Stock Price Forecasting

Abstract:

Since there is no guarantee in a corporate setting, forecasting stock prices is quite challenging. The stock market for a company's financial performance becomes incredibly unpredictable, dynamic, and nonlinear due to a variety of macro and microelements, including politics, world economic circumstances, and unforeseen occurrences. Deep learning, commonly referred to as deep structured learning, is one of several machine learning techniques built on artificial neural networks and representation learning. Several deep learning-related techniques may be useful to estimate stock values. Drift, Naive, Artificial Neural Networks, and Random Forests are examples. Despite not being frequently used for financial time series prediction, they are intrinsically appropriate to this domain [1]. Long short-term memory (LSTM) networks will be employed in this study to anticipate the stock price. The data for "Apple Inc." will be obtained from the Yahoo finance website starting in January 1990 and ending in July 2022. The model will be trained using the first 80% of the data. The model will next be validated using the subsequent 10% of data. The remaining information will be utilized to evaluate the forecasting model. The same process will be done using data collected starting from January 2020. Finally, this paper will compare the findings to actual data to see what actually happened and how accurate the predictions are. Additionally, it will describe how the models respond when given various weights in the training set of data. Python will be utilized as the programming language for the data analysis and implementation of the results to show this article. We will see a more accurate forecast for the short-term data than the long-term data because of the theoretical mechanism of the LSTM approach, which is heavily reliant on the training data set and the test data set.

In [27]:
```
# Import data
import pandas as pd
df = pd.read_csv("AAPL01.csv")
df.head()
```

Out[27]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1990-01-02	0.314732	0.334821	0.312500	0.332589	0.265325	183198400
1	1990-01-03	0.339286	0.339286	0.334821	0.334821	0.267106	207995200
2	1990-01-04	0.341518	0.345982	0.332589	0.335938	0.267997	221513600
3	1990-01-05	0.337054	0.341518	0.330357	0.337054	0.268887	123312000
4	1990-01-08	0.334821	0.339286	0.330357	0.339286	0.270668	101572800

In [28]:
```
# Taking only the closing price and date coumn
df = df[['Date', 'Close']]
df.head()
```

Out[28]:

	Date	Close
0	1990-01-02	0.332589
1	1990-01-03	0.334821
2	1990-01-04	0.335938
3	1990-01-05	0.337054
4	1990-01-08	0.339286

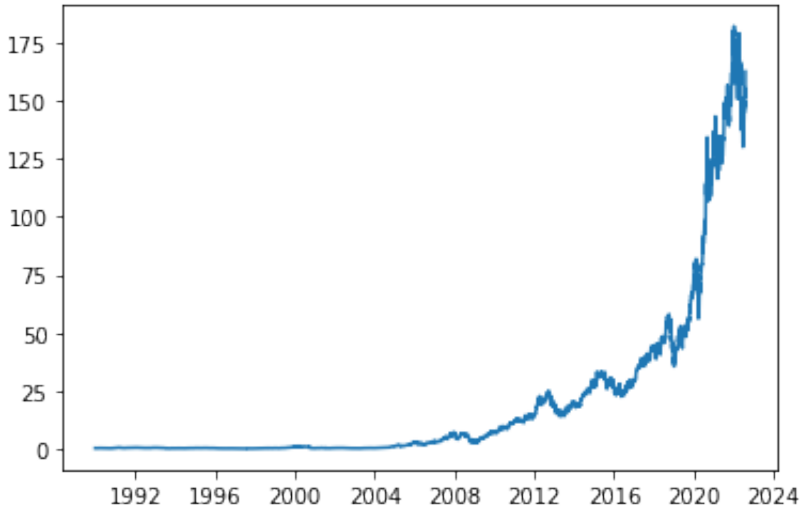
In [13]:
```
# Formetting date and index
import datetime

def datetime_str(s):
  split = s.split('-')
  year, month, day = int(split[0]), int(split[1]), int(split[2])
  return datetime.datetime(year=year, month=month, day=day)
datetime_object = datetime_str('1990-01-02')

df['Date'] = df['Date'].apply(datetime_str)
df['Date']
df.index = df.pop('Date')
df.head()
```

