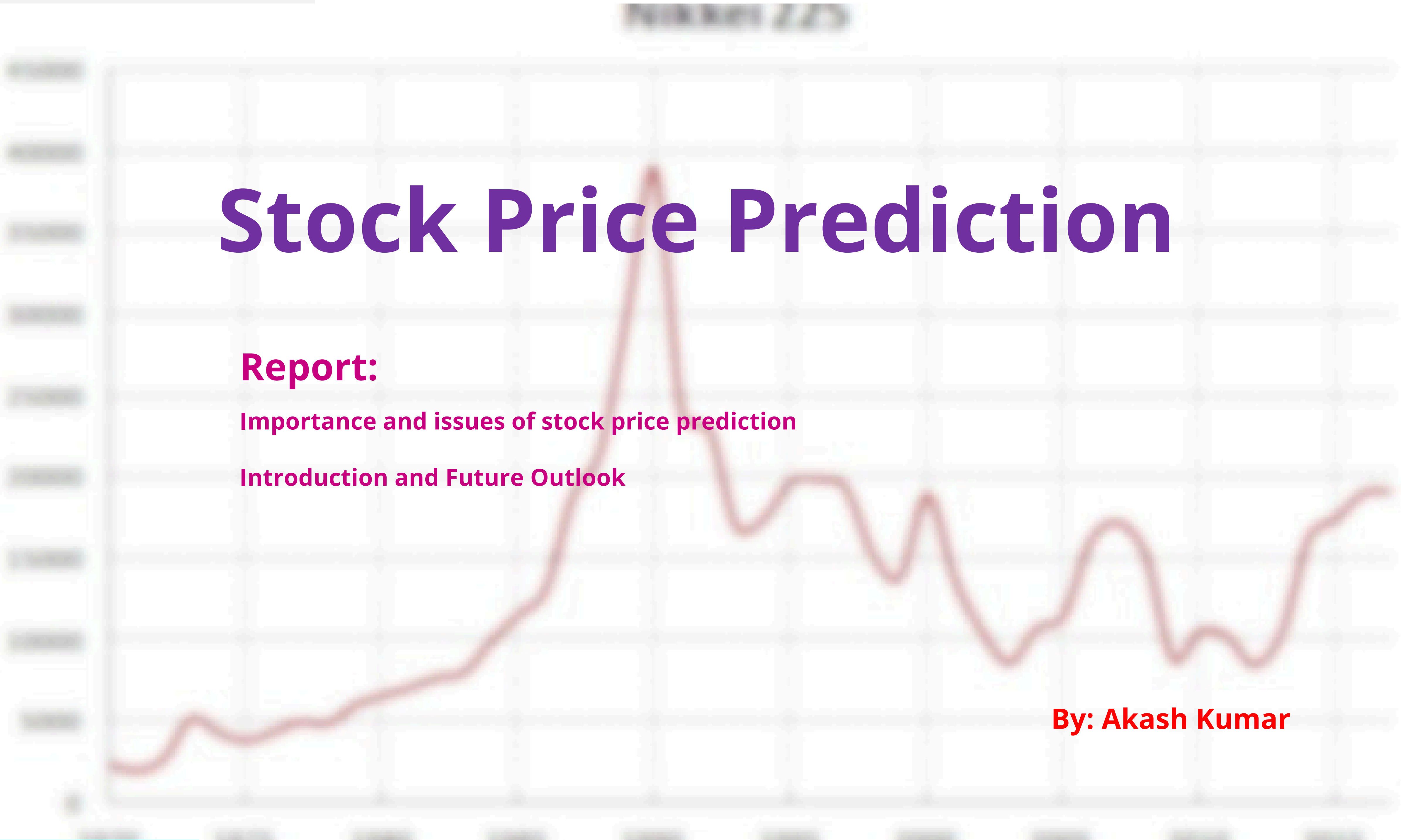


Stock Price Prediction

Report:

Importance and issues of stock price prediction

Introduction and Future Outlook

By: Akash Kumar

My Introduction

Name - AKASH KUMAR

COLLEGE - IIT(ISM) DHANBAD

SKILLS - Python , C++ , JupyterNotebook , Tableau , Numpy , Pandas , Matplotlib, Seaborn , Docker and Excel

Hobbies - I like to invest my time in Crypto-market analysis and NFT meta on web3 .

"Dear Deepcraft.ai Team,

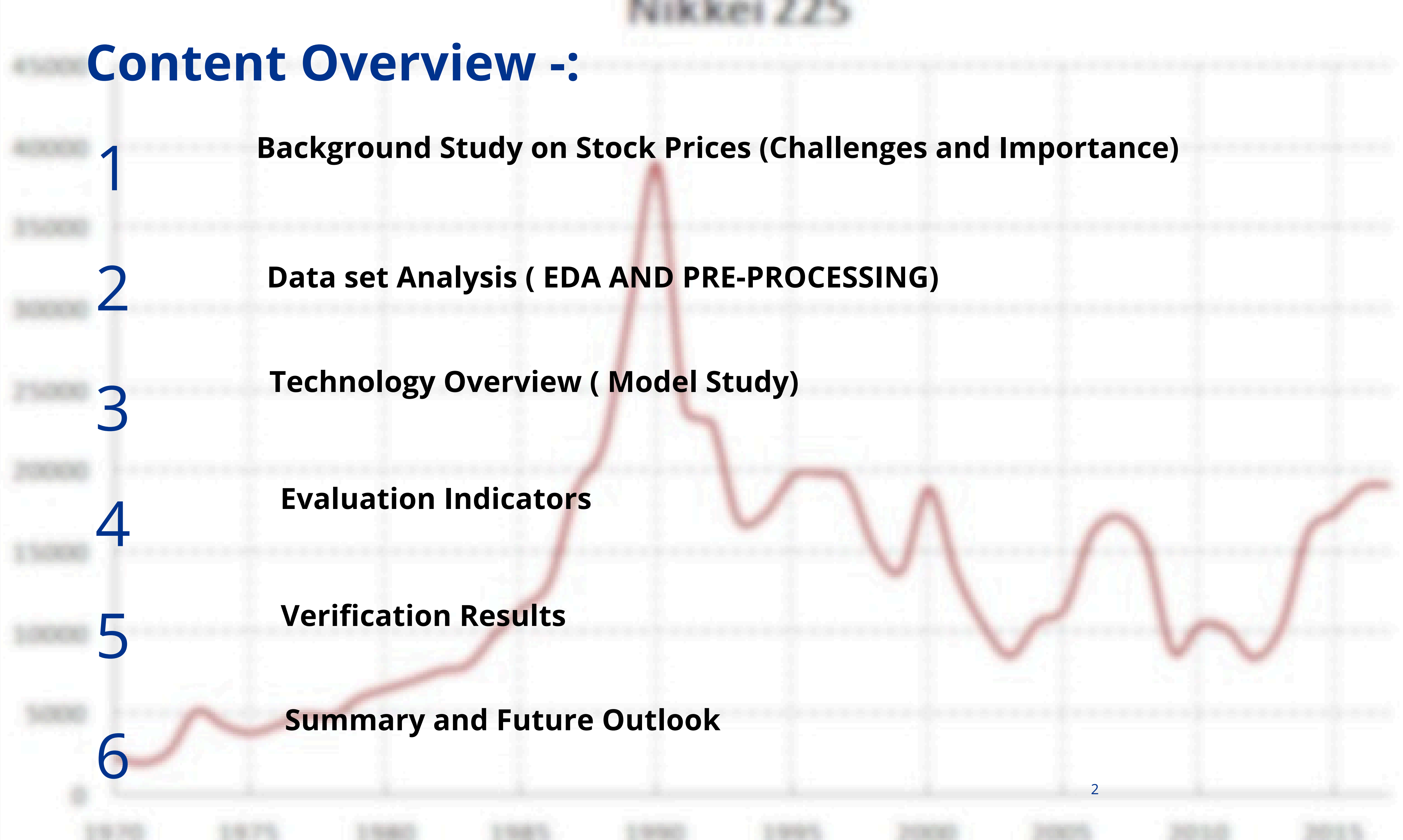
I hope this message finds you well. I am pleased to submit the completed project as per outlined objectives. This project represents my dedication and effort to align with Deepcraft.ai's vision and goals.

The final deliverables include:

- [Report for the Trainee Project]
- [Google Collab Notebook and Git-hub Link]

I appreciate the opportunity to contribute to your exciting initiatives, and I look forward to your feedback.

Thank you for the opportunity to work on this project."



Content Overview -:

"Why D&A REQUIRED ???"

Stock price prediction is critical for maximizing investment returns, managing risk, and guiding economic policy.

Prediction can be performed By studying data from the real - time market.

Investment Decisions

1% improvement in prediction accuracy could translate into billions in additional profits for institutional investors managing portfolios worth hundreds of billions.

Risk Management

Investors can hedge their positions, manage risks, and prevent losses.

7.9$ Trillion loss is mitigated yearly by risk assesement

High-Frequency Trading (HFT)

- Precise, short-term price predictions, often based on milliseconds or seconds, enable traders to profit from rapid price fluctuations.
- HFT firms generate $8-10 billion in annual revenue by
- exploiting small stock price movements predicted through algorithms.

Economic Indicators

Stock indices like the S&P 500 or Dow Jones are closely watched indicators of economic sentiment. Forbes estimates that stock market fluctuations can affect 5-7% of U.S. GDP through the wealth effect.

Challenges to be Faced:

Market Volatility and Uncertainty:

Predicting such sharp and sudden movements is extremely challenging due to the chaotic nature of the market.

