



گزارش پایانی آزمایشگاه سخت افزار

دیبا مسیحی ۹۷۱۱۰۲۷۴
محمدرضا دویران ۹۸۱۰۱۵۶۶
آرین یزدان پرست ۹۸۱۱۰۰۹۵

چکیده

در این گزارش، ما یک روند گام به گام برای پیاده سازی استراتژی های معاملاتی مالی با استفاده از Back Test و ادغام پلتفرم مالی Alpaca و مدل های شبکه های عصبی LSTM ارائه می دهیم. تمامی کدهای استفاده شده در طول این پروژه در این لینک موجود می باشد.

۱ مقدمه

پیش بینی قیمت های آتی سهام به کمک اطلاعات تاریخی درباره ی حرکت های قیمت و حجم معاملات سهام انجام می شود. از طریق تحلیل الگوها و روندها در داده های گذشته، از جمله رابطه بین تغییرات قیمت و حجم معاملات، مدل های پیش بینی برای تخمین حرکات آینده قیمت توسعه داده می شود. این اطلاعات برای سرمایه گذاران و معامله گرانی که به تصمیم گیری آگاهانه در مورد خرید یا فروش سهام و شناسایی فرصت های سرمایه گذاری علاقه مند هستند، بسیار ارزشمند هستند. در این پروژه، از مدل LSTM مبتنی بر TinyML برای پیش بینی قیمت های سهام در آینده استفاده شده است. با آموزش این مدل با استفاده از داده های تاریخی مربوط به قیمت و حجم معاملات، از الگوها و ارتباطات موجود در داده ها برای پیش بینی حرکات آتی قیمت ها بهره برداری شده است. Pi Raspberry به عنوان پلتفرمی برای استقرار مدل های آموزش دیده عمل می کند و امکان پیش بینی در زمان واقعی و تصمیم گیری های معاملاتی خودکار را فراهم می آورد. هدف اصلی این پروژه، ایجاد یک راه حل فشرده و با مصرف مناسب با بهینه سازی مصرف CPU و حافظه است. این ابزار می تواند به معامله گران به منظور تصمیم گیری مبتنی بر داده و افزایش بازدهی در بازار پویای سهام کمک کند. این پروژه به اجرای موفقیت آمیز مدل LSTM بر روی یک Pi Raspberry با استفاده از TinyML و اتصال به پلتفرم معاملات سهام پرداخته است. این کار با هدف تسهیل تصمیم گیری های معاملاتی بر اساس پیش بینی قیمت های آتی سهام انجام شده است.

۲ مراحل پروژه

۱.۲ نصب نیازمندی ها

در ابتدا، نیازمندی های پروژه را بر روی Pi Raspberry نصب می کنیم. این نیازمندی ها شامل کتابخانه های Tensorflow و yfinance برای استخراج داده های مالی می باشند. همچنین، کتابخانه های pandas و numpy نیز به صورت پیش فرض در سیستم عامل موجود هستند. تمام کتابخانه های مورد استفاده در این پروژه به شرح زیر می باشند:

```
import pandas as pd
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import Callback
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.optimizers import Adam
```

```
import tensorflow as tf
import tensorflow.lite as tflite
import tensorflow_model_optimization as tfmot
```

```
import backtrader as bt
import pandas as pd
```

```
import yfinance as yf
import numpy as np
```

```
import datetime
```

```
import IPython.display as display
import seaborn as sns
```

```
import mplfinance as mpf
```

۲.۲ استخراج داده

داده‌های مالی مرتبط با سهام مورد نظر را با استفاده از کد مناسب از منابع مالی دریافت می‌کنیم. این داده‌ها شامل اطلاعاتی نظیر اولین و آخرین قیمت برای هر دارایی در روز، حداکثر و حداقل قیمت آن در روز، حجم معاملات و دیگر مشخصات مرتبط می‌شوند. در مرحله بعدی، داده‌ها را پیش‌پردازش می‌کنیم. این مراحل شامل تبدیل داده‌های ورودی به مقادیر بین ۰ و ۱ (نرمالایز کردن داده‌ها) و تقسیم داده‌ها به دو بخش X و y برای استفاده در مدل‌های یادگیری ماشین می‌شود.

Implementation

```
def download_stock_data(ticker, start_date, end_date, interval_time = '1h'):
    | return yf.download(ticker, start=start_date, end=end_date, interval=interval_time)

def min_max_scaling(x):
    | return (x - x.min()) / (x.max() - x.min())
```

Data Preprocessing

```
def preprocess_data(data):
    print(data['Close'].max(), data['High'].values[0])
    data = data.apply(min_max_scaling)
    print(data['Close'].max(), data['High'].values[0])

    look_back = 60
    X, y = [], []
    for i in range(len(data) - look_back):
        | X.append(data[['Open', 'High', 'Low', 'Close', 'Volume']].values[i:i + look_back])
        | y.append(data['Close'].values[i + look_back])
    X = np.array(X)
    y = np.array(y)

    return X, y
```

۳.۲ ساخت مدل LSTM

این مدل برای پیش‌بینی قیمت‌های سهام استفاده می‌شود. مدل به صورت زیر تعریف می‌شود:

```
def build_lstm_model(input_shape):
    | model = Sequential()
    | model.add(LSTM(units=60, return_sequences=True, input_shape=input_shape))
    | model.add(LSTM(units=50))
    | model.add(Dense(units=1))
    learning_rate = 1e-5
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model
```

```
def inverse_min_max_scaling(x, min_val, max_val):
    | return x * (max_val - min_val) + min_val
```

