```
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-data-from-2010-to-2020:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F500872%2F927894%2Fbundle%

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-data-from-2010-to-2020, 47194 bytes compressed
[==================================================] 47194 bytes downloaded
Downloaded and uncompressed: tesla-stock-data-from-2010-to-2020
Data source import complete.
```

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM,Dropout
from tensorflow.keras.callbacks import ModelCheckpoint,EarlyStopping

import warnings
warnings.filterwarnings("ignore")

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="3"
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)


data=pd.read_csv("../input/tesla-stock-data-from-2010-to-2020/TSLA.csv")


data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

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

```
def check_df(dataframe,head=5):
    print("#################### Shape #################### ")
    print(dataframe.shape)
    print("#################### Types #################### ")
    print(dataframe.dtypes)
    print("#################### Head #################### ")
    print(dataframe.head(head))
    print("#################### Tail #################### ")
    print(dataframe.dtypes)
    print("#################### NA #################### ")
    print(dataframe.isnull().sum())
    print("#################### Quantiles #################### ")
    print(dataframe.quantile([0,0.5,0.50,0.95,0.99,1]).T)


check_df(data)
```

```
#################### Shape ####################
(2416, 7)
#################### Types ####################
Date          object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### Head ####################
         Date       Open   High        Low      Close  Adj Close    Volume
0  2010-06-29  19.000000  25.00  17.540001  23.889999  23.889999  18766300
1  2010-06-30  25.790001  30.42  23.299999  23.830000  23.830000  17187100
2  2010-07-01  25.000000  25.92  20.270000  21.959999  21.959999   8218800
3  2010-07-02  23.000000  23.10  18.709999  19.200001  19.200001   5139800
4  2010-07-06  20.000000  20.00  15.830000  16.110001  16.110001   6866900
#################### Tail ####################
Date          object
Open         float64
High         float64
Low          float64
```

```
Close        float64
Adj Close    float64
Volume         int64
dtype: object
#################### NA ####################
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
#################### Quantiles ####################
                    0.00          0.50          0.50          0.95  \
Open           16.139999  2.130350e+02  2.130350e+02  3.519100e+02
High           16.629999  2.167450e+02  2.167450e+02  3.567500e+02
Low            14.980000  2.088700e+02  2.088700e+02  3.460250e+02
Close          15.800000  2.129600e+02  2.129600e+02  3.511675e+02
Adj Close      15.800000  2.129600e+02  2.129600e+02  3.511675e+02
Volume     118500.000000  4.578400e+06  4.578400e+06  1.476090e+07

                   0.99          1.00
Open       4.235790e+02  6.736900e+02
High       4.284705e+02  7.861400e+02
Low        4.122865e+02  6.735200e+02
Close      4.243455e+02  7.800000e+02
Adj Close  4.243455e+02  7.800000e+02
Volume     2.459959e+07  4.706500e+07
```

```
data["Date"]=pd.to_datetime(data["Date"])
```

```
data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	20.000000	20.00	15.830000	16.110001	16.110001	6866900

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

```
tesla_data=data[["Date","Close"]]
```

```
tesla_data.head()
```

	Date	Close
0	2010-06-29	23.889999
1	2010-06-30	23.830000
2	2010-07-01	21.959999
3	2010-07-02	19.200001
4	2010-07-06	16.110001

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

```
print("Min. Tarih:",tesla_data["Date"].min())
print("Max. Tarih:",tesla_data["Date"].max())
```

```
Min. Tarih: 2010-06-29 00:00:00
Max. Tarih: 2020-02-03 00:00:00
```

```
tesla_data.index=tesla_data["Date"]
```

```
tesla_data
```

Date	Date	Close
2010-06-29	2010-06-29	23.889999
2010-06-30	2010-06-30	23.830000
2010-07-01	2010-07-01	21.959999
2010-07-02	2010-07-02	19.200001
2010-07-06	2010-07-06	16.110001
...	...	...
2020-01-28	2020-01-28	566.900024
2020-01-29	2020-01-29	580.989990
2020-01-30	2020-01-30	640.809998
2020-01-31	2020-01-31	650.570007
2020-02-03	2020-02-03	780.000000

2416 rows × 2 columns

```
tesla_data.drop("Date",axis=1,inplace=True)
```

```
tesla_data
```

Date	Close
2010-06-29	23.889999
2010-06-30	23.830000
2010-07-01	21.959999
2010-07-02	19.200001
2010-07-06	16.110001
...	...
2020-01-28	566.900024
2020-01-29	580.989990
2020-01-30	640.809998
2020-01-31	650.570007
2020-02-03	780.000000

2416 rows × 1 columns

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

