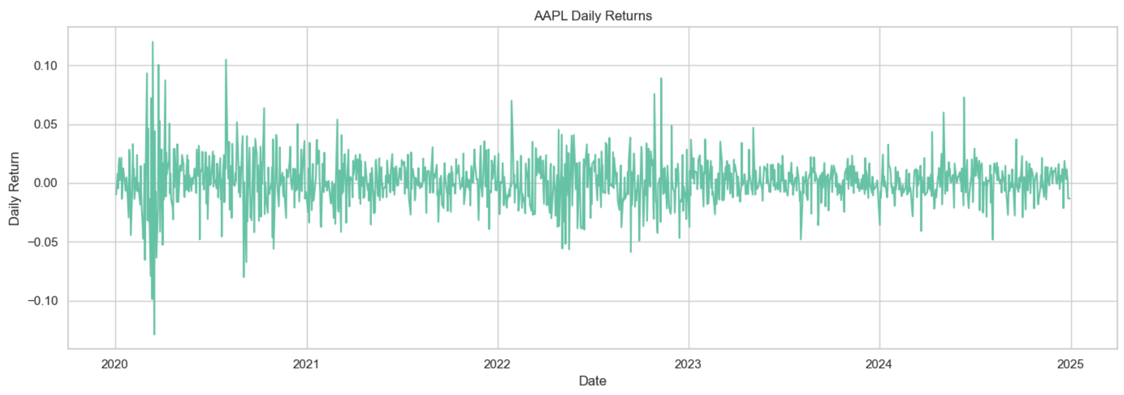
we understand that the correlation between the features is very high, thanks to the fact that they are all related to the same stock. This means that we need to be careful when training our model, as it may lead to overfitting.

[57]:
```
plt.scatter(df['Open'], df['Close'])
plt.xlabel('Open')
plt.ylabel('Close')
plt.title('Open vs Close')
plt.show()
```



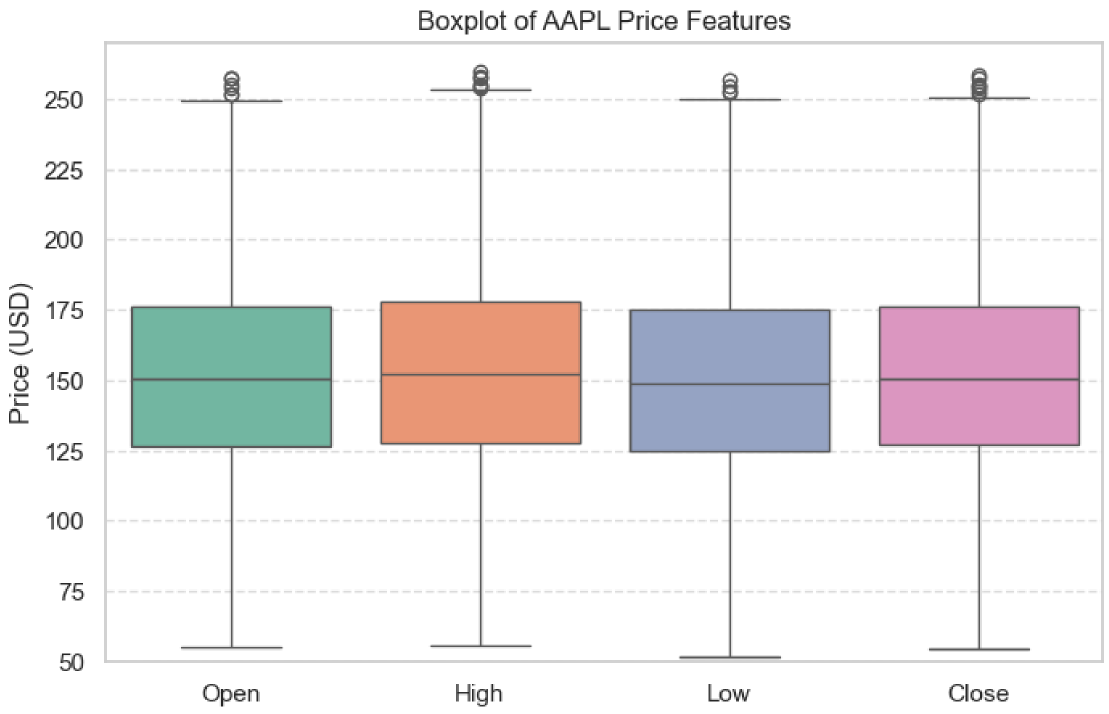
Daily returns

[58]:
```
df['Daily_Return'] = df[('Close', 'AAPL')].pct_change()

df['Daily_Return'].describe()

plt.figure(figsize=(14,5))
plt.plot(df.index, df['Daily_Return'])
plt.title('AAPL Daily Returns')
plt.xlabel('Date')
plt.ylabel('Daily Return')
plt.grid(True)
plt.tight_layout()
plt.show()
```

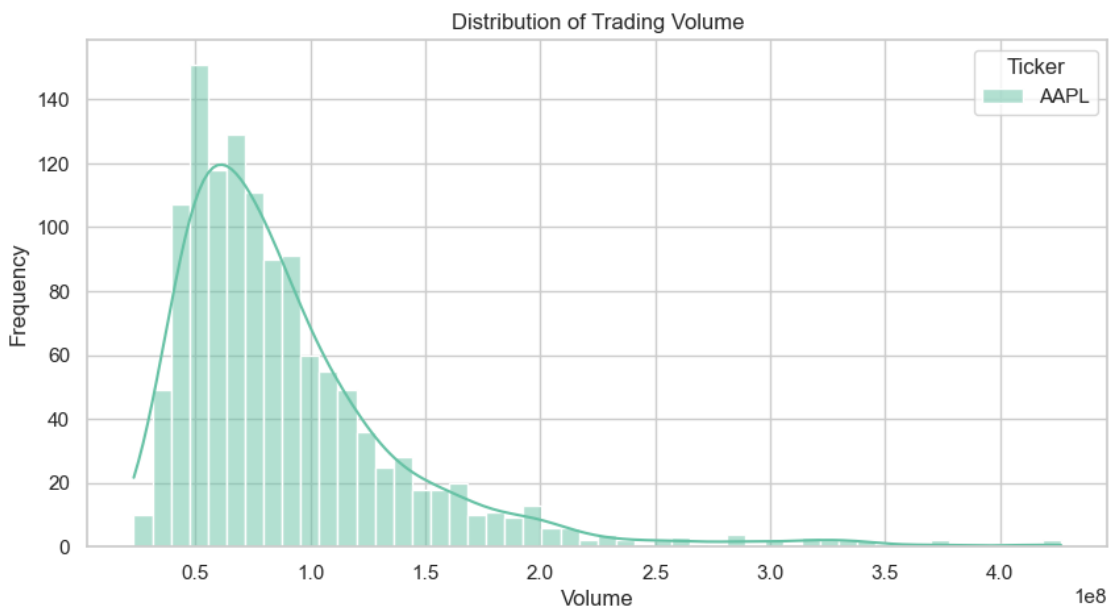


Box plot of price distribution

```
[59]: plt.figure(figsize=(8, 5))
sns.boxplot(data=df[['Open', 'High', 'Low', 'Close']], palette='Set2')
plt.title('Boxplot of AAPL Price Features')
plt.ylabel('Price (USD)')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.ylim(50, 270)  #              ,
plt.show()
```



[60]:
```
plt.figure(figsize=(10, 5))
sns.histplot(df['Volume'], bins=50, kde=True)
plt.title('Distribution of Trading Volume')
plt.xlabel('Volume')
plt.ylabel('Frequency')
plt.show()
```

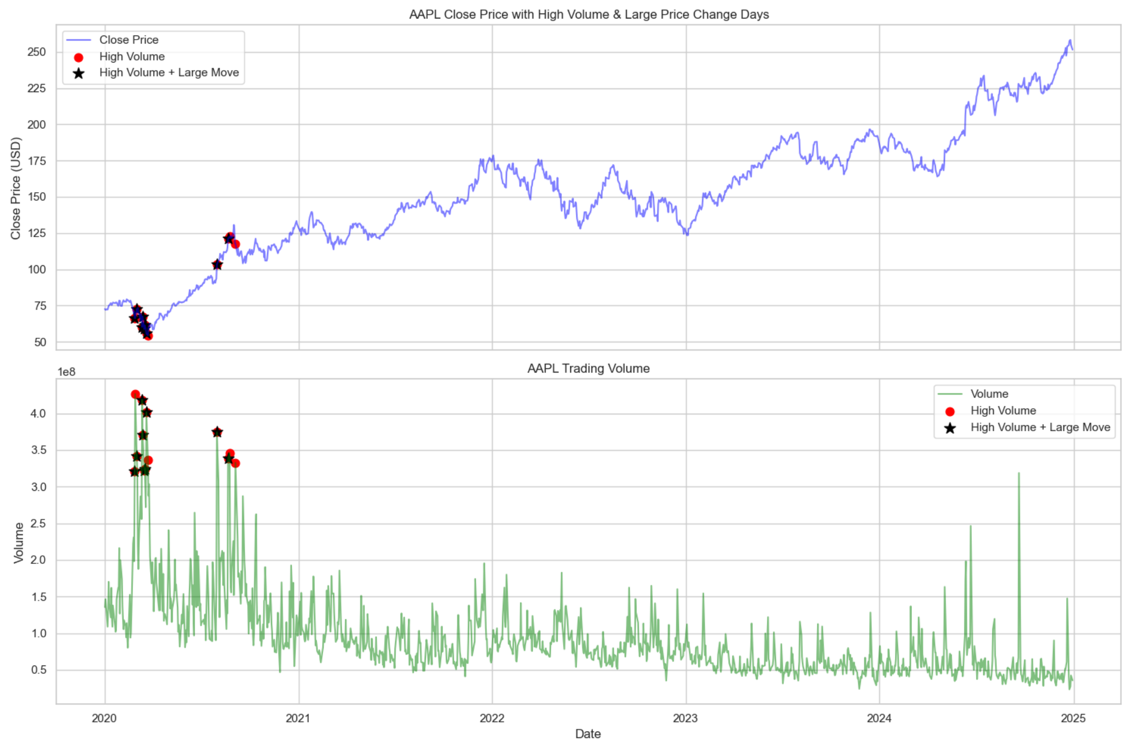


[61]:
```
threshold = df['Volume'].quantile(0.99)  #   1%-
high_volume_days = df[df['Volume'] > threshold]
print("Number of high-volume days:", len(high_volume_days))
```

```
Number of high-volume days: 1257
```

