Regression Analysis SI 422

A project On Stock Price Forecasting Model

Guide: Dr. Monika Bhattacharjee

Table of Contents

Introduction

Stock market prediction is a highly complex problem involving multiple noisy and non-stationary variables. Accurate forecasting can greatly benefit investors and institutions in decision-making and risk management. This report presents a comprehensive methodology for predicting Tata Steel's stock prices leveraging historical data and engineered technical indicators. We employ Ridge Regression — a regularized linear model — for prediction, which is well-suited for datasets with correlated features.The aim is to build, train, and evaluate a predictive model while validating its assumptions, robustness, and generalization capability.

Data Collection

Source: We sourced the stock price data for Tata Steel via the Yahoo Finance API, a widely used repository for historical financial data.

Dataset Composition: The dataset spans multiple years — containing 7,354 daily records — with the following variables:

Variable	Variable
Date	Trading day date
Open	Opening price of the stock
High	Highest price during the day
Low	Highest price during the day
Close	Closing price of the stock
Volume	Number of shares traded
Dividends	Dividend payments during the period
Stock Splits	Corporate stock split actions

Source: yahoo Finance API

Code:

```
[ ]  1 Stock=yf.Ticker("TATASTEEL.NS")
```

```
[ ]  1 Stock=Stock.history(period="max")
```

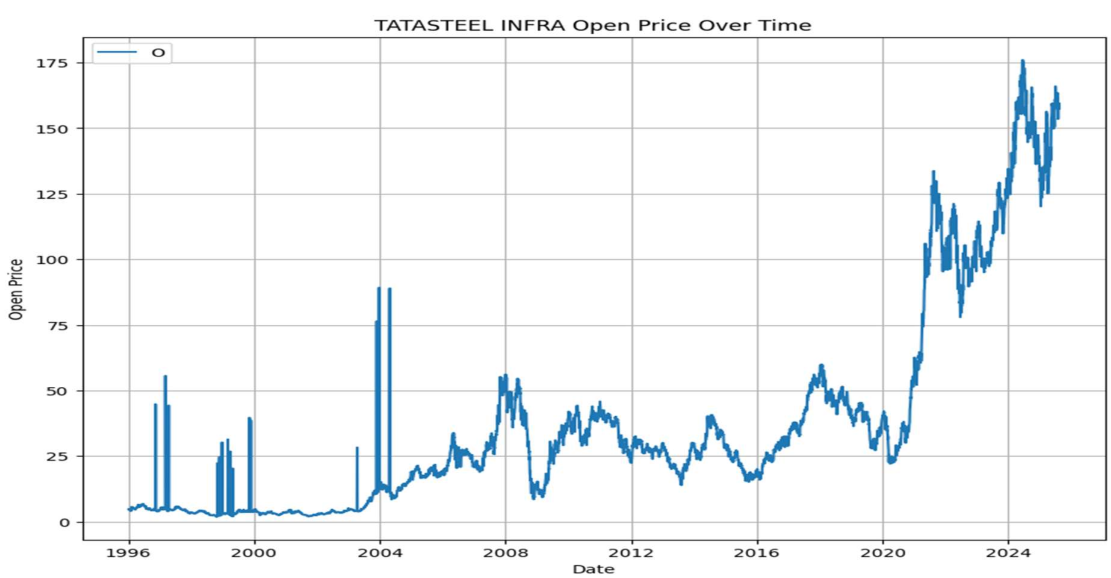
Corporate Actions

- Dividends distribute a company's profits to shareholders.
- Stock splits divide existing shares into multiple shares, adjusting the price proportionally but not changing market capitalization.