۴.۲ پیش‌بینی قیمت‌ها

پس از آموزش مدل با استفاده از داده‌های آموزش، می‌توانیم از مدل برای پیش‌بینی قیمت‌های سهام در آینده استفاده کنیم. در اینجا، تابعی برای پیش‌بینی قیمت‌ها تعریف شده است:

```
def predict_stock_price(model, X_new):
    X_new = X_new.reshape(1, X_new.shape[0], X_new.shape[1])
    predicted_price = model.predict(X_new)
    return predicted_price[0][0]
```

```
predictions = []
for i in range(len(X_new)):
    prediction = predict_stock_price(model, X_new[i])
    predictions.append(prediction)
```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)

```
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 32ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 37ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 35ms/step
1/1 [=====] - 0s 31ms/step
1/1 [=====] - 0s 29ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 34ms/step
1/1 [=====] - 0s 30ms/step
1/1 [=====] - 0s 27ms/step
...
```

Mode: 0 0 0

۵.۲ تبدیل پیش‌بینی‌های مدل به مقادیر واقعی

برای ارزیابی دقیق‌تر پیش‌بینی‌های مدل، ما از تابع تبدیل معکوس زیر برای تبدیل پیش‌بینی‌های مدل به مقادیر واقعی سهام استفاده کردیم:

```
def inverse_min_max_scaling(x, min_val, max_val):
    return x * (max_val - min_val) + min_val
```

سپس مقادیر واقعی را با مقادیر واقعی سهام مقایسه کردیم تا دقت و عملکرد پیش‌بینی‌های مدل را ارزیابی کنیم.

۶.۲ معیارهای ارزیابی

ما سه معیار مهم برای ارزیابی مدل معرفی کردیم:

• SMAPE که بر اساس رابطه زیر محاسبه می‌شود:

$$SMAPE = \frac{100}{n} \sum_{i=1}^n \frac{|A_i - P_i|}{\frac{|A_i| + |P_i|}{2}}$$

SMAPE

```
import numpy as np

def calculate_smape(actual, predicted):

    numerator = np.abs(predicted - actual)
    denominator = (np.abs(predicted) + np.abs(actual)) / 2

    smaпе = np.mean(numerator / denominator) * 100

    return smaпе
```

• MAPE که بر اساس رابطه زیر محاسبه می‌شود:

$$MAPE = \frac{1}{n} \sum_{i=1}^n \frac{|A_i - P_i|}{|A_i|} \times 100$$

MAPE

```
import numpy as np

def calculate_mape(actual, predicted):

    actual = np.array(actual)
    predicted = np.array(predicted)

    absolute_percentage_errors = np.abs((actual - predicted) / actual)
    mape = np.mean(absolute_percentage_errors) * 100

    return mape
```

- RMSE که بر اساس رابطه زیر محاسبه می‌شود:

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (A_i - P_i)^2}$$

RSME

```
def calculate_rmse(actual, predicted):
    actual = np.array(actual)
    predicted = np.array(predicted)

    squared_errors = (predicted - actual) ** 2
    mean_squared_error = np.mean(squared_errors)
    rmse = np.sqrt(mean_squared_error)

    return rmse
```

۷.۲ بررسی سهام‌های مختلف

تعریف نمادهای مورد نظر برای اپل، بیت‌کوین و مایکروسافت به صورت زیر است:

```
apple_ticker = 'AAPL'
bitcoin_ticker = "BTC-USD"
msft_ticker = "MSFT"
ticker = apple_ticker

start_date = '2022-04-12'
end_date = '2023-04-01'
data = download_stock_data(ticker, start_date, end_date)
X_train, y_train = preprocess_data(data)
```

```
[*****100%*****] 1 of 1 completed
175.6649932861328 169.8699951171875
```

۸.۲ کوانتایز کردن مدل

در این قسمت ابتدا کتابخانه‌های مد نظر را نصب می‌کنیم.

```
tensorflow_model_optimization install !pip
```

سپس با استفاده از کد زیر مدل را کوانتایز می‌کنیم.

```

annotated_layers = [
    tfmot.quantization.keras.quantize_annotate_layer(model.layers[0]),
    tfmot.quantization.keras.quantize_annotate_layer(model.layers[1])

]

quantize_model = tf.keras.Sequential(annotated_layers + model.layers[2:])

quantize_model.compile(optimizer='adam', loss='mean_squared_error')

converter = tf.lite.TFLiteConverter.from_keras_model(quantize_model)

converter.experimental_new_converter = True

converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS, tf.lite.OpsSet.SELECT_TF_OPS]
converter._experimental_lower_tensor_list_ops = False

```

در مرحله بعدی با استفاده از مدل کوانتایز شده پیش‌بینی را انجام می‌دهیم.

Run inference

```

def predict_stock_price_quantized(X_new):
    X_new = X_new.reshape(1, X_new.shape[0], X_new.shape[1])
    X_new = X_new.astype(np.float32)
    interpreter.set_tensor(input_details[0]['index'], X_new)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])

    return output_data[0][0]

```

مدل اول که بر مبنای شبکه‌های LSTM استفاده می‌کند، پیچیده و مصرف‌کننده منابع محاسباتی است و دقت پیش‌بینی بالایی را ارائه می‌دهد، اما به زمان و منابع آموزش قابل توجهی نیاز دارد. در مقابل، مدل کوانتایز شده، یک نسخه ساده‌تر از مدل اول است که به بهینگی و سهولت استقرار تمرکز دارد. این مدل، برخی از دقت پیش‌بینی را فدای بهره‌وری منابع می‌کند و برای برنامه‌های معاملات در زمان واقعی در دستگاه‌های منابع محدود، بهینه است. انتخاب بین این دو مدل به وابستگی به مورد استفاده خاص برنامه بوده و مدل اول برای تحقیقات و بازآزمایش‌ها مناسب است، در حالی که مدل کوانتایز شده برای سناریوهای تصمیم‌گیری سریع که به بهره‌وری منابع حیاتی است، عالی عمل می‌کند.

