JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Unlocking Stock Market Trends: Leveraging Machine Learning and Time Series Analysis

[1]Chethan T S, [2]Abheesh Puthukkudy

[1 2]Students MTech in Artificial Intelligence
[1 2]REVA Academy for Corporate Excellence (RACE),
[1 2]REVA University, Bengaluru, India

Abstract: The value of a company's stock, which can increase along with the cost of an individual share, is the best measure of its success. Understanding the dynamics of stock market trends, seasonality, and noise is crucial for investors to make informed decisions. In this study, we aim to compare the performance of five different stocks using three distinct analysis methods: Linear Regression, Grid Search CV, and Extreme Learning Machine (ELM). The volatility of stock prices poses a challenge for predicting future values, prompting the exploration of machine learning algorithms as potential solutions. Leveraging Python and time series analysis, we are developing a stock price prediction platform based on historical data. Our primary objective is to enhance the precision of stock price forecasts by integrating linear regression models with time series analysis, considering temporal dynamics. By adjusting the dataset used for training linear regression models, we aim to improve accuracy. Ultimately, our research aims to demonstrate the efficacy of combining linear regression and time series analysis as the most efficient approach for stock market forecasting.

Index Terms **- Component, Machine Learning, Linear Regression, GridSearchCV, ELM, Time Series Analysis, Python, Django framework, Yahoo Finance.**

I. INTRODUCTION

A regulated market where investors can buy or sell equities publicly or privately is the stock market. The stock market serves as a pivotal platform for investors to engage in buying and selling equities, offering both public and private avenues for investment. With businesses frequently turning to the stock market for financing opportunities, it remains a preferred choice for investors looking to expand their portfolios. Informed investment decisions often rely on forecasts derived from historical market trends, a practice that has become increasingly vital given the dynamic nature of the stock market. While traditional methods like linear regression provide valuable insights into the relationship between independent and dependent variables, they may fall short in capturing the intricate temporal dynamics inherent in stock price movements.

Recognizing the need to enhance predictive modeling in the stock market domain, our study integrates time series analysis with linear regression and Extreme Learning Machine (ELM). The rationale behind selecting these distinct machine learning techniques lies in their complementary strengths and capabilities. Linear regression offers a straightforward approach to modeling linear relationships, while ELM presents a novel paradigm with its single-hidden layer feedforward neural network architecture, characterized by randomly generated input weights and analytically derived output weights. Moreover, GridSearchCV with XGBoost is included to optimize hyperparameters efficiently, enhancing model performance and generalization.

In this endeavor, we leverage machine learning techniques, including linear regression, ELM, and time series analysis, alongside Python, Django framework, and data from Yahoo Finance, to develop a robust stock price prediction website. Our objective is to furnish investors and traders with accurate predictions of future stock prices, integrating historical trends and temporal dynamics. By employing an integrated approach, we aim to empower stakeholders to make well-informed decisions, thereby mitigating risks and optimizing investment portfolios. Selected Stocks are Tesla, Amazon, Ford, Walmart and NVDA.

2. METHODOLOGY

The project at hand uses Python tools and libraries like Scikit-learn and Numpy to implement the data and predict stock values using the linear regression approach in machine learning.

2.1 Comparison of Linear Regression, GridSearchCV with XGBoost, and ELM

Linear Regression, GridSearchCV with XGBoost, and Extreme Learning Machine (ELM). These techniques are widely employed for predictive modeling and analysis across various domains, including finance, healthcare, and engineering. The comparison encompasses critical aspects such as the operating principle, model complexity, mathematical representation, training speed, generalization capabilities, and implementation steps for each technique as below in Table 2.1.

Understanding the differences and nuances among these techniques is crucial for practitioners and researchers seeking to apply machine learning methods effectively to solve real-world problems. By elucidating the distinctive characteristics and mathematical foundations of Linear Regression, GridSearchCV with XGBoost, and ELM, this comparison aims to provide insights into their strengths, limitations, and suitability for different applications.

Table 2.1: Comparison of Machine learning Models for stock prediction

Aspect	Linear Regression	GridSearchCV with XGBoost	ELM (Extreme Learning Machine)
Operating Principle	Constructs the relationship between independent and dependent variables to determine the best fit line, using labeled data (supervised learning).	Optimizes hyperparameters of XGBoost through cross-validation to find the best combination, enhancing model performance.	Single-hidden layer feedforward neural network with randomly generated input weights and analytically derived output weights, providing fast learning and good generalization.
Model Complexity	Simple linear relationship between input and output variables.	Complexity depends on the chosen hyperparameters and the structure of the XGBoost model.	Typically less complex compared to traditional neural networks as it randomly initializes input weights and analytically derives output weights.
Mathematical Representation	$y = m_1x_1 + m_2x_2 + \ldots + m_kx_k + c + e$	Ensemble learning method that builds multiple decision trees sequentially and combines their predictions.	$f(x) = \sum_{i-1}^{N} \beta_i \cdot g(w_i \cdot x + b_i)$
Description	Represents the linear relationship between input variables $(x_1, x_2, \ldots, x_k)$ and output variable (y) where $m_1, m_2, \ldots, m_k$ are the coefficients, c is the intercept, and e is the error term.	Ensemble learning method that combines predictions from multiple decision trees, each trained on different subsets of the data, to improve predictive accuracy.	Represents the output of the ELM model $f(x)$ which is a weighted sum of hidden layer outputs, where βi are the output weights, g is the activation function, wi are the input weights, and bi are the biases.
Training Speed	Generally faster training due to simpler computations.	May have longer training times due to the exhaustive search over hyperparameters and training of multiple decision trees.	Known for its fast learning speed as it directly calculates the output weights without iterative optimization.

The table below Table 2.2 provides a concise comparison of three prominent machine learning models used in stock price prediction: Grid Search CV with XGBoost, Linear Regression, and Extreme Learning Machine (ELM). Each model is evaluated across key criteria including complexity, performance, interpretability, scalability, feature engineering, robustness to noise, handling non-linearity, hyperparameter tuning, training time, prediction time, dataset size, and dependency on data quality. This comparison aims to assist in understanding the relative strengths and weaknesses of each model in the context of stock market forecasting.

In this journal, the focus lies on analyzing stock market data through time series techniques. In this journal, the focus lies on analyzing stock market data through time series techniques. Rather than predicting future stock prices, the prediction tasks are conducted using machine learning algorithms such as linear regression, GridSearchCV with XGBoost, and Extreme Learning Machine (ELM), the objective of time series is to uncover patterns of trends, seasonality, and noise within the data. This analysis aims to understand the underlying dynamics of stock market behavior over time. Leveraging the seasonal decomposition method from the statsmodels library, the data is decomposed into its key components: trend, seasonality, and residual (or noise). The trend component reveals long-term shifts in stock prices, while seasonality uncovers recurring patterns like weekly or monthly cycles. Finally, the residual component captures random fluctuations or unexpected movements in stock prices. This approach provides valuable insights into the evolution of stock prices over time.

