دانشگاه صنعتی شریف
تابستان ۱۴۰۲

گزارش پایانی آزمایشگاه سخت‌افزار

دیبا مسیحی ۹۷۱۱۰۲۷۴
محمدرضا دویران ۹۸۱۰۱۵۶۶
آرین یزدان‌پرست ۹۸۱۱۰۰۹۵

چکیده

در این گزارش، ما یک روند گام به گام برای پیاده‌سازی استراتژی‌های معاملاتی مالی با استفاده از Back Test و ادغام پلتفرم مالی Alpaca و مدل‌های شبکه‌های عصبی LSTM ارائه می‌دهیم. تمامی کدهای استفاده شده در طول این پروژه در این لینک موجود می‌باشد.

۱ مقدمه

پیش‌بینی قیمت‌های آتی سهام به کمک اطلاعات تاریخی درباره‌ی حرکت‌های قیمت و حجم معاملات سهام انجام می‌شود. از طریق تحلیل الگوها و روندها در داده‌های گذشته، از جمله رابطه بین تغییرات قیمت و حجم معاملات، مدل‌های پیش‌بینی برای تخمین حرکات آینده قیمت توسعه داده می‌شود. این اطلاعات برای سرمایه‌گذاران و معامله‌گرانی که به تصمیم‌گیری آگاهانه در مورد خرید یا فروش سهام و شناسایی فرصت‌های سرمایه‌گذاری علاقه‌مند هستند، بسیار ارزشمند هستند. در این پروژه، از مدل LSTM مبتنی بر TinyML برای پیش‌بینی قیمت‌های سهام در آینده استفاده شده است. با آموزش این مدل با استفاده از داده‌های تاریخی مربوط به قیمت و حجم معاملات، از الگوها و ارتباطات موجود در داده‌ها برای پیش‌بینی حرکات آتی قیمت‌ها بهره‌برداری شده است. Pi Raspberry به عنوان پلتفرمی برای استقرار مدل‌های آموزش‌دیده عمل می‌کند و امکان پیش‌بینی در زمان واقعی و تصمیم‌گیری‌های معاملاتی خودکار را فراهم می‌آورد. هدف اصلی این پروژه، ایجاد یک راه‌حل فشرده و با مصرف مناسب با بهینه‌سازی مصرف CPU و حافظه است. این ابزار می‌تواند به معامله‌گران به منظور تصمیم‌گیری مبتنی بر داده و افزایش بازدهی در بازار پویای سهام کمک کند. این پروژه به اجرای موفقیت‌آمیز مدل LSTM بر روی یک Pi Raspberry با استفاده از TinyML و اتصال به پلتفرم معاملات سهام پرداخته است. این کار با هدف تسهیل تصمیم‌گیری‌های معاملاتی بر اساس پیش‌بینی قیمت‌های آتی سهام انجام شده است.

۲ مراحل پروژه

۱.۲ نصب نیازمندی‌ها

در ابتدا، نیازمندی‌های پروژه را بر روی Pi Raspberry نصب می‌کنیم. این نیازمندی‌ها شامل کتابخانه‌های Tensorflow و yfinance برای استخراج داده‌های مالی می‌باشند. همچنین، کتابخانه‌های pandas و numpy نیز به صورت پیش‌فرض در سیستم عامل موجود هستند. تمام کتابخانه‌های مورد استفاده در این پروژه به شرح زیر می‌باشند:

```python
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 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 os
import psutil
import time
import IPython.display as display
import backtrader as bt
import psutil
import seaborn as sns
from datetime import datetime, timedelta
import time
import mplfinance as mpf
import plotly.graph_objs as go
```

شکل ۱: نیازمندی‌ها

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 نظیراولین 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.

mplementation

```python
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

```python
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
```

شکل ۲: استخراج و پیش پردازش داده

۳.۲ ساخت مدل LSTM

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

```python
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
```

شکل ۳: ساخت مدل

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:

```python
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]
```

شکل ۴: پیش‌بینی قیمت

```
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
```

شکل ۵: پیش‌بینی قیمت‌ها

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
```

شکل ۶: محاسبه مقادیر واقعی

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

```python
import numpy as np

def calculate_smape(actual, predicted):

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

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

    return smape
```

شکل ۷: SMAPE

- 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

```python
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
```

