FORECASTING DIRECTIONAL MOVEMENTS USING LSTM

SOFTWARE ENGINEERING PROJECT

PRESENTED BY GROUP-11

MEET OUR TEAM

B. Syam Sai Krishna- AM.EN.U4CSE21016

Project Lead

A.S.V Prasasya- AM.EN.U4CSE21067

Business Analyst

Purnima R - AM.EN.U4CSE21046

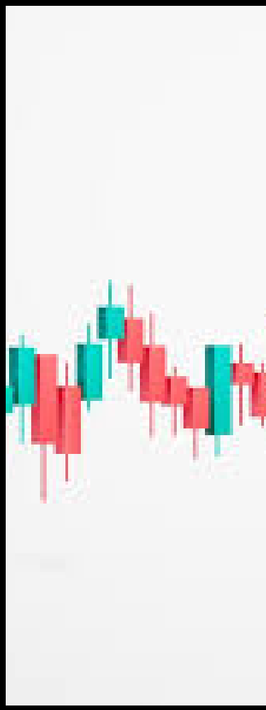
Scrum Master

INTRODUCTION

- The forex (foreign exchange) market involves the trading of currencies globally. It is the largest and most liquid financial market in the world, with a daily turnover of trillions of dollars. Having accurate forecasts can provide traders with a competitive advantage in the market. LSTM networks contain memory cells that can store information over extended periods, allowing them to retain important context.

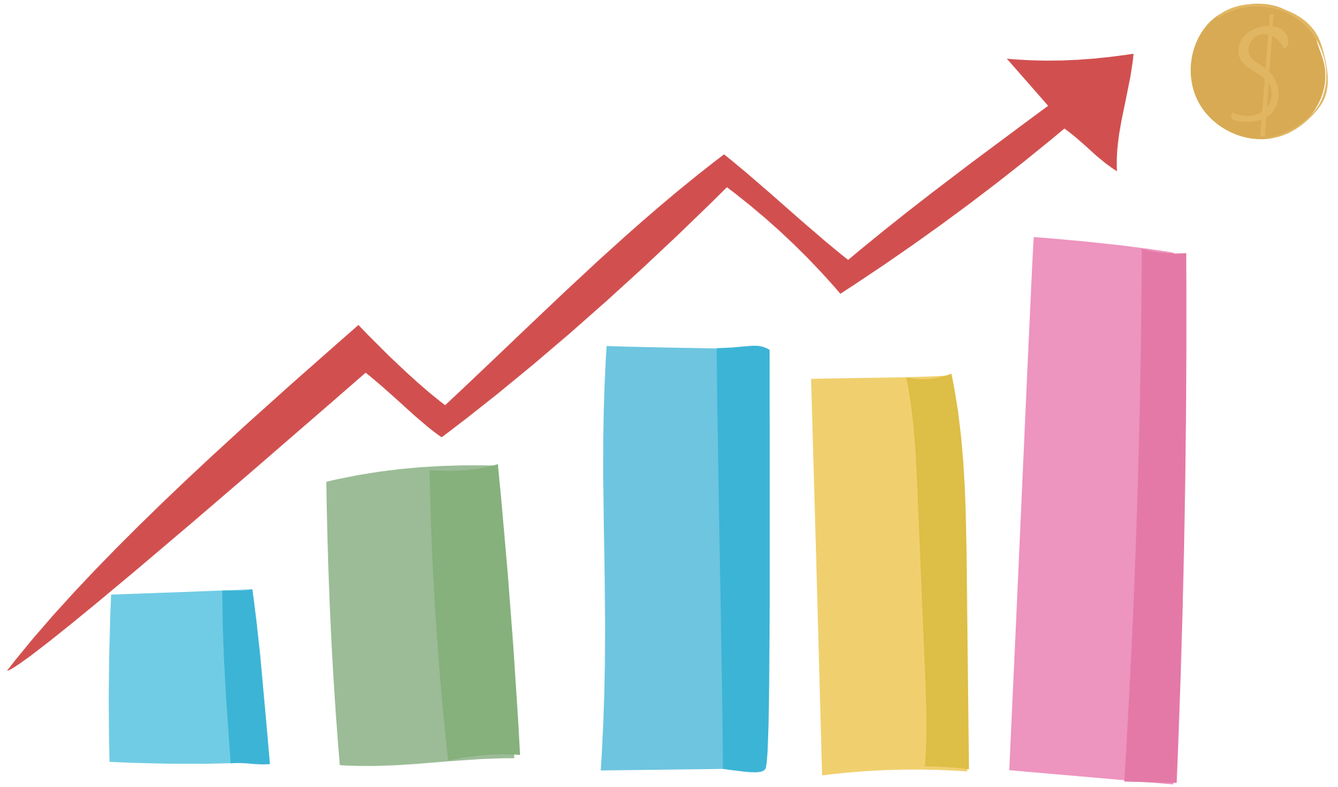
PROBLEM **STATEMENT**

- Forecasting directional movement in Forex (foreign exchange) data is a challenging task due to the inherent complexity and volatility of the currency markets.
- Traders and investors need accurate predictions of whether a currency pair will move up or down in order to make informed decisions and maximize profits. Traditional time series forecasting techniques often struggle to capture the nonlinear and dynamic nature of Forex data, making it difficult to achieve reliable predictions.



