MIDTERM ASSIGNMENT - IT APPLICATION FOR BANKING AND FINANCE

AUTHOR: VO TRAN HUONG GIANG
STUDENT ID: 2211345002

TABLE OF CONTENTS

I. INTRODUCTION

Rationale

Despite some lingering negative signals, the Vietnamese stock market has exhibited strong positive momentum in 2024, reinforcing my belief that VNINDEX has the potential to reach 1400 in the near future. This optimism is not only supported by technical indicators but also by fundamental economic factors, corporate performance, and investor sentiment. From a technical perspective, VNINDEX remains in an upward price channel, trading above its long-term moving averages, with stochastic indicators showing bullish momentum. Fundamentally, Vietnam's trade balance has reached record highs, economic growth continues to show resilience, and the number of retail investors entering the market is steadily increasing. However, risks remain—foreign investors have been net sellers despite the country's strong export performance, partly due to currency depreciation and capital outflows from emerging markets. Additionally, liquidity has not yet fully shifted into the stock market, with significant capital flowing into alternative assets like gold and cryptocurrencies.

Research scope and application

In this study, I analyze both the bullish (technical and fundamental) and bearish (monetary and cash flow) factors to assess the probability of a sustained uptrend in 2025. To strengthen the argument for an upward trajectory, I employ machine learning techniques, specifically Random Forest Regression, to forecast VNINDEX based purely on historical data movements. Additionally, for robustness and comparability, I incorporate ARIMA combined with GARCH to perform a statistical regression analysis, benchmarking its results against the machine learning model. This study aims to serve as a valuable reference for investors seeking to understand market trends, manage risk, and optimize their investment strategies.

II. OVERVIEW

Positive market overview:

- The VNINDEX is poised for potential growth, aiming towards the 1400 points mark by 2025.
- Recent trading sessions have shown positive trends, supported by increased trading volumes.

Technical signals indicating growth

FIGURE 1 AND FIGURE 2

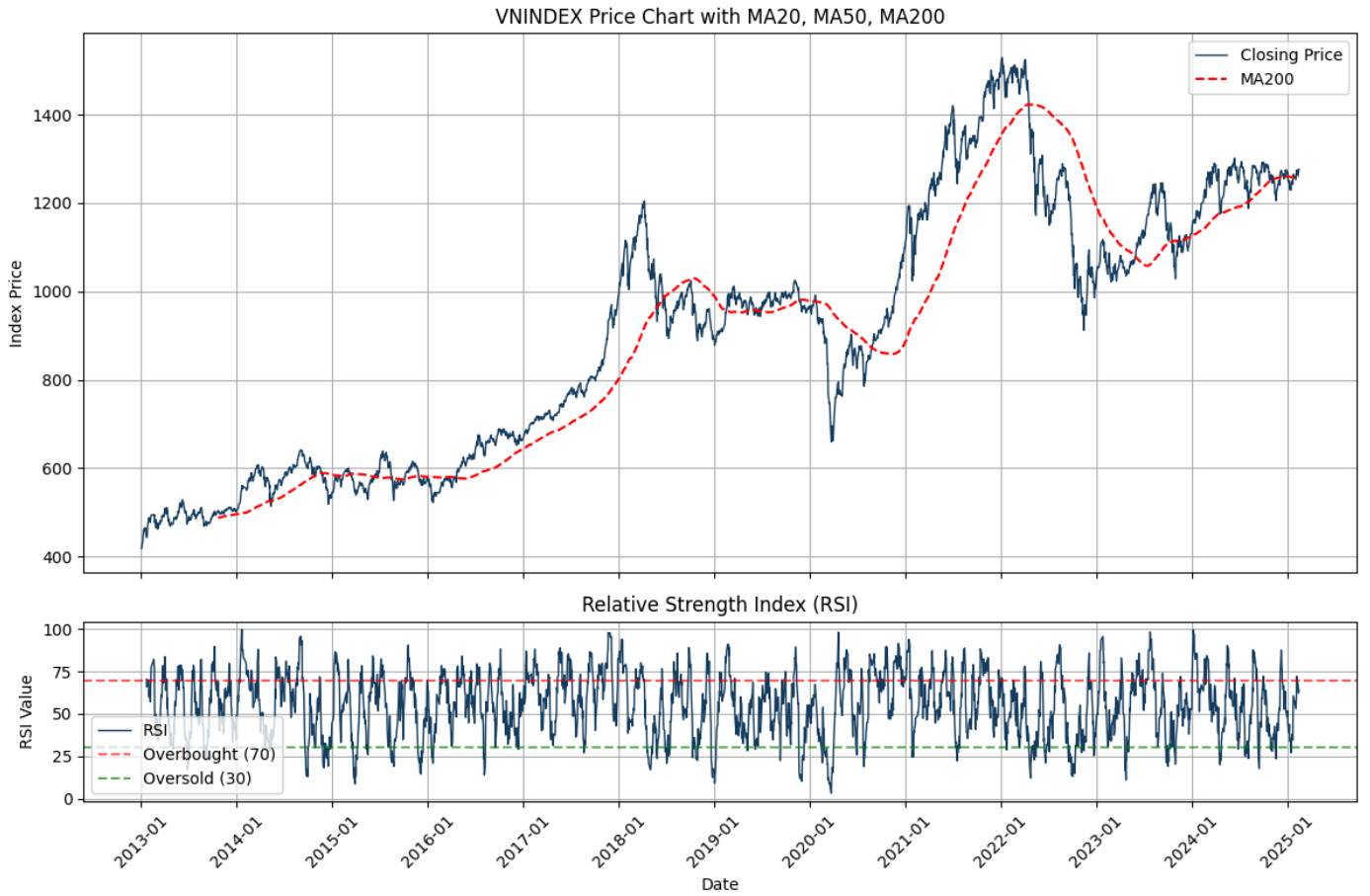
(1) Rising price channel:
Since late 2022, the VN-INDEX has maintained an upward price channel, even amid significant market events. Over the past two years, the index has consistently formed higher lows, with the second-lowest point around 1,000 in early 2024. **This is a strong bullish signal, as three consecutive higher lows indicate a well-established uptrend structure**.
However, one key factor to watch is that VNINDEX has yet to establish a significantly higher high, which would further confirm the trend. This could be attributed to continuous net foreign outflows, totaling over 95.2 trillion VND. Despite this selling pressure, VN-INDEX has demonstrated strong demand around the 1,200 level. If Vietnam's economic fundamentals were weak, the index would have likely broken below 1,200 and retested the 1,000 level due to overwhelming foreign investor selling. Additionally, for those with a keen eye, a **large inverse Head and Shoulders pattern** is subtly forming within the chart, hinting at a potential bullish reversal. If we closely observe the price action, we can see that the **price is consolidating, waiting for a breakout signal**. A key entry opportunity arises when the price breaks through the resistance zone of 1280 and forms a double-bottom effect—where the price initially breaks above resistance, pulls back to retest the level, and then bounces upward. This confirmation of resistance turning into support provides a strong entry point for a long position.

(2) Support from Moving Averages (MA):
Throughout 2024, the VN-INDEX has demonstrated strong resilience by repeatedly finding support at its long-term moving averages, such as the 20-day, 50-day, and 200-day moving averages. Each time the index has tested these levels, buying pressure has emerged, helping to maintain the broader uptrend.
This behavior indicates sustained investor confidence and suggests that key institutional players are accumulating positions at critical price points.

(3) Increasing Stochastic Indicator:
The Stochastic indicator has consistently remained above the 50 level, reflecting positive momentum and a continued appetite for risk among market participants.
This upward trend in the Stochastic oscillator suggests that buyers are actively driving the market higher, preventing any significant breakdowns in price action. **As long as this momentum remains strong, the VNINDEX is likely to sustain its bullish trajectory in the near future.**

In [2]:
```
# FIGURE 1
from IPython.display import display, Image
display(Image(filename="D:\Github\Repo1_HuongGiang\AI Application for Finance\Midterm Assignment\VNINDEX_2025-02-24_21-
```

In [3]:
```
# FIGURE 2
from IPython.display import display, Image
display(Image(filename="D:\Github\Repo1_HuongGiang\AI Application for Finance\Midterm Assignment\VNINDEX_RSI.png"))
```



Fundamental factors contributing to optimism

Vietnam's trade balance hits record highs (FIGURE 3)

Vietnam's trade activity surged in the first 11 months of 2024, with total exports and imports reaching 715 billion Dollar, surpassing 2023's total of 677 billion Dollar. The country also recorded a trade surplus of 24.31 billion Dollar, exceeding 2023's full-year surplus of 22.92 billion Dollar, reflecting a 6% increase. This significant surplus underscores Vietnam's strong economic performance and export-driven growth.

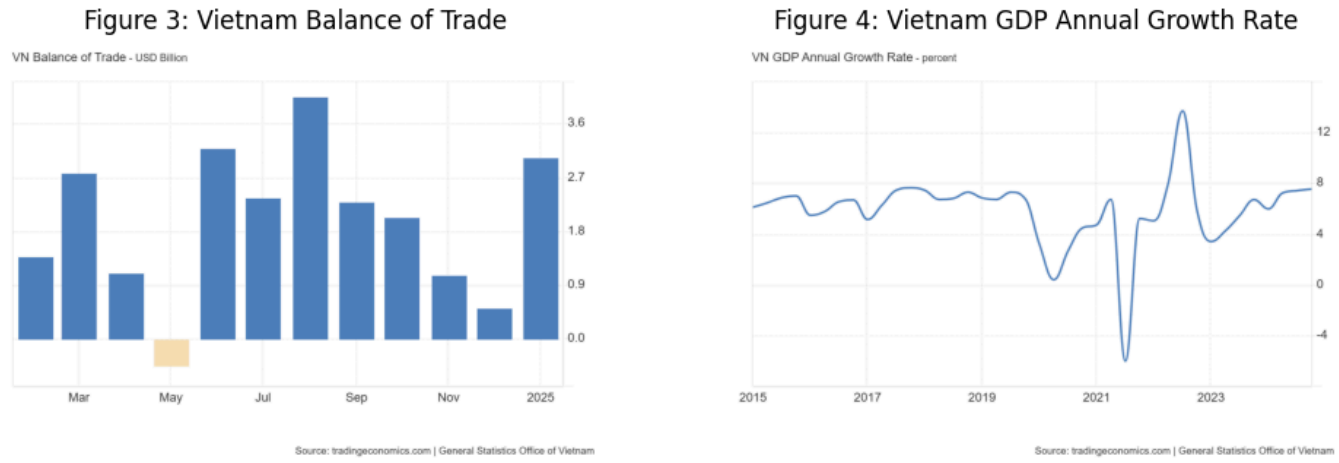
Vietnam's economy continues strong expansion (FIGURE 4)

Vietnam's GDP is projected to grow by around 7% in 2024, positioning it as the fastest-growing economy among the top six in Southeast Asia. The Industrial Production Index (IIP) increased by 8.4% year-over-year, fueling a 14.4% rise in exports. These figures reflect a solid industrial sector and a robust economic outlook for the country.

Stock market dees a durge in new accounts

The number of stock trading accounts in Vietnam experienced remarkable growth. By the end of 2023, there were 7,292,316 accounts, but by November 30, 2024, this number had risen to 9,157,109. The increase of 1,864,793 accounts, representing 25.55% growth, highlights a rising interest in the financial markets and stronger investor confidence.

In [4]:
```
# FIGURE 3 AND FIGURE 4
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# Đọc ảnh
img3 = mpimg.imread(r"D:\Github\Repo1_HuongGiang\AI Application for Finance\Midterm Assignment\VN_Balance_of_Trade.png"
img4 = mpimg.imread(r"D:\Github\Repo1_HuongGiang\AI Application for Finance\Midterm Assignment\VN_GDP_Annual_Growth_Rat

# Tạo figure với 2 cột
fig, axes = plt.subplots(1, 2, figsize=(12, 6))  # 1 hàng, 2 cột

# Hiển thị ảnh
axes[0].imshow(img3)
axes[0].set_title("Figure 3: Vietnam Balance of Trade")
axes[0].axis("off")  # Tắt trục

axes[1].imshow(img4)
axes[1].set_title("Figure 4: Vietnam GDP Annual Growth Rate")
axes[1].axis("off")  # Tắt trục

# Hiển thị hình ảnh
plt.show()
```



III. DATA ANALYSIS

In this section, I use the S&P 500 as a key variable due to the contrasting nature of the two markets. As a leading index in a developed economy, the U.S. market serves as a benchmark for comparison, providing valuable insights into global economic trends and market dynamics.

`fetchdata` library written by myself allow us to retrieve **historical stock data** from **Vietcap (VNINDEX)** and **Yahoo Finance (S&P500)**. The source codde in this library would handle potential rate limits and data format inconsistencies. Specifically, to avoid network issues and unexpected API responses when sending a large amount of requests to the website, we carefully used POST request and timesleep in the source code. This would break the first fire wall of these website and get access to the data.