شکل ۸: MAPE

- RMSE که بر اساس رابطه زیر محاسبه می‌شود:

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

RSME

```python
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
```

شکل ۹: RMSE

۷.۲ بررسی سهام‌های مختلف

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

```python
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
```

شکل ۱۰: دانلود اطلاعات سهام اپل

۸.۲ کوانتایز کردن مدل

در این قسمت ابتدا کتابخانه‌های مد نظر را نصب می‌کنیم.

```
!pip install tensorflow_model_optimization
```

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
```

```
tflite_model = converter.convert()
```

شکل ۱۱: کوانتیزاسیون

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]
```

شکل ۱۲: پیش‌بینی با مدل کوانتیزه شده

مدل اول که بر مبنای شبکه‌های LSTM است، 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 کوانتایزه 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 کوانتایزه برای سناریوهای تصمیم‌گیری سریع که به بهره‌وری منابع حیاتی است، عالی عمل می‌کند.

در مقابل، مدل کوانتایزه شده که ساده‌تر و کم‌منابع‌تر است، همچنان دقت قابل قبولی در پیش‌بینی دارد. علاوه بر این، مدل کوانتایزه 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
_________________________________________________________________
```

شکل ۱۳: مدل اولیه

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': {}}]
```

شکل ۱۴: ورودی کوانتیزه شده

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': {}}]
```

شکل ۱۵: خروجی کوانتیزه شده

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]
```

شکل ۱۶: محاسبه تعداد پارامتر

```
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
```