INNOVATION

- USE OF DIFFERENT TECHNICAL INDICATORS
- VISUALIZATION FOR ANALYSIS
- IMPORTANCE OF TECHNICAL INDICATORS IN RESEARCH



MODEL- DESIGN PHASE

01 USE-CASE

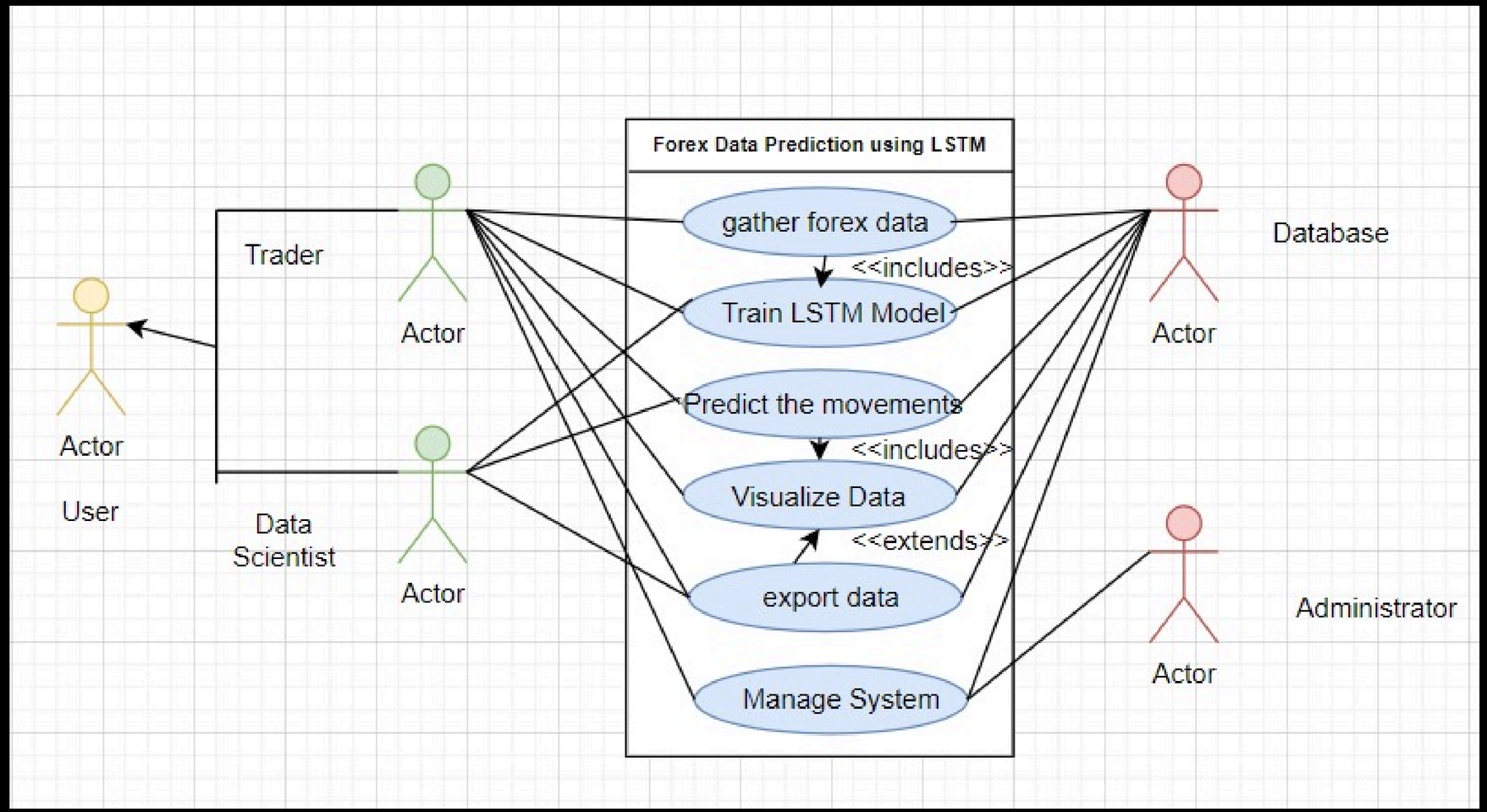
02 ACTIVITY

03 CLASS

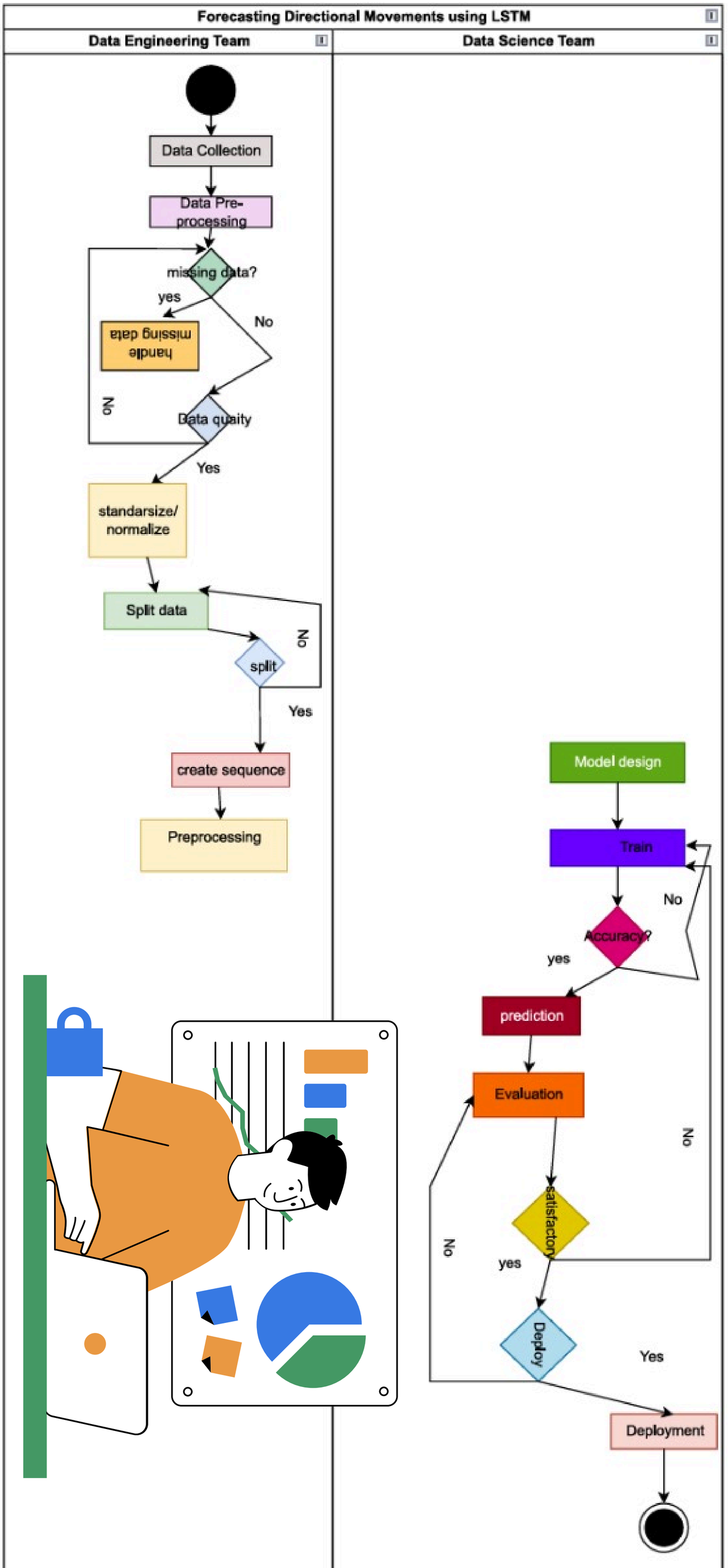
04 DATA FLOW

05 SEQUENCE

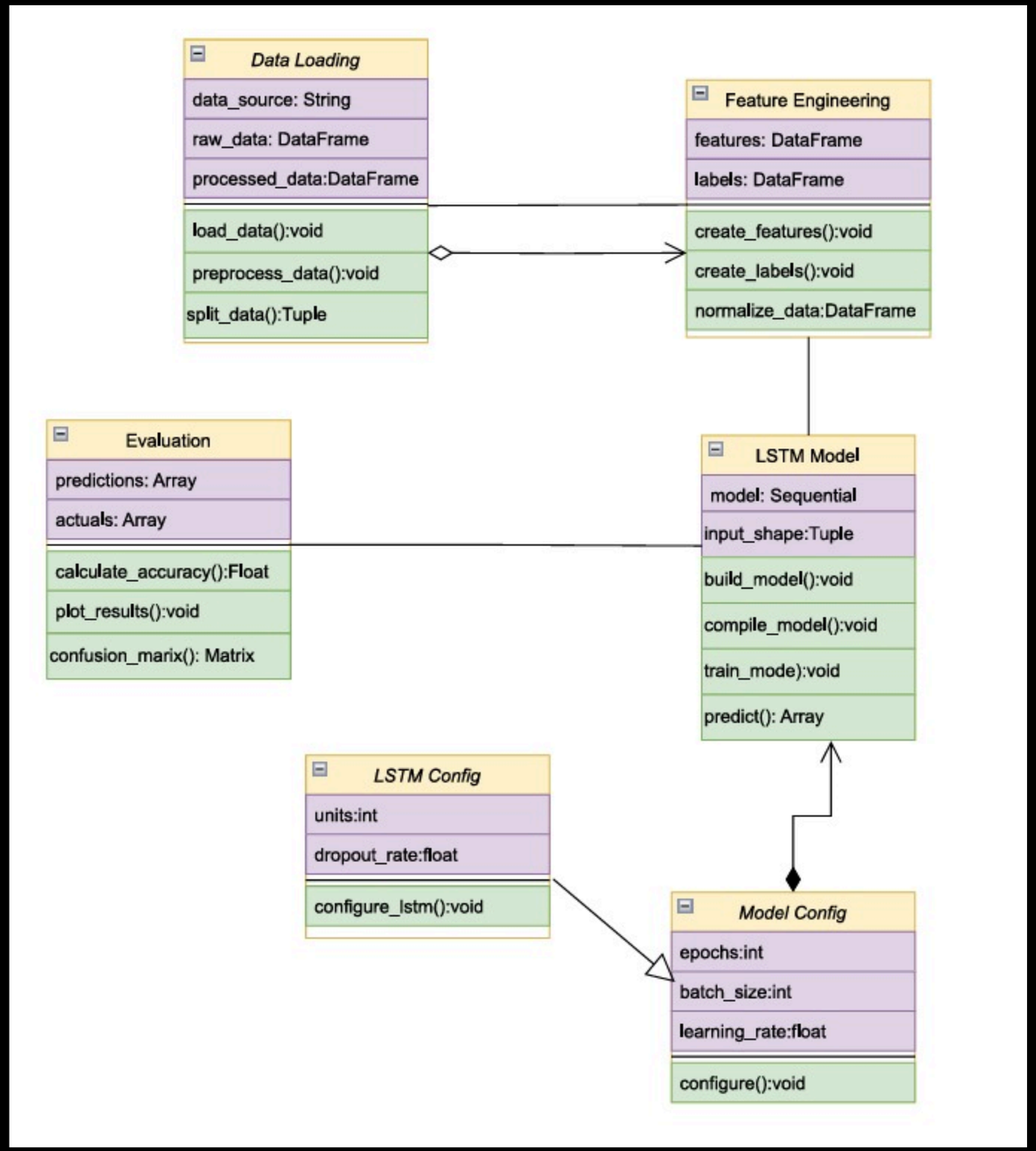
