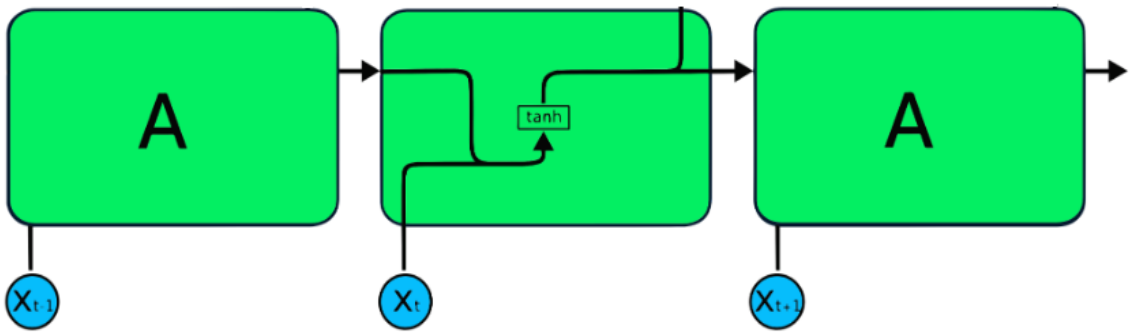
# RECURRENT NEURAL NETWORK

A **recurrent neural network** (RNN) is a type of neural network in which the output is determined by the current input and previously received inputs. RNN is used to model time series data. The simple RNN repeating modules have a basic structure with a single layer, and it remembers only one previous time step information. Simple RNN models usually run into two major issues:

• **Vanishing Gradient** problem occurs when the gradient becomes so small that updating parameters becomes insignificant; eventually, the algorithm stops learning.
• **Exploding Gradient** problem occurs when the gradient becomes too large, which makes the model unstable. In this case, larger error gradients accumulate, and the model weights become too large. This issue can cause longer training times and poor model performance.



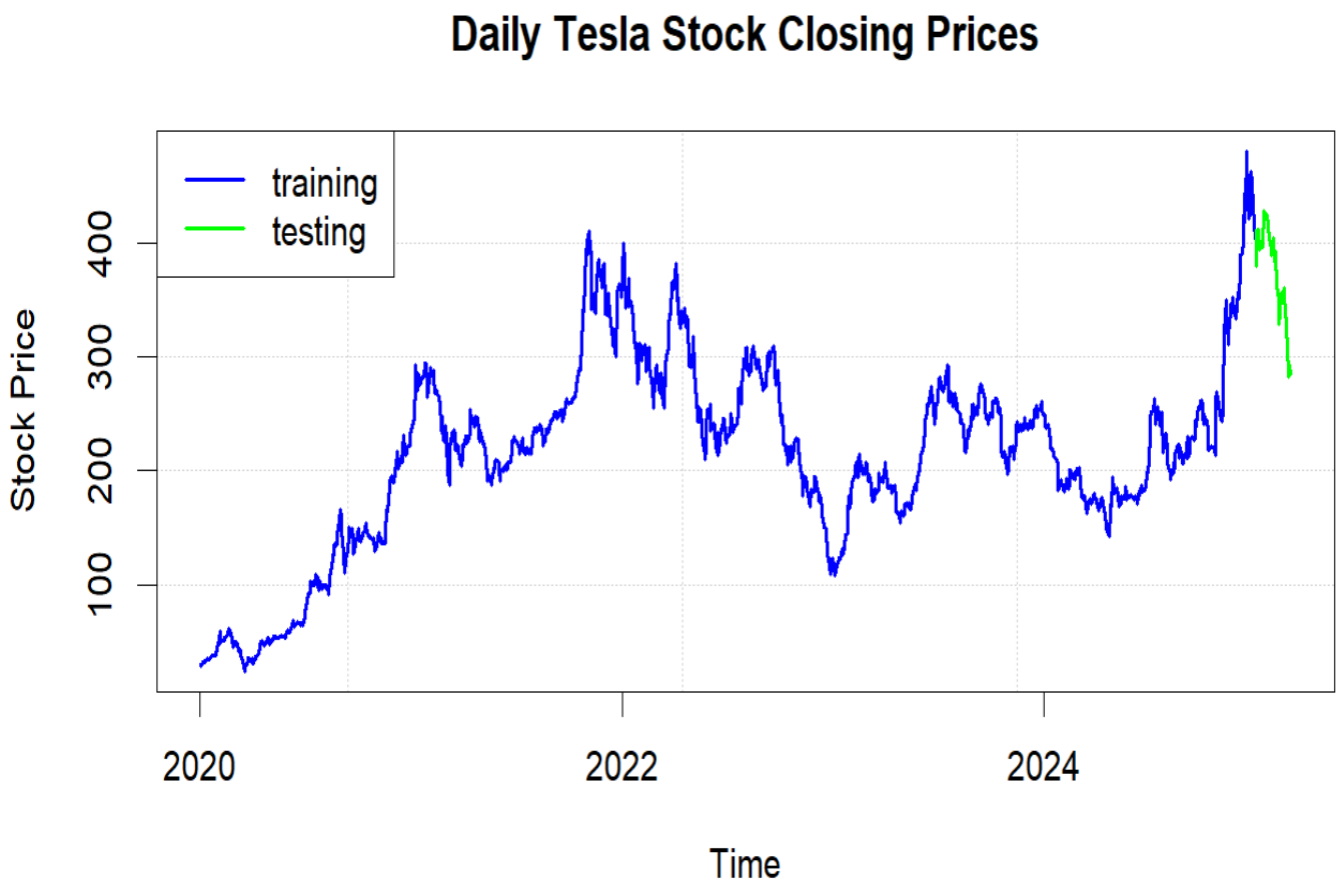

More advanced RNN architectures that easily solve these problems are **long short term memory** (LSTM) and **gated recurrent unit** (GRU), as they are capable of remembering long periods of information. The LSTM has four interacting layers that communicate with each other. This four-layered structure helps LSTM retain long-term memory. The gated recurrent unit (GRU) is a variation of LSTM. It uses an update gate and reset gate to solve the vanishing gradient problem. These gates decide what information is important and pass it to the output. The gates can be trained to store information from long ago.