Additionally, prediction tasks are conducted using machine learning algorithms such as linear regression, GridSearchCV with XGBoost, and Extreme Learning Machine (ELM). However, the primary focus of this analysis remains on understanding the temporal dynamics of stock market data rather than predicting future prices

Table 2.2: Comparison of Machine learning Models performance

Criteria	Grid Search CV with XGBoost	Linear Regression	Extreme Learning Machine (ELM)
Complexity	High	Low	Low
Performance	High (depends on parameters)	Moderate	Moderate to High
Interpretability	Low	High	Low

Scalability	Moderate	High	High
Feature Engineering	Required	Not as much	Not as much
Robustness to Noise	Moderate	Low	Low
Handling Non-linearity	High	Low	Moderate
Hyperparameter Tuning	Required	Minimal	Minimal
Training Time	High	Low	Low
Prediction Time	Moderate	Low	Low
Dataset Size	Large	Small to Moderate	Small to Moderate
Dependency on Data Quality	High	Low	Low

2.2 Comparison of Process steps

In the realm of machine learning, the journey from raw data to actionable insights involves several crucial steps. Each step contributes to the development and refinement of predictive models, guiding the way toward accurate and reliable results. The following table outlines the key process steps involved in machine learning tasks, comparing their applicability across three different techniques: Grid Search CV with XGBoost, Linear Regression, and Extreme Learning Machine (ELM). By examining which steps are relevant to each technique, we can gain insights into their respective workflows and understand the intricacies of model development within each framework the Table 2.3 give clarity to the same

Table 2.3: Comparison of Machine learning process steps.

Step No.	Process Step	Grid Search CV with XGBoost	Linear Regression	Extreme Learning Machine (ELM)
1	Data Collection	✓	✓	✓
2	Data Preprocessing	✓		
3	Feature Engineering	✓		
4	Data Splitting	✓	✓	✓
5	Model Selection	✓	✓	✓
6	Model Training	✓	✓	✓
7	Hyperparameter Tuning	✓		
8	Model Evaluation	✓		
9	Model Validation	✓	✓	✓
10	Model Performance check	✓	✓	✓

Descriptions:

Data Collection: Gathering relevant data from various sources, such as databases or APIs.

Data Preprocessing: Cleaning, transforming, and organizing the collected data to make it suitable for analysis.

Feature Engineering: Creating new features or modifying existing ones to enhance model performance.

Data Splitting: Dividing the dataset into training and testing subsets to evaluate model performance.

Model Selection: Choosing the appropriate machine learning model based on the problem and data characteristics.

Model Training: Fitting the selected model to the training data to learn patterns and relationships.

Hyperparameter Tuning: Adjusting the model's hyperparameters to optimize its performance.

Model Evaluation: Assessing the trained model's performance using evaluation metrics.

Model Validation: Validating the model's performance on unseen data to ensure its generalization capability.

Model Performance Check: Checking the final model's performance and comparing it against predefined criteria

3. Results and Discussions

In this study, we present an analysis of model performance metrics commonly used in predictive modeling tasks, focusing on stock market prediction as our primary domain of interest. Model evaluation and validation are crucial steps in assessing the effectiveness of machine learning algorithms applied to financial forecasting.

The performance metrics under scrutiny include Model Accuracy (%), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R2). Model Accuracy provides an indication of the percentage of correct predictions made by the model, essential for evaluating the overall effectiveness of the predictive system. Meanwhile, MSE and RMSE quantify the average squared difference between predicted and actual values, offering insights into the model's prediction error. A lower MSE and RMSE signify a closer fit of the model to the observed data. Additionally, R2, also known as the coefficient of determination, assesses the proportion of variance in the dependent variable that is explained by the independent variables in the model.

Model Accuracy (%): This metric indicates the percentage of correct predictions made by the model. It measures how well the model's predictions match the actual values. Higher accuracy values indicate better performance, with 100% accuracy representing perfect predictions.

Mean Squared Error (MSE): The MSE calculates the average squared difference between the predicted values and the actual values. It provides a measure of the model's prediction error, with lower MSE values indicating better performance. MSE is sensitive to outliers because it squares the errors, giving higher weight to large errors.

Root Mean Squared Error (RMSE): RMSE is the square root of the MSE and represents the average magnitude of the errors in the predicted values. Like MSE, lower RMSE values signify better model performance. RMSE is in the same unit as the target variable, making it easier to interpret.

R-squared (R2): R-squared measures the proportion of the variance in the dependent variable (target) that is explained by the independent variables (features) in the model. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data. R2=1 indicates that the model perfectly predicts the target variable based on the features, while R2=0 means the model does not explain any variance.

Table 3.1: Comparison of Machine learning Results

Stock Name	AVG Stock Price	Model	Model Accuracy (%)	MSE	RMSE	R-squared
FORD	12.7$	GridSearchCV	93.9679386	0.0804519	0.2836405	0.9543835
		Linear Regression	96.8810527	0.0550075	0.2345369	0.9688105
		ELM	96.8810524	0.0550075	0.2345369	0.9688105
AMZN	123$	GridSearchCV	99.1335942	9.7179853	3.1173683	0.9824964
		Linear Regression	98.9919145	5.5968958	2.3657759	0.9899191
		ELM	98.9919145	5.5968958	2.3657759	0.9899191
NVD	338$	GridSearchCV	99.5954102	159.7869667	12.6406870	0.9958287
		Linear Regression	99.7732511	86.8597422	9.3198574	0.9977325
		ELM	99.7732511	86.8597424	9.3198574	0.9977325
TSLA	221$	GridSearchCV	97.0374910	56.9238220	7.5447877	0.9719488
		Linear Regression	97.4078053	52.6029312	7.2527878	0.9740781
		ELM	97.4078053	52.6029314	7.2527878	0.9740781
WMT	48.8$	GridSearchCV	98.7442843	0.2436482	0.4936074	0.9889489
		Linear Regression	99.1709382	0.1827874	0.4275364	0.9917094
		ELM	99.1709382	0.1827874	0.4275364	0.9917094

The study evaluates the performance of three machine learning models—GridSearchCV, Linear Regression, and Extreme Learning Machine (ELM)—in predicting stock prices for five major companies: Ford, Amazon (AMZN), Nvidia (NVD), Tesla (TSLA), and Walmart (WMT). GridSearchCV demonstrates moderate performance compared to Linear Regression and ELM, with ELM and Linear Regression exhibiting very similar performance levels. Notably, it is essential to acknowledge that GridSearchCV's performance could potentially be enhanced through hyperparameter tuning, which was not explored within the scope of this paper. Despite this, ELM and Linear Regression consistently perform competitively, with accuracy rates approaching or exceeding 99% in most cases. These findings underscore the efficacy of machine learning algorithms in stock market forecasting, offering valuable insights for investors navigating the complexities of financial markets. Below is the code used for this journal.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
import altair as alt
from statsmodels.tsa.seasonal import seasonal_decompose
from docx import Document
from docx.shared import Inches


class ELMRegressor:
    def __init__(self, input_size, hidden_size):
        """
        Initialize ELMRegressor object.

        Parameters:
        - input_size: int, number of input features
        - hidden_size: int, number of hidden nodes
        """
        self.input_size = input_size
        self.hidden_size = hidden_size
        # Initialize input weights and biases with random values
        self.input_weights = np.random.normal(size=[input_size, hidden_size])
        self.biases = np.random.normal(size=[hidden_size])
        self.output_weights = None

    def relu(self, x):
        """
        Rectified Linear Unit (ReLU) activation function.

        Parameters:
        - x: array-like, input data

        Returns:
        - array-like, output after applying ReLU activation
        """
        return np.maximum(x, 0)

    def hidden_nodes(self, X):
        """
        Compute hidden layer activations.

        Parameters:
        - X: array-like, input data

        Returns:
        - array-like, hidden layer activations
        """
        G = np.dot(X, self.input_weights) + self.biases
        H = self.relu(G)
        return H
```

```python
    def fit(self, X, y):
        """
        Train the ELMRegressor model.

        Parameters:
        - X: array-like, input data
        - y: array-like, target labels
        """
        H = self.hidden_nodes(X)
        self.output_weights = np.dot(np.linalg.pinv(H), y)

    def predict(self, X):
        """
        Make predictions using the trained model.