Two code segments were employed to retrieve market data:

- **VNINDEX Data (Vietcap)**: Vietcap was selected as the data source for the VNINDEX due to:
  - Comprehensive dataset: A comparative analysis of several Vietnamese financial websites was conducted, and Vietcap offered the most extensive data.
  - Readily accessible data: Vietcap's data is publicly available and does not require user authentication, simplifying the retrieval process.
- **S&P500 Data (Yahoo Finance)**: Yahoo Finance was chosen as the data source for the S&P500. However, instead of using pre-built libraries, a custom solution was implemented using the Yahoo Finance API directly. This approach provided greater control and flexibility in data retrieval.

In [5]:
```
from fetchdata import *

# Create an instance of the FetchData class
fetcher = FetchData()
# Fetch S&P 500 data from Yahoo Finance
sp500 = fetcher.fetch_yahoo_finance("^GSPC")
# Fetch VNINDEX data from Vietcap
vnindex = fetcher.fetch_vietcap('VNINDEX')
```

```
d:\Github\Repo1_HuongGiang\AI Application for Finance\Midterm Assignment\fetchdata.py:74: FutureWarning: The behavior of
'to_datetime' with 'unit' when parsing strings is deprecated. In a future version, strings will be parsed as datetime st
rings, matching the behavior without a 'unit'. To retain the old behavior, explicitly cast ints or floats to numeric typ
e before calling to_datetime.
  df['time'] = pd.to_datetime(df['time'], unit='s')
```

In [6]:
```
vnindex
```

Out[6]:

time	open	high	low	close	volume	index
2013-01-02	415.50	419.05	414.41	418.35	64288223	VNINDEX
2013-01-03	420.68	420.68	413.68	419.27	99824757	VNINDEX
2013-01-04	418.75	426.48	417.50	426.06	64852320	VNINDEX
2013-01-07	428.84	434.64	426.86	434.19	78123432	VNINDEX
2013-01-08	437.25	447.16	437.15	447.16	119101143	VNINDEX
...	...	...	...	...	...	...
2025-02-10	1275.20	1275.20	1263.26	1263.26	794054460	VNINDEX
2025-02-11	1263.26	1268.45	1263.11	1268.45	622260600	VNINDEX
2025-02-12	1268.45	1272.86	1266.59	1266.91	504507839	VNINDEX
2025-02-13	1266.91	1270.67	1263.85	1270.35	511055562	VNINDEX
2025-02-14	1272.15	1280.60	1272.15	1276.08	647411257	VNINDEX

3020 rows × 6 columns

In [7]:
```
sp500
```

Out[7]:

time	volume	high	low	close	open	index
2013-01-02	4202600000	1462.430054	1426.189941	1462.420044	1426.189941	^GSPC
2013-01-03	3829730000	1465.469971	1455.530029	1459.369995	1462.420044	^GSPC
2013-01-04	3424290000	1467.939941	1458.989990	1466.469971	1459.369995	^GSPC
2013-01-07	3304970000	1466.469971	1456.619995	1461.890015	1466.469971	^GSPC
2013-01-08	3601600000	1461.890015	1451.640015	1457.150024	1461.890015	^GSPC
...	...	...	...	...	...	...
2025-02-05	4756250000	6062.859863	6007.060059	6061.479980	6020.450195	^GSPC
2025-02-06	4847120000	6084.029785	6046.830078	6083.569824	6072.220215	^GSPC
2025-02-07	4766900000	6101.279785	6019.959961	6025.990234	6083.129883	^GSPC
2025-02-10	4458760000	6073.379883	6044.839844	6066.439941	6046.399902	^GSPC
2025-02-11	4324880000	6076.279785	6042.339844	6068.500000	6049.319824	^GSPC

3047 rows × 6 columns

1. Graphing data and recognizing the relationship between S&P500 and VNINDEX.

Key insights:

- **Parallel Price Trends**: S&P 500 and VNINDEX prices generally move together.
- **VNINDEX More Volatile**: VNINDEX price swings are larger than S&P 500.
- **Diverging Volume**: S&P 500 volume is consistently high, VNINDEX volume surged in late 2020.
- **COVID-19 Hypothesis**: VNINDEX volume surge potentially linked to increased retail investing during the pandemic.

The graphs depict the price and volume trends of the S&P 500 and the VNINDEX from 2013 to 2025. Visually, both indices exhibit the **same trend** with a generally upward trajectory. However, the **magnitude of price** changes differs, with the VNINDEX showing more volatile swings compared to the smoother, more gradual growth of the S&P 500. This is reasonable with a young security market in Vietnam compared to a mature market in US.

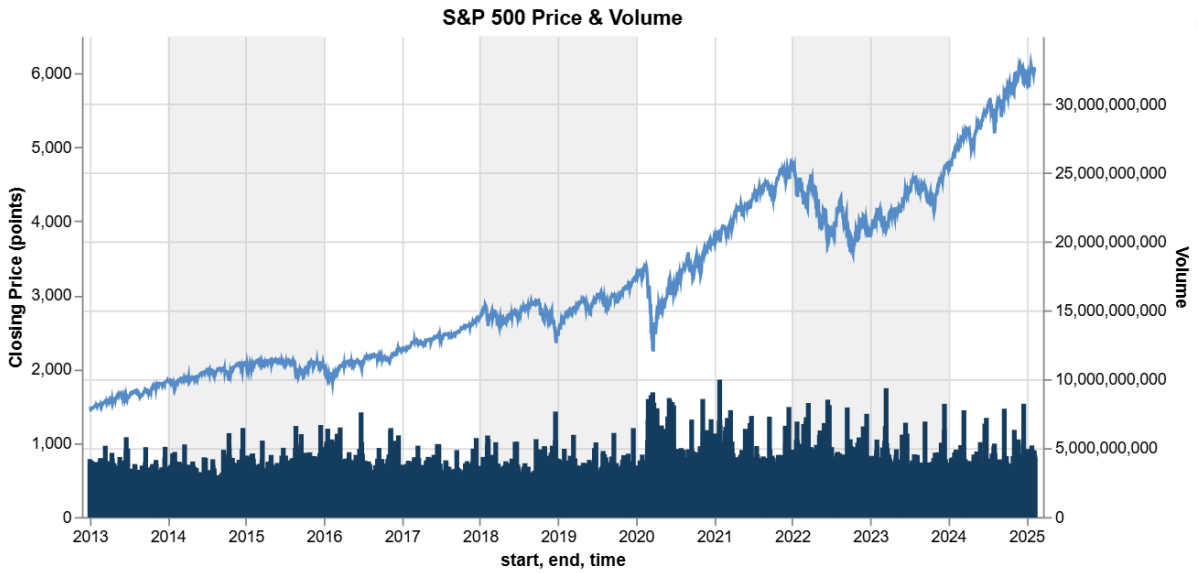
Examining the **volume patterns** reveals a key distinction: the S&P 500's volume fluctuates consistently between roughly 5 and 10 billion, while the VNINDEX volume remained relatively subdued until late 2020, after which it experienced a significant surge from around 0.5 billion to 1.5 billion. This late 2020 increase in VNINDEX volume potentially reflects the impact of the COVID-19 pandemic, which may have driven increased participation in the Vietnamese stock market due to factors such as stay-at-home orders, low interest rates, and a search for alternative investment opportunities.

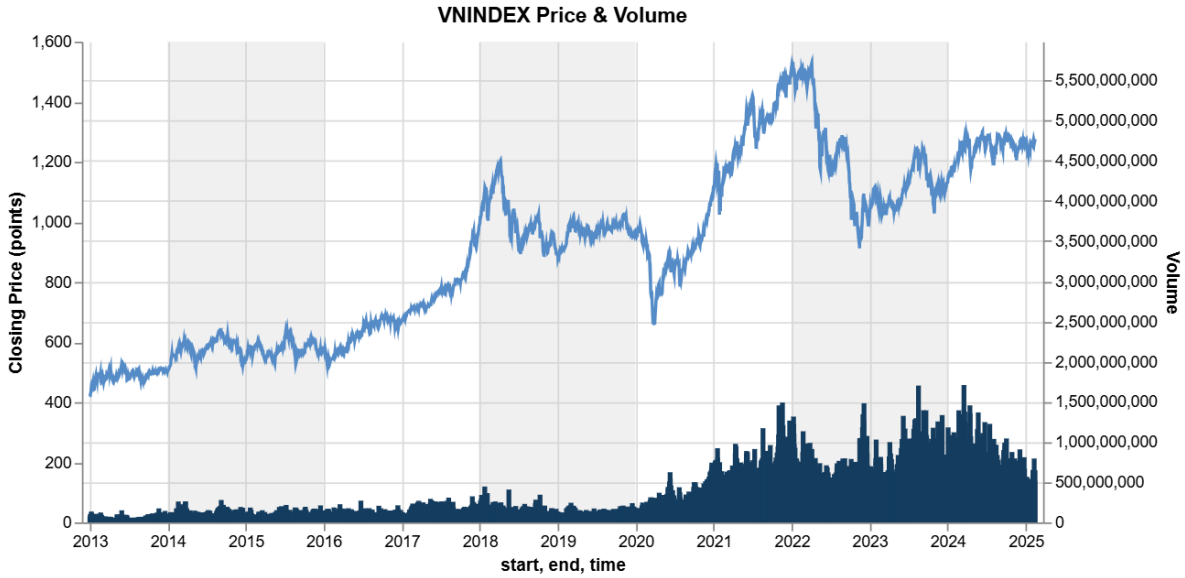
Moreover, with VNINDEX, we would observe the economic trend with 2-year period (2 years for up and 2 years for down).

In [8]:
```
# FIGURE 5
sp500 = sp500.reset_index()
vnindex = vnindex.reset_index()

def create_chart(df, title):
    base = alt.Chart(df).encode(x='time:T')

    # Define background shading for multiple periods
    background_data = pd.DataFrame({
        'start': ['2014-01-01', '2018-01-01', '2022-01-01'],
        'end': ['2015-12-31', '2019-12-31', '2023-12-31'],
        'color': ['lightgrey', 'lightgrey', 'lightgrey']
    })

    background = alt.Chart(background_data).mark_rect(opacity=0.3).encode(
        x='start:T',
        x2='end:T',
        color=alt.Color('color:N', scale=None)  # Keep color mapping fixed
    )
```

```
    # Volume bar chart (Secondary y-axis)
    volume_chart = base.mark_bar(color='#143D60', size=3).encode(
        y=alt.Y('volume:Q',
                axis=alt.Axis(title='Volume', orient='right', titleColor='black', tickCount=10),
                scale=alt.Scale(domain=[0, df['volume'].max() * 3.5], nice=False))
    )

    # Closing price line chart (Primary y-axis)
    close_chart = base.mark_line(color='#578FCA', strokeWidth=2).encode(
        y=alt.Y('close:Q', axis=alt.Axis(title='Closing Price (points)', titleColor='black'))
    )

    # Combine all layers
    combined_chart = (
        alt.layer(background, volume_chart, close_chart)
        .resolve_scale(y='independent')
        .properties(title=title, width=600, height=300)
    )

    return combined_chart

# Create individual charts
sp500_chart = create_chart(sp500, "S&P 500 Price & Volume")
vnindex_chart = create_chart(vnindex, "VNINDEX Price & Volume")

# Stack the charts vertically
final_chart = alt.vconcat(sp500_chart, vnindex_chart).configure_view(
    strokeWidth=0  # Remove default border
)

# Display the final chart
final_chart
```

Out[8]:





2. Data processing

When processing data, we begin by considering the **data requirements of each model** we plan to apply. To ensure consistency across the dataset for all models, we transform the data to meet the necessary conditions for each model type.

In this analysis, we use **ARIMA (an econometrics model)** and **RandomForest Regression (a machine learning model)** to assess the accuracy of these models in forecasting future price movements of the VNINDEX (a detailed discussion on model selection is provided in the "*Model Assessment*" *section* below). Since ARIMA has stricter requirements for accurate modeling compared to machine learning models, we prioritize **ARIMA's criteria** when preprocessing the dataset.

One of ARIMA's key requirements is **stationarity**. By visually inspecting the graph above, we can see that the data is non-stationary, as the price exhibits a continuous upward trend ("*Model Assessment*" *section*). This means we need to transform the data into a stationary form.

Instead of using differencing, we apply **returns** to achieve stationarity. This approach allows for a more precise comparison between the two models' outputs. Since each model operates on different benchmarks, using differencing would not be appropriate for maintaining the interpretability of the results.

In [9]:
```
# Calculate returns
vnindex['close_pct'] = vnindex['close'].pct_change()*100
sp500['close_pct'] = sp500['close'].pct_change()*100

# Drop NaN rows
```

```
vnindex = vnindex.dropna(subset=['close_pct'])
sp500 = sp500.dropna(subset=['close_pct'])
```

3. Summary statistics

a. Descriptive statistics

Based on the summary statistics of S&P 500 (`close_pct_sp500`) and VNINDEX (`close_pct_vnindex`), below is my comments:

- **Mean Return**
  - The average daily return of S&P 500 is 0.0538%, slightly **higher than VNINDEX** at 0.0449%.
  - Both indices have positive mean returns, indicating an **overall upward trend** over the observed period.
- **Standard Deviation (Volatility)**
  - S&P 500 has a standard deviation of 1.0684, whereas VNINDEX has a slightly higher 1.1256.
  - This suggests that **VNINDEX exhibits greater price fluctuations** (higher volatility) compared to S&P 500.
- **Minimum and Maximum Returns**
  - S&P 500 has a larger range of returns (riskier), with a minimum of -11.98% and a maximum of 9.38%.
  - VNINDEX shows a smaller range, with a minimum of -6.67% and a maximum of 4.98%.
  - This suggests that **S&P 500 experienced more extreme price movements (riskier)** compared to VNINDEX.
- **Percentiles (Distribution of Returns)**
  - The median (50%) return for S&P 500 is 0.0716%, whereas for VNINDEX, it is 0.1195%.
  - This indicates that VNINDEX tends to have slightly higher median returns despite its lower mean return.
  - The interquartile range (IQR) for VNINDEX (difference between 25th and 75th percentiles) is larger than that of S&P 500, reinforcing **VNINDEX's relatively higher volatility**.

