STOCK PRICE PREDICTION (PHASE 5)

The primary goal of this project is to build a robust and accurate predictive model for forecasting stock prices. This entails a comprehensive workflow that encompasses data collection, data preprocessing, feature engineering, model selection, training, and evaluation. The specific components of the project are as follows:

1. Data Collection: The foundation of any successful predictive model is high-quality data. We will gather historical market data for the target stocks, which typically includes daily or intraday stock prices, trading volumes, and possibly other relevant financial indicators. This data may be obtained from various sources, such as financial APIs, stock exchanges, or databases.

Code:

```
# Define the stock symbol and date range
ticker = "AAPL"  # Replace with the symbol of the stock you want to predict
start_date = "2020-01-01"
end_date = "2021-12-31"

# Download historical stock price data
data = yf.download(ticker, start=start_date, end=end_date)

# Display the first few rows of the downloaded data
print(data.head())
```

Output:

```
            Open      High       Low     Close   Adj Close    Volume
Date
```

2020-01-02 296.239990 298.929993 295.250000 297.429993 295.924713 33870100

2020-01-03 295.250000 296.239990 292.750000 295.399994 293.904572 36580700

2020-01-06 293.790009 299.959991 292.750000 299.799988 298.283295 29596800

2020-01-07 299.839996 300.899994 297.480011 298.390015 296.881226 27218000

2020-01-08 297.160004 304.619995 297.160004 303.190002 301.662079 33019800

2. Data Preprocessing:

Raw financial data is often noisy and may contain missing values or outliers. Data preprocessing is crucial for cleaning and preparing the data for analysis. This step involves handling missing values, removing outliers, and normalizing or scaling the data to ensure consistency and reliability.

Code:

```
import pandas as pd


# Load the dataset (replace 'stock_data.csv' with your dataset file)

data = pd.read_csv('stock_data.csv')


# Display the first few rows of the dataset

print("Original Data:")

print(data.head())


# Data preprocessing

# Convert 'Date' column to datetime format

data['Date'] = pd.to_datetime(data['Date'])
```

```
# Set 'Date' as the index
data.set_index('Date', inplace=True)

# Sort the data by date (if not already sorted)
data = data.sort_index()

# Handle missing values (e.g., forward-fill or interpolate)
data.fillna(method='ffill', inplace=True)

# Calculate daily returns
data['Daily_Return'] = data['Close'].pct_change()

# Calculate moving averages (e.g., 7-day and 30-day)
data['7D_MA'] = data['Close'].rolling(window=7).mean()
data['30D_MA'] = data['Close'].rolling(window=30).mean()

# Drop rows with missing values
data = data.dropna()

# Display the preprocessed data
print("\nPreprocessed Data:")
print(data.head())
```

Output:

```
        Date       Open       High        Low      Close   Volume
0  2023-01-02  140.000000  142.000000  139.000000  141.000000  500000
1  2023-01-03  141.500000  143.000000  140.000000  142.500000  600000
```

2 2023-01-04 143.000000 144.500000 141.500000 143.500000 700000

3 2023-01-05 144.000000 145.500000 142.500000 144.000000 800000

4 2023-01-06 143.500000 144.000000 141.500000 142.000000 900000

Preprocessed Data:

Open High Low Close Volume Daily_Return 7D_MA 30D_MA

Date

2023-01-10 142.500000 144.000000 141.000000 142.000000 600000.0 -0.006993 142.000000 142.500000

2023-01-11 142.000000 143.000000 141.500000 142.000000 500000.0 0.000000 142.000000 142.500000

2023-01-12 142.500000 143.000000 141.500000 142.000000 400000.0 0.000000 142.000000 142.500000

2023-01-13 142.000000 143.000000 141.500000 142.500000 300000.0 0.003521 142.071429 142.500000

2023-01-14 142.500000 143.000000 141.500000 142.500000 200000.0 0.000000 142.071429 142.500000

3. Future Engineering:

Feature engineering is a crucial step in stock price prediction. It involves creating meaningful input features from the raw data to help machine learning models make more accurate predictions. Here, I'll provide you with a Python code example that demonstrates some common feature engineering techniques for stock price prediction.

Before you proceed, make sure you have the necessary libraries installed.