Out[13]:

	Close
Date	
1990-01-02	0.332589
1990-01-03	0.334821
1990-01-04	0.335938
1990-01-05	0.337054
1990-01-08	0.339286

In [14]:
```
# See how the data data looks like
import matplotlib.pyplot as plt

plt.plot(df.index, df['Close'])
```

Out[14]: [<matplotlib.lines.Line2D at 0x7fc4f387f810>]



In [15]:
```
# This function will give us output on the based of the input.
# Target 1, Target 2, Target 3 are the previous value respectively
# Target is the predicted outut.
```

```
import numpy as np

def windowed_df(dataframe, first_date_str, last_date_str, n=3):
  first_date = datetime_str(first_date_str)
  last_date  = datetime_str(last_date_str)

  target_date = first_date

  dates = []
  X, Y = [], []

  last_time = False
  while True:
    df_subset = dataframe.loc[:target_date].tail(n+1)

    if len(df_subset) != n+1:
      print(f'Error: Window of size {n} is too large for date {targ
      return

    values = df_subset['Close'].to_numpy()
    x, y = values[:-1], values[-1]

    dates.append(target_date)
    X.append(x)
    Y.append(y)

    next_week = dataframe.loc[target_date:target_date+datetime.time
    next_datetime_str = str(next_week.head(2).tail(1).index.values[
    next_date_str = next_datetime_str.split('T')[0]
    year_month_day = next_date_str.split('-')
    year, month, day = year_month_day
    next_date = datetime.datetime(day=int(day), month=int(month), y

    if last_time:
      break

    target_date = next_date

    if target_date == last_date:
      last_time = True

  ret_df = pd.DataFrame({})
  ret_df['Target Date'] = dates

  X = np.array(X)
  for i in range(0, n):
    X[:, i]
    ret_df[f'Target-{n-i}'] = X[:, i]

  ret_df['Target'] = Y

  return ret_df
```

```
# Start day second time around: '2021-03-25'
windowed_df = windowed_df(df,
                                '1990-01-05',
                                '2022-07-29',
                                n=3)
windowed_df
```

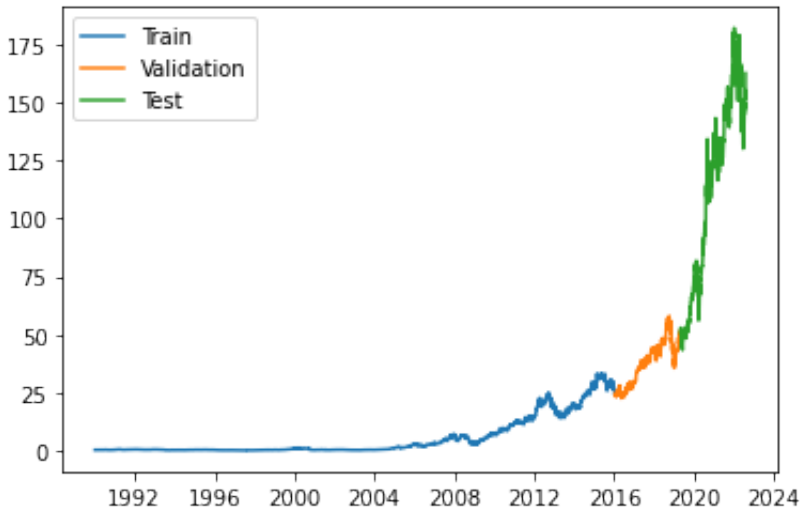
Out[15]:

	Target Date	Target-3	Target-2	Target-1	Target
0	1990-01-05	0.332589	0.334821	0.335938	0.337054
1	1990-01-08	0.334821	0.335938	0.337054	0.339286
2	1990-01-09	0.335938	0.337054	0.339286	0.335938
3	1990-01-10	0.337054	0.339286	0.335938	0.321429
4	1990-01-11	0.339286	0.335938	0.321429	0.308036
...	...	...	...	...	...
8200	2022-07-25	153.039993	155.350006	154.089996	152.949997
8201	2022-07-26	155.350006	154.089996	152.949997	151.600006
8202	2022-07-27	154.089996	152.949997	151.600006	156.789993
8203	2022-07-28	152.949997	151.600006	156.789993	157.350006
8204	2022-07-29	151.600006	156.789993	157.350006	162.509995

8205 rows × 5 columns

In [16]:
```
# Here this function will separate the data to train and validate t
def windowed_df_to_date_X_y(windowed_dataframe):
  df_as_np = windowed_dataframe.to_numpy()

  dates = df_as_np[:, 0]

  middle_matrix = df_as_np[:, 1:-1]
  X = middle_matrix.reshape((len(dates), middle_matrix.shape[1], 1)

  Y = df_as_np[:, -1]

  return dates, X.astype(np.float32), Y.astype(np.float32)

dates, X, y = windowed_df_to_date_X_y(windowed_df)

dates.shape, X.shape, y.shape
D80 = int(len(dates) * .8) # 80% data to train the model
D90 = int(len(dates) * .9) # The next 10% data to validate the mode

dates_train, X_train, y_train = dates[:D80], X[:D80], y[:D80]

dates_val, X_val, y_val = dates[D80:D90], X[D80:D90], y[D80:D90]
dates_test, X_test, y_test = dates[D90:], X[D90:], y[D90:]

plt.plot(dates_train, y_train)
plt.plot(dates_val, y_val)
plt.plot(dates_test, y_test)

plt.legend(['Train', 'Validation', 'Test'])
```

Out[16]: <matplotlib.legend.Legend at 0x7fc4f248f190>



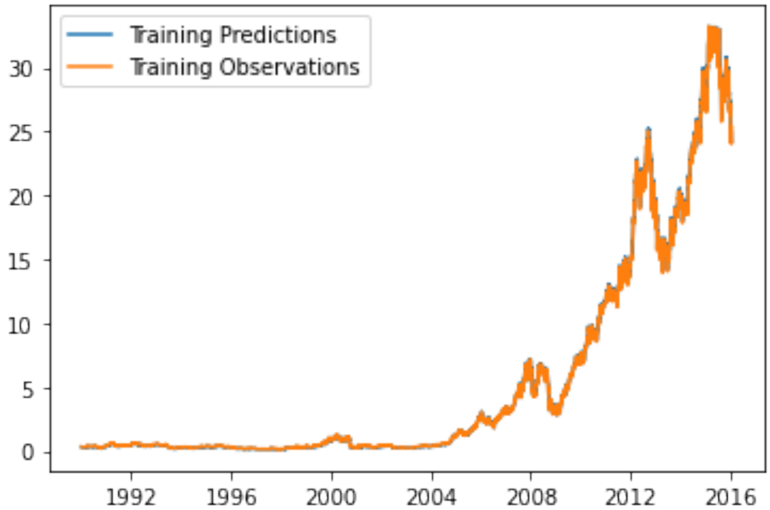
In [17]:
```
# Building our model by using the 'tensorflow' package.
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers

model = Sequential([layers.Input((3, 1)),
                    layers.LSTM(64),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(1)])

model.compile(loss='mse',
              optimizer=Adam(learning_rate=0.001),
              metrics=['mean_absolute_error'])

model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=
```

```
Epoch 1/100
206/206 [==============================] - 4s 8ms/step - loss: 24.
8224 - mean_absolute_error: 1.7050 - val_loss: 80.5861 - val_mean_
absolute_error: 6.5196
Epoch 2/100
206/206 [==============================] - 1s 3ms/step - loss: 0.0
454 - mean_absolute_error: 0.0970 - val_loss: 63.4219 - val_mean_a
bsolute_error: 5.6155
Epoch 3/100
206/206 [==============================] - 1s 3ms/step - loss: 0.0
527 - mean_absolute_error: 0.1027 - val_loss: 59.6900 - val_mean_a
bsolute_error: 5.3607
Epoch 4/100
206/206 [==============================] - 1s 3ms/step - loss: 0.0
415 - mean_absolute_error: 0.0934 - val_loss: 56.7811 - val_mean_a
bsolute_error: 5.2206
Epoch 5/100
206/206 [==============================] - 1s 3ms/step - loss: 0.0
399 - mean_absolute_error: 0.0916 - val_loss: 55.2770 - val_mean_a
```

