Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import pandas_datareader as data
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense,LSTM,Dropout
```

Load Dataset

```
df = pd.read_csv('HHL Historical Data.csv')
```

```
# Display the first few rows
df.head()
```

	Date	Price	Open	High	Low	Vol.	Change %
0	02/22/2024	73.5	73.4	73.9	73.0	53.45K	0.68%
1	02/21/2024	73.0	72.3	73.0	71.7	8.33K	1.39%
2	02/20/2024	72.0	73.0	73.0	72.0	39.00K	-1.37%
3	02/19/2024	73.0	73.4	73.4	72.5	1.46K	-0.54%
4	02/16/2024	73.4	72.0	73.4	71.1	57.84K	2.37%

```
# Display concise information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 508 entries, 0 to 507
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      508 non-null    object
 1   Price     508 non-null    float64
 2   Open      508 non-null    float64
 3   High      508 non-null    float64
 4   Low       508 non-null    float64
 5   Vol.      508 non-null    object
 6   Change %  508 non-null    object
dtypes: float64(4), object(3)
memory usage: 27.9+ KB
```

Convert 'Vol.' column to numeric format handling 'K' and 'M' suffixes

```
# Convert only the string values to numeric, leave other types as they are
df['Vol.'] = df['Vol.'].apply(lambda x: x if pd.isna(x) or isinstance(x, str) else str(x))

# Remove 'K' and 'M' from string values
df['Vol.'] = df['Vol.'].str.replace('K', 'e3').str.replace('M', 'e6')

# Convert the column to numeric, handle non-numeric values by coercing them to NaN
df['Vol.'] = pd.to_numeric(df['Vol.'], errors='coerce')

# Display to check the changes
```

```
# Display to check the changes
df
```

	Date	Price	Open	High	Low	Vol.	Change %
0	01/02/2023	56.5	56.2	57.5	56.2	30930.0	0.53%
1	01/02/2024	68.9	67.0	68.9	67.0	9030.0	0.00%
2	01/03/2022	67.4	67.0	67.9	66.5	577220.0	1.05%
3	01/03/2023	56.7	56.5	57.3	56.5	10590.0	0.35%
4	01/03/2024	69.0	68.9	69.1	68.0	58520.0	0.15%
...	...	...	...	...	...	...	...
503	12/28/2022	56.6	58.8	58.8	56.6	22570.0	-3.74%
504	12/28/2023	67.5	68.3	68.3	67.4	192900.0	-0.74%
505	12/29/2022	56.1	56.5	56.5	56.0	2370.0	-0.88%
506	12/29/2023	68.9	67.9	68.9	66.5	273140.0	2.07%
507	12/30/2022	56.2	56.2	57.5	56.2	8060.0	0.18%

508 rows × 7 columns

```
# Convert the 'Date' column to datetime format
df['Date']= pd.to_datetime(df['Date'])

# Display concise information after the conversion
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 508 entries, 0 to 507
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      508 non-null    datetime64[ns]
 1   Price     508 non-null    float64
 2   Open      508 non-null    float64
 3   High      508 non-null    float64
 4   Low       508 non-null    float64
 5   Vol.      508 non-null    float64
 6   Change %  508 non-null    object
dtypes: datetime64[ns](1), float64(5), object(1)
memory usage: 27.9+ KB
```

Change the order of the dataset

```
df = df.sort_values(by='Date')
df= df.reset_index(drop=True)

df
```

	Date	Price	Open	High	Low	Vol.	Change %
0	2022-01-03	67.4	67.0	67.9	66.5	577220.0	1.05%
1	2022-01-04	71.9	68.0	74.6	68.0	1040000.0	6.68%
2	2022-01-05	70.9	71.9	73.0	70.6	447150.0	-1.39%
3	2022-01-06	69.9	71.4	71.4	69.4	310490.0	-1.41%
4	2022-01-07	69.4	69.9	71.2	69.4	655940.0	-0.72%
...	...	...	...	...	...	...	...
503	2024-02-16	73.4	72.0	73.4	71.1	57840.0	2.37%
504	2024-02-19	73.0	73.4	73.4	72.5	1460.0	-0.54%
505	2024-02-20	72.0	73.0	73.0	72.0	39000.0	-1.37%
506	2024-02-21	73.0	72.3	73.0	71.7	8330.0	1.39%
507	2024-02-22	73.5	73.4	73.9	73.0	53450.0	0.68%

508 rows × 7 columns

Next steps: Generate code with `df` View recommended plots

Split the Dataset into training and test datasets