**Example.** We fit LSTM and GRU recurrent neural networks to Tesla stock closing prices. We extract data from *https://finance.yahoo.com*, using R and Python.

In R:

```
library(yahoofinancer)
TSLA<- Ticker$new('TSLA')
TSLA.stock<- TSLA$get_history(start = '2020-04-08', end = '2025-04-08', interval = '1d')
```

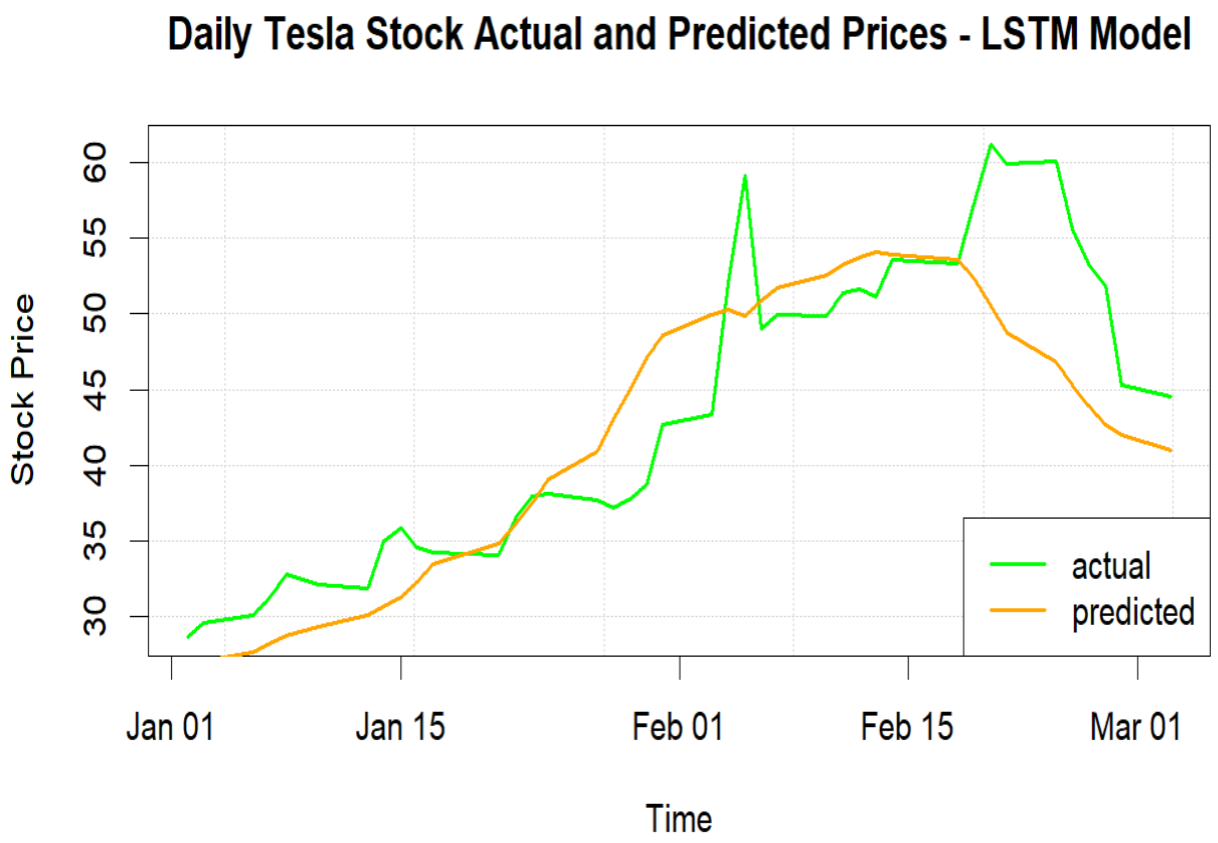
```
TSLA.stock$date<- substr(as.POSIXct(strftime(TSLA.stock$date, format="%Y-%m-%d %H:%M:%S"),
format = "%Y-%m-%d", tz=""), 1, 10)

TSLA.data<- as.data.frame(list(TSLA.stock$date, TSLA.stock$close), col.names=c("Date", "Close"),
make.names=FALSE)

tsla.data<- na.omit(TSLA.data)

#splitting data into testing and training sets
tsla.data$Year<- as.numeric(format(as.Date(tsla.data$Date, format="%Y-%m-%d"),"%Y"))

train.data<- tsla.data[which(tsla.data$Year<2025),1:2]
test.data<- tsla.data[which(tsla.data$Year>=2025),1:2]

#plotting training and testing data
plot(as.POSIXct(tsla.data$Date), tsla.data$Close, main="Daily Tesla Stock Closing Prices", xlab="Time",
ylab="Stock Price", pch="", panel.first=grid())
lines(as.POSIXct(train.data$Date), train.data$Close, lwd=2, col="blue")
lines(as.POSIXct(test.data$Date), test.data$Close, lwd=2, col="green")
legend("topleft", c("training", "testing"), lty=1, col=c("blue","green"))
```




```
#scaling prices to fall in [0,1]
price<- tsla.data$Close
price.sc<- (price-min(price))/(max(price)-min(price))

#creating train.x and train.y
nsteps<- 60 #width of sliding window
train.matrix <- matrix(nrow=nrow(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(nrow(train.data)-nsteps))
    train.matrix[i,]<- price.sc[i:(i+nsteps)]

train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]

#creating test.x and test.y
test.matrix<- matrix(nrow=nrow(test.data), ncol=nsteps+1)
```

```
for (i in 1:nrow(test.data))
    test.matrix[i,]<- price.sc[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix))]

test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#FITTING LSTM MODEL
library(keras3)
LSTM.model <- keras_model_sequential()

#specifying model structure
LSTM.model %>% layer_lstm(input_shape=dim(train.x)[2:3], units=nsteps)
LSTM.model %>% layer_dense(units=1, activation="tanh")
LSTM.model %>% compile(loss="mean_squared_error")

#training model
    LSTM.model %>% fit(train.x, train.y, batch_size=32, epochs=5)

#predicting for testing data
pred.y<- LSTM.model %>% predict(test.x, batch_size=32)

#rescaling test.y and pred.y back to the original scale
test.y.re<- test.y*(max(price)-min(price))+min(price)
pred.y.re<- pred.y*(max(price)-min(price))+min(price)

#computing prediction accuracy
accuracy10<- ifelse(abs(test.y.re-pred.y.re)<0.10*test.y.re,1,0)
accuracy15<- ifelse(abs(test.y.re-pred.y.re)<0.15*test.y.re,1,0)
accuracy20<- ifelse(abs(test.y.re-pred.y.re)<0.20*test.y.re,1,0)

print(paste("accuracy within 10%:", round(mean(accuracy10),4)))


"accuracy within 10%: 0.625"

print(paste("accuracy within 15%:", round(mean(accuracy15),4)))


"accuracy within 15%: 0.725"

print(paste("accuracy within 20%:", round(mean(accuracy20),4)))


"accuracy within 20%: 0.95"
```

```
#plotting actual and predicted values for testing data
plot(as.POSIXct(test.data$Date), test.y.re, type="l", lwd=2, col="green", main="Daily Tesla Stock
Actual and Predicted Prices - LSTM Model", xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="orange")
legend("bottomright", c("actual", "predicted"), lty=1, lwd=2, col=c("green","orange"))
```




```
#FITTING GRU MODEL
GRU.model <- keras_model_sequential()

#specifying model structure
GRU.model %>% layer_gru(input_shape=dim(train.x)[2:3], units=nsteps)
GRU.model %>% layer_dense(units=1, activation="tanh")
GRU.model %>% compile(loss="mean_squared_error")

#training model
    GRU.model %>% fit(train.x, train.y, batch_size=32, epochs=5)
```

```
#predicting for testing data
pred.y<- GRU.model %>% predict(test.x, batch_size=32)

#rescaling pred.y back to the original scale
pred.y<- pred.y*(max(price)-min(price))+min(price)

#computing prediction accuracy
accuracy10<- ifelse(abs(test.y.re-pred.y.re)<0.10*test.y.re,1,0)
accuracy15<- ifelse(abs(test.y.re-pred.y.re)<0.15*test.y.re,1,0)
accuracy20<- ifelse(abs(test.y.re-pred.y.re)<0.20*test.y.re,1,0)

print(paste("accuracy within 10%:", round(mean(accuracy10),4)))
```

```
"accuracy within 10%: 0.425"
```

