Cory Suzuki

STAT 574

Dr. Olga

12 April 2025

Homework 4

Problem 1

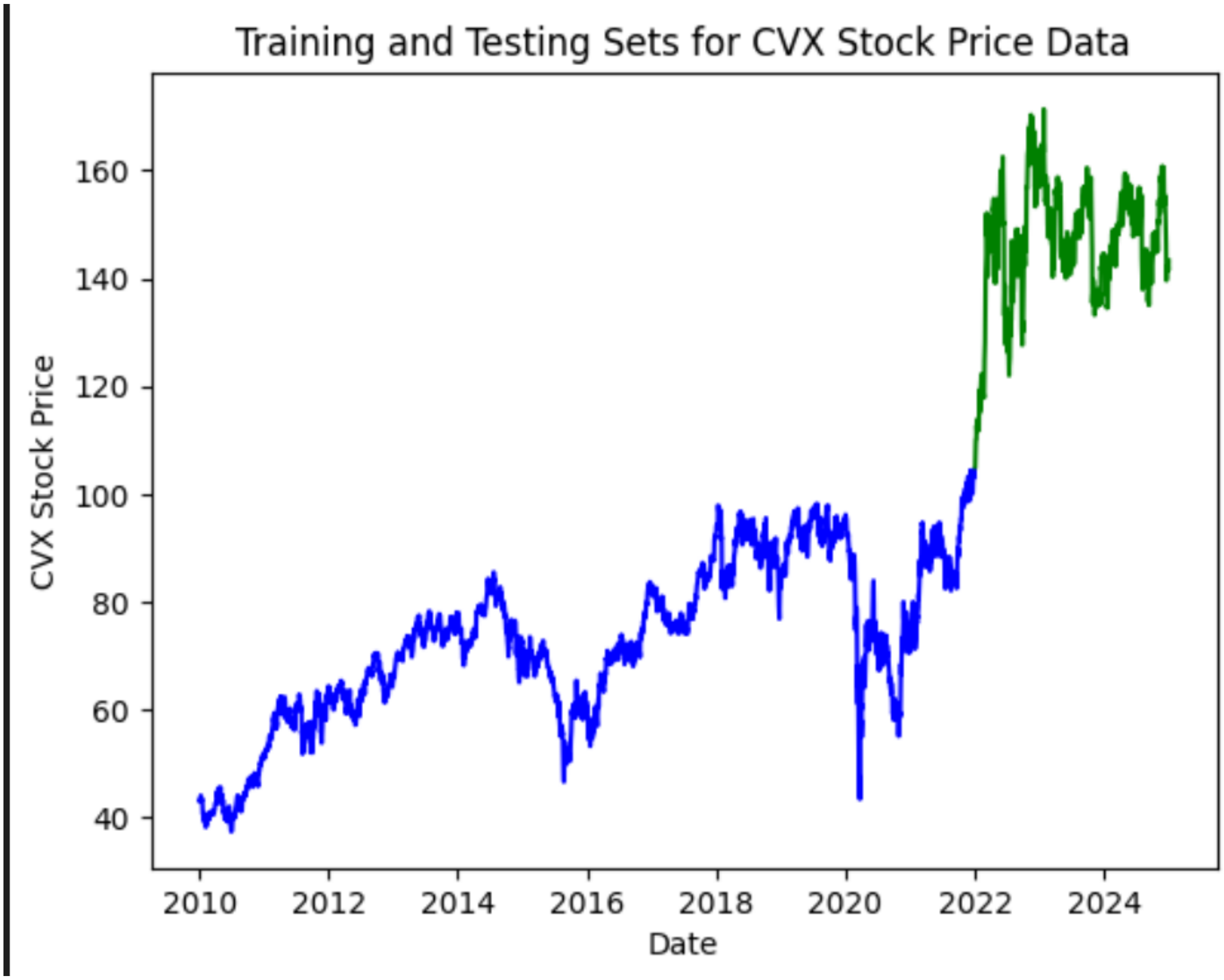
Python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import yfinance as yf
from statistics import mean
import tensorflow
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from sklearn.metrics import mean_squared_error