These statistics suggest that S&P 500 is more prone to sharp crashes and surges, whereas VNINDEX is relatively more stable but still volatile.

The experiencing of S&P 500 with sharper crashes and surges compared to the VNINDEX would be explained by 4 factors:

- **Market Structure & Liquidity**
  - The S&P 500 has high liquidity and deep market participation, leading to rapid price movements during uncertainty or speculation.
  - VNINDEX, with a smaller market size and lower trading volume, experiences relatively smoother price changes.
- **Investor Behavior**
  - The U.S. market has more institutional investors and algorithmic trading, leading to sharper price swings
  - VNINDEX has a higher retail investor base, resulting in more gradual movements.
- **Sector Composition**
  - The S&P 500's heavy tech stock exposure increases volatility
  - VNINDEX, with a focus on banking, real estate, and industrials, exhibits a different risk profile.
- **Regulations & Trading Mechanisms**
  - High-frequency trading and leverage contribute to rapid S&P 500 fluctuations
  - Vietnam's market regulations and trading restrictions help moderate extreme spikes.

In [12]:
```
# Compute summary statistics table
pd.options.display.float_format = '{:,.4f}'.format
summary_stats = database.describe()
summary_stats
```

Out[12]:

	close_pct_sp500	close_pct_vnindex
count	2,924.0000	2,924.0000
mean	0.0538	0.0449
std	1.0684	1.1256
min	-11.9841	-6.6744
25%	-0.3630	-0.4328
50%	0.0716	0.1195
75%	0.5610	0.6365
max	9.3828	4.9801

b. Histogram and times series plot

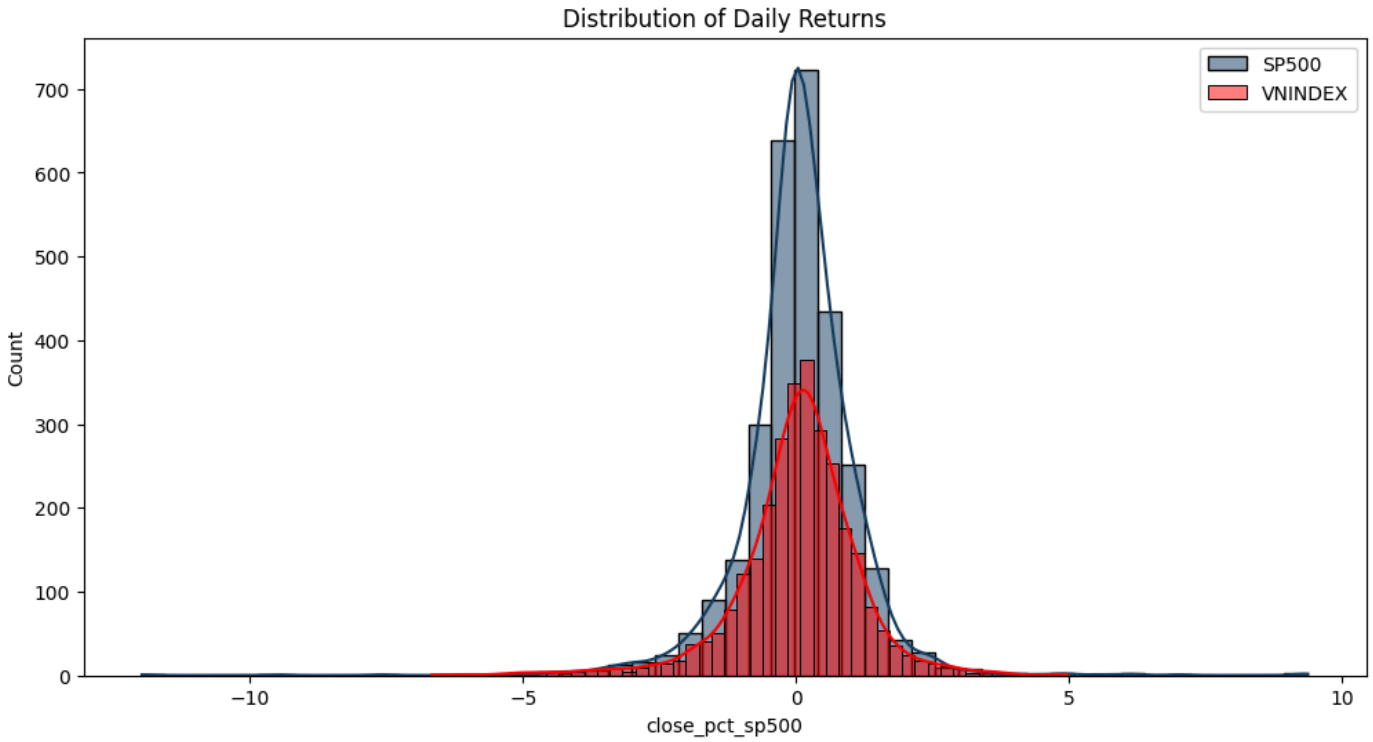
Key insights:: The S&P 500 exhibits lower daily volatility, with extreme movements occurring but not frequently. In contrast, the VNINDEX shows a wider distribution, indicating higher volatility with more frequent moderate returns.

The **S&P 500's distribution** is more concentrated around zero, indicating lower daily volatility. While extreme movements do occur, they are less frequent, suggesting a more stable market. In contrast, the **VNINDEX** has a wider spread, reflecting higher daily volatility with more frequent moderate-sized returns.

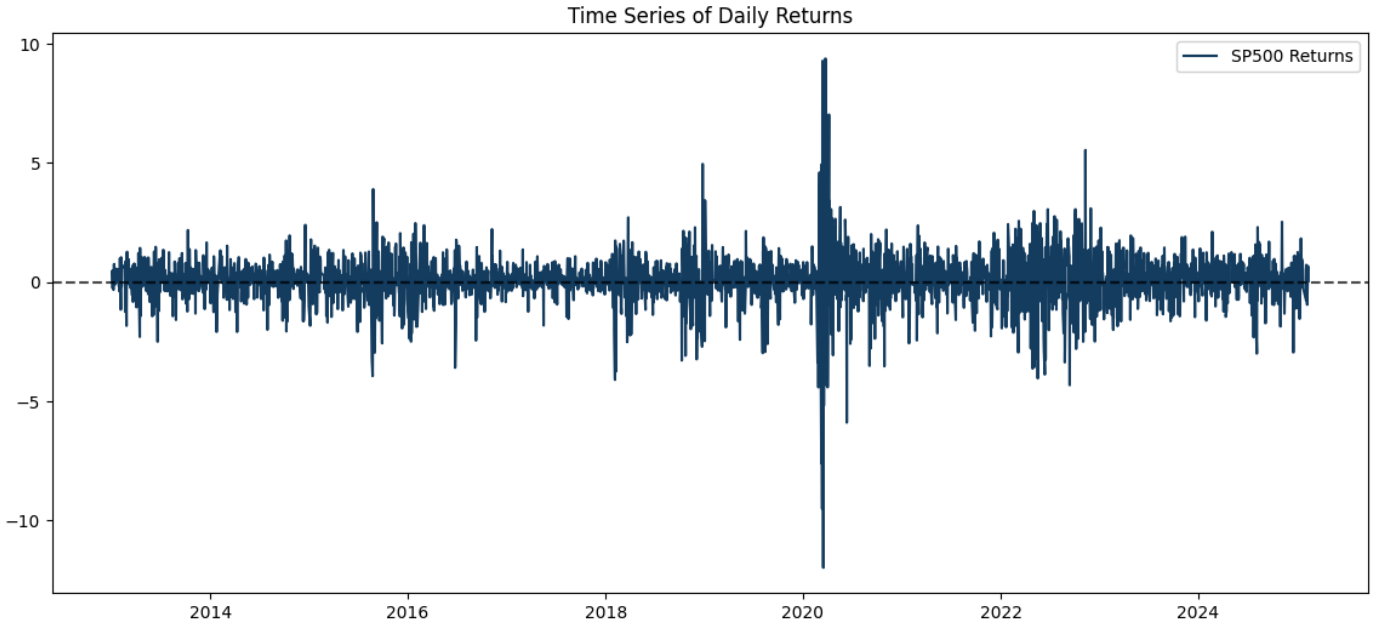
One **reason** for this difference is ***(1) market structure and liquidity***. The S&P 500 is one of the most liquid markets globally, allowing prices to adjust smoothly, while the VNINDEX has lower liquidity, making it more prone to larger price swings. ***(2) Investor composition*** also plays a role. The S&P 500 is dominated by institutional investors who trade systematically, reducing excess volatility, whereas the VNINDEX has a higher proportion of retail investors, whose emotional trading can lead to sharper price movements.

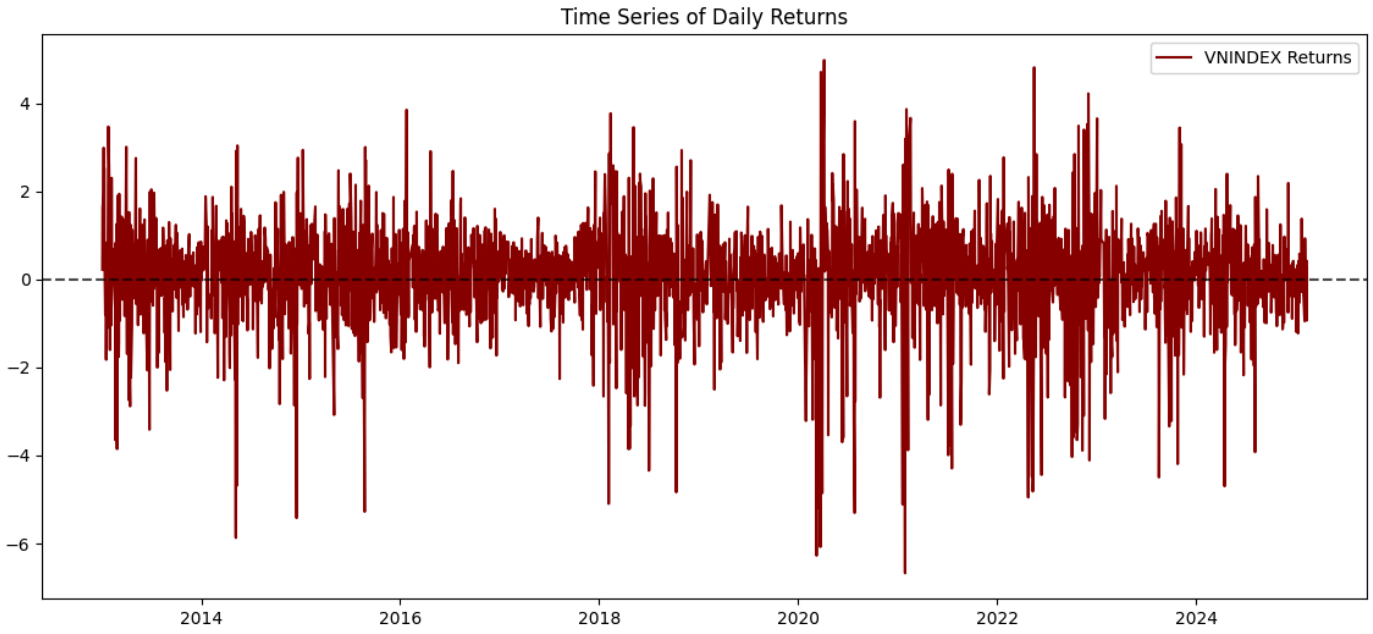
Additionally, ***(3) regulations and circuit breakers*** impact volatility. The U.S. market has strict trading halts that prevent extreme price swings, while Vietnam's regulatory mechanisms may allow for greater short-term fluctuations. Unlike Vietnam's stock market, U.S. exchanges such as NYSE and NASDAQ do not have daily price limits (price ceilings or floors). Instead, they use circuit breakers and Limit Up-Limit Down (LULD) mechanisms to temporarily pause trading during extreme market volatility, allowing prices to adjust freely based on supply and demand. These factors collectively explain why the S&P 500 experiences fewer dramatic shifts, while the VNINDEX remains more volatile.

