使用LSTM预测股票价格：机器学习案例研究

在这篇文章中，我们将创建一个由简单的长短期记忆层和二元分类器组成的循环神经网络。该机器学习模型的目的是根据前30天的 close, open, high, low price和volume预测股票在第二天上涨或下跌。我们将其准确性与“基线模型”进行比较，“基线模型”总是选择测试集中最常见的值（如果不查看股票价格模式，则可获得最高精度）。

假设如果存在股票价格模式，那么需要一个非常复杂的神经网络来学习它们，因此优于我们的基线模型。

机器学习模型

在进行结果之前，这些是我采取的步骤：

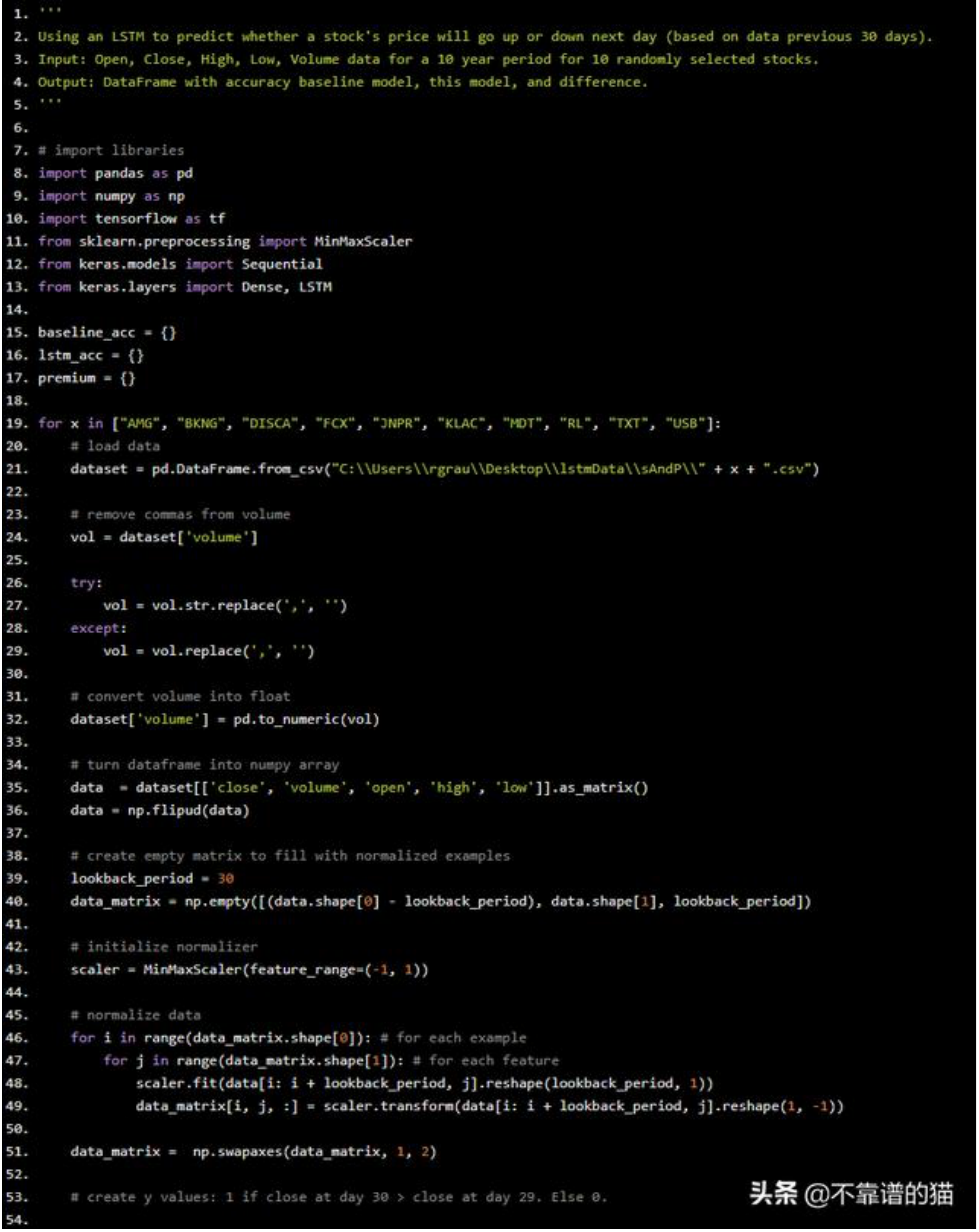
1. 从标准普尔500指数中随机挑选10只股票。
2. 获取特征：过去10年中每种股票的Open, Close, High, Low 和 Volum数据（来自纳斯达克网站）。
3. 对于每个股票，对于每个特征，使用特征数据创建30天（第1天，……，第30天，第2天，……，第31天等）的数组。
4. 创建一个由一个LSTM层（32个单元）和一个sigmoid神经元（模型