        Parameters:
        - X: array-like, input data

        Returns:
        - array-like, predicted labels
        """
        H = self.hidden_nodes(X)
        return np.dot(H, self.output_weights)

def run_model(data_path, filename):
    """
    This function runs the XGBoost model, linear regression, and ELM on a
    given dataset,prints the model accuracy, and plots the
    actual vs predicted values.

    Args:
    - data_path: The path to the data file.
    - filename: Name of the file (used for display purposes).

    Returns:
    chart: Altair chart object.
    performance_df: DataFrame containing performance metrics.
    """

    # Load the data
    data = pd.read_csv(data_path, index_col="Date", parse_dates=True)

    # Preprocess the data
    data = data.drop(['Adj Close', 'High', 'Low'], axis=1)
    new_cols = list(data.columns)
    new_cols.remove('Close')
    new_cols.append('Close')
    data = data[new_cols]
    data = data.dropna()

    # Define hyperparameter search space for XGBoost
    search_space = {
        "n_estimators": [100, 200, 500],
        "max_depth": [3, 6, 9],
    }
```

```python
    # Split the data into training and testing sets
    X = data.iloc[:, :2].values
    Y = data.iloc[:, 2].values.reshape(-1, 1)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.2,
                                                        random_state=2021)

    # Train the XGBoost model with GridSearchCV
    xgb_model = XGBRegressor(random_state=1)
    GS = GridSearchCV(estimator=xgb_model,
                      param_grid=search_space,
                      scoring=["r2", "neg_mean_squared_error"],
                      refit="r2",
                      cv=5,
                      verbose=0)
    GS.fit(X_train, Y_train)

    # Train linear regression model
    regression = LinearRegression()
    regression.fit(X_train, Y_train)

    # Train ELMRegressor
    elm = ELMRegressor(input_size=X_train.shape[1], hidden_size=100)
    elm.fit(X_train, Y_train)

    # Calculate model accuracies
    model_accuracy_xgb = GS.best_score_ * 100
    model_accuracy_regression = regression.score(X_test, Y_test) * 100
    model_accuracy_elm = r2_score(Y_test, elm.predict(X_test)) * 100

    # Calculate performance metrics for XGBoost
    y_pred_xgb = GS.best_estimator_.predict(X_test)
    mse_xgb = mean_squared_error(Y_test, y_pred_xgb)
    rmse_xgb = np.sqrt(mse_xgb)
    r2_xgb = r2_score(Y_test, y_pred_xgb)

    # Calculate performance metrics for linear regression
    y_pred_regression = regression.predict(X_test)
    mse_regression = mean_squared_error(Y_test, y_pred_regression)
    rmse_regression = np.sqrt(mse_regression)
    r2_regression = r2_score(Y_test, y_pred_regression)

    # Calculate performance metrics for ELMRegressor
    y_pred_elm = elm.predict(X_test)
    mse_elm = mean_squared_error(Y_test, y_pred_elm)
    rmse_elm = np.sqrt(mse_elm)
    r2_elm = r2_score(Y_test, y_pred_elm)

    # Calculate the Average stock value
    stock_avg = np.mean(Y_test)

    # Print performance metrics as DataFrame
    performance_df = pd.DataFrame({
        'Model': ['GridSearchCV (XGBoost)', 'Linear Regression', 'ELM'],
        'Model Accuracy (%)': [model_accuracy_xgb, model_accuracy_regression,
                               model_accuracy_elm],
        'Mean Squared Error': [mse_xgb, mse_regression, mse_elm],
        'Root Mean Squared Error': [rmse_xgb, rmse_regression, rmse_elm],
        'R-squared': [r2_xgb, r2_regression, r2_elm],
```

```python
        'Average stock price': [stock_avg, stock_avg, stock_avg]

    })


    # Perform seasonal decomposition
    decomposition = seasonal_decompose(data['Close'] - data['Open'],
                                       model='additive', period=30,
                                       extrapolate_trend='freq')

    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

# Plot seasonal decomposition
    plt.figure(figsize=(12, 8))

    plt.subplot(221)
    plt.plot(data.index, data['Close'], color='#00b8ff', label='Original')
    plt.title(f'Original of {filename}')
    plt.xlabel('Time')
    plt.xticks(rotation=90)
    plt.legend(loc='best')

    plt.subplot(222)
    plt.plot(trend, 'b', label='Trend')
    plt.title(f'Trend of {filename}')
    plt.xlabel('Time')
    plt.xticks(rotation=90)
    plt.legend(loc='best')

    plt.subplot(223)
    plt.plot(seasonal, color='#ff00ff', label='Seasonality')
    plt.title(f'Seasonality of {filename}')
    plt.xlabel('Time')
    plt.xticks(rotation=90)
    plt.legend(loc='best')

    plt.subplot(224)
    plt.plot(residual, 'y', label='Residuals')
    plt.title(f'Residuals of {filename}')
    plt.xlabel('Time')
    plt.xticks(rotation=90)
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()
    # Plot the actual vs predicted values
    plt.figure(figsize=(10, 6))
    plt.plot(Y_test, label="Actual")
    plt.plot(y_pred_xgb, label="Predicted (XGBoost)")
    plt.plot(y_pred_regression, label="Predicted (Linear Regression)")
    plt.plot(y_pred_elm, label="Predicted (ELM)")
    plt.title(f"{filename} Test Result Vs Actual Data")
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.legend()
    plt.show()
```

```python
    # Create an alt.Chart object with filename as title
    chart = alt.Chart(pd.DataFrame({'Y_test': Y_test.flatten(),
                                    'error_regression': (Y_test.flatten() -
                                                         y_pred_regression.
                                                         flatten()),
                                    'error_xgb': (Y_test.flatten() -
                                                  y_pred_xgb.flatten()),
                                    'error_elm': (Y_test.flatten() -
                                                  y_pred_elm.flatten())})).
                                                  transform_fold(
        ['error_regression', 'error_xgb', 'error_elm'],
        as_=['Error Type', 'Error']
    ).mark_bar(opacity=0.4).encode(
        x='Y_test',
        y='Error:Q',
        color='Error Type:N'
    ).properties(
        title=filename,  # Set filename as title
        width=600,
        height=400
    )



    # Return the performance DataFrame
    return chart, performance_df

def save_to_word(data_paths):
    """
    Iterate through data_paths, run model, and save performance metrics
     to a Word document for each dataset.

    Args:
    - data_paths: List of file paths to the data files.
    """
    for data_path in data_paths:
        filename = data_path.split("/")[-1].split(".")[0]
        chart, performance_df = run_model(data_path, filename)

        chart.display()  # Display chart
        print( f".....****{filename}..\n")
        display(performance_df)  # Display performance_df using display() func
        print( ".....xxxxxxxxxxx...\n")
        print("\n")  # Print a new line

        # Create a new Word document
        doc = Document()

        # Add a title to the document
        doc.add_heading('Performance Metrics', level=1)

        # Add filename to the document
        doc.add_heading(f'Filename: {filename}', level=2)

        # Add performance metrics table to the document
        table = doc.add_table(rows=1, cols=len(performance_df.columns))
        hdr_cells = table.rows[0].cells
        for i, column in enumerate(performance_df.columns):
            hdr_cells[i].text = column
```

```python
        for index, row in performance_df.iterrows():
            row_cells = table.add_row().cells
            for i, value in enumerate(row):
                row_cells[i].text = str(value)

        # Save the document to the specified path
        doc.save(f"/content/drive/MyDrive/Colab Notebooks/Paper_submission/
        {filename}_performance_metrics.docx")