In [22]:
```
# Train the model.
train_predictions = model.predict(X_train).flatten()

plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
plt.legend(['Training Predictions', 'Training Observations'])
```

```
206/206 [==============================] - 0s 1ms/step
```

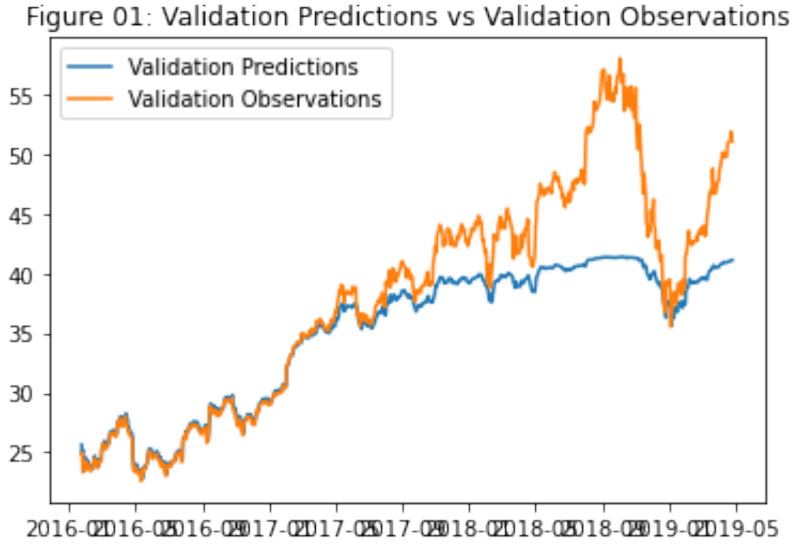
Out[22]:
```
<matplotlib.legend.Legend at 0x7fc4885c9dd0>
```



In [23]:
```
# Validate the model
val_predictions = model.predict(X_val).flatten()

plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.title("Figure 01: Validation Predictions vs Validation Observat
plt.legend(['Validation Predictions', 'Validation Observations'])
```

```
26/26 [==============================] - 0s 2ms/step
```

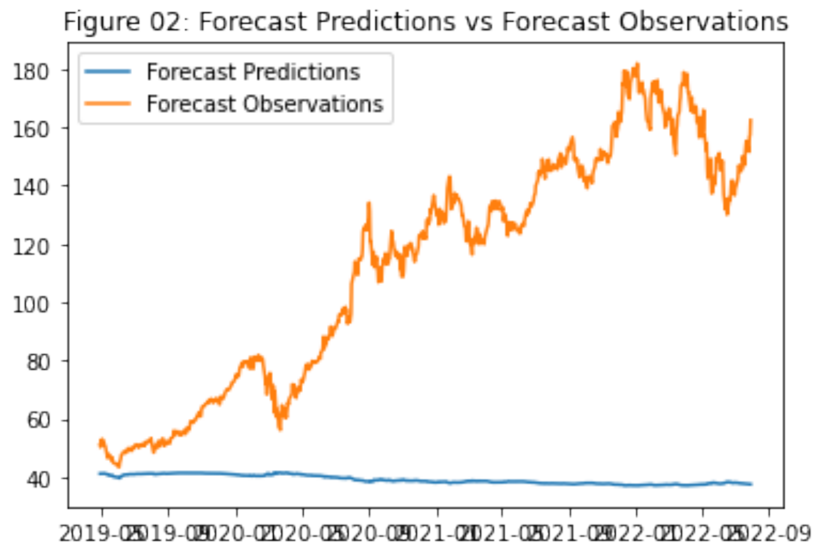
Out[23]: <matplotlib.legend.Legend at 0x7fc4887fd790>