```
split_dataset = int(0.8 * len(df))
training_data = df.iloc[:split_dataset]
test_data = df.iloc[split_dataset:]
```

```
training_data
```

	Date	Price	Open	High	Low	Vol.	Change %
0	2022-01-03	67.4	67.0	67.9	66.5	577220.0	1.05%
1	2022-01-04	71.9	68.0	74.6	68.0	1040000.0	6.68%
2	2022-01-05	70.9	71.9	73.0	70.6	447150.0	-1.39%
3	2022-01-06	69.9	71.4	71.4	69.4	310490.0	-1.41%
4	2022-01-07	69.4	69.9	71.2	69.4	655940.0	-0.72%
...	...	...	...	...	...	...	...
401	2023-09-15	82.0	81.5	82.9	81.4	27020.0	0.49%
402	2023-09-18	81.0	82.5	82.5	80.8	122100.0	-1.22%
403	2023-09-19	80.7	81.5	81.5	79.6	173880.0	-0.37%
404	2023-09-20	80.0	80.0	81.0	79.9	65540.0	-0.87%
405	2023-09-21	80.0	81.0	81.1	80.0	20310.0	0.00%

406 rows × 7 columns

Next steps: Generate code with `training_data` View recommended plots

```
test_data
```

	Date	Price	Open	High	Low	Vol.	Change %
406	2023-09-22	79.6	80.0	80.1	79.6	106070.0	-0.50%
407	2023-09-25	79.0	79.5	79.9	79.0	14270.0	-0.75%
408	2023-09-26	78.5	78.1	79.9	77.9	16540.0	-0.63%
409	2023-09-27	79.5	79.5	80.1	78.6	48380.0	1.27%
410	2023-10-02	77.9	78.6	79.0	77.9	26990.0	-2.01%
...	...	...	...	...	...	...	...
503	2024-02-16	73.4	72.0	73.4	71.1	57840.0	2.37%
504	2024-02-19	73.0	73.4	73.4	72.5	1460.0	-0.54%
505	2024-02-20	72.0	73.0	73.0	72.0	39000.0	-1.37%
506	2024-02-21	73.0	72.3	73.0	71.7	8330.0	1.39%
507	2024-02-22	73.5	73.4	73.9	73.0	53450.0	0.68%

102 rows × 7 columns

Next steps: Generate code with `test_data` View recommended plots

Prepare training data by excluding 'Date' and 'Change %' columns

```
train_data = training_data.drop(['Date','Change %'],axis=1)
train_data.head()
```

	Price	Open	High	Low	Vol.
0	67.4	67.0	67.9	66.5	577220.0
1	71.9	68.0	74.6	68.0	1040000.0
2	70.9	71.9	73.0	70.6	447150.0
3	69.9	71.4	71.4	69.4	310490.0
4	69.4	69.9	71.2	69.4	655940.0

Next steps: Generate code with `train_data` View recommended plots

```
# Scale the training data using Min-Max Scaling
scaler = MinMaxScaler()
train_data = scaler.fit_transform(train_data)

train_data
```

```
array([[0.61725664, 0.60572687, 0.60784314, 0.61797753, 0.11633759],
       [0.71681416, 0.6277533 , 0.75381264, 0.65168539, 0.20964396],
       [0.69469027, 0.71365639, 0.71895425, 0.71011236, 0.09011269],
       ...,
       [0.91150442, 0.92511013, 0.90413943, 0.91235955, 0.0350156 ],
       [0.8960177 , 0.89207048, 0.89324619, 0.91910112, 0.01317193],
       [0.8960177 , 0.91409692, 0.89542484, 0.92134831, 0.00405259]])
```