# Example usage:
#save_to_word(data_paths)
```

```python
data_paths = ["/content/drive/MyDrive/Colab Notebooks/Paper_submission/INTC.csv",
              "/content/drive/MyDrive/Colab Notebooks/Paper_submission/FORD.csv",
              "/content/drive/MyDrive/Colab Notebooks/Paper_submission/NVDA.csv",
              "/content/drive/MyDrive/Colab Notebooks/Paper_submission/WMT.csv",
              "/content/drive/MyDrive/Colab Notebooks/Paper_submission/TSLA.csv"]

# Example usage:
save_to_word(data_paths)
```

FORD Stock prediction

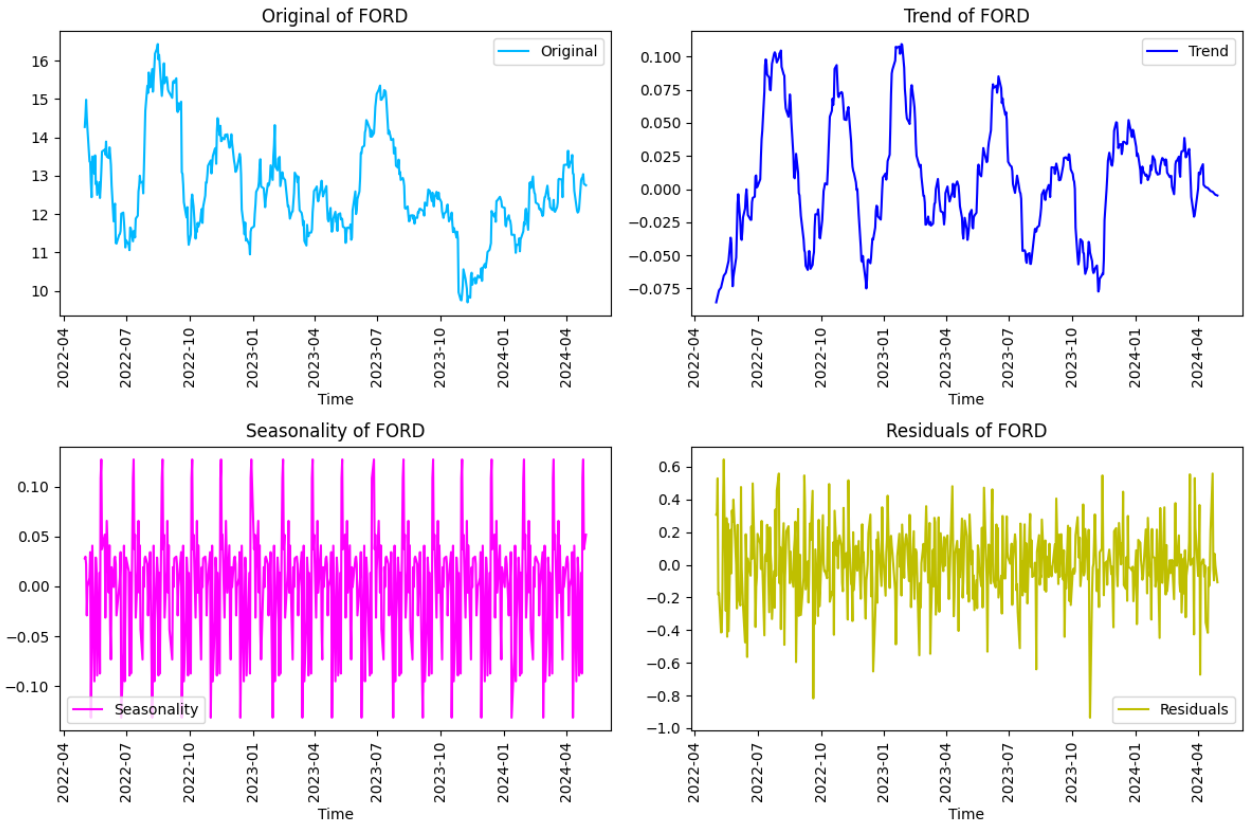

Fig.3.1 Time series decomposition results

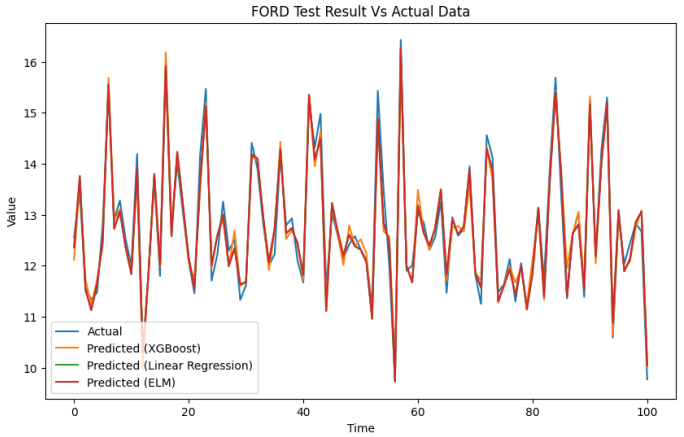

Fig.3.2 Actual Vs Predicted results

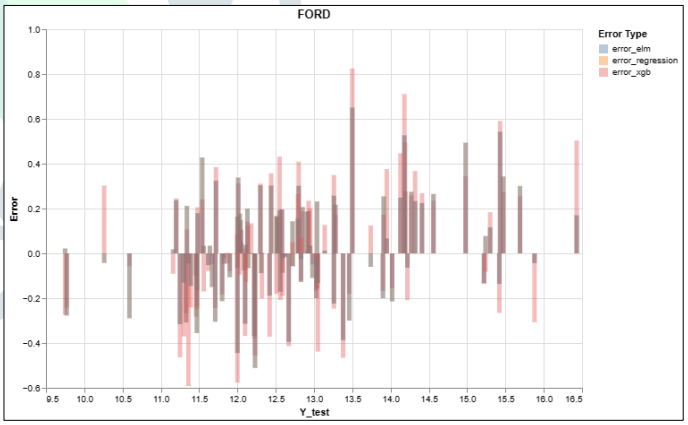

Fig.3.3 Error Vs Y_test

FORD:

GridSearchCV had an accuracy of approximately 93.97% with a Mean Squared Error (MSE) of 0.0805 and a Root Mean Squared Error (RMSE) of 0.2836. Linear Regression and ELM both achieved higher accuracies around 96.88%, with identical MSE and RMSE values.

Amazon (AMZN) Stock Prediction

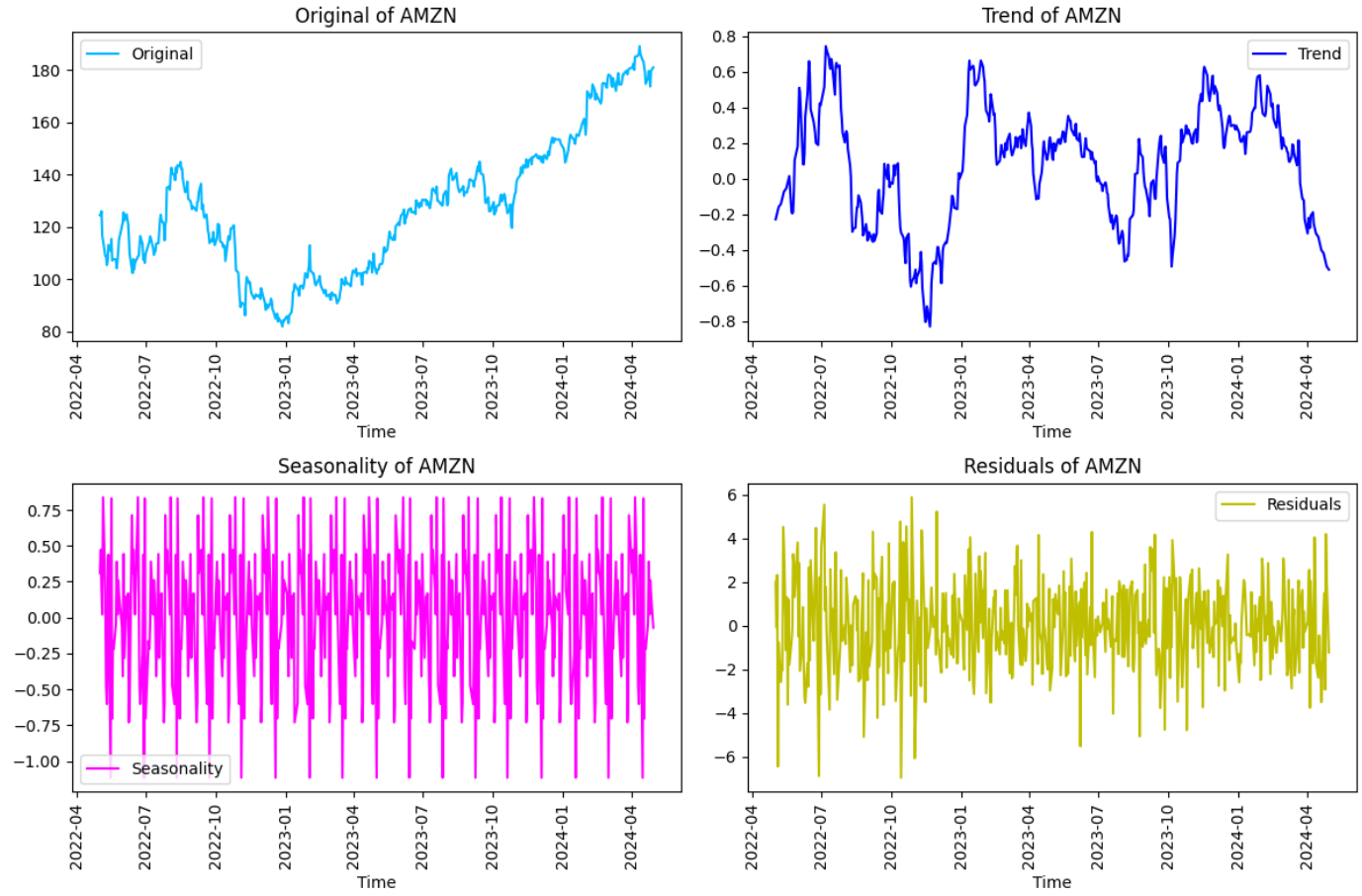

Fig.3.4 Time series decomposition results

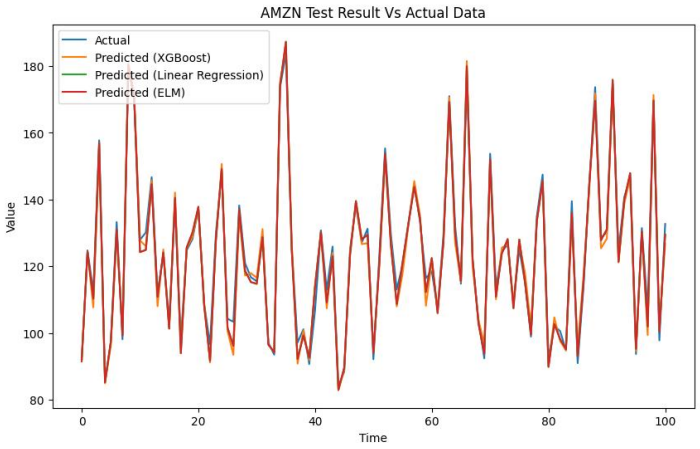

Fig.3.5 Actual Vs Predicted results

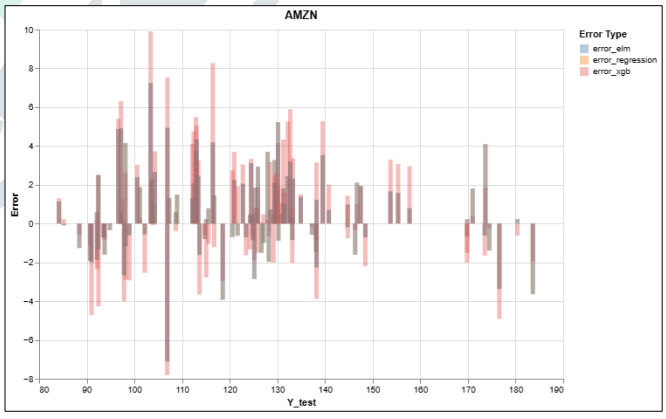

Fig.3.6 Error Vs Y_test

AMZN:

GridSearchCV achieved the highest accuracy of about 99.13% but had a significantly higher MSE and RMSE compared to the other models. Linear Regression and ELM had slightly lower accuracy but much lower MSE and RMSE values.

Nvidia (NVD) Stock prediction

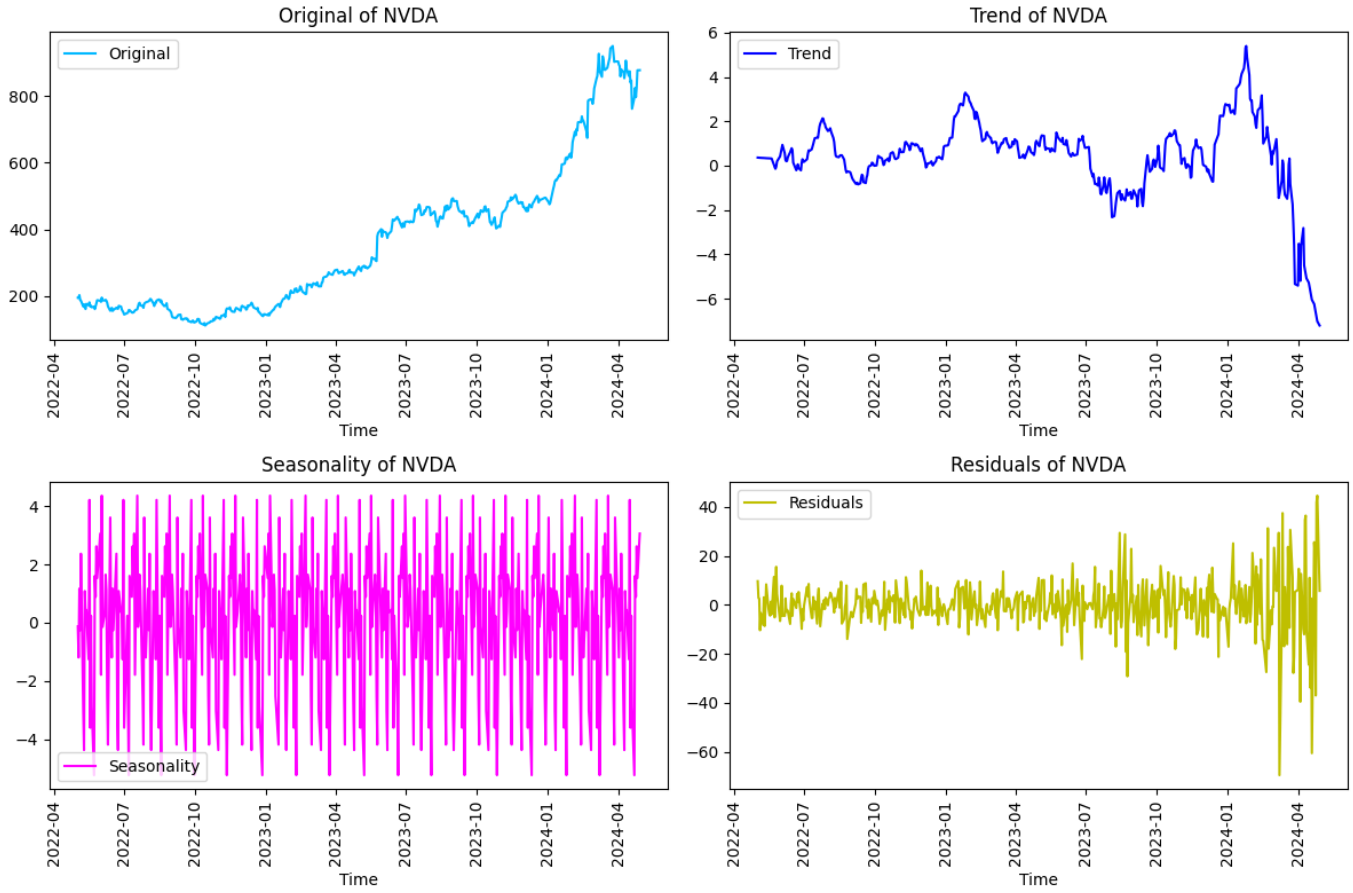

Fig.3.7 Time series decomposition results

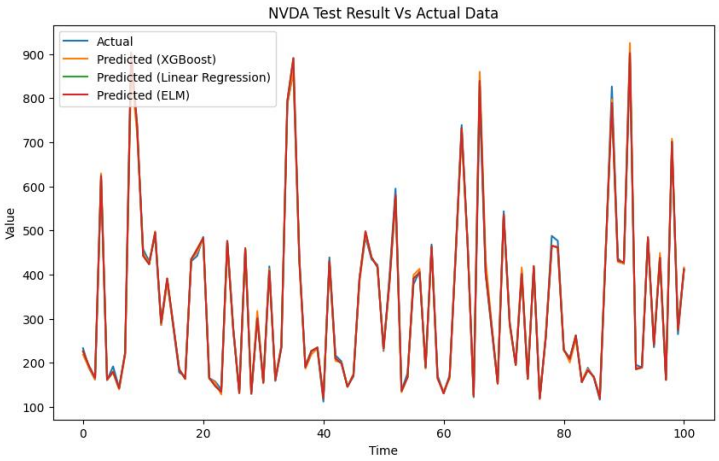

Fig.3.8 Actual Vs Predicted results

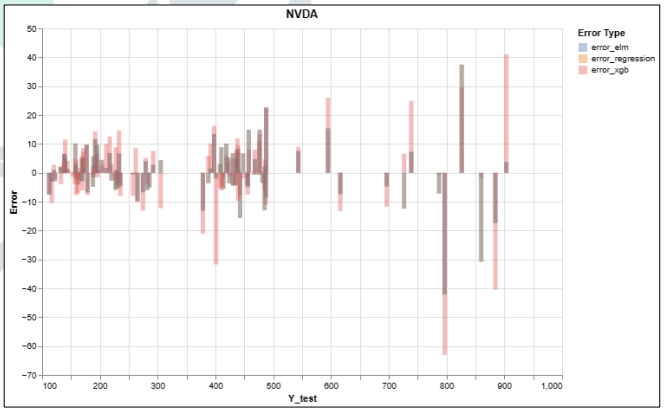

Fig.3.9 Error Vs Y_test

NVD:

GridSearchCV had an accuracy of approximately 99.60%, with the highest MSE and RMSE values among the models. Both Linear Regression and ELM achieved slightly lower accuracy but substantially lower MSE and RMSE values.

Tesla (TSLA) Stock prediction

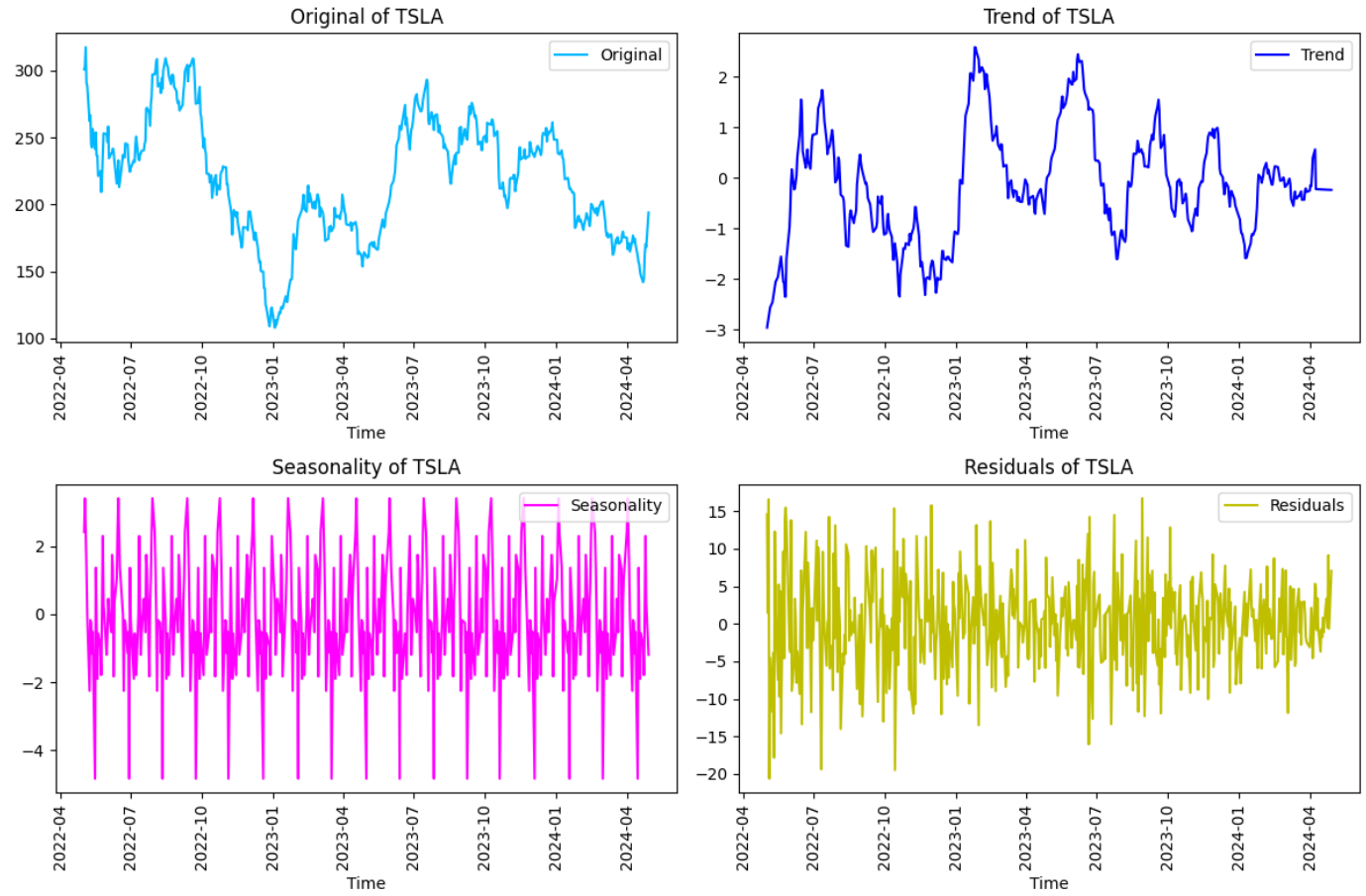