In [13]:
```
# Histogram & Density Plot
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12,6))
sns.histplot(database['close_pct_sp500'], bins=50, kde=True, color='#143D60', label="SP500")
sns.histplot(database['close_pct_vnindex'], bins=50, kde=True, color='red', label="VNINDEX")
plt.legend()
plt.title("Distribution of Daily Returns")
plt.show()
```



In [14]:
```
# Time Series Plot
plt.figure(figsize=(14,6))
plt.plot(database.index, database['close_pct_sp500'], label='SP500 Returns', color='#143D60')
plt.axhline(y=0, color='black', linestyle='--', alpha=0.7)
plt.legend()
plt.title("Time Series of Daily Returns")
plt.show()

plt.figure(figsize=(14,6))
plt.plot(database.index, database['close_pct_vnindex'], label='VNINDEX Returns', color='darkred')
plt.axhline(y=0, color='black', linestyle='--', alpha=0.7)
plt.legend()
plt.title("Time Series of Daily Returns")
plt.show()
```





c. Correlation matrix

Correlation of 0.097377 between the S&P 500 and the VNINDEX is quite low (0.0974), indicating a weak positive linear relationship between the two markets. This partly suggests that:

- **Market independence**: The VNINDEX and S&P 500 operate largely independently, meaning U.S. stock market fluctuations have limited impact on Vietnam's market.
  - **Domestic-driven VNINDEX**: Vietnam's stock market is influenced more by local economic policies, interest rates, and investor sentiment rather than global market trends.
  - **Limited globalization effect**: While some connection exists due to global capital flows, the weak correlation suggests Vietnam's market does not strongly follow U.S. market movements.
- **Diversification benefits**: For investors, a low correlation means that investing in both markets can help reduce portfolio risk by spreading exposure across different economic environments.

Application:
Given the low correlation (0.097377), a linear regression model would not be suitable to capture the relationship between the S&P 500 and VNINDEX. A non-linear approach would be more appropriate to explore any potential hidden patterns or interactions. However, **this is not our problem** because in this study, we do not aim to model the relationship between the two indices. Instead, we only apply **models individually to VNINDEX** to assess their effectiveness in **forecasting future price movements** within their own market and use S&P 500 as benchmark of standardized data to make further compare. This approach ensures that each model is optimized for predicting its respective index, independent of external market influences.

In [15]:
```
# Calculate correlation
correlation = database[['close_pct_sp500','close_pct_vnindex']].corr()
correlation
```

Out[15]:

	close_pct_sp500	close_pct_vnindex
close_pct_sp500	1.0000	0.0974
close_pct_vnindex	0.0974	1.0000

IV. MODEL ASSESSMENTS

Set up libraries

1. Prepares time-series data for model training and testing

`data_preprocessing` prepares time-series data for model training and testing. It takes the time-series data, the number of lags to consider (`num_lags`), and the train-test split ratio (`train_test_split`) as input. The function creates lagged features (x) and corresponding target values (y). Specifically, for each point in the time series (excluding the last `num_lags` points), it creates a feature vector by taking the preceding `num_lags` values. The corresponding target value is the value at the next time step. The data is then split into training and testing sets based on the provided `train_test_split` ratio. The function returns the training features, training targets, testing features, and testing targets.

In [16]:
```
def data_preprocessing(data, num_lags, train_test_split):
    """Prepares time-series data for model training and testing."""
    x, y = [], []
    for i in range(len(data) - num_lags):
        x.append(data[i:i + num_lags])
        y.append(data[i + num_lags])

    x, y = np.array(x), np.array(y)
    split_index = int(train_test_split * len(x))
    return x[:split_index], y[:split_index], x[split_index:], y[split_index:]
```

2. RandomForest Regressor

`get_random_forest_regressor` function written by myself simplifies the initialization of a Random Forest model for regression tasks. Built upon `BaggingRegressor` , it allows users to easily set key parameters such as the number of trees (`n_estimators`), the maximum depth of each tree (`max_depth`), and the `random_state` for reproducibility. Using `BaggingRegressor` with `DecisionTreeRegressor` as the base estimator is a common approach to building Random Forest models, leveraging the benefits of bagging to reduce overfitting and improve accuracy.

In [17]:
```
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import BaggingRegressor, BaggingClassifier

def get_random_forest_regressor(n_estimators=100, max_depth=None, random_state=None):
    """
    Bagging-based Random Forest Regressor.
    Defaults:
    - n_estimators=100
    - max_depth=None
    - random_state=None
    """
    return BaggingRegressor(
        estimator=DecisionTreeRegressor(max_depth=max_depth, random_state=random_state),
        n_estimators=n_estimators,
        bootstrap=True,
        n_jobs=-1,
        random_state=random_state
    )
```

3. Plots training, test, and predicted values for visualization

`plot_train_test_values` visualizes the training, test, and predicted values of a time series. It takes the window size, training window size, training data, test data, and predicted values as input. The function generates a plot displaying the training data, test data, and the model's predictions, clearly separated by a vertical line indicating the split between training and testing periods. This allows for a direct visual comparison of the model's performance on the test set.

In [18]:
```
def plot_train_test_values(window, train_window, y_train, y_test, y_predicted):
    """Plots training, test, and predicted values for visualization."""
    plt.figure(figsize=(15, 6))
    prediction_window, first, second = window, train_window, window - train_window
    y_predicted, y_test = np.reshape(y_predicted, (-1, 1)), np.reshape(y_test, (-1, 1))

    plotting_time_series = np.zeros((prediction_window, 3))
    plotting_time_series[0:first, 0], plotting_time_series[first:, 1], plotting_time_series[first:, 2] = (
        y_train[-first:], y_test[:second, 0], y_predicted[:second, 0]
    )
    plotting_time_series[:first, 1:] = np.nan
    plotting_time_series[first:, 0] = np.nan

    plt.plot(plotting_time_series[:, 0], label='Training data', color='#143D60', linewidth=2.5)
    plt.plot(plotting_time_series[:, 1], label='Test data', color='#143D60', linestyle='dashed', linewidth=2)
```

```
    plt.plot(plotting_time_series[:, 2], label='Predicted data', color='red', linewidth=1)
    plt.axvline(x=first, color='#143D60', linestyle='--', linewidth=1)
    plt.grid()
    plt.legend()
    plt.show()
```

4. ARIMA + GARCH Model

`fit_arima_garch` fits ARIMA and, if necessary, GARCH models to time series data. It first fits an ARIMA model and then performs an ARCH test on the residuals to detect heteroskedasticity. If significant heteroskedasticity is present, a GARCH model is fitted to the residuals. The function returns the fitted ARIMA model and the fitted GARCH model (or None if no GARCH model was fitted).

In [19]:
```
def fit_arima_garch(data, arima_order=(2, 1, 1), garch_order=(1, 1), dist='normal'):
    """
    Fits ARIMA and GARCH models to the provided data.

    Parameters:
    data (array-like): Time series data
    arima_order (tuple): Order of the ARIMA model (p, d, q)
    garch_order (tuple): Order of the GARCH model (p, q)
    dist (str): Distribution for the GARCH model. Default is 'normal'.

    Returns:
    result_arima: Fitted ARIMA model
    result_garch: Fitted GARCH model (if heteroskedasticity is detected)
    """
    # Fit ARIMA Model
    model_arima = ARIMA(data, order=arima_order)
    result_arima = model_arima.fit()

    # Extract residuals from ARIMA model
    residuals = result_arima.resid

    # Perform ARCH test to check for heteroskedasticity
    arch_test = het_arch(residuals)
    p_value = arch_test[1]
    print(f"P-value from ARCH Test: {p_value}")

    result_garch = None
    # If p-value < 0.05, it indicates presence of heteroskedasticity → use GARCH
    if p_value < 0.05:
        print("Significant heteroskedasticity detected. Applying GARCH model...")

        # Fit GARCH(1,1) model to ARIMA residuals
        garch_model = arch_model(residuals, vol='Garch', p=garch_order[0], q=garch_order[1], dist=dist)
        result_garch = garch_model.fit(disp="off")  # Suppress output for cleaner results

        # Print GARCH results summary
        print(result_garch.summary())

        # Show p-values after GARCH
        garch_residuals = result_garch.resid  # Extract residuals from GARCH
        arch_test_after_garch = het_arch(garch_residuals)
        print(f"P-value after GARCH: {arch_test_after_garch[1]}")
    else:
        print("No significant heteroskedasticity detected. GARCH model may not be needed.")

    return result_arima, result_garch
```

5. Evaluating ratios

A. R-SQUARED METRIC

Python:

```
def r_squared(predicted_returns, real_returns):
    return 1 - np.sum((real_returns - predicted_returns) ** 2) / np.sum((real_returns -
np.mean(real_returns)) ** 2)
```

Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where:

- y_i represents the i-th real return.
- $\hat{y}_i$ represents the i-th predicted return.
- $\bar{y}$ represents the mean of the real returns.
- n is the number of data points.

B. PREDICTIVE ACCURACY

This function measures the difference between the predictions and the real (test) values.

Python:

```
class LossFunctions:
    @staticmethod
    def mean_squared_error(predicted_returns, real_returns):
        return np.mean((predicted_returns - real_returns) ** 2)

    @staticmethod
    def mean_absolute_error(predicted_returns, real_returns):
        return np.mean(np.abs(predicted_returns - real_returns))

    @staticmethod
    def mean_absolute_percentage_error(predicted_returns, real_returns):
        return np.mean(np.abs((predicted_returns - real_returns) / real_returns)) * 100
```

Formula:

- **Mean Squared Error (MSE):** $MSE = \frac{1}{N}\sum_{i=1}^{N}(r_i^p - r_i^r)^2$
- **Mean Absolute Error (MAE):** $MAE = \frac{1}{N}\sum_{i=1}^{N}|r_i^p - r_i^r|$
- **Mean Absolute Percentage Error (MAPE):** $MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{r_i^p - r_i^r}{r_i^r}\right| \times 100\%$

Where:

- N is the total number of samples.
- r_i^p is the predicted return for sample i.
- r_i^r is the real return for sample i. Note that MAPE is undefined if any r_i^r is zero. The Python code implicitly handles this (though it might be better to explicitly handle it to avoid NaNs).

C. DIRECTIONAL ACCURACY

The accuracy function counts the number of times the predicted return's sign matches the real return's sign.

Python:

```
def accuracy(predicted_returns, real_returns):
    predicted_returns = np.reshape(predicted_returns, (-1, 1))
    real_returns = np.reshape(real_returns, (-1, 1))
    hits = np.sum(np.sign(predicted_returns) == np.sign(real_returns))
    total_samples = len(predicted_returns)
    return (hits / total_samples) * 100
```

Formula:

$$Accuracy = \frac{\sum_{i=1}^{N} I(\text{sign}(r_i^p) = \text{sign}(r_i^r))}{N} \times 100\%$$

Where:

- N is the total number of samples.
- r_i^p is the predicted return for sample i.
- r_i^r is the real return for sample i.

- $I(x)$ is the indicator function, equal to 1 if x is true, and 0 otherwise. (This represents the `np.sign(predicted_returns) == np.sign(real_returns)` part).

D. MODEL BIAS

This function calculates the ratio of bullish to bearish forecasts.

Python:

```
def model_bias(predicted_returns):
    bullish_forecasts = np.sum(predicted_returns > 0)
    bearish_forecasts = np.sum(predicted_returns < 0)
    return bullish_forecasts / bearish_forecasts if bearish_forecasts != 0 else np.inf
```

Formula:

$$Bias = \frac{\sum_{i=1}^{N} I(r_i^p > 0)}{\sum_{i=1}^{N} I(r_i^p < 0)}$$

Where:

- N is the total number of predictions.
- r_i^p is the predicted return for sample i.
- The denominator is zero if there are no bearish predictions, hence the `np.inf` handling in the Python code.