USE-CASE

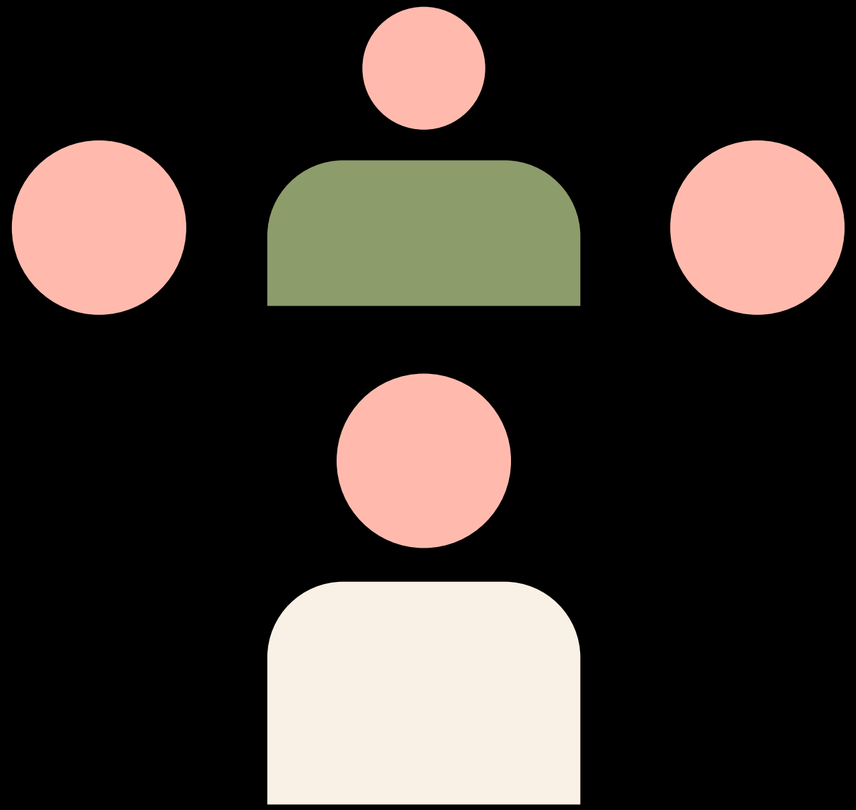

Forex Data Prediction using LSTM
gather forex data
<<includes>>
Train LSTM Model
Predict the movements
<<includes>>
Visualize Data
<<extends>>
export data
Manage System
Trader
Actor
Actor
User
Data
Scientist
Actor
Actor
Database
Actor
Administrator