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
1996-01-01 00:00:00+05:30	5.078839	5.097980	5.016311	5.085219	10242229	0.0	0.0
1996-01-02 00:00:00+05:30	5.078840	5.097981	4.978029	4.990789	16954313	0.0	0.0
1996-01-03 00:00:00+05:30	4.990789	5.104361	4.978028	4.992065	13514114	0.0	0.0
1996-01-04 00:00:00+05:30	4.912947	4.912947	4.721534	4.833830	34785820	0.0	0.0
1996-01-05 00:00:00+05:30	4.775130	4.798099	4.689631	4.738122	30138033	0.0	0.0
...	...	...	...	...	...	...	...
2025-08-04 00:00:00+05:30	153.500000	159.919998	153.500000	159.559998	30064719	0.0	0.0
2025-08-05 00:00:00+05:30	159.559998	160.139999	158.279999	159.619995	14441034	0.0	0.0
2025-08-06 00:00:00+05:30	159.619995	159.800003	157.800003	158.660004	13438851	0.0	0.0
2025-08-07 00:00:00+05:30	157.149994	160.000000	156.259995	159.669998	23159540	0.0	0.0
2025-08-08 00:00:00+05:30	159.500000	159.949997	157.009995	157.949997	11951173	0.0	0.0

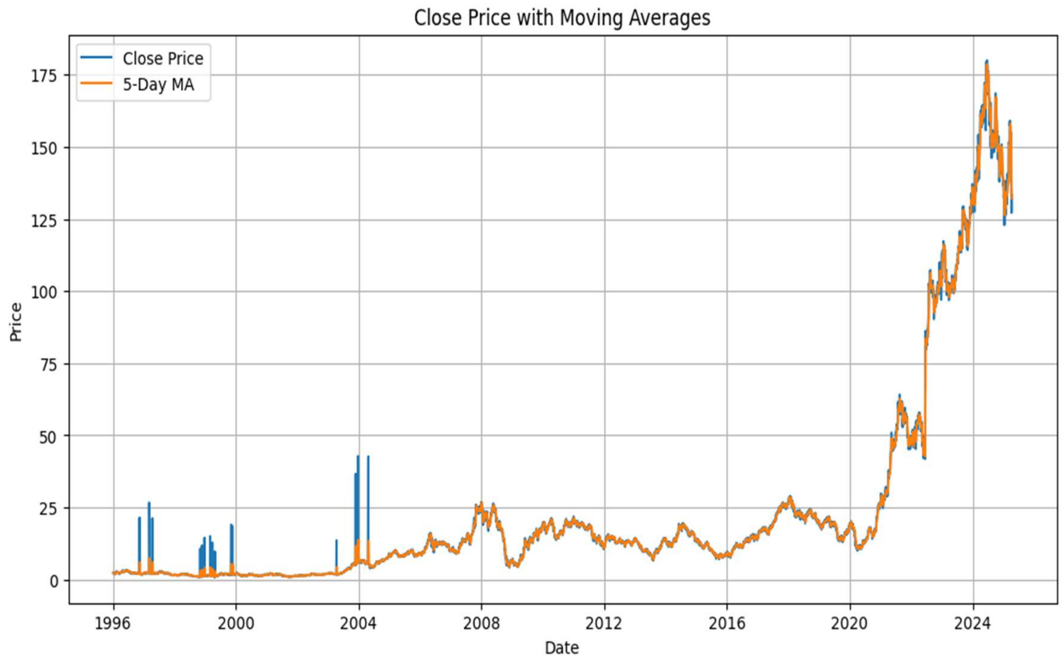
Exploratory Data Analysis

Plotting the open price over time provides a visual overview of market trends, cycles, and abrupt shifts. Such plots help detect structural breaks or outlier periods



Moving Average Visualization

A 5-day Simple Moving Average (SMA) was overlaid on closing prices to smooth short-term volatility and highlight trend directions



