Stock Price Prediction

Development Part-1

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
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
```

```python
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

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()
```

```python
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

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