In [24]:
```
# forcast and compare
test_predictions = model.predict(X_test).flatten()

plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.title("Figure 02: Forecast Predictions vs Forecast Observations
plt.legend(['Forecast Predictions', 'Forecast Observations'])
```

```
26/26 [==============================] - 0s 1ms/step
```

Out[24]: <matplotlib.legend.Legend at 0x7fc4887fd7d0>



In [2]:
```
# Import data
import pandas as pd
df = pd.read_csv("AAPL01.csv")
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1990-01-02	0.314732	0.334821	0.312500	0.332589	0.265325	183198400
1	1990-01-03	0.339286	0.339286	0.334821	0.334821	0.267106	207995200
2	1990-01-04	0.341518	0.345982	0.332589	0.335938	0.267997	221513600
3	1990-01-05	0.337054	0.341518	0.330357	0.337054	0.268887	123312000
4	1990-01-08	0.334821	0.339286	0.330357	0.339286	0.270668	101572800

In [3]:
```

# Taking only the closing price and date coumn
df = df[['Date', 'Close']]
df.head()
```

Out[3]:

	Date	Close
0	1990-01-02	0.332589
1	1990-01-03	0.334821
2	1990-01-04	0.335938
3	1990-01-05	0.337054
4	1990-01-08	0.339286

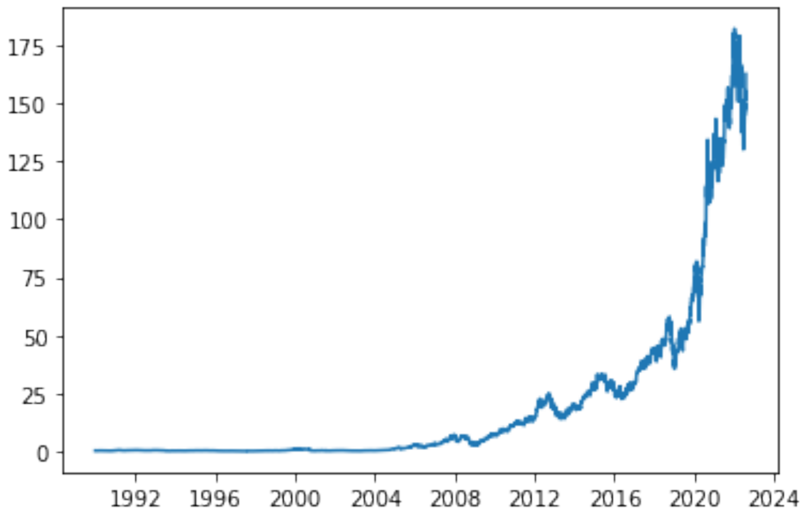
In [4]:
```
# Formetting date and index
import datetime

def datetime_str(s):
  split = s.split('-')
  year, month, day = int(split[0]), int(split[1]), int(split[2])
  return datetime.datetime(year=year, month=month, day=day)
datetime_object = datetime_str('1990-01-02')

df['Date'] = df['Date'].apply(datetime_str)
df['Date']
df.index = df.pop('Date')
df.head()
```

Out[4]:

Date	Close
1990-01-02	0.332589
1990-01-03	0.334821
1990-01-04	0.335938
1990-01-05	0.337054
1990-01-08	0.339286

In [5]:
```
import matplotlib.pyplot as plt

plt.plot(df.index, df['Close'])
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x7f0b60441e90>]
```



In [6]:
```
# This function will give us output on the based of the input.
# Target 1, Target 2, Target 3 are the previous value respectively
# Target is the predicted outut.
```