در مقابل، مدل کوانتایز شده که ساده‌تر و کم‌منابع‌تر است، همچنان دقت قابل قبولی در پیش‌بینی دارد. علاوه بر این، مدل کوانتایز شده برای اجرا در زمان واقعی و در محیط‌های با منابع محدود مناسب است. این مدل، به خصوص برای تصمیم‌گیری‌های سریع و مواجهه با نوسانات بازار، عملکرد خوبی از خود نشان می‌دهد. انتخاب مدل مناسب به وابستگی به مورد استفاده و منابع موجود برنامه بستگی دارد.

۹.۲ مقایسه مدل اولیه و مدل کوانتیزه شده

برای بررسی و عملکرد مدل کوانتیزه شده این مدل را با مدل اولیه از جهات مختلف مقایسه می‌کنیم. ابتدا اطلاعات مربوط به مدل اصلی را مشاهده می‌کنیم:

```
model_path = 'apple_model.h5'
plain_model = tf.keras.models.load_model(model_path)
plain_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 60, 60)	15840
lstm_1 (LSTM)	(None, 50)	22200
dense (Dense)	(None, 1)	51
Total params: 38,091		
Trainable params: 38,091		
Non-trainable params: 0		

حال اطلاعات مربوط به ورودی مدل کوانتیزه شده را نمایش می‌دهیم:

input_details

```
[{'name': 'serving_default_quantize_annotate_2_input:0',
  'index': 0,
  'shape': array([ 1, 60,  5], dtype=int32),
  'shape_signature': array([-1, 60,  5], dtype=int32),
  'dtype': numpy.float32,
  'quantization': (0.0, 0),
  'quantization_parameters': {'scales': array([], dtype=float32),
  'zero_points': array([], dtype=int32),
  'quantized_dimension': 0},
  'sparsity_parameters': {}}]
```

حال اطلاعات خروجی این مدل را بررسی می‌کنیم:

output_details

```
[{'name': 'StatefulPartitionedCall:0',  
  'index': 48,  
  'shape': array([1, 1], dtype=int32),  
  'shape_signature': array([-1, 1], dtype=int32),  
  'dtype': numpy.float32,  
  'quantization': (0.0, 0),  
  'quantization_parameters': {'scales': array([], dtype=float32),  
  'zero_points': array([], dtype=int32),  
  'quantized_dimension': 0},  
  'sparsity_parameters': {}}]
```

۱.۹.۲ تعداد پارامترهای قابل آموزش

در ادامه پارامترهای قابل آموزش هر یک از این دو مدل را نمایش می‌دهیم:

```
full_model_trainable_params = plain_model.count_params()  
tflite_model_trainable_params = 0  
for tensor in interpreter.get_tensor_details():  
    shape = tensor['shape']  
    if len(shape) > 0:  
        tflite_model_trainable_params += shape[-1]
```

```
print("Full Model Trainable Parameters:", full_model_trainable_params)  
print("TFLite Model Trainable Parameters:", tflite_model_trainable_params)
```

```
Full Model Trainable Parameters: 38091  
TFLite Model Trainable Parameters: 1148
```

مشاهده می‌شود که تعداد پارامترهای قابل آموزش مدل اولیه ۳۸۰۹۱ است در حالی که همین مورد برای مدل کوانتیزه شده برابر ۱۱۴۸ است که حدود ۹۷ درصد کمتر است و این مورد نشان دهنده‌ی سریع‌تر عمل کردن مدل کوانتیزه شده می‌باشد.

۲.۹.۲ تعداد لایه‌ها

با اینکه تعداد پارامترها و نودهای قابل آموزش مدل کوانتیزه شده کمتر است اما این نودها در لایه‌های بیشتری قرار دارند که این را می‌توان در تصویر زیر مقایسه کرد:

Number of Layers

```
full_model_num_layers = len(plain_model.layers)

tflite_model_num_layers = len(interpreter.get_tensor_details())

print("Full Model Number of Layers:", full_model_num_layers)
print("TFLite Model Number of Layers:", tflite_model_num_layers)
```

```
Full Model Number of Layers: 3
TFLite Model Number of Layers: 49
```

۳.۹.۲ سایز مدل

اگر سایز مدل‌ها را نیز بررسی کنیم مشاهده می‌کنیم که سایز مدل اولیه سه برابر مدل کوانتیزه شده است:

Size

```
full_model_size = os.path.getsize('apple_model.h5')
tflite_model_size = os.path.getsize('apple_quantized_model.tflite')

print("Full Model Size (bytes):", full_model_size)
print("TFLite Model Size (bytes):", tflite_model_size)
```

```
Full Model Size (bytes): 499784
TFLite Model Size (bytes): 170536
```

۴.۹.۲ سرعت پیش‌بینی

یکی از فاکتورهای مهم برای مقایسه دو مدل سرعت تصمیم‌گیری و پیش‌بینی آن‌ها است. برای مقایسه این فاکتور ابتدا روی یک داده رندوم این مقایسه را انجام می‌دهیم:

Inference time on random data

```
input_shape = plain_model.input_shape[1:]
sample_input = tf.random.uniform((1, *input_shape))

start_time = time.time()
_ = plain_model.predict(sample_input)
full_model_inference_time = time.time() - start_time

tflite_input_index = interpreter.get_input_details()[0]['index']
tflite_output_index = interpreter.get_output_details()[0]['index']

start_time = time.time()
interpreter.set_tensor(tflite_input_index, sample_input)
interpreter.invoke()
tflite_model_inference_time = time.time() - start_time

print("Full Model Inference Time:", full_model_inference_time)
print("TFLite Model Inference Time:", tflite_model_inference_time)
```

```
1/1 [=====] - 0s 135ms/step
Full Model Inference Time: 0.2532010078430176
TFLite Model Inference Time: 0.010959625244140625
```

مشاهده می‌شود که مدت زمان مورد نیاز برای انجام یک پیش‌بینی توسط مدل کوانتیزه شده حدود ۴ درصد زمان مورد نیاز برای مدل اولیه است. حال برای بررسی دقیق‌تر بر روی داده تست خود، که شامل ۴۴۴ داده مربوط به ساعات مختلف است، زمان مورد نیاز را برای پیش‌بینی به ازای هر مدل به دست می‌آوریم. برای این کار ابتدا داده تست را دانلود می‌کنیم:

```

ticker = 'AAPL'
start_date = '2023-05-02'
end_date = '2023-08-03'

data = download_stock_data(ticker, start_date, end_date)
x_min = data.min()
x_max = data.max()
data.shape

```

```

[*****100%*****] 1 of 1 completed

(444, 6)

```

حال مدت زمان مورد نیاز برای پیش‌بینی‌ها توسط هر مدل را به دست می‌آوریم:

```

start_time = time.time()
for i in range(len(X_new)):
    _ = predict_stock_price_quantized(X_new[i])
tflite_model_inference_time = time.time() - start_time

```

```

start_time = time.time()
for i in range(len(X_new)):
    _ = predict_stock_price(X_new[i])
full_model_inference_time = time.time() - start_time

```

```

print("Full Model Inference Time:", full_model_inference_time)
print("TFLite Model Inference Time:", tflite_model_inference_time)

```

```

Full Model Inference Time: 42.152159452438354
TFLite Model Inference Time: 4.159500360488892

```

مشاهده می‌کنیم که در این حالت نیز سرعت مدل کوانتیزه شده بسیار بهتر است و نزدیک به ۱۰ برابر مدل اولیه است.

۵.۹.۲ دقت مدل با استفاده از متریک‌ها

در نهایت می‌خواهیم پیش‌بینی‌های دو مدل را با توجه به متریک‌های MAPE و SMAPE و RMSE مقایسه کنیم. حال پیش‌بینی‌ها را انجام داده و ذخیره می‌کنیم و متریک‌ها را برای هر یک از مدل‌ها به دست می‌آوریم:

```
lite_predictions = []
for i in range(len(X_new)):
    prediction = predict_stock_price_quantized(X_new[i])
    lite_predictions.append(prediction)

predictions = np.asarray(lite_predictions)
predictions_reversed = inverse_min_max_scaling(predictions, x_min['Close'], x_max['Close'])

y_new = np.asarray(y_new)
y_new_rev = inverse_min_max_scaling(y_new, x_min['Close'], x_max['Close'])

for i in range(len(predictions_reversed)):
    date = data.index[i]
    predicted_price = predictions_reversed[i]
    print(f"Date: {date}, Predicted Price: {predicted_price:.2f}, Actual Price: {y_new_rev[i]}")

lite_mape = calculate_mape(y_new_rev, predictions_reversed)
lite_smape = calculate_smape(y_new_rev, predictions_reversed)
lite_rmse = calculate_rmse(y_new_rev, predictions_reversed)
```

```
plain_predictions = []
for i in range(len(X_new)):
    prediction = predict_stock_price(X_new[i])
    plain_predictions.append(prediction)

predictions = np.asarray(plain_predictions)
predictions_reversed = inverse_min_max_scaling(predictions, x_min['Close'], x_max['Close'])

y_new = np.asarray(y_new)
y_new_rev = inverse_min_max_scaling(y_new, x_min['Close'], x_max['Close'])

for i in range(len(predictions_reversed)):
    date = data.index[i]
    predicted_price = predictions_reversed[i]
    print(f"Date: {date}, Predicted Price: {predicted_price:.2f}, Actual Price: {y_new_rev[i]}")

full_mape = calculate_mape(y_new_rev, predictions_reversed)
full_smape = calculate_smape(y_new_rev, predictions_reversed)
full_rmse = calculate_rmse(y_new_rev, predictions_reversed)
```

```
print(f'Full Model --- MAPE: {full_mape:.2f}, SMAPE: {full_smape:.2f}, RMSE: {full_rmse:.2f}')
print(f'Quantized Model --- MAPE: {lite_mape:.2f}, SMAPE: {lite_smape:.2f}, RMSE: {lite_rmse:.2f}')
```

```
Full Model --- MAPE: 1.14, SMAPE: 1.14, RMSE: 2.47
Quantized Model --- MAPE: 1.14, SMAPE: 1.14, RMSE: 2.47
```

می‌بینیم که عملکرد هر دو مدل با توجه به متریک‌های مختلف تا دو رقم اعشار یکسان است! پس عملاً عملکرد مدل کوانتیزه در پیش‌بینی با مدل اولیه تفاوتی ندارد؛ در حالی که حجم آن کمتر است و سرعت بسیار بالاتری دارد. این اتفاق نشان‌دهنده موثر بودن و مفید بودن عملی کوانتایز کردن مدل می‌باشد و از این پس تنها از مدل کوانتیزه شده برای پیش‌بینی‌ها استفاده می‌کنیم.

Alpaca یک پلتفرم مالی مجازی است که به ما امکان می‌دهد تا به داده‌های تاریخی و درجریان بازارهای مالی دسترسی داشته باشیم و حتی معاملات واقعی را انجام دهیم. این ابزار توانایی‌هایی مانند دسترسی به قیمت‌های سهام، آنالیز تکنیکال، و انجام معاملات در بازارهای مختلف را فراهم می‌کند. برای استفاده از این پلتفرم، ابتدا کتابخانه Backtrader را نصب و سپس با استفاده از مدل کوانتیزه‌ای که در بخش قبلی توضیح داده شد، اتصال به Alpaca را برقرار می‌کنیم.

در ابتدا برای پیاده‌سازی استراتژی معاملاتی خود، از کلاس LSTMStrategy به عنوان یک زیرکلاس از کلاس استراتژی Backtrader استفاده کردیم. در این کلاس، متد next برای هر نقطه داده جدید اجرا می‌شود. اگر تعداد داده‌ها برابر یا بیشتر از ۶۰ ساعت باشد، مدل LSTM را برای پیش‌بینی استفاده می‌کنیم. در صورتی که پیش‌بینی نشان دهد که قیمت نسبت به یک ساعت قبل افزایش یافته است، سیگنال خرید صادر می‌شود و اگر نشان دهد که کاهش یافته است، سیگنال بستن معامله صادر می‌شود.

برای پیش‌بینی قیمت‌ها از تابعی به نام quantized price stock predict استفاده می‌کنیم که با در نظر گرفتن داده‌های قدیمی‌تر، قیمت بازار سهام را برای زمان بعدی پیش‌بینی می‌کند. این تابع روی مدل کوانتیزه‌شده اجرا می‌شود و به توانایی مدل در پیش‌بینی قیمت‌ها کمک می‌کند.