```
x_train = []
y_train = []
```

```
# Prepare sequential training data for a time-series model
```

```
for i in range(60, train_data.shape[0]):
    x_train.append(train_data[i-60:i])
    y_train.append(train_data[i, 0])

# Convert lists to numpy arrays
x_train, y_train = np.array(x_train), np.array(y_train)

# Display the shape of the input training data
x_train.shape
```

```
(346, 60, 5)
```

Building LSTM

```
# Add the 4 LSTM layers, ReLU activation, and input shape
regressior = Sequential()

regressior.add(LSTM(units=50,activation='relu',return_sequences=True, input_shape=(x_train.shape[1],5)))
regressior.add(Dropout(0.2))

regressior.add(LSTM(units=60,activation='relu',return_sequences=True ))
regressior.add(Dropout(0.3))

regressior.add(LSTM(units=80,activation='relu',return_sequences=True))
regressior.add(Dropout(0.4))

regressior.add(LSTM(units=120,activation='relu'))
regressior.add(Dropout(0.5))


# Add the output Dense layer with 1 unit
regressior.add(Dense(units=1))

regressior.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_16 (LSTM)              (None, 60, 50)            11200

 dropout_16 (Dropout)        (None, 60, 50)            0

 lstm_17 (LSTM)              (None, 60, 60)            26640

 dropout_17 (Dropout)        (None, 60, 60)            0

 lstm_18 (LSTM)              (None, 60, 80)            45120

 dropout_18 (Dropout)        (None, 60, 80)            0

 lstm_19 (LSTM)              (None, 120)               96480

 dropout_19 (Dropout)        (None, 120)               0

 dense_4 (Dense)             (None, 1)                 121

=================================================================
Total params: 179561 (701.41 KB)
Trainable params: 179561 (701.41 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Compile and train the LSTM-based regression model
regressior.compile(optimizer='adam', loss='mean_squared_error')
```

```
regressor.fit(x_train,y_train, epochs=10,batch_size=32)
```

```
Epoch 1/10
11/11 [==============================] - 7s 156ms/step - loss: 0.1676
Epoch 2/10
11/11 [==============================] - 2s 159ms/step - loss: 0.0421
Epoch 3/10
11/11 [==============================] - 2s 157ms/step - loss: 0.0265
Epoch 4/10
11/11 [==============================] - 2s 155ms/step - loss: 0.0210
Epoch 5/10
11/11 [==============================] - 2s 156ms/step - loss: 0.0220
Epoch 6/10
11/11 [==============================] - 3s 271ms/step - loss: 0.0202
Epoch 7/10
11/11 [==============================] - 2s 156ms/step - loss: 0.0184
Epoch 8/10
11/11 [==============================] - 2s 155ms/step - loss: 0.0159
Epoch 9/10
11/11 [==============================] - 2s 157ms/step - loss: 0.0162
Epoch 10/10
11/11 [==============================] - 2s 156ms/step - loss: 0.0118
<keras.src.callbacks.History at 0x7c2520a657e0>
```

Prepare test dataset

```
test_data.head()
```

	Date	Price	Open	High	Low	Vol.	Change %
406	2023-09-22	79.6	80.0	80.1	79.6	106070.0	-0.50%
407	2023-09-25	79.0	79.5	79.9	79.0	14270.0	-0.75%
408	2023-09-26	78.5	78.1	79.9	77.9	16540.0	-0.63%
409	2023-09-27	79.5	79.5	80.1	78.6	48380.0	1.27%
410	2023-10-02	77.9	78.6	79.0	77.9	26990.0	-2.01%

Next steps: Generate code with test_data View recommended plots