# Historical Stock Extraction from Yahoo Finance

# data_download = yf.download('CVX', start='2010-01-01', end='2024-01-01')
# data_download = data_download.drop(columns=["High", "Low", "Open", "Volume"],
axis=1)
# print(data_download.head())

# data_download.to_csv('CVX_historical_data.csv')

chevron_data =
pd.read_csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/CVX_historical_
data.csv",
                           index_col="Date", parse_dates=["Date"])

# Plotting daily Chevron stock closing prices

train = chevron_data[chevron_data.index < pd.to_datetime("2022-01-02",
format='%Y-%m-%d')]
test = chevron_data[chevron_data.index >= pd.to_datetime("2022-01-02",
format='%Y-%m-%d')]

plt.plot(train, color = "blue")
```

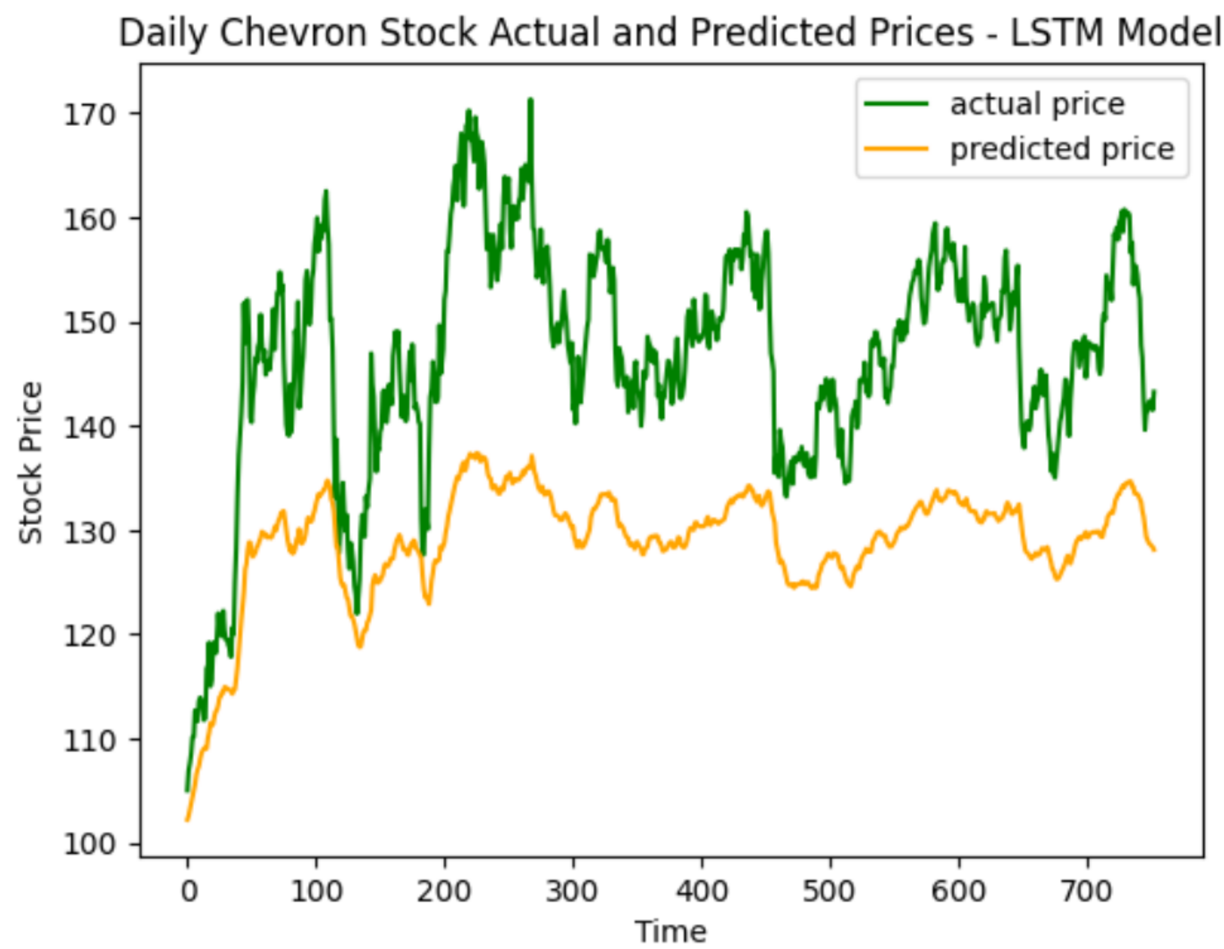
```
plt.plot(test, color = "green")
plt.ylabel('CVX Stock Price')
plt.xlabel('Date')
plt.title("Training and Testing Sets for CVX Stock Price Data")
plt.show()
# Rescaling Chevron data