```
def predict_price_lstm(historical_data):
    | return predict_stock_price_quantized(historical_data)
```

برای انجام بازاریابی، از موتور Cerebro که بخشی از کتابخانه Backtrader است، استفاده می‌کنیم. در این مرحله، نقدینگی اولیه تعیین می‌شود و داده‌های تاریخی به عنوان ورودی اضافه می‌شوند. سپس استراتژی LSTM به Cerebro اضافه می‌شود و بازاریابی با اجرای نمونه Cerebro انجام می‌شود.

```
cerebro = bt.Cerebro(stdstats=True)
cerebro.broker.setcash(10000)
cerebro.adddata(feed)
cerebro.addstrategy(LSTMStrategy, look_back=60)
```

برای توسعه‌ی استراتژی خود در هفته‌های بعدی، استراتژی را کمی تغییر و بهبود دادیم. در استراتژی جدید همچنان نیاز داریم که از داده‌های ۶۰ ساعت گذشته برای پیش‌بینی استفاده کنیم.

```

class LSTMStrategy(bt.Strategy):
    params = (
        ("look_back", 60),
    )

    def __init__(self):
        self.lstm_predictions = []
        self.X = X
        self.counter = 0
        self.unit = 10

    def next(self):
        global pf, lstm_predictions, date
        if self.counter == 0:
            predicted_price = predict_price_lstm(X)
            self.counter += 1
            lstm_predictions.append(inverse_min_max_scaling(predicted_price, x_min['Close'], x_max['Close']))
        if pf.cash > self.unit * inverse_min_max_scaling(X[-1][3], x_min['Close'], x_max['Close']) and predicted_price < X[-1][3]:
            self.buy(unit = self.unit)
            pf.volume += self.unit
            pf.cash -= self.unit * inverse_min_max_scaling(X[-1][3], x_min['Close'], x_max['Close'])
            pf.buy.append(date)
        elif pf.volume > self.unit and predicted_price > X[-1][3]:
            self.sell(unit = self.unit)
            pf.sell.append(date)
            pf.cash += self.unit * inverse_min_max_scaling(X[-1][3], x_min['Close'], x_max['Close'])
            pf.volume -= self.unit

```

در این استراتژی، های unit خرید و فروش به تعداد ۱۰ تایی در نظر گرفته شده‌اند. تغییراتی که نسبت به کلاس قبلی انجام شده شامل اضافه کردن Portfolio و unit و volume و cash می‌شود. به عبارت دیگر، مدیریت مجموع دارایی‌های شخص است. برای مثال، زمانی که خرید انجام می‌شود، حجم سهام (volume) اضافه می‌شود و پول (cash) شخص به اندازه سهام‌های خریداری شده کاهش می‌یابد. در هنگام فروش سهام نیز برعکس این اتفاق رخ می‌دهد. کلاس Portfolio به صورت زیر تعریف شده‌است:

```

class Portfolio():
    def __init__(self, cash):
        self.cash = cash
        self.volume = 0
        self.sell = []
        self.buy = []

```

در ابتدا و قبل از انجام پیش‌بینی‌ها و خرید و فروش‌ها، ابتدا یک Portfolio برای شخص مشخص می‌کنیم:

```
Portfolio(100000) = pf
```

با این خط، شخص به صورت پیش‌فرض ۱۰۰ هزار دلار دارد و حجم سهام‌های شخص نیز صفر می‌باشد و او هیچ خرید و فروش سهامی را هم انجام نداده‌است.

باز هم مانند استراتژی قبلی، داده‌های پیشین را با توجه به تاریخ شروع و اتمام آن داده‌ها می‌گیریم. سپس تابع download stock data نیز همانطور که در مدل گذشته به آن اشاره شده پیاده‌سازی شده است. حال داده ۳۷۷ ساعت را به عنوان داده تست داریم. حال قبل از بررسی و استفاده از این ۳۷۷ داده، ابتدا استراتژی را cerebro می‌دهیم و دارایی اولیه را تعیین می‌کنیم:

سپس با استفاده از استفاده از مدل خود و با استفاده از داده تست و استراتژی تعیین شده پیش‌بینی‌های خود را انجام می‌دهیم و براساس این پیش‌بینی‌ها خرید و فروش‌های سهام‌ها را در های unit ۱۰ تایی انجام می‌دهیم.

پس از اتمام این مرحله، تمامی پیش‌بینی‌ها و خرید و فروش‌ها و زمان آن‌ها و همچنین قیمت‌های close حداقلی و حداکثری برای هر ساعت ذخیره شده‌است تا در مرحله ارزیابی از آن‌ها استفاده کنیم.

