Using an LSTM Model To Predict Closing Stock Prices

Goals:

- Predict as accurately as possible the closing stock price for a given stock.
- Fit three models on three different time intervals. Ten days, thirty days, and sixty days.
- Show model performance over test data so as to provide user with the basis of trust in the model.
- Open the door for further development of a minute to minute stock prediction platform.

Step 1: Choosing a Model

Why an LSTM Model and what is it?

- An LSTM Model is a recurrent neural network (RNN) and over time, RNNS have proved themselves as one of the most powerful models for processing sequential data. In our case this is time series data.
- LSTM introduces the concept of a "memory cell", a unit of computation that replaces traditional artificial neurons in the hidden layer of the network.

- With these memory cells, networks are able to effectively associate "memories" and input remote in time, therefore prove to be well suited to grasp structure of data dynamically over time with high prediction capacity.
- Obviously, this type of model sacrifices much interoperability and because of this a level of artistry in tuning is required.
- It would therefore be impotant for the user in this particular instance to possibly develop an algorithmic trading strategy on top that takes in to account the models performance over time and whether it's predictions should be trusted based on other factors.

Step 2: Training Data

- Five years of stock data was collected spanning years 2010-2016.
- Metrics used to train models were open, low, high, close and volume.

```
df = pd.read_csv("Resources/prices.csv")
```

```
df['date'] = pd.to_datetime(df.date,format='%Y-%m-%d')
df.set_index('symbol', inplace=True)
df = df.sort_values('date')
tickers = df.index.unique()
tickers
```

Step 3: Pre-processing Traning Data

- Test data was scaled and preproccessed for each of the models (10 day, 30 day, 60 day)

```
for stock in test_tickers:
    train_df = df.loc[stock, :]
    train_df.set_index('date', inplace=True)
    train_df.drop('open', axis=1, inplace=True)
    train_dataset = train_df.values
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(train_dataset)
    X_train_sixty, y_train_sixty = [], []
    for i in range(61,len(train_df)):
        X_train_sixty.append(scaled_data[i-61:i-1,0:4])
        y_train_sixty.append(scaled_data[i,0])
    X_train_sixty, y_train_sixty = np.array(X_train_sixty), np.array(y_train_sixty)
    model_sixty_day.fit(X_train_sixty, y_train_sixty, epochs=1, batch_size=1, verbose=2)
```

Step 4: Construction of Model and Layers

- Layers are added for each respective model

```
model_sixty_day = Sequential()
model_sixty_day.add(LSTM(units=50, return_sequences=True, input_shape=(60,4)))
model_sixty_day.add(LSTM(units=50))
model_sixty_day.add(Dense(1))
model_sixty_day.compile(loss='mean_squared_error', optimizer='adam')
```

```
model_thirty_day = Sequential()
model_thirty_day.add(LSTM(units=50, return_sequences=True, input_shape=(30,4)))
model_thirty_day.add(LSTM(units=50))
model_thirty_day.add(Dense(1))
model_thirty_day.compile(loss='mean_squared_error', optimizer='adam')
```

```
model_ten_day = Sequential()
model_ten_day.add(LSTM(units=50, return_sequences=True, input_shape=(10,4)))
model_ten_day.add(LSTM(units=50))
model_ten_day.add(Dense(1))
model_ten_day.compile(loss='mean_squared_error', optimizer='adam')
```

Step 4: Training Models

```
Epoch 97/100
1198/1198 [==============================] - 6s 5ms/step - loss:
0.0018
Epoch 98/100
1198/1198 [==============================] - 6s 5ms/step - loss:
0.0014
Epoch 99/100
1198/1198 [==============================] - 6s 5ms/step - loss:
0.0014
Epoch 100/100
1198/1198 [==============================] - 6s 5ms/step - loss:
0.0015
```

Step 5: ETL and API Call for Real Data

- In order to pull real recent data for models to perform on real data an ETL pipeline with an API call was constructed.
- Quandl is no longer updated and Yahoo finance and Google Finance have been deprecated.
- We settled on Alpha Vantage to pull the real data for the last year for the models to perform on.

```
#https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol=MSFT&outputsize=full&apikey=demo
from alpha_vantage.timeseries import TimeSeries
url = "https://www.alphavantage.co/query?"
query_url = f"{url}function=TIME_SERIES_DAILY&symbol={user_ticker}&outputsize=full&apikey={apiKey}"
```

Data is pre-processed

```

dataset = df_filtered.values
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(dataset)
X_test, y_test = [], []
for i in range(61,len(dataset)):
    X_test.append(scaled_data[i-61:i-1,0:4])
    y_test.append(scaled_data[i,0])
X_test, y_test = np.array(X_test), np.array(y_test)
```

Models are loaded

```
import random
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from keras.models import load_model
ten_day_model = load_model("../ML/model_ten_day.h5")
thirty_day_model = load_model("../ML/model_thirty_day.h5")
sixty_day_model = load_model("../ML/model_sixty_day.h5")
```

Predictions are calculated

```
final = []
for price in closing_price:
    final.append(np.pad(price, (0, 3), 'constant'))
final_price = scaler.inverse_transform(final)
close = []
for price in final_price:
    close.append(price[0])
```

Future Possibilities

- More layers of machine learning used to develop trading strategies which utilize the model's predictions and react to the model's accuracy over time.
- Reinforcement learning to improve model's performance by learning from it's real time predictions.
- Create minute to minute predictions using models to open the door for high volume trading potential.

In [ ]:

```

```