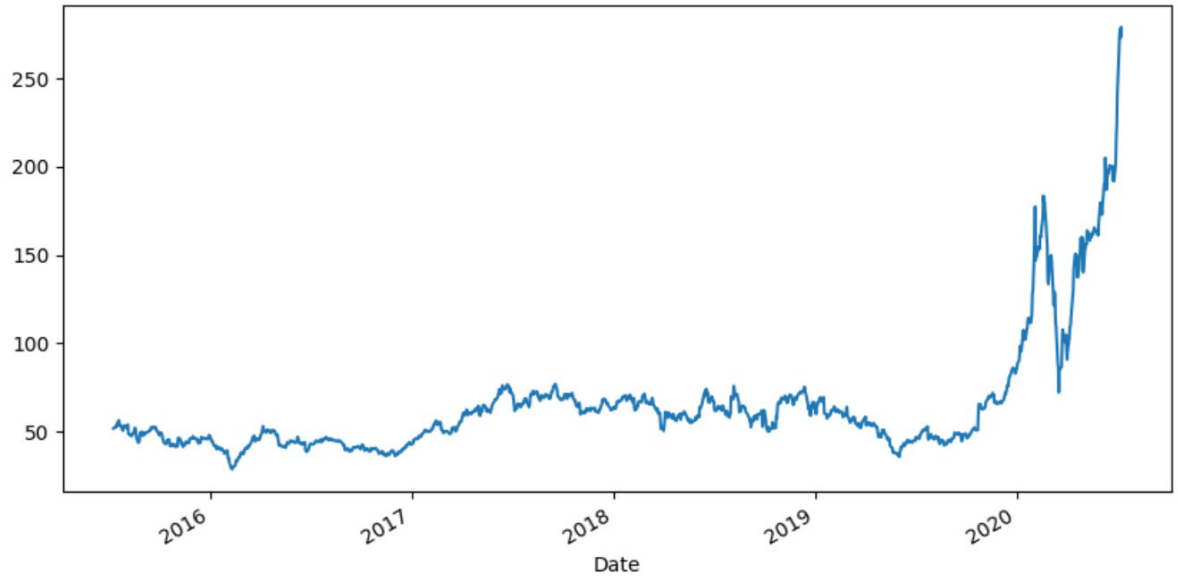
TSLA stock prediction Updated

1. Load the data and inspect them



This is the visulization of the whole dataset, and we will use a part of it. Also, we will use closing price to do prediction.

2. Split the date

```
[5]  #test data starts from 04/01/2020 to 07/09/2020
     test_size = (data['Date'] >= '2020-04-01') & (data['Date'] <= '2020-07-09')
     count = data.loc[test_size].shape[0]

     print(count)
```

```
69
```

```
[6]  #choose the training set as the updated requirement, one year and 3 months
     #the historical data is 3 months (before 04/01/2020)
     train_size = (data['Date'] >= '2019-01-01') & (data['Date'] <= '2020-03-31')
     count2 = data.loc[train_size].shape[0]

     print(count2)
```

```
314
```

```
#define training and test sets
training_set = data_normalized[-(69+314):-69,:]
test_set = data_normalized[-69:,:]

training_set.shape , test_set.shape
```

The test set start from 04/01/2020 to 07/09/2020;

The training set starts from 01/01/2019 to 03/31/2020

3. Different time lag (Time lag=1)

```
def input_features(data, lag):
    X, Y = [], []
    for i in range(len(data) -lag):
        X.append(data[i:(i+1), 0])
        Y.append(data[i + lag, 0])
    return np.array(X), np.array(Y)
```

```
lag=1
```

```
X_train, Y_train = input_features(training_set, lag)
X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
```

```
X_test_initial = training_set[-1:].reshape(1, 1,1)
first_prediction = model1.predict(X_test_initial)

  # Assuming X_test_initial and first_prediction are already defined and sha
X_test = [X_test_initial.flatten()]  # Start with the initial test set value
Y_test = [first_prediction.flatten()]  # Start with the initial prediction

for i in range(1, len(test_set)):
    # Reshape the last prediction to fit the model's expected input format
  current_input = np.array([Y_test[-1]]).reshape(1, 1, 1)
  current_prediction = model1.predict(current_input).flatten()

  X_test.append(current_input)
  Y_test.append(current_prediction)

Y_test = np.array(Y_test).reshape(-1, )
X_test = np.array(X_test).reshape(-1, 1, 1)
```