```
# Extract the most recent 60 days of training data
past_60_days =training_data.tail(60)
past_60_days
```

	Date	Price	Open	High	Low	Vol.	Change %
346	2023-06-23	68.0	67.0	68.0	67.0	15140.0	0.74%
347	2023-06-26	69.4	67.2	69.4	66.5	57380.0	2.06%
348	2023-06-27	69.3	69.4	70.2	69.0	226430.0	-0.14%
349	2023-06-28	69.5	69.5	69.9	69.0	47440.0	0.29%
350	2023-07-04	73.0	72.0	74.5	72.0	1820000.0	5.04%
351	2023-07-05	74.0	73.5	74.1	73.0	519160.0	1.37%
352	2023-07-06	74.0	74.0	74.0	73.0	198620.0	0.00%
353	2023-07-07	77.0	74.0	77.1	73.8	1570000.0	4.05%
354	2023-07-10	76.0	75.0	77.4	75.0	215900.0	-1.30%
355	2023-07-11	74.0	75.6	75.6	73.6	436780.0	-2.63%
356	2023-07-12	73.6	74.0	74.0	73.1	131770.0	-0.54%
357	2023-07-13	74.0	73.7	74.5	73.7	1480000.0	0.54%
358	2023-07-14	74.6	74.0	75.0	74.0	56460.0	0.81%
359	2023-07-17	73.4	73.8	74.6	72.5	237970.0	-1.61%
360	2023-07-18	75.4	74.0	75.9	72.5	894260.0	2.72%
361	2023-07-19	74.5	75.7	75.7	74.0	473920.0	-1.19%
362	2023-07-20	74.5	74.0	75.0	74.0	55590.0	0.00%
363	2023-07-21	76.0	75.0	76.5	74.1	275740.0	2.01%
364	2023-07-24	75.0	74.1	75.0	74.0	49630.0	-1.32%
365	2023-07-25	73.6	75.0	75.0	73.5	73010.0	-1.87%
366	2023-07-26	74.0	74.0	74.2	73.8	236300.0	0.54%
367	2023-07-27	74.3	74.0	74.3	73.3	88480.0	0.41%
368	2023-07-28	76.0	74.0	76.2	74.0	265340.0	2.29%
369	2023-07-31	76.8	76.2	76.8	75.1	612330.0	1.05%
370	2023-08-02	77.8	76.0	77.9	76.0	1040000.0	1.30%
371	2023-08-03	82.0	77.1	82.4	77.1	1160000.0	5.40%
372	2023-08-04	80.5	82.0	82.0	80.0	206270.0	-1.83%
373	2023-08-07	83.0	81.0	83.0	80.1	358430.0	3.11%
374	2023-08-08	84.7	80.2	84.8	80.2	679640.0	2.05%
375	2023-08-09	84.0	84.5	85.9	83.5	580520.0	-0.83%
376	2023-08-10	83.0	84.9	84.9	83.0	125060.0	-1.19%
377	2023-08-11	81.0	82.0	82.0	81.0	77540.0	-2.41%
378	2023-08-14	79.8	80.7	80.7	78.1	106510.0	-1.48%
379	2023-08-15	78.6	78.8	79.7	78.6	83240.0	-1.50%
380	2023-08-16	80.0	78.8	80.0	78.5	290820.0	1.78%
381	2023-08-17	80.0	80.0	80.0	79.0	68120.0	0.00%
382	2023-08-18	79.7	80.0	80.0	78.8	1050000.0	-0.38%
383	2023-08-21	80.0	79.6	80.0	79.0	440280.0	0.38%
384	2023-08-22	79.9	79.2	80.0	79.2	120920.0	-0.12%

385	2023-08-23	80.2	79.3	80.5	79.3	343960.0	0.38%
386	2023-08-24	80.0	80.2	80.4	80.0	134940.0	-0.25%
387	2023-08-25	80.3	80.0	80.4	80.0	84690.0	0.38%
388	2023-08-28	80.0	80.0	80.4	79.5	111340.0	-0.37%
389	2023-08-29	79.8	80.0	80.6	79.8	28020.0	-0.25%
390	2023-08-31	81.0	79.8	81.0	79.8	302650.0	1.50%
391	2023-09-01	83.0	79.8	83.0	79.8	185230.0	2.47%
392	2023-09-04	82.5	83.0	83.5	82.5	256470.0	-0.60%
393	2023-09-05	83.2	83.0	83.2	81.0	343670.0	0.85%
394	2023-09-06	82.0	82.0	83.2	81.8	145650.0	-1.44%
395	2023-09-07	83.0	83.0	83.1	82.0	42350.0	1.22%
396	2023-09-08	82.5	82.5	82.5	82.0	26810.0	-0.60%
397	2023-09-11	82.5	82.0	83.0	81.0	43080.0	0.00%
398	2023-09-12	81.5	81.9	82.2	81.0	28530.0	-1.21%
399	2023-09-13	81.1	82.0	82.0	81.0	678100.0	-0.49%
400	2023-09-14	81.6	81.2	82.0	81.1	15340.0	0.62%
401	2023-09-15	82.0	81.5	82.9	81.4	27020.0	0.49%
402	2023-09-18	81.0	82.5	82.5	80.8	122100.0	-1.22%
403	2023-09-19	80.7	81.5	81.5	79.6	173880.0	-0.37%
404	2023-09-20	80.0	80.0	81.0	79.9	65540.0	-0.87%
405	2023-09-21	80.0	81.0	81.1	80.0	20310.0	0.00%

Next steps: Generate code with `past_60_days` View recommended plots