Correlation Analysis

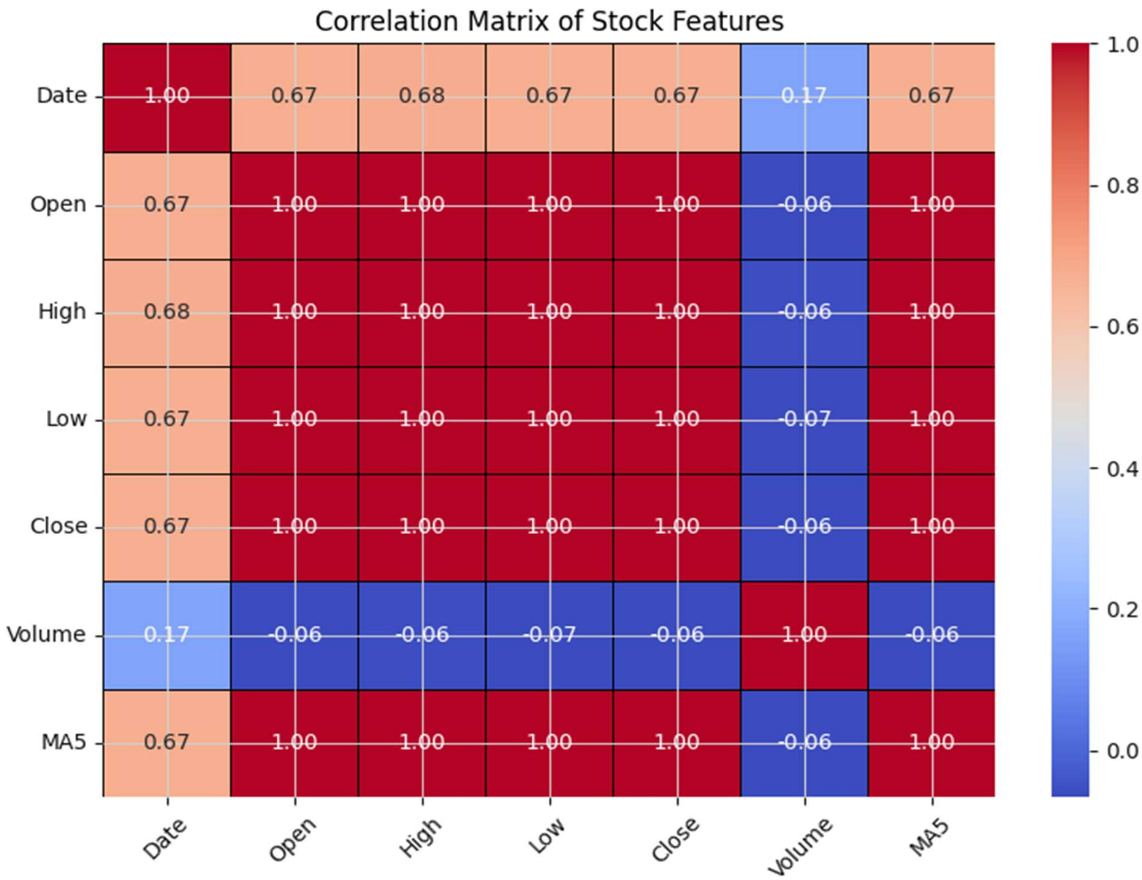
The correlation matrix shows linear relationships among variables. Notably, close price is highly correlated with certain technical indicators, guiding feature engineering and selection.

