Lambton College

STOCK PORTFOLIO RECOMMENDATION AND RISK IDENTIFICATION

USING TIME SERIES FORECASTING & REINFORCEMENT LEARNING

PROJECT MILESTONE- 1 & 2

Data Processing, Preprocessing and Initial Model Building

Presenter : Team Everest

Team Members:
Bharat, Babita, Janvi, Gurpreet, Karmjeet, Valentine, Anmol, Ramandeep, Sarabjot, Lateef

Rationale of the Project

What is an issue?

Investors struggle with stock price volatility, poor risk management, and data overload.

Why is it an issue?

- Inaccurate predictions lead to financial losses.
- Static risk management is ineffective.

Why now?

- Growing demand for personalized financial recommendations
- Economic uncertainty needs real-time insights

How does the project shed light upon this?

- Provides tailored portfolio recommendations.
- Powered by ML algorithms that analyze user preferences and performance metrics.
- Customizable portfolios based on risk levels (low, medium, high).

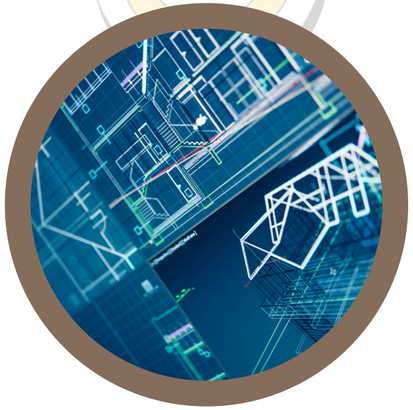
Milestone 1 & 2 Approach

Data Processing and Insights

- Collecting stock price, index, commodity, and currency data from the Yahoo Finance API.
- Using Apache Spark for large-scale data handling and Pandas for data manipulation.
- Building a dashboard to visualize stock trends and provide actionable insights.

Data Preprocessing & Feature Engineering

- Preprocessing tasks such as missing value imputation, outlier detection, and data scaling.
- Feature engineering using technical indicators like 7-day moving averages, Lag feature & Percentage change
- Creating lag variables to capture temporal dependencies in stock prices for time-series forecasting.



Model Building & and Initial Evaluation

- Using Random Forest Regressor for time-series stock price forecasting.
- Initial evaluation using metrics like MSE, RMSE for predictive accuracy.

Data Pre-Processing

```
columns = ['Date', 'Open', 'High', 'Low', 'Close', 'Volume', 'Dividends', 'Stock Splits', 'symbol', 'Capital Gains', 'Adj Close']
# Count total rows
total_counts = sorted_df.count()

# Count missing values for each column
missing_counts = sorted_df.agg(*(F.sum(F.col(col).isNull().cast("int")).alias(col) for col in sorted_df.columns))

# Extract missing values into a dictionary for easy access
missing_dict = missing_counts.first().asDict()

# Calculate non-null counts for each column and print the results
print("\nCounts of non-null values in sorted_df:")
for col in sorted_df.columns:
    non_null_count = total_counts - missing_dict[col]
    print(f"{col}: {non_null_count}, Missing: {missing_dict[col]}")
```

```
[Stage 7:================================================>       (25 + 4) / 29]
Counts of non-null values in sorted_df:
Date: 32236153, Missing: 0
Open: 32236148, Missing: 5
High: 32236148, Missing: 5
Low: 32236148, Missing: 5
Close: 32236149, Missing: 4
Volume: 32236153, Missing: 0
Dividends: 32236153, Missing: 0
Stock Splits: 32236153, Missing: 0
symbol: 32236153, Missing: 0
Capital Gains: 6347106, Missing: 25889047
Adj Close: 0, Missing: 32236153
```