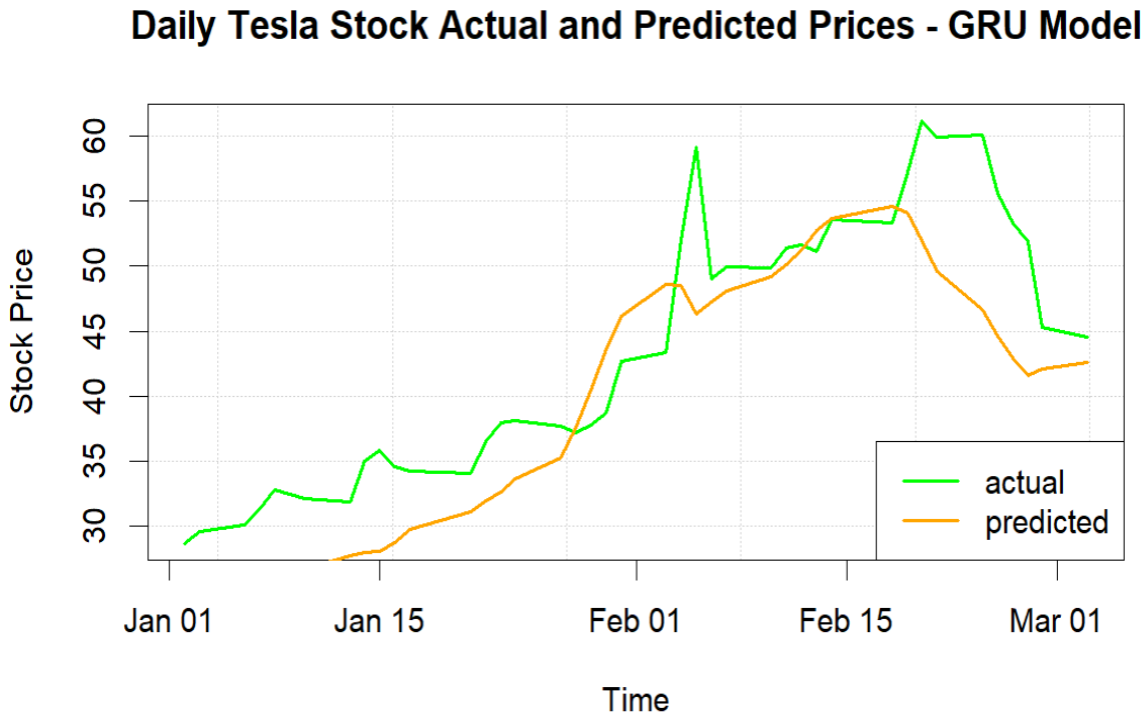
```
print(paste("accuracy within 15%:", round(mean(accuracy15),4)))
```

```
"accuracy within 15%: 0.6"
```

```
print(paste("accuracy within 20%:", round(mean(accuracy20),4)))
```

