```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'tesla-stock-price:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F3161600%2F5475006%2Fbundle%2Farchive.zip%

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading tesla-stock-price, 44065 bytes compressed
[==================================================] 44065 bytes downloaded
```

```
Downloaded and uncompressed: tesla-stock-price
Data source import complete.
```

Load Necessary Packages

```
import pandas as pd
from sklearn import preprocessing
import numpy as np

import keras
import tensorflow as tf
from keras.models import Model
from keras.layers import Dense, Dropout, LSTM, Input, Activation, concatenate
from keras import optimizers
from keras.callbacks import EarlyStopping

np.random.seed(4)

from tensorflow.random import set_seed
set_seed(4)
```

Use past 50 days closing price to predict next day closing price

```
history_points = 50
```

Load dataset

```
data = pd.read_csv('/kaggle/input/tesla-stock-price/TSLA_daily.csv')
data.head()
```

	date	1. open	2. high	3. low	4. close	5. volume
0	2020-06-19	1012.78	1015.97	991.3400	1000.90	8502314.0
1	2020-06-18	1003.00	1019.20	994.4708	1003.96	9751936.0
2	2020-06-17	987.71	1005.00	982.5710	991.79	9890800.0
3	2020-06-16	1011.85	1012.88	962.3900	982.13	14051078.0
4	2020-06-15	917.79	998.84	908.5000	990.90	15697178.0

Next steps: Generate code with data View recommended plots

Reverse the time series order so that Last days comes at the last

```
data = data.iloc[::-1]
data.reset_index(drop = True, inplace=True)
data.head()
```

	date	1. open	2. high	3. low	4. close	5. volume
0	2020-06-19	1012.78	1015.97	991.3400	1000.90	8502314.0
1	2020-06-18	1003.00	1019.20	994.4708	1003.96	9751936.0
2	2020-06-17	987.71	1005.00	982.5710	991.79	9890800.0
3	2020-06-16	1011.85	1012.88	962.3900	982.13	14051078.0
4	2020-06-15	917.79	998.84	908.5000	990.90	15697178.0

Next steps: Generate code with data View recommended plots

Drop Date Column

```
data = data.drop('date', axis=1)
```

Perform MinMax Scalar Normalization of the time series using sklearn preprocessing package

```
data_normaliser = preprocessing.MinMaxScaler()
data_normalised = data_normaliser.fit_transform(data)
```

Using the last {history_points} open high low close volume data points, predict the next close value

```
ohlcv_histories_normalised = np.array([data_normalised[i  : i + history_points].copy() for i in range(len(data_normalised) - history_points)

next_day_close_values_normalised = np.array([data_normalised[:,3][i + history_points].copy() for i in range(len(data_normalised) - history_p

next_day_close_values_normalised = np.expand_dims(next_day_close_values_normalised, -1)

next_day_close_values = np.array([data.to_numpy()[:,3][i + history_points].copy() for i in range(len(data) - history_points)])
next_day_close_values = next_day_close_values.reshape(next_day_close_values.shape[0], 1)

y_normaliser = preprocessing.MinMaxScaler()
y_normaliser.fit(next_day_close_values)
```

```
▾ MinMaxScaler
MinMaxScaler()
```

Traing-test split in the ratio of 9:1

```
test_split = 0.9 # the percent of data to be used for training
n = int(ohlcv_histories_normalised.shape[0] * test_split)

# splitting the dataset up into train and test sets

x_train = ohlcv_histories_normalised[:n]
y_train = next_day_close_values_normalised[:n]

x_test = ohlcv_histories_normalised[n:]
y_test = next_day_close_values_normalised[n:]
```

Unscaled values for y-train and y-test will be used for calculating the model's RMSE later

```
unscaled_y_train = next_day_close_values[:n]
unscaled_y_test = next_day_close_values[n:]
```

Feature Engineering - Simple Moving Average for the closing prices is used as an additional input feature in the LSTM model.

```
technical_indicators = []

for his in ohlcv_histories_normalised:
  # since we are using his[3] we are taking the SMA of the closing price
  sma = np.mean(his[:,3])
  technical_indicators.append(np.array([sma]))

technical_indicators = np.array(technical_indicators)

tech_ind_scaler = preprocessing.MinMaxScaler()
technical_indicators_normalised = tech_ind_scaler.fit_transform(technical_indicators)
technical_indicators_normalised.shape
```

```
(2462, 1)
```

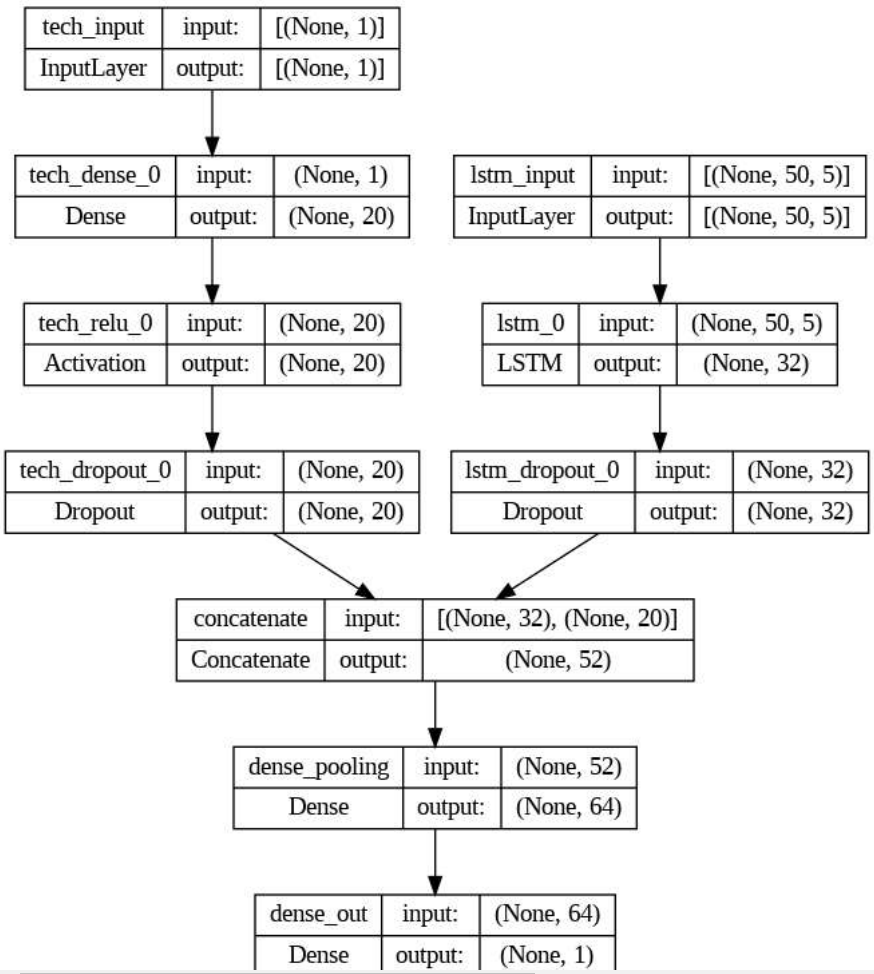
```
tech_ind_train = technical_indicators_normalised[:n]
tech_ind_test = technical_indicators_normalised[n:]
```

Time Series Forecasting using LSTM

```
# define two sets of inputs
lstm_input = Input(shape=(history_points, 5), name='lstm_input')
dense_input = Input(shape=(technical_indicators.shape[1],), name='tech_input')
```

```
dense_input = Input(shape=(technical_indicators.shape[1],), name='tech_input')

# the first branch operates on the first input
x = LSTM(32, name='lstm_0')(lstm_input)
x = Dropout(0.2, name='lstm_dropout_0')(x)
lstm_branch = Model(inputs=lstm_input, outputs=x)

# the second branch opreates on the second input
y = Dense(20, name='tech_dense_0')(dense_input)
y = Activation("relu", name='tech_relu_0')(y)
y = Dropout(0.2, name='tech_dropout_0')(y)
technical_indicators_branch = Model(inputs=dense_input, outputs=y)

# combine the output of the two branches
combined = concatenate([lstm_branch.output, technical_indicators_branch.output], name='concatenate')

z = Dense(64, activation="sigmoid", name='dense_pooling')(combined)
z = Dense(1, activation="linear", name='dense_out')(z)

# our model will accept the inputs of the two branches and then output a single value
model = Model(inputs=[lstm_branch.input, technical_indicators_branch.input], outputs=z)

adam = optimizers.Adam(lr=0.0005)
model.compile(optimizer=adam,
              loss='mse')

from keras.utils import plot_model
plot_model(model, to_file='model.png', show_shapes=True)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use th
```




```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)
history = model.fit(x=[x_train, tech_ind_train], y=y_train, batch_size=32, epochs=300, shuffle=True, validation_split=0.2, callbacks=[es])
```

```
Epoch 88/300
56/56 [==============================] - 1s 23ms/step - loss: 2.8585e-04 - val_loss: 3.4304e-04
Epoch 89/300
56/56 [==============================] - 1s 22ms/step - loss: 2.4332e-04 - val_loss: 1.1658e-06
Epoch 90/300
56/56 [==============================] - 1s 23ms/step - loss: 1.9015e-04 - val_loss: 5.5133e-05
Epoch 91/300
56/56 [==============================] - 1s 23ms/step - loss: 1.8911e-04 - val_loss: 1.8470e-05
Epoch 92/300
56/56 [==============================] - 1s 23ms/step - loss: 2.3598e-04 - val_loss: 7.2434e-05
Epoch 93/300
56/56 [==============================] - 1s 22ms/step - loss: 1.7883e-04 - val_loss: 8.9438e-07
Epoch 94/300
56/56 [==============================] - 1s 22ms/step - loss: 1.6726e-04 - val_loss: 3.9484e-05
Epoch 95/300
56/56 [==============================] - 2s 28ms/step - loss: 2.9475e-04 - val_loss: 5.5155e-05
Epoch 96/300
56/56 [==============================] - 2s 34ms/step - loss: 1.9691e-04 - val_loss: 2.5755e-05
Epoch 97/300
56/56 [==============================] - 1s 25ms/step - loss: 2.1780e-04 - val_loss: 2.7433e-04
Epoch 98/300
56/56 [==============================] - 1s 23ms/step - loss: 1.7949e-04 - val_loss: 1.9948e-04
Epoch 99/300
56/56 [==============================] - 1s 23ms/step - loss: 1.8141e-04 - val_loss: 4.5188e-05
Epoch 100/300
56/56 [==============================] - 1s 23ms/step - loss: 1.7592e-04 - val_loss: 8.3190e-05
Epoch 101/300
56/56 [==============================] - 1s 22ms/step - loss: 1.8010e-04 - val_loss: 2.0334e-05
Epoch 102/300
56/56 [==============================] - 1s 23ms/step - loss: 2.2684e-04 - val_loss: 2.0176e-05
Epoch 103/300
56/56 [==============================] - 2s 29ms/step - loss: 2.0647e-04 - val_loss: 1.4275e-05
Epoch 104/300
56/56 [==============================] - 2s 44ms/step - loss: 1.4286e-04 - val_loss: 1.5122e-06
Epoch 105/300
56/56 [==============================] - 2s 35ms/step - loss: 1.7961e-04 - val_loss: 2.4010e-05
Epoch 106/300
56/56 [==============================] - 1s 22ms/step - loss: 1.9650e-04 - val_loss: 6.9224e-05
Epoch 107/300
56/56 [==============================] - 1s 24ms/step - loss: 1.8117e-04 - val_loss: 7.5077e-06
Epoch 108/300
56/56 [==============================] - 1s 23ms/step - loss: 2.7585e-04 - val_loss: 1.4102e-04
Epoch 109/300
56/56 [==============================] - 1s 23ms/step - loss: 2.0648e-04 - val_loss: 9.6288e-06
Epoch 110/300
56/56 [==============================] - 1s 22ms/step - loss: 1.9226e-04 - val_loss: 3.9367e-05
Epoch 111/300
56/56 [==============================] - 1s 22ms/step - loss: 1.7749e-04 - val_loss: 8.1708e-06
Epoch 112/300
56/56 [==============================] - 1s 23ms/step - loss: 1.8459e-04 - val_loss: 2.7174e-05
Epoch 113/300
56/56 [==============================] - 2s 28ms/step - loss: 1.7581e-04 - val_loss: 6.1981e-05
Epoch 113: early stopping
```

```
evaluation = model.evaluate([x_test, tech_ind_test], y_test)
print(evaluation)
```

```
8/8 [==============================] - 0s 12ms/step - loss: 6.4515e-05
6.451529043260962e-05
```

Calculating Train RMSE

```
y_predicted_train = model.predict([x_train, tech_ind_train])
y_predicted_train = y_normaliser.inverse_transform(y_predicted_train)

real_mse_train = np.mean(np.square(unscaled_y_train - y_predicted_train))
print("Train RMSE = {}".format(real_mse_train))
```

```
70/70 [==============================] - 1s 7ms/step
Train RMSE = 739.2374257179218
```

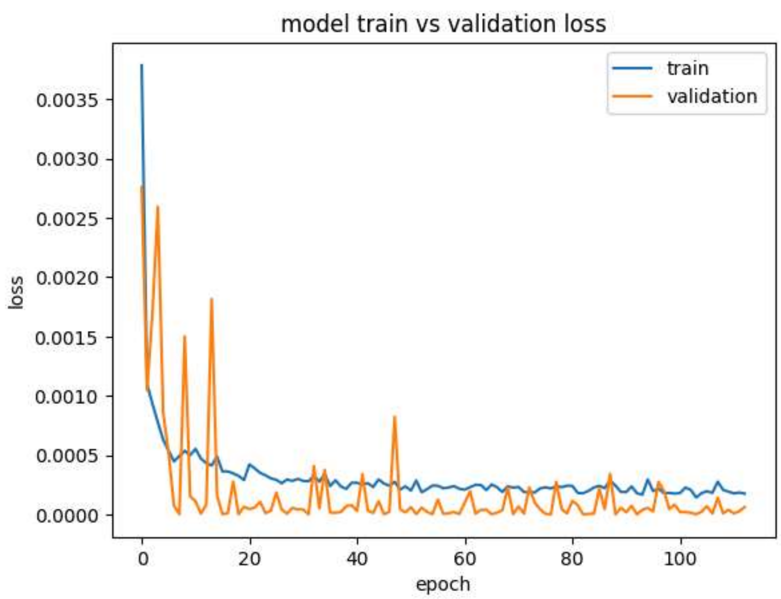
Calculating Test RMSE

```
y_test_predicted = model.predict([x_test, tech_ind_test])
y_test_predicted = y_normaliser.inverse_transform(y_test_predicted)

real_mse_test = np.mean(np.square(unscaled_y_test - y_test_predicted))
print("Test RMSE = {}".format(real_mse_test))
```

```
8/8 [==============================] - 0s 8ms/step
Test RMSE = 66.64677774946736
```