```
import numpy as np

def windowed_df(dataframe, first_date_str, last_date_str, n=3):
  first_date = datetime_str(first_date_str)
  last_date  = datetime_str(last_date_str)

  target_date = first_date

  dates = []
  X, Y = [], []

  last_time = False
  while True:
    df_subset = dataframe.loc[:target_date].tail(n+1)

    if len(df_subset) != n+1:
      print(f'Error: Window of size {n} is too large for date {targ
      return

    values = df_subset['Close'].to_numpy()
    x, y = values[:-1], values[-1]

    dates.append(target_date)
    X.append(x)
    Y.append(y)

    next_week = dataframe.loc[target_date:target_date+datetime.time
    next_datetime_str = str(next_week.head(2).tail(1).index.values[
    next_date_str = next_datetime_str.split('T')[0]
    year_month_day = next_date_str.split('-')
    year, month, day = year_month_day
    next_date = datetime.datetime(day=int(day), month=int(month), y

    if last_time:
      break

    target_date = next_date

    if target_date == last_date:
      last_time = True

  ret_df = pd.DataFrame({})
  ret_df['Target Date'] = dates

  X = np.array(X)
  for i in range(0, n):
    X[:, i]
    ret_df[f'Target-{n-i}'] = X[:, i]

  ret_df['Target'] = Y

  return ret_df
```

```
# Start day second time around: '2021-03-25'
windowed_df = windowed_df(df,
                          '2020-01-01',
                          '2022-07-29',
                          n=3)
windowed_df
```

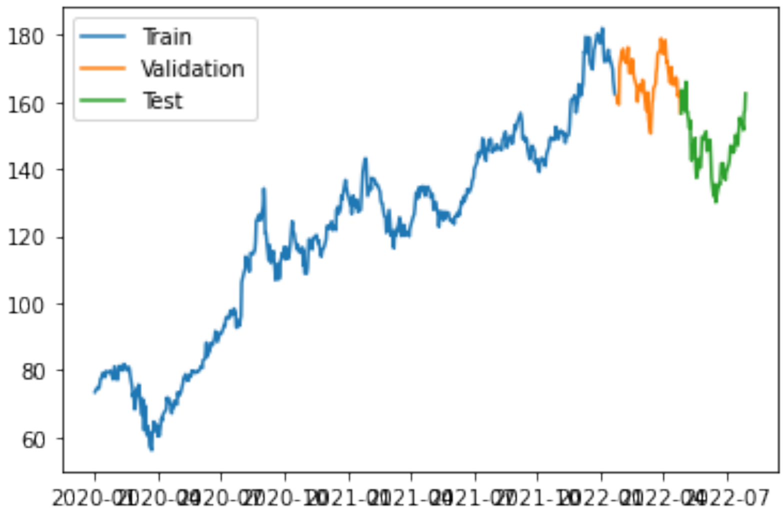
Out[6]:

	Target Date	Target-3	Target-2	Target-1	Target
0	2020-01-01	72.477501	72.449997	72.879997	73.412498
1	2020-01-03	72.879997	73.412498	75.087502	74.357498
2	2020-01-06	73.412498	75.087502	74.357498	74.949997
3	2020-01-07	75.087502	74.357498	74.949997	74.597504
4	2020-01-08	74.357498	74.949997	74.597504	75.797501
...	...	...	...	...	...
644	2022-07-25	153.039993	155.350006	154.089996	152.949997
645	2022-07-26	155.350006	154.089996	152.949997	151.600006
646	2022-07-27	154.089996	152.949997	151.600006	156.789993
647	2022-07-28	152.949997	151.600006	156.789993	157.350006
648	2022-07-29	151.600006	156.789993	157.350006	162.509995

649 rows × 5 columns

In [7]:
```
# Here this function will separate the data to train and validate t
def windowed_df_to_date_X_y(windowed_dataframe):
  df_as_np = windowed_dataframe.to_numpy()

  dates = df_as_np[:, 0]

  middle_matrix = df_as_np[:, 1:-1]
  X = middle_matrix.reshape((len(dates), middle_matrix.shape[1], 1)