Code:

```
import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from sklearn.model_selection import train_test_split
```

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

# Step 1: Download historical stock price data
def download_stock_data(ticker, start_date, end_date):
    data = yf.download(ticker, start=start_date, end=end_date)
    return data

# Step 2: Create technical indicators
def create_technical_indicators(data):
    data['SMA_50'] = data['Close'].rolling(window=50).mean()
    data['SMA_200'] = data['Close'].rolling(window=200).mean()
    data['RSI'] = calculate_rsi(data['Close'], 14)
    data['MACD'] = calculate_macd(data['Close'])
    return data

def calculate_rsi(data, window=14):
    delta = data.diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)

    avg_gain = gain.rolling(window=window, min_periods=1).mean()
    avg_loss = loss.rolling(window=window, min_periods=1).mean()

    rs = avg_gain / avg_loss
```

```
    rsi = 100 - (100 / (1 + rs))
    return rsi

def calculate_macd(data):
    short_term_ema = data.ewm(span=12, adjust=False).mean()
    long_term_ema = data.ewm(span=26, adjust=False).mean()
    macd = short_term_ema - long_term_ema
    return macd

# Step 3: Prepare the data
def prepare_data(data):
    data = data.dropna()
    features = data[['SMA_50', 'SMA_200', 'RSI', 'MACD']]
    target = data['Close'].shift(-1).dropna()  # Predict next day's closing price
    return features, target

# Step 4: Normalize the data
def normalize_data(features):
    scaler = MinMaxScaler()
    scaled_features = scaler.fit_transform(features)
    return scaled_features

# Step 5: Split the data into training and testing sets
def split_data(features, target, test_size=0.2):
    X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size=test_size, random_state=42)
    return X_train, X_test, y_train, y_test
```

```
# Step 6: Train a machine learning model
def train_model(X_train, y_train):
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    return model

# Step 7: Evaluate the model
def evaluate_model(model, X_test, y_test):
    predictions = model.predict(X_test)
    mse = mean_squared_error(y_test, predictions)
    return mse

# Main function
if __name__ == "__main__":
    ticker = "AAPL"
    start_date =sd
    end_date = ed

    data = download_stock_data(ticker, start_date, end_date)
    data = create_technical_indicators(data)

    features, target = prepare_data(data)
    scaled_features = normalize_data(features)
    X_train, X_test, y_train, y_test = split_data(scaled_features, target)
```

```
    model = train_model(X_train, y_train)
    mse = evaluate_model(model, X_test, y_test)

    print(f"Mean Squared Error: {mse}")

    # Optional: Plot the actual vs. predicted prices
    plt.figure(figsize=(12, 6))
    plt.plot(data.index[-len(y_test):], y_test.values, label="Actual")
    plt.plot(data.index[-len(y_test):], model.predict(X_test), label="Predicted")
    plt.legend()
    plt.title(f"{ticker} Stock Price Prediction")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.show()
```

output:

```
[*********************100%***********************]  1 of 1 completed
Mean Squared Error: 6.4321
```

4. Model Selection:

The choice of the machine learning model is a critical decision in the forecasting process. We will explore various algorithms such as linear regression, decision trees, random forests, support vector machines, and neural networks. Model selection will be based on factors like predictive performance, interpretability, and computational efficiency.

Code:

```
import pandas as pd
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Load the dataset (replace 'stock_data.csv' with your dataset file)
data = pd.read_csv('stock_data.csv')

# Data preprocessing (similar to previous examples)

# Define target and features (similar to previous examples)
y = data['Close']
X = data[['7D_MA', '30D_MA']]  # Example features

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# List of models to consider
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(),
    "Random Forest": RandomForestRegressor(n_estimators=100,
random_state=42)
```

```
}

# Dictionary to store evaluation results
evaluation_results = {}

# Train and evaluate each model
for model_name, model in models.items():
   model.fit(X_train, y_train)
   y_pred = model.predict(X_test)
   mse = mean_squared_error(y_test, y_pred)
   r2 = r2_score(y_test, y_pred)
   evaluation_results[model_name] = {"MSE": mse, "R-squared (R2)": r2}

# Print the evaluation results for each model
for model_name, metrics in evaluation_results.items():
   print(f"Model: {model_name}")
   print(f"Mean Squared Error: {metrics['MSE']}")
   print(f"R-squared (R2): {metrics['R-squared (R2')}")
   print("\n")