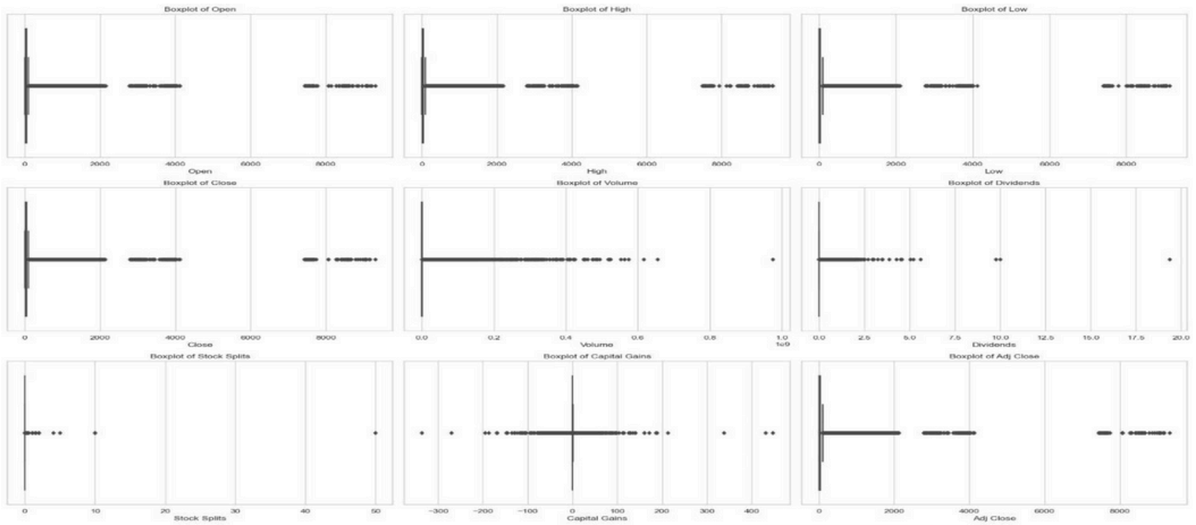
```
pyspark.sql import functions as F
pyspark.sql.window import Window
pyspark.sql.types import DoubleType

st of columns to convert to DoubleType
mns_to_convert = ['Open', 'High', 'Low', 'Close']

st columns to DoubleType
column in columns_to_convert:
sorted_df = sorted_df.withColumn(column, F.col(column).cast(DoubleType()))

eck schema after casting
ed_df.printSchema()

ll missing values with the median
column in columns_to_convert:
# Calculate the median (50th percentile)
median_value = sorted_df.approxQuantile(column, [0.5], 0.01)[0]

# Replace null values with the median value
sorted_df = sorted_df.withColumn(column, F.when(F.col(column).isNull(), F.lit(median_value)).otherwise(F.col(column)))

ow the updated DataFrame
ed_df.show()
```

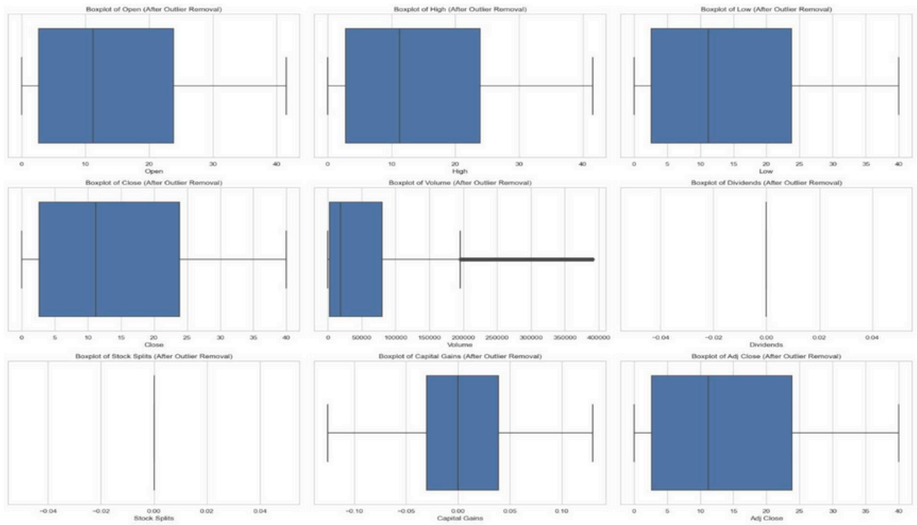
```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Define a window partitioned by 'symbol' for performing the calculation per symbol
windowSpec = Window.partitionBy("symbol").orderBy("Date")

# Calculate the 'Adj Close' using the given logic
sorted_df = sorted_df.withColumn(
    "Adj Close",
    F.when(
        F.col("Stock Splits") == 0,
        F.col("Close") - F.col("Dividends")
    ).otherwise(
        (F.col("Close") - F.col("Dividends")) / F.col("Stock Splits")
    )
)

# Show the result for verification
sorted_df.select("*").orderBy("symbol", "Date").show(10)
```

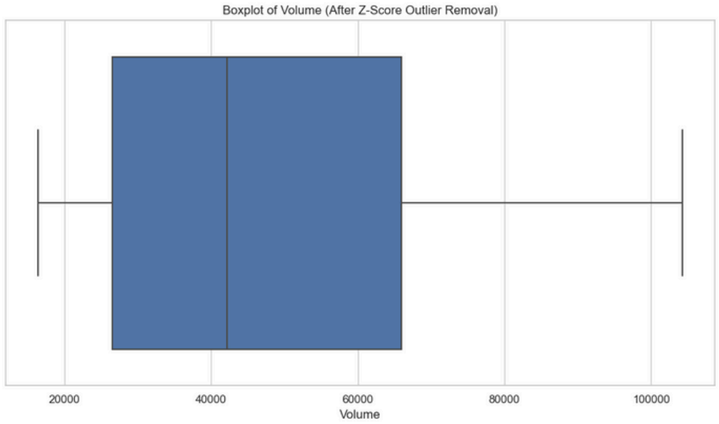
1. *Checking for Missing Values (top left)*
2. *Filling Null Values with Median (top right)*
3. *Calculating Adjusted Close (bottom left)*