```
result_data=tesla_data.copy()
```

```
plt.figure(figsize=(12,6))
plt.plot(tesla_data["Close"],color="blue");
plt.ylabel("Stock Price")
plt.title("Tesla Stock Price")
plt.xlabel("Time")
plt.show()
```


```
tesla_data=tesla_data.values
```

```
tesla_data[0:5]
```

```
     array([[23.889999],
            [23.83    ],
            [21.959999],
            [19.200001],
            [16.110001]])
```

```
tesla_data=tesla_data.astype("float32")
```

```
def split_data(dataframe,test_size):
    pos=int(round(len(dataframe)*(1-test_size)))
    train=dataframe[:pos]
    test=dataframe[pos:]
    return train,test,pos
```

```
train,test,pos=split_data(tesla_data,0.20)
```

```
print(train.shape,test.shape)
```

```
scaler_train=MinMaxScaler(feature_range=(0,1))
```

```
train=scaler_train.fit_transform(train)
```

```
scaler_test=MinMaxScaler(feature_range=(0,1))
```

```
test=scaler_test.fit_transform(test)
```

```
train[0:5]
```

```
     array([[0.02191224],
            [0.02174973],
            [0.01668472],
            [0.0092091 ],
            [0.00083966]], dtype=float32)
```

```
test[0:5]
```

```
     array([[0.25685903],
            [0.24829045],
            [0.25511202],
            [0.24978784],
            [0.2465767 ]], dtype=float32)


def create_features(data,lookback):
    X,Y=[],[]
    for i in range(lookback,len(data)):
        X.append(data[i-lookback:i,0])
        Y.append(data[i,0])
    return np.array(X),np.array(Y)


lookback=20


X_train,y_train=create_features(train,lookback)


X_test,y_test=create_features(test,lookback)


print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)

     (1913, 20) (1913,) (463, 20) (463,)


X_train[0:5]

     array([[0.02191224, 0.02174973, 0.01668472, 0.0092091 , 0.00083966,
             0.        , 0.00449621, 0.00433369, 0.0033857 , 0.00633803,
             0.01094258, 0.011078  , 0.01310942, 0.0165493 , 0.01218851,
             0.01197183, 0.01408451, 0.01486999, 0.01394908, 0.01286566],
            [0.02174973, 0.01668472, 0.0092091 , 0.00083966, 0.        ,
             0.00449621, 0.00433369, 0.0033857 , 0.00633803, 0.01094258,
             0.011078  , 0.01310942, 0.0165493 , 0.01218851, 0.01197183,
             0.01408451, 0.01486999, 0.01394908, 0.01286566, 0.01332611],
            [0.01668472, 0.0092091 , 0.00083966, 0.        , 0.00449621,
             0.00433369, 0.0033857 , 0.00633803, 0.01094258, 0.011078  ,
             0.01310942, 0.0165493 , 0.01218851, 0.01197183, 0.01408451,
             0.01486999, 0.01394908, 0.01286566, 0.01332611, 0.01232395],
            [0.0092091 , 0.00083966, 0.        , 0.00449621, 0.00433369,
             0.0033857 , 0.00633803, 0.01094258, 0.011078  , 0.01310942,
             0.0165493 , 0.01218851, 0.01197183, 0.01408451, 0.01486999,
             0.01394908, 0.01286566, 0.01332611, 0.01232395, 0.01121344],
            [0.00083966, 0.        , 0.00449621, 0.00433369, 0.0033857 ,
             0.00633803, 0.01094258, 0.011078  , 0.01310942, 0.0165493 ,
             0.01218851, 0.01197183, 0.01408451, 0.01486999, 0.01394908,
             0.01286566, 0.01332611, 0.01232395, 0.01121344, 0.01386782]],
           dtype=float32)


X_train=np.reshape(X_train,(X_train.shape[0],1,X_train.shape[1]))


X_test=np.reshape(X_test,(X_test.shape[0],1,X_test.shape[1]))


y_train=y_train.reshape(-1,1)


y_test=y_test.reshape(-1,1)


print(X_train.shape,y_train.shape,X_test.shape,y_test.shape)

     (1913, 1, 20) (1913, 1) (463, 1, 20) (463, 1)


model=Sequential()
model.add(LSTM(units=50,
              activation="relu",
              input_shape=(X_train.shape[1],lookback)))
model.add(Dropout(0.2))
model.add(Dense(1))


model.summary()