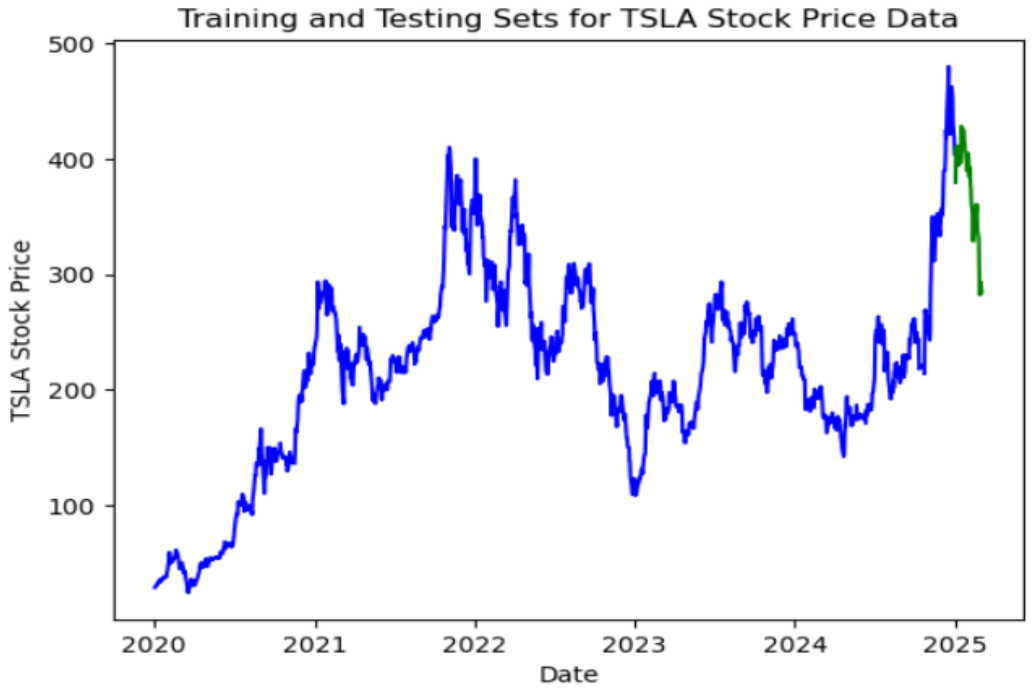
```
"accuracy within 20%: 0.85"
```

```
#plotting actual and predicted values for testing data
plot(as.POSIXct(test.data$Date), test.y.re, type="l", lwd=2, col="green", main="Daily Tesla Stock
Actual and Predicted Prices - GRU Model", xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="orange")
legend("bottomright", c("actual", "predicted"), lty=1, lwd=2, col=c("green","orange"))
```


Daily Tesla Stock Actual and Predicted Prices - GRU Model
Stock Price
30
35
40
45
50
55
60
Jan 01
Jan 15
Feb 01
Feb 15
Mar 01
Time
actual
predicted

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from statistics import mean

from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from tensorflow.random import set_seed

tsla_data=pandas.read_csv('./TSLA.csv', index_col="Date", parse_dates=["Date"])

#plotting daily Tesla stock closing prices
train = tsla_data[tsla_data.index < pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]
test = tsla_data[tsla_data.index >= pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]

plt.plot(train, color = "blue")
plt.plot(test, color = "green")
plt.ylabel('TSLA Stock Price')
plt.xlabel('Date')
plt.title("Training and Testing Sets for TSLA Stock Price Data")
plt.show()
```

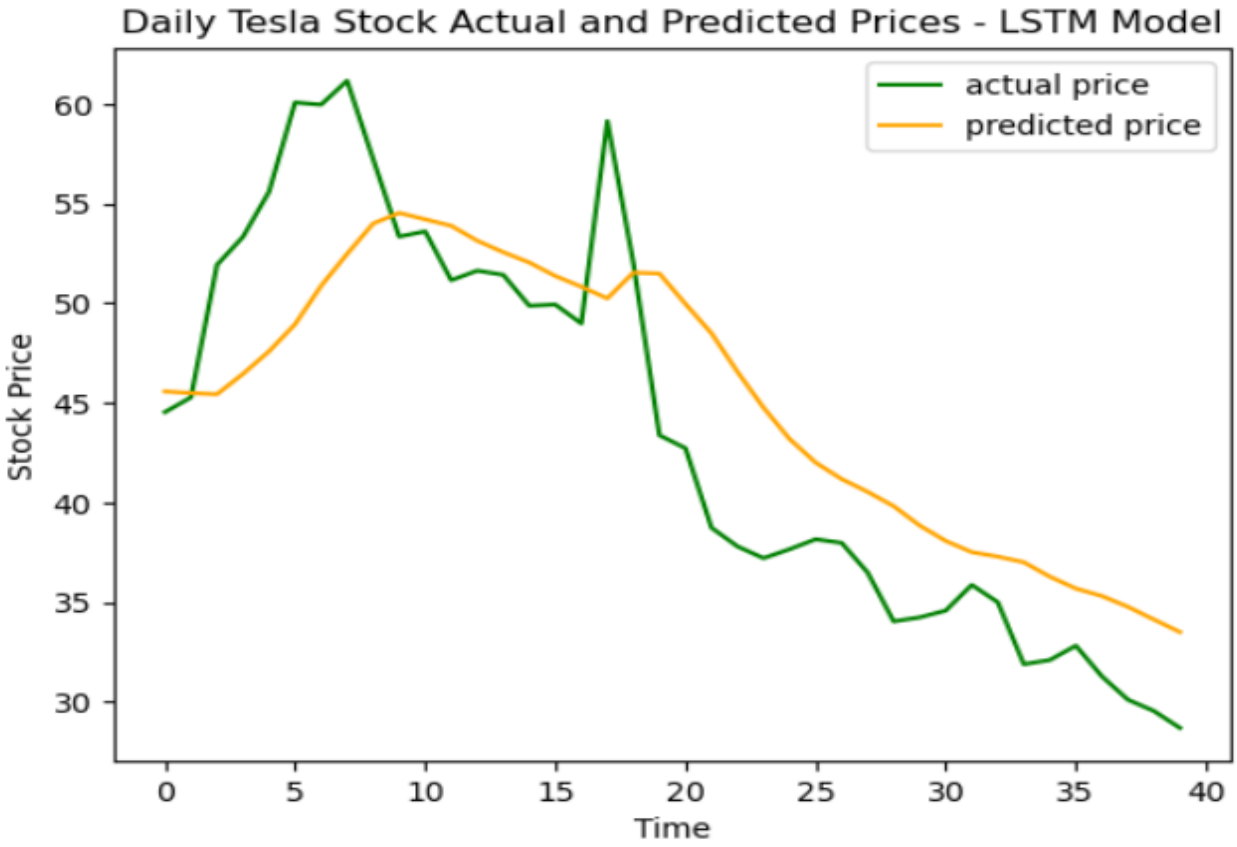