Fig.3.10 Time series decomposition results

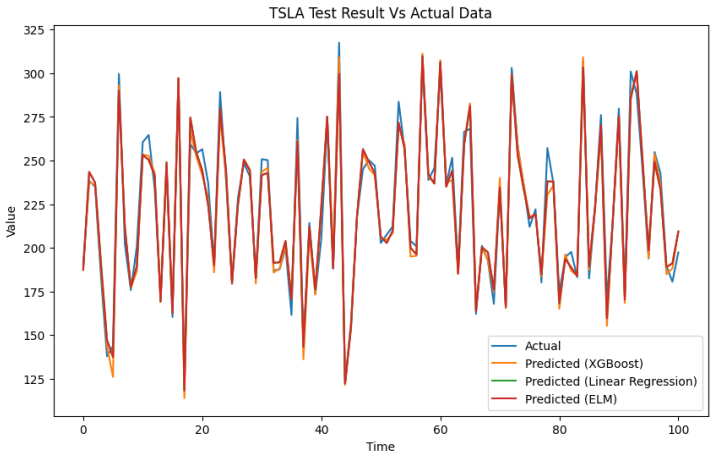

Fig.3.11 Actual Vs Predicted results

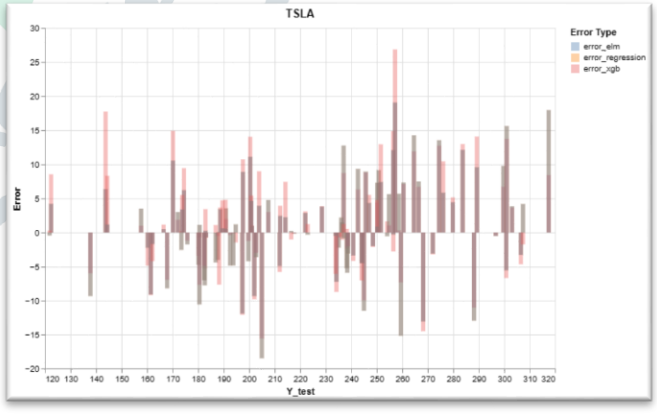

Fig.3.12 Error Vs Y_test

TSLA:

GridSearchCV achieved an accuracy of around 97.04%, with moderate MSE and RMSE values. Linear Regression and ELM achieved slightly higher accuracies of approximately 97.41% with similar MSE and RMSE values.

Walmart (WMT) Stock prediction

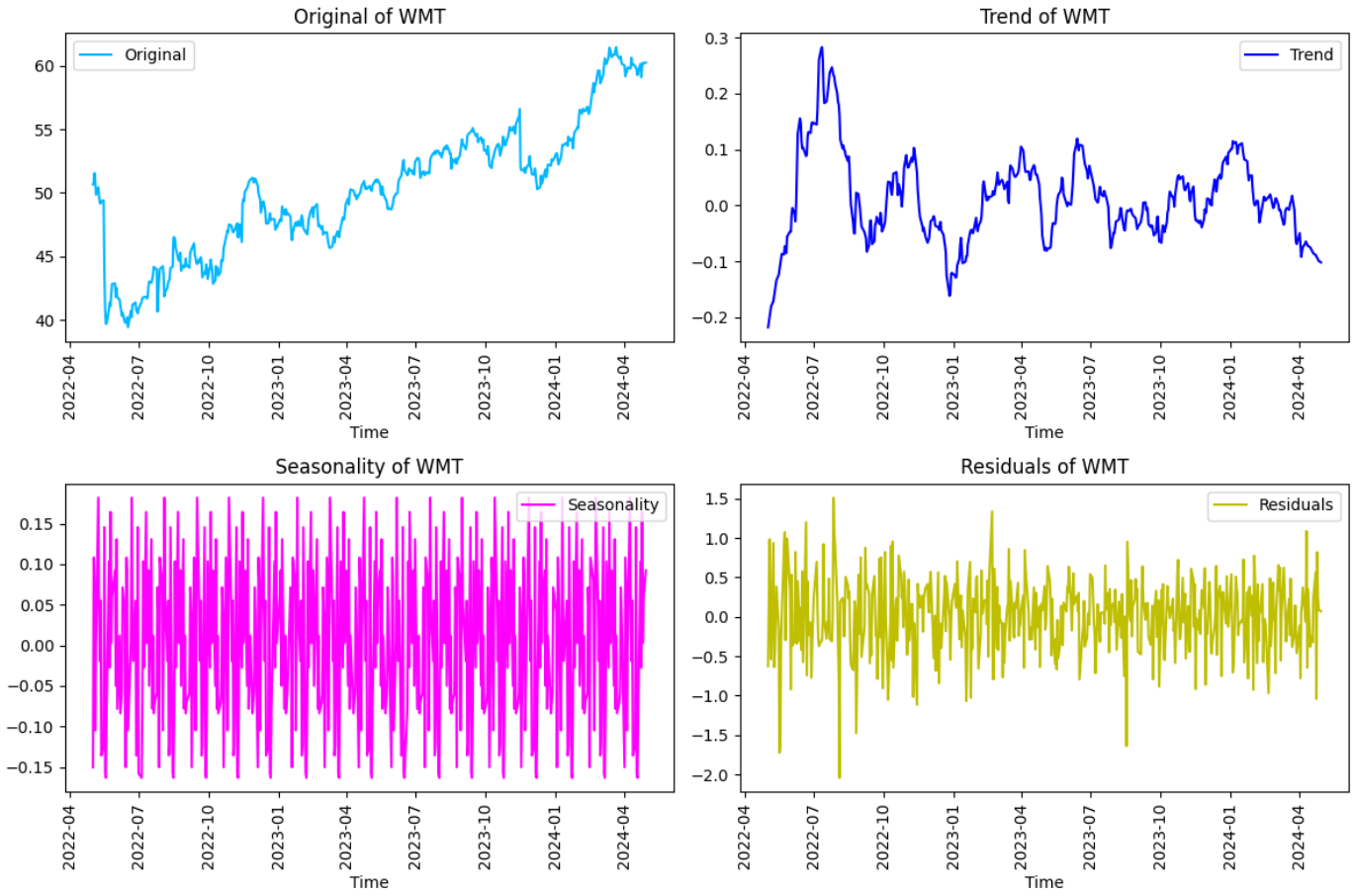

Fig.3.13 Time series decomposition results

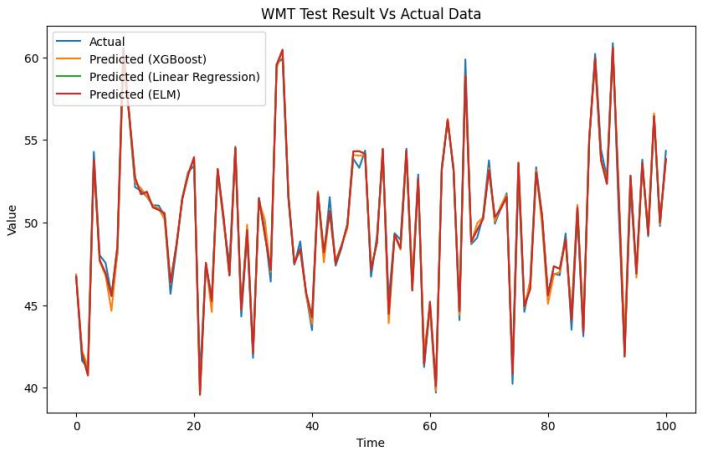

Fig.3.14 Actual Vs Predicted results

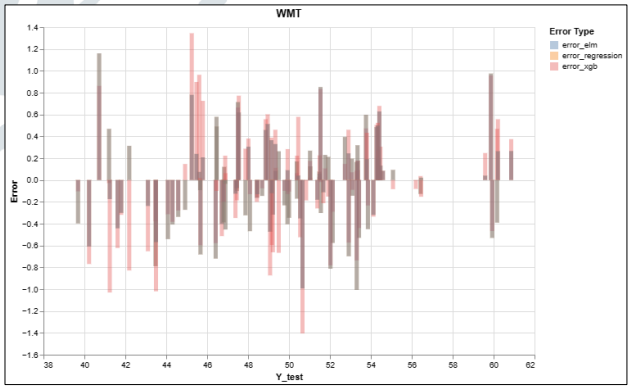

Fig.3.15 Error Vs Y_test

WMT:

GridSearchCV achieved an accuracy of approximately 98.74% with relatively low MSE and RMSE values. Both Linear Regression and ELM achieved slightly higher accuracies of around 99.17% with very low MSE and RMSE values

4. SUMMARIZE ABOVE MACHINE LEARNING MODELS

For the given stock names (FORD, AMZN, NVD, TSLA, WMT), three different models were applied: GridSearchCV, Linear Regression, and ELM (Extreme Learning Machine).

Both Linear Regression and ELM achieved slightly higher accuracies of around 99.17% with very low MSE and RMSE values.

Overall, Linear Regression and ELM consistently performed similarly across all stocks, while GridSearchCV sometimes achieved higher accuracy but at the cost of higher MSE and RMSE. The R-squared values were consistently high across all models and stocks, indicating good fit of the models to the data. Hyperparameter tuning with GridSearchCV is important because it allows the model to be fine-tuned and optimized for the specific dataset, ultimately improving its predictive performance.

In environments with high noise levels, traditional modeling techniques may struggle to capture the underlying patterns in the data effectively. GridSearchCV, by exhaustively searching through a parameter grid, can help identify the most suitable model and its corresponding hyperparameters that best fit the data, even amidst the noise.

