1.) Import an asset price from Yahoo Finance

```
pip install yfinance
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting yfinance
  Downloading yfinance-0.2.12-py2.py3-none-any.whl (59 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 59.2/59.2 KB 2.0 MB/s eta 0:00:0
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.8/dist
Collecting requests>=2.26
  Downloading requests-2.28.2-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.8/62.8 KB 2.6 MB/s eta 0:00:0
Collecting frozendict>=2.3.4
  Downloading frozendict-2.3.5-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 111.2/111.2 KB 5.3 MB/s eta 0:00
Collecting beautifulsoup4>=4.11.1
  Downloading beautifulsoup4-4.11.2-py3-none-any.whl (129 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.4/129.4 KB 9.5 MB/s eta 0:00
Collecting cryptography>=3.3.2
  Downloading cryptography-39.0.1-cp36-abi3-manylinux_2_28_x86_64.whl (4.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 20.1 MB/s eta 0:00:0
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python3.8
Collecting html5lib>=1.1
  Downloading html5lib-1.1-py2.py3-none-any.whl (112 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 112.2/112.2 KB 4.7 MB/s eta 0:00
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.8/dist-pa
Collecting soupsieve>1.2
  Downloading soupsieve-2.4-py3-none-any.whl (37 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.8/dist-packa
Requirement already satisfied: webencodings in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pyth
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: pycparser in /usr/local/lib/python3.8/dist-pack
Installing collected packages: soupsieve, requests, html5lib, frozendict, cryp
  Attempting uninstall: requests
    Found existing installation: requests 2.25.1
    Uninstalling requests-2.25.1:
      Successfully uninstalled requests-2.25.1
  Attempting uninstall: html5lib
```

```
          Found existing installation: html5lib 1.0.1
          Uninstalling html5lib-1.0.1:
            Successfully uninstalled html5lib-1.0.1
        Attempting uninstall: beautifulsoup4
          Found existing installation: beautifulsoup4 4.6.3
          Uninstalling beautifulsoup4-4.6.3:
            Successfully uninstalled beautifulsoup4-4.6.3
      Successfully installed beautifulsoup4-4.11.2 cryptography-39.0.1 frozendict-2.
```

```
import yfinance as yf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout


########################################
####Pick your ticker and time period####
########################################
stock_data = yf.download("^IRX", start="1990-01-01", end="2022-02-21")



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


# Split data into training and test sets
training_data_len = int(len(scaled_data) * 0.8)
train_data = scaled_data[0:training_data_len, :]
```

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

2.) Create your x_train/y_train data so that your RNN uses percentage change data to make a binary forecast where the stock moves up or down the next day

Build an RNN Architecture accordingly

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout


x_train = []
y_train = []


###############################################################
####Pick your input size and edit to make binary forecast####
###############################################################
input_size = 5
for i in range(input_size, len(train_data)):
    x_train.append(train_data[i-input_size:i, 0])
    if train_data[i, 0] >= 0:
        y_train.append(1)
    else:
        y_train.append(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))

####################################
####Build Your RNN Architecture####
####################################
model = Sequential()
model.add(LSTM(x_train.shape[1], return_sequences=True, input_shape=(x_train.shape
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=50))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=10)
```

```
    Epoch 1/10
    202/202 [==============================] - 16s 19ms/step - loss: 0.6844 - accu
    Epoch 2/10
    202/202 [==============================] - 5s 24ms/step - loss: 0.6828 - accu
    Epoch 3/10
    202/202 [==============================] - 4s 18ms/step - loss: 0.6827 - accu
    Epoch 4/10
    202/202 [==============================] - 4s 18ms/step - loss: 0.6824 - accu
    Epoch 5/10
    202/202 [==============================] - 5s 23ms/step - loss: 0.6818 - accu
    Epoch 6/10
    202/202 [==============================] - 4s 20ms/step - loss: 0.6825 - accu
    Epoch 7/10
    202/202 [==============================] - 4s 19ms/step - loss: 0.6821 - accu
    Epoch 8/10
    202/202 [==============================] - 4s 22ms/step - loss: 0.6818 - accu
    Epoch 9/10
    202/202 [==============================] - 4s 21ms/step - loss: 0.6819 - accu
    Epoch 10/10
    202/202 [==============================] - 4s 19ms/step - loss: 0.6820 - accu
    <keras.callbacks.History at 0x7f83016ded60>
```

3.) Test your model and compare insample Accurracy, insample random walk assumption Accuracy, Out of sample Accuracy and out of sample random walk assumption Accuracy using a bar chart