1）组成的神经网络，另一个由两个LSTM层（32和8个单元）和一个sigmoid神经元（模型2）组成。
5. 在训练数据上训练机器学习模型。
6. 在测试数据上测试机器学习模型。

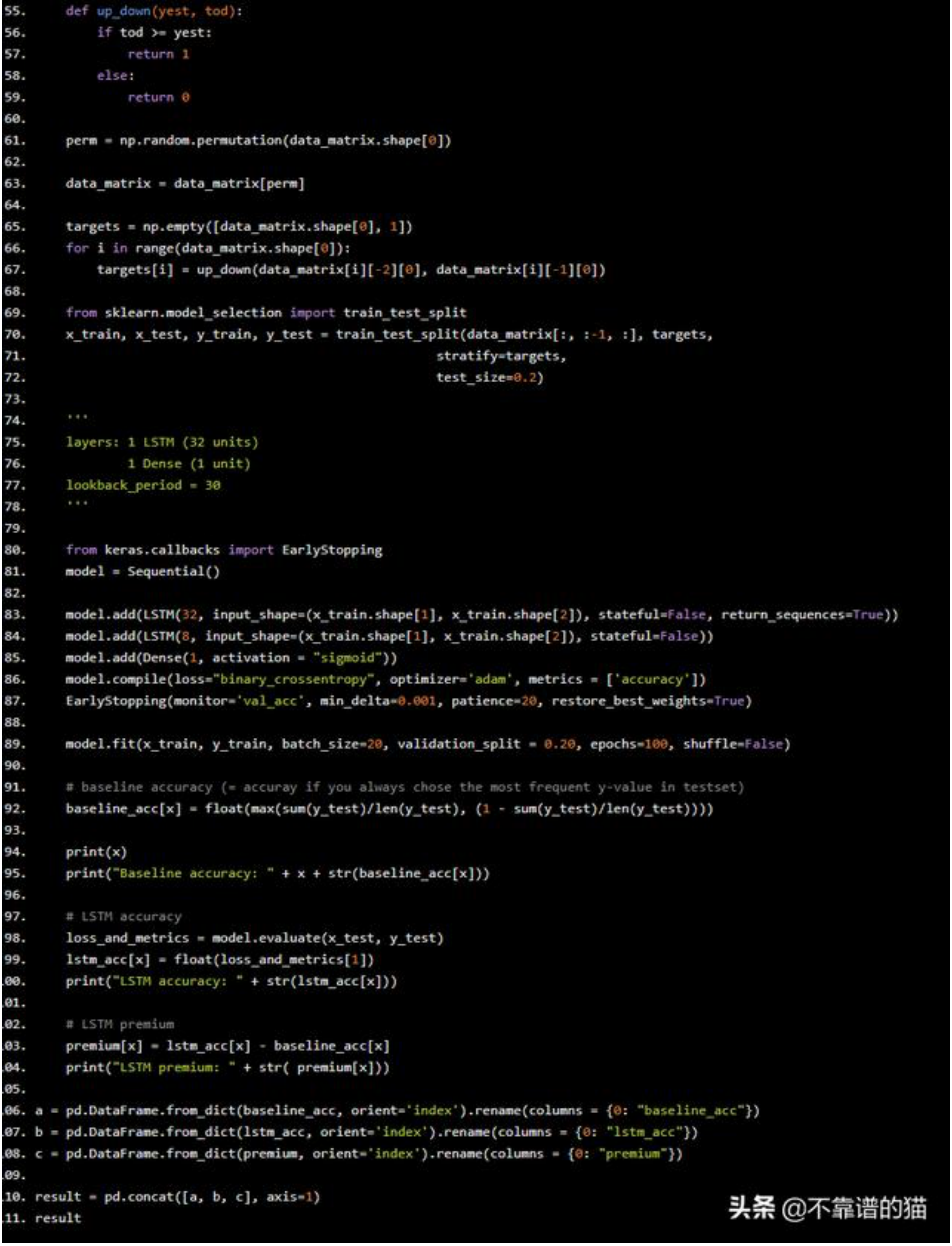
Python完整代码：

```
'''
Using an LSTM to predict whether a stock's price will go up or down next
day (based on data previous 30 days).
Input: Open, Close, High, Low, Volume data for a 10 year period for 10
randomly selected stocks.
Output: DataFrame with accuracy baseline model, this model, and
difference.
'''
# import libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM
baseline_acc = {}
lstm_acc = {}
premium = {}
for x in ["AMG", "BKNG", "DISCA", "FCX", "JNPR", "KLAC", "MDT", "RL",
"TXT", "USB"]:
 # load data
 dataset = pd.DataFrame.from_csv("C:\Users\rgrau\Desktop\lstmData\sAndP\"
+ x + ".csv")
 # remove commas from volume
 vol = dataset['volume']

 try:
 vol = vol.str.replace(',', '')
 except:
 vol = vol.replace(',', '')

 # convert volume into float
 dataset['volume'] = pd.to_numeric(vol)
 # turn dataframe into numpy array
 data = dataset[['close', 'volume', 'open', 'high', 'low']].as_matrix()
 data = np.flipud(data)
 # create empty matrix to fill with normalized examples
 lookback_period = 30
 data_matrix = np.empty([(data.shape[0] - lookback_period), data.shape[1],
lookback_period])
 # initialize normalizer
 scaler = MinMaxScaler(feature_range=(-1, 1))
 # normalize data
 for i in range(data_matrix.shape[0]): # for each example
 for j in range(data_matrix.shape[1]): # for each feature
 scaler.fit(data[i: i + lookback_period, j].reshape(lookback_period, 1))
 data_matrix[i, j, :] = scaler.transform(data[i: i + lookback_period,
j].reshape(1, -1))
 data_matrix = np.swapaxes(data_matrix, 1, 2)
 # create y values: 1 if close at day 30 > close at day 29. Else 0.
 def up_down(yest, tod):
 if tod >= yest:
 return 1
 else:
 return 0
 perm = np.random.permutation(data_matrix.shape[0])
 data_matrix = data_matrix[perm]
 targets = np.empty([data_matrix.shape[0], 1])
 for i in range(data_matrix.shape[0]):
 targets[i] = up_down(data_matrix[i][-2][0], data_matrix[i][-1][0])

 from sklearn.model_selection import train_test_split
 x_train, x_test, y_train, y_test = train_test_split(data_matrix[:, :-1,
:], targets,
 stratify=targets,
 test_size=0.2)
 '''
 layers: 1 LSTM (32 units)
 1 Dense (1 unit)
 lookback_period = 30
 '''
 from keras.callbacks import EarlyStopping
 model = Sequential()
 model.add(LSTM(32, input_shape=(x_train.shape[1], x_train.shape[2]),
stateful=False, return_sequences=True))
 model.add(LSTM(8, input_shape=(x_train.shape[1], x_train.shape[2]),
stateful=False))
 model.add(Dense(1, activation = "sigmoid"))
 model.compile(loss="binary_crossentropy", optimizer='adam', metrics =
['accuracy'])