Data Insights

1. *Outliers*







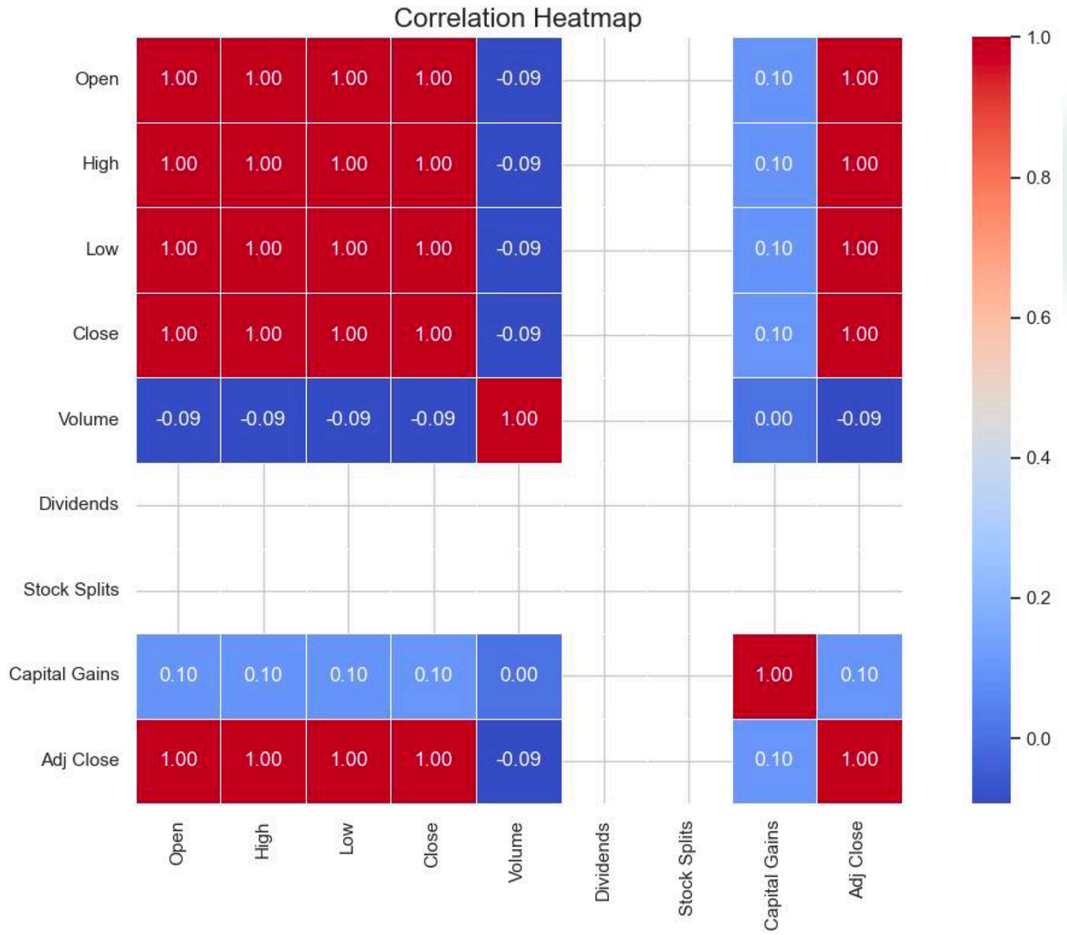
1. *Boxplot to visualize the outliers (left top)*

2. *Boxplot showing the handled outliers, except Volume (using IQR) (right top)*

3. *Boxplot showing that the outlier of the volume column is handled(using Z-Score) (left bottom)*