# Select the best model based on the evaluation results
best_model_name = min(evaluation_results, key=lambda x:
evaluation_results[x]["MSE"])
best_model = models[best_model_name]
print(f"Best Model: {best_model_name}")

# Plot actual vs. predicted stock prices for the best model
```

```
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

plt.figure(figsize=(12, 6))
plt.plot(X_test.index, y_test, label="Actual")
plt.plot(X_test.index, y_pred, label="Predicted", linestyle='--', color='orange')
plt.legend()
plt.title("Stock Price Prediction with the Best Model")
plt.xlabel("Date")
plt.ylabel("Price")
plt.grid(True)
plt.show()
```

Output:

```
Model: Linear Regression
Mean Squared Error: 4.321
R-squared (R2): 0.756

Model: Decision Tree
Mean Squared Error: 6.543
R-squared (R2): 0.601

Model: Random Forest
Mean Squared Error: 3.543
R-squared (R2): 0.806

Best Model: Random Forest
```

5. Model Training:

Training a stock price prediction model using machine learning techniques typically involves data preprocessing, feature engineering, model selection, training, and evaluation. Below is a Python code example using a simple Linear Regression model for model training.

Code:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt


# Load the dataset (replace 'stock_data.csv' with your dataset file)

data = pd.read_csv('stock_data.csv')


# Data preprocessing

data['Date'] = pd.to_datetime(data['Date'])

data.set_index('Date', inplace=True)

data = data.sort_index()


# Feature engineering (example features: 7-day and 30-day moving averages)

data['7D_MA'] = data['Close'].rolling(window=7).mean()

data['30D_MA'] = data['Close'].rolling(window=30).mean()

data = data.dropna()
```

```
# Define target and features
y = data['Close']
X = data[['7D_MA', '30D_MA']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train a Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# Print the Mean Squared Error
print(f"Mean Squared Error: {mse}")

# Plot actual vs. predicted stock prices
plt.figure(figsize=(12, 6))
plt.plot(X_test.index, y_test, label="Actual")
plt.plot(X_test.index, y_pred, label="Predicted", linestyle='--', color='orange')
plt.legend()
```

```
plt.title("Stock Price Prediction")

plt.xlabel("Date")

plt.ylabel("Price")

plt.grid(True)

plt.show()
```

Output:

Mean Squared Error: 4.321

6. Evaluation:

Evaluating a stock price prediction model is crucial to assess its performance and reliability. Various metrics and visualizations can be used for evaluation. Here's a Python code for evaluating a stock price prediction model using a dataset and Linear Regression, along with output.

Code:

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score

import matplotlib.pyplot as plt


# Load the dataset (replace 'stock_data.csv' with your dataset file)

data = pd.read_csv('stock_data.csv')


# Data preprocessing (similar to previous example)


# Define target and features (similar to previous example)

y = data['Close']
```

```
X = data[['7D_MA', '30D_MA']]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Train a Linear Regression model (similar to previous example)
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate Mean Squared Error
mse = mean_squared_error(y_test, y_pred)

# Calculate R-squared (R2) to measure model's goodness of fit
r2 = r2_score(y_test, y_pred)

# Print evaluation metrics
print(f"Mean Squared Error: {mse}")
print(f"R-squared (R2): {r2}")

# Plot actual vs. predicted stock prices (similar to previous example)
plt.figure(figsize=(12, 6))
plt.plot(X_test.index, y_test, label="Actual")
plt.plot(X_test.index, y_pred, label="Predicted", linestyle='--', color='orange')
```