  Y = df_as_np[:, -1]

  return dates, X.astype(np.float32), Y.astype(np.float32)

dates, X, y = windowed_df_to_date_X_y(windowed_df)

dates.shape, X.shape, y.shape
D80 = int(len(dates) * .8) # 80% data to train the model
D90 = int(len(dates) * .9) # The next 10% data to validate the mode

dates_train, X_train, y_train = dates[:D80], X[:D80], y[:D80]

dates_val, X_val, y_val = dates[D80:D90], X[D80:D90], y[D80:D90]
dates_test, X_test, y_test = dates[D90:], X[D90:], y[D90:]

plt.plot(dates_train, y_train)
plt.plot(dates_val, y_val)
plt.plot(dates_test, y_test)

plt.legend(['Train', 'Validation', 'Test'])
```

Out[7]: <matplotlib.legend.Legend at 0x7f0b5ff0d610>



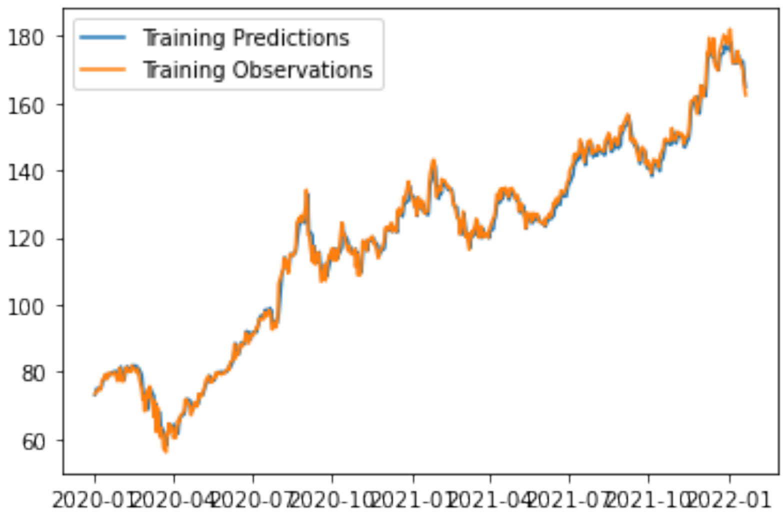
In [8]:
```
# Building our model by using the 'tensorflow' package.
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers

model = Sequential([layers.Input((3, 1)),
                    layers.LSTM(64),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(32, activation='relu'),
                    layers.Dense(1)])

model.compile(loss='mse',
              optimizer=Adam(learning_rate=0.001),
              metrics=['mean_absolute_error'])

model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=
```

```
14 - mean_absolute_error: 3.1580 - val_loss: 92.8608 - val_mean_ab
solute_error: 8.5644
Epoch 16/100
17/17 [==============================] - 0s 7ms/step - loss: 16.76
46 - mean_absolute_error: 2.7127 - val_loss: 49.2156 - val_mean_ab
solute_error: 5.8554
Epoch 17/100
17/17 [==============================] - 0s 8ms/step - loss: 13.24
07 - mean_absolute_error: 2.4800 - val_loss: 39.6247 - val_mean_ab
solute_error: 5.2672
Epoch 18/100
17/17 [==============================] - 0s 8ms/step - loss: 11.52
27 - mean_absolute_error: 2.4061 - val_loss: 28.6016 - val_mean_ab
solute_error: 4.4436
Epoch 19/100
17/17 [==============================] - 0s 7ms/step - loss: 9.777
2 - mean_absolute_error: 2.2274 - val_loss: 29.9593 - val_mean_abs
olute_error: 4.5878
Epoch 20/100
17/17 [==============================] - 0s 8ms/step - loss: 9.766
```

In [9]:
```
# Train the model.
train_predictions = model.predict(X_train).flatten()

plt.plot(dates_train, train_predictions)
plt.plot(dates_train, y_train)
plt.legend(['Training Predictions', 'Training Observations'])
```