3. Different time lag (Time lag=7 or 14)

```
[159] X_test_initial = training_set[-8:-1,:].reshape(1, lag,1)
```

```
[160] X_test_initial.shape
```

```
(1, 7, 1)
```

```
[161] first_prediction = model1.predict(np.array(X_test_initial).reshape(-1,7))
```

```
1/1 [==============================] - 0s 27ms/step
```

```
print(first_prediction)
```

```
[[[0.17132977]
  [0.21127424]
  [0.22984731]
  [0.25036794]
  [0.26689556]
  [0.26943368]
  [0.25739658]]]
```

```
[163] first_prediction = [first_prediction.flatten()]
```

```
[165] Y_test = [first_prediction[-1]]

for i in range(1, 9): #discard the last 69-9*7 nodes

    last_values = Y_test[-1]

    current_input = last_values.reshape(1, 7, 1)

    current_prediction = model1.predict(np.array(current_input).reshape(-1,7))

    Y_test.append(current_prediction[-1])
```

```
] first_prediction = model1.predict(np.array(X_test_initial).reshape(-1,14))
```

```
1/1 [==============================] - 2s 2s/step
```

```
] print(first_prediction)
```

```
[[[0.1434614 ]
  [0.18718675]
  [0.205569  ]
  [0.2106829 ]
  [0.21174264]
  [0.21434367]
  [0.22075137]
  [0.23076144]
  [0.24277344]
  [0.2548477 ]
  [0.26541516]
  [0.2735335 ]
  [0.27884057]
  [0.28138602]]]
```

```
] first_prediction = [first_prediction.flatten()]
```

```
] Y_test = [first_prediction[-1]]

for i in range(1,4 ): #discard the last 69-4*14 nodes

    last_values = Y_test[-1]

    current_input = last_values.reshape(1, 14, 1)

    current_prediction = model1.predict(np.array(current_input).reshape(-1,14))

    Y_test.append(current_prediction[-1])
```

4. Build LSTM model

```
def lstm(lstm_layer=6):
  model = Sequential()

  model.add(LSTM(units = 50, activation = 'relu', return_sequences=True,
                 input_shape = (X_train.shape[1], X_train.shape[2])))
  model.add(Dropout(0.2))

  for layer in range(lstm_layer-1):
    model.add(LSTM(50, return_sequences= False))
    model.add(Dropout(0.2))

    model.add(Dense(1))

    return model

model1=lstm(lstm_layer=6)
model2=lstm(lstm_layer=6)
```

```
def lstm2(lstm_layer=6):
  model = Sequential()

  model.add(LSTM(units = 50, activation = 'relu', return_sequences=True,
                 input_shape = (X_train.shape[1], X_train.shape[2])))
  model.add(Dropout(0.3))

  for layer in range(lstm_layer-1):
    model.add(LSTM(50, return_sequences= False))
    model.add(Dropout(0.3))

    model.add(Dense(1))

    return model

model3=lstm2(lstm_layer=6)
model4=lstm2(lstm_layer=4)
```

```
models={'drop_0.2_4_h_layer': model1, 'drop_0.2_6_h_layer': model2,
        'drop_0.3_4_h_layer': model3,'drop_0.3_6_h_layer': model4}
```

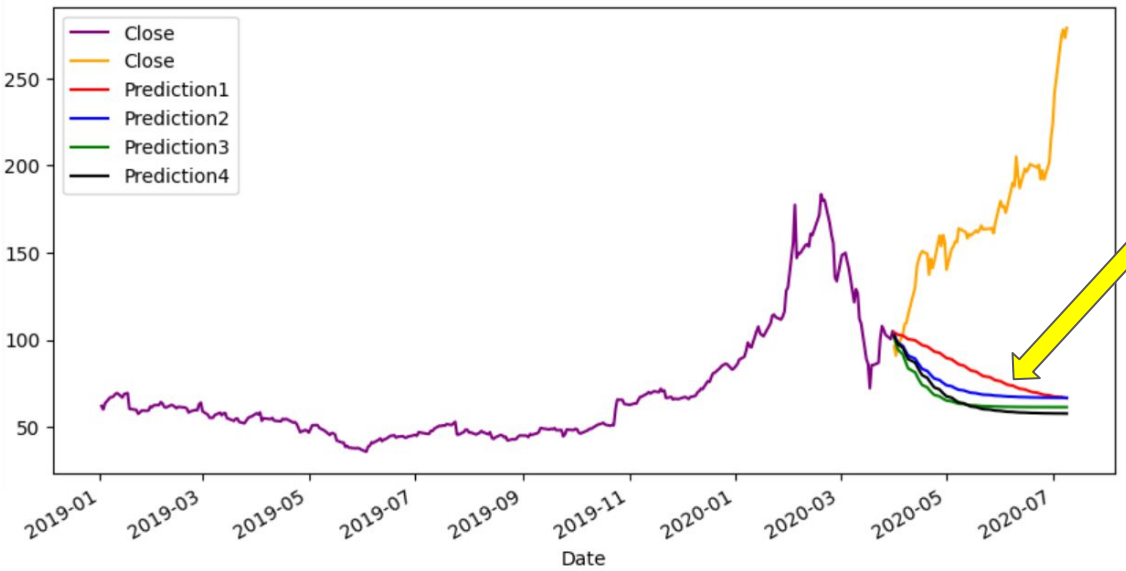
```
model1.compile(optimizer = 'adam', loss = 'mse' , metrics="mean_absolute_error")
model1.fit(X_train, Y_train, epochs = 100,batch_size = 20, verbose = 1, shuffle = False)
```