1. Machine Learning Model: RandomForest Regression

In [21]:
```
# close_pct_vnindex
values_vnindex = database['close_pct_vnindex'].values
values_vnindex
```

Out[21]:
```
array([ 0.21991156,  1.61948148,  1.90818195, ...,  0.29257244,
       -0.93632371,  0.41084179])
```

RandomForest Regression

Random Forest Regression is chosen over Classification when the target variable is continuous rather than categorical

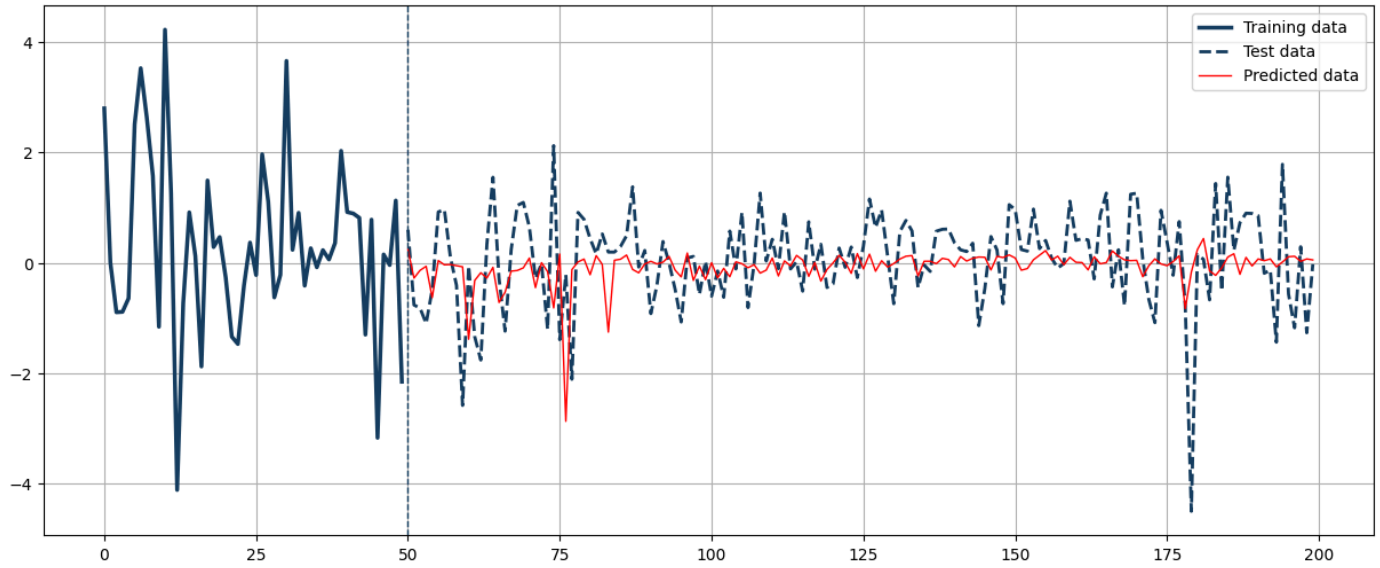
Regression predicts numerical values (e.g., stock prices, temperature) instead of discrete labels (e.g., yes or no). It optimizes using metrics like MSE, RMSE, and R^2, while classification focuses on accuracy and class probabilities. Since our data involves predicting real numbers, using Random Forest Regression ensures more accurate and meaningful predictions.

- The process starts with setting hyperparameters, where `num_lags = 500` determines how many past observations are used to predict future values, and `train_test_split = 0.80` specifies that 80% of the data is used for training while 20% is reserved for testing.
- Next, the `data_preprocessing` function transforms the time-series data into input-output pairs, creating `x_train`, `y_train` for training and `x_test`, `y_test` for testing.
- A Random Forest Regressor model is then initialized with 100 trees and a maximum depth of 20. The model is trained using `x_train` and `y_train`, learning the relationship between input features and target values.
- Finally, predictions are made both in-sample (on training data) and out-of-sample (on test data), reshaping them for further analysis and evaluation.

In [22]:
```
# Setting the hyperparameters
num_lags = 500
train_test_split = 0.80
# Creating the training and test sets
x_train, y_train, x_test, y_test = data_preprocessing(values_vnindex, num_lags, train_test_split)
# Fitting the model
model = get_random_forest_regressor(n_estimators=100, max_depth=20, random_state=123)
model.fit(x_train, y_train)

# Predicting in-sample
y_predicted_train = np.reshape(model.predict(x_train), (-1, 1))
```

```
# Predicting out-of-sample
y_predicted = np.reshape(model.predict(x_test), (-1, 1))
```

In [23]:
```
# plotting
plot_train_test_values(window=200, train_window=50,
                       y_train=y_train, y_test=y_test, y_predicted=y_predicted)
```



COMMENTS (Below performance evaluation)

Accuracy (Train: 80.04%, Test: 50.72%)

The model performs well on the training data, achieving 80.04% accuracy, but its performance drops significantly to 50.72% on the test data, indicating ***potential overfitting***. This suggests that while the model learns patterns effectively from historical data, it struggles to generalize to unseen data, which is a ***common issue in stock market prediction***.

Financial markets are highly volatile and influenced by numerous external factors such as investor sentiment, government policies, and unexpected events like pandemics or trade wars, which are difficult to capture in a purely data-driven model. The ***negative correlation in out-of-sample predictions*** further confirms that the model fails to predict future returns accurately.

To improve generalization, techniques such as feature engineering, incorporating macroeconomic indicators, applying regularization, or using alternative ensemble methods like boosting could be considered. Additionally, a rolling window approach, where the model is retrained periodically on the most recent data, could help it adapt better to changing market conditions. However, given the inherently stochastic nature of financial markets, even an optimized model will have limitations in predicting future trends with high certainty.

Root Mean Squared Error (RMSE) (Train: 0.739, Test: 0.971)

The RMSE measures the average error magnitude between predicted and actual values. The higher RMSE on the test set (0.971 vs. 0.739 on the train set) confirms the drop in model performance when applied to unseen data. A lower RMSE is preferable, indicating better predictive accuracy.

Correlation (In-Sample: 0.904, Out-of-Sample: -0.034)

The high in-sample correlation (0.904) indicates that the model captures strong relationships within the training data. However, the negative correlation on test data (-0.034) is concerning, suggesting that the model's predictions for new data may be random or even inversely related to actual returns. This is a strong sign of overfitting or model instability due to struggles to recognize unexpected data.

Model Bias (1.22)

The model bias is calculated as the ratio of bullish to bearish forecasts. A value of 1.22 indicates a slight bias toward bullish predictions, meaning the model predicts rising prices more often than declining ones. This could lead to inaccurate forecasts in bearish market conditions and should be analyzed further.

In [24]:
```
# Performance evaluation
print('---')
print('Accuracy Train = ', round(accuracy(y_predicted_train, y_train), 2), '%')
print('Accuracy Test = ', round(accuracy(y_predicted, y_test), 2), '%')
print('RMSE Train = ', round(np.sqrt(mean_squared_error(y_predicted_train, y_train)), 10))
print('RMSE Test = ', round(np.sqrt(mean_squared_error(y_predicted, y_test)), 10))
print('Correlation In-Sample Predicted/Train = ', round(np.corrcoef(np.reshape(y_predicted_train, (-1)), y_train)[0][1]
print('Correlation Out-of-Sample Predicted/Test = ', round(np.corrcoef(np.reshape(y_predicted, (-1)), np.reshape(y_test
print('Model Bias = ', round(model_bias(y_predicted), 2))
print('---')
```

```
---
Accuracy Train =  80.04 %
Accuracy Test =  50.72 %
RMSE Train =  0.7390302378
RMSE Test =  0.9706804717
Correlation In-Sample Predicted/Train =  0.904
Correlation Out-of-Sample Predicted/Test =  -0.034
Model Bias =  1.22
---
```

Below desciptive statistics

The prediction results show that the model tends to underestimate both the magnitude and volatility of VNINDEX returns.

While the mean predicted return is slightly lower than the actual, the standard deviation is significantly compressed, indicating the model struggles to capture extreme price movements. This is further reflected in the narrower interquartile range and underestimated tail risks. To improve accuracy, incorporating additional features, exploring nonlinear models, or adjusting for volatility could help the model better reflect market dynamics.

In [25]:
```
# Desciptive statistics
# Assuming y_predicted is a numpy array
y_predicted_series = pd.Series(y_predicted.flatten())
print('-----Prediction')
print(y_predicted_series.describe())
print('-----Actual')
# Assuming y_predicted is a numpy array
y_test_series = pd.Series(y_test.flatten())
print(y_test_series.describe())
```

```
-----Prediction
count   485.0000
mean     -0.0300
std       0.2494
min      -2.8698
25%      -0.0922
50%       0.0134
75%       0.0980
max       0.4400
dtype: float64
-----Actual
count   485.0000
mean      0.0313
std       0.9287
min      -4.6992
25%      -0.4104
50%       0.0964
75%       0.5560
max       3.4444
dtype: float64
```

2. Econometrics Model: ARIMA + GARCH

We evaluated multiple econometric models based on their characteristics and suitability for our dataset. The table below outlines the key models considered:

Model	When to use?	Advantages	Disadvantages
ARMA (Autoregressive Moving Average)	When dealing with a stationary time series and a linear relationship between	Easy to implement, clear interpretation of coefficients.	Cannot handle trends or non-stationary data. Assumes linearity.

Model	When to use?	Advantages	Disadvantages
	the current value and past values and past forecast errors.		
ARIMA (Autoregressive Integrated Moving Average)	When the data exhibits a trend or seasonality. Requires differencing to achieve stationarity.	Supports differencing to account for trends and seasonality, generally better forecasting ability than ARMA.	Not suitable for multivariate analysis (multiple time series). Differencing can lose information.
ARDL (Autoregressive Distributed Lag)	When analyzing the impact of one or more variables (e.g., S&P 500 on VNINDEX) allowing for both short-run and long-run effects. Useful even if variables are not co-integrated.	Differentiates short-term and long-term effects. Relatively easy to estimate.	May not be the most efficient estimator if variables are co-integrated.
VAR (Vector Autoregression)	When examining bidirectional relationships between multiple time series (e.g., VNINDEX and S&P 500).	Handles multiple variables, captures dynamic interactions between variables. Relatively straightforward to estimate.	Does not explicitly differentiate long-term equilibrium relationships. Can be computationally intensive with many variables. May suffer from over-parameterization.
VECM (Vector Error Correction Model)	When multiple time series (e.g., VNINDEX and S&P 500) are co-integrated (have a long-run equilibrium relationship).	Suitable for co-integrated variables, explicitly models both short-run dynamics and long-run equilibrium.	Requires prior cointegration testing (e.g., Johansen test). More complex to estimate and interpret than VAR.

Why ARIMA? Based on the data characteristics, we selected ARIMA for our analysis due to the following reasons:

- **Trend in the dataset**: ARIMA is appropriate for data with seasonality and allows for differencing or using return, making it suitable for handling non-stationary time series.
- **Univariate focus**: Since our objective is to forecast individual indices, ARIMA is appropriate as it is designed for single-variable time series forecasting.
- **Strong forecasting capability**: ARIMA is widely used for financial time series forecasting, making it a well-established method for our study.

Introduction: ARIMA(p, d, q) Model

The ARIMA(p, d, q) model is a combination of three main components:

1. AR (AutoRegressive - p): The autoregressive component indicates that the current value of the time series depends on its past values. This model is based on the relationship:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \varepsilon_t$$

Where:

- ϕ_i are the autoregressive coefficients.
- Y_{t-p} are the past values of the time series.
- ε_t is white noise.

Meaning:

- AR helps the model capture the trend and cyclical patterns in the data because it describes how past values influence the current value.
- If the data shows strong correlation with its own past values (a clear cutoff in the Partial Autocorrelation Function (PACF)), an AR(p) model is used.

2. I (Integrated - d): The integrated component transforms a non-stationary time series into a stationary one by differencing consecutive values:

$$Y'_t = Y_t - Y_{t-1}$$

If the series is still non-stationary after one differencing, higher-order differencing is applied.

Meaning:

- If the data shows a trend (increasing or decreasing over time), a `d > 0` is used to remove that trend.
- However, correctly determining d is important. If d is chosen too small, the series may still be non-stationary; if d is chosen too large, important information may be lost.

3. MA (Moving Average - q): The moving average component indicates that the current value of the time series depends on past white noise terms:

$$Y_t = \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

Where:

- θ_i are the moving average coefficients.
- ε_t is white noise - $N(0, \sigma^2)$.

Meaning:

- MA helps the model learn the random shocks in the data, instead of relying solely on the trend of the data itself like AR.
- If the data shows short-term autocorrelation (a clear cutoff in the Autocorrelation Function (ACF)), an MA(q) model is used.

4. The complete formula for ARIMA(p, d, q)

$$\Phi_p(B)(1-B)^d Y_t = \Theta_q(B)\epsilon_t$$

where:

- Y_t is the value of the time series at time t.
- B is the backshift operator, with $BY_t = Y_{t-1}$.
- d is the order of differencing to make the series stationary.
- ϵ_t is white noise, with a mean of 0 and constant variance.
- $\Phi_p(B)$ is the p^{th} order autoregressive polynomial, given by:

$$\Phi_p(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p$$

- $\Theta_q(B)$ is the q^{th} order moving average polynomial, given by:

$$\Theta_q(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q$$

Application: ARIMA Model + GARCH Model

1. Stationary for Dataset

VNINDEX has ADF test statistics that are highly negative and p-values of 0.000, indicating strong evidence to reject the null hypothesis of a unit root. This means both the index is stationary at its current value, meaning that using **first-order differencing** to achieve stationarity for time series modeling like ARIMA is appropriate. From that, we would have the ***value of Intergrated - d is 1***.