```
#rescaling data
tsla_data["Close_sc"]=(tsla_data["Close"]-min(tsla_data["Close"]))/(max(tsla_data["Close"].values)-min(tsla_data["Close"]))

train_set = tsla_data[tsla_data.index < pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]
test_set = tsla_data[tsla_data.index >= pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]

train_set=train_set.loc[:, "Close_sc"].values
test_set=test_set.loc[:, "Close_sc"].values

#splitting training data into samples
nsteps=60  #width of sliding window

def split_sequence(sequence, n_steps):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y=split_sequence(train_set, nsteps)
```

```
#################################################################
#FITTING LSTM MODEL
#################################################################
features=1 #predictors and response are the same variable
#reshaping train_x
train_x=train_x.reshape(train_x.shape[0], train_x.shape[1],features)

#specifying LSTM model architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=6, activation="tanh", input_shape=(nsteps, features)))
model_lstm.add(Dense(units=1))

#compiling the model
model_lstm.compile(loss="mse")
model_lstm.fit(train_x, train_y, epochs=5, batch_size=32)

#creating testing set by adding nsteps observations from training set to testing set
inputs=tsla_data.loc[:,"Close_sc"][len(tsla_data)-len(test_set)-nsteps:].values
inputs=inputs.reshape(-1, 1)

#splitting into samples
test_x, test_y=split_sequence(inputs, nsteps)

#reshaping
test_x=test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_y=model_lstm.predict(test_x)

#inverse transforming the values
pred_y=pred_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])
test_y=test_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])
```

```
#computing prediction accuracy
ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(pred_y, test_y):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('accuracy within 10%:', round(mean(ind10),4))
print('accuracy within 15%:', round(mean(ind15),4))
print('accuracy within 20%:', round(mean(ind20),4))

#plotting actual and predicted values for testing data
plt.plot(test_y, color="green", label="actual price")
plt.plot(pred_y, color="orange", label="predicted price")
plt.title("Daily Tesla Stock Actual and Predicted Prices - LSTM Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

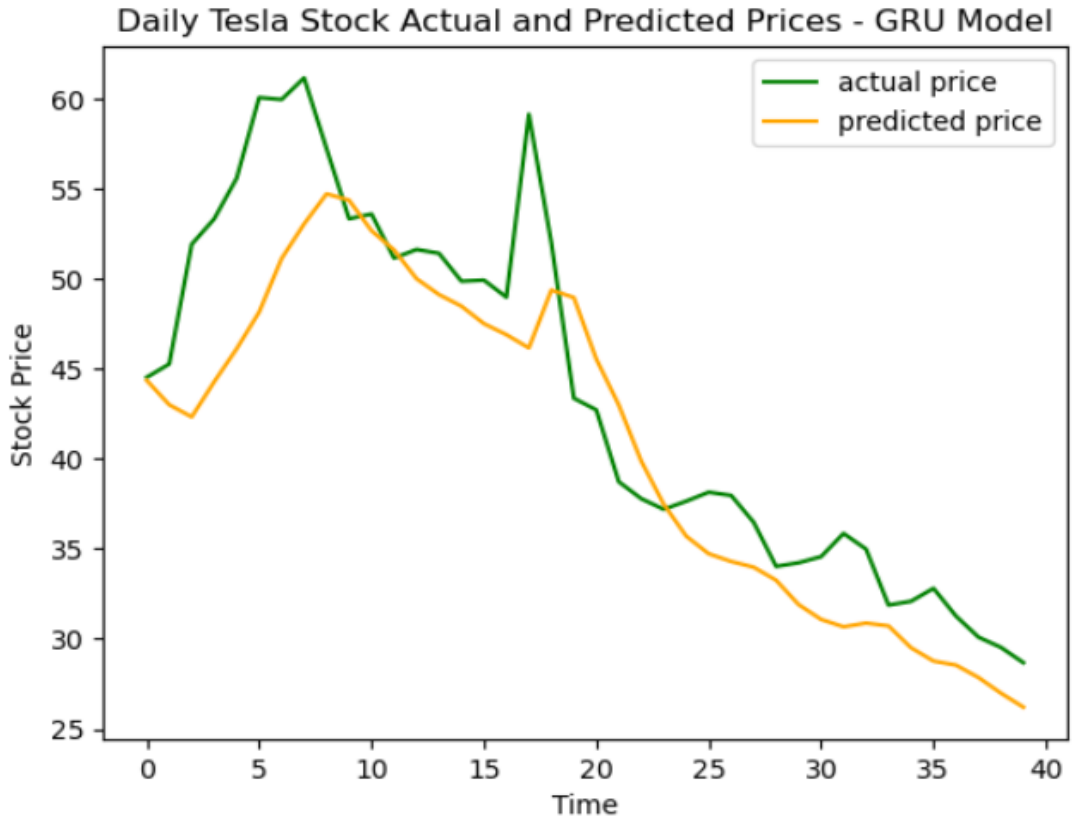
```
accuracy within 10%: 0.4
accuracy within 15%: 0.675
accuracy within 20%: 0.925
```



```
#####################################################################
#FITTING GRU MODEL
#####################################################################
#specifying GRU model architecture
model_gru = Sequential()
model_gru.add(GRU(units=6, activation="tanh", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

# Compiling the model
model_gru.compile(loss="mse")
model_gru.fit(train_x, train_y, epochs=5, batch_size=32)

#predicting for testing data
pred_y=model_gru.predict(test_x)

#inverse transforming the values
pred_y=pred_y*(max(tsla_data["Close"].values)-min(tsla_data["Close"]))+min(tsla_data["Close"])

#computing prediction accuracy
ind10=[]
ind15=[]
ind20=[]

for sub1, sub2 in zip(pred_y, test_y):
    ind10.append(1) if abs(sub1-sub2)<0.10*sub2 else ind10.append(0)
    ind15.append(1) if abs(sub1-sub2)<0.15*sub2 else ind15.append(0)
    ind20.append(1) if abs(sub1-sub2)<0.20*sub2 else ind20.append(0)

print('accuracy within 10%:', round(mean(ind10),4))
print('accuracy within 15%:', round(mean(ind15),4))
print('accuracy within 20%:', round(mean(ind20),4))
```

```
#plotting actual and predicted values for testing data
plt.plot(test_y, color="green", label="actual price")
plt.plot(pred_y, color="orange", label="predicted price")
plt.title("Daily Tesla Stock Actual and Predicted Prices - GRU Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```