Make comparisons of different performance of Drop out rate and the number of hidden layers: Drop out rate= 0.2 or 0.3; Hidden layers= 4 or 6

5. Results(1)

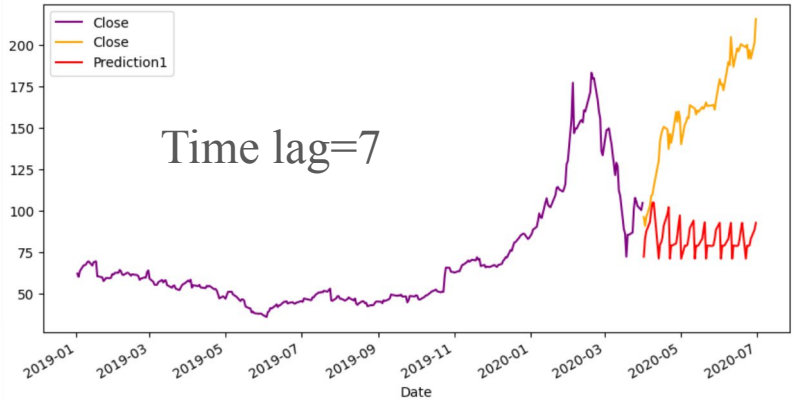
```
in the model 1, the test MAE is 89.37331106495557
in the model 2, the test MAE is 97.66939680961984
in the model 3, the test MAE i  104.21035317145748
in the model 4, the test MAE s 104.44902097458814
```

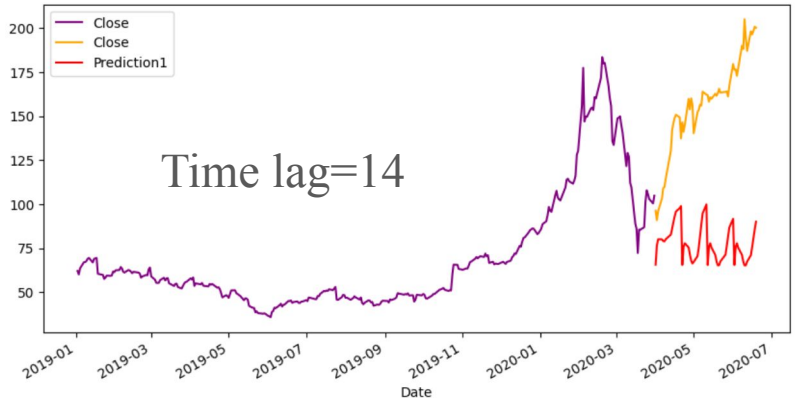
Only apply comparisons of different performance in time lag=1: The performance of the four models are similar and the performance of prediction 1, drop out rate=0.2, hidden layer=4 is perferred best

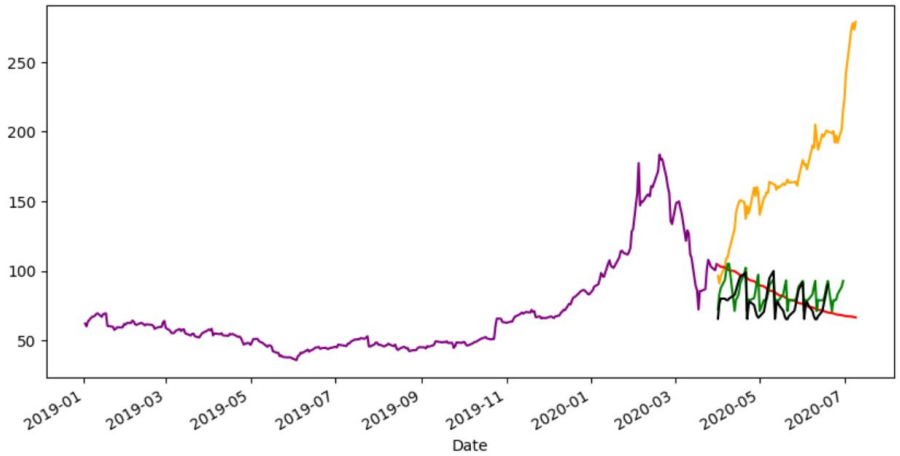


Time lag=1

5. Results(2)


Close
Close
Prediction1
Time lag=7
Date


Close
Close
Prediction1
Time lag=14
Date


Date