Non-Stationarity of Stock Prices:

Stock prices are inherently non-stationary, meaning their statistical properties (mean, variance, etc.) change over time. This violates assumptions of traditional models like ARIMA, making it difficult to apply certain predictive techniques without data transformation.

Complexity and Noise:

Stock prices are influenced by multiple factors like earnings reports, macroeconomic data, interest rates, and market sentiment. Disentangling these variables from the "noise" (random fluctuations) is complex.

Data Quality and Availability

The quality of data impacts the accuracy of stock price predictions. Missing data, incorrect reporting, and high-frequency noise can distort model result. Often the data is insufficient for proper analysis

Technological Disruptions:

Seasonality Issues ,too many features leading to overfitting and Model capability to limited to certain level of disruptions in Data

D&A Inluences in Global Companies For

Global companies understand their Risk management, Market Volatility, and compliance.

	Apple Inc.	Microsoft Corp. (MSFT)	Amazon
Market Volatility:	**15**%	**17%**	**20**%
Regulatory Risks:	**10**%	**9%**	**20**%
Market Sentiment and Behavioral Factors:	**10**%	**6%**	**15**%

Sources https://emeritus.org/in/learn/stock-price-prediction-using-machine-learning/

Data Analysis on ??

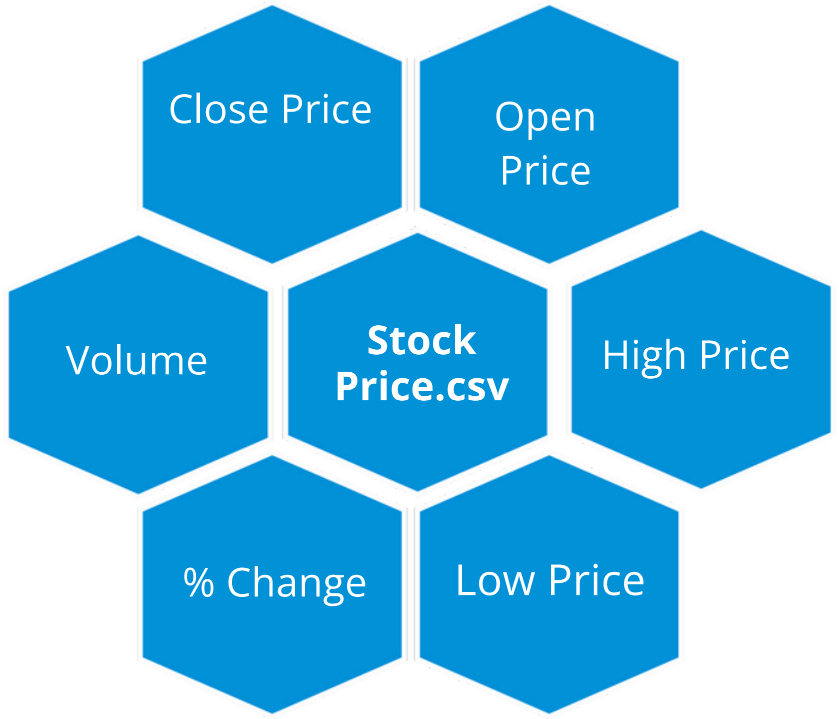
- **Pre- Processing**

  Handle missing data (e.g., imputation or removal).
  Remove duplicates to avoid redundant information.
  Detect and remove outliers (e.g., Z-score, IQR).
  Scale or normalize data to ensure consistent feature contribution.
  Date parsing and feature extraction (e.g., year, month, day, weekday)

- **EDA**

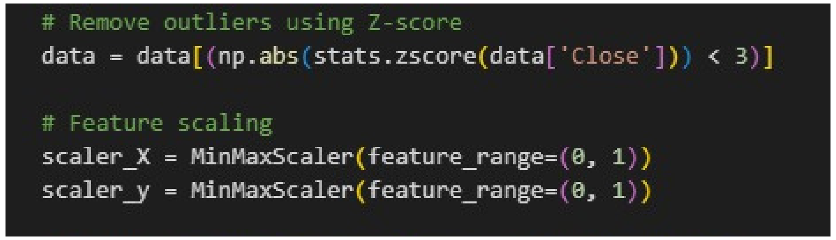
  EDA helps in understanding the data distribution, correlations, missing values, and outliers.

Raw File Charecterstics



For price prediction ‘Close’ was choosen to be the target and remaining as features.

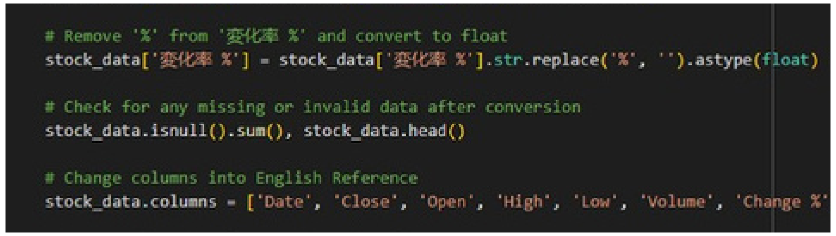
Pre- Processing



```
# Remove outliers using Z-score
data = data[(np.abs(stats.zscore(data['Close'])) < 3)]

# Feature scaling
scaler_X = MinMaxScaler(feature_range=(0, 1))
scaler_y = MinMaxScaler(feature_range=(0, 1))
```

Outlier Detection and Removal

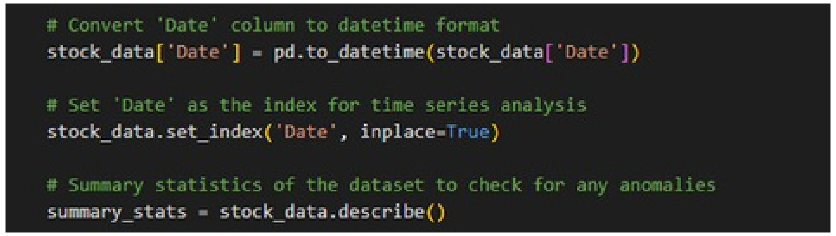
Outliers can distort statistical models and machine learning algorithms. Outliers from Close columns are removed by z score method



```
# Remove '%' from '变化率 %' and convert to float
stock_data['变化率 %'] = stock_data['变化率 %'].str.replace('%', '').astype(float)

# Check for any missing or invalid data after conversion
stock_data.isnull().sum(), stock_data.head()

# Change columns into English Reference
stock_data.columns = ['Date', 'Close', 'Open', 'High', 'Low', 'Volume', 'Change %'
```

Handling Missing Data

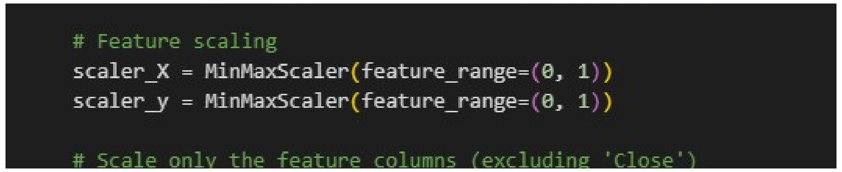
Missing values are common in time series datasets like stock price data. string values in %Change Column converted into float type values and the missing values were replaced by mean or dropped by dropna() funtion



```
# Convert 'Date' column to datetime format
stock_data['Date'] = pd.to_datetime(stock_data['Date'])

# Set 'Date' as the index for time series analysis
stock_data.set_index('Date', inplace=True)

# Summary statistics of the dataset to check for any anomalies
summary_stats = stock_data.describe()
```

Date Parsing

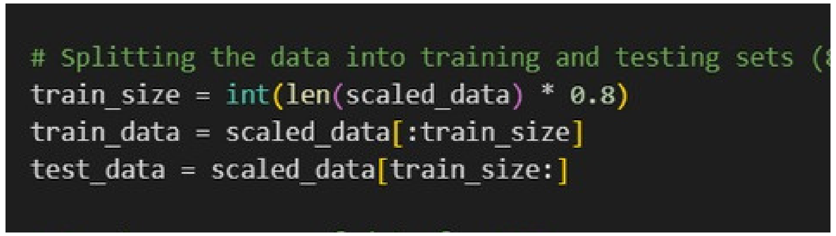
For Time series data we need to add data column and we did it with pandas lib funt() and the date was later used as index



```
  # Feature scaling
  scaler_X = MinMaxScaler(feature_range=(0, 1))
  scaler_y = MinMaxScaler(feature_range=(0, 1))

  # Scale only the feature columns (excluding 'Close')
```

Data Scaling and Normalization

Normalization (Min-Max Scaling): Scales the data to a range between 0 and 1.

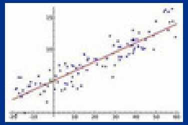


```
# Splitting the data into training and testing sets (
train_size = int(len(scaled_data) * 0.8)
train_data = scaled_data[:train_size]
test_data = scaled_data[train_size:]
```

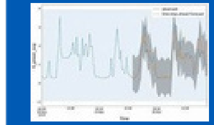
Train-Test Split

For time series data, it’s important to split the data based on time rather than randomly 80% Data used for training

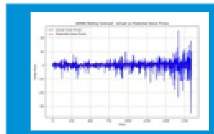
Model Selection - Total 6 Models were used to find best fit for Data


Linear Regression

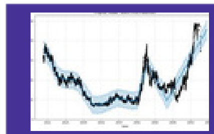
Basic, assumes linear relationships, not ideal for complex time dependencies.