```

pf = Portfolio(100000)
import IPython.display as display
import seaborn as sns

sns.set() # Set the Seaborn style

actual_values = []
lstm_predictions = []
historical_data.head()
test_counter = 0
for i in range(len(X_test)-1):
    date = historical_data.index[test_counter]
    cerebro.run()
    display.clear_output(wait=True)
    x_vals = range(i+1)

    test_counter += 1
    x = X_test[test_counter]

    actual_values.append(inverse_min_max_scaling(X[-1][3], x_min['Close'], x_max['Close']))
    # Create a new plot
    plt.plot(x_vals, lstm_predictions,color = 'red', label='Real-Time Predictions')

    plt.plot(x_vals, actual_values,color = 'green', label='Real-Time Values')

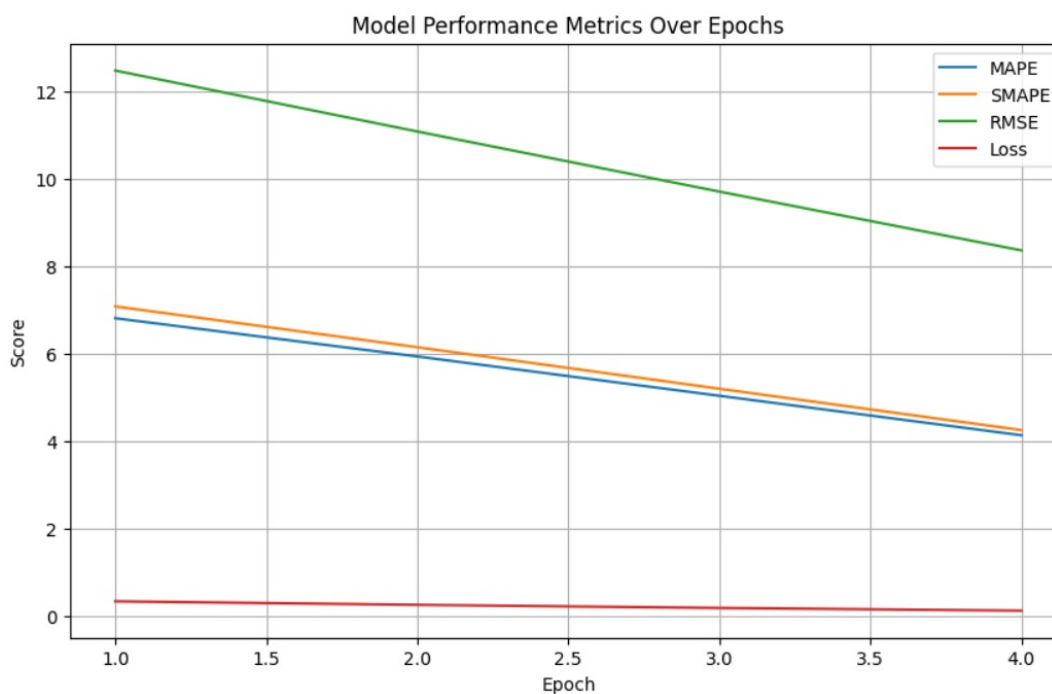
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.title('Real-Time Prediction Visualization')
    plt.legend()
    # Show the plot
    plt.tight_layout()
    plt.show(block=False)

```


۱۱.۲ ریسک به پاداش

۳ نتایج

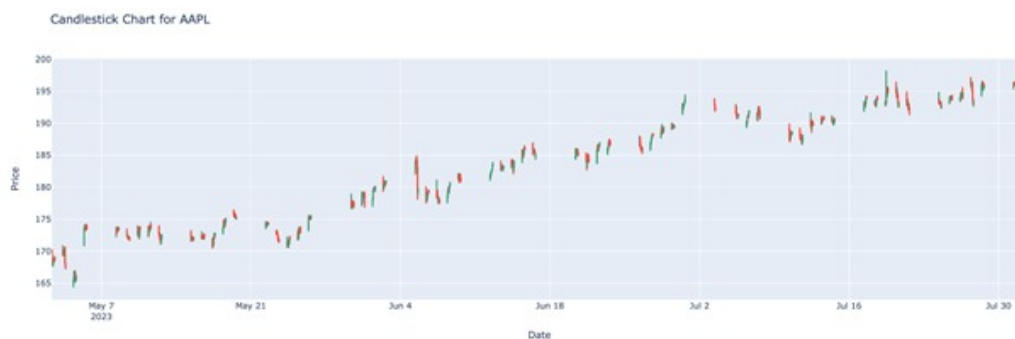
نمودار معیارها



۱.۳ نمودارهای شمعدانی

برای تجسم داده‌های مالی، ما از نمودارهای شمعدانی (Candlestick) استفاده کردیم. این نمودارها جزئیاتی از قیمت‌های باز و بسته شدن یک سهم در فواصل زمانی منظم نمایش می‌دهند. در این نمودارها، بدنه نشان‌دهنده قیمت باز و بسته شدن است. بدنه با رنگ قرمز و توخالی نمایانگر قیمت بسته شدن پایین و با رنگ سبز توخالی نمایانگر قیمت بسته شدن بالا است. همچنین، فتیله‌ها نشان‌دهنده بیشترین و کمترین قیمت‌های دارایی در طول زمان مشخص شده است.