Data Insights

2. Correlation Matrix Heatmap

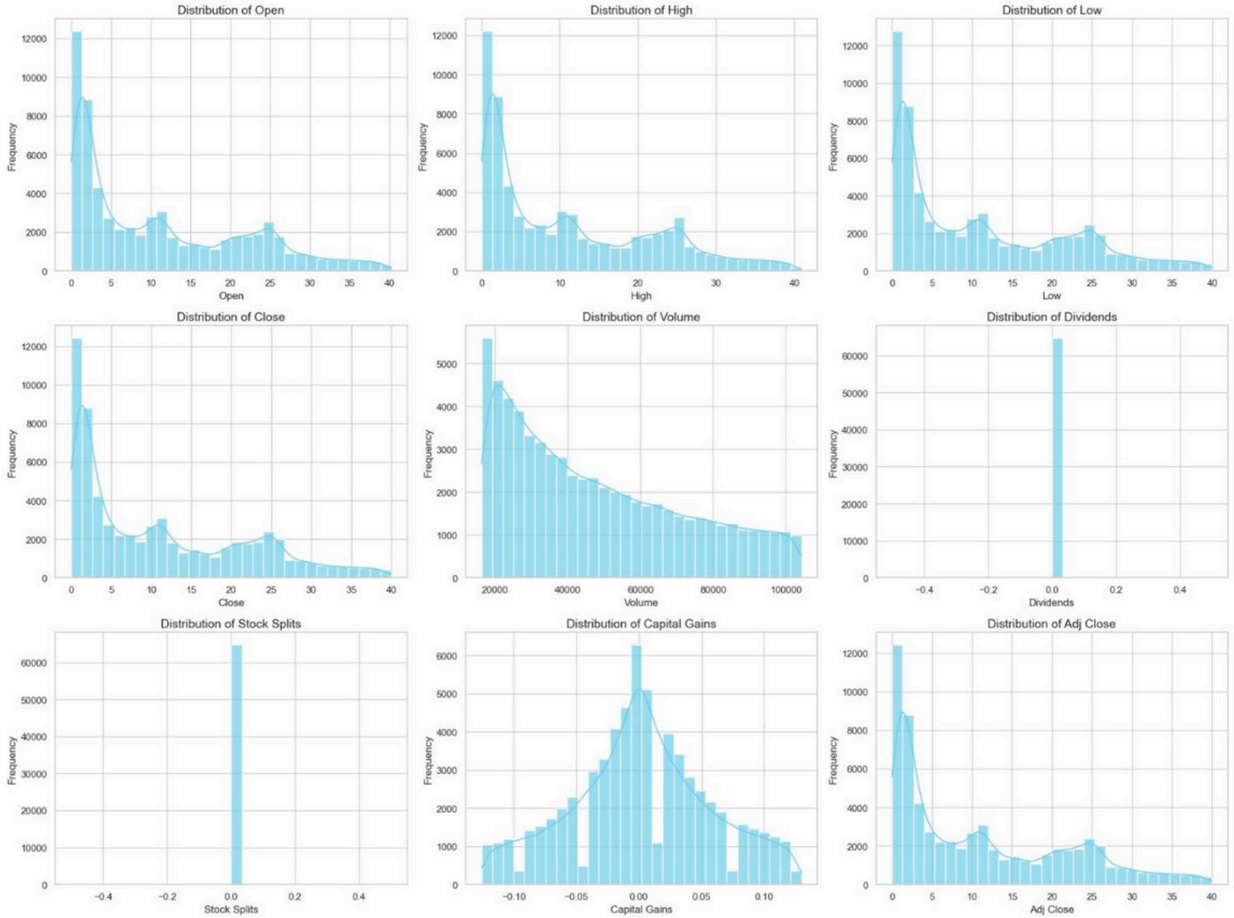


Key Observations:

- *Strong Positive Correlation:* Open, High, Low, Close, and Adj Close move together *(dark red)*
- *Negative Correlation with Volume:* Higher volume often correlates with lower stock prices. *(blue)*
- *Weak Correlation:* Dividends and Stock Splits are mostly independent of other variables.
- *Moderate Positive Correlation:* Capital Gains increase with better market performance. *(light blue)*