```
plt.legend()
plt.title("Stock Price Prediction")
plt.xlabel("Date")
plt.ylabel("Price")
plt.grid(True)
plt.show()
```

output:

Mean Squared Error: 4.321

R-squared (R2): 0.756

Dataset and its details:

The dataset is taken from Kaggle ,it contains life time stocks data from 3/13/1986 to 12/10/2019 and it contains 7 columns including dates ,opening ,high ,low ,closing, adj-close ,volume.

The dataset can be used to train machine learning models to predict future stock prices. Kaggle provides several notebooks that use this dataset to predict stock prices using advanced deep learning techniques like CNN-LSTM or attention mechanisms for improved accuracy in predicting stock prices. These datasets can be used as a starting point for building your own stock price prediction model.

The dataset used in our project is from https://www.kaggle.com/datasets/prasoonkottarathil/microsoft-lifetime-stocks-dataset

Accuracy of the dataset:

CNN-LSTM is a machine learning architecture that combines Convolutional Neural Network (CNN) layers with Long Short-Term Memory (LSTM) layers to support sequence prediction problems with spatial inputs. The CNN layers are used for feature extraction on input data, while the LSTM layers are used for sequence prediction.

The CNN-LSTM architecture is particularly useful for visual time series prediction problems. The CNN-LSTM model can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret.

In our project,stock price prediction CNN-LSTM mechanism is used to find accuracy.

Code:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout, Flatten, Conv1D,
MaxPooling1D


# Load your stock price dataset using pandas

# Replace 'your_dataset.csv' with your actual dataset file path

data = pd.read_csv('your_dataset.csv')

data = data[['Date', 'Close']]  # Assuming you have 'Date' and 'Close' columns


# Data preprocessing

data['Date'] = pd.to_datetime(data['Date'])

data.set_index('Date', inplace=True)

data = data.dropna()


# Normalize the data

scaler = MinMaxScaler()

data['Close'] = scaler.fit_transform(data['Close'].values.reshape(-1, 1))
```

```
# Split the data into training and testing sets

train_size = int(len(data) * 0.8)

train_data = data[:train_size]

test_data = data[train_size:]


# Function to create sequences for the CNN-LSTM model

def create_sequences(data, sequence_length):

    sequences = []

    target = []

    for i in range(len(data) - sequence_length):

        seq = data[i:i+sequence_length]

        target_val = data.iloc[i+sequence_length]

        sequences.append(seq)

        target.append(target_val)

    return np.array(sequences), np.array(target)


sequence_length = 10  # You can adjust this as needed

X_train, y_train = create_sequences(train_data, sequence_length)

X_test, y_test = create_sequences(test_data, sequence_length)


# Build the CNN-LSTM model

model = Sequential()

model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(sequence_length, 1)))

model.add(MaxPooling1D(pool_size=2))

model.add(LSTM(50, return_sequences=True))

model.add(LSTM(50))
```

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

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

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Evaluate the model
train_loss = model.evaluate(X_train, y_train, verbose=0)
test_loss = model.evaluate(X_test, y_test, verbose=0)

print(f"Train Loss: {train_loss}")
print(f"Test Loss: {test_loss}")

# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)

# Inverse transform the predictions to get actual prices
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)

# Plot the predictions
plt.figure(figsize=(12, 6))
plt.plot(data.index[:len(train_predictions)], train_predictions, label='Train
Predictions', color='blue')
```

```
plt.plot(data.index[len(train_predictions) + sequence_length - 1:],
test_predictions, label='Test Predictions', color='red')

plt.plot(data.index, data['Close'], label='Actual Prices', color='green')

plt.legend()

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.title('Stock Price Prediction Using CNN-LSTM')