شکل ۱۷: تعداد پارامترهای مدل‌ها

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())
```

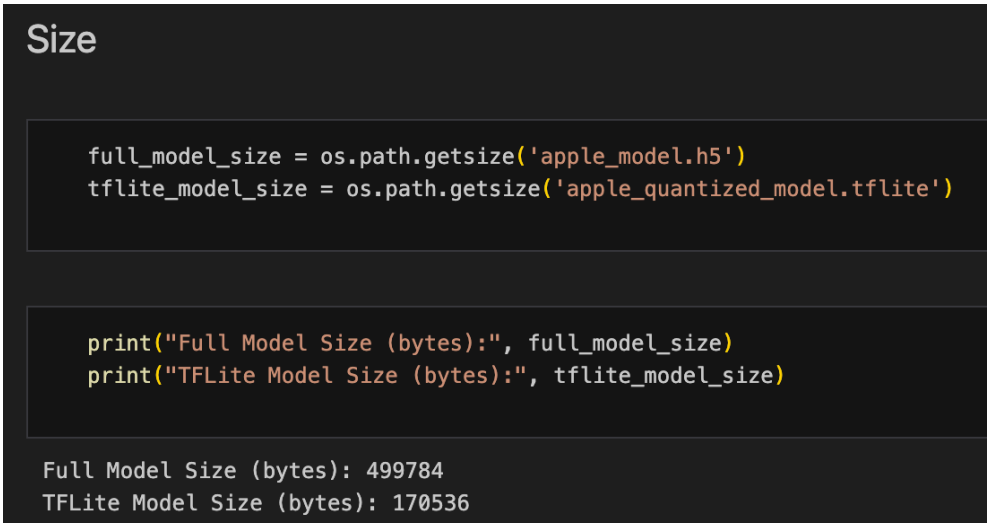
```
print("Full Model Number of Layers:", full_model_num_layers)
print("TFLite Model Number of Layers:", tflite_model_num_layers)
```

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

شکل ۱۸: تعداد لایه‌ها در مدل‌ها

۳.۹.۲ سایز مدل

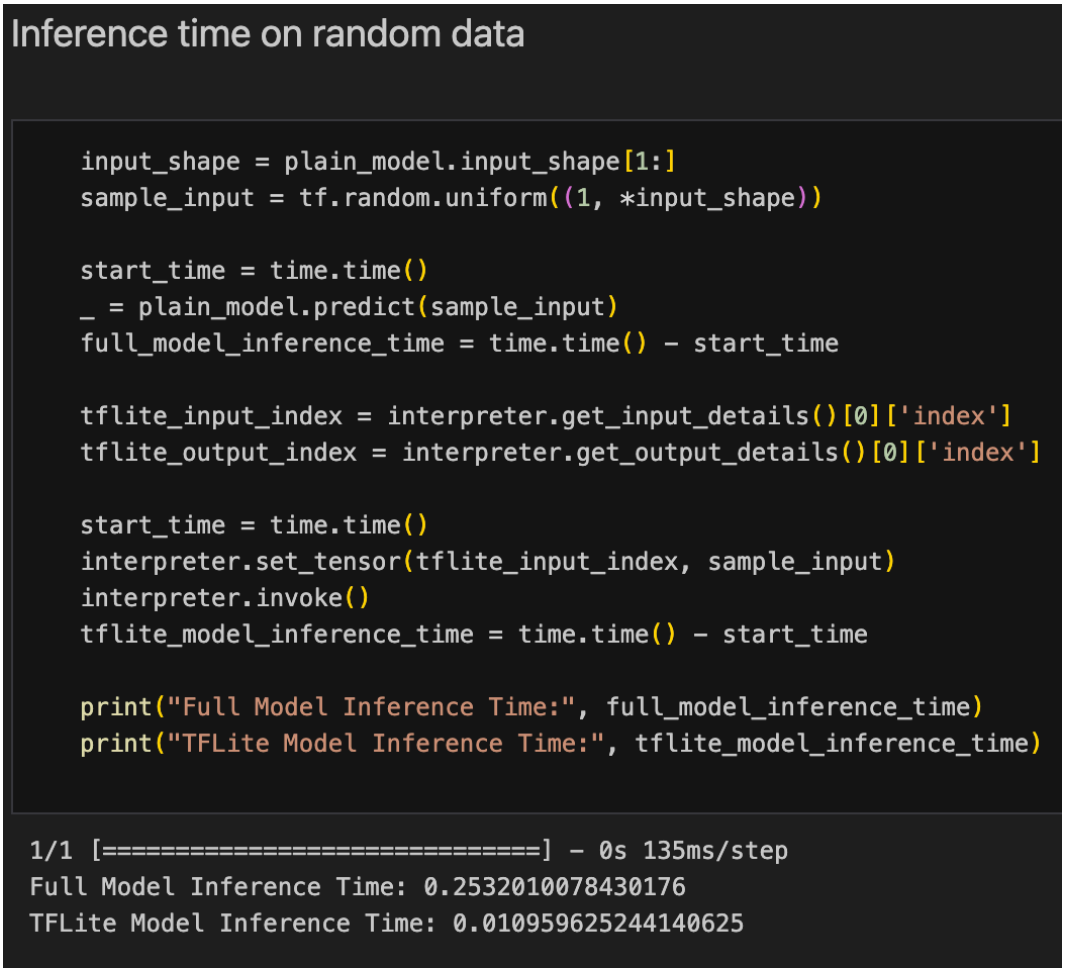
اگر سایز مدل‌ها را نیز بررسی کنیم مشاهده می‌کنیم که سایز مدل اولیه سه برابر مدل کوانتیزه شده است:



شکل ۱۹: سایز مدل‌ها

۴.۹.۲ سرعت پیش‌بینی

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



شکل ۲۰: سرعت استنتاج

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 دقیق تر 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:

```python
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)
```

شکل ۲۱: دانلود اطلاعات سهام اپل

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

```python
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
```

```python
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
```

شکل ۲۲: محاسبه زمان پیش‌بینی مدل‌ها

```python
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
```

شکل ۲۳: مقایسه زمان پیش‌بینی مدل‌ها

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)
```

شکل ۲۴: پیش‌بینی وضعیت سهام‌های اپل در آینده توسط مدل اولیه

```
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)
```

شکل ۲۵: پیش‌بینی وضعیت سهام‌های اپل در آینده توسط مدل کوانتیزه شده

```
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
```

شکل ۲۶: مقایسه عملکرد مدل‌ها

می‌بینیم که عملکرد هردو مدل با توجه به متریک‌های مختلف تا دو رقم اعشار یکسان است! پس عملا 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.

۱۰.۲ تست و ارزیابی عملکرد

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)
```

شکل ۲۷: پیش‌بینی توسط مدل کوانتیزه شده

برای انجام بازآزمایی، 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 بازآزمایی با اجرای نمونه Cerebro انجام می‌شود.

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

شکل ۲۸: استفاده از cerebro

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.