- ***ARIMA(p,1,q)***

In [26]:
```
values_vnindex = database['close_pct_vnindex'].values
```

In [27]:
```
# ADF
def adf_test(series):
    result = adfuller(series)
    adf_statistic = result[0]
    p_value = result[1]
    print('ADF Statistic: %f' % adf_statistic)
    print('p-value: %f' % p_value)
    return result

print('\nADF Test for VNINDEX:')
p_value_vnindex = adf_test(values_vnindex)[1]
```

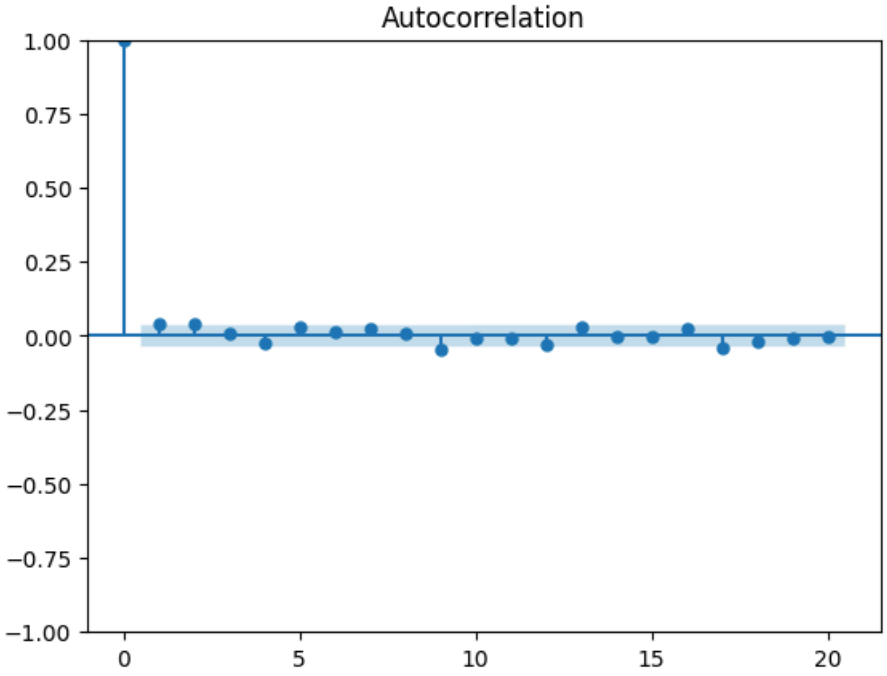
```
ADF Test for VNINDEX:
ADF Statistic: -17.942358
p-value: 0.000000
```

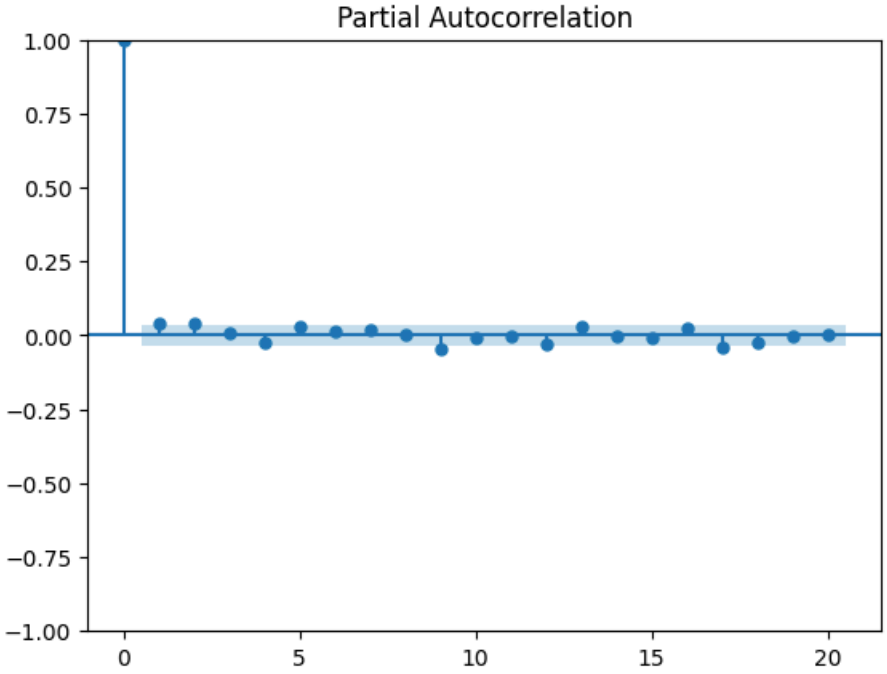
2. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)

The ACF plot shows the correlation between a time series and its own lagged values, with the horizontal axis representing the lag and the vertical axis representing the correlation coefficient. From the graph, we can see that with ***lag value at 1 and 2, the bars exceed the confidence interval***, indicating a significant correlation between the time series and its lagged variables. From that, **value of AR(p) would be 1 or 2**. Same can be observed in PACF with **MA(q) = 1 or 2**. Therefore, four options would be tested:

- **ARIMA(1,1,1)**
- **ARIMA(2,1,1)**
- **ARIMA(1,1,2)**
- **ARIMA(2,1,2)**

In [28]:
```
# plot ACF and PACF
plot_acf(values_vnindex, lags=20)
plt.show()
plot_pacf(values_vnindex, lags=20)
plt.show()
```





3. Testing for ARIMA Model

Model diagnostics

When evaluating an ARIMA model, we must check whether its residuals (errors) behave like white noise - random and uncorrelated. This ensures the model correctly captures the underlying data patterns. Here, we analyze three key diagnostic tests:

a. Ljung-Box Test (Q-Statistic)

- Checks if the residuals (errors) of the model are autocorrelated—meaning whether past errors influence future errors.
- Null hypothesis (H_0): Residuals are not autocorrelated (i.e., they are like white noise).
- Alternative hypothesis (H_1): Residuals are autocorrelated (indicating the model has not fully captured dependencies).

b. Jarque-Bera (JB) Test

- Checks whether residuals are normally distributed
- Skewness and Kurtosis
- Null hypothesis (H_0): Residuals follow a normal distribution.
- Alternative hypothesis (H_1): Residuals do not follow a normal distribution.

c. Heteroskedasticity (H) Test

- Heteroskedasticity means the variance of residuals is not constant over time. If present, model errors become unpredictable, making forecasting unreliable.
- Null hypothesis (H_0): Residual variance is constant (homoskedasticity).
- Alternative hypothesis (H_1): Residual variance changes over time (heteroskedasticity).

After running four ARIMA models, we ended up with ARIMA(2,1,1) because it had the best test results.
Result of ARIMA(2,1,1)

- **Ljung-Box Test**
  - Q-statistic = 0.00, p-value = 0.95
  - Since the p-value (0.95) is very high (>> 0.05), we fail to reject H_0, meaning that the residuals do not show significant autocorrelation.
  - ***Interpretation: The model has effectively removed any serial correlation, so it is well-specified in capturing time-dependent patterns.***
- **Jarque-Bera (JB) Test**
  - JB Statistic = 2532.91, p-value = 0.00
  - Since p-value = 0.00 (< 0.05), we reject H_0, meaning residuals are not normally distributed.
  - Skewness = -0.79 (negative means left-skewed distribution).
  - Kurtosis = 7.28 (higher than 3, indicating heavy tails or extreme values).
  - ***Interpretation***:
    - ***The residuals are not normally distributed, which might affect statistical inference (e.g., confidence intervals may be unreliable)***.
    - ***A non-normal residual distribution suggests potential outliers or extreme movements in VN-Index returns.***
  - ***Solution: a volatility model "GARCH"***
- **Heteroskedasticity (H) Test**
  - H-statistic = 1.39, p-value = 0.00
  - Since p-value = 0.00 (< 0.05), we reject H_0, meaning heteroskedasticity is present—residual variance changes over time.
  - ***Interpretation***:
    - Residuals have time-varying volatility, meaning large price movements cluster together.
    - This is common in financial time series (e.g., stock market returns).
  - ***Solution: a volatility model "GARCH"***

In [29]:
```
# ARIMA(2,1,1) model for VNINDEX
model = ARIMA(values_vnindex, order=(2, 1, 1))
model_fit = model.fit()
print(model_fit.summary())

# ARIMA(1,1,1) model for VNINDEX
```

```
# model = ARIMA(values_vnindex, order=(1, 1, 1))
# model_fit = model.fit()
# print(model_fit.summary())

# ARIMA(1,1,2) model for VNINDEX
# model = ARIMA(values_vnindex, order=(1, 1, 2))
# model_fit = model.fit()
# print(model_fit.summary())

# ARIMA(2,1,2) model for VNINDEX
# model = ARIMA(values_vnindex, order=(2, 1, 2))
# model_fit = model.fit()
# print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 2924
Model:                 ARIMA(2, 1, 1)   Log Likelihood               -4492.492
Date:                Tue, 25 Feb 2025   AIC                           8992.984
Time:                        22:07:01   BIC                           9016.905
Sample:                             0   HQIC                          9001.600
                                - 2924
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0381      0.013      3.014      0.003       0.013       0.063
ar.L2          0.0420      0.012      3.498      0.000       0.018       0.066
ma.L1         -1.0000      0.044    -22.770      0.000      -1.086      -0.914
sigma2         1.2629      0.055     22.781      0.000       1.154       1.372
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              2535.03
Prob(Q):                              0.95   Prob(JB):                         0.00
Heteroskedasticity (H):               1.39   Skew:                            -0.79
Prob(H) (two-sided):                  0.00   Kurtosis:                         7.28
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

4. GARCH(1,1) - to solve Heteroskedasticity

GARCH is applied after ARIMA because ARIMA models the conditional mean but assumes constant variance. Financial time series, however, show volatility clustering, where high volatility follows high volatility. **GARCH captures this by modeling ARIMA's residuals**, separating predictable trends from unpredictable volatility. This improves risk assessment and forecasting.

Since the p-value after applying GARCH remains extremely small but shows some improvement compared to ARIMA without GARCH (3.95e-66 < 5.15e-66), this indicates that **heteroskedasticity is still present in the residuals**. In other words, the GARCH(1,1) model has not fully captured the time-varying volatility in the dataset. To address this, we will apply a **Student's t-distribution**, which is expected to provide a better fit by accounting for the presence of heavy tails in financial time series data.

In [31]:
```
result_arima, result_garch = fit_arima_garch(values_vnindex, arima_order=(2, 1, 1), garch_order=(1, 1), dist='normal')

 # Show p-values after GARCH
garch_residuals = result_garch.resid  # Extract residuals from GARCH
arch_test_after_garch = het_arch(garch_residuals)

print(result_garch.summary())
print(f"P-value after GARCH: {arch_test_after_garch[1]}")
```

```
P-value from ARCH Test: 5.155940962705091e-66
Significant heteroskedasticity detected. Applying GARCH model...
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -4159.01
Distribution:                  Normal   AIC:                           8326.03
Method:            Maximum Likelihood   BIC:                           8349.95
                                        No. Observations:                 2924
Date:                Tue, Feb 25 2025   Df Residuals:                     2923
Time:                        22:07:03   Df Model:                            1
                                Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu         -7.0817e-03  1.788e-02     -0.396      0.692 [-4.212e-02,2.795e-02]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0303  1.006e-02      3.013  2.590e-03 [1.059e-02,5.003e-02]
alpha[1]       0.1168  1.699e-02      6.872  6.333e-12   [8.348e-02,  0.150]
beta[1]        0.8651  2.035e-02     42.512      0.000     [  0.825,  0.905]
============================================================================

Covariance estimator: robust
P-value after GARCH: 3.9460989704896473e-66
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -4159.01
Distribution:                  Normal   AIC:                           8326.03
Method:            Maximum Likelihood   BIC:                           8349.95
                                        No. Observations:                 2924
Date:                Tue, Feb 25 2025   Df Residuals:                     2923
Time:                        22:07:03   Df Model:                            1
                                Mean Model
==============================================================================
                 coef    std err          t      P>|t|       95.0% Conf. Int.
------------------------------------------------------------------------------
mu         -7.0817e-03  1.788e-02     -0.396      0.692 [-4.212e-02,2.795e-02]
                           Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0303  1.006e-02      3.013  2.590e-03 [1.059e-02,5.003e-02]
alpha[1]       0.1168  1.699e-02      6.872  6.333e-12   [8.348e-02,  0.150]
beta[1]        0.8651  2.035e-02     42.512      0.000     [  0.825,  0.905]
============================================================================

Covariance estimator: robust
P-value after GARCH: 3.9460989704896473e-66
```

5. GARCH with Student's t-distribution - to solve Heteroskedasticity

Even though the GARCH(1,1) model with a Student's t-distribution improved the model fit—as indicated by lower AIC/BIC values and a better log-likelihood—it still did ***not completely eliminate the volatility clustering effect***. However, it performed significantly better than the standard GARCH model with a normal distribution. This suggests that while the Student's t-distribution captures heavy tails more effectively, a more flexible model, such as GARCH with asymmetric effects (e.g., EGARCH or GJR-GARCH), may be needed to fully address the heteroskedasticity issue.

However, in many cases, due to ***the nature of the data, especially stock price or indices — such as high volatility influenced by various external factors - heteroskedasticity remains significant, making it difficult for the model to produce accurate predictions***. Even when applying advanced forecasting models and carefully analyzing residuals, it is often impossible to completely eliminate this issue. In such situations, we should consider incorporating additional variables that may impact the fluctuations of the main variable, rather than relying solely on its own lags and white noise, as models like ARIMA and GARCH typically do.

In [32]:
```
result_arima, result_garch = fit_arima_garch(values_vnindex, arima_order=(2, 1, 1), garch_order=(1, 1), dist='t')

 # Show p-values after GARCH
garch_residuals = result_garch.resid  # Extract residuals from GARCH
arch_test_after_garch = het_arch(garch_residuals)

print(result_garch.summary())
print(f"P-value after GARCH: {arch_test_after_garch[1]}")
```