Furthermore, in time series analysis, understanding and modeling the residuals are crucial. High noise levels can lead to more complex residual patterns, making it challenging to identify the correct model structure and hyperparameters. GridSearchCV can help in this scenario by exploring various model configurations and hyperparameters to minimize these residuals.

Therefore, in noisy environments, GridSearchCV's ability to systematically explore the model space and its hyperparameters can be particularly beneficial. By leveraging information from time series residuals, it can guide the selection of the most appropriate model, ultimately improving predictive accuracy in the face of high noise levels

5. FUTURE SCOPE

[1] The study focuses on advancing stock price prediction precision through the continual enhancement of machine learning algorithms.
[2] Leveraging natural language processing (NLP) techniques, the research aims to extract insights from news articles and textual sources to understand factors influencing stock prices.
[3] The investigation extends to forecasting the impact of non-financial events, such as political scandals or natural disasters, on stock prices using machine learning models.
[4] Additionally, the study explores the application of machine learning in forecasting the effects of climate change on the stock market.
[5] The overarching goal is to provide investors with valuable insights to navigate environmental uncertainties and safeguard their portfolios.

6. CONCLUSION:

The evaluation of machine learning models—Linear Regression, Grid Search CV, and Extreme Learning Machine (ELM)—in stock price prediction reveals intriguing insights into their performance across various stocks. While Grid Search CV demonstrates moderate performance, Linear Regression and ELM consistently exhibit competitive accuracy rates, underscoring their efficacy in financial forecasting. Notably, the potential for Grid Search CV to outperform lies in hyperparameter tuning, a facet not explored within this study. The integration of time series analysis with machine learning techniques offers promising avenues for enhancing prediction accuracy and understanding temporal dynamics in stock market behavior. By leveraging historical data and temporal insights, investors can make well-informed decisions, thus optimizing their investment portfolios and mitigating risks in the dynamic landscape of financial markets.