```
# Combine the most recent 60 days of training data with the test data
data = past_60_days.append(test_data, ignore_index = True)

data
```

```
<ipython-input-158-51b11c5fb72d>:2: FutureWarning: The frame.append method is deprecated and will be removed from
  data = past_60_days.append(test_data, ignore_index = True)
```

	Date	Price	Open	High	Low	Vol.	Change %
0	2023-06-23	68.0	67.0	68.0	67.0	15140.0	0.74%
1	2023-06-26	69.4	67.2	69.4	66.5	57380.0	2.06%
2	2023-06-27	69.3	69.4	70.2	69.0	226430.0	-0.14%
3	2023-06-28	69.5	69.5	69.9	69.0	47440.0	0.29%
4	2023-07-04	73.0	72.0	74.5	72.0	1820000.0	5.04%
...	...	...	...	...	...	...	...
157	2024-02-16	73.4	72.0	73.4	71.1	57840.0	2.37%
158	2024-02-19	73.0	73.4	73.4	72.5	1460.0	-0.54%
159	2024-02-20	72.0	73.0	73.0	72.0	39000.0	-1.37%
160	2024-02-21	73.0	72.3	73.0	71.7	8330.0	1.39%
161	2024-02-22	73.5	73.4	73.9	73.0	53450.0	0.68%

162 rows × 7 columns

Next steps: Generate code with data | View recommended plots

```
# Remove 'Date' and 'Change %' columns
data = data.drop(['Date', 'Change %'], axis = 1)
data.head()
```

	Price	Open	High	Low	Vol.
0	68.0	67.0	68.0	67.0	15140.0
1	69.4	67.2	69.4	66.5	57380.0
2	69.3	69.4	70.2	69.0	226430.0
3	69.5	69.5	69.9	69.0	47440.0
4	73.0	72.0	74.5	72.0	1820000.0

Next steps: Generate code with data | View recommended plots

```
# Scale the combined dataset using Min-Max Scaling
inputs = scaler.transform(data)
inputs
```