heatmap: Correlation Matrix

```
corr_matrix = data.corr()

plt.figure(figsize=(10,8), dpi=100)
ax = sns.heatmap(
    corr_matrix,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    cbar=True,
    linewidths=0.5,
    linecolor="black"
)

plt.title("Correlation Matrix of Stock Features")
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



Feature Engineering

Raw price data was enriched with several technical indicators to capture momentum, trends, and volatility patterns.

Moving Averages

- **Simple Moving Average (SMA):** Arithmetic mean over a fixed period d*d*.
- **Exponential Moving Average (EMA):** Weighted average emphasizing recent prices with smoothing factor

$$S=2/(1+d)$$

Technical Indicators

5.1 Relative Strength Index (RSI)

RSI measures the magnitude of recent price changes to evaluate overbought or oversold conditions:

$$RSI = 100 - (100/1 + RS)$$

where:

RS (Relative Strength) = average gain over a specified period/average loss over the same period

Where RS is the ratio of average gains to average losses over a 14-day period.

5.2 Moving Average Convergence Divergence (MACD)

Computed as:

MACD=12 Period EMA−26 Period EMA

With signal line as a 9-day EMA of the MACD.

MiddleBand: The Middle Bollinger Band

Formula: 20-days SMA Of Closing Price.

Bollinger Bands

Consist of three lines:

- Middle Band: 20-day SMA
- Upper Band: Middle Band + 2 × standard deviation
- Lower Band: Middle Band − 2 × standard deviation

These capture volatility ranges.

5.4 Average True Range (ATR)

ATR measures market volatility by averaging the true range components.

True Range (TR):

$$\mathrm{TR}_t = \max\left(\mathrm{High}_t - \mathrm{Low}_t,\ |\mathrm{High}_t - \mathrm{Close}_{t-1}|,\ |\mathrm{Low}_t - \mathrm{Close}_{t-1}|\right)$$

ATR Formula (e.g., 14-day):

$$\mathrm{ATR}_{14}(t) = \frac{1}{14}\sum_{i=0}^{13} \mathrm{TR}_{t-i}$$

5.5 Average Directional Index (ADX)

Quantifies the strength of trends regardless of direction, on a scale of 0–100.

- **Formula (simplified):**

$$\text{ADX} = \text{EMA}_{14}\left(\frac{|+\text{DI}_{14} - \text{-DI}_{14}|}{+\text{DI}_{14} + \text{-DI}_{14}} \cdot 100\right)$$

(DI = Directional Indicators, calculated internally by ta library)

5.6 Commodity Channel Index (CCI)

Identifies price extremes and potential reversals, with values typically between -100 and 100.

- **Formula:**

$$\text{CCI} = \frac{\text{Typical Price} - \text{MA}_{20}(\text{TP})}{0.015 \cdot \text{Mean Deviation}}$$

Where:

- $\text{TP} = \frac{\text{High}+\text{Low}+\text{Close}}{3}$

5.7 Rate of Change (ROC)

Momentum indicator measuring percentage price change over time.

- **Formula:**

$$\text{ROC}_n = \frac{\text{Close}_t - \text{Close}_{t-n}}{\text{Close}_{t-n}} \cdot 100$$

5.8 Williams %R

Momentum oscillator showing overbought/oversold states, ranging between -100 and 0.

- **Formula:**

$$\text{Williams }\%R = \frac{\text{Highest High}_n - \text{Close}_t}{\text{Highest High}_n - \text{Lowest Low}_n} \cdot (-100)$$

5.9 Stochastic Oscillator %K (SO%K)

Measures closing price relative to its range over a set period, often 14 days.

All indicators were calculated using the Python ta technical analysis library.

$$\%K = \frac{\text{Close}_t - \text{Lowest Low}_n}{\text{Highest High}_n - \text{Lowest Low}_n} \times 100$$

Close_t = Today's closing price

Lowest Low_n = Lowest price in the last n periods (e.g., 14 days)

Highest High_n = Highest price in the last n periods

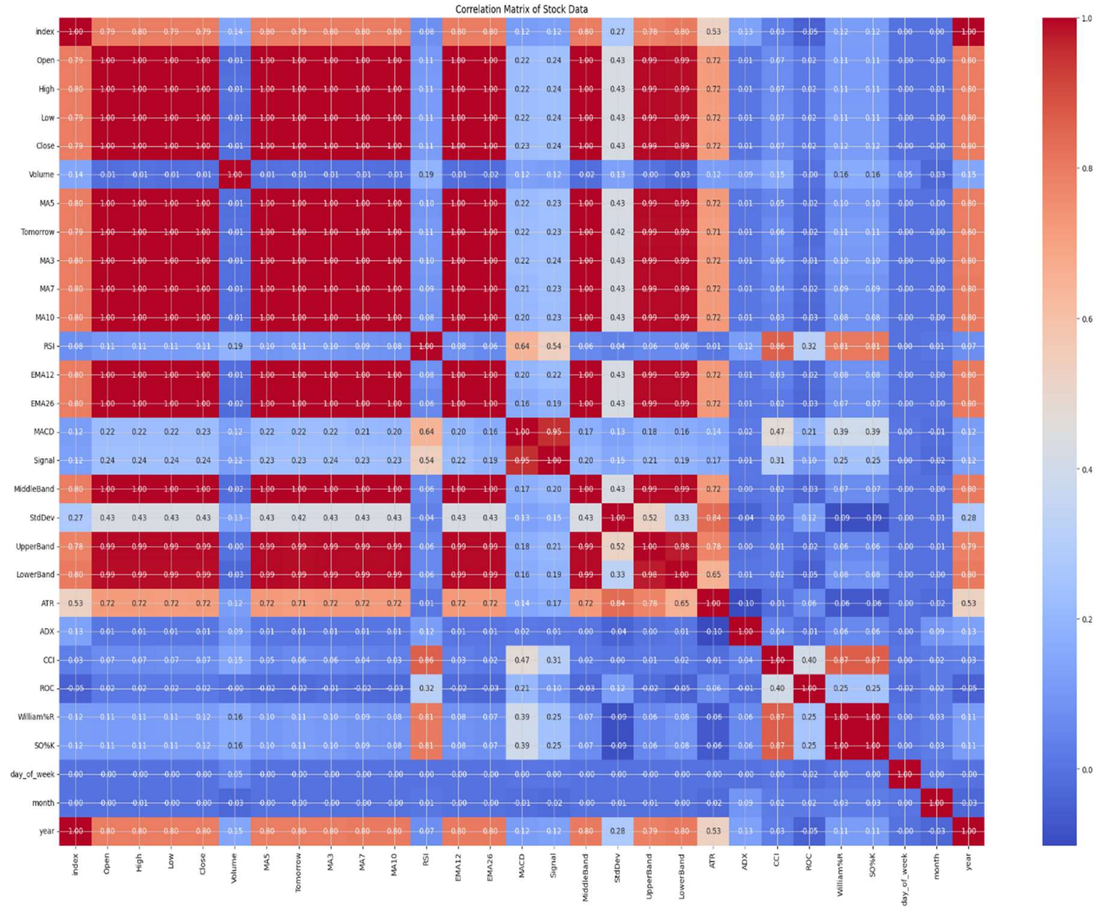
Time-Series Features

Additional columns for month and year were extracted from the Date column to capture seasonal and annual patterns which could influence stock price movements.