```
accuracy within 10%: 0.675
accuracy within 15%: 0.875
accuracy within 20%: 0.975
```




□

**Example.** We extract daily trading volumes of Tesla stock from *https://finance.yahoo.com* and de-

fine *shocks* as binary indicators representing whether the volume increased or decreased by at least 20% between two consecutive business days. The 20% threshold is selected to ensure a reasonably high proportion of ones in the dataset. We then fit LSTM and GRU neural networks to model the binary response variable, using both R and Python.

In R:

```
library(yahoofinancer)

TSLA<- Ticker$new('TSLA')
TSLA.stock<- TSLA$get_history(start = '2020-04-08', end = '2025-04-08', interval = '1d')

TSLA.stock$date<- substr(as.POSIXct(strftime(TSLA.stock$date, format="%Y-%m-%d %H:%M:%S"),
format = "%Y-%m-%d", tz=""), 1, 10)

TSLA.data<- as.data.frame(list(TSLA.stock$date, TSLA.stock$volume), col.names=c("Date", "Vol-
ume"), make.names=FALSE)

tsla.data<- na.omit(TSLA.data)

#computing shocks
tsla.data.lag<- tsla.data[-1,]
tsla.data<- tsla.data[-nrow(tsla.data),]

shock<- c()
for (i in 1:nrow(tsla.data))
shock[i]<- ifelse(abs(tsla.data.lag[i,2]-tsla.data[i,2])>0.2*tsla.data[i,2],1,0)
tsla.data$Shock<- shock

table(tsla.data$Shock)

   0 1 750 505
#splitting data into testing and training sets
tsla.data$Year<- as.numeric(format(as.Date(tsla.data$Date, format="%Y-%m-%d"),"%Y"))
train.data<- tsla.data[which(tsla.data$Year<2025),1:2]
test.data<- tsla.data[which(tsla.data$Year>=2025),1:2]

nsteps<- 60 #width of sliding window
train.matrix <- matrix(nrow=nrow(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(nrow(train.data)-nsteps))
    train.matrix[i,]<- tsla.data$Shock[i:(i+nsteps)]
```

```
train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]

#creating test.x and test.y
test.matrix<- matrix(nrow=nrow(test.data), ncol=nsteps+1)
for (i in 1:nrow(test.data))
test.matrix[i,]<- tsla.data$Shock[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix))]

test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#FITTING LSTM MODEL
library(keras3)
#defining model architecture
LSTM.biclass<- keras_model_sequential()
LSTM.biclass %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
LSTM.biclass %>% layer_lstm(units=25)
LSTM.biclass %>% layer_dense(units=1, activation="sigmoid")
LSTM.biclass %>% compile(loss="binary_crossentropy")

#training model
LSTM.biclass %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing prediction accuracy for testing data
pred.prob<- LSTM.biclass %>% predict(test.x)
match<- cbind(test.y, pred.prob)
tp<- matrix(NA, nrow=nrow(match), ncol=99)
tn<- matrix(NA, nrow=nrow(match), ncol=99)

for (i in 1:99) {
    tp[,i]<- ifelse(match[,1]==1 & match[,2]>0.01*i,1,0)
    tn[,i]<- ifelse(match[,1]==0 & match[,2]<=0.01*i,1,0)
}

trueclassrate<- matrix(NA, nrow=99, ncol=2)
for (i in 1:99){
    trueclassrate[i,1]<- 0.01*i
    trueclassrate[i,2]<- sum(tp[,i]+tn[,i])/nrow(match)
}

print(trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),])
```

```
      [,1]     [,2]
  [1,] 0.01 0.640625
  [2,] 0.02 0.640625
  [3,] 0.03 0.640625
  [4,] 0.04 0.640625
  [5,] 0.05 0.640625
         . . .
 [95,] 0.95 0.640625
 [96,] 0.96 0.640625
 [97,] 0.97 0.640625
 [98,] 0.98 0.640625
 [99,] 0.99 0.640625
```

The prediction accuracy is 64.0625% for any cut-off between 0.01 and 0.99.

```
#FITTING GRU MODEL
#defining model architecture
GRU.biclass<- keras_model_sequential()
GRU.biclass %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
GRU.biclass %>% layer_gru(units=25)
GRU.biclass %>% layer_dense(units=1, activation="sigmoid")
GRU.biclass %>% compile(loss="binary_crossentropy")

#training model
GRU.biclass %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing prediction accuracy for testing data
pred.prob<- GRU.biclass %>% predict(test.x)
match<- cbind(test.y, pred.prob)
tp<- matrix(NA, nrow=nrow(match), ncol=99)
tn<- matrix(NA, nrow=nrow(match), ncol=99)

for (i in 1:99) {
    tp[,i]<- ifelse(match[,1]==1 & match[,2]>0.01*i,1,0)
    tn[,i]<- ifelse(match[,1]==0 & match[,2]<=0.01*i,1,0)
}

trueclassrate<- matrix(NA, nrow=99, ncol=2)
for (i in 1:99) {
    trueclassrate[i,1]<- 0.01*i
    trueclassrate[i,2]<- sum(tp[,i]+tn[,i])/nrow(match)
}
```

```
print(trueclassrate[which(trueclassrate[,2]==max(trueclassrate[,2])),])
```

```
     [,1]     [,2]
 [1,] 0.32 0.640625
 [2,] 0.33 0.640625
 [3,] 0.34 0.640625
 [4,] 0.35 0.640625
 [5,] 0.36 0.640625
       .   .   .
[64,] 0.95 0.640625
[65,] 0.96 0.640625
[66,] 0.97 0.640625
[67,] 0.98 0.640625
[68,] 0.99 0.640625
```

The prediction accuracy is 64.0625% for any cut-off between 0.32 and 0.99.

In Python:

```
import numpy
import pandas
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from tensorflow.random import set_seed

tsla_data=pandas.read_csv('./TSLA_Shocks.csv', index_col="Date", parse_dates=["Date"])

#splitting into training and testing sets
train = tsla_data[tsla_data.index < pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]
test = tsla_data[tsla_data.index >= pandas.to_datetime("2025-01-02", format='%Y-%m-%d')]

train_set=train.loc[:, "Shock"].values
test_set=test.loc[:, "Shock"].values

#splitting training data into samples
nsteps = 60

def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence)-1:
            break
        seq_x, seq_y=sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)
```

```
#################################################################
#FITTING LSTM MODEL
#################################################################
#reshaping train_x
features = 1
train_x=train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying model architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=6, activation="sigmoid", input_shape=(nsteps, features)))
model_lstm.add(Dense(units=1))

# Compiling the model
model_lstm.compile(loss="binary_crossentropy")
model_lstm.fit(train_x, train_y, epochs=5, batch_size=32)
inputs = tsla_data.loc[:,"Shock"][len(tsla_data.loc[:,"Shock"])-len(test_set)-nsteps :].values

#splitting into samples
test_x, test_y = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob=model_lstm.predict(test_x)
```

```
cutoff=[]
accuracy=[]
for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(pred_prob, test_y):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind
    accuracy_i=(tp+tn)/len(pred_prob)
    accuracy.append(accuracy_i)

df=pandas.DataFrame({'accuracy': accuracy,'cut-off': cutoff})
max_accuracy=max(accuracy)
optimal=df[df['accuracy']==max_accuracy]
print(optimal)
```

```
    accuracy  cut-off
24      0.75     0.25
25      0.75     0.26
26      0.75     0.27
27      0.75     0.28
28      0.75     0.29
..       ...      ...
94      0.75     0.95
95      0.75     0.96
96      0.75     0.97
97      0.75     0.98
98      0.75     0.99
```

The prediction accuracy is 75% for any cut-off between 0.25 and 0.99.

```
#####################################################################
#FITTING GRU MODEL
#####################################################################
#specifying model architecture
model_gru = Sequential()
model_gru.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

# Compiling the model
model_gru.compile(loss="binary_crossentropy")
model_gru.fit(train_x, train_y, epochs=5, batch_size=32)

#predicting for testing data
pred_prob=model_gru.predict(test_x)

cutoff=[]
accuracy=[]
for i in range(99):
    tp=0
    tn=0
    cutoff.append(0.01*(i+1))
    for sub1, sub2 in zip(pred_prob, test_y):
        tp_ind=1 if (sub1>0.01*(i+1) and sub2==1) else 0
        tn_ind=1 if (sub1<0.01*(i+1) and sub2==0) else 0
        tp+=tp_ind
        tn+=tn_ind

    accuracy_i=(tp+tn)/len(pred_prob)
    accuracy.append(accuracy_i)

df=pandas.DataFrame({'accuracy': accuracy,'cut-off': cutoff})
max_accuracy=max(accuracy)
optimal=df[df['accuracy']==max_accuracy]
print(optimal)
```

```
    accuracy  cut-off
0       0.75     0.01
1       0.75     0.02
2       0.75     0.03
3       0.75     0.04
```

```
4       0.75     0.05
..       ...      ...
94      0.75     0.95
95      0.75     0.96
96      0.75     0.97
97      0.75     0.98
98      0.75     0.99
```

The prediction accuracy is 75% for any cut-off between 0.01 and 0.99. □

**Example.** The file "LA_weather.csv" contains hourly weather conditions in LA (rain/fog/clear/cloud) between 10/1/2019 12 PM and 11/30/2024 12 AM. There data were downloaded from kaggle.com and cleaned. We use this data set to fit RNN for multinomial classification in R and Python.

In R:

```
LA.weather<- read.csv(file="./LA_weather.csv", header=TRUE, sep=",")

LA.weather$Rain<- ifelse(LA.weather$Condition=="rain",1, 0)
LA.weather$Fog<- ifelse(LA.weather$Condition=="fog",1, 0)
LA.weather$Clear<- ifelse(LA.weather$Condition=="clear",1,0)
LA.weather$Cloud<- ifelse(LA.weather$Condition=="cloud",1,0)
LA.weather$Year<- format(as.Date(LA.weather$Date, format="%Y-%m-%d"),"%Y")

#DEFINING FUNCTION THAT FITS RNN MODEL

rnn.model<- function(modelname, varname) {

#creating train.x, train.y, test.x, and test.y sets
train.data<- LA.weather[which(LA.weather$Year<2024),varname]
test.data<- LA.weather[which(LA.weather$Year==2024),varname]

nsteps<- 60
train.matrix <- matrix(nrow=length(train.data)-nsteps, ncol=nsteps+1)
for (i in 1:(length(train.data)-nsteps))
    train.matrix[i,]<- LA.weather[i:(i+nsteps),varname]

train.x<- array(train.matrix[,-ncol(train.matrix)],dim=c(nrow(train.matrix),nsteps,1))
train.y<- train.matrix[,ncol(train.matrix)]
```

```
test.matrix<- matrix(nrow=length(test.data), ncol=nsteps+1)
for (i in 1:length(test.data))
    test.matrix[i,]<- LA.weather[(i+nrow(train.matrix)):(i+nsteps+nrow(train.matrix)),varname]

test.x<- array(test.matrix[,-ncol(test.matrix)],dim=c(nrow(test.matrix),nsteps,1))
test.y<- test.matrix[,ncol(test.matrix)]

#defining model architecture
library(keras3)
fitted.model<- keras_model_sequential()
fitted.model %>% layer_dense(input_shape=dim(train.x)[2:3], units=nsteps)
if (modelname=='lstm') {
    fitted.model %>% layer_lstm(units=6)
} else fitted.model %>% layer_gru(units=6)
fitted.model %>% layer_dense(units=1, activation='sigmoid')
fitted.model %>% compile(loss='binary_crossentropy')

#training model
fitted.model %>% fit(train.x, train.y, batch_size=32, epochs=5)