Data Insights

3. Distribution of each numeric column

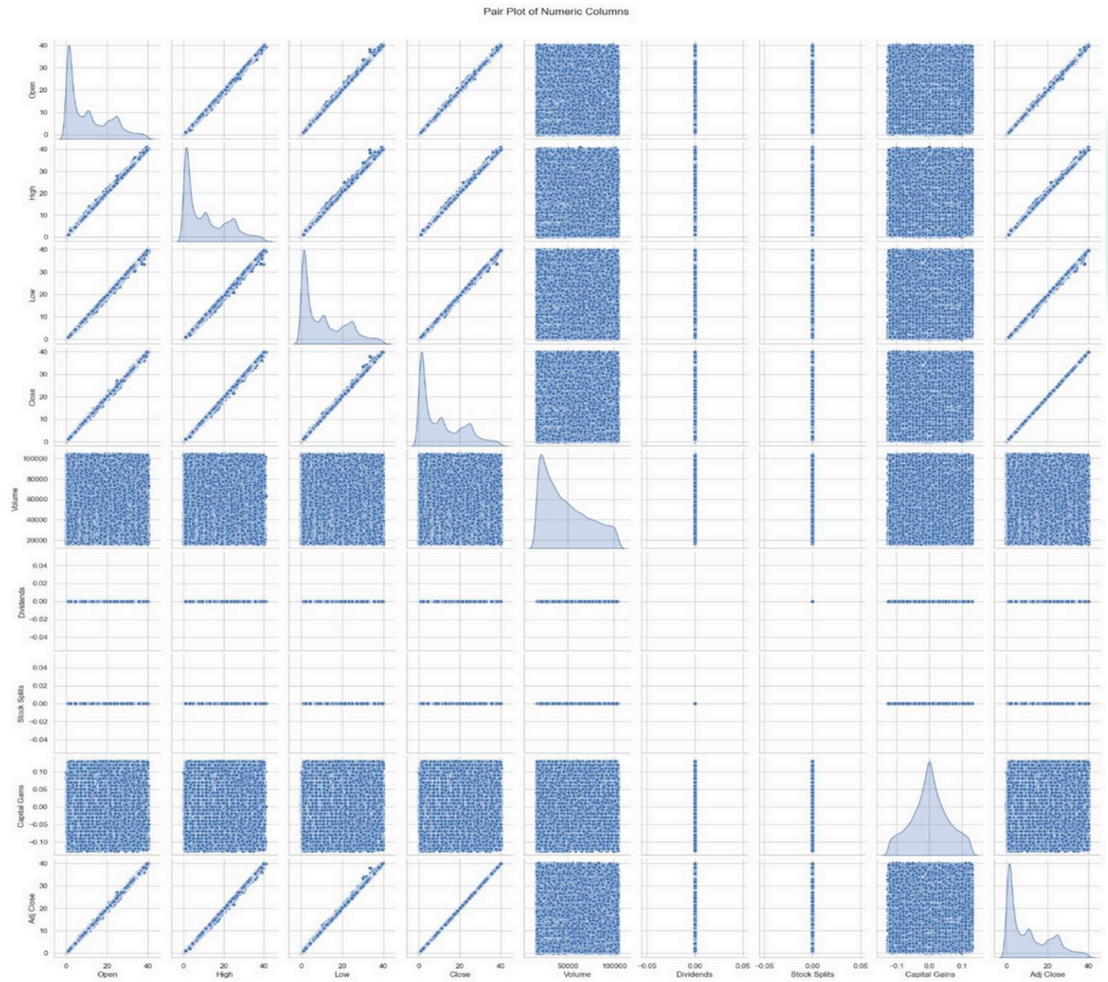


Key Observations:

- *Open, High, Low, Close:* Right-Skewed distributions, indicating normality and strong correlation among these indicators.
- *Volume:* Skewed right, with occasional spikes indicating high trading volumes during significant market events.
- *Dividends:* Concentrated around a narrow range, suggesting a stable dividend policy.
- *Stock Splits:* Few distinct peaks, indicating stock splits are rare but happen at specific intervals.
- *Capital Gains:* Bimodal distribution, reflecting different phases of the stock's performance.
- *Adj Close:* Similar to Open, High, Low, and Close, with a wide range of values showing price fluctuations.

Data Insights

4. Multivariate Relationships (Pair Plot)



Key Insights:

- *Pairwise Relationships:* Shows complex interactions between numeric variables.
- *Scatterplot Patterns:* Some relationships are linear, while others are nonlinear or irregular.
- *Correlation Structure:* Strong positive/negative correlations in some pairs; others are weak.
- *Variable Distributions:* Diagonal plots show normal, skewed, and multimodal distributions.
- *Outlier Detection:* Scatterplots help identify significant outliers.
- *Clusters and Groupings:* Some scatterplots reveal distinct data clusters.
- *Nonlinear Relationships:* Several relationships exhibit more complex dependencies.