```
risk_reward = 1.001
bought_flag = False
bought_price = 0

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,bought_flag, test_counter, risk_reward, bought_price, X
      if self.counter == 0:
          now_price = inverse_min_max_scaling(X[-1][3], x_min['Close'], x_max['Close'])
          predicted_price = predict_price_lstm(X)
          lstm_predictions.append(inverse_min_max_scaling(predicted_price, x_min['Close'], x_max['Close']))
          if not bought_flag:
            for i in range(10):
              predicted_price = predict_price_lstm(X)
              X = X[1:,:]
              X = np.append(X, [[predicted_price, predicted_price, predicted_price, predicted_price, predicted_price]],axis = 0)
          predicted_price = inverse_min_max_scaling(predicted_price, x_min['Close'], x_max['Close'])
          if not bought_flag and pf.cash > self.unit * now_price and predicted_price * self.unit > risk_reward * self.unit * now_price:
              self.buy(unit = self.unit)
              pf.volume += self.unit
              pf.cash -= self.unit * now_price
              bought_flag = True
              pf.buy.append(date)
```

شکل ۲۹: استراتژی تعیین شده

```
            bought_price = now_price
        elif (test_counter == 20 and bought_flag) or bought_price < risk_reward * now_price:
            self.sell(unit = self.unit)
            pf.sell.append(date)
            bought_flag = False
            pf.cash += self.unit * now_price
            pf.volume -= self.unit
            test_counter = 0
        else:
          test_counter += 1
        self.counter += 1
```

شکل ۳۰: استراتژی تعیین شده

در این استراتژی، هایunit خرید و فروش به تعداد ۱۰ تایی در نظر گرفته‌شده‌اند. تغییراتی که نسبت به کلاس قبلی انجام شده شامل اضافه کردن Portfolio و unit و volume و cash می‌شود. به عبارت دیگر، Portfolio 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 = []
```

شکل ۳۱: Portfolio

در ابتدا و قبل از انجام پیش‌بینی‌ها و خریدوفروش‌ها، 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.

داده‌های ورودی به صورت بی‌درنگ هستند. در هر دقیقه اطلاعات مربوط به ۶۰ دقیقه‌ی اخیر دریافت شده و از این داده‌ها استفاده می‌شود. به طور کلی استراتژی خود را به صورت متوالی برای ۱۰۰ دقیقه اجرا می‌کنیم به طوری که در انتهای هر دقیقه داده‌ها اپدیت شده و استراتژی به روی آن‌ها اجرا می‌گردد.

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 هایunit 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.

```python
pf = Portfolio(100000)
import IPython.display as display
import seaborn as sns
from datetime import datetime, timedelta
import time

sns.set()  # Set the Seaborn style

actual_values = []
lstm_predictions = []
historical_data.head()
test_counter = 0
for i in range(100):
  date = historical_data.index[test_counter]

  cerebro.run()
  time.sleep(60)
  historical_data = ticker.history(period="1d", interval="1m").tail(60)

  X_test, y_test= preprocess_data(historical_data)
  x_min = historical_data.min()
  x_max = historical_data.max()
  X = X_test[0]
```