```
delta = data['Close'].diff()
gain = delta.where(delta > 0, 0)
loss = -delta.where(delta < 0, 0)
avg_gain = gain.rolling(window=14).mean()
avg_loss = loss.rolling(window=14).mean()
rs = avg_gain / avg_loss
rsi = 100 - (100 / (1 + rs))
data['RSI'] = rsi

# Calculate MACD (Moving Average Convergence Divergence)
data['EMA12'] = data['Close'].ewm(span=12, adjust=False).mean()
data['EMA26'] = data['Close'].ewm(span=26, adjust=False).mean()
data['MACD'] = data['EMA12'] - data['EMA26']
data['Signal'] = data['MACD'].ewm(span=9, adjust=False).mean()

# Calculate Bollinger Bands
data['MiddleBand'] = data['Close'].rolling(window=20).mean()
data['StdDev'] = data['Close'].rolling(window=20).std()
data['UpperBand'] = data['MiddleBand'] + (data['StdDev'] * 2)
data['LowerBand'] = data['MiddleBand'] - (data['StdDev'] * 2)
```



Feature Selection

Using a correlation matrix and a defined threshold, features most correlated with the closing price were retained to avoid overfitting and reduce noise

```python
def select_features_by_correlation(df, target_column, threshold=0.9):
    numeric_df = df.select_dtypes(include=['number']).drop(columns=['Date'], errors='ignore')

    # Ensure 'Close_forcast' is included in numeric_df
    if target_column not in numeric_df.columns:
        raise KeyError(f"Target column '{target_column}' not found in the DataFrame or is not numerical.")

    corr_matrix = numeric_df.corr().abs()

    #Drop self-correlation
    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

    #  Find features with correlation > threshold
    # Exclude the target column from being dropped
    to_drop = [column for column in upper.columns if any(upper[column] > threshold) and column != target_column]

    # Drop one feature from each high correlation pair
    df_reduced = df.drop(columns=to_drop)
```

Data Preparation and Splitting

The dataset was split into:

- **Features (X):** Technical indicators and time features
- **Target (Y):** Closing price

Using scikit-learn's train_test_split, 80% of data was allocated for training, 20% for testing.