 EarlyStopping(monitor='val_acc', min_delta=0.001, patience=20,
restore_best_weights=True)
 model.fit(x_train, y_train, batch_size=20, validation_split = 0.20,
epochs=100, shuffle=False)
 # baseline accuracy (= accuray if you always chose the most frequent y-
value in testset)
```

```
 baseline_acc[x] = float(max(sum(y_test)/len(y_test), (1 -
sum(y_test)/len(y_test))))

 print(x)
 print("Baseline accuracy: " + x + str(baseline_acc[x]))
 # LSTM accuracy
 loss_and_metrics = model.evaluate(x_test, y_test)
 lstm_acc[x] = float(loss_and_metrics[1])
 print("LSTM accuracy: " + str(lstm_acc[x]))
 # LSTM premium
 premium[x] = lstm_acc[x] - baseline_acc[x]
 print("LSTM premium: " + str( premium[x]))
a = pd.DataFrame.from_dict(baseline_acc, orient='index').rename(columns =
{0: "baseline_acc"})
b = pd.DataFrame.from_dict(lstm_acc, orient='index').rename(columns = {0:
"lstm_acc"})
c = pd.DataFrame.from_dict(premium, orient='index').rename(columns = {0:
"premium"})
result = pd.concat([a, b, c], axis=1)
result
```



```python
'''
Using an LSTM to predict whether a stock's price will go up or down next day (based on data previous 30 days).
Input: Open, Close, High, Low, Volume data for a 10 year period for 10 randomly selected stocks.
Output: DataFrame with accuracy baseline model, this model, and difference.
'''

# import libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM

baseline_acc = {}
lstm_acc = {}
premium = {}

for x in ["AMG", "BKNG", "DISCA", "FCX", "JNPR", "KLAC", "MDT", "RL", "TXT", "USB"]:
    # load data
    dataset = pd.DataFrame.from_csv("C:\\Users\\rgrau\\Desktop\\lstmData\\sAndP\\" + x + ".csv")

    # remove commas from volume
    vol = dataset['volume']

    try:
        vol = vol.str.replace(',', '')
    except:
        vol = vol.replace(',', '')

    # convert volume into float
    dataset['volume'] = pd.to_numeric(vol)

    # turn dataframe into numpy array
    data  = dataset[['close', 'volume', 'open', 'high', 'low']].as_matrix()
    data = np.flipud(data)

    # create empty matrix to fill with normalized examples
    lookback_period = 30
    data_matrix = np.empty([(data.shape[0] - lookback_period), data.shape[1], lookback_period])

    # initialize normalizer
    scaler = MinMaxScaler(feature_range=(-1, 1))

    # normalize data
    for i in range(data_matrix.shape[0]): # for each example
        for j in range(data_matrix.shape[1]): # for each feature
            scaler.fit(data[i: i + lookback_period, j].reshape(lookback_period, 1))
            data_matrix[i, j, :] = scaler.transform(data[i: i + lookback_period, j].reshape(1, -1))

    data_matrix =  np.swapaxes(data_matrix, 1, 2)

    # create y values: 1 if close at day 30 > close at day 29. Else 0.

```

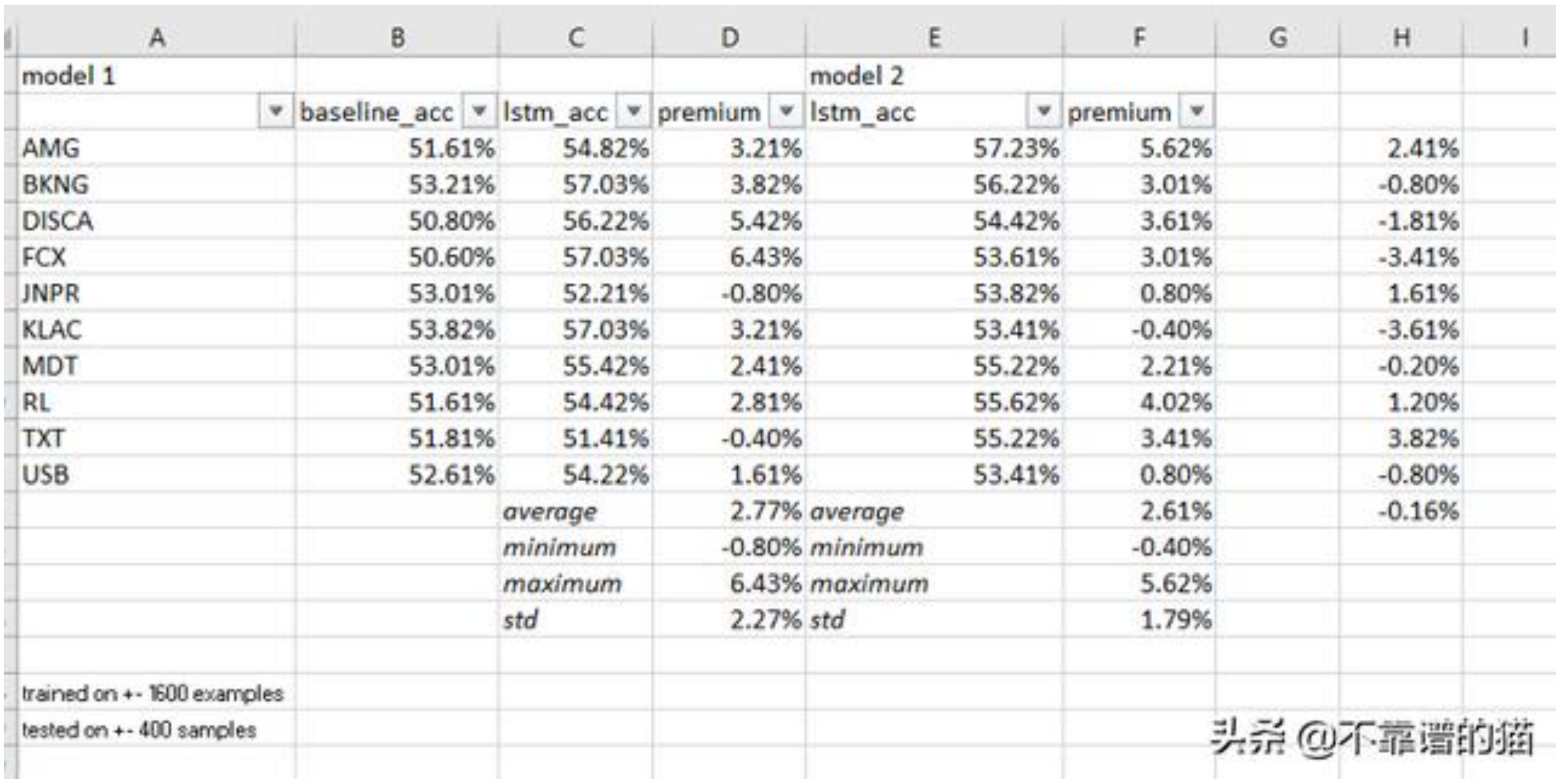


```
    def up_down(yest, tod):
        if tod >= yest:
            return 1
        else:
            return 0

    perm = np.random.permutation(data_matrix.shape[0])

    data_matrix = data_matrix[perm]

    targets = np.empty([data_matrix.shape[0], 1])
    for i in range(data_matrix.shape[0]):
        targets[i] = up_down(data_matrix[i][-2][0], data_matrix[i][-1][0])

    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(data_matrix[:, :-1, :], targets,
                                                        stratify=targets,
                                                        test_size=0.2)

    '''
    layers: 1 LSTM (32 units)
            1 Dense (1 unit)
    lookback_period = 30
    '''

    from keras.callbacks import EarlyStopping
    model = Sequential()

    model.add(LSTM(32, input_shape=(x_train.shape[1], x_train.shape[2]), stateful=False, return_sequences=True))
    model.add(LSTM(8, input_shape=(x_train.shape[1], x_train.shape[2]), stateful=False))
    model.add(Dense(1, activation = "sigmoid"))
    model.compile(loss="binary_crossentropy", optimizer='adam', metrics = ['accuracy'])
    EarlyStopping(monitor='val_acc', min_delta=0.001, patience=20, restore_best_weights=True)

    model.fit(x_train, y_train, batch_size=20, validation_split = 0.20, epochs=100, shuffle=False)

    # baseline accuracy (= accuray if you always chose the most frequent y-value in testset)
    baseline_acc[x] = float(max(sum(y_test)/len(y_test), (1 - sum(y_test)/len(y_test))))

    print(x)
    print("Baseline accuracy: " + x + str(baseline_acc[x]))

    # LSTM accuracy
    loss_and_metrics = model.evaluate(x_test, y_test)
    lstm_acc[x] = float(loss_and_metrics[1])
    print("LSTM accuracy: " + str(lstm_acc[x]))

    # LSTM premium
    premium[x] = lstm_acc[x] - baseline_acc[x]
    print("LSTM premium: " + str( premium[x]))

a = pd.DataFrame.from_dict(baseline_acc, orient='index').rename(columns = {0: "baseline_acc"})
b = pd.DataFrame.from_dict(lstm_acc, orient='index').rename(columns = {0: "lstm_acc"})
c = pd.DataFrame.from_dict(premium, orient='index').rename(columns = {0: "premium"})

result = pd.concat([a, b, c], axis=1)
result
```