chevron_data["Close_sc"] = (chevron_data["Close"]-
min(chevron_data["Close"]))/(max(chevron_data["Close"].values)-
min(chevron_data["Close"]))

train_set = chevron_data[chevron_data.index < pd.to_datetime("2022-01-02",
format='%Y-%m-%d')]
test_set = chevron_data[chevron_data.index >= pd.to_datetime("2022-01-02",
format='%Y-%m-%d')]
train_set = train_set.loc[:, "Close_sc"].values
test_set = test_set.loc[:, "Close_sc"].values

nsteps=60

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 np.array(x), np.array(y)

train_x, train_y=split_sequence(train_set, nsteps)
# Fitting LSTM model

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

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

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=chevron_data.loc[:,"Close_sc"][len(chevron_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(chevron_data["Close"].values)-
min(chevron_data["Close"]))+min(chevron_data["Close"])
test_y=test_y*(max(chevron_data["Close"].values)-
min(chevron_data["Close"]))+min(chevron_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 Chevron Stock Actual and Predicted Prices - LSTM Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
# Fitting GRU Architecture Model
```

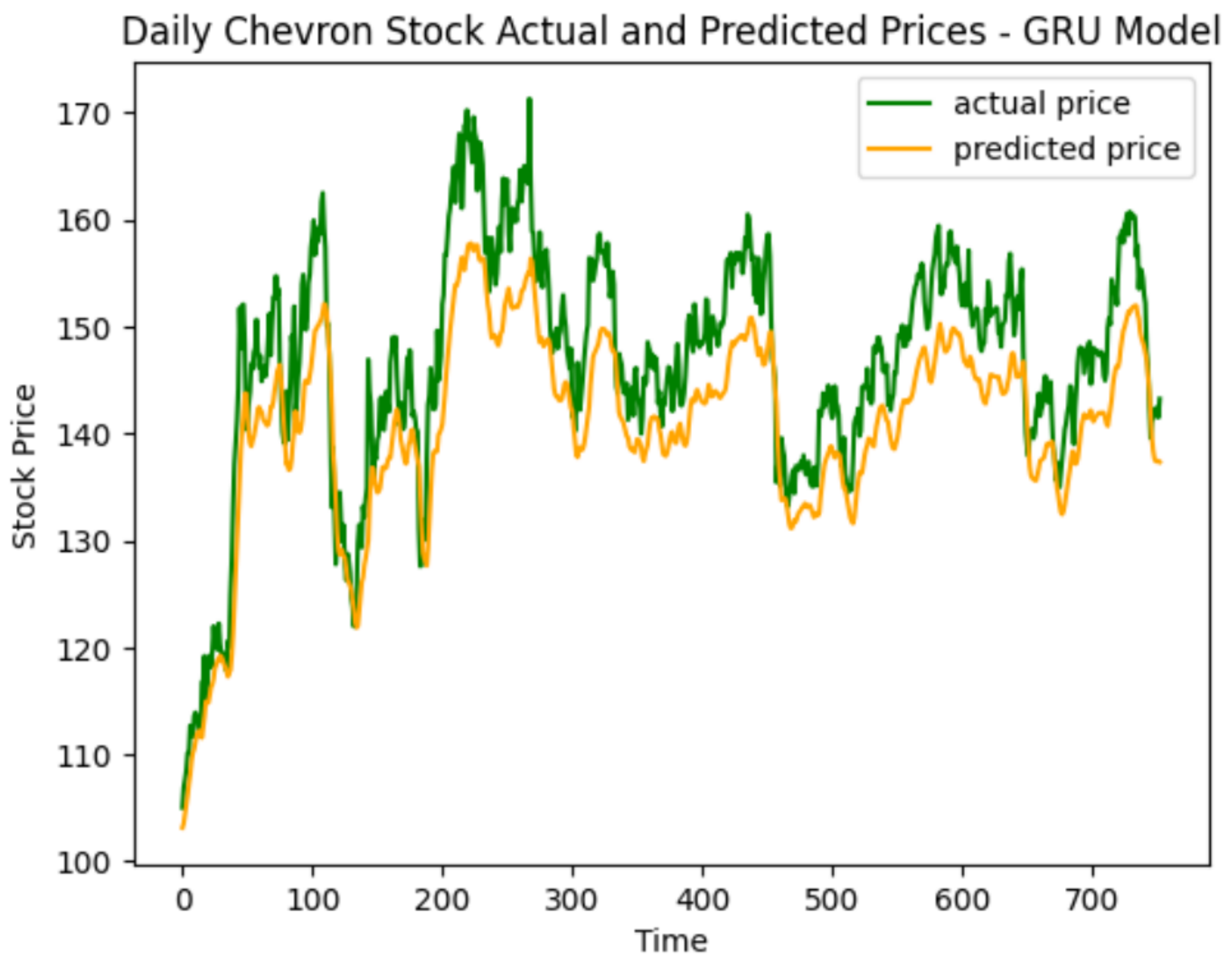
```
model_gru = Sequential()
model_gru.add(GRU(units=6, activation="tanh", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