ARIMA Model

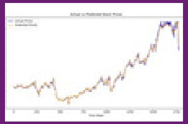
Captures linear trends, seasonality, and autocorrelations, but limited to univariate data.


SARIMA Model

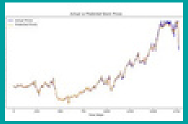
Extends ARIMA with seasonal components, better for seasonal data but still limited to linear relationships.


Prophet Model

Good for handling irregular, missing data, and capturing trends/seasonality, but not as advanced for capturing complex patterns as LSTM.


LSTM Model

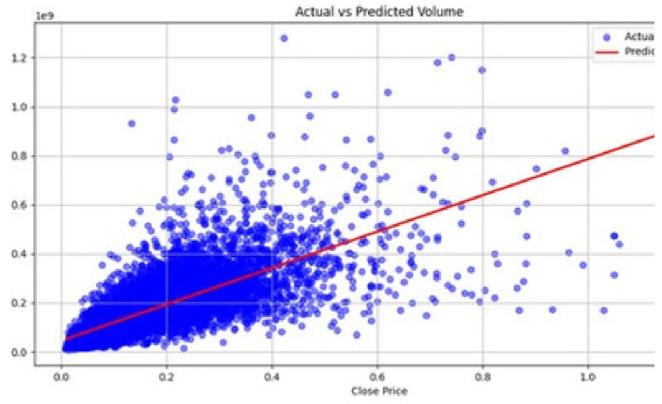
Most effective at capturing complex, non-linear relationships in long-term time series data, handles multivariate inputs well, best for sequential data like stock prices.

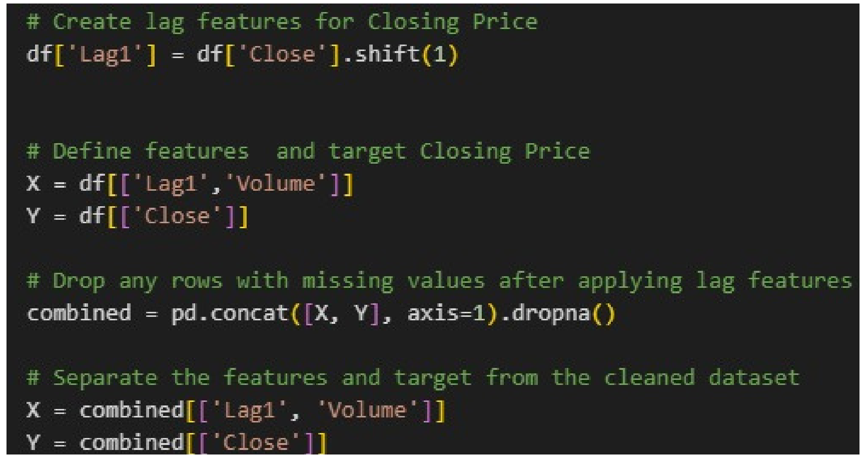

Chronos LSTM Model

Specialized for time series, supports various deep learning models, tailored to handle multivariate and complex patterns.

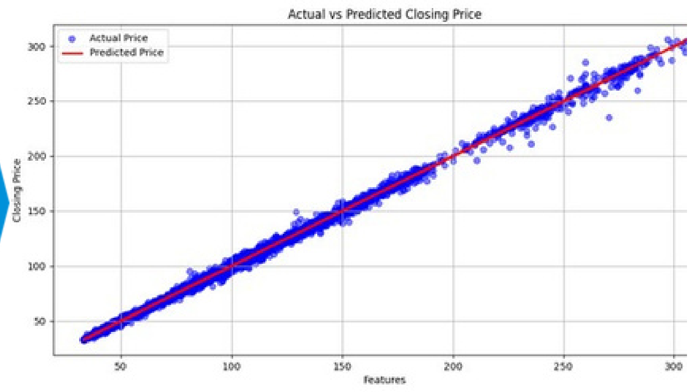
Linear Regression

Drawback: Only Basic features depicted and no seasonality and features were considered , so the model may fail to predict in real time





```
# Create lag features for Closing Price
df['Lag1'] = df['Close'].shift(1)


# Define features  and target Closing Price
X = df[['Lag1','Volume']]
Y = df[['Close']]

# Drop any rows with missing values after applying lag features
combined = pd.concat([X, Y], axis=1).dropna()

# Separate the features and target from the cleaned dataset
X = combined[['Lag1', 'Volume']]
Y = combined[['Close']]
```



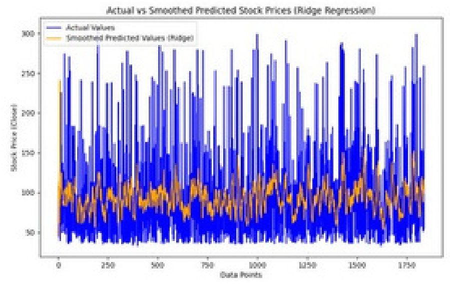
Check for Featuring Variable eg. [Volume] Column

Incrementing Nature of graph shows the direct relevance of Volume and closing price and hence it can be choose as Feature

Create Lag Features: eg. [Close] Column

Feature Engineering used to develop Lag column for Close values shifted by 1 block and added as feature in Linear Model

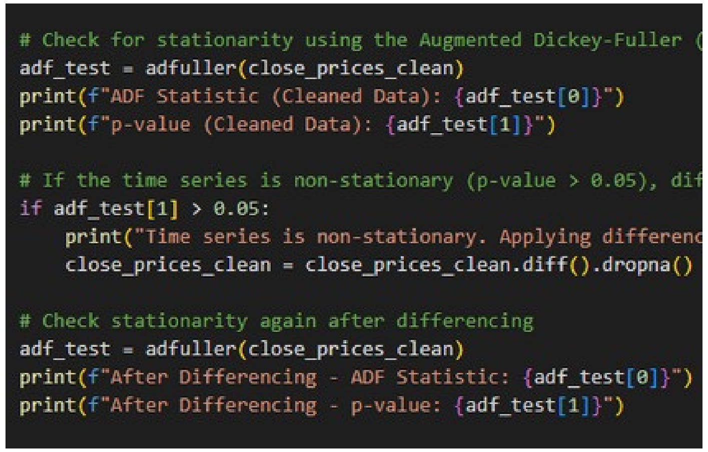
Final Plot After Removing Overfitting eg. All features were not included to avoid overfitting



Ridge Regression was performed to further increase the clarity and efficiency of model but this is not included in code file because its old and outdated

ARIMA Model

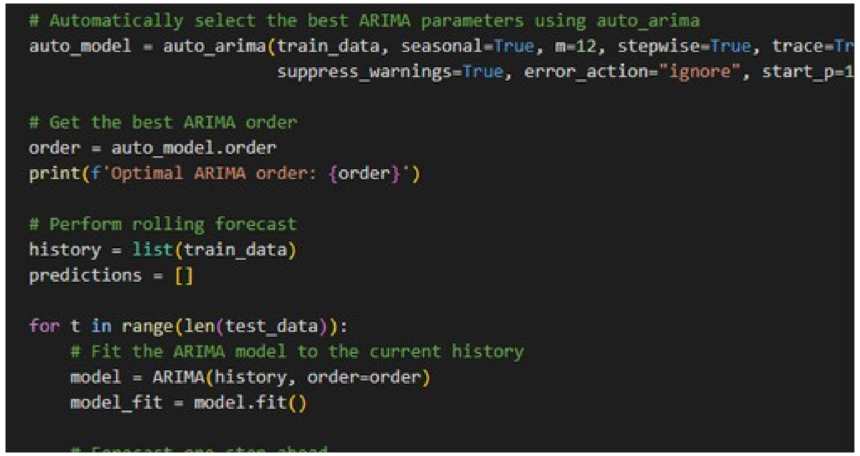
Drawback: Only applicable on stationary data and seasonality is not included upto much extent



```
# Check for stationarity using the Augmented Dickey-Fuller (
adf_test = adfuller(close_prices_clean)
print(f"ADF Statistic (Cleaned Data): {adf_test[0]}")
print(f"p-value (Cleaned Data): {adf_test[1]}")

# If the time series is non-stationary (p-value > 0.05), dif
if adf_test[1] > 0.05:
    print("Time series is non-stationary. Applying differenc
    close_prices_clean = close_prices_clean.diff().dropna()

# Check stationarity again after differencing
adf_test = adfuller(close_prices_clean)
print(f"After Differencing - ADF Statistic: {adf_test[0]}")
print(f"After Differencing - p-value: {adf_test[1]}")
```

Check Stationarity
eg. [Close] Column

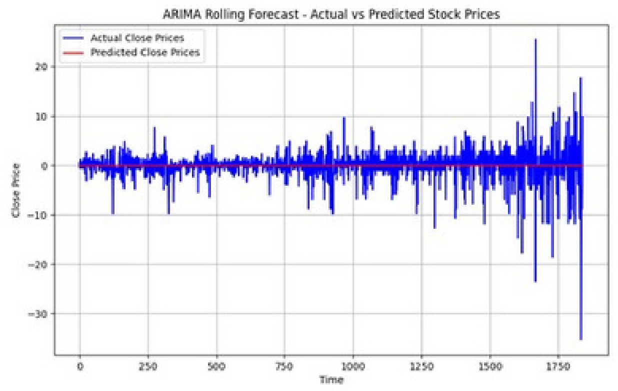
ARIMA Model is used for Stationary values model so p-value is greater than 0.05 than value is stationary else diffrence is applied to make it st.



```
# Automatically select the best ARIMA parameters using auto_arima
auto_model = auto_arima(train_data, seasonal=True, m=12, stepwise=True, trace=Tr
                        suppress_warnings=True, error_action="ignore", start_p=1

# Get the best ARIMA order
order = auto_model.order
print(f'Optimal ARIMA order: {order}')

# Perform rolling forecast
history = list(train_data)
predictions = []

for t in range(len(test_data)):
    # Fit the ARIMA model to the current history
    model = ARIMA(history, order=order)
    model_fit = model.fit()
```