نمودار اول مربوط به داده‌های ساعتی سهام شرکت اپل برای دوره آزمایش است:



با توجه به تفاوت‌های زمانی در داده‌های ساعتی، این نمودار شکل زیادی ندارد.

نمودار دوم برای داده‌های روزانه سهام اپل رسم شده است:

AAPL Candlestick Chart

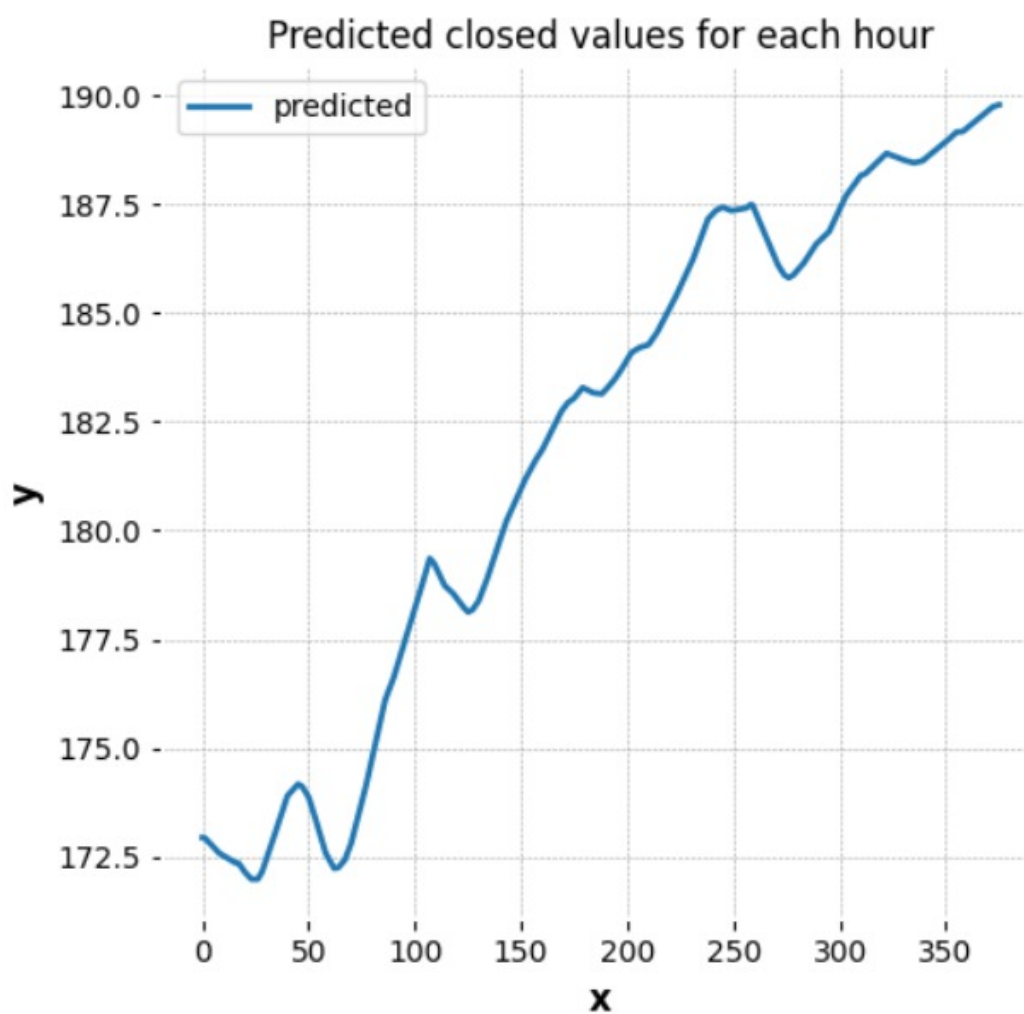


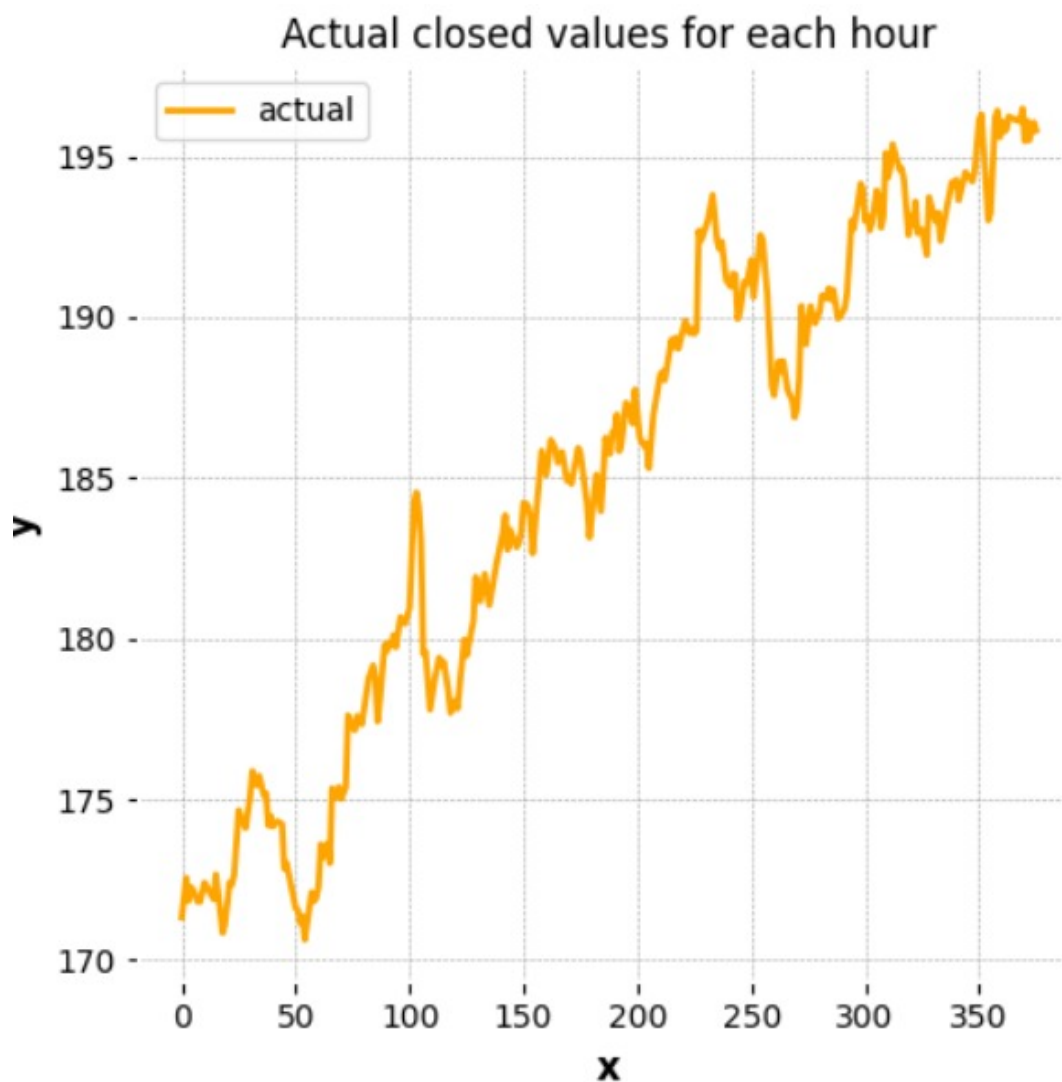
با استفاده از این نمودار روزانه، تفاوت‌های بیشتری در داده‌ها به وضوح قابل مشاهده هستند. برای رسم این نمودارها، از کتابخانه‌های Plotly و mplfinance در Python استفاده شده است.

۲.۳ مقایسه پیش‌بینی‌های مدل با مقادیر واقعی

در ادامه، ما پیش‌بینی‌های مدل را با مقادیر واقعی مقایسه می‌کنیم. این مقایسه نشان‌دهنده دقت و عملکرد مدل ما در پیش‌بینی قیمت‌ها است.

برای این منظور، نموداری را رسم کرده‌ایم. این نمودار نشان‌دهنده تطابق بین پیش‌بینی‌های مدل و واقعیت مقادیر بسته شدن سهام به ازای هر ساعت می‌باشد. همانطور که در نمودار مشاهده می‌شود، پیش‌بینی‌های مدل با مقادیر واقعی بسیار مشابه هستند که نشان‌دهنده دقت و عملکرد موفق مدل ما می‌باشد.





۳.۳ بررسی عملکرد Portfolio

در این قسمت، ما عملکرد Portfolio را بررسی می‌کنیم. این بررسی شامل میزان نقدینگی در حساب و ارزش سهامی است که در حال حاضر داریم. در نهایت، مجموع این دو به عنوان دارایی کل ما نشان داده می‌شود. این بررسی نشان‌دهنده سود یا زیان کلی ما نسبت به دارایی اولیه است:

```
print("Portfolio's cash: ", pf.cash)

Portfolio's cash: 1755.7154846191406

stock_value = pf.volume*inverse_min_max_scaling(actual_values[-1], x_min['Close'], x_max['Close'])
print("Portfolio's stocks value: ", pf.volume*inverse_min_max_scaling(actual_values[-1], x_min['Close'], x_max['Close']))

Portfolio's stocks value: 3455295.4696408007

print("Portfolio's total value: ", pf.cash + stock_value)

Portfolio's total value: 3457051.18512542
```

در ادامه نیز مشاهده می‌شود که در چه ساعاتی خرید اتفاق افتاده و در چه زمان‌هایی با استفاده از استراتژی برخی از سهام‌ها فروخته شده‌اند:

Stock is bought in these dates

2023-05-05 13:30:00-04:00
2023-05-05 14:30:00-04:00
2023-05-05 15:30:00-04:00
2023-05-08 09:30:00-04:00
2023-05-08 10:30:00-04:00
2023-05-08 11:30:00-04:00
2023-05-08 12:30:00-04:00
2023-05-08 13:30:00-04:00
2023-05-08 14:30:00-04:00
2023-05-08 15:30:00-04:00
2023-05-09 09:30:00-04:00
2023-05-09 10:30:00-04:00
2023-05-09 11:30:00-04:00
2023-05-09 12:30:00-04:00
2023-05-12 15:30:00-04:00
2023-05-15 11:30:00-04:00
2023-05-15 13:30:00-04:00
2023-05-15 14:30:00-04:00
2023-05-15 15:30:00-04:00
2023-05-16 09:30:00-04:00
2023-05-16 10:30:00-04:00
2023-05-16 11:30:00-04:00
2023-05-16 12:30:00-04:00
2023-05-16 13:30:00-04:00
2023-05-16 14:30:00-04:00
2023-05-16 15:30:00-04:00
2023-05-17 09:30:00-04:00
2023-05-17 10:30:00-04:00
2023-05-17 11:30:00-04:00
2023-05-17 12:30:00-04:00
2023-05-17 13:30:00-04:00
2023-05-17 14:30:00-04:00
2023-05-17 15:30:00-04:00
2023-05-18 09:30:00-04:00

Stock is sold in these dates

2023-05-09 13:30:00-04:00
2023-05-09 14:30:00-04:00
2023-05-09 15:30:00-04:00
2023-05-10 09:30:00-04:00
2023-05-10 10:30:00-04:00
2023-05-10 11:30:00-04:00
2023-05-10 12:30:00-04:00
2023-05-10 13:30:00-04:00
2023-05-10 14:30:00-04:00
2023-05-10 15:30:00-04:00
2023-05-11 09:30:00-04:00
2023-05-11 10:30:00-04:00
2023-05-11 11:30:00-04:00
2023-05-15 09:30:00-04:00
2023-05-15 12:30:00-04:00
2023-05-18 12:30:00-04:00
2023-05-23 11:30:00-04:00
2023-05-23 12:30:00-04:00
2023-05-23 13:30:00-04:00
2023-05-23 14:30:00-04:00
2023-05-23 15:30:00-04:00
2023-05-24 09:30:00-04:00
2023-05-24 10:30:00-04:00
2023-05-24 11:30:00-04:00
2023-05-24 12:30:00-04:00
2023-05-24 13:30:00-04:00
2023-05-24 14:30:00-04:00
2023-05-24 15:30:00-04:00
2023-05-25 09:30:00-04:00
2023-05-25 10:30:00-04:00
2023-05-25 11:30:00-04:00
2023-05-25 12:30:00-04:00
2023-05-25 13:30:00-04:00
2023-05-25 14:30:00-04:00
2023-05-25 15:30:00-04:00
2023-05-26 09:30:00-04:00
2023-05-26 10:30:00-04:00
2023-06-02 10:30:00-04:00