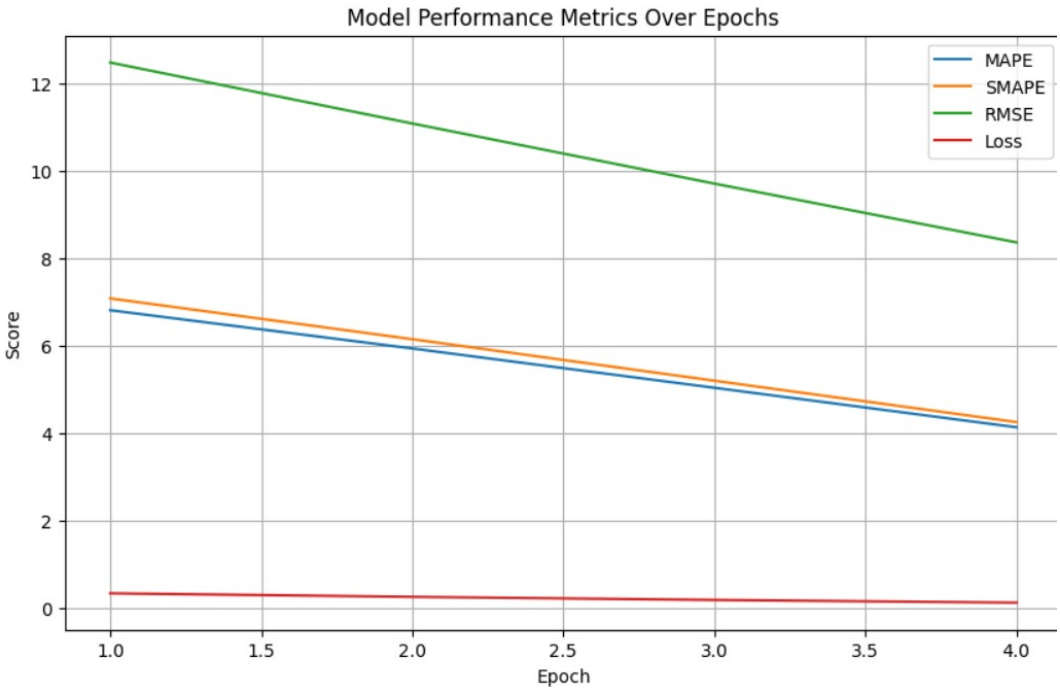
شکل ۳۲: پیش‌بینی

۱۱.۲ ریسک به پاداش

استراتژی ما بر اساس ضریب ریسک به پاداش بدست می‌آید به طوری که داده‌های مربوط به ۱۰ دقیقه‌ی آینده را پیشبینی کرده و سپس در صورتی که داده‌ی بدست آمده نسبت به قیمت فعلی برابر با ضریب ریسک به پاداش بود آنگاه خرید صورت می‌گیرد و پرچم bought برابر True می‌شود. همچنین قیمت خرید نیز در bought-price برای محاسبه زمان فروش ذخیره می‌گردد. این پرچم به معنای این است که خرید صورت گرفته است و صاحب سهام می‌تواند آن را بفروشد. فروش زمانی در دو زمان رخ می‌دهد. یک اینکه خواسته ما صورت بگیرد و قیمت سهام به اندازه ضریب ریسک به پاداش رشد کند یا مدت زمان ۱۰ دقیقه (که برابر پیشبینی سهام ما در آن زمان بود) زمان بگذرد.