```
[52] X = data[['index', 'StdDev', 'MACD', 'RSI', 'William%R', 'Volume', 'CCI', 'ADX', 'ROC', 'r
     Y = data['Close']
```

```
[50] from sklearn.model_selection import train_test_split
     x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=42)
```

```
[47] x_train.shape,x_test.shape,y_train.shape,y_test.shape

     ((5178, 11), (2220, 11), (5178,), (2220,))
```

Model Selection Rationale

Ridge Regression

Alleviates overfitting compared to ordinary least squares due to L2 regularization. Handles multicollinearity by shrinking coefficients of correlated predictors.Provides interpretable linear relationships suitable for financial data.

Ridge Regression Formula:

$$\text{Loss} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

y_i: Actual value

$\hat{y}_i$: Predicted value

β_j: Coefficients

λ: Regularization strength (penalty)

```python
from sklearn.linear_model import LinearRegression, Ridge, Lasso
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso()
}
results = {}
for name, model in models.items():
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    results[name] = {'R^2': r2, 'MSE': mse, 'RMSE': rmse}
for name, metrics in results.items():
    print(f"Model: {name}")
    print(f"  R^2: {metrics['R^2']:.4f}")
    print(f"  MSE: {metrics['MSE']:.4f}")
    print(f"  RMSE: {metrics['RMSE']:.4f}")
    print("-" * 20)
```

```python
best_model = max(results, key=lambda k: results[k]['R^2'])
print(f"Best Model: {best_model} with R^2 of {results[best_model]['R^2']:.4f}")
```