```
17/17 [==============================] - 1s 3ms/step
```

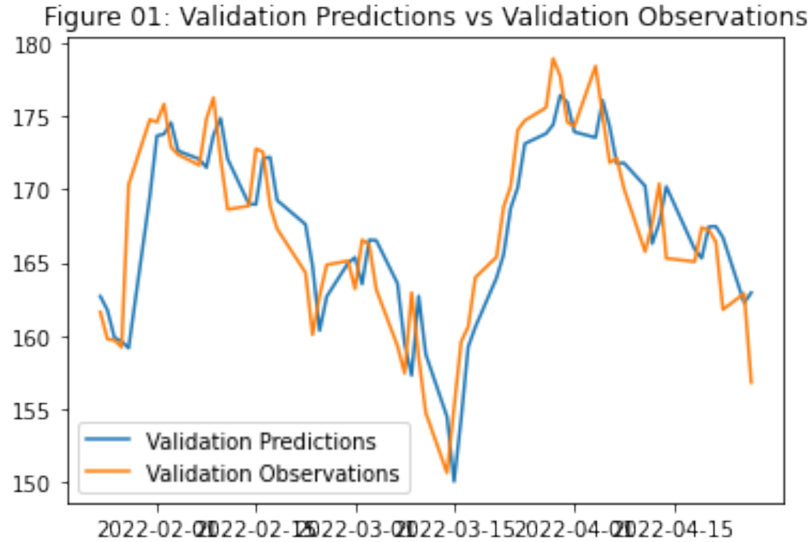
Out[9]: <matplotlib.legend.Legend at 0x7f0af63ac710>



In [10]:
```
# Validate the model
val_predictions = model.predict(X_val).flatten()

plt.plot(dates_val, val_predictions)
plt.plot(dates_val, y_val)
plt.title("Figure 01: Validation Predictions vs Validation Observat
plt.legend(['Validation Predictions', 'Validation Observations'])
```

```
3/3 [==============================] - 0s 6ms/step
```

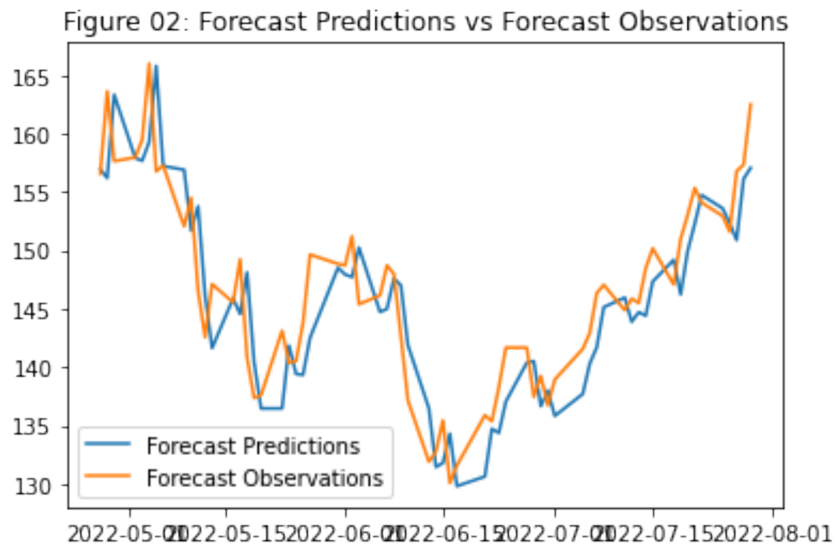
Out[10]:
```
<matplotlib.legend.Legend at 0x7f0af6277f50>
```



In [11]:
```
# forcast and compare
test_predictions = model.predict(X_test).flatten()

plt.plot(dates_test, test_predictions)
plt.plot(dates_test, y_test)
plt.title("Figure 02: Forecast Predictions vs Forecast Observations
plt.legend(['Forecast Predictions', 'Forecast Observations'])
```

```
3/3 [==============================] - 0s 4ms/step
```

Out[11]: <matplotlib.legend.Legend at 0x7f0af6288390>



In [ ]: