```
import tensorflow as tf
import pandas as pd
from google.colab import drive
import matplotlib.pyplot as plt
import numpy as np
import sklearn
import seaborn as sns
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.wrappers.scikit_learn import KerasRegressor
import datetime
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import confusion_matrix, accuracy_score, plot_confusion_matrix, c
```

```
drive.mount('/content/gdrive/', force_remount = True)
```

```
Mounted at /content/gdrive/
```

```
stock_data = yf.download("MSFT", start="1990-01-01", end="2022-02-21")
```

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

```
stock_data
```

Date	Open	High	Low	Close	Adj Close	Volume

```
stock_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 8098 entries, 1990-01-02 to 2022-02-18
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Open       8098 non-null   float64
 1   High       8098 non-null   float64
 2   Low        8098 non-null   float64
 3   Close      8098 non-null   float64
 4   Adj Close  8098 non-null   float64
 5   Volume     8098 non-null   int64  
dtypes: float64(5), int64(1)
memory usage: 442.9 KB
```

```
scaled_data = np.array(stock_data["Close"].pct_change().dropna()).reshape(-1,1)


training_data_len = int(len(scaled_data) * 0.8)
train_data = scaled_data[0:training_data_len, :]



x_train = []
y_train = []

input_size = 6
for i in range(input_size, len(train_data)):
  x_train.append(train_data[i-input_size:i, 0])
  y_train.append(train_data[i, 0])


x_train, y_train = np.array(x_train), np.array(y_train)
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))


mu = y_train.mean()
sd = y_train.std()


RW_returns = np.random.normal(mu,sd,len(x_train))


df_in_sample = pd.DataFrame(RW_returns, columns=['RW_returns'])


df_in_sample['RW_returns'] = df_in_sample['RW_returns'].apply(lambda y: 1 if y>0 else
```

```
y= pd.DataFrame(y_train, columns=['y_train'])


y['y_train'] = y['y_train'].apply(lambda y: 1 if y>0 else 0)


y_train = np.array(y)


model = Sequential()
model.add(LSTM(x_train.shape[1], return_sequences=True, input_shape=(x_train.shape[1],
#Examples
model.add(LSTM(120, return_sequences=False))

model.add(Dense(1, activation = 'sigmoid'))




model.compile(optimizer='adam', loss='binary_crossentropy', metrics = ['AUC'])
model.fit(x_train, y_train, epochs=10)
```

```
Epoch 1/10
203/203 [==============================] - 11s 17ms/step - loss: 0.6933 - auc: 0
Epoch 2/10
203/203 [==============================] - 3s 17ms/step - loss: 0.6932 - auc: 0.
Epoch 3/10
203/203 [==============================] - 5s 26ms/step - loss: 0.6932 - auc: 0.
Epoch 4/10
203/203 [==============================] - 4s 18ms/step - loss: 0.6932 - auc: 0.
Epoch 5/10
203/203 [==============================] - 3s 17ms/step - loss: 0.6932 - auc: 0.
Epoch 6/10
203/203 [==============================] - 3s 17ms/step - loss: 0.6932 - auc: 0.
Epoch 7/10
203/203 [==============================] - 5s 27ms/step - loss: 0.6932 - auc: 0.
Epoch 8/10
203/203 [==============================] - 3s 16ms/step - loss: 0.6932 - auc: 0.
Epoch 9/10
203/203 [==============================] - 3s 16ms/step - loss: 0.6931 - auc: 0.
Epoch 10/10
203/203 [==============================] - 3s 16ms/step - loss: 0.6932 - auc: 0.
<keras.callbacks.History at 0x7ff0916e20a0>
```

```
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_8 (LSTM)               (None, 6, 6)              192
```

```
 lstm_9 (LSTM)               (None, 120)               60960

 dense_1 (Dense)             (None, 1)                 121

=================================================================
Total params: 61,273
Trainable params: 61,273
Non-trainable params: 0
_________________________________________________________________
```

```
y_train_pred = model.predict(x_train)
df_in_sample['y_train_pred'] = y_train_pred
df_in_sample['y_train_pred'] = df_in_sample['y_train_pred'].apply(lambda y: 1 if y>0.5
df_in_sample
```

```
203/203 [==============================] - 2s 10ms/step
```

	RW_returns	y_train_pred
0	1	0
1	1	0
2	1	0
3	1	0
4	0	0
...	...	...
6466	1	0
6467	1	0
6468	0	0
6469	1	0
6470	1	0

6471 rows × 2 columns

```
df_in_sample['y_train'] = y_train
in_sample_acc = round(accuracy_score(df_in_sample['y_train'], df_in_sample['y_train_pr
in_sample_acc
```

```
0.5
```

```
RW_sample_acc = round(accuracy_score(df_in_sample['y_train'], df_in_sample['RW_returns
RW_sample_acc
```

```
0.5
```

```
test_data = scaled_data[training_data_len - input_size:, :]

x_test = []
y_test = np.array(stock_data[["Close"]].pct_change().dropna())[training_data_len:, :]
for i in range(input_size, len(test_data)):
    x_test.append(test_data[i-input_size:i, 0])

x_test = np.array(x_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))
```

```
predictions = model.predict(x_test)
```

```
51/51 [==============================] - 1s 15ms/step
```

```
y= pd.DataFrame(y_test, columns=['y_test'])
```

```
y['y_test'] = y['y_test'].apply(lambda y: 1 if y>0 else 0)
```

```
y['pred'] = predictions
y
```

	y_test	pred
0	1	0.493718
1	0	0.493944
2	0	0.494008
3	1	0.494383
4	0	0.493874
...	...	...
1615	0	0.494557
1616	1	0.494299
1617	0	0.493893
1618	0	0.494171
1619	0	0.494631

1620 rows × 2 columns

```
y['pred'] = y['pred'].apply(lambda y: 1 if y>0 else 0)
RW_test = np.random.normal(mu,sd,len(x_test))
```

```
y['RW_test'] = RW_test
y['RW_test'] = y['RW_test'].apply(lambda y: 1 if y>0 else 0)
y
```

	y_test	pred	RW_test
0	1	1	0
1	0	1	0
2	0	1	0
3	1	1	1
4	0	1	1
...	...	...	...
1615	0	1	1
1616	1	1	1
1617	0	1	0
1618	0	1	1
1619	0	1	1

1620 rows × 3 columns

```
out_sample_acc = round(accuracy_score(y['y_test'], y['pred']),2)
out_sample_acc
```

```
0.55
```

```
RW_test_acc = round(accuracy_score(y['y_test'], y['RW_test']),2)
RW_test_acc
```

```
0.53
```

```
acc = {'RNN_in_sample_acc': in_sample_acc, 'RW_in_sample_acc': RW_sample_acc,
'RNN_out_sample_acc': out_sample_acc, 'RW_out_sample_acc':RW_test_acc}
```

```
result = pd.DataFrame(acc, index= [0])
result
```

	RNN_in_sample_acc	RW_in_sample_acc	RNN_out_sample_acc	RW_out_sample_acc
0	0.5	0.5	0.55	0.53

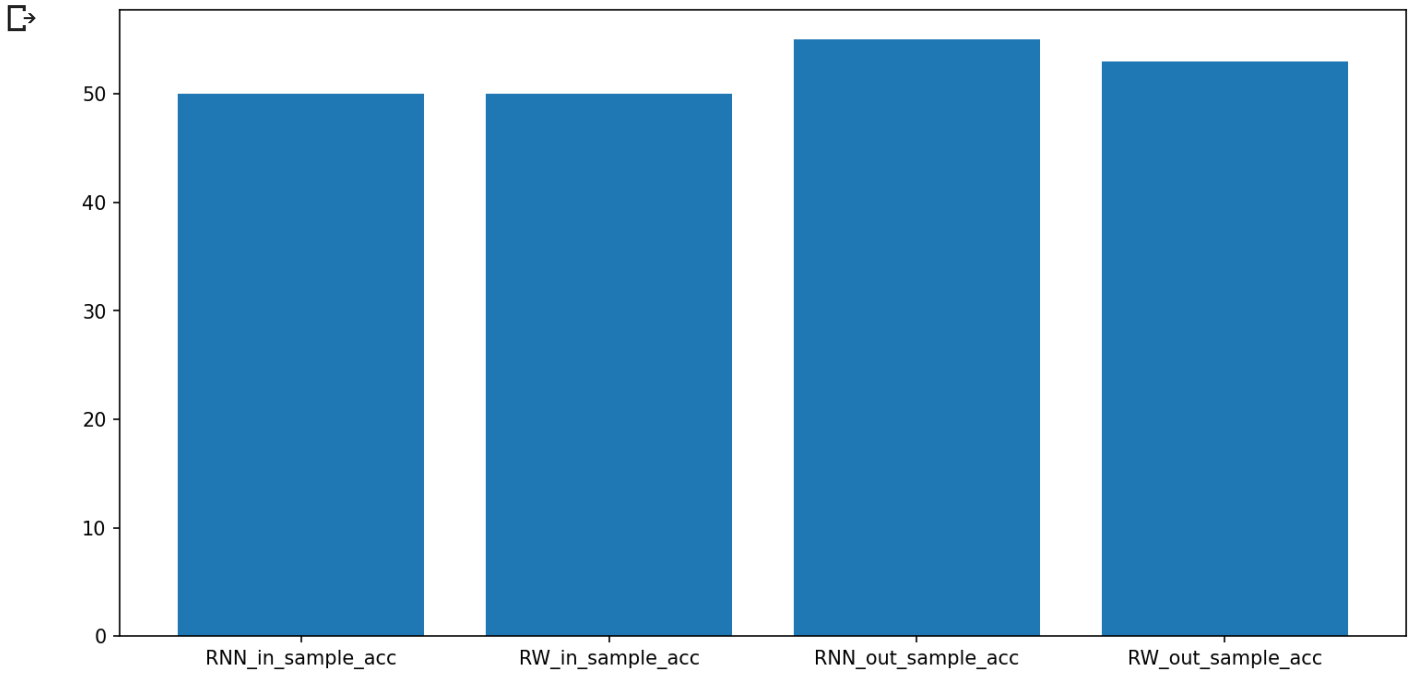
```
plt.figure(figsize=(12,6), dpi=150)
```

```
# Make a random dataset:
height = [50, 50, 55, 53]
bars = ('RNN_in_sample_acc', 'RW_in_sample_acc', 'RNN_out_sample_acc', 'RW_out_sample_
y_pos = np.arange(len(bars))

# Create bars
plt.bar(y_pos, height)

# Create names on the x-axis
plt.xticks(y_pos, bars)

# Show graphic
plt.show()
```



+ Code + Text

The RNN and RW in sample acccuracies are the same but the RNN out of sample accuracy was higher than RW out of sample accuracy indicating that relying on historical data has a lower predictive accuracy for this dataset.

✓ 0s completed at 4:42 PM