```
P-value from ARCH Test: 5.155940962705091e-66
Significant heteroskedasticity detected. Applying GARCH model...
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                           y   R-squared:                       0.000
Mean Model:                   Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           GARCH   Log-Likelihood:               -4026.18
Distribution:      Standardized Student's t   AIC:                           8062.36
Method:                 Maximum Likelihood   BIC:                           8092.26
                                             No. Observations:                2924
Date:                      Tue, Feb 25 2025   Df Residuals:                    2923
Time:                              22:07:05   Df Model:                           1
                            Mean Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
mu             0.0368  1.494e-02      2.467  1.363e-02 [7.573e-03,6.613e-02]
                          Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0485  1.486e-02      3.265  1.094e-03 [1.939e-02,7.764e-02]
alpha[1]       0.1405  2.185e-02      6.430  1.272e-10   [9.768e-02,  0.183]
beta[1]        0.8286  2.820e-02     29.378 1.038e-189     [  0.773,  0.884]
                          Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             4.8616      0.441     11.030  2.738e-28 [  3.998,  5.725]
========================================================================

Covariance estimator: robust
P-value after GARCH: 2.420546756970962e-65
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                           y   R-squared:                       0.000
Mean Model:                   Constant Mean   Adj. R-squared:                  0.000
Vol Model:                           GARCH   Log-Likelihood:               -4026.18
Distribution:      Standardized Student's t   AIC:                           8062.36
Method:                 Maximum Likelihood   BIC:                           8092.26
                                             No. Observations:                2924
Date:                      Tue, Feb 25 2025   Df Residuals:                    2923
Time:                              22:07:05   Df Model:                           1
                            Mean Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
mu             0.0368  1.494e-02      2.467  1.363e-02 [7.573e-03,6.613e-02]
                          Volatility Model
============================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
----------------------------------------------------------------------------
omega          0.0485  1.486e-02      3.265  1.094e-03 [1.939e-02,7.764e-02]
alpha[1]       0.1405  2.185e-02      6.430  1.272e-10   [9.768e-02,  0.183]
beta[1]        0.8286  2.820e-02     29.378 1.038e-189     [  0.773,  0.884]
                          Distribution
========================================================================
                 coef    std err          t      P>|t|  95.0% Conf. Int.
------------------------------------------------------------------------
nu             4.8616      0.441     11.030  2.738e-28 [  3.998,  5.725]
========================================================================

Covariance estimator: robust
P-value after GARCH: 2.420546756970962e-65
```

6. Predictions: ARIMA + GARCH with Student's t-distribution

`fit_arima_garch` function combines ARIMA and GARCH models for forecasting (mentioned in "*Set up libraries*"). As described in the "*Introduction to ARIMA*", this model relies on two key components: AutoRegressive (AR), which uses past values of the series, and Moving Average (MA), which captures past residuals or white noise. The forecast function from the `statsmodels` library applies Maximum Likelihood Estimation (MLE) to fit the ARIMA model by maximizing the likelihood of observing the actual data.

However, due to the presence of heteroskedasticity (changing variance over time), we incorporate GARCH (Generalized Autoregressive Conditional Heteroskedasticity). The GARCH model focuses on analyzing and modeling residuals, also using MLE to improve forecasting accuracy by capturing volatility dynamics.

Unlike machine learning models, such as Random Forest Regression, ARIMA+GARCH does not require splitting the dataset into training and testing sets. Instead, it makes an `adjusted_forecast` (`y_predicted`) based on historical data (`y_train`) without needing `y_test` . To ensure consistency when comparing results with Random Forest Regression, we use `y_train` as an input for this model instead of whole dataset (values_vnindex).

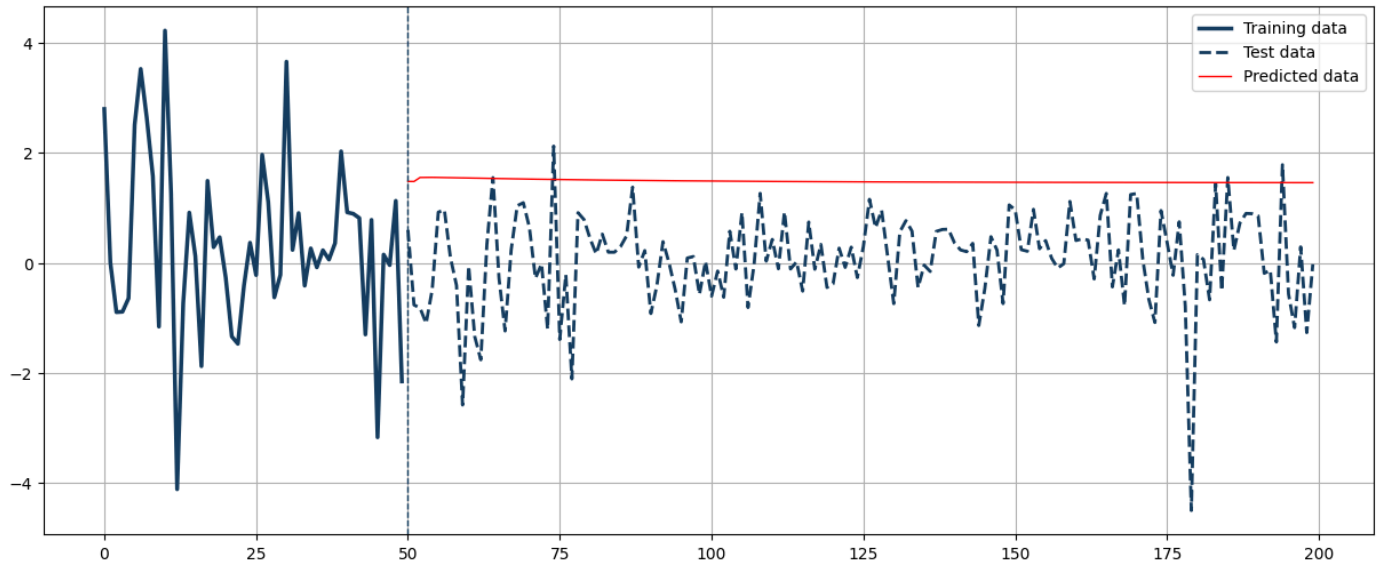
In [ ]:
```
result_arima, result_garch = fit_arima_garch(y_train, arima_order=(2, 1, 1), garch_order=(1, 1), dist='t') # not show c
```

In [34]:
```
forecast_horizon = 485  # Define number of future steps to predict, choose 485 to match the test set

# forecast arima
arima_forecast = result_arima.forecast(steps=forecast_horizon)

# forecast garch
garch_forecast_df = result_garch.forecast(horizon=forecast_horizon).variance
# Extract the last row (as a Series) and convert variance to standard deviation
garch_forecast = np.sqrt(garch_forecast_df.iloc[-1, :].values)
```

Below performance evaluation and Descriptive statistics
The ARIMA + GARCH model's performance metrics indicate certain limitations in capturing the true dynamics of the VNINDEX returns. An accuracy test result of 55.67% and an RMSE of 1.707 suggest that while the model has some predictive capability for long-term, it may not be robust enough for precise forecasting the fluctuation in short-term. Notably, the negative correlation of -0.037 between out-of-sample predictions and actual test data implies a potential disconnect between the model's outputs and the observed market behavior.

The forecast plot's linear appearance reflects the model's reliance on linear regression assumptions, which may not fully capture the data's volatility over time. Descriptive statistics further highlight this discrepancy: the predicted data shows a mean of 1.4635 with a standard deviation of 0.0196, indicating minimal variability, whereas the actual data has a mean of 0.0313 and a much higher standard deviation of 0.9287, reflecting significant fluctuations. This contrast suggests that while ARIMA + GARCH models are adept at modeling linear trends, they may fall short in accounting for the complex, non-linear patterns inherent in financial time series data.

In [35]:
```
# Calculate adjusted_forecast
adjusted_forecast = arima_forecast + garch_forecast
#reshape adjusted_forecast to 2D array
adjusted_forecast = np.reshape(adjusted_forecast, (-1, 1))
# plotting
plot_train_test_values(window=200, train_window=50,
                       y_train=y_train, y_test=y_test, y_predicted=adjusted_forecast)
```



In [36]:
```
# Performance evaluation
print('---')
print('Accuracy Test = ', round(accuracy(adjusted_forecast, y_test), 2), '%')
print('RMSE Test = ', round(np.sqrt(mean_squared_error(adjusted_forecast, y_test)), 10))
print('Correlation Out-of-Sample Predicted/Test = ', round(np.corrcoef(np.reshape(adjusted_forecast, (-1)), np.reshape(
print('Model Bias = ', round(model_bias(adjusted_forecast), 2))
print('---')
```

```
---
Accuracy Test =  55.67 %
RMSE Test =  1.7069870448
Correlation Out-of-Sample Predicted/Test =  -0.037
Model Bias =  inf
---
```

In [37]:
```
# Desciptive statistics
# Assuming y_predicted is a numpy array
y_predicted_series = pd.Series(adjusted_forecast.flatten())
print('-----Prediction')
print(y_predicted_series.describe())
print('-----Actual')
# Assuming y_predicted is a numpy array
y_test_series = pd.Series(y_test.flatten())
print(y_test_series.describe())
```

```
-----Prediction
count   485.0000
mean      1.4635
std       0.0196
min       1.4548
25%       1.4548
50%       1.4550
75%       1.4600
max       1.5535
dtype: float64
-----Actual
count   485.0000
mean      0.0313
std       0.9287
min      -4.6992
25%      -0.4104
50%       0.0964
75%       0.5560
max       3.4444
dtype: float64
```

3. Comparison: Econometrics Method and Machine Learning Method

Set of metrics to compare ARIMA+GARCH and Random Forest models, especially for financial time series forecasting:
(Detail about formula mentioned in "Set up libraries")

A. R-Squared (Coefficient of Determination)

R-squared measures the proportion of the variance in the dependent variable (real returns) that is predictable from the independent variables (past returns, etc.). A higher R-squared suggests a better fit. While not perfect for time series (due to potential spurious regressions), it still provides a general sense of how well the model captures the overall variability.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

B. Predictive Accuracy Metrics (MSE, MAE, MAPE)

These metrics evaluate the magnitude of the errors between the predicted and actual returns.

- **MSE (Mean Squared Error)**:
  - Penalizes larger errors more heavily due to the squaring. Sensitive to outliers.
  - $MSE = \frac{1}{N}\sum_{i=1}^{N}(r_i^p - r_i^r)^2$
- **MAE (Mean Absolute Error)**:

- Treats all errors equally. Less sensitive to outliers.
- $MAE = \frac{1}{N}\sum_{i=1}^{N}|r_i^p - r_i^r|$

- **MAPE (Mean Absolute Percentage Error)**:
  - Expresses errors as a percentage of the actual returns. Useful for comparing models across different scales, but can be problematic if actual returns are close to zero.
  - $MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{r_i^p - r_i^r}{r_i^r}\right| \times 100\%$

C. Directional Accuracy

In financial trading, often the direction of the price movement (up or down) is more important than the exact magnitude. Directional accuracy measures how often the model correctly predicts the direction of the return.

$$Accuracy = \frac{\sum_{i=1}^{N} I(\text{sign}(r_i^p) = \text{sign}(r_i^r))}{N} \times 100\%$$

D. Directional Bias

This metric reveals if the model has a tendency to be overly bullish or bearish. A bias can lead to unbalanced trading signals and potentially significant losses. It helps identify if the model is systematically over- or under-predicting positive or negative returns.

$$Bias = \frac{\sum_{i=1}^{N} I(r_i^p > 0)}{\sum_{i=1}^{N} I(r_i^p < 0)}$$

In [38]:
```
# ARIMA + GARCH Model
r_squared_arima_garch = r_squared(adjusted_forecast, y_test)
mean_squared_error_arima_garch = predictiveaccuracy.mean_squared_error(adjusted_forecast, y_test)
mean_absolute_error_arima_garch = predictiveaccuracy.mean_absolute_error(adjusted_forecast, y_test)
mean_absolute_percentage_error_arima_garch = predictiveaccuracy.mean_absolute_percentage_error(adjusted_forecast, y_tes
accuracy_arima_garch = accuracy(adjusted_forecast, y_test)
bias_arima_garch = model_bias(adjusted_forecast)

print("ARIMA + GARCH Model")
print("R-squared:", r_squared_arima_garch)
print("Mean Squared Error:", mean_squared_error_arima_garch)
print("Mean Absolute Error:", mean_absolute_error_arima_garch)
print("Mean Absolute Percentage Error:", mean_absolute_percentage_error_arima_garch)
print("Accuracy:", accuracy_arima_garch)
print("Bias:", bias_arima_garch)
```