# 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(chevron_data["Close"].values)-
min(chevron_data["Close"]))+min(chevron_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 Chevron Stock Actual and Predicted Prices - GRU Model")
plt.xlabel("Time")
plt.ylabel("Stock Price")
plt.legend()
plt.show()
```




```
Epoch 1/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - loss: 0.0045
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 3.1114e-04
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 3.0604e-04
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 2.1896e-04
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 2.1180e-04
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
Accuracy within 10%: 0.2709
Accuracy within 15%: 0.8194
Accuracy within 20%: 0.9987
```




```
  super().__init__(**kwargs)
93/93 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0524
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - loss: 0.0041
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step - loss: 4.3160e-04
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - loss: 1.4043e-04
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 1.4531e-04
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step
accuracy within 10%: 0.9973
accuracy within 15%: 1
accuracy within 20%: 1
```



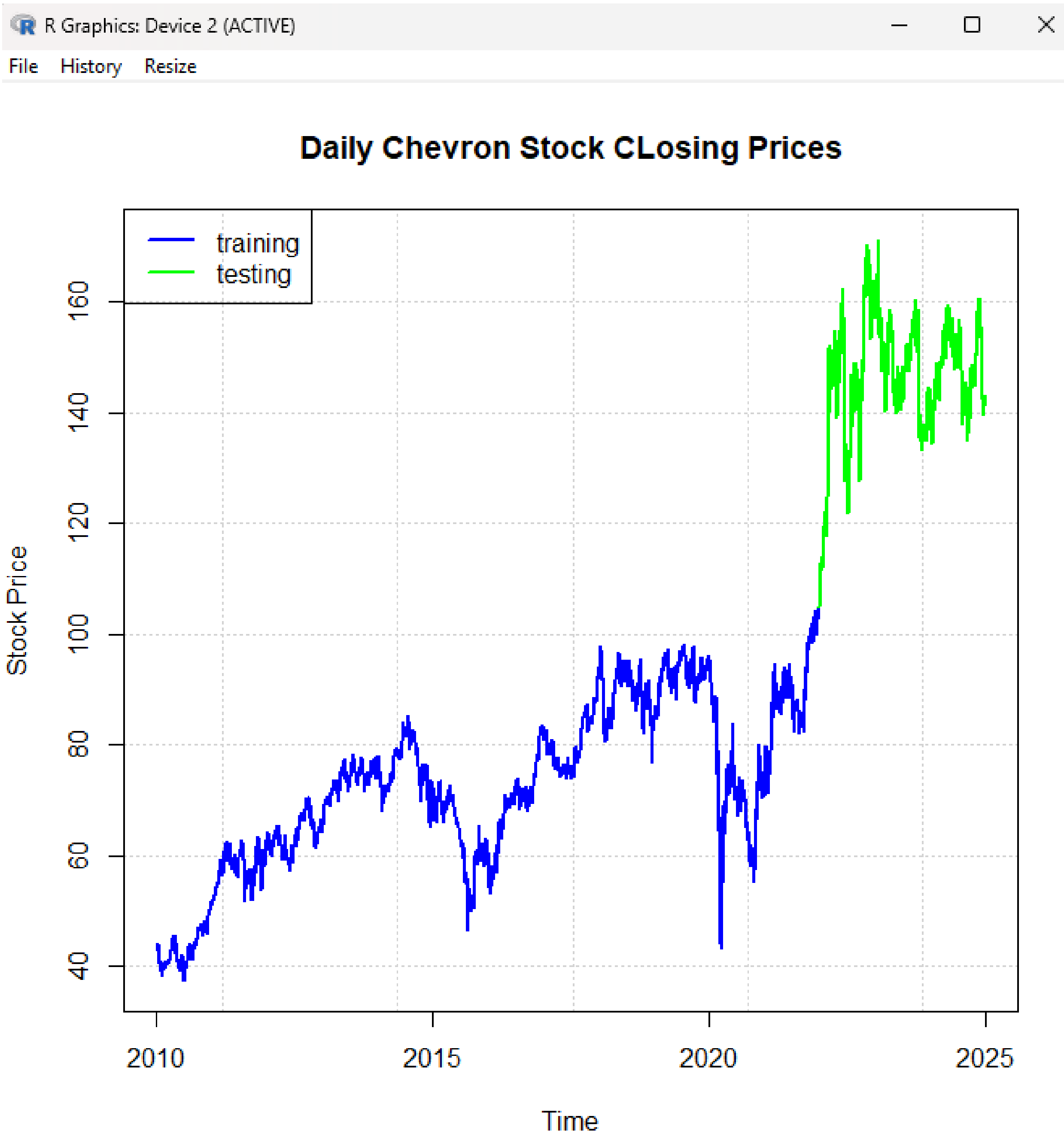
R

```
# STAT 574 HW4 Problem 1

# install.packages("keras3")
library(readr)
library(keras3)

cvx_data =
read.csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/CVX_historical_dat
a.csv",
header=T, sep=",")

# Splitting data into training and testing sets.

cvx_data$Year = as.numeric(format(as.Date(cvx_data$Date, format="%Y-%m-%d"),
"%Y"))