Select ARIMA Parameters (p, d, q):
eg. [Close] Column

Lag , First Differencing and MA terms (p,q,d) is applied and then model is fitted with these values



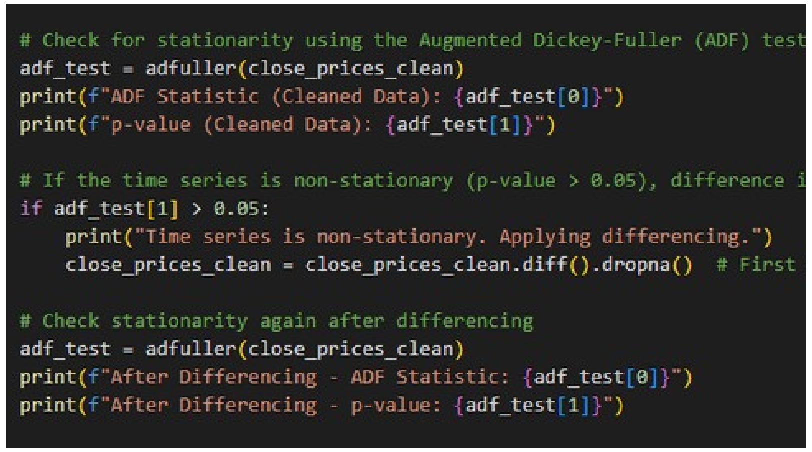
Final Plot After appling MSE AND MAE

Output was order (0,0,0) which means that model didn’t find and trend or pattern in the data

SARIMA - Extends ARIMA with seasonal components, better for s
stands for Seasonality

SARIMA Model

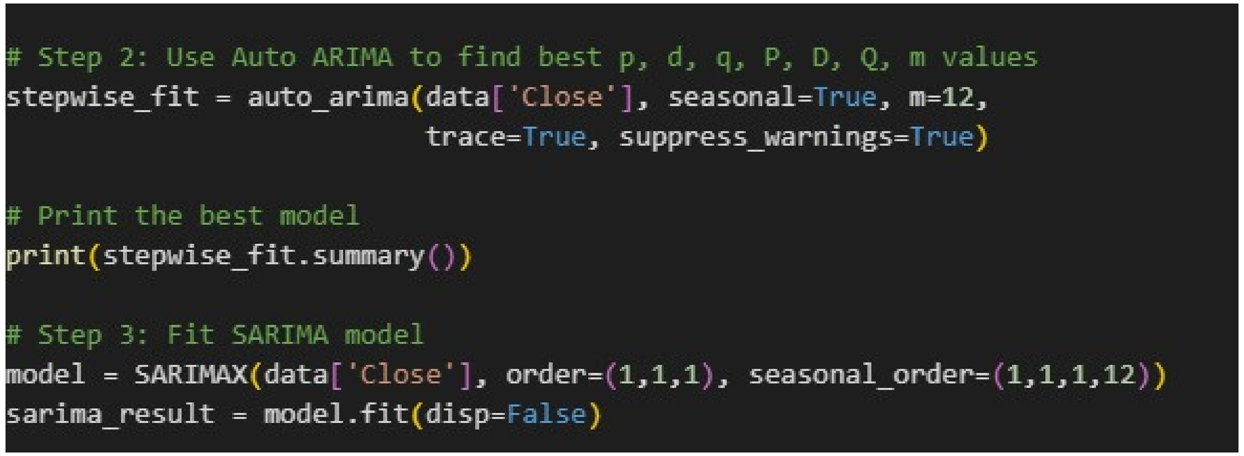
Drawback: Only applicable on stationary data and it is still limited to Linear relationships



```
# Check for stationarity using the Augmented Dickey-Fuller (ADF) test
adf_test = adfuller(close_prices_clean)
print(f"ADF Statistic (Cleaned Data): {adf_test[0]}")
print(f"p-value (Cleaned Data): {adf_test[1]}")

# If the time series is non-stationary (p-value > 0.05), difference i
if adf_test[1] > 0.05:
    print("Time series is non-stationary. Applying differencing.")
    close_prices_clean = close_prices_clean.diff().dropna()  # First

# Check stationarity again after differencing
adf_test = adfuller(close_prices_clean)
print(f"After Differencing - ADF Statistic: {adf_test[0]}")
print(f"After Differencing - p-value: {adf_test[1]}")
```

Check Stationarity
eg. [Close] Column

SARIMA Model is used for Stationary values model so p-value is greater than 0.05 than value is stationary else diffrence is applied to make it st.



```
# Step 2: Use Auto ARIMA to find best p, d, q, P, D, Q, m values
stepwise_fit = auto_arima(data['Close'], seasonal=True, m=12,
                          trace=True, suppress_warnings=True)

# Print the best model
print(stepwise_fit.summary())

# Step 3: Fit SARIMA model
model = SARIMAX(data['Close'], order=(1,1,1), seasonal_order=(1,1,1,12))
sarima_result = model.fit(disp=False)
```

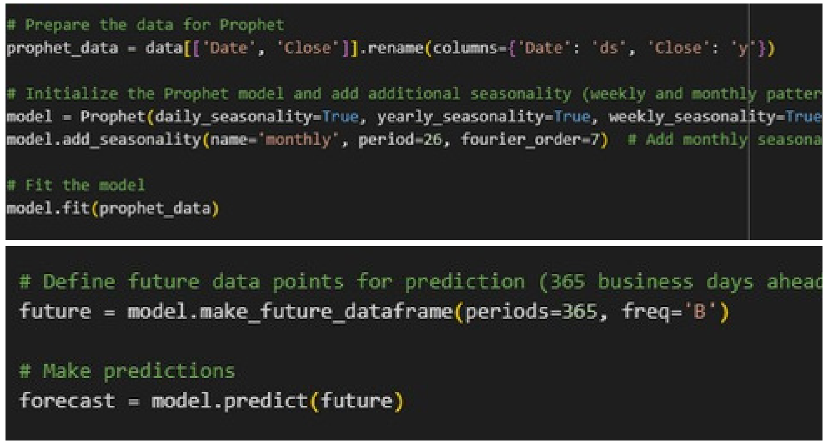
Select SARIMA Parameters (p, d, q,M):
eg. [Close] Column

Lag , First Differencing and MA terms (p,q,d) is applied and then model is fitted with these values and seasonality is added by adding ‘m’ which represents seasonality cycles (m=12 means 12 months)

SARIMA - Was proved better than ARIM A b u t d a t a s till have sudd
which needed to be analysed so we moved to **Prophet model by hugging face** to analyse the Tend in Dataset

Prophet Model

Drawback: It is limited to less complex patterns and can’t handle more complex patterns and forecasting as depicted in 2nd graph

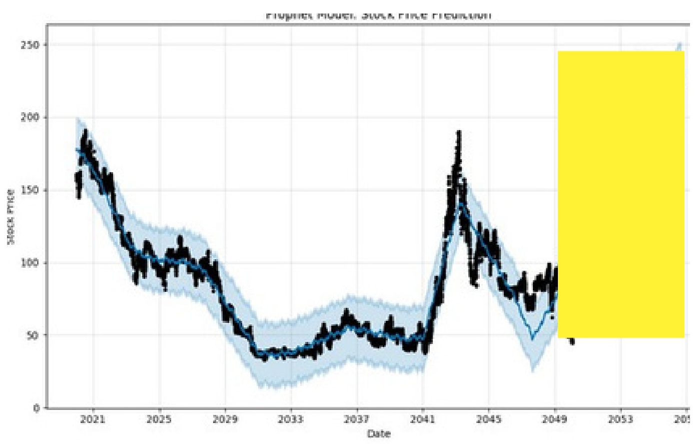


```
# Prepare the data for Prophet
prophet_data = data[['Date', 'Close']].rename(columns={'Date': 'ds', 'Close': 'y'})

# Initialize the Prophet model and add additional seasonality (weekly and monthly patter
model = Prophet(daily_seasonality=True, yearly_seasonality=True, weekly_seasonality=True
model.add_seasonality(name='monthly', period=26, fourier_order=7)  # Add monthly seasona

# Fit the model
model.fit(prophet_data)
```

```
# Define future data points for prediction (365 business days ahead
future = model.make_future_dataframe(periods=365, freq='B')

# Make predictions
forecast = model.predict(future)
```

Seasonality and Future Data points added

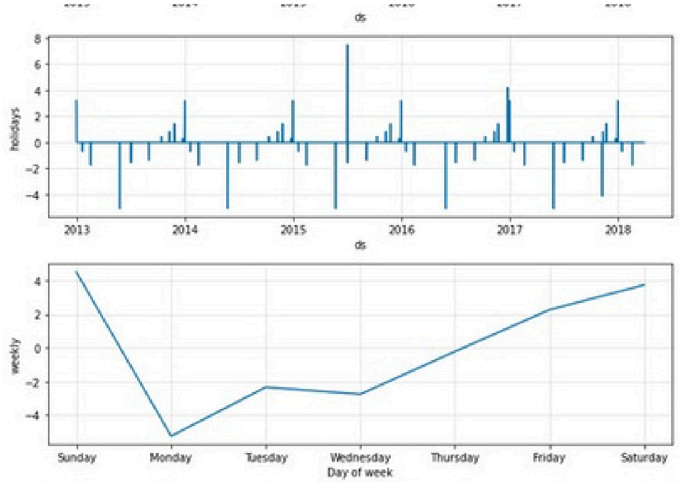
Prophet is more advanced model to deal with monthly, weekly and yearly seasonality. It further forecast the future trend by date column on x-axis