ACTIVITY

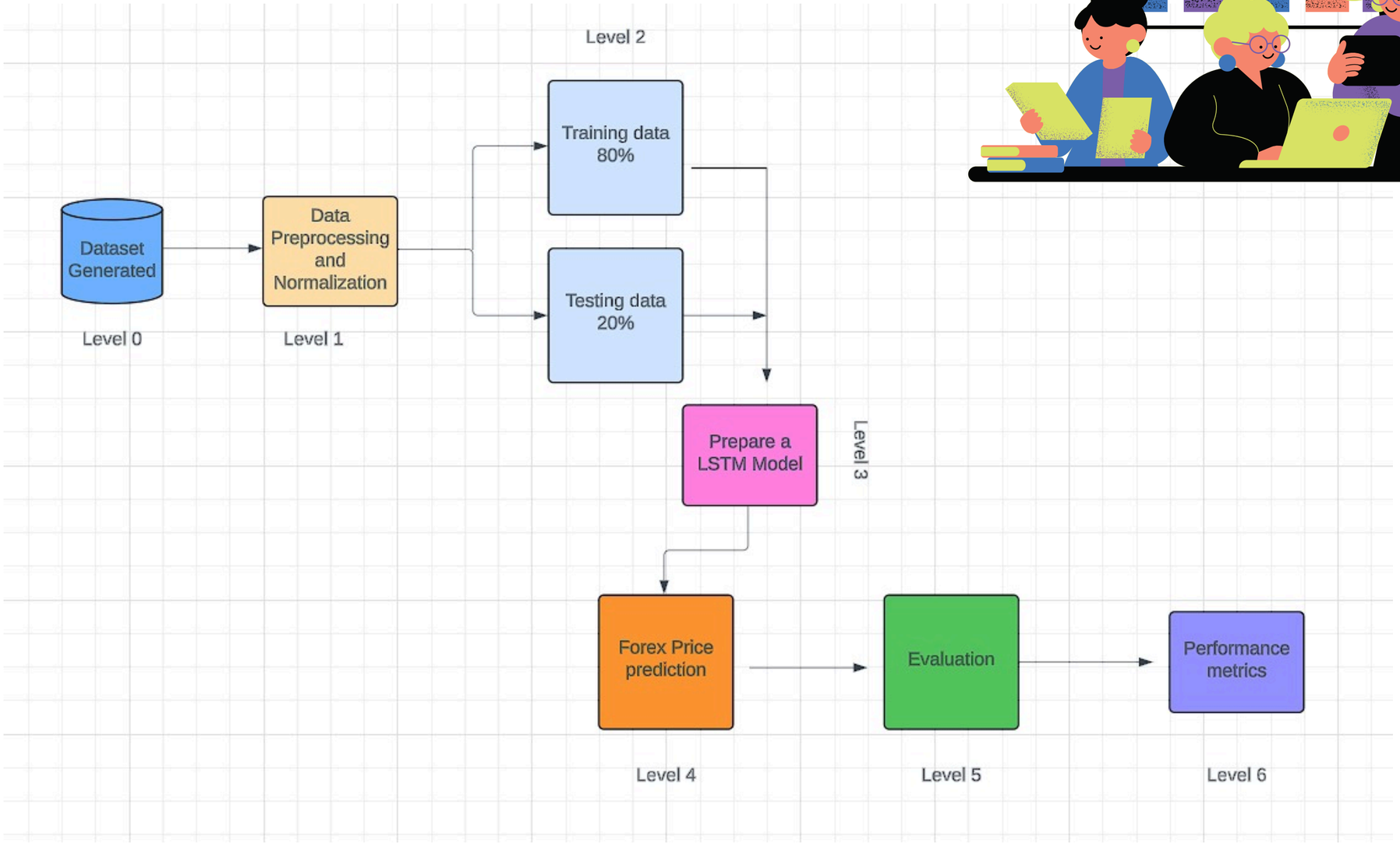

Forecasting Directional Movements using LSTM
Data Engineering Team
Data Science Team
Data Collection
Data Pre-processing
missing data?
yes
No
handle missing data
No
Data quaity
Yes
standarsize/ normalize
Split data
No
split
Yes
create sequence
Preprocessing
Model design
Train
No
Accuracy?
yes
prediction
Evaluation
No
satisfactory
No
yes
Deploy
Yes
Deployment

CLASS DIAGRAM

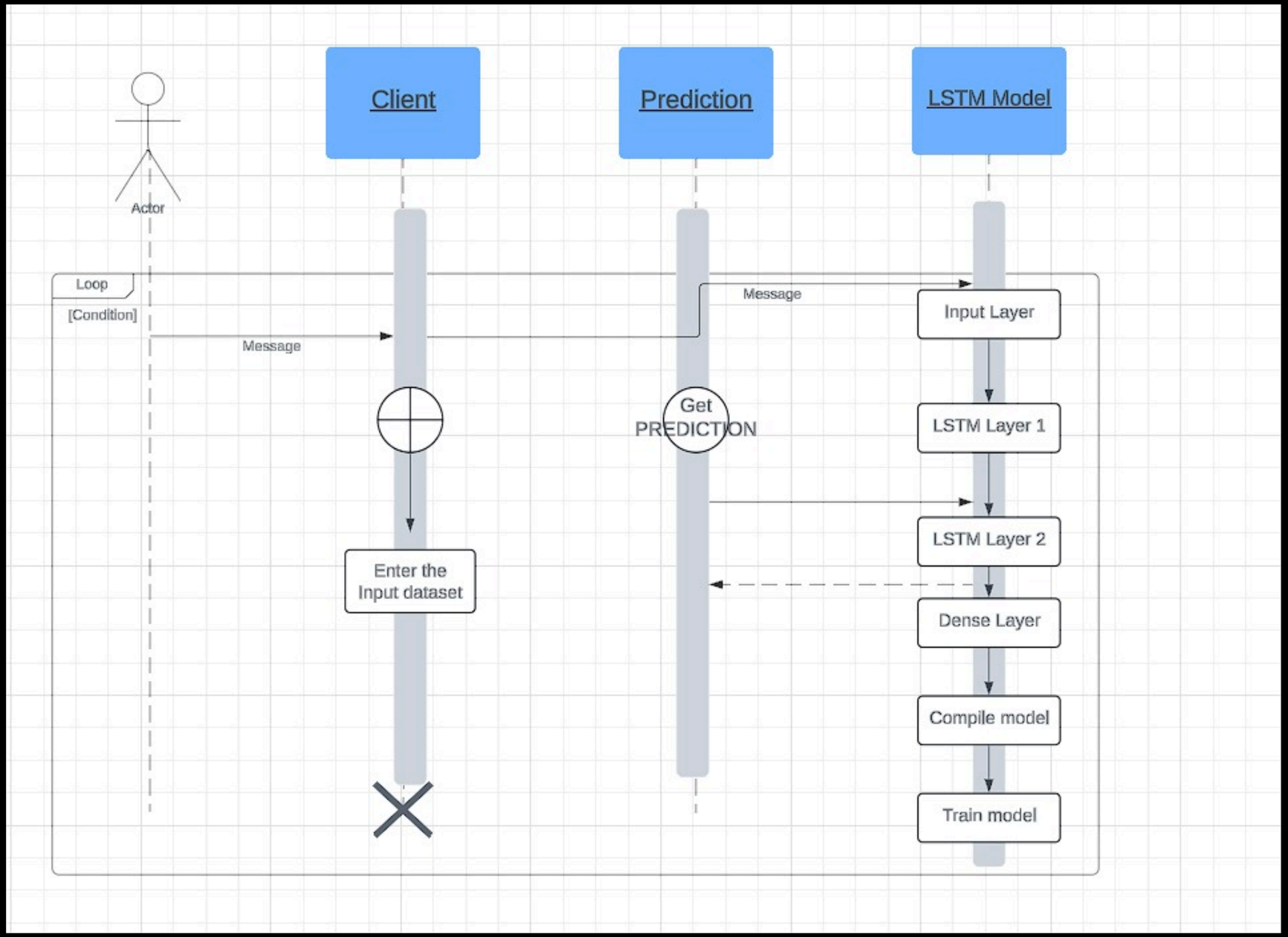




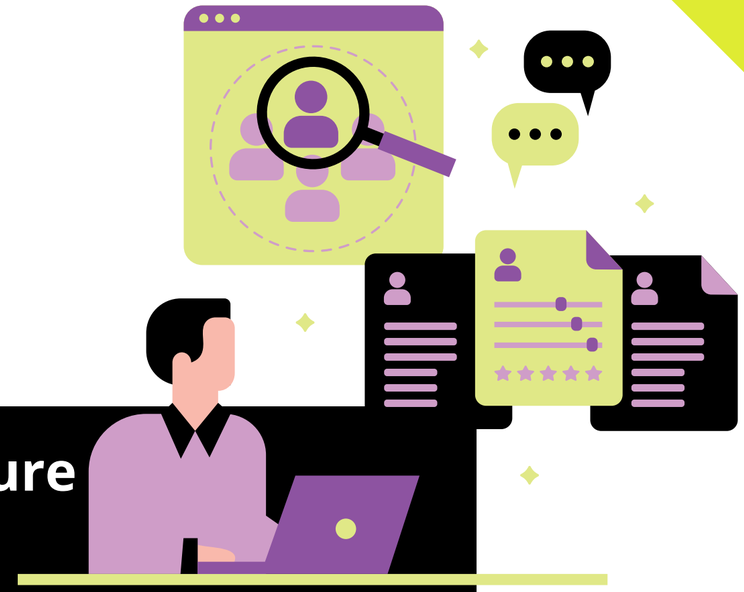
DATA FLOW DIAGRAM



SEQUENCE DIAGRAM



CODING



Data pre-processing

Train-Test: 80:20
Synthetic and processed dataset
Samples: 5000

Model Architecture

LSTM
Sequence_Length: 10
units: 50
learning rate: 0.0001
batch_size: 32

Training and validation

100 epochs
NVIDIA GPU

Evaluation Metrics

Various plots
loss

IMPORT ALL
NECESSARY LIBRARIES

```
import os
import sys
import json
import time
import pickle
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

DATASET