۳ نتایج

نمودار معیارها



شکل ۳۳: معیارها

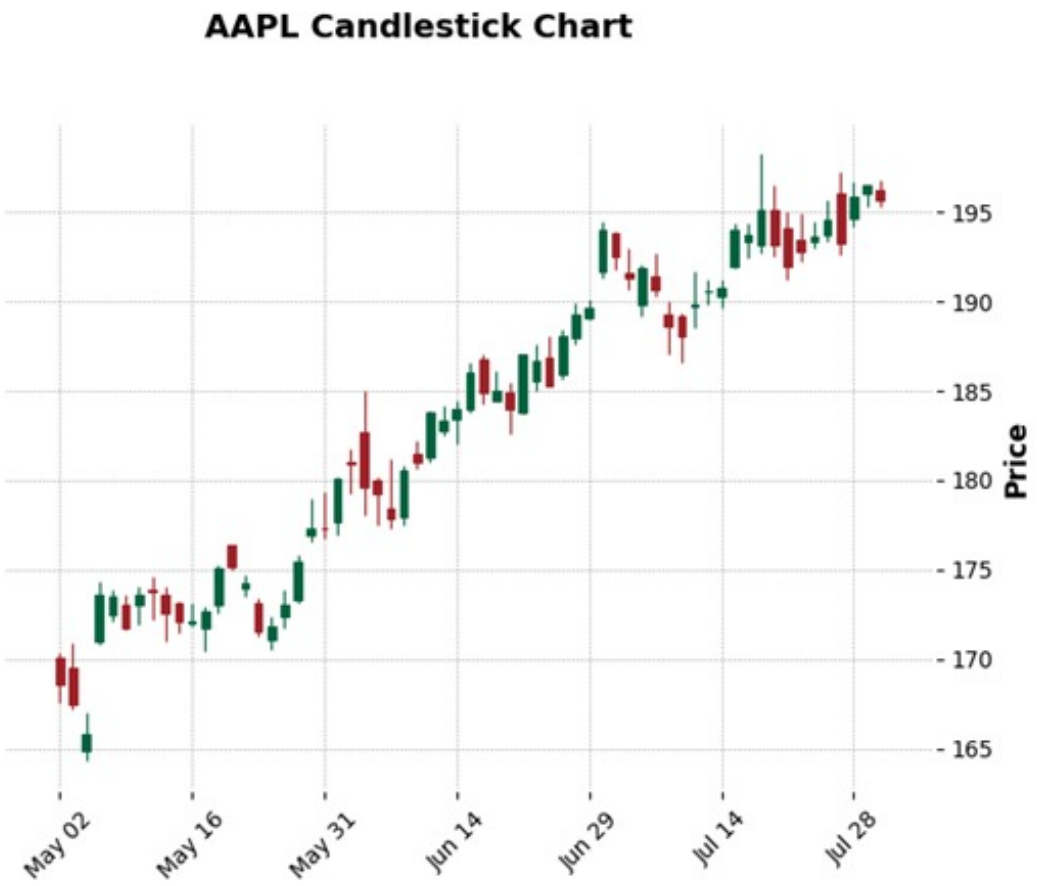
۱.۳ نمودارهای شمعدانی

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:

شکل ۳۴: نمودار شمعدانی براساس ساعت

با توجه به تفاوت‌های زمانی در داده‌های ساعتی، این نمودار شکل‌زیادی ندارد.
نمودار دوم برای داده‌های روزانه سهام اپل رسم شده است:



شکل ۳۵: نمودار شمعدانی براساس روز

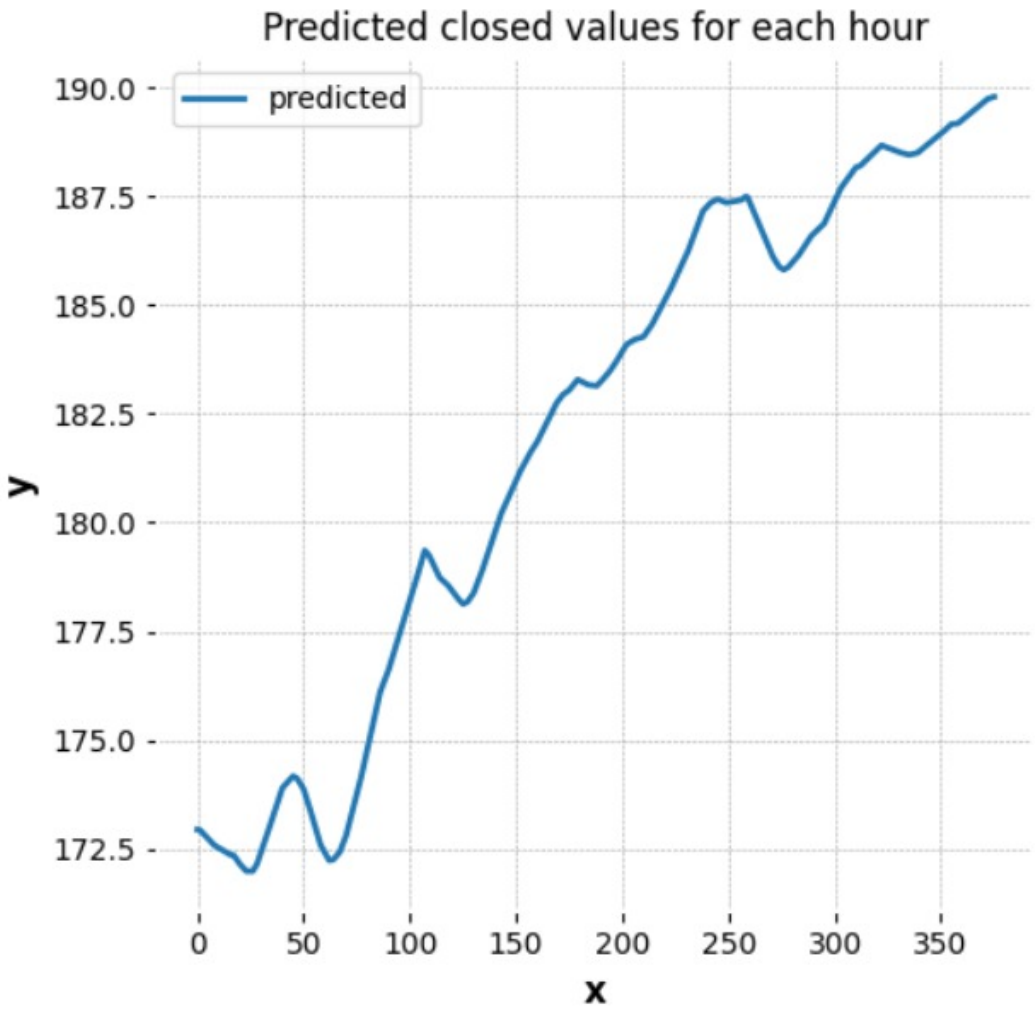
با استفاده از این نمودار روزانه، تفاوت‌های بیشتری در داده‌ها به وضوح قابل مشاهده هستند.