```
ARIMA + GARCH Model
R-squared: -1640.2364071629095
Mean Squared Error: 2.9124576127707096
Mean Absolute Error: 1.470220250450455
Mean Absolute Percentage Error: 1587.5242122716486
Accuracy: 55.670103092783506
Bias: inf
```

In [39]:
```
# RandomForest Regression
r_squared_rf = r_squared(y_predicted, y_test)
mean_squared_error_rf = predictiveaccuracy.mean_squared_error(y_predicted, y_test)
mean_absolute_error_rf = predictiveaccuracy.mean_absolute_error(y_predicted, y_test)
mean_absolute_percentage_error_rf = predictiveaccuracy.mean_absolute_percentage_error(y_predicted, y_test)
accuracy_rf = accuracy(y_predicted, y_test)
bias_rf = model_bias(y_predicted)

print("RandomForest Regression Model")
print("R-squared:", r_squared_rf)
print("Mean Squared Error:", mean_squared_error_rf)
print("Mean Absolute Error:", mean_absolute_error_rf)
print("Mean Absolute Percentage Error:", mean_absolute_percentage_error_rf)
print("Accuracy:", accuracy_rf)
print("Bias:", bias_rf)
```

```
RandomForest Regression Model
R-squared: -521.1068536990434
Mean Squared Error: 0.9265052091819735
Mean Absolute Error: 0.6954713133689381
Mean Absolute Percentage Error: 221.45946356094046
Accuracy: 50.72164948453608
Bias: 1.224770642201835
```

Metric	ARIMA + GARCH	RandomForest Regression
R-squared	-1640.2364	-521.1069
Mean Squared Error (MSE)	2.9124	0.9265
Mean Absolute Error (MAE)	1.4702	0.6955
Mean Absolute Percentage Error (MAPE)	1587.52421	221.4595
Accuracy (%)	55.67	50.72
Bias	inf	1.2248

The table compares the performance of two forecasting models — ARIMA + GARCH and RandomForest Regression — across various metrics. Notably, the R-squared values for both models are negative (-1640.2364 for ARIMA + GARCH and -521.1069 for RandomForest), indicating that **R-squared may not be a reliable metric** here. Negative R-squared values typically suggest that the models perform worse than a simple mean-based baseline, which frequently occurs in time series forecasting when the data exhibits high volatility or non-linear patterns that linear metrics like R-squared fail to capture effectively. Additionally, the low R-squared values are partly due to the models relying entirely on a single variable. This is something I must acknowledge, as the models are trained and making forecasts solely based on historical price data. Such an approach inherently limits their ability to capture external factors or complex market dynamics, making them highly sensitive to past fluctuations and reducing their overall predictive power.

Instead, **metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE) offer more meaningful insights**. RandomForest Regression outperforms ARIMA + GARCH across these metrics, with a significantly lower MSE (0.9265 vs. 2.9125), MAE (0.6955 vs. 1.4702), and MAPE (221.4595 vs. 1587.5242), as well as a slightly higher accuracy (50.72% vs. 55.67%). **This suggests that RandomForest provides better predictive power and precision in this context**.

The outperformance of machine learning models like RandomForest Regression can be explained by several key factors.
(1) ARIMA and GARCH models rely heavily on strict statistical assumptions, such as stationarity and specific error distributions, which require rigorous pre-testing and data transformation. If these conditions are not fully met, their performance can degrade significantly. In contrast, RandomForest, a machine learning model, does not impose such prerequisites and excels at capturing non-linear relationships and complex patterns in data, making it more adaptable for forecasting tasks with noisy or irregular time series.

Additionally, ***(2) RandomForest leverages ensemble learning, reducing overfitting and improving generalization, whereas ARIMA + GARCH may struggle with out-of-sample predictions when underlying assumptions falter***. These factors collectively position RandomForest as a more optimal choice for forecasting in scenarios where econometric models face limitations.

Moreover, ***econometric models like ARIMA/GARCH are traditionally used for analyzing relationships between economic variables and testing hypotheses rather than being specialized for forecasting, as is the case with machine learning models***. This fundamental distinction further explains why RandomForest outperforms ARIMA + GARCH in predictive tasks where complex and non-linear dynamics are at play.

Key insights

RandomForest Regression outperforms ARIMA + GARCH across most metrics (lower MSE, MAE, MAPE, and higher accuracy), highlighting its greater accuracy and robustness. Its advantage stems from flexibility in handling non-linear data without strict statistical assumptions, unlike ARIMA + GARCH, which may falter if pre-validation criteria are unmet.

IV. FORECAST 2025 VNINDEX

In [40]:
```
# forecast2025_vnindex_historical
forecast2025_vnindex_historical = database['close_pct_vnindex']
forecast2025_vnindex_historical = forecast2025_vnindex_historical.values
forecast2025_vnindex_historical
```

Out[40]:
```
array([ 0.21991156,  1.61948148,  1.90818195, ...,  0.29257244,
       -0.93632371,  0.41084179])
```

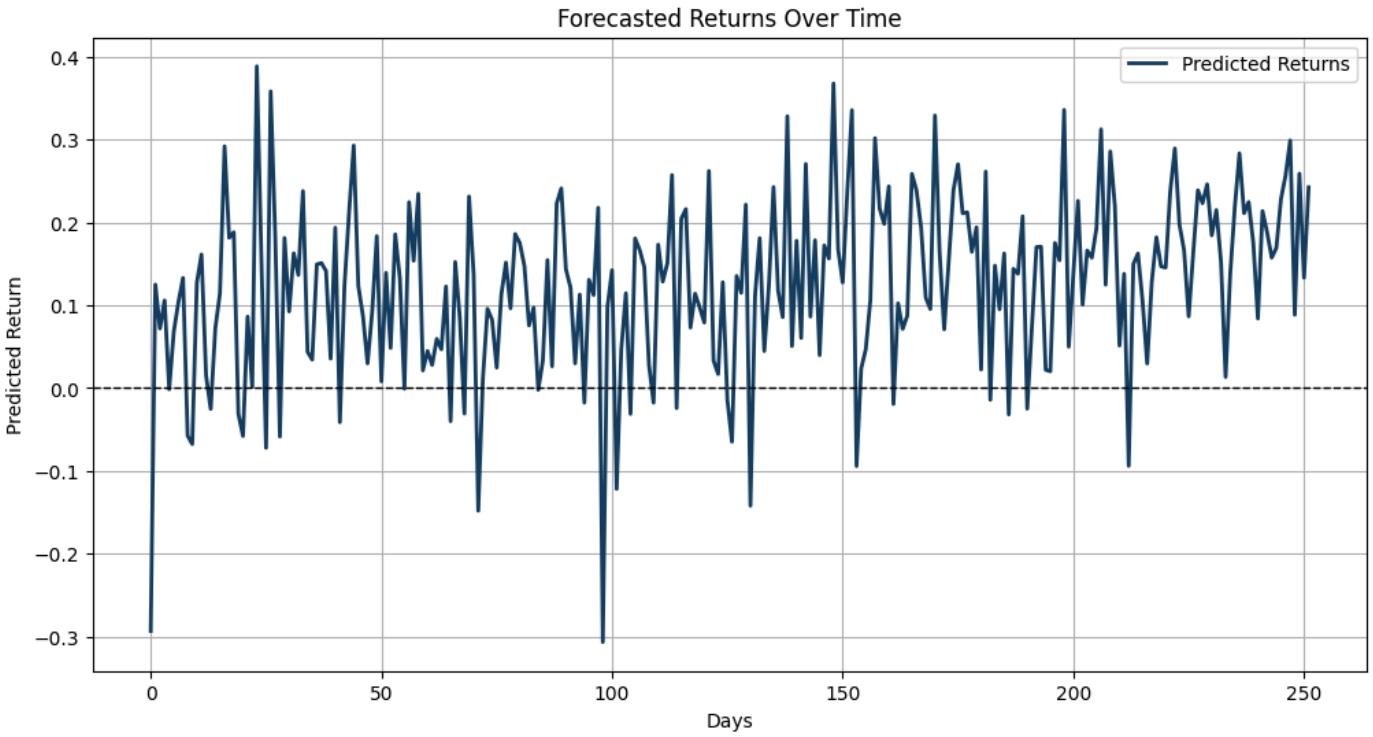
Descriptive statistics

The forecast for VNINDEX in 2025 suggests a stronger and more stable market performance compared to 2024. The expected mean return for 2025 is 9.93%, significantly higher than 2024's 2.73%, indicating a more optimistic outlook. Additionally, the standard deviation for 2025 is much lower (0.0857 vs. 0.8582 in 2024), suggesting reduced market volatility and more consistent returns. Unlike 2024, which experienced extreme fluctuations (ranging from -4.6992 to 2.3960), the forecasted 2025 range is much narrower (-0.1804 to 0.3464), indicating fewer market shocks. Furthermore, the median return of 0.1010 in 2025 surpasses 2024's 0.0603, reinforcing the expectation of stronger and steadier growth. While these projections indicate a promising market outlook, real-world factors such as macroeconomic conditions, global policies, and investor sentiment will ultimately determine whether these forecasts materialize.

In [45]:
```
# Descriptive statistics
# Assuming y_predicted is a numpy array
print('----Forecast 2025----')
predicted_return_series = pd.Series(predicted_returns.flatten())
print(predicted_return_series.describe())

# Assuming y_test is a numpy array
forecast2025_vnindex_historical_last_252 = forecast2025_vnindex_historical[-252:]
historical_return_series = pd.Series(forecast2025_vnindex_historical_last_252.flatten())
print('-----Historical 2024----')
print(historical_return_series.describe())
```

```
----Forecast 2025----
count   252.0000
mean      0.1243
std       0.1057
min      -0.3066
25%       0.0573
50%       0.1358
75%       0.1880
max       0.3882
dtype: float64
-----Historical 2024----
count   252.0000
mean      0.0273
std       0.8582
min      -4.6992
25%      -0.3806
50%       0.0603
75%       0.4667
max       2.3960
dtype: float64
```

In [46]:
```
# Plot forecast 2025
import numpy as np
import matplotlib.pyplot as plt

def plot_forecast(predicted_returns):
    """Plots the predicted returns over the forecast horizon."""
    plt.figure(figsize=(12, 6))
    plt.plot(predicted_returns, label="Predicted Returns", color="#143D60", linewidth=2)
    plt.axhline(y=0, color="black", linestyle="--", linewidth=1)  # Đường tham chiếu mức 0%

    plt.xlabel("Days")
    plt.ylabel("Predicted Return")
    plt.title("Forecasted Returns Over Time")
    plt.grid(True)
    plt.legend()
    plt.show()

# Gọi hàm để vẽ
plot_forecast(predicted_returns)
```



In [44]:
```
# Forecast_horizon=252 from historical values as input
x_input = np.array(forecast2025_vnindex_historical[-num_lags:]).reshape(1, -1)

forecast_horizon = 252
predicted_returns = []

for _ in range(forecast_horizon):
    # Predict the next return value
    predicted_return = model.predict(x_input)[0]
    predicted_returns.append(predicted_return)

    # Update the input for the next step
    x_input = np.roll(x_input, -1)  # Shift values to the left
    x_input[0, -1] = predicted_return  # Add the new predicted value

# Convert the list to a NumPy array
predicted_returns = np.array(predicted_returns)
# Tính trung bình cộng
mean_predicted_return = np.mean(predicted_returns)

# In kết quả
print(f"Mean of Predicted Returns: {mean_predicted_return:.2%}")
```

```
Mean of Predicted Returns: 12.43%
```

The RandomForest Regression model forecasts a 12.43% increase in VNINDEX from the current price level until the end of 2025. This projection provides a relatively positive outlook for the market. The dataset used for model training consists of 500 daily returns tracked retrospectively from early 2025. Given that this dataset exhibits no clear trend (sideways movement) and has yet to break through the resistance zone around 1,280 points, ***the forecast aligns with the expectation that a strong momentum shift will be required for a significant breakout.*** If positive signals emerge, the market could potentially surge toward the 1,500-point threshold.

The model predicts a strong upward trend, with an expected 12.43% growth, surpassing the initial 7-8% target required to push VNINDEX past 1,400 points. This remains a positive signal, reinforced by historical trends and overall market expectations for an uptrend in the coming year. While this projection exceeds the original forecast range, it is important to note that the model is based solely on past data. Sustained growth momentum will likely depend on a combination of technical indicators and fundamental economic factors, including Vietnam's macroeconomic performance. A preliminary analysis of these factors is outlined in the "OVERVIEW" section.

However, while such expectations are optimistic, it is equally important to objectively consider potential risks and unforeseen events that could hinder growth. (1) One key concern is the political and economic landscape following Donald Trump's election

victory, particularly his administration's imposition of tariffs on Vietnamese exports. As of February 25, 2025, U.S. President Donald Trump has announced potential tariffs on Vietnamese exports, which could impact over $142 billion worth of goods, representing about 30% of Vietnam's GDP. Additionally, there has been a surge in Chinese investments into Vietnam, as companies seek to circumvent U.S. tariffs. This trend has raised concerns about potential U.S. retaliation, further complicating Vietnam's trade dynamics. Such measures could negatively impact Vietnam's trade surplus and, consequently, the country's economic stability. Secondly, (2) the strength of the Vietnamese currency (VND) remains a critical factor, as a high exchange rate could lead to an outflow of Foreign Direct Investment (FDI), redirecting capital toward more established developed markets.

While growth remains the anticipated scenario, multiple possible market outcomes exist. Given inherent uncertainties, the best approach is to assess the probability of each scenario and make investment decisions accordingly, balancing expected returns with an appropriate level of risk management.