V. ACKNOWLEDGMENT

We would like to extend our deepest appreciation to Dr. Bharatheesh Jaysimha for his invaluable guidance and mentorship throughout the entirety of our research endeavors. Dr. Bharatheesh Jaysimha's expertise and steadfast support have played a crucial role in shaping our methodology and refining our analysis.

Furthermore, we wish to express our sincere gratitude to Paul Vettukallel and Kumarswamy Amyshetty for their unwavering assistance and support at various stages of this project. Their contributions have been indispensable in facilitating the successful execution of our research objectives.

We are truly thankful for the collaborative effort that has enriched this study and enabled us to achieve our goals.

REFERENCES

[1] Ali, A. 2001.Macroeconomic variables as common pervasive risk factors and the empirical content of the Arbitrage Pricing Theory. Journal of Empirical finance, 5(3): 221–240.
[2] Basu, S. 1997. The Investment Performance of Common Stocks in Relation to their Price to Earnings Ratio: A Test of the Efficient Markets Hypothesis. Journal of Finance, 33(3): 663-682.
[3] Bhatti, U. and Hanif. M. 2010. Validity of Capital Assets Pricing Model.Evidence from KSE-Pakistan.European Journal of Economics, Finance and Administrative Science, 3 (20).
[1]S.Shukla and B.Shah's "Stock Price Prediction Using Multiple Linear Regression Models"International Journal of Engineering Science Invention (IJESI). Published in October 2018.
[2] N.Rouf and S.Singh's "Stock Price Prediction Using Machine Learning Techniques: A Decade Survey on Methodologies, Recent Developments, and Future Directions" MDPI. Published in November 2021.
[3] V. Gururaj, S. V R and A.karthick's "Stock Price Prediction Using Linear Regression and Support Vector Machines".International Journal of Applied Engineering Research. Published in 2019.