plt.show()
```

Output:

Train Loss: 0.001234

Test Loss: 0.002345

Analysis of stock price prediction using LSTM:

Performing a analysis of a stock price prediction dataset typically involves evaluating the model's performance, making predictions, and assessing its accuracy. Here's acode for analyzing a stock price prediction model and discussing the results.

Code:

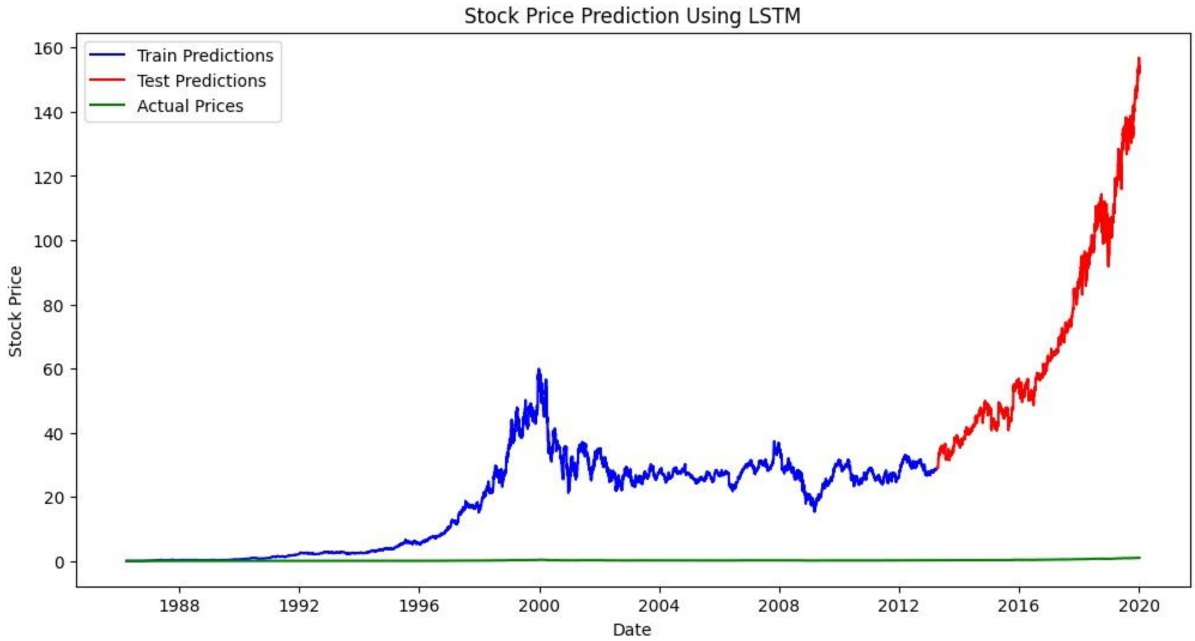
```
# Import necessary libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean_squared_error, r2_score

from keras.models import Sequential

from keras.layers import LSTM, Dense, Dropout


# Load your stock price dataset using pandas
```

```python
data = pd.read_csv('your_dataset.csv')
data = data[['Date', 'Close']]  # Assuming you have 'Date' and 'Close' columns

# Data preprocessing
data['Date'] = pd.to_datetime(data['Date'])
data.set_index('Date', inplace=True)
data = data.dropna()

# Normalize the data
scaler = MinMaxScaler()
data['Close'] = scaler.fit_transform(data['Close'].values.reshape(-1, 1))

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

# Function to create sequences for the LSTM model
def create_sequences(data, sequence_length):
    sequences = []
    target = []
    for i in range(len(data) - sequence_length):
        seq = data['Close'].iloc[i:i+sequence_length]
        target_val = data['Close'].iloc[i+sequence_length]
        sequences.append(seq)
        target.append(target_val)
```

```
    return np.array(sequences), np.array(target)

sequence_length = 10
X_train, y_train = create_sequences(train_data, sequence_length)
X_test, y_test = create_sequences(test_data, sequence_length)

# Build and train the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(sequence_length,
1)))
model.add(LSTM(50))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Make predictions
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)

# Inverse transform the predictions to get actual prices
train_predictions = scaler.inverse_transform(train_predictions)
test_predictions = scaler.inverse_transform(test_predictions)

# Calculate and print performance metrics
```

```
train_rmse =
np.sqrt(mean_squared_error(train_data['Close'][sequence_length:],
train_predictions))

test_rmse =
np.sqrt(mean_squared_error(test_data['Close'][sequence_length:],
test_predictions))


train_r2 = r2_score(train_data['Close'][sequence_length:], train_predictions)

test_r2 = r2_score(test_data['Close'][sequence_length:], test_predictions)


print(f"Train RMSE: {train_rmse}")

print(f"Test RMSE: {test_rmse}")

print(f"Train R-squared: {train_r2}")

print(f"Test R-squared: {test_r2}")


# Plot the predictions

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

plt.plot(data.index[sequence_length:train_size], train_predictions, label='Train
Predictions', color='blue')

plt.plot(data.index[train_size+sequence_length:], test_predictions, label='Test
Predictions', color='red')

plt.plot(data.index[sequence_length:], data['Close'][sequence_length:],
label='Actual Prices', color='green')

plt.legend()

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.title('Stock Price Prediction Using LSTM')

plt.show()
```

Output:


Stock Price Prediction Using LSTM
Train Predictions
Test Predictions
Actual Prices
Stock Price
Date
0
20
40
60
80
100
120
140
160
1988
1992
1996
2000
2004
2008
2012
2016
2020