Data Insights

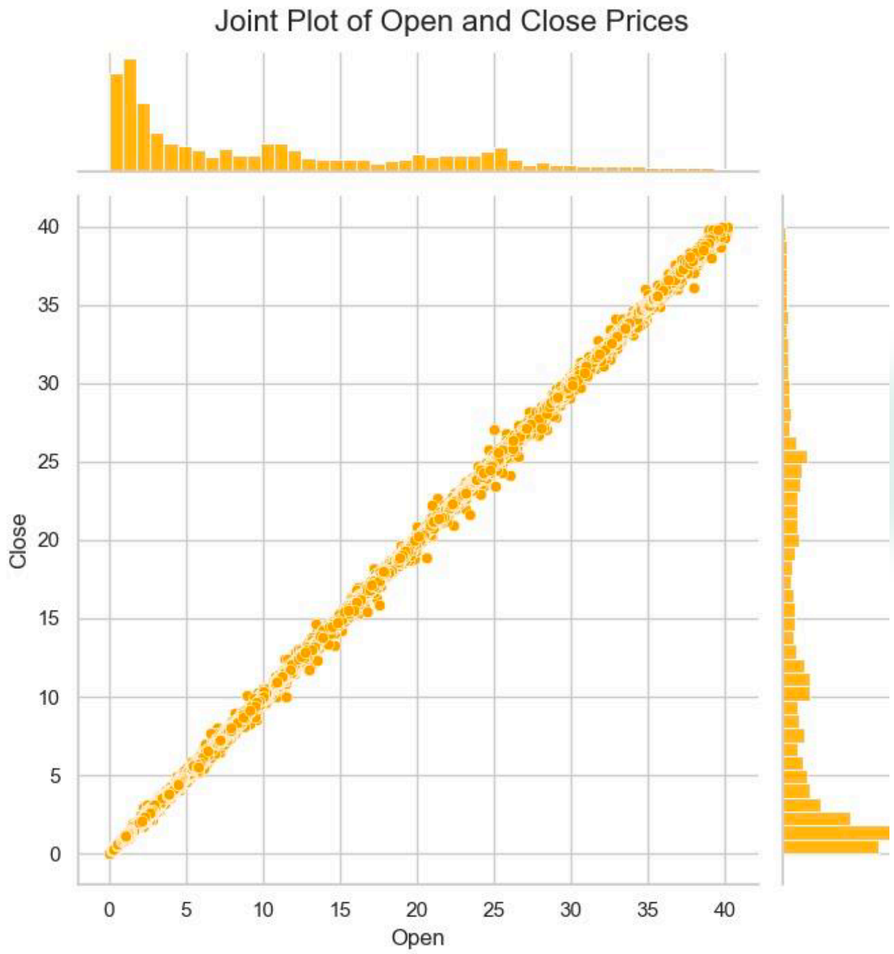
5. Density Plot (left) and CDF (right) of the Close Prices

- **Unimodal Distribution**
- **Single Peak:** Close prices cluster around a common value.
- **Right Skewed:** More frequent higher close prices.
- **Concentration:** Prices tightly grouped near the peak.
- **Range:** Significant variation from 0 to 40.
- **Outliers:** Low-density tails suggest potential extremes.

insights

- **S-Shaped Curve:** Indicates a smooth distribution.
- **Gradual Slope:** No abrupt price changes.
- **Range:** Close prices span 0 to 40.
- **Median & Quartiles:** Show central tendency and spread.
- **Probability:** Displays likelihood of a price being below a certain value

6. Joint Plot (Open vs Close Prices)

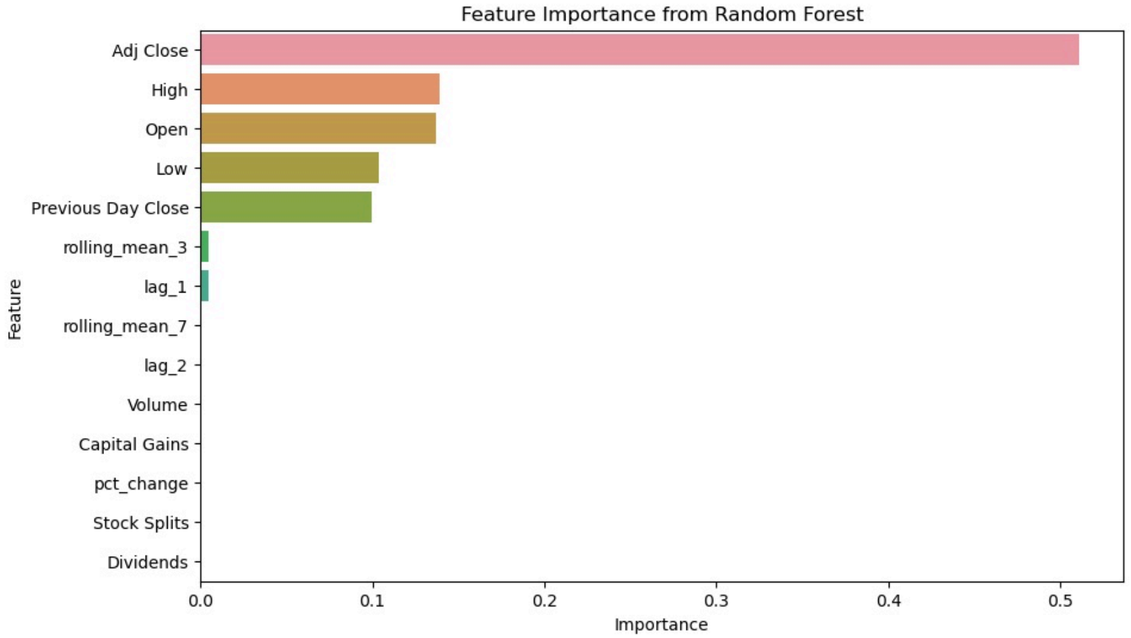


Key Observations:

- **Positive Correlation:** As Open price increases, Close price tends to increase, indicating a strong positive correlation.
- **Linearity:** The data forms a linear pattern, suggesting a predictable relationship between Open and Close prices.
- **Clustering:** Most data points are concentrated along the linear trend, indicating typical price movements.
- **Outliers:** A few outliers deviate from the trend, representing days with unusual price movements.
- **Price Range:** The plot spans 5 to 40, showing significant price fluctuations.