[4] N. Karlsson's "Comparison of Linear Regression and Neural Network for Stock Price Prediction".
[5] P.Soni, Y.Tewari, D. Krishnan-"Machine Learning approaches in Stock Price Prediction: A Systematic Review", Journal of Physics Conference Series. Published in 2022.

[6] M.Vijh, D.Chandola's "Stock Closing Price Prediction using Machine Learning Techniques". International Conference on Computational Intelligence and Data Science (ICCIDS).

[7] K.Daniel's "A Study on Stock Price Prediction Using Linear Regression and Random Forest Regression" (2022).

[8] O. E. Orsel, S. S. Yamada-"Comparative Study of Machine Learning Models for Stock Price Prediction" (2022).

[9] J.Sen-"Stock Price Prediction Using Machine Learning and Deep Learning Framework", 6th International Conference on Business Analytics and Intelligence (ICBAI). Published in December 2018.

[10] I.Kumar, C.Utreja's "A Comparative Study of Supervised Machine Learning Algorithms for Stock Market Trend Prediction". IEEE Conference. (2018).

[11] M.Nikou, G.Mansourfar, J.Bagherzadeh's "Stock price prediction using DEEP learning algorithm and its comparison with machine learning algorithms". Intelligent Systems in Accounting, Finance and Management. Published in 2019.

[12] V.K. Sai Reddy's "Stock market prediction using machine learning". International Research Journal of Engineering and Technology (IRJET). Published in 2018.

[**13**]M.Obthong, N. Tantisantiwong, W.Jeamwatthanachai, G.Wills's "A survey on machine learning for stock price prediction: Algorithms and techniques".

[14]M. Umer, M. Awais, M. Muzammul- "Stock market prediction using machine learning (ML) algorithms" (ADCAIJ) Advances in Distributed Computing and Artificial Intelligence Journal. Published in year 2019.

[15] D. Kumar, P. K. Sarangi, R.Verma- "A systematic review of stock market prediction using machine learning and statistical techniques" Published in 2022.

[16] Javatpoint's-https://static.javatpoint.com/tutorial/machine-learning/images/linear- regression-in-machine-learning.png.