Yearly Plot of Trend

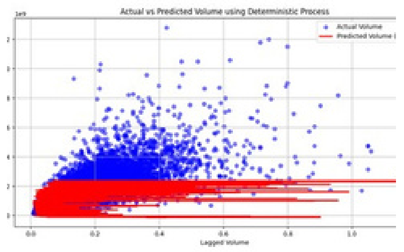
eg. [Close] Column

The error in Future years (towards the end of graph) is due to lack of inavalibility of data . Can be fixed with future models or with more avaliable data



Trend of all Seasonality

Weekly , Monthly and Yearly Trend is captured also Propeht allows us to have **Responsive data analysis and charts**



Before moving to next m o d e l I t r ie d imple m e n t i

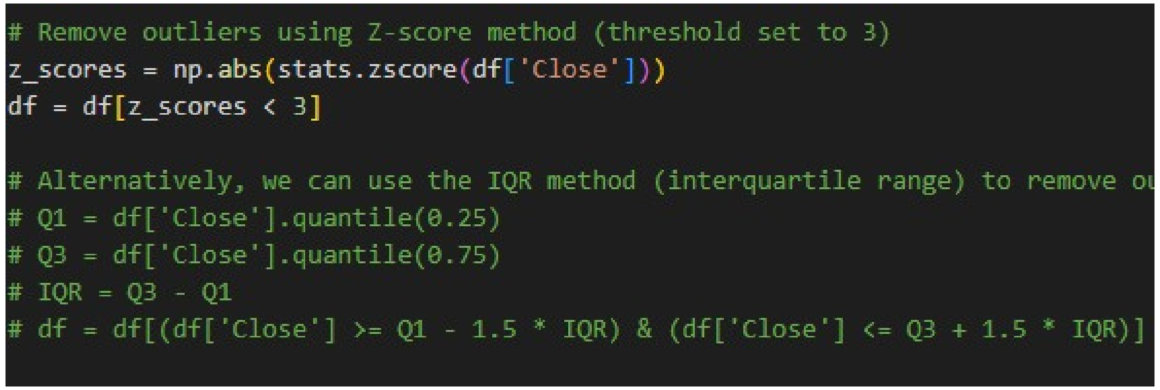
for trend and pattern in 2nd and 3rd degree equations instead of linear.

eg .instead of y = mx+tx+px+c I moved to y=mx^2+p^3x^2+p^2x^3 etc

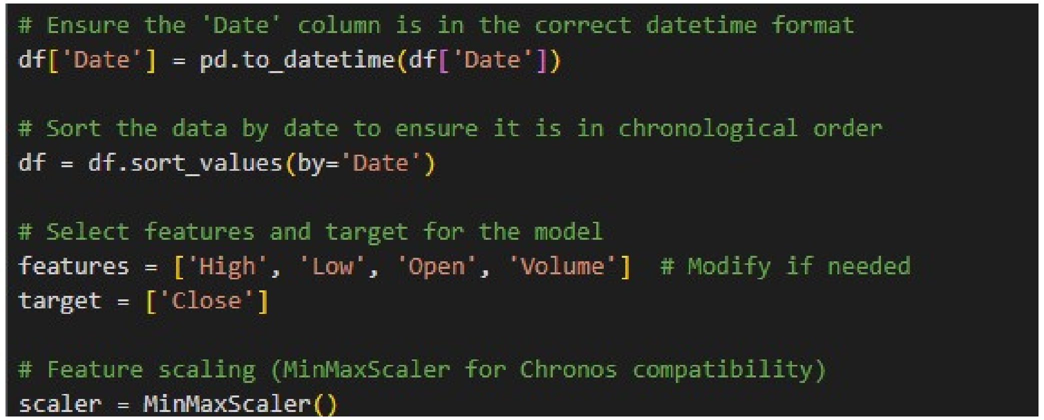
Wasn’t much successfull but worth mentioning because it can be applied to other datasets

Chronos Model

Drawback: Complex to run and need higher level GPU to run as it takes too much time



```
# Remove outliers using Z-score method (threshold set to 3)
z_scores = np.abs(stats.zscore(df['Close']))
df = df[z_scores < 3]

# Alternatively, we can use the IQR method (interquartile range) to remove ou
# Q1 = df['Close'].quantile(0.25)
# Q3 = df['Close'].quantile(0.75)
# IQR = Q3 - Q1
# df = df[(df['Close'] >= Q1 - 1.5 * IQR) & (df['Close'] <= Q3 + 1.5 * IQR)]
```



```
# Ensure the 'Date' column is in the correct datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Sort the data by date to ensure it is in chronological order
df = df.sort_values(by='Date')

# Select features and target for the model
features = ['High', 'Low', 'Open', 'Volume']  # Modify if needed
target = ['Close']

# Feature scaling (MinMaxScaler for Chronos compatibility)
scaler = MinMaxScaler()
```

IQR Method used for outliers

Instead of Dropping the outliers but z-score we can use IQR method to fill null places with upper and lower quartile values

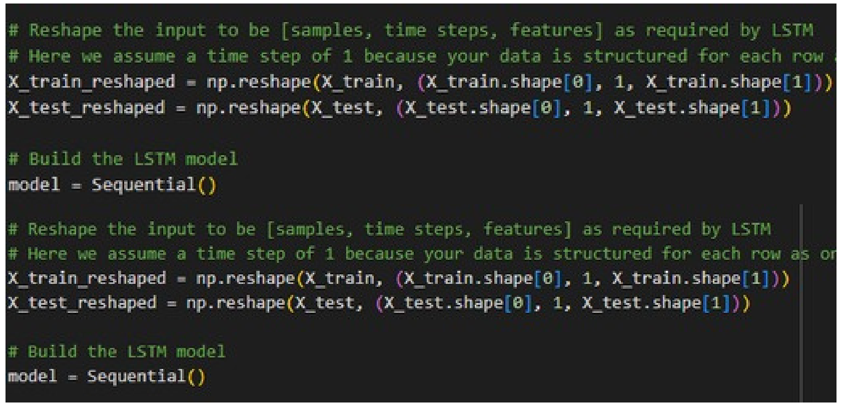
Collected all features avaliable

This Model Is one of the most advanced model as it used all the features in dataset to form collective prediction (Cloe , Open , High, Low)

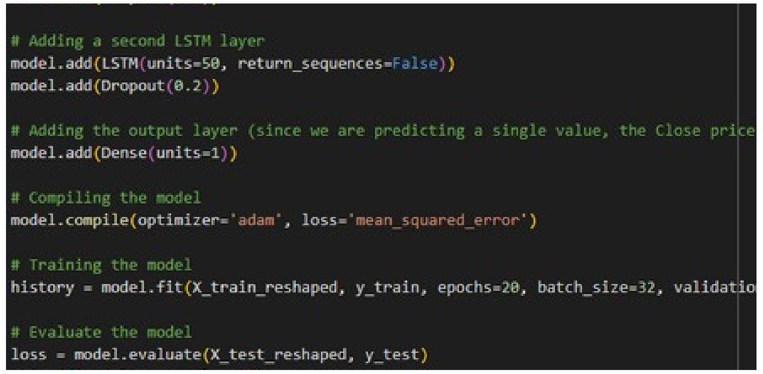
Chronos Model LSTM is one of the mo s t a d v a n c e d method but it t
and we can achieve the similar result with the **LSTM Method and it will be one of the best prediction method in all 6 methods because it removes all the drawbacks**

LSTM Model

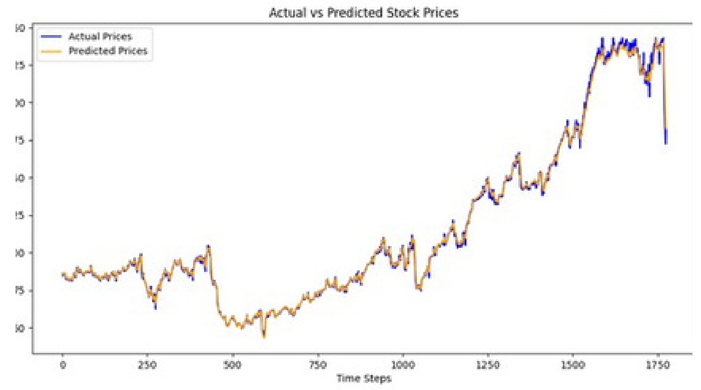
Drawback: LSTM models have several hyperparameters (e.g., number of layers, units per layer, dropout rates) that need to be carefully tuned.



```
# Reshape the input to be [samples, time steps, features] as required by LSTM
# Here we assume a time step of 1 because your data is structured for each row
X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()

# Reshape the input to be [samples, time steps, features] as required by LSTM
# Here we assume a time step of 1 because your data is structured for each row as or
X_train_reshaped = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test_reshaped = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))

# Build the LSTM model
model = Sequential()
```



```
# Adding a second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))

# Adding the output layer (since we are predicting a single value, the Close price
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Training the model
history = model.fit(X_train_reshaped, y_train, epochs=20, batch_size=32, validatio

# Evaluate the model
loss = model.evaluate(X_test_reshaped, y_test)
```



Reshape the data columns

Data Columns are reshaped according to LSTM fitting data in 3d model by sample ,time steps and data format

Add Layer of LSTM like Neural networks

Input Layer consist of 3d data , LSTM Layer consist of RNN layers that handle complex pattern and give to last layer - Dropout Layer

Final Plot

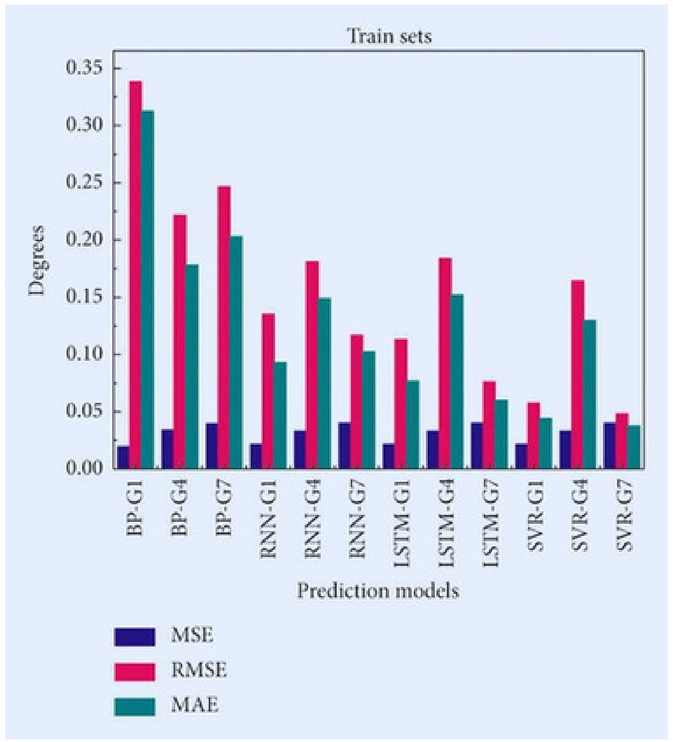
Final Plot was revived after decreasing 2 layers so as to minimize the time without affecting efficiency
Epoch was also decreased to 20

After studying all these models the **LSTM model** was one of the m
model that can be adopted for Stock Prdiction because of following features:

- Memory Cells: LSTMs use memory cells to store information over long periods, allowing the model to learn dependencies that span across time steps.
- Gates: LSTMs utilize input, output, and forget gates to control the flow of information, making them effective for sequence prediction tasks.
- Robust to Noise: LSTMs can handle noise and outliers in time series data, making them suitable for real-world applications.

Evaluation Indicators

While all models have their strengths, LSTM stands out due to its ability to model complex relationships and temporal dynamics effectively.



Indicators	Description
Mean Absolute Error	• Measures the average magnitude of errors in a set of predictions, without considering their direction. • Lower MAE indicates better predictive performance.
Mean Square Error	• Calculates the average of the squares of the errors, giving more weight to larger errors. • Lower MSE signifies better model accuracy.
Mean Percentage Error	• Measures the accuracy as a percentage, allowing for easy interpretation. • Lower MAPE values indicate better performance.

While all models have their strengths, LSTM stands out due to its ability to model complex relationships and temporal dynamics effectively. The combination of various evaluation indicators typically shows that LSTM models consistently achieve lower MAE, MSE, and RMSE values, making them the best choice for time series forecasting, especially in volatile domains like stock prices.

Verification Results

- **Hypothesis 1: Model Improvement through feature selection**

  By selecting relevant features that contribute most to the prediction of stock prices, we can improve the model's accuracy and efficiency, leading to a better predictive performance compared to using a broader set of features.

- Correlation Analysis: Use correlation coefficients to identify highly correlated features and retain only one feature from each highly correlated pair.

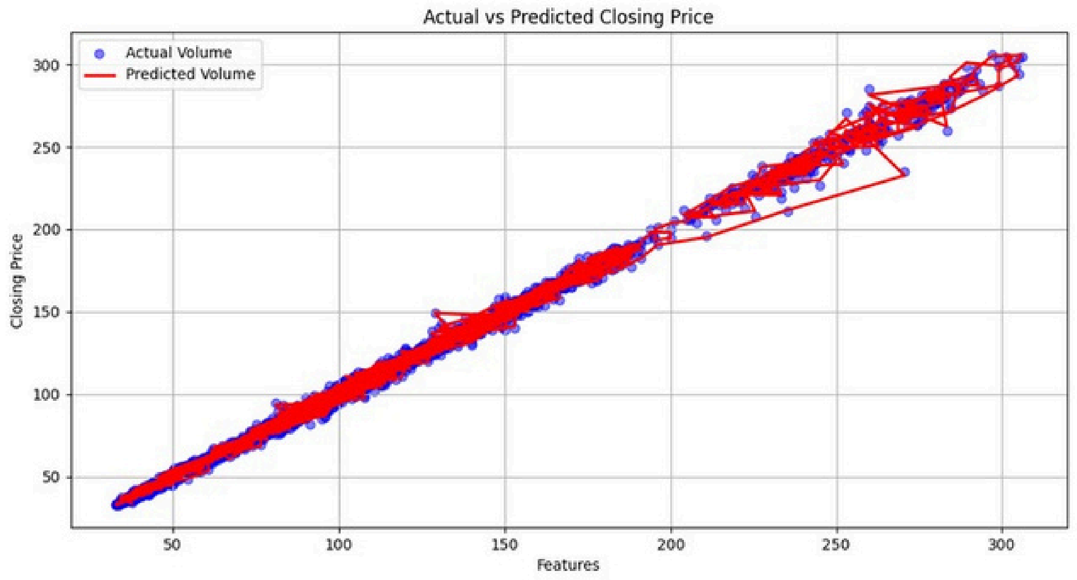
- Feature Importance from Models: Use algorithms (e.g., Deterministic Model, Ridge regression) to assess feature importance and retain the top N features based on their importance scores.

- Recursive Feature Elimination (RFE): Implement RFE to recursively remove features and evaluate model performance until the optimal number of features is reached.

Co-Relation Analysis

Use correlation coefficients to identify highly correlated features and retain only one feature from each highly correlated pair.

Example : In Linear regression we didn’t included all feature because the model was depicting over fitting and hence less accuracy so to avoid the problem we will use Lag and Trend charecterstics to include only the high corelated features like IQR , LAG etc



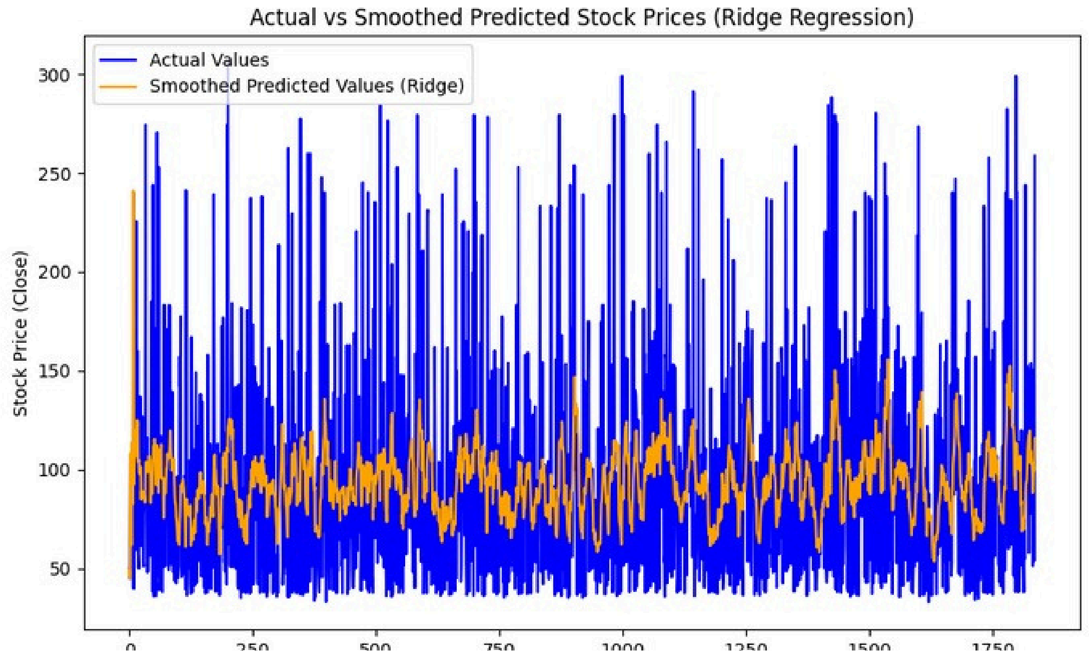
Expected Outcome:

Increased Predictive Accuracy: The model is expected to demonstrate improved accuracy metrics (e.g., lower MAE, MSE, RMSE) after applying feature selection.

Feature Importance

Use algorithms (e.g., Deterministic Model, Ridge regression) to assess feature importance and retain the top N features based on their importance scores

Example : In linear models the complex patterns are often over shadow so to get max accuracy we can list top N features by using alternative features and limit them like Quad instead of linear model and updating the data features on each iterations



- Ridge Regression on quad polymer model for same data set is better than linear model

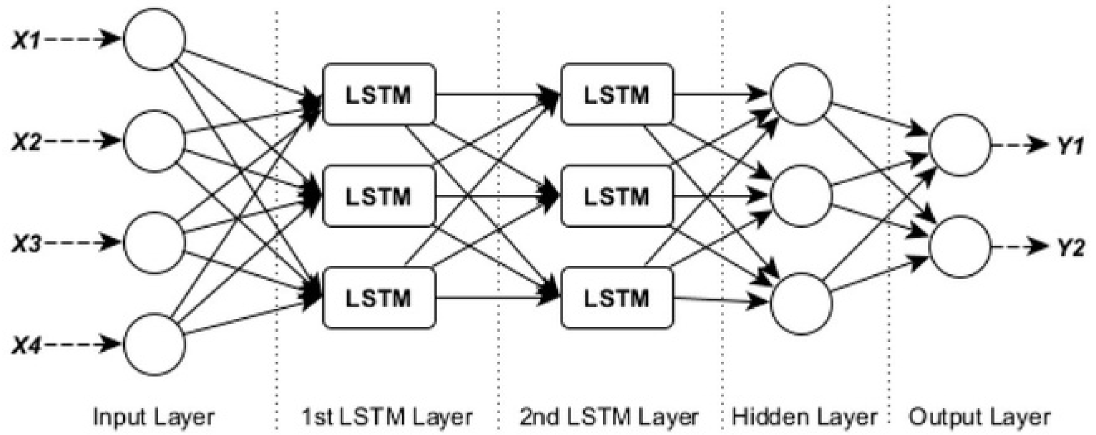
Expected Outcome:

Faster Training Time: The training process is anticipated to be quicker due to a reduced feature set, leading to more efficient experimentation.

Recursive Feature Elimination (RFE)

Implement RFE to recursively remove features and evaluate model performance until the optimal number of features is reached.

- lstm_regressor = KerasRegressor(build_fn=create_lstm_model, epochs=50, batch_size=32, verbose=0)
- selector = RFE(estimator=lstm_regressor, n_features_to_select=5, step=1) selector = selector.fit(X_train_reshaped, y_train)
- selected_features = selector.support_ print(f"Selected Features: {np.where(selected_features)[0]}")



Expected Outcome:

Better Generalization: The selected feature set is likely to enhance the model's ability to generalize to new, unseen data, thereby improving performance on validation and test sets

- LSTM can use RFE model to elimate the lag in time without hurting efficiency

Verification Results

- **Hypothesis 2: Effects of data preprocessing on accuracy.**

  Effective data pre-processing, including techniques such as handling missing values, outlier removal, normalization, and feature selection, significantly improves the accuracy of predictive models for stock price forecasting.

- Handling Missing Values - Filling Missing Values
- Outlier Removal - Noise removal from raw data set
- Normalization and Scaling - Data scaling gives better result in forecasting
- Feature Selection - Select Feature best for dataset and add feature from own analysis

Handling missing Values

Null Values: Missing data can lead to inaccurate predictions and biased results. Proper imputation methods (mean, median, mode, or advanced techniques like KNN imputation) can help retain valuable information.

Expected Outcome:

Models trained on datasets with appropriately handled missing values will demonstrate higher accuracy compared to those with unaddressed missing values.

Expected Outcome:

Models trained on datasets with removed outliers will yield lower error rates and higher predictive performance.

Outlier Removal

Impact of Outliers: Outliers can skew the results and mislead the model's learning process. Identifying and removing outliers using techniques like Z-score or IQR can stabilize the data distribution.

Normalization and Scaling

Normalizing or standardizing features (e.g., Min-Max scaling or Z-score normalization) ensures that all features contribute equally to model training, especially in algorithms sensitive to feature scales (**e.g., LSTM)**.

Expected Outcome:

Models using normalized datasets will demonstrate better convergence and performance metrics compared to those using raw, unscaled features.

Expected Outcome:

Models trained on a carefully selected subset of features will have higher accuracy and lower complexity compared to models using all available features.

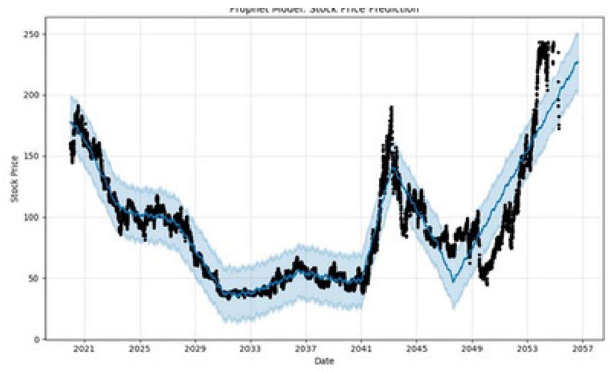
Feature Selection

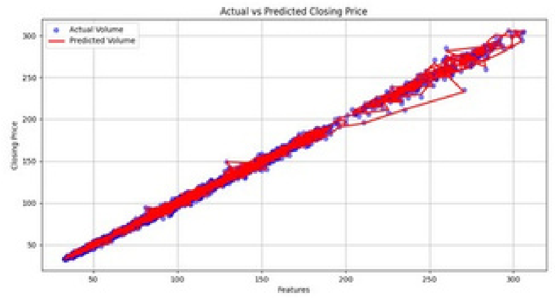
Using techniques such as Recursive Feature Elimination (RFE) or other feature selection methods can help in identifying the most relevant features for model training.

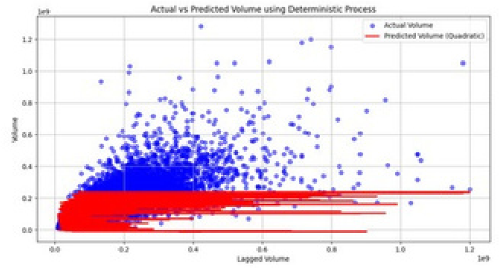
Summary of Results

- Linear Regression: Simple model but struggled with non-linearity and time dependencies in stock data, resulting in suboptimal predictions.
- ARIMA/SARIMA: Good for stationary data, but lacked flexibility for handling complex patterns like seasonality and trends over long periods.
- Prophet: Efficient at detecting seasonality and trend changes but had limited accuracy for short-term predictions compared to more complex models.
- LSTM: Outperformed other models due to its ability to capture long-term dependencies and sequential patterns in time series data. It was more effective in handling the non-linearity and temporal correlations of stock prices.

- LSTM's Superiority: LSTM outperformed simpler models like Linear Regression, which struggled with time dependencies, and more traditional models like ARIMA/SARIMA, which were limited in handling dynamic patterns in non-stationary data. Even models designed for trend and seasonality detection, such as Prophet, were less accurate in capturing the short-term price fluctuations compared to LSTM.







- Evaluation Metrics: LSTM demonstrated better performance across key evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), confirming its suitability for stock price forecasting tasks where accuracy and handling of sequential patterns are critical.

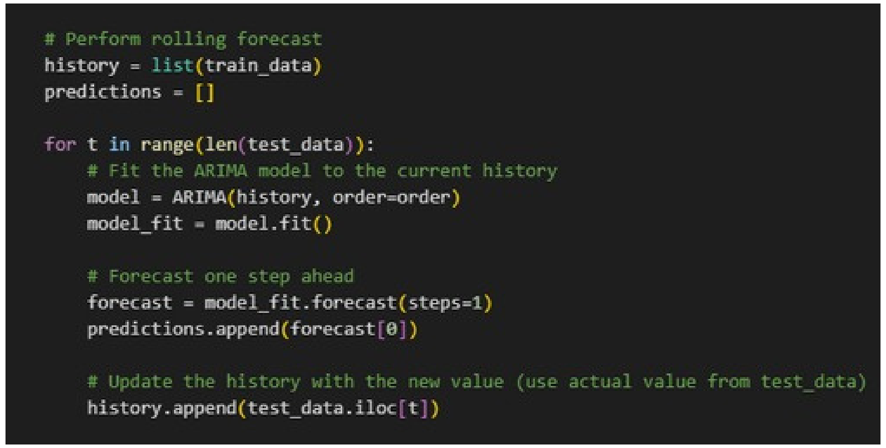
Future Outlook

Looking ahead, further improvement in LSTM's performance can be achieved by integrating more advanced feature engineering, optimizing hyperparameters, and using data preprocessing techniques such as outlier removal and normalization.

- Feature Engineering
- Optimizing Hyperparameters
- Pre- processing Techniques
- Outlier Removal

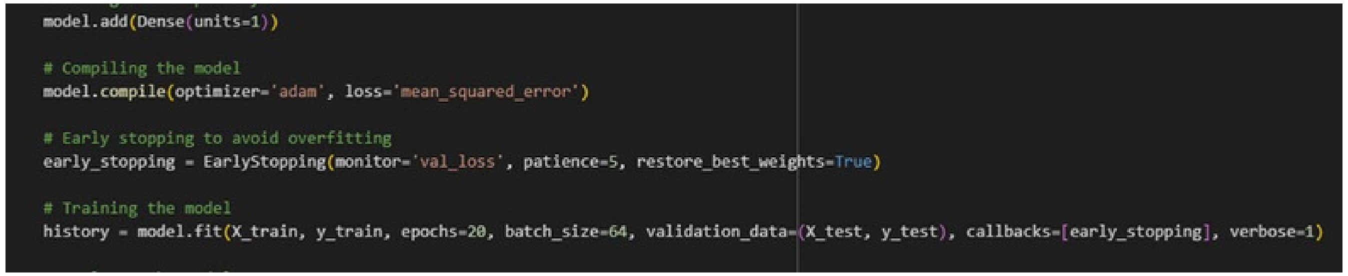
Advanced Feature Engineering

For stock prediction, features like **Moving averages**, **Rolling forecast, EarlyStopping** or technical indicators (e.g., Relative Strength Index, Bollinger Bands) can be added. These features provide LSTM with additional signals to learn complex trends and patterns in stock prices, beyond basic inputs like Open, High, Low, Close, and Volume.



```
# Perform rolling forecast
history = list(train_data)
predictions = []

for t in range(len(test_data)):
    # Fit the ARIMA model to the current history
    model = ARIMA(history, order=order)
    model_fit = model.fit()

    # Forecast one step ahead
    forecast = model_fit.forecast(steps=1)
    predictions.append(forecast[0])

    # Update the history with the new value (use actual value from test_data)
    history.append(test_data.iloc[t])
```

In a rolling forecast, the model is trained on a window of past data and then updated periodically as new data arrives. After each new data point is added, the model is retrained using the most recent set of data points (a sliding window), ensuring it adapts to changing trends.

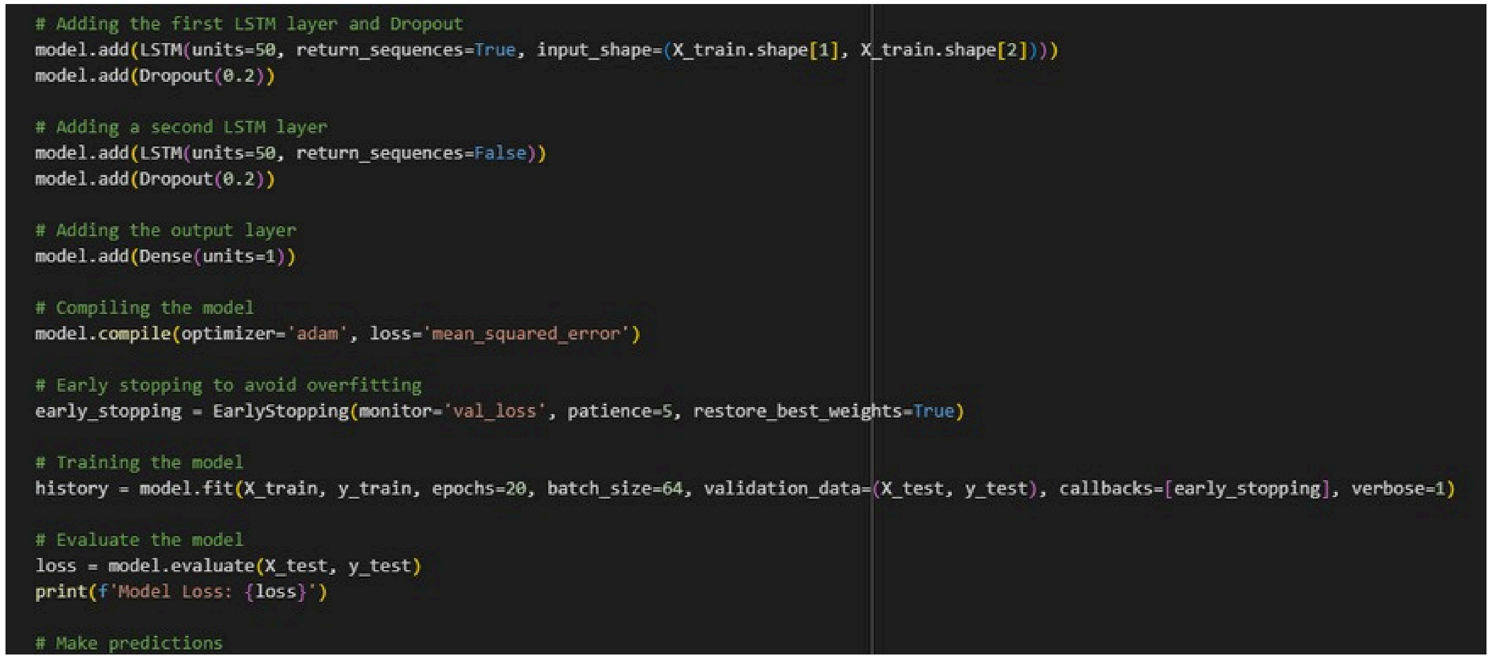


```
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Training the model
history = model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_test, y_test), callbacks=[early_stopping], verbose=1)
```

Early stopping is a form of regularization used during training. It monitors the model's performance on a validation set, and when performance (e.g., accuracy or loss) starts to worsen after a certain number of epochs, training is stopped to avoid overfitting to the training data.

Optimizing HyperParametrs

Hyperparameters are values set before the learning process, such as the **learning rate, number of layers, units in each layer, dropout rates, epochs and batch size**. Unlike model parameters, hyperparameters control how the model learns and how fast it converges.

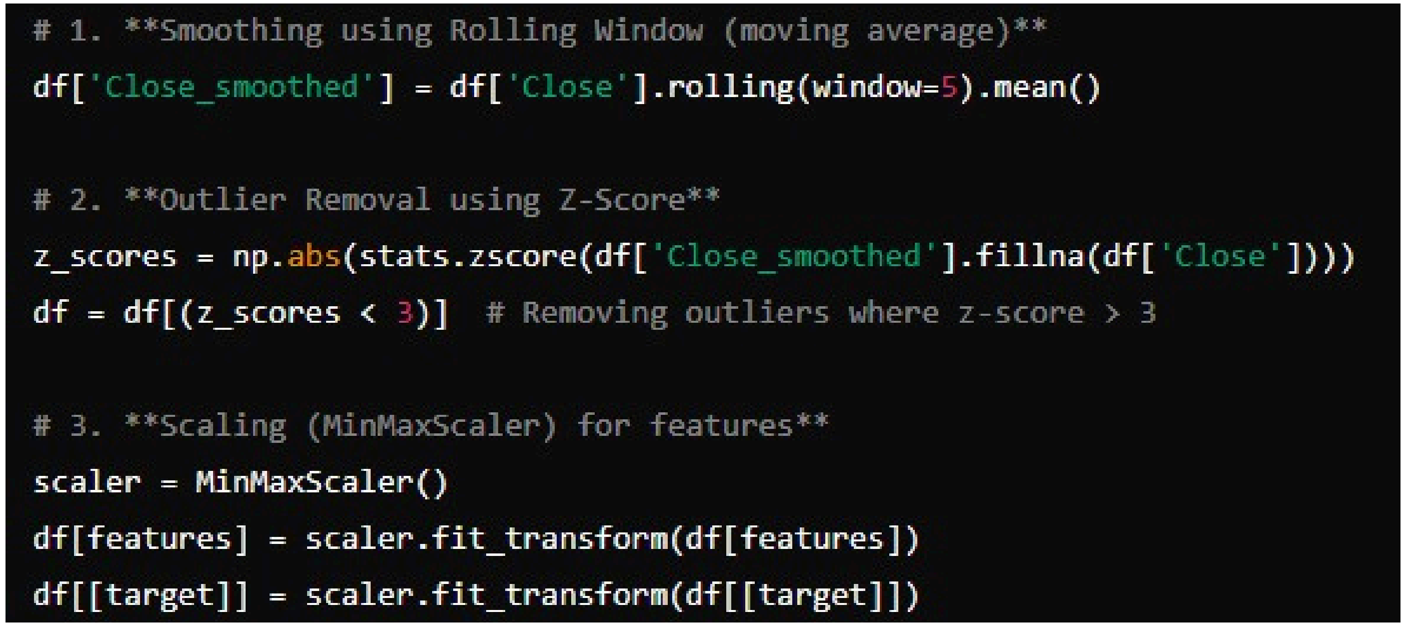


```
# Adding the first LSTM layer and Dropout
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(0.2))

# Adding a second LSTM layer
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))

# Adding the output layer
model.add(Dense(units=1))

# Compiling the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Training the model
history = model.fit(X_train, y_train, epochs=20, batch_size=64, validation_data=(X_test, y_test), callbacks=[early_stopping], verbose=1)

# Evaluate the model
loss = model.evaluate(X_test, y_test)
print(f'Model Loss: {loss}')

# Make predictions
```

- **Grid Search: Tries all combinations of hyperparameters within a specified range. Though exhaustive, it is computationally expensive, especially for LSTM models with many hyperparameters.**
- **Random Search: Randomly samples combinations of hyperparameters within a range. It's more efficient than grid search and can find good combinations faster.**
- **Bayesian Optimization: A more sophisticated approach that builds a probabilistic model to predict the best hyperparameters based on previous trials, optimizing the search process efficiently.**
- **Automated Hyperparameter Tuning Tools (AutoML): Tools like Optuna or Hyperpt can be used to automate the process of finding the best hyperparameters for the LSTM model.**

Pre-Processing Techniques and Outliers Removal

These techniques not only enhance the model's predictive accuracy but also ensure more reliable and robust forecasts by minimizing noise and irrelevant variations in the data. Here's how they shape the future outlook of **LSTM models**:



```
# 1. **Smoothing using Rolling Window (moving average)**
df['Close_smoothed'] = df['Close'].rolling(window=5).mean()

# 2. **Outlier Removal using Z-Score**
z_scores = np.abs(stats.zscore(df['Close_smoothed'].fillna(df['Close'])))
df = df[(z_scores < 3)]  # Removing outliers where z-score > 3

# 3. **Scaling (MinMaxScaler) for features**
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
df[[target]] = scaler.fit_transform(df[[target]])
```

- **Smoothing methods (e.g., moving averages, exponential smoothing) help remove short-term fluctuations and highlight long-term trends in time series data like stock prediction**

- **Future Outlook: More sophisticated smoothing techniques like seasonal decomposition or wavelet transformations may be employed to decompose time series into trend, seasonal, and residual components before feeding them into the LSTM model, enhancing trend prediction accuracy.**