**Final Project: ICICI Bank Stock Closing Price Prediction**

**Step 1: Importing Required Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error

* **pandas** (pd) – Handles data manipulation.
* **numpy** (np) – Used for numerical operations.
* **matplotlib.pyplot** (plt) & **seaborn** (sns) – Used for data visualization.
* **sklearn.model\_selection.train\_test\_split** – Splits dataset into training and testing sets.
* **sklearn.preprocessing.StandardScaler** – Standardizes features.
* **sklearn.ensemble.RandomForestRegressor** – Implements the Random Forest regression model.
* **sklearn.metrics** – Provides performance metrics.

**Loading Dataset:**

file\_path = "icici\_dataset.csv"

df = pd.read\_csv(file\_path)

df.info()

* Reads the ICICI stock price dataset from a CSV file (df).
* df.info() displays dataset summary, including column names, data types, and missing values.

**Step 2: Cleaning and Preprocessing Data**

df.columns = df.columns.str.strip()

df["Date"] = pd.to\_datetime(df["Date"], errors="coerce")

df["Year"] = df["Date"].dt.year

df["Month"] = df["Date"].dt.month

df["Day"] = df["Date"].dt.day

* Removes leading and trailing spaces from column names to avoid errors during access.
* Converts the "Date" column to a datetime format, allowing date-related operations.
* Extracts year, month, and day from the "Date" column.

num\_cols = ["OPEN", "HIGH", "LOW", "PREV. CLOSE", "ltp", "close", "vwap", "VOLUME", "VALUE"]

for col in num\_cols:

df[col] = df[col].astype(str).str.replace(",", "").astype(float)

print(df.info())

print(df.isna().values.any())

* Defines a list of numeric columns.
* Converts values to strings, removes commas (if any), and converts them back to float for analysis.
* Displays dataset info again to check for updates.
* df.isna().values.any() checks if any missing values exist.

**Step 3: Data Visualization**

# Correlation Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(df[["Date", "OPEN", "HIGH", "LOW", "PREV. CLOSE", "close", "VOLUME", "VALUE"]].corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation Matrix")

* Visualizes correlations between numerical features using a heatmap.

# Set plot style

sns.set\_style("whitegrid")

* Sets a white grid background.

plt.figure(figsize=(12, 6))

for i, col in enumerate(num\_cols, 1):

plt.subplot(3, 3, i)

sns.boxplot(data=df, x=col)

plt.title(f"Boxplot of {col}")

plt.tight\_layout()

* Creates boxplots for numerical columns to detect outliers.

**Step 4: Time Series Visualization**

# Create a figure and a 3x1 grid of subplots

fig, axs = plt.subplots(3, 1, figsize=(8, 12))

# First subplot for OPEN and CLOSE prices

axs[0].plot(df['Date'], df["OPEN"], label='OPEN')

axs[0].plot(df['Date'], df["close"], label='CLOSE')

axs[0].set\_title('Stock Opening and Closing Prices')

axs[0].set\_xlabel('Date')

axs[0].set\_ylabel('Price in Rupees')

axs[0].legend()

# Second subplot for HIGH and LOW prices

axs[1].plot(df['Date'], df["HIGH"], label='HIGH')

axs[1].plot(df['Date'], df["LOW"], label='LOW')

axs[1].set\_title('Stock High and Low Prices')

axs[1].set\_xlabel('Date')

axs[1].set\_ylabel('Price in Rupees')

axs[1].legend()

# Third subplot for VOLUME and VALUE

axs[2].plot(df['Date'], df["VOLUME"], label='VOLUME')

axs[2].plot(df['Date'], df["VALUE"], label='VALUE')

axs[2].set\_title('Stock Volume and Value')

axs[2].set\_xlabel('Date')

axs[2].set\_ylabel('Value')

axs[2].legend()

plt.tight\_layout()

* Creates three subplots:
  + OPEN & CLOSE Prices
  + HIGH & LOW Prices
  + VOLUME & VALUE
* Helps in understanding trends in stock data.

**Step 5: Unsupervised Learning: PCA**

# Select features for PCA  
features = ["HIGH", "LOW", "VOLUME", "VALUE"]  
X = df[features]

pca = PCA(n\_components=2) # Reduce dimensions to 2 for plotting

X\_pca = pca.fit\_transform(X\_scaled) # Apply PCA

df["PCA1"] = X\_pca[:, 0] # Store first principal component

df["PCA2"] = X\_pca[:, 1] # Store second principal component

* PCA reduces high-dimensional data, making it easier to visualize

# Visualizing PCA components  
plt.figure(figsize=(8, 6))  
sns.scatterplot(data=df, x="PCA1", y="PCA2")  
plt.title("PCA Visualization of Stock Data")  
plt.xlabel("Principal Component 1")  
plt.ylabel("Principal Component 2")  
plt.show()

* This scatterplot shows how stock data is grouped into different clusters.

**Step 6: Supervised Learning: Random Forest Regression**

# Select features and target for regression  
features = ["Year", "Month", "Day", "PCA1", "PCA2"]  
target = "close"  
  
X = df[features]  
y = df[target]

* The model will predict the close stock price using selected features.
* The model achieved an R² score of 0.85.
* Then I will try with original features.

**Step 7: Feature Selection Without PCA & Preprocessing**

# Select features and target

features = ["Year", "Month", "Day", "HIGH", "LOW", "VOLUME"]

target = "close"

X = df[features]

y = df[target]

* Defines feature variables (X) and target variable (y) for modeling.

# Standardize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* Ensures all features have the same scale.

# Split dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

* Splits data into training (80%) and testing (20%).

**Step 8: Model Training**

# Train Random Forest Model

model = RandomForestRegressor(n\_estimators=100, random\_state=42)

model.fit(X\_train, y\_train)

* Trains a Random Forest Regressor with 100 trees.

**Step 9: Making Predictions & Evaluating Performance**

# Predictions

y\_pred = model.predict(X\_test)

# Calculate accuracy metrics

r2 = r2\_score(y\_test, y\_pred)

mae = mean\_absolute\_error(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(r2)

* Predicts stock prices y\_pred for test data.
* Evaluates model performance.

**Visualizing Predictions**

plt.figure(figsize=(10, 6))

plt.scatter(range(len(y\_train)), y\_train, edgecolor='w', label='Actual Price')

plt.plot(range(len(y\_train)), model.predict(X\_train), color='r', label='Predicted Price')

plt.title('Linear Regression | Price vs Time')

plt.xlabel('Time Index')

plt.ylabel('Stock Price')

plt.legend()

* Plots actual vs predicted stock prices for training data.

**Step 10: Saving the Model**

import joblib

# Save the trained model

joblib.dump(model, "stock\_price\_model.pkl")

# Save the scaler

joblib.dump(scaler, "scaler.pkl")

* Saves the trained model & scaler into pickle file.

**Step 11: Creating the web application**

1. **Importing Required Libraries**

from flask import Flask, render\_template, request

import pandas as pd

import numpy as np

import joblib

* + Flask Creates the web application.

**2. Initializing Flask App**

app = Flask(\_\_name\_\_)

* Initializes a Flask web application.

**3. Loading the Pre-Trained Model and Scaler**

model = joblib.load("stock\_price\_model.pkl")

scaler = joblib.load("scaler.pkl")

* Loads the trained Random Forest model
* Loads the StandardScaler.

**4. Defining the Home Route (/)**

@app.route("/", methods=["GET", "POST"])

def index():

* Defines the homepage route (/).
* Allows both GET and POST requests:
  + GET for displays the webpage.
  + POST for receives user input and makes predictions.

**5. Handling User Input from HTML Form**

if request.method == "POST":

try:

# Get user input from form

date = request.form["date"]

high = float(request.form["high"])

low = float(request.form["low"])

volume = float(request.form["volume"])

**6. Extracting Date Features**

# Convert date

date\_obj = pd.to\_datetime(date)

year, month, day = date\_obj.year, date\_obj.month, date\_obj.day

* Converts user input date to datetime format.
* Extracts year, month, and day as features.

**7. Preparing Features for Prediction**

# Prepare input for model

features = np.array([[year, month, day, high, low, volume]])

features\_scaled = scaler.transform(features)

**8. Making Prediction**

# Predict closing price

predicted\_price = model.predict(features\_scaled)[0]

**9. Returning the Prediction to the Web Page**

return render\_template("index.html", prediction=predicted\_price)

**10. Handling Errors**

except Exception as e:

return render\_template("index.html", error=str(e))

**11. Handling GET Requests**

return render\_template("index.html", prediction=None, error=None)

**12. Running the Flask App**

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

* Runs the Flask web server.