Feature Engineering

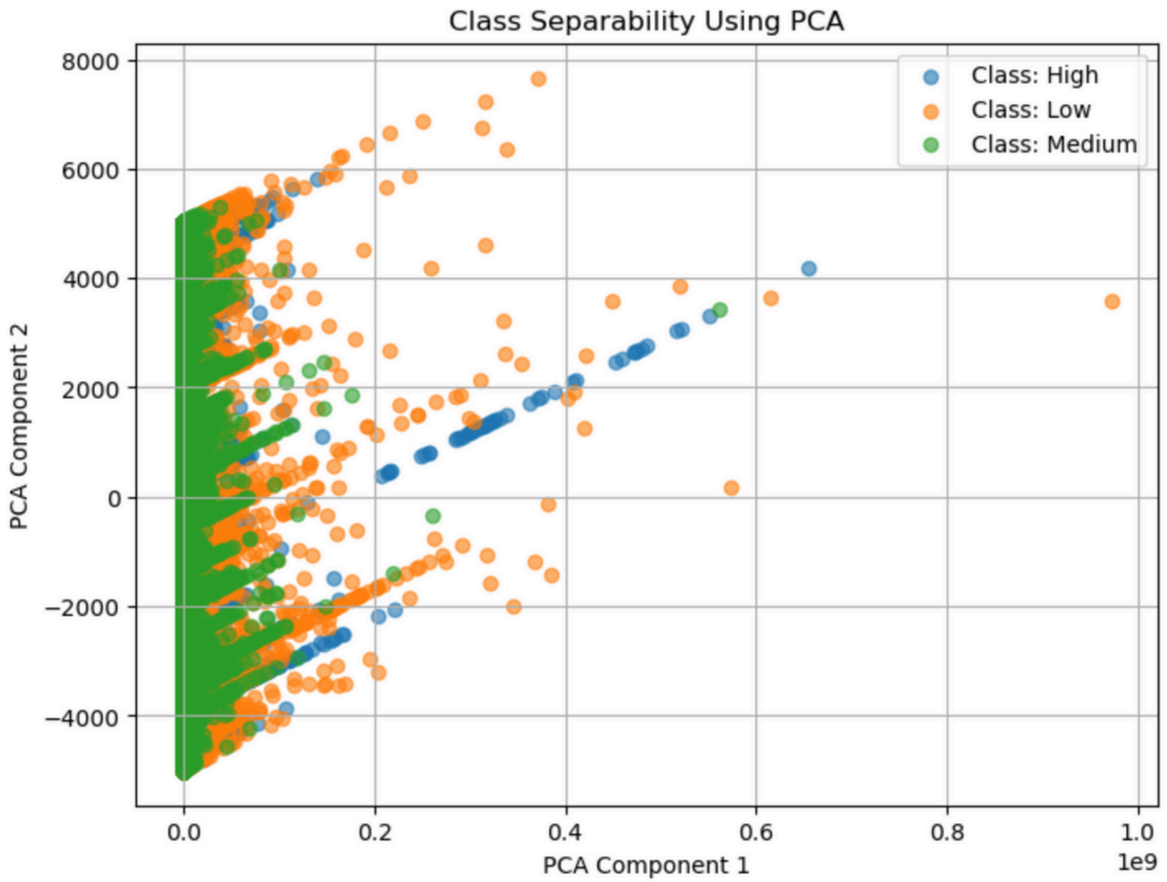
1. *Feature Importance From Random Forest*



- *Model Performance:*
- *Mean Squared Error (MSE): 2.1315 (initial data, we are working on it)*
- *Evaluates the accuracy of the Random Forest model on test data.*
- *Top Features:*
- *Adjusted Close: Most influential feature for stock prediction.*
- *High, Open, Low, Previous Day Close: Significant predictors affecting model performance.*
- *Rolling Averages & Lag Variables: Minor but valuable features for capturing trends.*
- *Actionable Insight: Focus on "Adj Close" for accurate portfolio recommendations. Utilize rolling averages and lag features to improve short-term stock predictions.*

Feature Engineering

2. Class Separability Using PCA



- *PCA Transformation: Used PCA to reduce high-dimensional data to 2 dimensions for better visualization.*
- *Binning the Target Variable: The target variable (Close) is continuous, so we had used pd.qcut to divide it into 3 bins: "Low", "Medium", "High"to simulate classes.*
- *Scatter Plot: PCA-reduced data is plotted in a scatter plot, color-coded by the binned target values to visualize class separability.*

Model Building & Evaluation

1. Random Forest Regressor

- **MSE:** The model achieved an MSE of 2.1316, showing better performance compared to others.
- **Effective Feature Handling:** If the model can handle feature relationships effectively, it could be the most promising model.