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                Output Shape              Param #
```

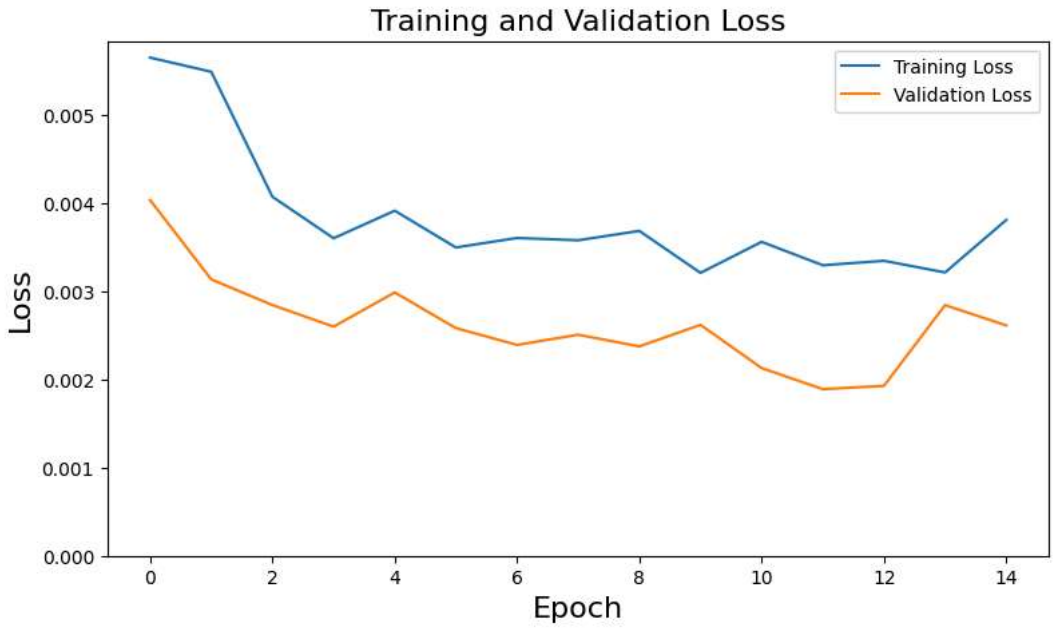
```
 lstm (LSTM)                 (None, 50)                14200

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 14251 (55.67 KB)
Trainable params: 14251 (55.67 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss="mean_squared_error",optimizer="adam")


callbacks=[EarlyStopping(monitor="val_loss",patience=3,verbose=1,mode="min"),
          ModelCheckpoint(filepath="mymodel.h5",monitor="val_loss",mode="min",
                          save_best_only=True,save_weights_only=False,verbose=1)]


history = model.fit(x=X_train,
                    y=y_train,
                    epochs=100,
                    batch_size=20,
                    validation_data=(X_test,y_test),
                    callbacks=callbacks,
                    shuffle=False)
```

```
96/96 [==============================] - 4s 13ms/step - loss: 0.0056 - val_loss: 0.0040
Epoch 2/100
90/96 [===========================>..] - ETA: 0s - loss: 0.0049
Epoch 2: val_loss improved from 0.00403 to 0.00313, saving model to mymodel.h5
96/96 [==============================] - 0s 5ms/step - loss: 0.0055 - val_loss: 0.0031
Epoch 3/100
77/96 [=====================>......] - ETA: 0s - loss: 0.0031
Epoch 3: val_loss improved from 0.00313 to 0.00284, saving model to mymodel.h5
96/96 [==============================] - 0s 4ms/step - loss: 0.0041 - val_loss: 0.0028
Epoch 4/100
93/96 [============================>.] - ETA: 0s - loss: 0.0034
Epoch 4: val_loss improved from 0.00284 to 0.00260, saving model to mymodel.h5
96/96 [==============================] - 0s 4ms/step - loss: 0.0036 - val_loss: 0.0026
Epoch 5/100
85/96 [=========================>....] - ETA: 0s - loss: 0.0030
Epoch 5: val_loss did not improve from 0.00260
96/96 [==============================] - 0s 4ms/step - loss: 0.0039 - val_loss: 0.0030
Epoch 6/100
88/96 [==========================>...] - ETA: 0s - loss: 0.0030
Epoch 6: val_loss improved from 0.00260 to 0.00258, saving model to mymodel.h5
96/96 [==============================] - 0s 5ms/step - loss: 0.0035 - val_loss: 0.0026
Epoch 7/100
88/96 [==========================>...] - ETA: 0s - loss: 0.0031
Epoch 7: val_loss improved from 0.00258 to 0.00239, saving model to mymodel.h5
96/96 [==============================] - 0s 5ms/step - loss: 0.0036 - val_loss: 0.0024
Epoch 8/100
89/96 [==========================>...] - ETA: 0s - loss: 0.0030
Epoch 8: val_loss did not improve from 0.00239
96/96 [==============================] - 0s 4ms/step - loss: 0.0036 - val_loss: 0.0025
Epoch 9/100
94/96 [============================>.] - ETA: 0s - loss: 0.0035
Epoch 9: val_loss improved from 0.00239 to 0.00237, saving model to mymodel.h5
96/96 [==============================] - 0s 4ms/step - loss: 0.0037 - val_loss: 0.0024
Epoch 10/100
80/96 [========================>.....] - ETA: 0s - loss: 0.0022
Epoch 10: val_loss did not improve from 0.00237
96/96 [==============================] - 0s 4ms/step - loss: 0.0032 - val_loss: 0.0026
Epoch 11/100
89/96 [==========================>...] - ETA: 0s - loss: 0.0030
Epoch 11: val_loss improved from 0.00237 to 0.00213, saving model to mymodel.h5
96/96 [==============================] - 0s 4ms/step - loss: 0.0036 - val_loss: 0.0021
Epoch 12/100
84/96 [=========================>....] - ETA: 0s - loss: 0.0023
Epoch 12: val_loss improved from 0.00213 to 0.00189, saving model to mymodel.h5
96/96 [==============================] - 0s 3ms/step - loss: 0.0033 - val_loss: 0.0019
Epoch 13/100
77/96 [=====================>......] - ETA: 0s - loss: 0.0021
Epoch 13: val_loss did not improve from 0.00189
96/96 [==============================] - 0s 4ms/step - loss: 0.0033 - val_loss: 0.0019
Epoch 14/100
93/96 [============================>.] - ETA: 0s - loss: 0.0031
```