```
train_data=pd.read_csv("C:\\Users\\91630\\Anaconda files\\New_Forex\\train_data.csv")
test_data=pd.read_csv("C:\\Users\\91630\\Anaconda files\\New_Forex\\test_data.csv")
```

```
train_data.head()
```

	Open	High	Low	Close	Volume	SMA	EMA	RSI	MACD	Upper_BB	Middle_BB	Lower_BB	%K	%D	ATR	Outcome

HANDLING **MISSING VALUES**

```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# ... (load your data into train_data and test_data)

# Explicitly Exclude Specific Columns
columns_to_normalize = train_data.drop(columns=[ 'Outcome', 'Upper_BB', 'Lower_BB', 'STD', 'Upper', 'Lower']).columns

scaler = MinMaxScaler()
scaler.fit(train_data[columns_to_normalize])  # Fit on train_data

# Transform Data
train_data_normalized = train_data.copy()
test_data_normalized = test_data.copy()

train_data_normalized[columns_to_normalize] = scaler.transform(train_data[columns_to_normalize])
test_data_normalized[columns_to_normalize] = scaler.transform(test_data[columns_to_normalize])
```

```python
# Drop the first 19 rows with missing values from the training data
train_data_normalized = train_data_normalized.iloc[19:].copy()
```

```python
print(train_data_normalized.isnull().sum())
print(test_data_normalized.isnull().sum())
```

```
Open          0
High          0
Low           0
Close         0
Volume        0
SMA           0
EMA           0
RSI           0
MACD          0
```

TRAIN-TEST SPLIT 80:20

```python
print(train_data.shape)
print(test_data.shape)
```

```
(3984, 16)
(997, 16)
```

CREATE SEQUENCES

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense,Input
from keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
tf.debugging.enable_check_numerics()
from keras.optimizers import Adam

# ... (load your data into train_data and test_data)

# 1. Data Preprocessing (Already Done - You Have Normalized Data)

# 2. Prepare Sequences for LSTM
def create_sequences(data, sequence_length):
    Xs, ys = [], []
    for i in range(len(data) - sequence_length):
        Xs.append(data.iloc[i:(i + sequence_length)].values)
        ys.append(data.iloc[i + sequence_length]['Outcome'])  # Predict 'Outcome'
    return np.array(Xs), np.array(ys)

sequence_length = 10  # Adjust this based on your time series characteristics
X_train, y_train = create_sequences(train_data_normalized, sequence_length)
X_test, y_test = create_sequences(test_data_normalized, sequence_length)
```

LET'S BUILD A **LSTM MODEL**

MEAN IMPUTATION

```
# Check for NaNs in sequences
print(np.isnan(X_train).any())
print(np.isnan(X_test).any())

# Impute all NaNs in train_data_normalized and test_data_normalized
for col in train_data_normalized.columns:
    train_data_normalized[col] = train_data_normalized[col].fillna(train_data_normalized[col].mean())

for col in test_data_normalized.columns:
    test_data_normalized[col] = test_data_normalized[col].fillna(test_data_normalized[col].mean())

# ... (rest of your code)
```

```
# Create sequences
X_train, y_train = create_sequences(train_data_normalized, sequence_length)
X_test, y_test = create_sequences(test_data_normalized, sequence_length)
# ...your code for checking and imputing missing values...
# Re-create sequences after imputation
X_train, y_train = create_sequences(train_data_normalized, sequence_length)
X_test, y_test = create_sequences(test_data_normalized, sequence_length)
```

SEQUENTIAL **MODEL TRAINING**

```python
n_features = train_data_normalized.shape[1]  # Number of features

model = Sequential()
model.add(Input(shape=(sequence_length, n_features)))
model.add(LSTM(units=50, activation='relu', return_sequences=True))
model.add(LSTM(units=50, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))  # Output layer for binary classification
model.compile(optimizer=Adam(learning_rate=0.0001, clipvalue=1.0), loss='binary_crossentropy', metrics=['accuracy'])
# 4. Train Model with Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, callbacks=[early_stopping])
```

67%
Profit 2021

incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam.

PREDICT AND EVALUATE

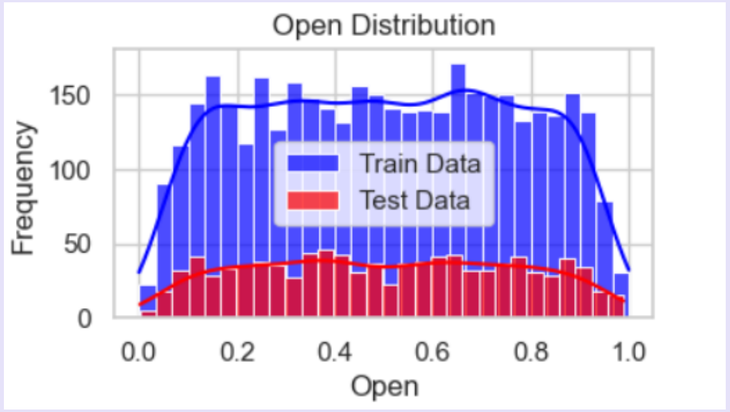
```python
# Print training accuracy
train_accuracy = history.history['accuracy'][-1]  # Get final training accuracy from the history object
print(f'Training Accuracy: {train_accuracy:.4f}')

# 5. Make Predictions and Evaluate (Modified)
y_pred_prob = model.predict(X_test)  # Get probabilities
y_pred = (y_pred_prob > 0.5).astype(int)  # Convert probabilities to binary labels (0 or 1)

# ... (inverse transform your predictions and labels if needed)

# Calculate and print test accuracy
test_accuracy = accuracy_score(y_test, y_pred)
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
Epoch 91/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.6755 - loss: 0.6038 - val_accuracy: 0.6497 - val_loss: 0.6310
Epoch 92/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.6721 - loss: 0.6120 - val_accuracy: 0.6434 - val_loss: 0.6321
Epoch 93/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.6554 - loss: 0.6090 - val_accuracy: 0.6497 - val_loss: 0.6403
Epoch 94/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.6781 - loss: 0.6020 - val_accuracy: 0.6459 - val_loss: 0.6341
Epoch 95/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.6719 - loss: 0.6051 - val_accuracy: 0.6371 - val_loss: 0.6493
Epoch 96/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.6510 - loss: 0.6253 - val_accuracy: 0.6459 - val_loss: 0.6346
Epoch 97/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.6676 - loss: 0.6035 - val_accuracy: 0.6472 - val_loss: 0.6353
Epoch 98/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 40ms/step - accuracy: 0.6664 - loss: 0.6126 - val_accuracy: 0.6282 - val_loss: 0.6500
Epoch 99/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 39ms/step - accuracy: 0.6655 - loss: 0.5986 - val_accuracy: 0.6561 - val_loss: 0.6379
Epoch 100/100
99/99 ━━━━━━━━━━━━━━━━━━━━ 4s 38ms/step - accuracy: 0.6874 - loss: 0.5980 - val_accuracy: 0.6536 - val_loss: 0.6286
```