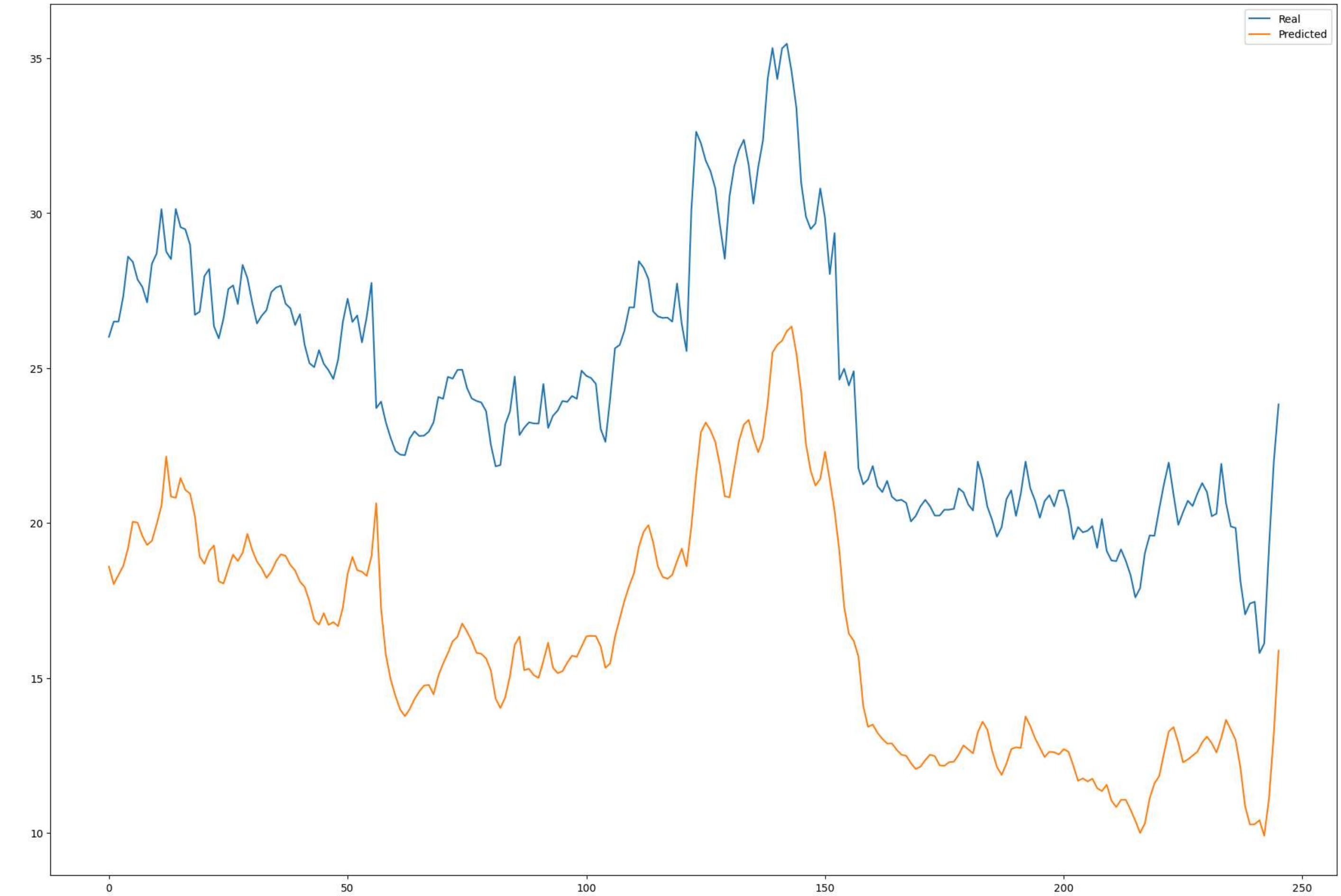
Train Vs Validation Loss

```
from matplotlib import pyplot

pyplot.plot(history.history['loss'])
pyplot.plot(history.history['val_loss'])
pyplot.title('model train vs validation loss')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train', 'validation'], loc='upper right')
pyplot.show()
```



Real Vs Predicted Time Series

```
import matplotlib.pyplot as plt
plt.gcf().set_size_inches(22, 15, forward=True)

start = 0
end = -1

real = plt.plot(unscaled_y_test[start:end], label='real')
pred = plt.plot(y_test_predicted[start:end], label='predicted')

plt.legend(['Real', 'Predicted'])

plt.show()
```


Real
Predicted
35
30
25
20
15
10
0
50
100
150
200
250