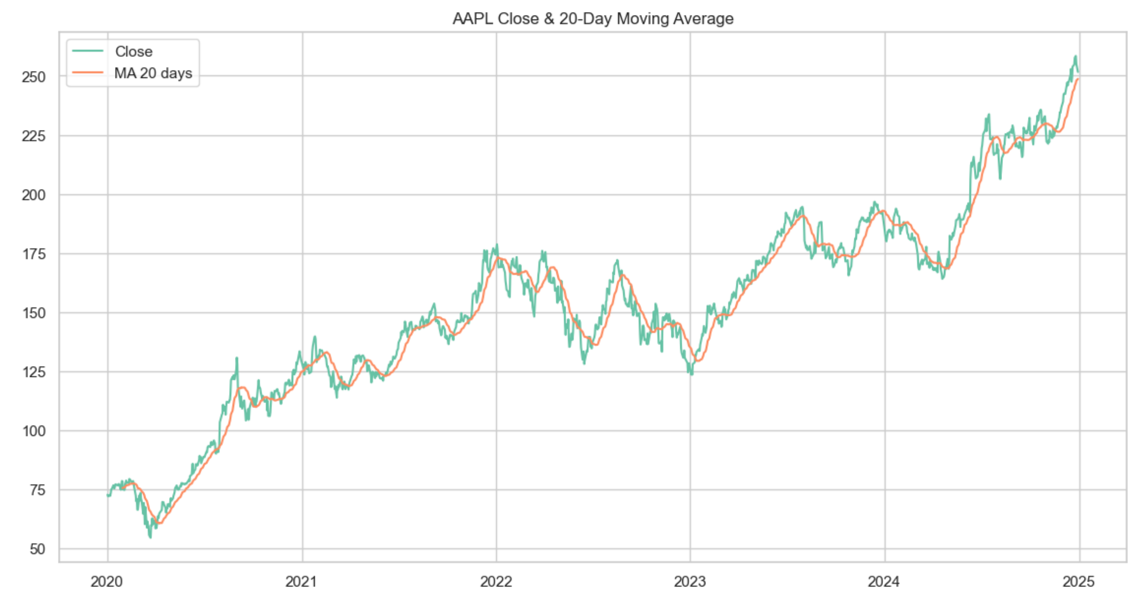
[62]:
```
# flatten col names
df.columns = ['_'.join(col) if isinstance(col, tuple) else col for col in df.
 ↪columns]

df['Daily_Return'] = df['Close_AAPL'].pct_change()

# unusuals volume days
volume_threshold = df['Volume_AAPL'].quantile(0.99)
ret_std = df['Daily_Return'].std()
ret_mean = df['Daily_Return'].mean()


high_volume_days = df[df['Volume_AAPL'] > volume_threshold]
```

```
large_move_days = df[abs(df['Daily_Return'] - ret_mean) > 2 * ret_std]

special_days = df[(df['Volume_AAPL'] > volume_threshold) &␣
 ↪(abs(df['Daily_Return'] - ret_mean) > 2 * ret_std)]

# plotting
fig, axes = plt.subplots(2, 1, figsize=(15, 10), sharex=True)

axes[0].plot(df.index, df['Close_AAPL'], label='Close Price', color='blue',␣
 ↪alpha=0.5)
axes[0].scatter(high_volume_days.index, high_volume_days['Close_AAPL'],␣
 ↪color='red', label='High Volume', marker='o', s=60)
axes[0].scatter(special_days.index, special_days['Close_AAPL'], color='black',␣
 ↪label='High Volume + Large Move', marker='*', s=120)
axes[0].set_ylabel('Close Price (USD)')
axes[0].set_title('AAPL Close Price with High Volume & Large Price Change Days')
axes[0].legend()
axes[0].grid(True)

axes[1].plot(df.index, df['Volume_AAPL'], label='Volume', color='green',␣
 ↪alpha=0.5)
axes[1].scatter(high_volume_days.index, high_volume_days['Volume_AAPL'],␣
 ↪color='red', label='High Volume', marker='o', s=60)
axes[1].scatter(special_days.index, special_days['Volume_AAPL'], color='black',␣
 ↪label='High Volume + Large Move', marker='*', s=120)
axes[1].set_ylabel('Volume')
axes[1].set_xlabel('Date')
axes[1].set_title('AAPL Trading Volume')
axes[1].legend()
axes[1].grid(True)

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



rolling mean

[63]:
```
df['MA20'] = df[('Close_AAPL')].rolling(window=20).mean()
plt.figure(figsize=(14,7))
plt.plot(df.index, df[('Close_AAPL')], label='Close')
plt.plot(df.index, df['MA20'], label='MA 20 days')
plt.legend()
plt.title('AAPL Close & 20-Day Moving Average')
plt.show()
```



1.0.6 2. Data Preprocessing

1.1 Inflation Adjustment: Converting Stock Prices to Current Dollars

To enable accurate comparison of AAPL price data over multiple years and eliminate the effect of inflation, all price columns are adjusted to current U.S. dollars using the Consumer Price Index (CPI) from the Federal Reserve Economic Data (FRED). This adjustment ensures that long-term trends reflect real (inflation-adjusted) price changes, not just nominal increases.

[64]:
```
# TODO
```

seasonal decomposition

[65]:
```
df['Year'] = df.index.year
df['DayOfWeek'] = df.index.dayofweek
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
df['DayOfWeek'] = df['DayOfWeek'].map(dict(enumerate(day_names)))
df['DayOfWeek'] = pd.Categorical(df['DayOfWeek'],␣
 ↪categories=day_names,ordered=True)
print(df['DayOfWeek'].unique())
df['Month'] = df.index.month
month_names = list(calendar.month_name)[1:] # ['January', ..., 'December']
df['Month'] = df['Month'].map(dict(zip(range(1, 13), month_names)))
df['Month'] = pd.Categorical(df['Month'], categories=month_names, ordered=True)
print(df['Month'].unique())
```

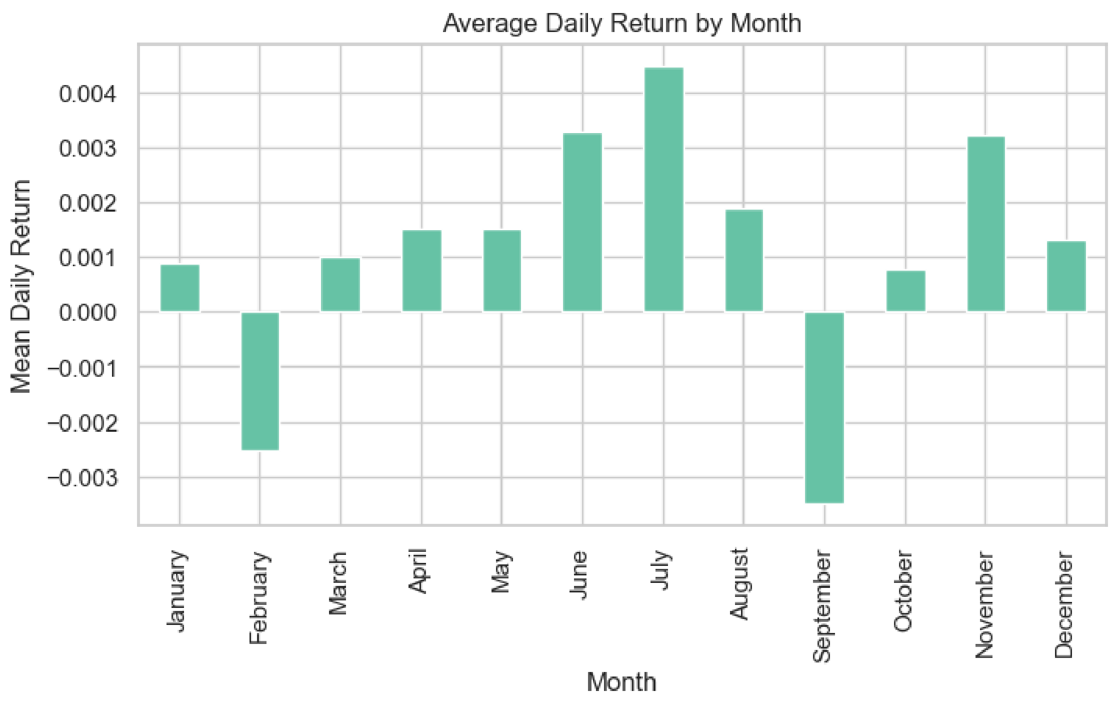
```
['Thursday', 'Friday', 'Monday', 'Tuesday', 'Wednesday']
Categories (5, object): ['Monday' < 'Tuesday' < 'Wednesday' < 'Thursday' <
'Friday']
```

```
['January', 'February', 'March', 'April', 'May', …, 'August', 'September',
'October', 'November', 'December']
Length: 12
Categories (12, object): ['January' < 'February' < 'March' < 'April' …
'September' < 'October' < 'November' < 'December']
```

seasonality by month

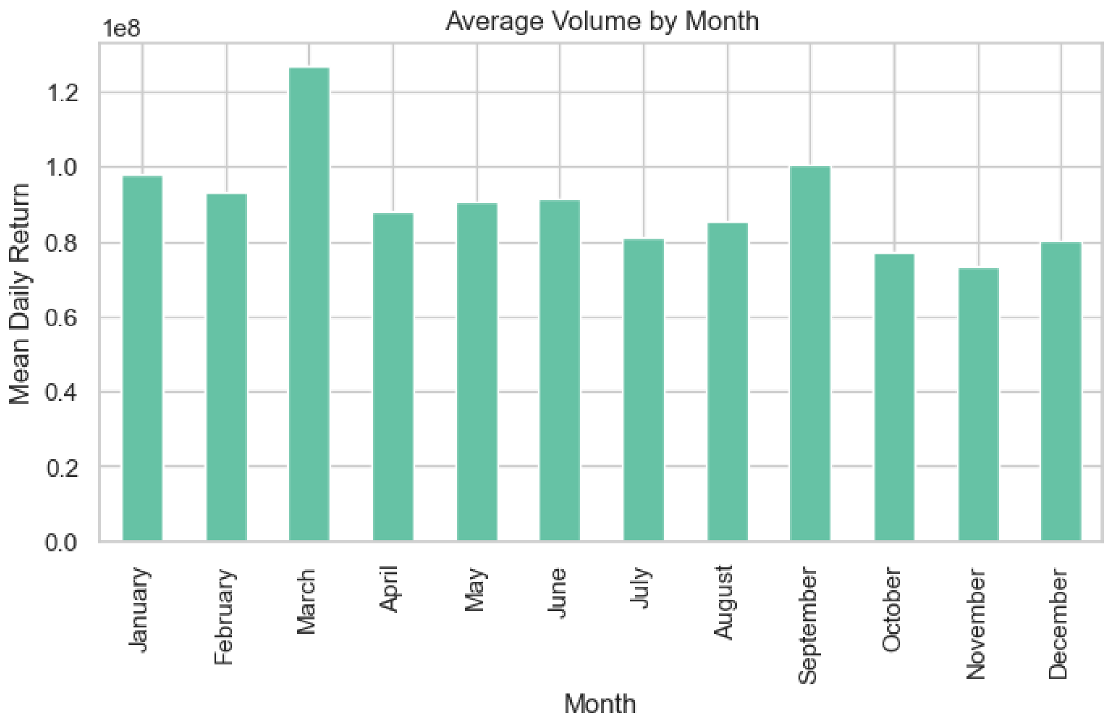
[66]:
```
monthly_avg = df.groupby('Month')['Daily_Return'].mean()

plt.figure(figsize=(8, 4))
monthly_avg.plot(kind='bar')
plt.title('Average Daily Return by Month')
plt.ylabel('Mean Daily Return')
plt.xlabel('Month')
plt.grid(True)
plt.show()
```

```
/var/folders/x0/x2vffgpn4mv88fvwb47hvlmc0000gn/T/ipykernel_61651/4200205056.py:1
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  monthly_avg = df.groupby('Month')['Daily_Return'].mean()
```



[67]:
```
monthly_avg = df.groupby('Month')['Volume_AAPL'].mean()

plt.figure(figsize=(8, 4))
monthly_avg.plot(kind='bar')
plt.title('Average Volume by Month')
plt.ylabel('Mean Daily Return')
plt.xlabel('Month')
plt.grid(True)
plt.show()
```

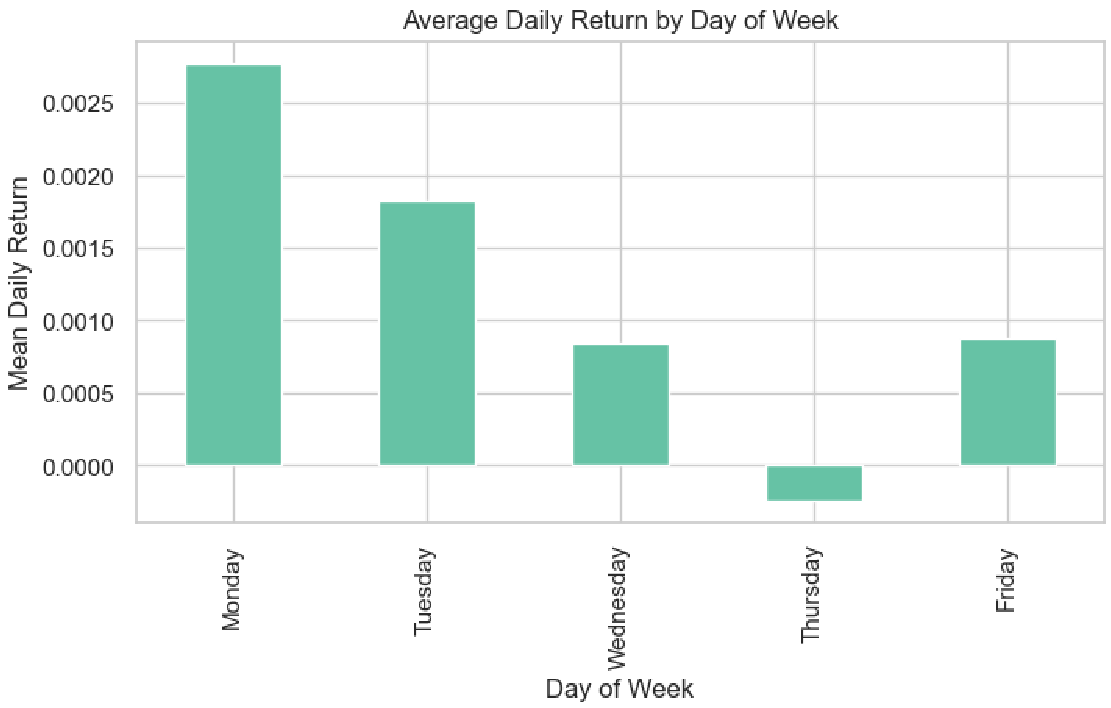
```
/var/folders/x0/x2vffgpn4mv88fvwb47hvlmc0000gn/T/ipykernel_61651/243346374.py:1:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  monthly_avg = df.groupby('Month')['Volume_AAPL'].mean()
```



seasonality by day of week

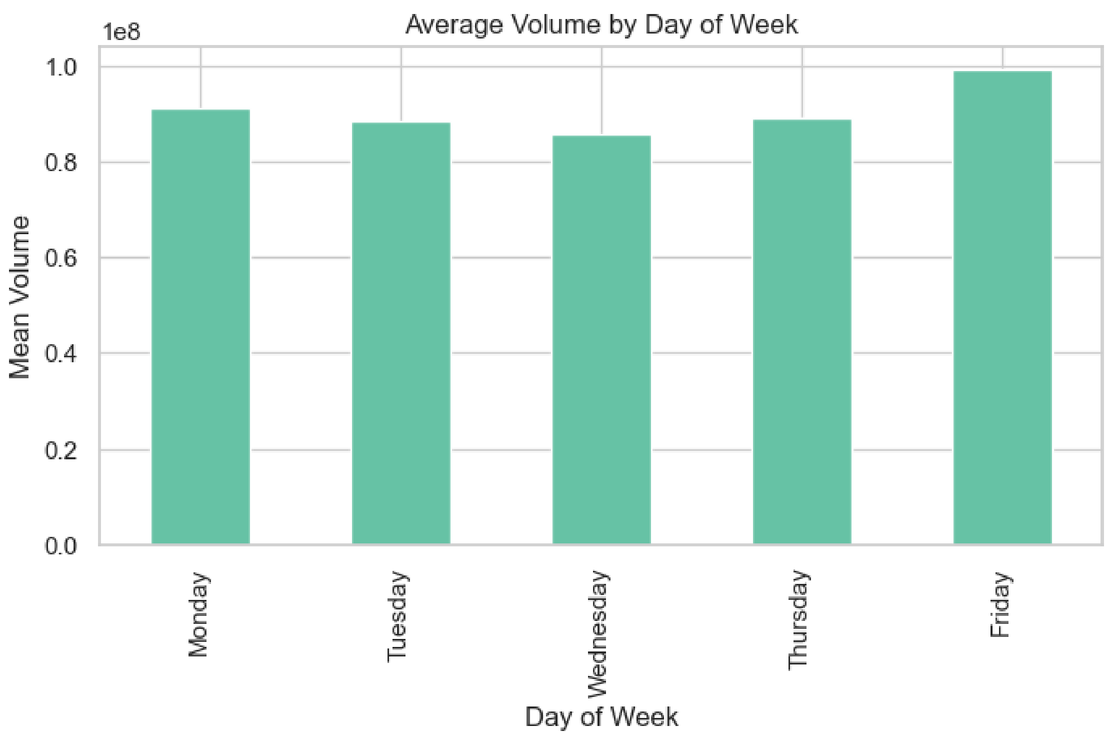
[68]:
```
dow_avg = df.groupby('DayOfWeek')['Daily_Return'].mean().reindex(day_names)
plt.figure(figsize=(8, 4))
dow_avg.plot(kind='bar')
plt.title('Average Daily Return by Day of Week')
plt.ylabel('Mean Daily Return')
plt.xlabel('Day of Week')
plt.grid(True)
plt.show()
```

```
/var/folders/x0/x2vffgpn4mv88fvwb47hvlmc0000gn/T/ipykernel_61651/1238027972.py:1
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  dow_avg = df.groupby('DayOfWeek')['Daily_Return'].mean().reindex(day_names)
```



```
[69]: dow_avg = df.groupby('DayOfWeek')['Volume_AAPL'].mean().reindex(day_names)
plt.figure(figsize=(8, 4))
dow_avg.plot(kind='bar')
plt.title('Average Volume by Day of Week')
plt.ylabel('Mean Volume')
plt.xlabel('Day of Week')
plt.grid(True)
plt.show()
```

```
/var/folders/x0/x2vffgpn4mv88fvwb47hvlmc0000gn/T/ipykernel_61651/1877633182.py:1
: FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  dow_avg = df.groupby('DayOfWeek')['Volume_AAPL'].mean().reindex(day_names)
```



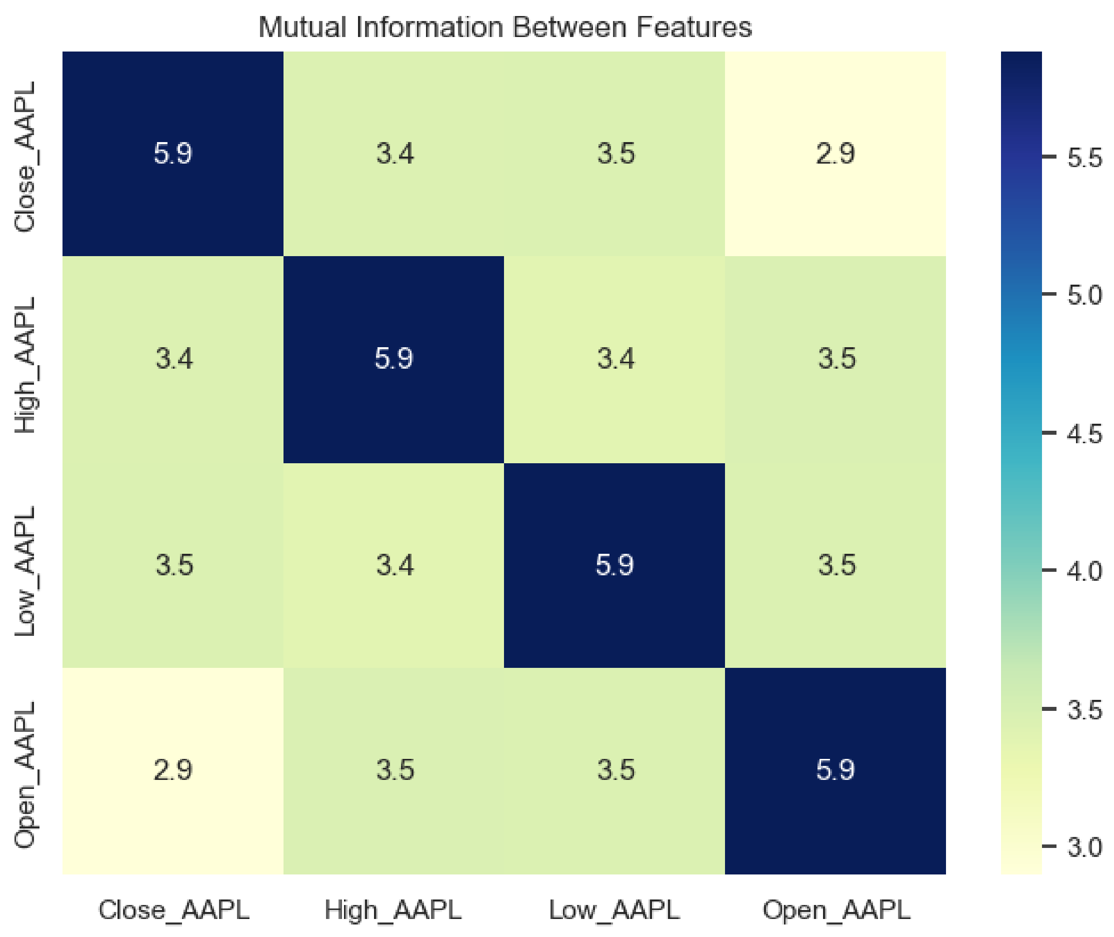
Mutual information

```
[70]: mi_matrix = pd.DataFrame(index= features_for_sns, columns=features_for_sns)

      X = df_for_sns[features_for_sns]

      for f1 in features_for_sns:
          for f2 in features_for_sns:
              mi_score = mutual_info_regression(X[[f1]], X[[f2]])
              mi_matrix.loc[f1, f2] = mi_score[0]

      mi_matrix = mi_matrix.astype(float)

      # plot
      plt.figure(figsize=(8, 6))
      sns.heatmap(mi_matrix, annot=True, cmap='YlGnBu')
      plt.title('Mutual Information Between Features')
      plt.show()
```

```
/Users/shakedschnarch/Documents/       /    /           /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

```
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

```
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/Users/shakedschnarch/Documents/      /   /          /finyx-
test/lib/python3.11/site-packages/sklearn/utils/validation.py:1406:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```



1.1.1 3. Feature Engineering

— Feature Engineering Template —

1. Feature name: Daily_Return

`Description: predicts the percentage change in stock price from one day to the next, which can`

[71]:
```
# 1. Feature name: Daily_Return
df['Daily_Return'] = df['Close_AAPL'].pct_change()

# 2. Handle missing values created by rolling/shift (if any)
df['Daily_Return'] = df['Close_AAPL'].fillna(0)

# 3.Check that feature is ready for modeling
print(df['Daily_Return'].describe())
print("Any missing?", df['Daily_Return'].isnull().sum())
```

```
count    1257.000000
```

```
mean      151.900499
std        41.943932
min        54.378582
25%       126.750748
50%       150.537064
75%       176.443726
max       258.396667
Name: Daily_Return, dtype: float64
Any missing? 0
```

2. Feature name: High_Low_Percent

Description: measures the volatility of the stock by calculating the percentage difference betw

[72]:
```
# 1. Featurw name: High_Low_Pct
df['High_Low_Pct'] = (df['High_AAPL'] - df['Low_AAPL']) / df['Open_AAPL']

# 2. Handle missing values created by rolling/shift (if any)
df['High_Low_Pct'] = df['High_Low_Pct'].fillna(0)

# 3.Check that feature is ready for modeling
print(df['High_Low_Pct'].describe())
print("Any missing?", df['High_Low_Pct'].isnull().sum())
```

```
count    1257.000000
mean        0.022457
std         0.012566
min         0.005404
25%         0.014274
50%         0.019061
75%         0.027472
max         0.106688
Name: High_Low_Pct, dtype: float64
Any missing? 0
```

3. Feature name: Moving_Average

Description: calculates the average stock price over a specified number of days, which can smoo

[73]:
```
# Feature name: Moving_Average
df['MA20'] = df['Close_AAPL'].rolling(window=20).mean()

# 2. Handle missing values created by rolling/shift (if any)
df['MA20'] = df['MA20'].combine_first(df['Close_AAPL'].expanding().mean())

# 3. Check that feature is ready for modeling
print(df['MA20'].describe())
print("Any missing?", df['MA20'].isnull().sum())
```

```
count    1257.000000
mean      150.552194
std        41.312233
min        60.459788
25%       125.801204
50%       148.815122
75%       175.484521
max       248.668144
Name: MA20, dtype: float64
Any missing? 0
```

4. Feature name: Relative_Volume

Description: compares the current trading volume to the average volume over a specified period

[74]:
```python
# Feature name: Volume_Change
df['Rel_Volume'] = df['Volume_AAPL'] / df['Volume_AAPL'].rolling(20).mean()

# 2. Handle missing values created by rolling/shift (if any)
df['Rel_Volume'] = df['Rel_Volume'].combine_first(df['Volume_AAPL'].expanding().
 ↪mean())

# 3. Check that feature is ready for modeling
print(df['Rel_Volume'].describe())
print("Any missing?", df['Rel_Volume'].isnull().sum())
```

```
count    1.257000e+03
mean     2.013405e+06
std      1.626455e+07
min      4.085035e-01
25%      7.936592e-01
50%      9.244703e-01
75%      1.138437e+00
max      1.409016e+08
Name: Rel_Volume, dtype: float64
Any missing? 0
```

5. Feature name: Quarter_Number

This feature encodes the business quarter of each date (1=Q1, 2=Q2, 3=Q3, 4=Q4), regardless of

[75]:
```python
# Ensure Month_Num is numeric month (1-12)
df['Month_Num'] = df.index.month

# Calculate Quarter_Number (1-4)
df['Quarter_Number'] = ((df['Month_Num'] - 1) // 3 + 1).astype(int)

# Check the feature
print(df['Quarter_Number'].describe())
print("Any missing?", df['Quarter_Number'].isnull().sum())
print("Unique values:", df['Quarter_Number'].unique())
```

```
count    1257.000000
mean        2.513126
std         1.115907
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         4.000000
Name: Quarter_Number, dtype: float64
Any missing? 0
Unique values: [1 2 3 4]
```

6. Feature name: Days_To_Quarter_End

Description: calculates the number of days remaining until the end of the current business qua

[76]:
```
# Feature name: Days_To_Quarter_End
# Description: Counts the number of days from each date to the end of the␣
 ↪business quarter (useful for modeling behavior around earnings reports, etc.)

# 1. Create feature: Days to end of quarter
quarter_end = df.index.to_period('Q').end_time
df['Days_To_Quarter_End'] = (quarter_end - df.index).days

# 2. Handle any missing values (should not happen, but just in case)
df['Days_To_Quarter_End'] = df['Days_To_Quarter_End'].fillna(-1).astype(int)

# 3. Check that feature is ready for modeling
print(df['Days_To_Quarter_End'].describe())
print("Any missing?", df['Days_To_Quarter_End'].isnull().sum())
```

```
count    1257.000000
mean       44.984089
std        26.207642
min         0.000000
25%        22.000000
50%        45.000000
75%        67.000000
max        91.000000
Name: Days_To_Quarter_End, dtype: float64
Any missing? 0
```

1.1.2 One Hot Encoding for Categorical Features

[77]:
```python
# List of categorical features for one-hot encoding
categorical_features = ['DayOfWeek', 'Month', 'Quarter_Number']

# Apply one-hot encoding using pandas get_dummies
df = pd.get_dummies(df, columns=categorical_features, drop_first=True)

# Check the new columns
print("Columns after one-hot encoding:")
print(df.columns)
```

```
Columns after one-hot encoding:
Index(['Close_AAPL', 'High_AAPL', 'Low_AAPL', 'Open_AAPL', 'Volume_AAPL',
       'Daily_Return_', 'Daily_Return', 'MA20', 'Year', 'High_Low_Pct',
       'Rel_Volume', 'Month_Num', 'Days_To_Quarter_End', 'DayOfWeek_Tuesday',
       'DayOfWeek_Wednesday', 'DayOfWeek_Thursday', 'DayOfWeek_Friday',
       'Month_February', 'Month_March', 'Month_April', 'Month_May',
       'Month_June', 'Month_July', 'Month_August', 'Month_September',
       'Month_October', 'Month_November', 'Month_December', 'Quarter_Number_2',
       'Quarter_Number_3', 'Quarter_Number_4'],
      dtype='object')
```

1.1.3 Checking for missing values

[91]:
```python
print(df.isnull().sum())
# Optionally, fill NA (example: mean imputation for numeric columns)
df = df.fillna(df.mean(numeric_only=True))
```

```
Close_AAPL             0
High_AAPL              0
Low_AAPL               0
Open_AAPL              0
Volume_AAPL            0
Daily_Return_          0
Daily_Return           0
MA20                   0
Year                   0
High_Low_Pct           0
Rel_Volume             0
Month_Num              0
Days_To_Quarter_End    0
DayOfWeek_Tuesday      0
DayOfWeek_Wednesday    0
DayOfWeek_Thursday     0
DayOfWeek_Friday       0
Month_February         0
Month_March            0
Month_April            0
```

```
Month_May              0
Month_June             0
Month_July             0
Month_August           0
Month_September        0
Month_October          0
Month_November         0
Month_December         0
Quarter_Number_2       0
Quarter_Number_3       0
Quarter_Number_4       0
dtype: int64
```

1.1.4 Outliers detection and removal

[92]:
```
# Clip extreme values in 'Daily_Return' to the 1st and 99th percentiles
df['Daily_Return'] = df['Daily_Return'].clip(lower=df['Daily_Return'].
  ↪quantile(0.01),
                                             upper=df['Daily_Return'].
  ↪quantile(0.99))
```

1.1.5 Numerical Features Scaling

[93]:
```
numeric_features = ['Open_AAPL', 'High_AAPL', 'Low_AAPL', 'Close_AAPL',␣
  ↪'Volume_AAPL', 'High_Low_Pct', 'Rel_Volume']
scaler = StandardScaler()
df[numeric_features] = scaler.fit_transform(df[numeric_features])
```

Drop unnecessary columns

[ ]:
```
# --- Choose features to keep ---
features_to_keep = [
    'High_Low_Pct', 'MA20', 'Rel_Volume', 'Days_To_Quarter_End', 'Close_AAPL'
    # Add more as needed
] + [col for col in df.columns if col.startswith('DayOfWeek_')
                              or col.startswith('Month_')
                              or col.startswith('Quarter_Number_')
                              or col == 'Month_Num'
]

# (Optional) Print to verify which features exist
print("All columns in df:")
print(df.columns)
print()

# Only keep features that actually exist
existing_features = [f for f in features_to_keep if f in df.columns]
missing = [f for f in features_to_keep if f not in df.columns]
```

```
print("Missing features:", missing)

# Create your final X
X = df[existing_features].copy()

print("Selected features:", X.columns.tolist())
print("Shape:", X.shape)
```

```
All columns in df:
Index(['Close_AAPL', 'High_AAPL', 'Low_AAPL', 'Open_AAPL', 'Volume_AAPL',
       'Daily_Return_', 'Daily_Return', 'MA20', 'Year', 'High_Low_Pct',
       'Rel_Volume', 'Month_Num', 'Days_To_Quarter_End', 'DayOfWeek_Tuesday',
       'DayOfWeek_Wednesday', 'DayOfWeek_Thursday', 'DayOfWeek_Friday',
       'Month_February', 'Month_March', 'Month_April', 'Month_May',
       'Month_June', 'Month_July', 'Month_August', 'Month_September',
       'Month_October', 'Month_November', 'Month_December', 'Quarter_Number_2',
       'Quarter_Number_3', 'Quarter_Number_4'],
      dtype='object')

Missing features: []
Selected features: ['High_Low_Pct', 'MA20', 'Rel_Volume', 'Days_To_Quarter_End',
'Close_AAPL', 'Month_Num', 'DayOfWeek_Tuesday', 'DayOfWeek_Wednesday',
'DayOfWeek_Thursday', 'DayOfWeek_Friday', 'Month_February', 'Month_March',
'Month_April', 'Month_May', 'Month_June', 'Month_July', 'Month_August',
'Month_September', 'Month_October', 'Month_November', 'Month_December',
'Quarter_Number_2', 'Quarter_Number_3', 'Quarter_Number_4']
Shape: (1257, 24)
```

1.1.6 Features and Target Variable Selection

[101]:
```
print("Features shape:", X.shape)
print("Features columns:", X.columns)


y = df['Daily_Return']
print("Target shape:", df['Daily_Return'].shape)
```

```
Features shape: (1257, 24)
Features columns: Index(['High_Low_Pct', 'MA20', 'Rel_Volume',
'Days_To_Quarter_End',
       'Close_AAPL', 'Month_Num', 'DayOfWeek_Tuesday', 'DayOfWeek_Wednesday',
       'DayOfWeek_Thursday', 'DayOfWeek_Friday', 'Month_February',
       'Month_March', 'Month_April', 'Month_May', 'Month_June', 'Month_July',
       'Month_August', 'Month_September', 'Month_October', 'Month_November',
       'Month_December', 'Quarter_Number_2', 'Quarter_Number_3',
       'Quarter_Number_4'],
      dtype='object')
Target shape: (1257,)
```

1.1.7 4. Train/Test Split

[102]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=False  # shuffle=False␣
 ↪preserves time order for time series!
)
```

Indexing the DataFrame

[103]:
```
X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

Dimention checking

[104]:
```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(1005, 24) (252, 24) (1005,) (252,)
```

1.1.8 5. Model Training

1.2 Linear Regression Model

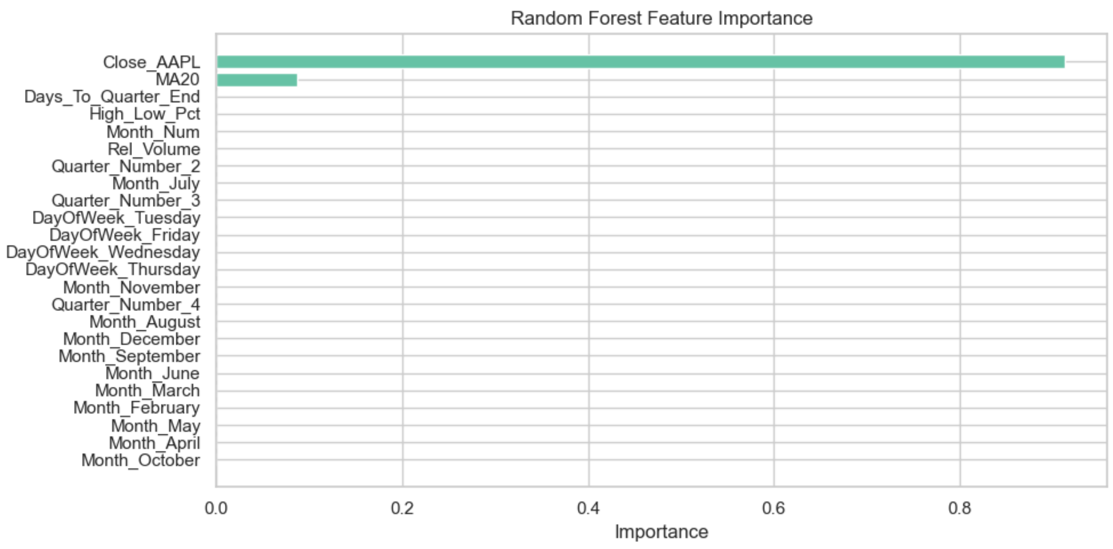
[107]:
```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# 1. Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# 2. Random Forest
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

# 3. XGBoost Regressor
xgb = XGBRegressor(n_estimators=100, random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)

# 4. Evaluation function
def print_scores(y_true, y_pred, model_name):
    print(f"\nModel: {model_name}")
    print(f"MAE: {mean_absolute_error(y_true, y_pred):.4f}")
    print(f"MSE: {mean_squared_error(y_true, y_pred):.4f}")
    print(f"R^2: {r2_score(y_true, y_pred):.4f}")
```

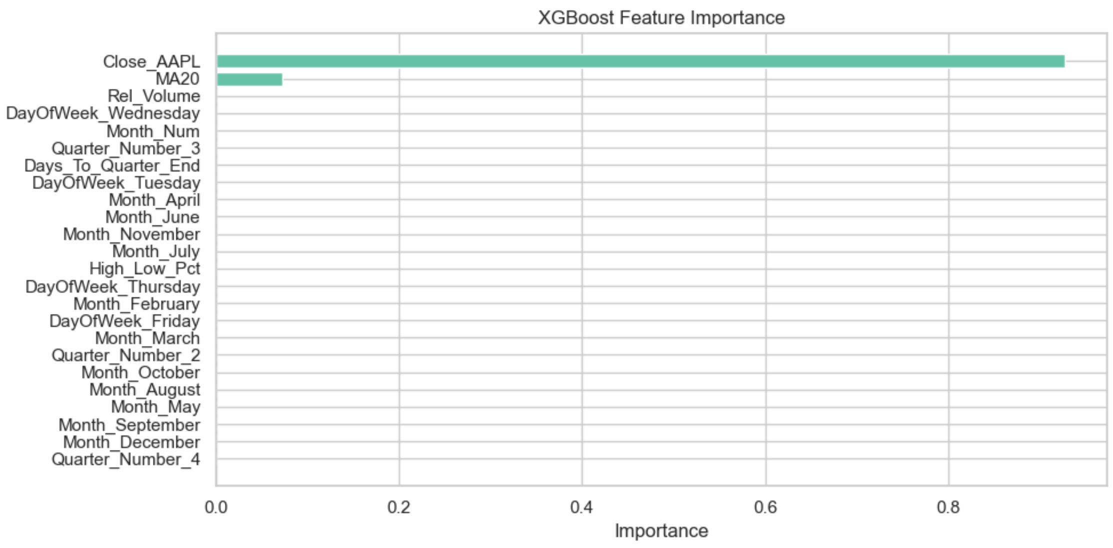
```
# 5. Print scores for all models
print_scores(y_test, y_pred_lr, "Linear Regression")
print_scores(y_test, y_pred_rf, "Random Forest")
print_scores(y_test, y_pred_xgb, "XGBoost")
```

```
Model: Linear Regression
MAE: 0.4083
MSE: 2.1577
R^2: 0.9965

Model: Random Forest
MAE: 17.0015
MSE: 574.6729
R^2: 0.0789

Model: XGBoost
MAE: 17.4977
MSE: 596.9373
R^2: 0.0433
```

1.2.1 Feature Importance

[108]:
```
import matplotlib.pyplot as plt
import numpy as np

# --- Random Forest Feature Importance ---
plt.figure(figsize=(10, 5))
indices = np.argsort(rf.feature_importances_)
plt.barh(range(len(indices)), rf.feature_importances_[indices], align="center")
plt.yticks(range(len(indices)), [X_train.columns[i] for i in indices])
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()

# --- XGBoost Feature Importance ---
plt.figure(figsize=(10, 5))
indices = np.argsort(xgb.feature_importances_)
plt.barh(range(len(indices)), xgb.feature_importances_[indices], align="center")
plt.yticks(range(len(indices)), [X_train.columns[i] for i in indices])
plt.title("XGBoost Feature Importance")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```

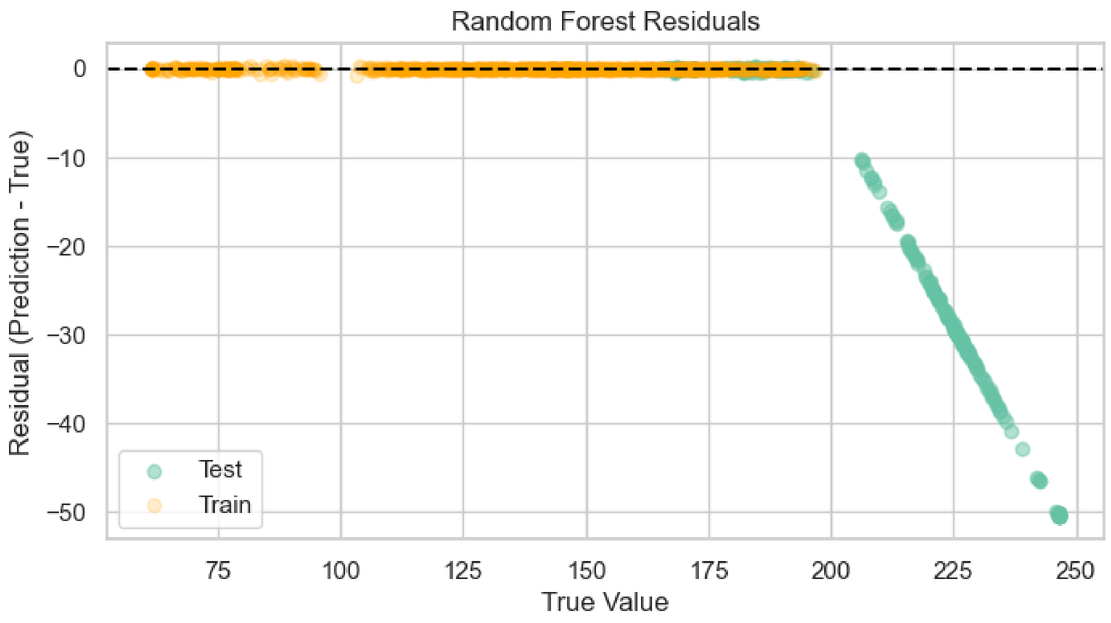




1.2.2 Residual Analysis

[109]:
```
# Choose a model, e.g., Random Forest
y_pred_train_rf = rf.predict(X_train)
y_pred_test_rf = rf.predict(X_test)

plt.figure(figsize=(8,4))
plt.scatter(y_test, y_pred_rf - y_test, alpha=0.5, label='Test')
plt.scatter(y_train, y_pred_train_rf - y_train, alpha=0.2, label='Train',␣
 ↪color='orange')
```

```
plt.axhline(0, linestyle='--', color='black')
plt.xlabel('True Value')
plt.ylabel('Residual (Prediction - True)')
plt.title('Random Forest Residuals')
plt.legend()
plt.show()
```



1.2.3 Overfitting/Underfitting check

[110]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def print_scores(y_true, y_pred, set_name=""):
    print(f"{set_name} MAE:  {mean_absolute_error(y_true, y_pred):.4f}")
    print(f"{set_name} MSE:  {mean_squared_error(y_true, y_pred):.4f}")
    print(f"{set_name} R2:   {r2_score(y_true, y_pred):.4f}")
    print("")

print("Random Forest - Train:")
print_scores(y_train, y_pred_train_rf, set_name="Train")
print("Random Forest - Test:")
print_scores(y_test, y_pred_rf, set_name="Test")
```

```
Random Forest - Train:
Train MAE:  0.0361
Train MSE:  0.0046
Train R2:   1.0000
```

```
Random Forest - Test:
Test MAE:  17.0015
Test MSE:  574.6729
Test R2:   0.0789
```

1.2.4 cross-validation

[111]:
```python
from sklearn.model_selection import TimeSeriesSplit, cross_val_score

tscv = TimeSeriesSplit(n_splits=5)
rf_scores = cross_val_score(rf, X, y, cv=tscv,␣
 ↪scoring='neg_mean_absolute_error')
print("Random Forest TimeSeries CV MAE (mean):", -np.mean(rf_scores))
```

```
Random Forest TimeSeries CV MAE (mean): 8.541497271886215
```

2 Another Model Training

2.1 Model 2

2.2 Model 2

[ ]:
```python
# Exclude Close_AAPL and re-run
features_no_close = [col for col in X_train.columns if col != 'Close_AAPL']

print("Final features used for modeling:")
print(X.columns.tolist())
print("Shape of X:", X.shape)
```

```
Final features used for modeling:
['High_Low_Pct', 'MA20', 'Rel_Volume', 'Days_To_Quarter_End', 'Close_AAPL',
'Month_Num', 'DayOfWeek_Tuesday', 'DayOfWeek_Wednesday', 'DayOfWeek_Thursday',
'DayOfWeek_Friday', 'Month_February', 'Month_March', 'Month_April', 'Month_May',
'Month_June', 'Month_July', 'Month_August', 'Month_September', 'Month_October',
'Month_November', 'Month_December', 'Quarter_Number_2', 'Quarter_Number_3',
'Quarter_Number_4']
Shape of X: (1257, 24)
```

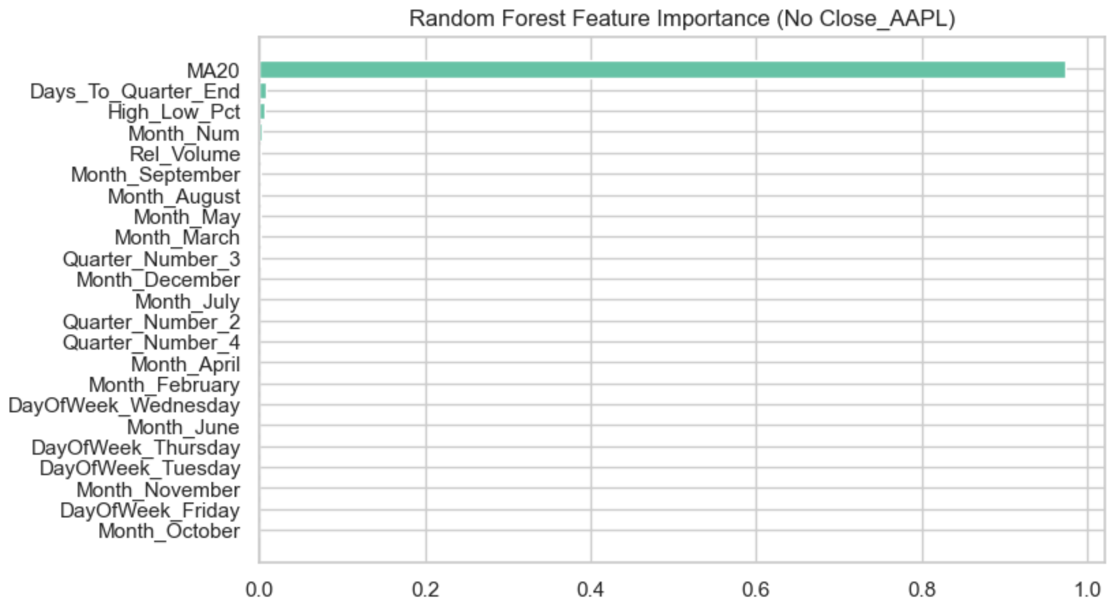
retrain

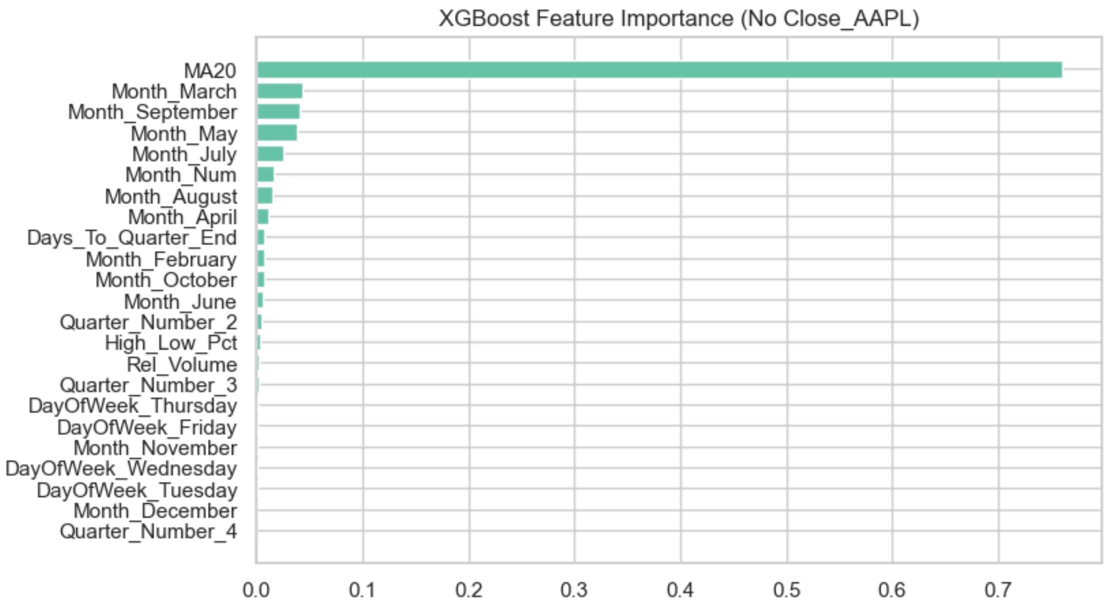
[ ]:
```python
# --- Random Forest without Close_AAPL ---
rf_no_close = RandomForestRegressor(n_estimators=100, random_state=42)
rf_no_close.fit(X_train[features_no_close], y_train)
y_pred_rf_nc = rf_no_close.predict(X_test[features_no_close])

# --- XGBoost without Close_AAPL ---
xgb_no_close = XGBRegressor(n_estimators=100, random_state=42)
xgb_no_close.fit(X_train[features_no_close], y_train)
```

```
y_pred_xgb_nc = xgb_no_close.predict(X_test[features_no_close])
```

```
[ ]: # Print scores as
print_scores(y_test, y_pred_rf_nc, "Random Forest (No Close_AAPL)")
print_scores(y_test, y_pred_xgb_nc, "XGBoost (No Close_AAPL)")

# Plot importances (if needed)
def plot_feature_importance(model, feature_names, title):
    importances = model.feature_importances_
    sorted_idx = np.argsort(importances)
    plt.figure(figsize=(8, 5))
    plt.barh(np.array(feature_names)[sorted_idx], importances[sorted_idx])
    plt.title(title)
    plt.show()

plot_feature_importance(rf_no_close, features_no_close, "Random Forest Feature␣
↪Importance (No Close_AAPL)")
plot_feature_importance(xgb_no_close, features_no_close, "XGBoost Feature␣
↪Importance (No Close_AAPL)")
```

```
Random Forest (No Close_AAPL) MAE:  22.4531
Random Forest (No Close_AAPL) MSE:  786.2467
Random Forest (No Close_AAPL) R2:   -0.2602

XGBoost (No Close_AAPL) MAE:  22.4688
XGBoost (No Close_AAPL) MSE:  796.3717
XGBoost (No Close_AAPL) R2:   -0.2764
```





2.2.1 model saving

```
[ ]: import joblib
     joblib.dump(rf, 'random_forest_model.joblib')
```

2.3 model 3

```
[125]: engineered_features = [
           'High_Low_Pct',
           #'MA20',
           'Rel_Volume',
           'Days_To_Quarter_End',
           'Quarter_Number'
           #              '
       ]

       #    DataFrame-       X  ) df)
       #                       engineered_features /             one hot encoding
       selected_columns = []
       for feat in engineered_features:
           selected_columns += [col for col in X.columns if col == feat or col.
        ↪startswith(f"{feat}_")]

       X_filtered = X[selected_columns].copy()
```

```
print("Features used for modeling:", X_filtered.columns.tolist())
print("Shape:", X_filtered.shape)
```

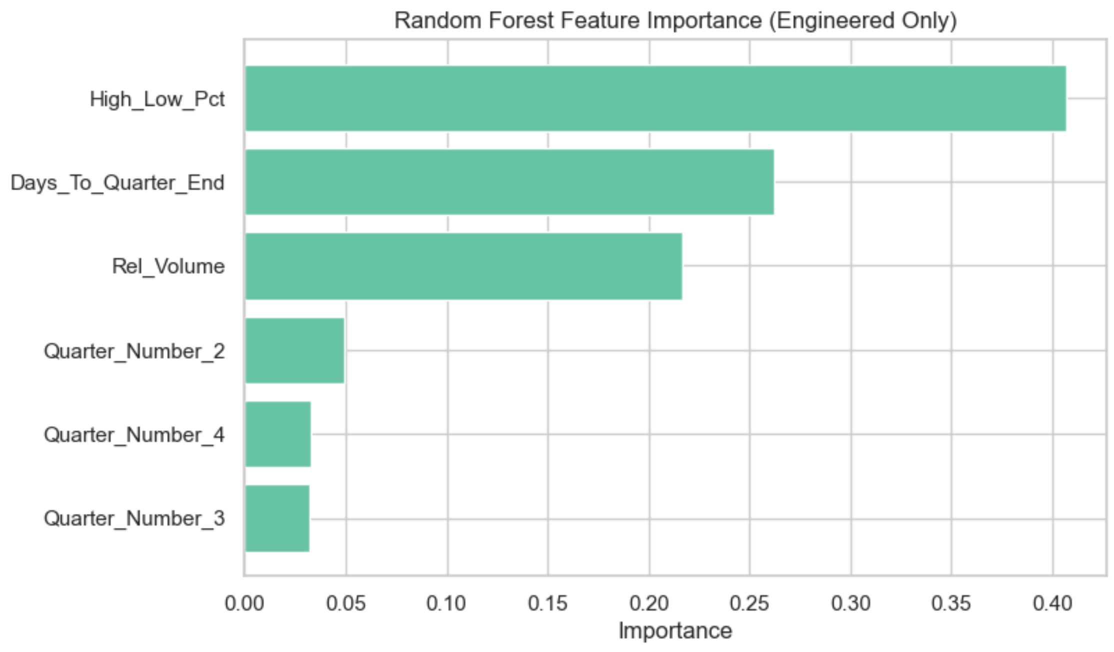
```
Features used for modeling: ['High_Low_Pct', 'Rel_Volume',
'Days_To_Quarter_End', 'Quarter_Number_2', 'Quarter_Number_3',
'Quarter_Number_4']
Shape: (1257, 6)
```

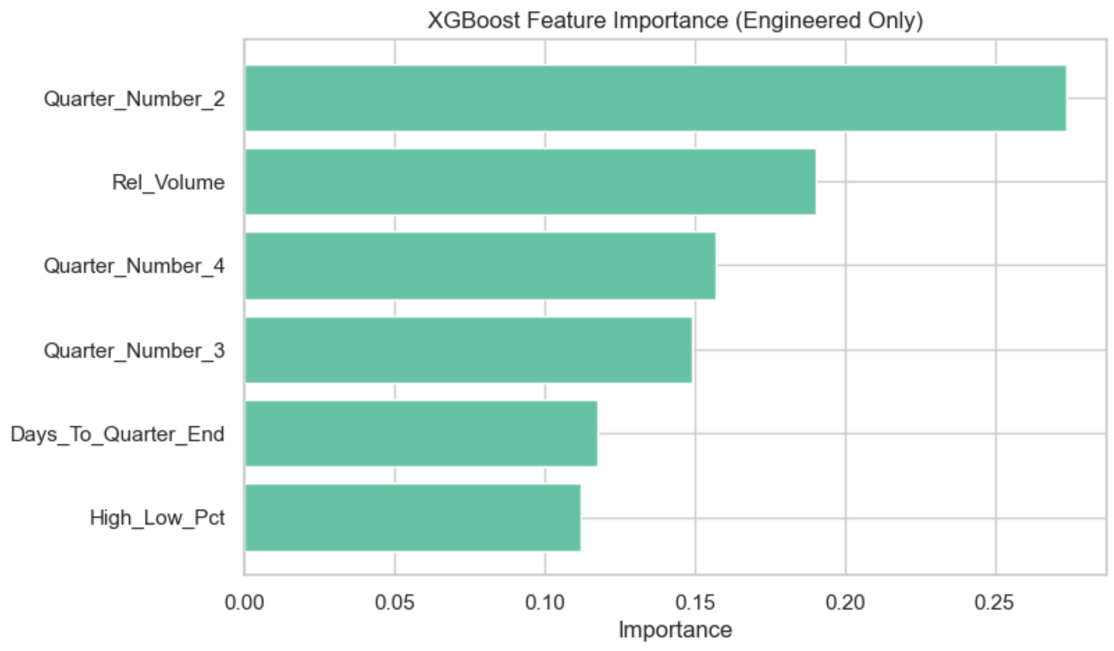
```
[126]: X_train, X_test, y_train, y_test = train_test_split(
           X_filtered, y, test_size=0.2, random_state=42, shuffle=False   #␣
        ↪shuffle=False
       )
```