train_data = cvx_data[which(cvx_data$Year<2022), 1:2]
```

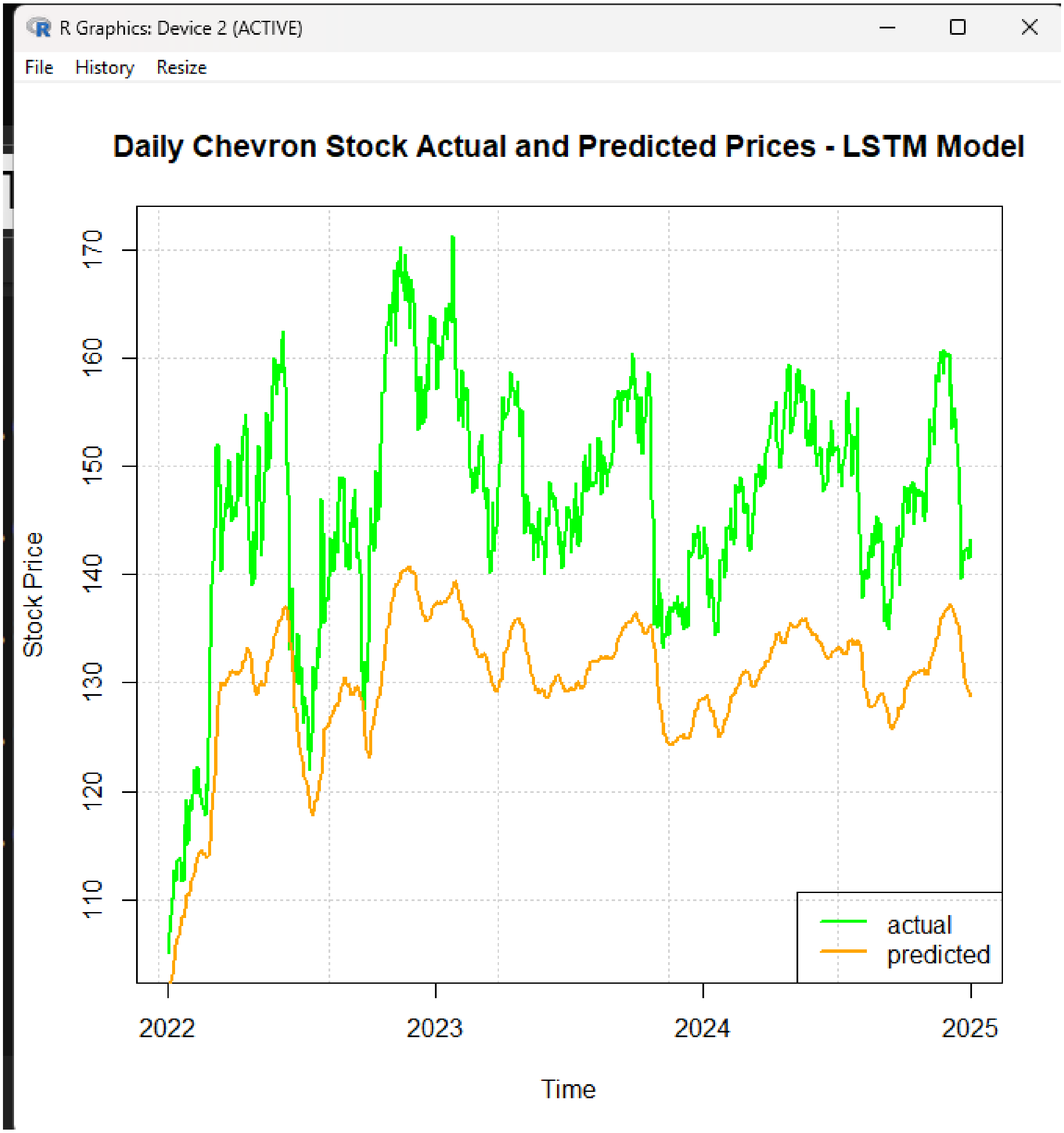
```
test_data = cvx_data[which(cvx_data$Year>=2022), 1:2]

# Plotting training and testing data

plot(as.POSIXct(cvx_data$Date), cvx_data$Close, main="Daily Chevron 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, lwd=2, col=c("blue", "green"))

# Scaling prices to fall within [0,1]

price = as.numeric(cvx_data$Close)
price_sc = (price-min(price))/(max(price)-min(price))

# Creating train_x and train_y

nsteps = 60
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

LSTM_model = keras_model_sequential()