برای رسم این نمودارها، از کتابخانه‌های Plotly و mplfinance در Python استفاده شده است.

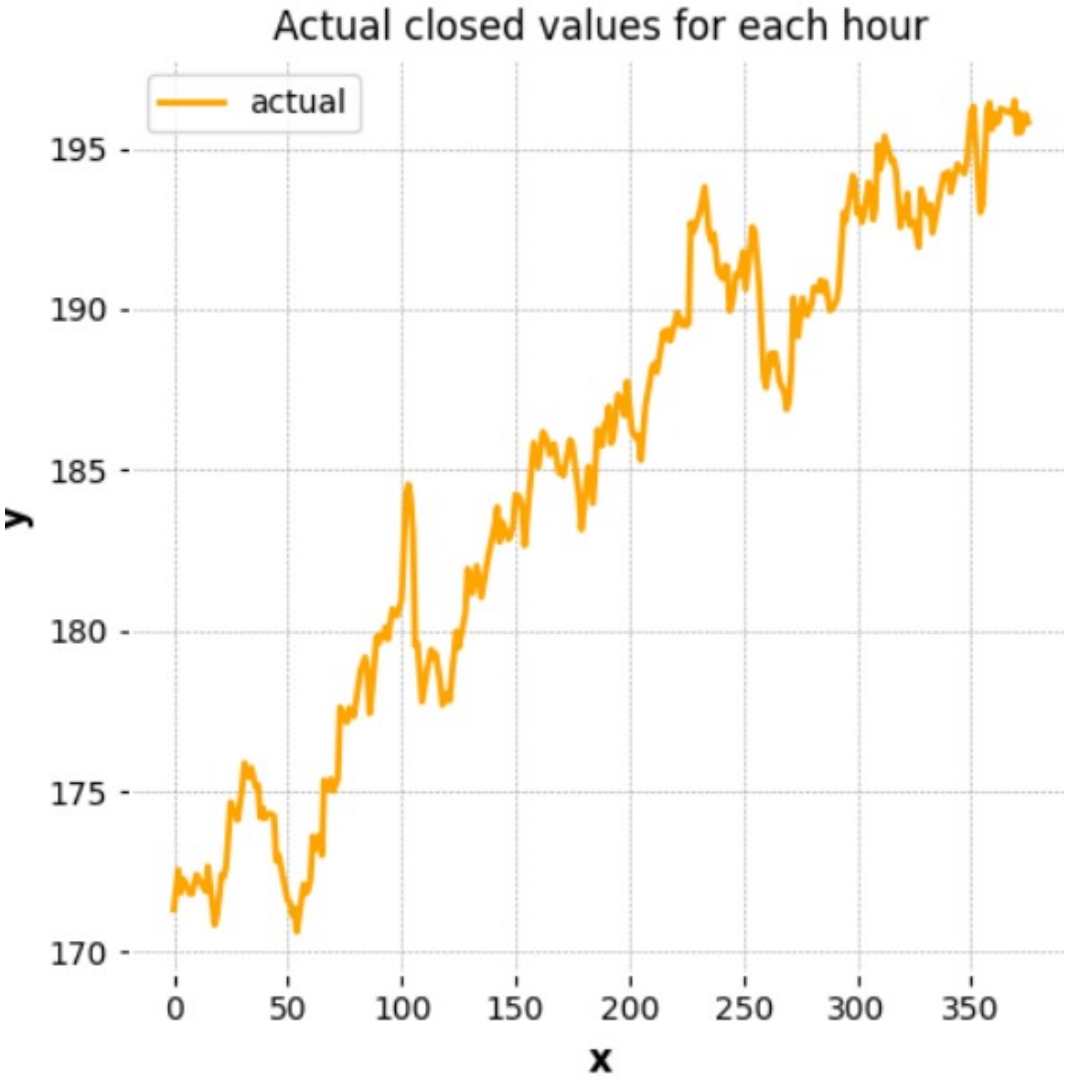
۲.۳ مقایسه پیش‌بینی‌های مدل با مقادیر واقعی