```
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 [==============================] - 2s 7ms/step
```

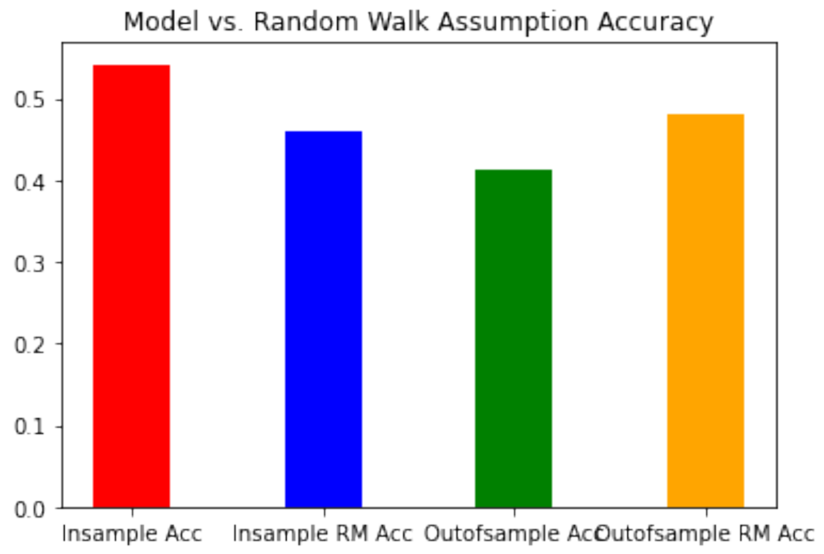
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
```

```
y_train_bi = np.where(y_train > 0, 1, 0)
y_test_bi = np.where(y_test > 0, 1, 0)
predictions = model.predict(x_test)
predictions_bi = np.where(predictions > 0, 1, 0)

insample_acc = accuracy_score(y_train_bi[input_size:], np.where(y_train_bi[input_si
insample_rw_acc = accuracy_score(y_train_bi[input_size:], np.where(y_train_bi[input

outsample_acc = accuracy_score(y_test_bi, predictions_bi)
outsample_rw_acc = accuracy_score(y_test_bi[input_size:], np.where(y_test_bi[input_
# Plot
labels = ['Insample Acc', 'Insample RM Acc', 'Outofsample Acc', 'Outofsample RM Acc
values = [insample_acc, insample_rw_acc, outsample_acc, outsample_rw_acc]
plt.bar(labels, values, width=0.4, color=['red', 'blue', 'green', 'orange'])
plt.title('Model vs. Random Walk Assumption Accuracy')
plt.show()
```

```
51/51 [==============================] - 0s 8ms/step
```



DONT DO# 4.) Plot in and out of sample accuracy

```
import matplotlib.pyplot as plt

# Make predictions on full dataset

test_predict = model.predict(x_test)
test_predictions = (test_predict+1).reshape(1,-1) * np.cumprod(y_test+1)

train_predict = model.predict(x_train)
train_predictions = (train_predict+1).reshape(1,-1) * np.cumprod(y_train+1)



plt.plot(stock_data[:training_data_len- input_size].index, np.cumprod(y_train+1), l
plt.plot(stock_data[:training_data_len- input_size].index, train_predictions[0], la
end_val = np.cumprod(y_train+1)[-1]
test_predict = model.predict(x_test)
test_predictions = (test_predict+1).reshape(1,-1) * (np.cumprod((y_test+1))*end_val
plt.plot(stock_data[training_data_len+1:].index, np.cumprod((y_test+1))*end_val,lab
plt.plot(stock_data[training_data_len+1:].index, test_predictions[0], label="Test P
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

5.) Write an observation/conclusion about the graphs from Q4 and Q3

In sample accuracy of the model represents how well the model fits the training data, which is ploted in red. The graph shows that the LSTM model outperforms the random walk assumption for in-sample data. This means that LSTM model is better to captire the patterns andtrends in the data than just simply assuming that the nsxt value will be the same as the previous value. However, the random walk ourperfors the LSTM model for the out of sample data, which indicates that the LSTM model does not generalize well to the new data.

6.) Create a parameter for number of lags in your input layer. Do a 3-fold CV to test three different time lags. i.e. Tested using 5,10,20 days of previous price data to forecast

```
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.wrappers.scikit_learn import KerasRegressor

# Define the Keras model
###Edit here to create your optimizer
def create_model(n_lags):
    model = Sequential()
    model.add(LSTM(units=10, input_shape=(n_lags, 1)))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'
    return model
# Wrap the Keras model in a scikit-learn compatible estimator
model = KerasRegressor(build_fn=create_model, verbose=0)
# Define the hyperparameters to search over
param_grid = {'batch_size': [10, 20, 100],
              'epochs': [10, 100],
              'n_lags': [5, 10, 20]}

n_samples = 1000
n_features = 1
data = np.random.randn(n_samples, n_features)

targets = np.random.randint(0, 2, size=(n_samples, 1))
```

```
# Perform the grid search over the hyperparameters
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result = grid.fit(data, targets)
# Print the results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
<ipython-input-59-7ae6a3bed8ca>:14: DeprecationWarning: KerasRegressor is dep
  model = KerasRegressor(build_fn=create_model, verbose=0)
```

正在执行（已持续 7 分 22 秒） C > fi > _run_... > evaluate_c... > __... > ret... > wrap_futu... > r... > w...