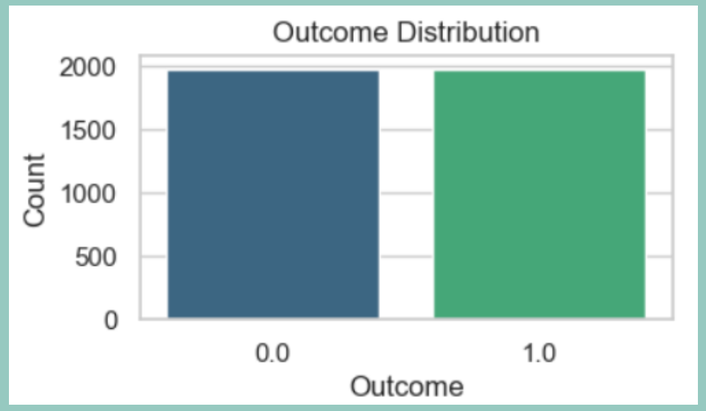
EPOCHS

VISUALIZATIONS

HISTOGRAMS

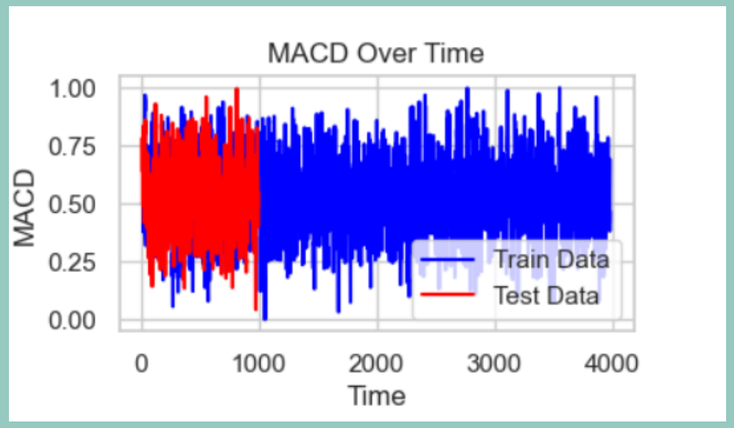


BAR CHART

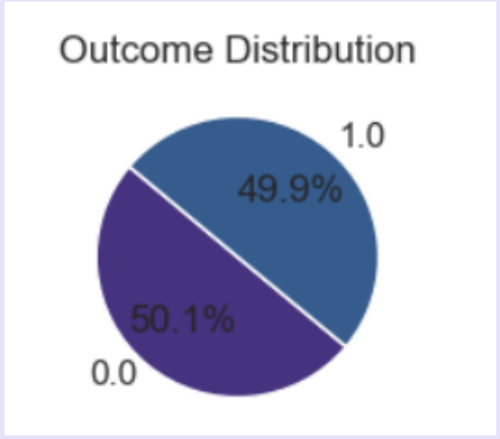


BOX PLOT

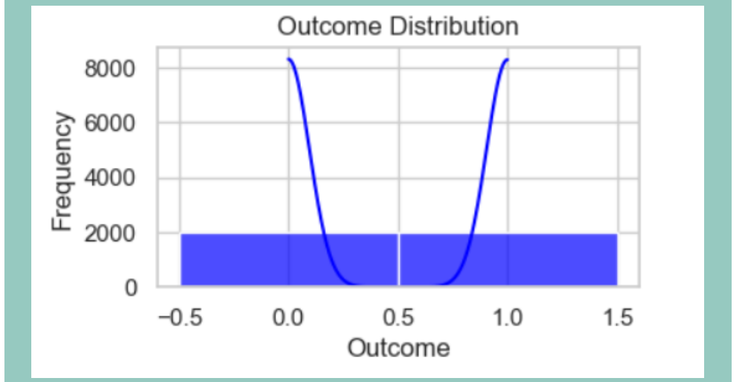
LINE CHART

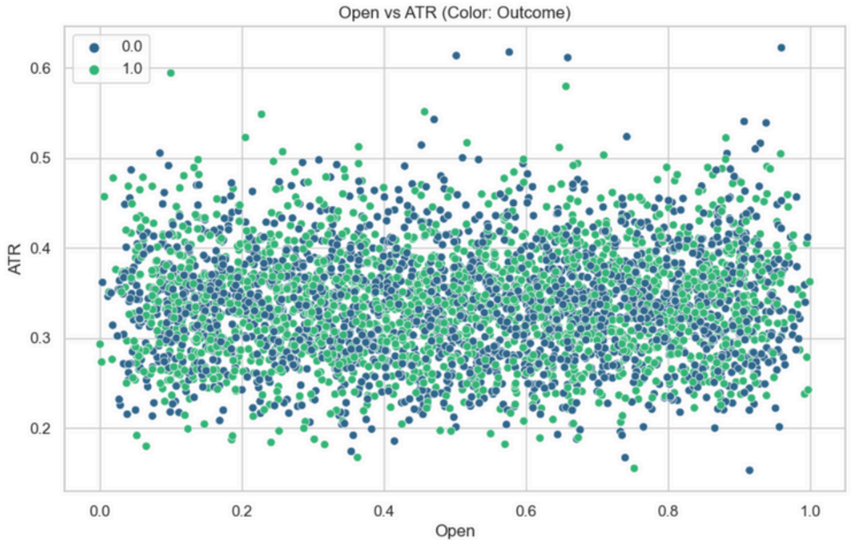


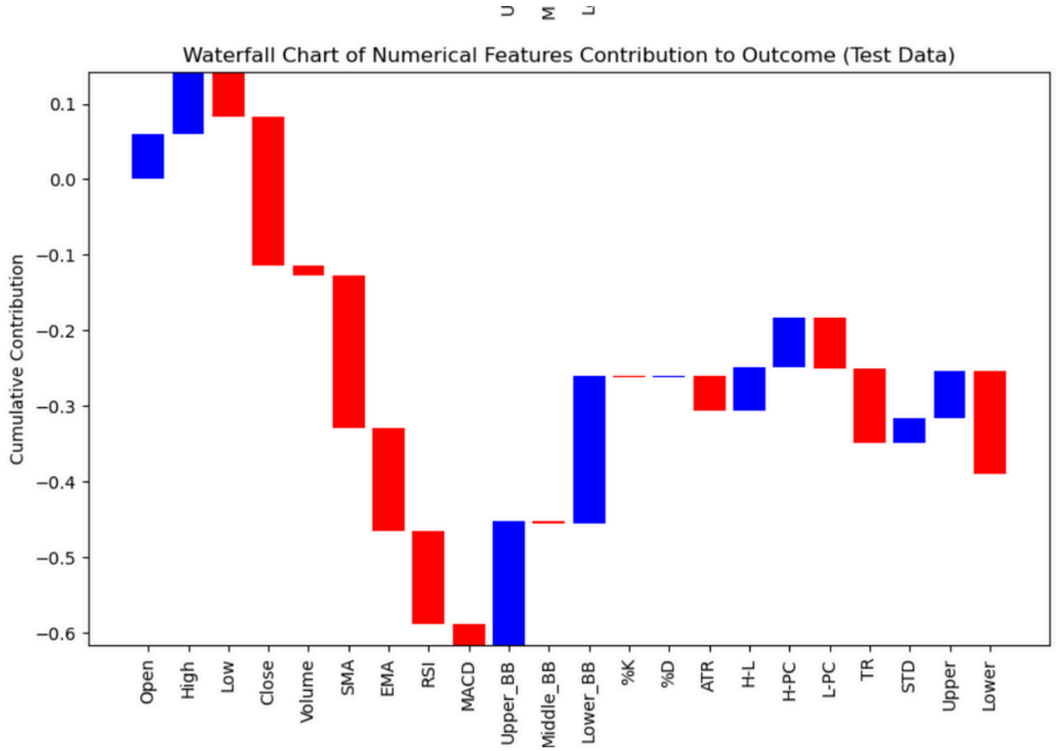
PIE CHART

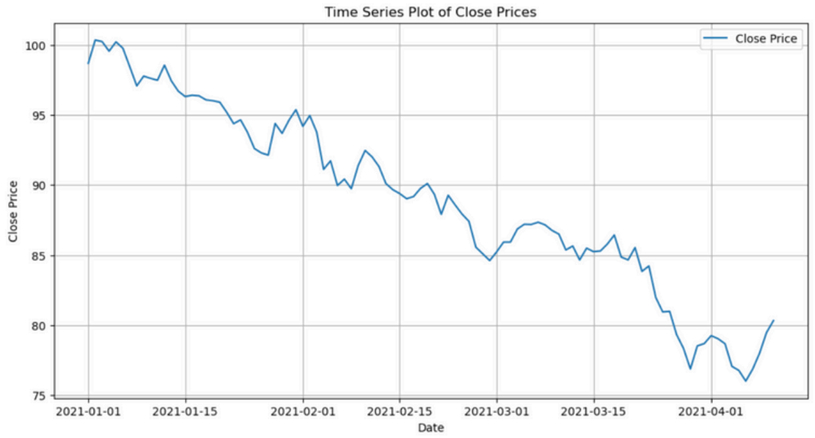