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.



شکل ۳۶: نمودار سهام‌های پیش‌بینی شده



شکل ۳۷: نمودار سهام‌های واقعی

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 (دقت کنید این نتایج برای داده‌ها بی‌درنگ نیست و مربوط به ۳۷۷ داده‌ی دانلود شده در گذشته 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
```

شکل ۳۸: مقادیر نهایی پول و سهام

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
```

شکل ۳۹: زمان‌های خرید سهام

```
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
```

شکل ۴۰: زمان‌های فروش سهام

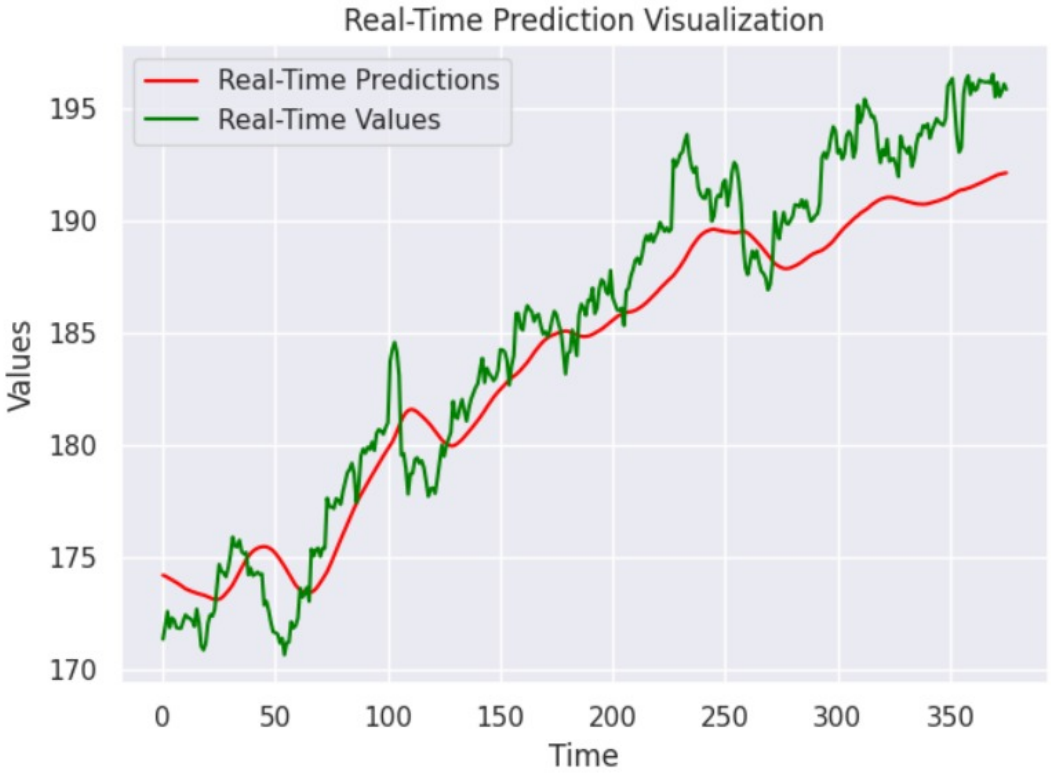
نتایج مربوط به داده‌های بی‌درنگ (اجرا برای ۵ دقیقه)

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

```
Portfolio's cash:  109451.98196411133
```

شکل ۴۱: پول مربوط به استفاده از داده‌های بی درنگ

۴.۳ نمایش time real پیشبینی



شکل ۴۲: نمودار real-time