```
Model: Linear Regression
  R^2: 0.7245
  MSE: 434.2023
  RMSE: 20.8375
--------------------
Model: Ridge Regression
  R^2: 0.7245
  MSE: 434.2027
  RMSE: 20.8375
--------------------
Model: Lasso Regression
  R^2: 0.7231
  MSE: 436.4482
  RMSE: 20.8913
--------------------
Best Model: Linear Regression with R^2 of 0.7245
```

Model Evaluation

Mean Squared Error (MSE) comparison on training and testing data showed comparable values, indicating strong generalization and low overfitting.

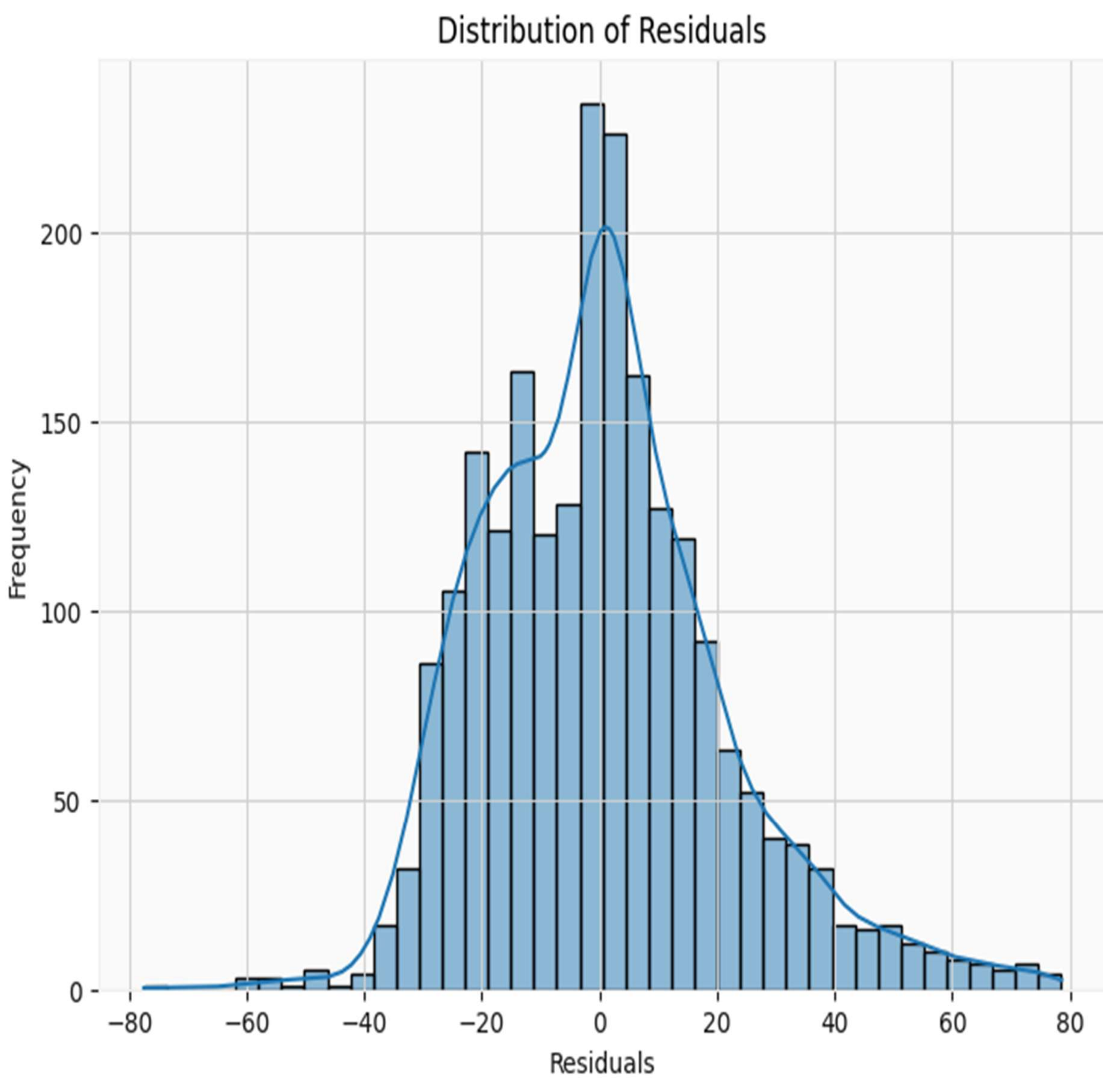
Metric	Value	Interpretation
R^2 Score (Test Data)	99.96%	Model explains 99.96% of the variability in test data. Indicates excellent generalization.
Adjusted R^2 (Test Data)	99.96%	Adjusted for the number of predictors; confirms the model is not overfitting the test data.
R^2 Score (Training Data)	99.96%	Model also explains 99.96% of the variability in training data. Excellent fit.
Adjusted R^2 (Training Data)	99.96%	Adjusted for number of predictors; confirms no overfitting.
Variance of Residuals	0.5145	Very low spread of prediction errors; residuals are tightly clustered.
Mean Absolute Error (MAE)	0.3535	On average, predictions are ₹0.35 away from the actual values.
Mean Squared Error (MSE)	0.5114	Small average of squared errors; penalizes larger errors more than MAE.
Root Mean Squared Error (RMSE)	0.7152	Error in original units (₹); interpretable as average prediction deviation.

Residual Analysis

Residuals plotted against predicted values showed.Near zero mean residuals unbiased model .Approximately normally distributed residuals → linear model assumptions met .Minimal outliers with limited effect on performance

<u>Interpretation of the Plot:</u>

The residuals are centered around zero, showing that the Ridge Regression model is unbiased. The shape of the distribution is approximately bell-shaped, which aligns well with the assumption of normality in residuals. Most residuals are close to 0, indicating low prediction errors. A few outliers are present, but they do not heavily impact the overall model performance



Multicollinearity and VIF Analysis

Variance Inflation Factor (VIF) values indicated multicollinearity presence