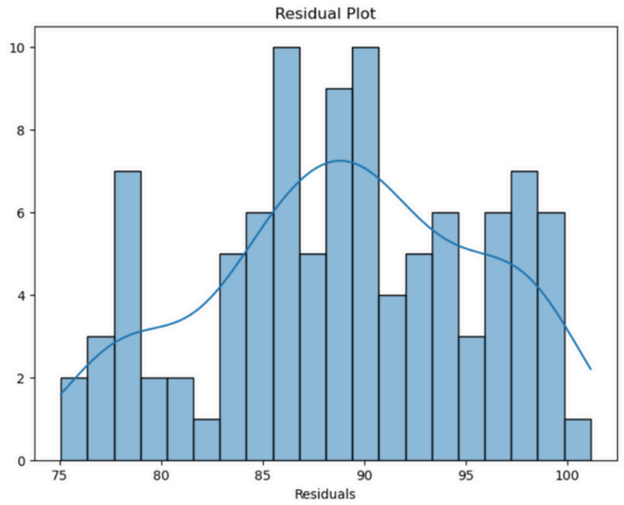
AREA CHART

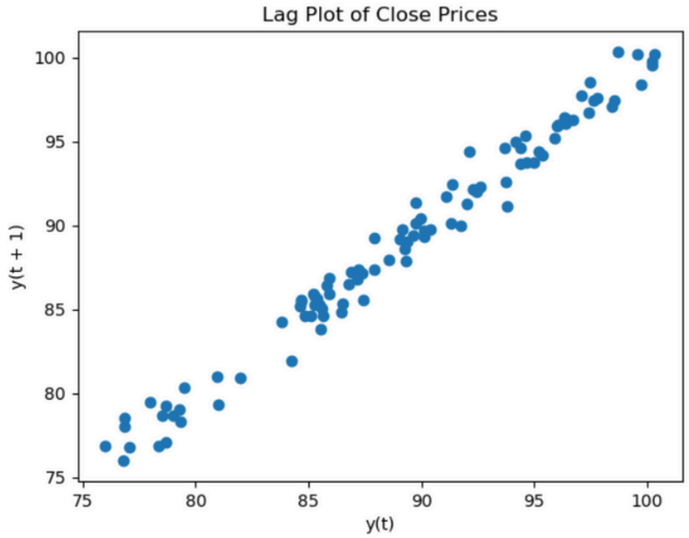



Open vs ATR (Color: Outcome)
0.0
1.0
ATR
Open


Waterfall Chart of Numerical Features Contribution to Outcome (Test Data)
Cumulative Contribution
Open
High
Low
Close
Volume
SMA
EMA
RSI
MACD
Upper_BB
Middle_BB
Lower_BB
%K
%D
ATR
H-L
H-PC
L-PC
TR
STD
Upper
Lower