retrain

```
[127]: # Random Forest
       rf = RandomForestRegressor(n_estimators=100, random_state=42)
       rf.fit(X_train, y_train)
       y_pred_rf = rf.predict(X_test)

       # XGBoost
       import xgboost as xgb
       xgb_model = xgb.XGBRegressor(n_estimators=100, random_state=42)
       xgb_model.fit(X_train, y_train)
       y_pred_xgb = xgb_model.predict(X_test)
```

```
[128]: import matplotlib.pyplot as plt
       import pandas as pd

       def plot_feature_importance(model, feature_names, title):
           importance = model.feature_importances_
           sorted_idx = importance.argsort()
           plt.figure(figsize=(8, 5))
           plt.barh(range(len(feature_names)), importance[sorted_idx])
           plt.yticks(range(len(feature_names)), [feature_names[i] for i in␣
        ↪sorted_idx])
           plt.title(title)
           plt.xlabel("Importance")
           plt.show()

       plot_feature_importance(rf, X_filtered.columns, "Random Forest Feature␣
        ↪Importance (Engineered Only)")
       plot_feature_importance(xgb_model, X_filtered.columns, "XGBoost Feature␣
        ↪Importance (Engineered Only)")
```


Random Forest Feature Importance (Engineered Only)
High_Low_Pct
Days_To_Quarter_End
Rel_Volume
Quarter_Number_2
Quarter_Number_4
Quarter_Number_3
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance


XGBoost Feature Importance (Engineered Only)
Quarter_Number_2
Rel_Volume
Quarter_Number_4
Quarter_Number_3
Days_To_Quarter_End
High_Low_Pct
0.00
0.05
0.10
0.15
0.20
0.25
Importance

```
[ ]:
```

2.3.1 model saving

```
[ ]: import joblib
     joblib.dump(rf, 'random_forest_model.joblib')
```

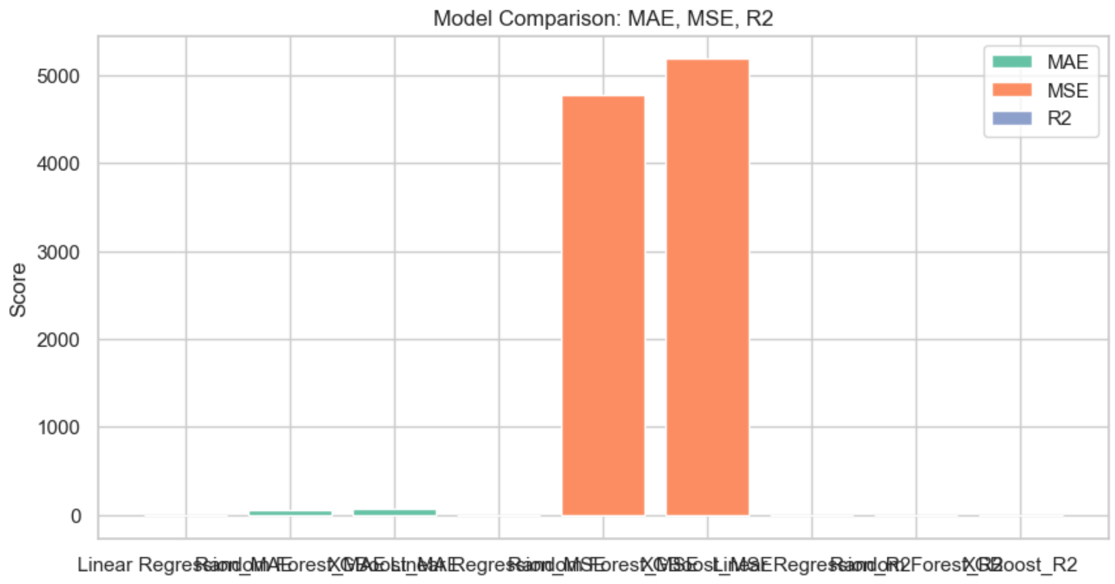
```
[ ]:
```

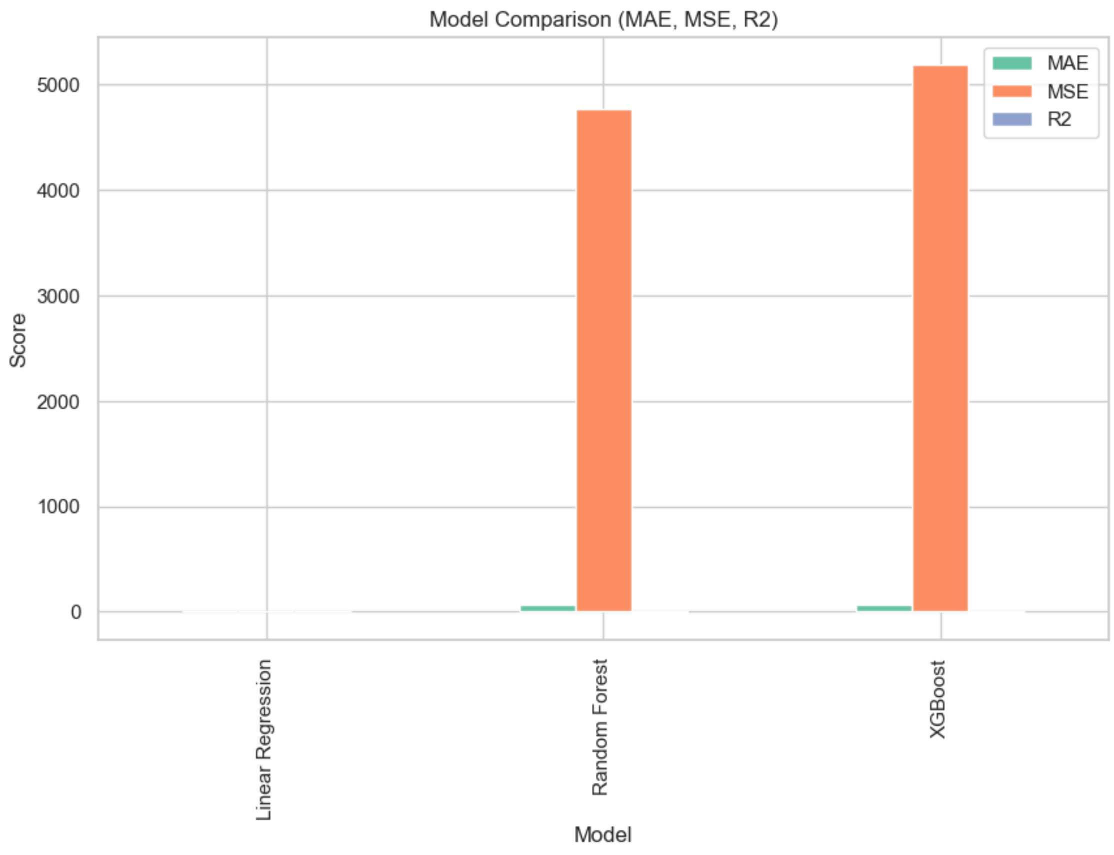
2.4 6. Evaluation

```
[129]: import pandas as pd
       import numpy as np
       from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
       import matplotlib.pyplot as plt

       # Gather predictions from your models:
       results = []

       # Linear Regression
       results.append({
           'Model': 'Linear Regression',
           'MAE': mean_absolute_error(y_test, y_pred_lr),
           'MSE': mean_squared_error(y_test, y_pred_lr),
           'R2': r2_score(y_test, y_pred_lr)
       })

       # Random Forest
       results.append({
           'Model': 'Random Forest',
           'MAE': mean_absolute_error(y_test, y_pred_rf),
           'MSE': mean_squared_error(y_test, y_pred_rf),
           'R2': r2_score(y_test, y_pred_rf)
       })

       # XGBoost
       results.append({
           'Model': 'XGBoost',
           'MAE': mean_absolute_error(y_test, y_pred_xgb),
           'MSE': mean_squared_error(y_test, y_pred_xgb),
           'R2': r2_score(y_test, y_pred_xgb)
       })

       # Create a DataFrame for easy comparison
       results_df = pd.DataFrame(results)
       print(results_df)

       # Optional: plot comparison
       results_df_melt = results_df.melt(id_vars='Model', var_name='Metric',␣
        ↪value_name='Score')
```

```
plt.figure(figsize=(10,5))
for metric in ['MAE', 'MSE', 'R2']:
    plt.bar(results_df['Model'] + f'_{metric}', results_df[metric],␣
 ↪label=metric)
plt.legend()
plt.ylabel('Score')
plt.title('Model Comparison: MAE, MSE, R2')
plt.show()

# (or plot separately for each metric, for clarity)
results_df.set_index('Model')[['MAE', 'MSE', 'R2']].plot(kind='bar',␣
 ↪figsize=(10,6))
plt.title("Model Comparison (MAE, MSE, R2)")
plt.ylabel("Score")
plt.show()
```

```
               Model        MAE          MSE        R2
0  Linear Regression   0.408268     2.157695  0.996542
1      Random Forest  61.624641  4766.299065 -6.639202
2            XGBoost  63.294175  5186.756385 -7.313091
```





2.4.1 7. Feature Importance & Explainability

2.4.2 8. Discussion & Next Steps