```
           [6.41592920e-01, 6.60792952e-01, 6.42701525e-01, 6.58426966e-01,
            2.25070013e-02],
           [6.30530973e-01, 6.34361233e-01, 6.23093682e-01, 6.51685393e-01,
            1.56740507e-02],
           [6.32743363e-01, 6.49779736e-01, 6.42701525e-01, 6.51685393e-01,
            4.22195295e-03],
           [6.37168142e-01, 6.38766520e-01, 6.31808279e-01, 6.51685393e-01,
            1.13795141e-02],
           [6.41592920e-01, 6.60792952e-01, 6.42701525e-01, 6.51685393e-01,
            1.66277201e-02],
           [6.52654867e-01, 6.49779736e-01, 6.31808279e-01, 6.62921348e-01,
            5.40143837e-03],
           [6.52654867e-01, 6.49779736e-01, 6.31808279e-01, 6.62921348e-01,
            2.41240053e-02],
           [6.59292035e-01, 6.49779736e-01, 6.42701525e-01, 6.74157303e-01,
            1.14984707e-02],
           [6.46017699e-01, 6.65198238e-01, 6.75381264e-01, 6.67415730e-01,
            2.00754467e-02],
           [6.74778761e-01, 6.71806167e-01, 6.55773420e-01, 6.71910112e-01,
            1.41376953e-02],
           [6.96902655e-01, 6.82819383e-01, 7.03703704e-01, 7.01123596e-01,
            3.58704703e-02],
           [7.85398230e-01, 6.93832599e-01, 7.62527233e-01, 6.96629213e-01,
            2.47449993e-02],
           [7.19026549e-01, 7.59911894e-01, 7.62527233e-01, 7.41573034e-01,
            2.80072342e-02],
           [7.07964602e-01, 7.13656388e-01, 6.97167756e-01, 7.30337079e-01,
            2.34304275e-02],
           [7.19026549e-01, 7.04845815e-01, 7.18954248e-01, 7.23595506e-01,
            4.46531002e-02],
           [7.12389381e-01, 7.15859031e-01, 6.97167756e-01, 7.30337079e-01,
            5.01835763e-03],
           [7.50000000e-01, 7.15859031e-01, 7.27668845e-01, 7.21348315e-01,
            1.16194436e-02],
           [7.41150442e-01, 7.46696035e-01, 7.27668845e-01, 7.52808989e-01,
            2.52026800e-04],
           [7.19026549e-01, 7.37885463e-01, 7.18954248e-01, 7.41573034e-01,
            7.82089564e-03],
           [7.41150442e-01, 7.22466960e-01, 7.18954248e-01, 7.34831461e-01,
            1.63716609e-03],
           [7.52212389e-01, 7.46696035e-01, 7.38562092e-01, 7.64044944e-01,
            1.07343254e-02]])
```

```
# Prepare sequential test data for the time-series model
X_test =[]
y_test=[]

for i in range(60,inputs.shape[0]):
  X_test.append(inputs[i-60:i])
  y_test.append(inputs[i,0])


# Convert lists to numpy arrays for test data
X_test, y_test = np.array(X_test),np.array(y_test)
X_test.shape, y_test.shape
```

```
((102, 60, 5), (102,))
```

```
# Predict using the trained LSTM-based regression model on the test data
y_pred = regressor.predict(X_test)
y_pred
```

```
       [0.78409046],
       [0.7728549 ],
       [0.76219714],
       [0.752254  ],
       [0.74305457],
       [0.7345099 ],
       [0.72651285],
       [0.7190334 ],
       [0.71213704],
       [0.7059081 ],
       [0.70062006],
       [0.69651836],
       [0.69382924],
       [0.69276315],
       [0.6932579 ],
       [0.69507724],
       [0.69789463],
       [0.7013435 ],
       [0.7050379 ],
       [0.7086441 ],
       [0.7119128 ],
       [0.71461725],
       [0.71653837],
       [0.717497  ],
       [0.7174116 ],
       [0.7162867 ],
       [0.7142554 ],
       [0.711566  ],
       [0.7084399 ],
       [0.70518535],
       [0.70206934],
       [0.69929415],
       [0.69699043],
       [0.6951804 ],
       [0.69383544],
       [0.6929515 ],
       [0.6926505 ],
       [0.69301516],
       [0.6940548 ],
       [0.69553906],
       [0.6971809 ],
       [0.6987227 ],
       [0.6998753 ],
       [0.7003606 ],
       [0.6999883 ],
       [0.6986329 ],
       [0.6962454 ],
       [0.6928607 ],
       [0.68856925],
       [0.6836141 ],
```

```
# Access the scaling factor used by the MinMaxScaler
scaler.scale_
```

```
array([2.21238938e-02, 2.20264317e-02, 2.17864924e-02, 2.24719101e-02,
       2.01621440e-07])
```

```
# Set a custom scaling factor
scale = 1/8.18605127e-04
scale
```

```
1221.5901990069017
```

Visualize the predicted and actual stock prices

```
# Rescale predicted and actual values by the custom scaling factor
```

```
    y_pred = y_pred*scale
```