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")
```

```
LSTM_model %>% fit(train_x, train_y, batch_size=32, epochs=5)
pred_y = LSTM_model %>% predict(test_x, batch_size=32)

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)))
print(paste("Accuracy within 15%:", round(mean(accuracy15), 4)))
print(paste("Accuracy within 20%:", round(mean(accuracy20), 4)))

# Plotting actual and predicted values for testing data

plot(as.POSIXct(test_data$Date), test_y_re, type="l", lwd=2, col="green",
main="Daily Chevron Stock Actual and Predicted Prices - LSTM Model",
xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test_data$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()

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")

gru_model %>% fit(train_x, train_y, batch_size=32, epochs=5)

pred_y_gru = gru_model %>% predict(test_x, batch_size=32)

pred_y_re_gru = pred_y_gru*(max(price)-min(price))+min(price)

# Computing prediction accuracy

accuracy10_gru = ifelse(abs(test_y_re-pred_y_re_gru)<0.10*test_y_re, 1, 0)
accuracy15_gru = ifelse(abs(test_y_re-pred_y_re_gru)<0.15*test_y_re, 1, 0)
accuracy20_gru = ifelse(abs(test_y_re-pred_y_re_gru)<0.20*test_y_re, 1, 0)
```

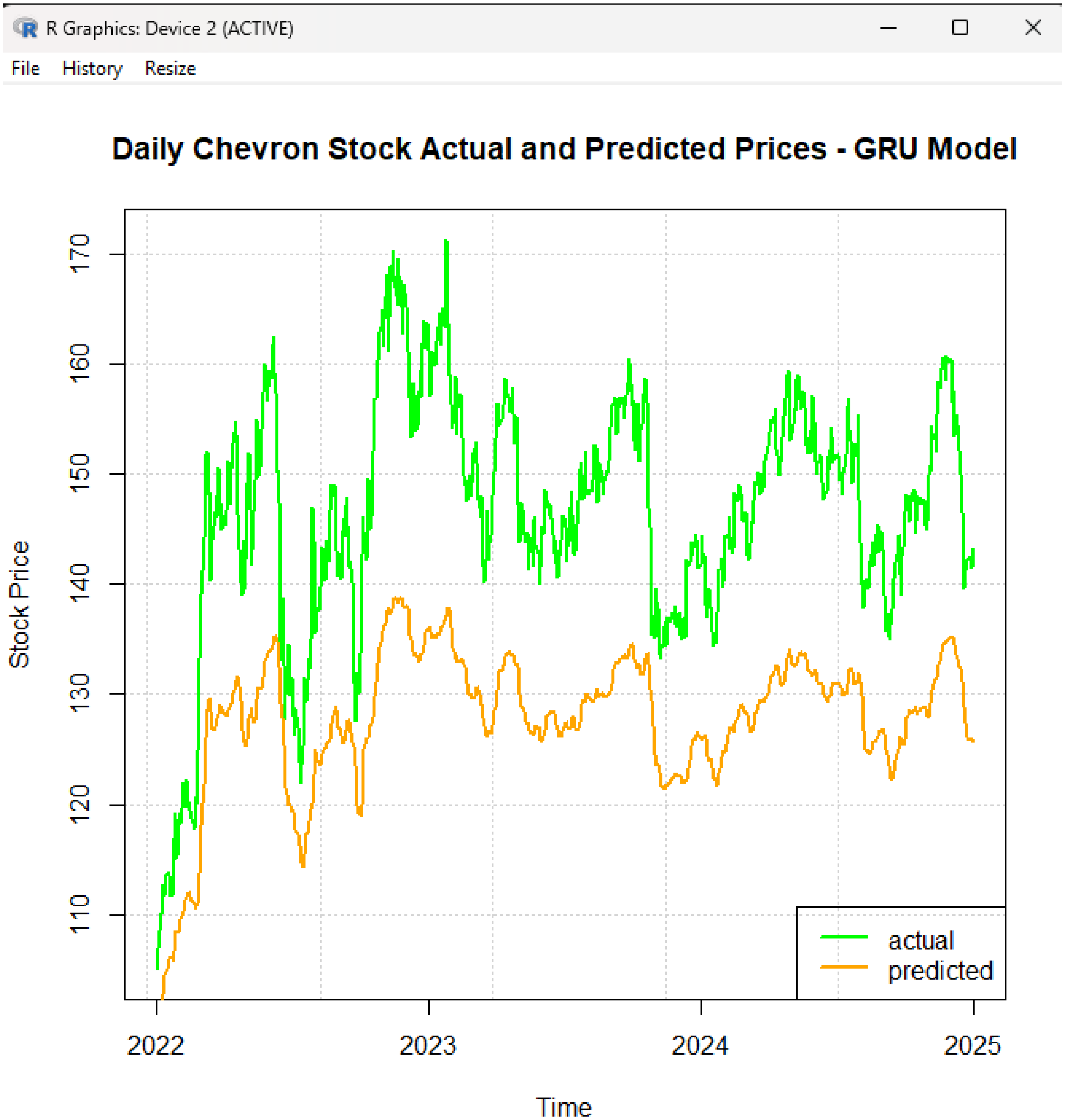
```r
print(paste("Accuracy within 10%:", round(mean(accuracy10_gru), 4)))
print(paste("Accuracy within 15%:", round(mean(accuracy15_gru), 4)))
print(paste("Accuracy within 20%:", round(mean(accuracy20_gru), 4)))