#computing predicted probability of rain for testing data
pred.prob<- fitted.model %>% predict(test.x)
return(list(test.y, pred.prob))
}

#DEFINING FUNCTION THAT COMPUTES PREDICTION ACCURACY
library(dplyr)

accuracy<- function() {

test.y<- bind_cols(test.rain, test.fog, test.clear, test.cloud)
colnames(test.y)<- 1:4
true.class<- as.numeric(apply(test.y, 1, function(x) colnames(test.y)[which.max(x)]))

pred.prob<- bind_cols(pred.prob.rain, pred.prob.fog, pred.prob.clear, pred.prob.cloud)
colnames(pred.prob)<- 1:4
pred.class<- as.numeric(apply(pred.prob, 1, function(x) colnames(pred.prob)[which.max(x)]))

match<- c()
for (i in 1:length(pred.class)) {
    match[i]<- ifelse(pred.class[i]==true.class[i],1,0)
}
```

```
return(round(mean(match),4))
}

#RUNNING LSTM BINARY CLASSIFICATION MODELS
list.rain<- (rnn.model('lstm', 'Rain'))
test.rain<- list.rain[1]
pred.prob.rain<- list.rain[2]

list.fog<- rnn.model('lstm', 'Fog')
test.fog<- list.fog[1]
pred.prob.fog<- list.fog[2]

list.clear<- rnn.model('lstm', 'Clear')
test.clear<- list.clear[1]
pred.prob.clear<- list.clear[2]

list.cloud<- rnn.model('lstm', 'Cloud')
test.cloud<- list.cloud[1]
pred.prob.cloud<- list.cloud[2]
print(accuracy())
```

```
0.7774
```

```
#RUNNING GRU BINARY CLASSIFICATION MODELS
list.rain<- (rnn.model('gru', 'Rain'))
test.rain<- list.rain[1]
pred.prob.rain<- list.rain[2]

list.fog<- rnn.model('gru', 'Fog')
test.fog<- list.fog[1]
pred.prob.fog<- list.fog[2]

list.clear<- rnn.model('gru', 'Clear')
test.clear<- list.clear[1]
pred.prob.clear<- list.clear[2]

list.cloud<- rnn.model('gru', 'Cloud')
test.cloud<- list.cloud[1]
pred.prob.cloud<- list.cloud[2]
print(accuracy())
```

0.7771

In Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from statistics import mean
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU

LA_weather=pandas.read_csv('./LA_weather.csv', index_col="Date", parse_dates=["Date"])

#creating dummy variables
LA_weather=pandas.get_dummies(LA_weather['Condition'])


#FITTING LSTM MODEL

#########################################################
#Building Model for Rain
#########################################################

#splitting into training and testing sets
train = LA_weather[LA_weather.index < pandas.to_datetime("2024-09-01", format='%Y-%m-%d')]
test = LA_weather[LA_weather.index >= pandas.to_datetime("2024-09-01", format='%Y-%m-%d')]

train_set=train.loc[:, "rain"].values
test_set=test.loc[:, "rain"].values

#splitting training data into samples
nsteps = 60
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)
```

```
train_x, train_y = split_sequence(train_set)

#reshaping train_x
features = 1
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6, activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"rain"][len(LA_weather.loc[:,"rain"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_rain = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_rain = fitted_model.predict(test_x)


#########################################################
#Building Model for Fog
#########################################################
#defining training and testing sets
train_set=train.loc[:, "fog"].values
test_set=test.loc[:, "fog"].values


#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
```

```
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"fog"][len(LA_weather.loc[:,"fog"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_fog = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_fog = fitted_model.predict(test_x)
```

```
#######################################################
#Building Model for Clear
#######################################################
#defining training and testing sets
train_set=train.loc[:, "clear"].values
test_set=test.loc[:, "clear"].values


#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)

#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"clear"][len(LA_weather.loc[:,"clear"]) - len(test_set) - nsteps : ].values
```

```
#splitting into samples
test_x, test_clear = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_clear = fitted_model.predict(test_x)

#########################################################
#Building Model for Cloud
#########################################################
#defining training and testing sets
train_set=train.loc[:, "cloud"].values
test_set=test.loc[:, "cloud"].values

#splitting training data into samples
def split_sequence(sequence):
    x, y = list(), list()
    for i in range(len(sequence)):
        end_i = i + nsteps
        if end_i > len(sequence) - 1:
            break
        seq_x, seq_y = sequence[i:end_i], sequence[end_i]
        x.append(seq_x)
        y.append(seq_y)
    return numpy.array(x), numpy.array(y)

train_x, train_y = split_sequence(train_set)

#reshaping train_x
train_x = train_x.reshape(train_x.shape[0],train_x.shape[1],features)
```

```
#specifying LSTM model architecture
fitted_model = Sequential()
fitted_model.add(LSTM(units=6,  activation="sigmoid", input_shape=(nsteps, features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

#compiling model
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
inputs=LA_weather.loc[:,"cloud"][len(LA_weather.loc[:,"cloud"]) - len(test_set) - nsteps : ].values

#splitting into samples
test_x, test_cloudy = split_sequence(inputs)

#reshaping
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

#predicting for testing data
pred_prob_cloudy = fitted_model.predict(test_x)

#########################################################
#Computing Prediction Accuracy
#########################################################
pred_prob_all=numpy.concatenate((pred_prob_rain, pred_prob_fog,pred_prob_clear,pred_prob_cloudy), axis=1)
pred_prob_all=pandas.DataFrame(pred_prob_all)
pred_class=pred_prob_all.idxmax(axis=1)

test_all=numpy.c_[test_rain,test_fog, test_clear, test_cloudy]
test_all=pandas.DataFrame(test_all)
true_class=test_all.idxmax(axis=1)

match=[]
for i in range(len(pred_class)):
    if pred_class[i]==true_class[i]:
            match.append(1)
    else:
            match.append(0)

print(round(mean(match),4))
```

```
0.789
```

We fit a GRU model by running the code identical to the one for LSTM above, but with 'LSTM'

replaced by 'GRU' (in four places). The prediction accuracy for this model is

```
0.7871
```

□