结果

现在，模型是怎么样呢？比预期的要好得多。例如，模型2 在10次中超过基线模型9次。因此，对于10只股票中的9只，该模型更好地预测股票是

否会在第二天上涨或下跌，而不仅仅是总是选择最常见的数据。



A	B	C	D	E	F	G	H	I
model 1				model 2				
	baseline_acc	lstm_acc	premium	lstm_acc	premium			
AMG	51.61%	54.82%	3.21%	57.23%	5.62%		2.41%	
BKNG	53.21%	57.03%	3.82%	56.22%	3.01%		-0.80%	
DISCA	50.80%	56.22%	5.42%	54.42%	3.61%		-1.81%	
FCX	50.60%	57.03%	6.43%	53.61%	3.01%		-3.41%	
JNPR	53.01%	52.21%	-0.80%	53.82%	0.80%		1.61%	
KLAC	53.82%	57.03%	3.21%	53.41%	-0.40%		-3.61%	
MDT	53.01%	55.42%	2.41%	55.22%	2.21%		-0.20%	
RL	51.61%	54.42%	2.81%	55.62%	4.02%		1.20%	
TXT	51.81%	51.41%	-0.40%	55.22%	3.41%		3.82%	
USB	52.61%	54.22%	1.61%	53.41%	0.80%		-0.80%	
		average	2.77%	*average*	2.61%		-0.16%	
		minimum	-0.80%	*minimum*	-0.40%			
		maximum	6.43%	*maximum*	5.62%			
		std	2.27%	*std*	1.79%			
trained on +- 1600 examples								
tested on +- 400 samples								

这是否意味着你可以用这种模式赚钱呢? 也许不是。即使我们可以绝对肯定地预测某只股票明天会涨还是会跌，我们仍然不知道涨多少。这很重要。假设你猜对了57%的概率，但你猜对的时候只赚100美元，猜错的时候损失200美元。