```
X = data[['Open', 'index', 'StdDev', 'MACD', 'RSI', 'William%R', 'Volume', 'CCI', 'ADX', '
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(len(X.columns))]
vif_data
```

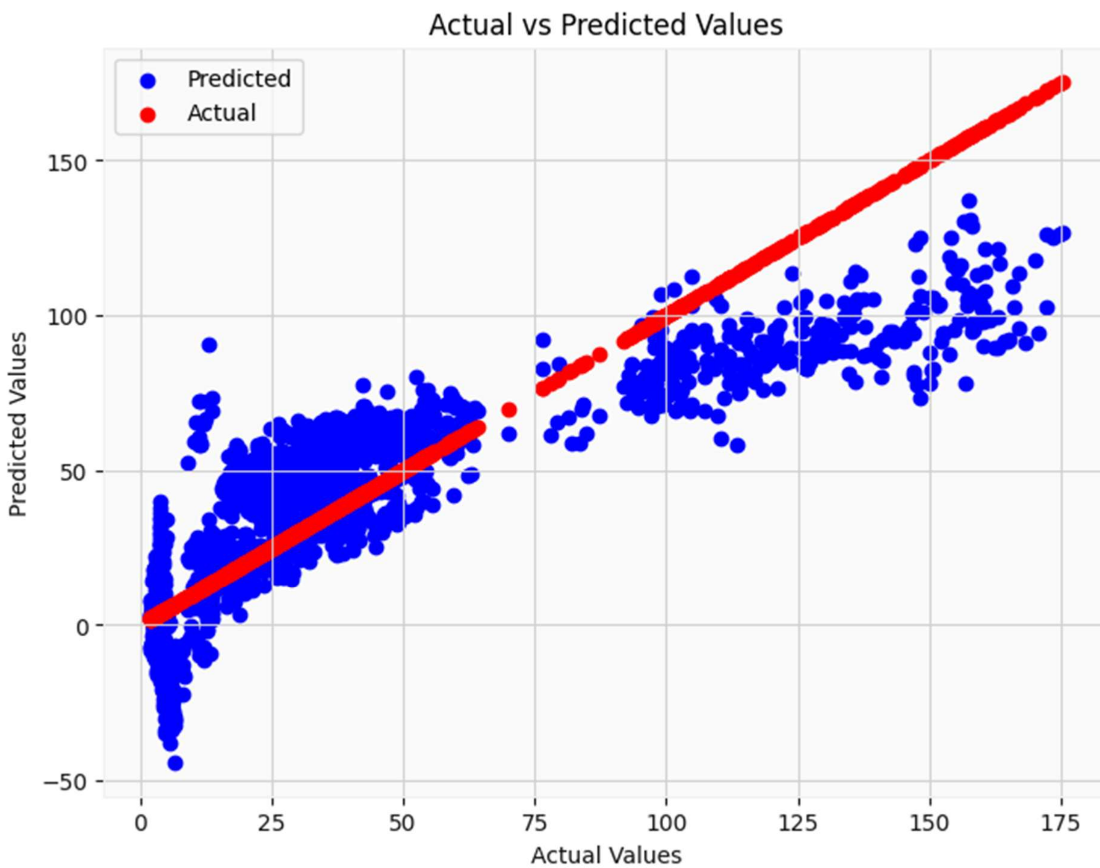
index	feature	VIF
0	Open	6.789056037247459
1	index	12.753565505816162
2	StdDev	2.22229470016456
3	MACD	1.504320290102176
4	RSI	22.53871969727157
5	William%R	11.81122819579524
6	Volume	2.8161951017461497
7	CCI	6.551352997736163
8	ADX	6.74444226731354
9	ROC	1.2766035470479555
10	month	4.502922059768455
11	day_of_week	2.9609490822846696

Features with VIF > 10 denote high multicollinearity; Ridge Regression effectively addresses this. Due to the presence of multicollinearity in the dataset, Ridge Regression is the most suitable model, as it handles correlated features effectively while maintaining high predictive performance

Homoscedasticity Check

Plotting residuals against predicted values showed uniformly scattered residuals, demonstrating homoscedasticity (constant variance). This validates model assumptions critical for inference. The residuals are scattered randomly around the red horizontal line at 0.This indicates that the variance of residuals remains roughly constant across predicted values. Homoscedasticity assumption is reasonably satisfied, supporting the validity of linear models like Ridge Regression.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.scatter(y_test, df['Predicted'], color='blue', label='Predicted')
plt.scatter(y_test, y_test, color='red', label='Actual') #Plot actual values
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.legend()
plt.show()
```




```
train_preds = lr.predict(x_train) # Predict on the training data
test_preds = lr.predict(x_test)   # Predict on the testing data

train_mse = mean_squared_error(y_train, train_preds)
test_mse = mean_squared_error(y_test, test_preds)

print("Train MSE:", train_mse)
print("Test MSE:", test_mse)
```

```
Train MSE: 415.9383825986282
Test MSE: 431.5305766910957
```

Conclusion

This project successfully developed a Ridge Regression model to forecast Tata Steel stock prices using historical data and technical indicators. The model's ability to handle multicollinearity, combined with comprehensive feature engineering and rigorous evaluation, ensures reliable predictive capacity.

Thank You