```
83/96 [=========================>.....] - ETA: 0s - loss: 0.0024
Epoch 15: val_loss did not improve from 0.00189
96/96 [==============================] - 0s 4ms/step - loss: 0.0038 - val_loss: 0.0026
Epoch 15: early stopping
```

```
plt.figure(figsize=(20,5))
plt.subplot(1,2,2)
plt.plot(history.history["loss"],label="Training Loss")
plt.plot(history.history["val_loss"],label="Validation Loss")
plt.legend(loc="upper right")
plt.xlabel("Epoch",fontsize=16)
plt.ylabel("Loss",fontsize=16)
plt.ylim([0,max(plt.ylim())])
plt.title("Training and Validation Loss",fontsize=16)
plt.show()
```




```
loss=model.evaluate(X_test,y_test,batch_size=20)
```

```
24/24 [==============================] - 0s 2ms/step - loss: 0.0026
```

```
print("\nTest loss:%.1f%%"%(100.0*loss))
```

```
Test loss:0.3%
```

```
train_predict=model.predict(X_train)
test_predict=model.predict(X_test)
```

```
60/60 [==============================] - 0s 2ms/step
15/15 [==============================] - 0s 2ms/step
```

```
train_predict=scaler_train.inverse_transform(train_predict)
test_predict=scaler_test.inverse_transform(test_predict)
```

```
y_train=scaler_train.inverse_transform(y_train)
y_test=scaler_test.inverse_transform(y_test)
```

train veri setine RMSE değeri (RMSE value to train dataset)

```
train_rmse=np.sqrt(mean_squared_error(y_train,train_predict))
```

```
test_rmse=np.sqrt(mean_squared_error(y_test,test_predict,))
```

```
print(f"Train RMSE:{train_rmse}")
print(f"Test RMSE:{test_rmse}")
```

```
Train RMSE:13.78125286102295
Test RMSE:30.717008590698242
```

```
train_prediction_data=result_data[lookback:pos]
```

```
train_prediction_data["Predicted"]=train_predict
```

```
train_prediction_data.head()
```

Date	Close	Predicted
2010-07-28	20.719999	35.126179
2010-07-29	20.350000	35.126179
2010-07-30	19.940001	35.126179
2010-08-02	20.920000	35.126179
2010-08-03	21.950001	35.126179

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

```
test_prediction_data=result_data[pos+lookback:]
```

```
test_prediction_data["Predicted"]=test_predict
```

```
test_prediction_data.head()
```

Date	Close	Predicted
2018-04-03	267.529999	321.148895
2018-04-04	286.940002	314.951172
2018-04-05	305.720001	313.678802
2018-04-06	299.299988	315.122559
2018-04-09	289.660004	312.168915

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

```
plt.figure(figsize=(14,5))
plt.plot(result_data,label="Real Values")
plt.plot(train_prediction_data["Predicted"],color="blue",label="Train Predicted")
plt.plot(test_prediction_data["Predicted"],color="red",label="Test Predicted")
plt.xlabel("Time")
plt.ylabel("Stock Values")
plt.legend()
plt.show()
```


Real Values
Train Predicted
Test Predicted
800
700
600
500
400
Values