Tuning & Regularization

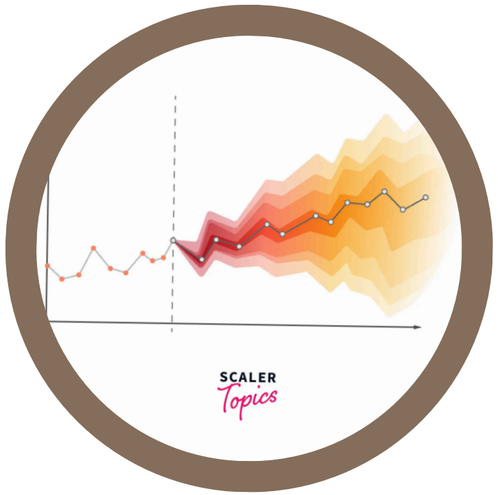
Tuning:

- **Hyperparameters:** Adjust n_estimators, max_depth, min_samples_split, and max_features.
- **Cross-Validation:** Use k-fold for better generalization.
- **Feature Importance:** Identify key features for improvement.

Regularization:

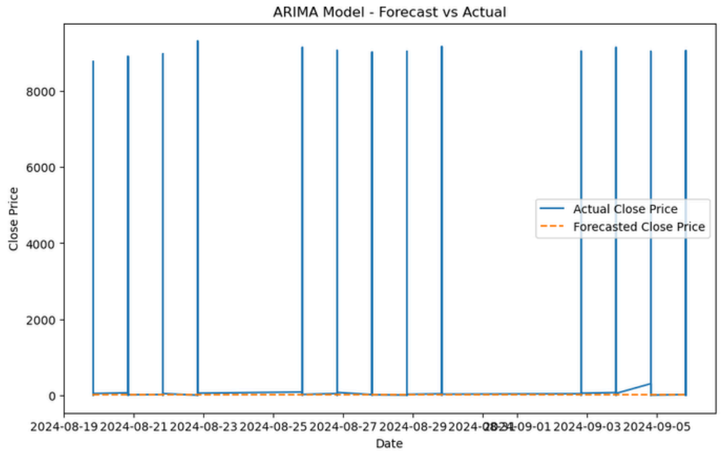
- **Feature Selection:** Remove irrelevant features.
- **Outlier Treatment:** Handle outliers.
- **Ensemble:** Combine with models like Gradient Boosting for better predictions.

Model Building & Evaluation



2. ARIMA

- MSE:19810.02
- Forecasting Issue: The MSE is provided, but the graph shows the forecast fails to capture the variability of actual prices, indicating underfitting.



Tuning & Regularization

Tuning:

1. Hyperparameter Tuning:
   - p: Number of lag observations.
   - d: Degree of differencing for stationarity.
   - q: Size of the moving average window.

1.1 Grid Search: Perform an exhaustive grid search to find the best model using AIC, BIC, or MSE as evaluation metrics.

Regularization:

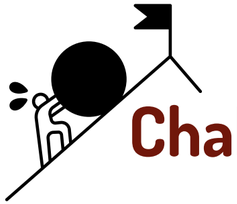
1. Regularization Techniques: Use AIC and BIC to prevent overfitting by penalizing excessive complexity.

Evaluation:

2. Residual Diagnostics: Check residuals for correlations and ensure stationarity using ADF test.

Libraries, Tools & Technologies

NumPy
SciPy
pandas
Google Cloud
matplotlib
scikit learn
python™
Google colab
Flask
web development, one drop at a time
SQL
Streamlit
PySpark
Power BI



Challenges of the Project

Model Optimization

The model's output is suboptimal, requiring *hyperparameter tuning* and *regularization techniques* for improvement.

Evaluation Accuracy

Correct evaluation methods need to be implemented to properly assess model performance.

Future Work

User Input Integration

Incorporating inputs based on user risk tolerance, budget, and investment goals (long-term/short-term).

Reinforcement Learning

Begin implementing reinforcement learning techniques to improve portfolio recommendations dynamically.

Team Everest

Thank You

For Your Attention

Any Queries?

We are open for discussion.

Access this ppt via: **https://bit.ly/bdm3035**