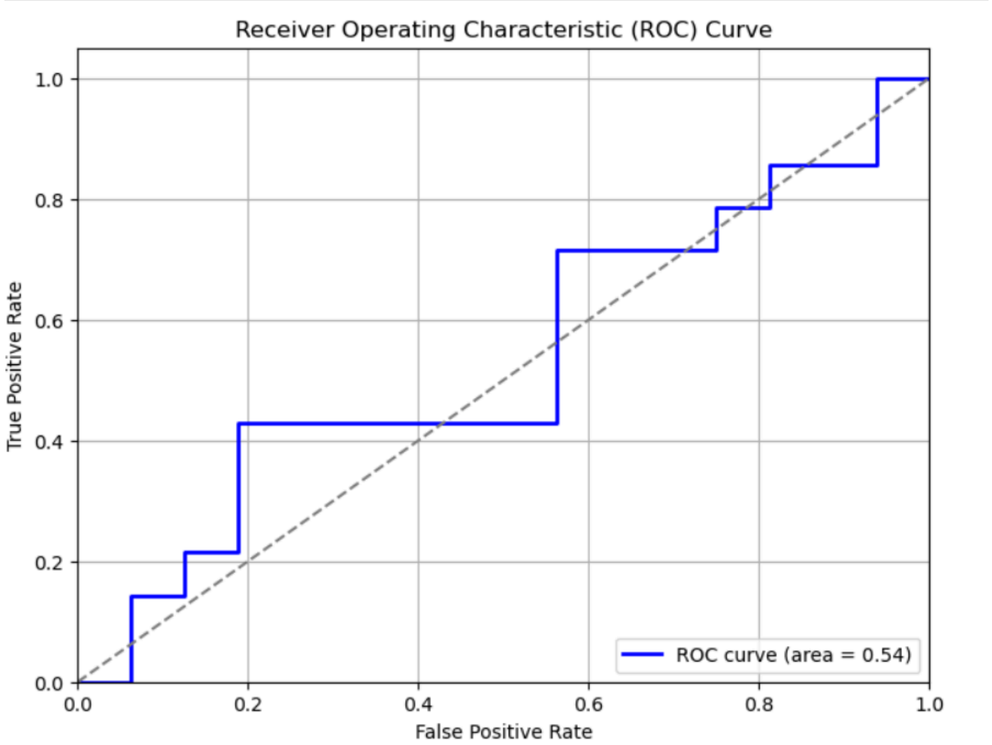

Time Series Plot of Close Prices
Close Price
Close Price
Date


Residual Plot
Residuals

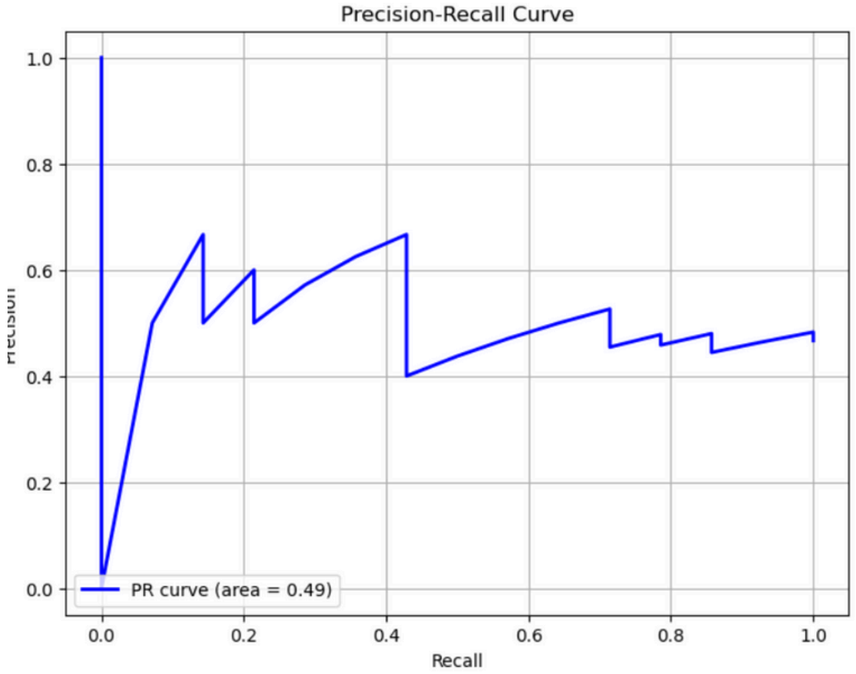

Lag Plot of Close Prices
y(t + 1)
y(t)

PLOTTINGS

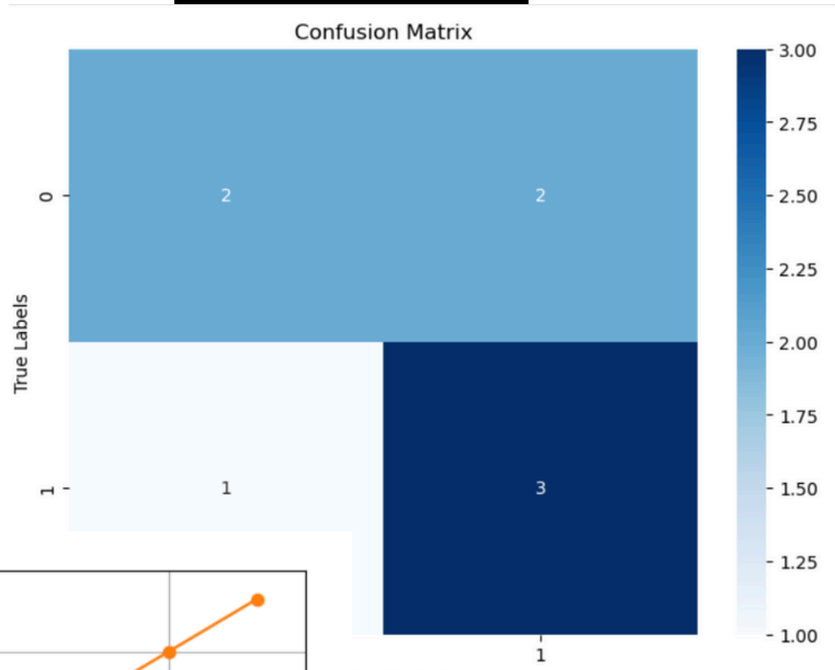
ROC CURVE

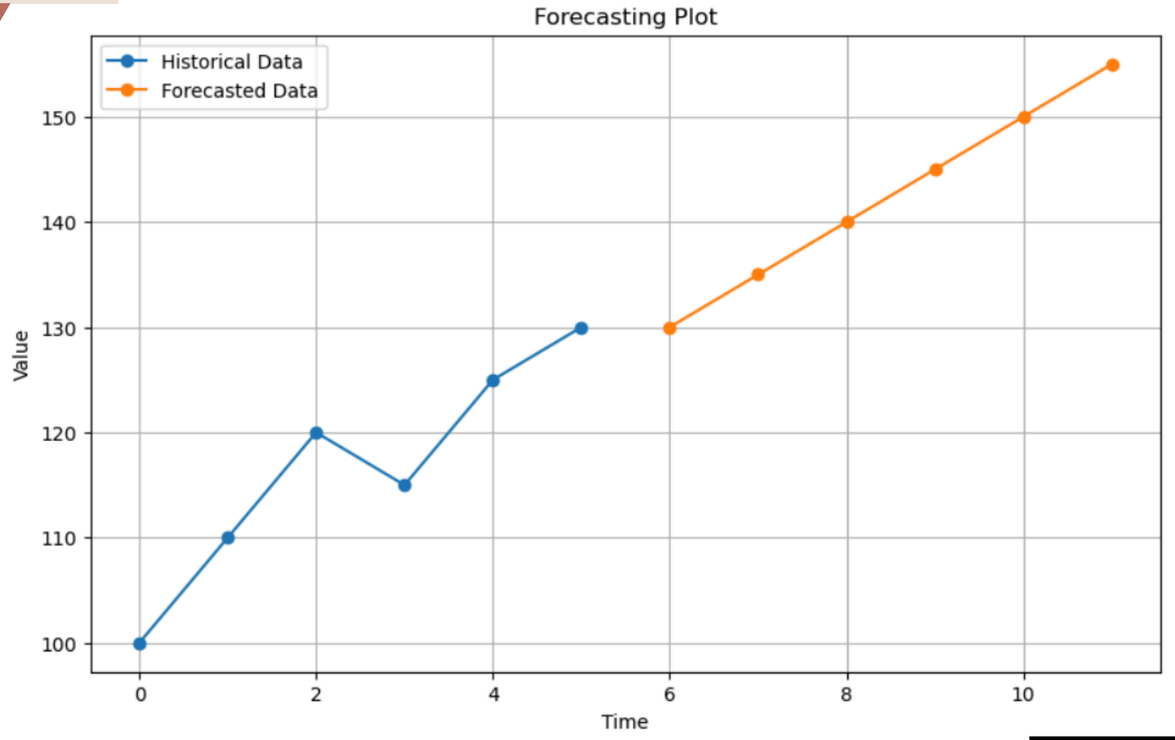


PR CURVE



CONFUSION MATRIX





TESTING: BLACKBOX

```python
class LSTMBlackBoxTest(unittest.TestCase):
    def setUp(self):
        # Load and preprocess data
        self.X_train, self.X_test, self.y_train, self.y_test = load_and_preprocess_data()
        self.sequence_length = self.X_train.shape[1]
        self.input_shape = (self.X_train.shape[1], self.X_train.shape[2])
        self.model = create_lstm_model(self.input_shape)

    def test_valid_input(self):
        # Create a valid input
        valid_input = np.random.rand(1, self.sequence_length, self.input_shape[1])

        # Get the prediction
        prediction = self.model.predict(valid_input)

        # Assertions
        self.assertEqual(prediction.shape, (1, 1))
        self.assertTrue(0 <= prediction[0, 0] <= 1)

# Function to run unittests
def run_tests():
    suite = unittest.TestLoader().loadTestsFromTestCase(LSTMBlackBoxTest)
    unittest.TextTestRunner().run(suite)

# Run the tests
run_tests()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```

```
.
----------------------------------------------------------------------
Ran 1 test in 1.796s

OK
```

CONCLUSION

Continued advancements in LSTM-based forecasting methodologies hold promise for further improving decision-making processes in financial markets, albeit with ongoing challenges in real-world implementation and deployment. These conclusions underscore LSTM's role as a powerful tool in forecasting directional movements, with ongoing research aimed at refining its capabilities and addressing current limitations.

THANK YOU