# Plotting actual and predicted values for testing data

plot(as.POSIXct(test_data$Date), test_y_re, type="l", lwd=2, col="green",
main="Daily Chevron Stock Actual and Predicted Prices - GRU Model",
xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test_data$Date), pred_y_re_gru, lwd=2, col="orange")
legend("bottomright", c("actual", "predicted"), lty=1, lwd=2,
col=c("green","orange"))
```


R Graphics: Device 2 (ACTIVE)
File History Resize
Daily Chevron Stock CLosing Prices
training
testing
Stock Price
40
60
80
100
120
140
160
2010
2015
2020
2025
Time

```
  super().__init__(**kwargs)
Epoch 1/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 8s 48ms/step - loss: 0.0065
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 61ms/step - loss: 6.0162e-04
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - loss: 4.8289e-04
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - loss: 3.6237e-04
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 6s 64ms/step - loss: 4.0895e-04
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 39ms/step
[1] "Accuracy within 10%: 0.3347"
[1] "Accuracy within 15%: 0.9057"
[1] "Accuracy within 20%: 1"
```


R Graphics: Device 2 (ACTIVE)
File History Resize
Daily Chevron Stock Actual and Predicted Prices - LSTM Model
Stock Price
170
160
150
140
130
120
110
2022
2023
2024
2025
Time
actual
predicted

```
Epoch 1/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 4s 23ms/step - loss: 0.0070
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 3s 27ms/step - loss: 3.7110e-04
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - loss: 2.8554e-04
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - loss: 3.1382e-04
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - loss: 2.6224e-04
24/24 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
[1] "Accuracy within 10%: 0.1633"
[1] "Accuracy within 15%: 0.8035"
[1] "Accuracy within 20%: 0.9987"
```



Problem 2

Python

```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from sklearn.metrics import mean_squared_error
from statistics import mean
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU

cvx_data =
pd.read_csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/CVX_historical_
data_shock.csv",
                       index_col="Date", parse_dates=["Date"])
cvx_data = cvx_data.drop(["Close"], axis=1)

# Splitting the data into training and testing sets.

train = cvx_data[cvx_data.index < pd.to_datetime("2022-01-02", format='%Y-%m-
%d')]
test = cvx_data[cvx_data.index >= pd.to_datetime("2022-01-02", 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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)
# Fitting LSTM Model

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

model_lstm = Sequential()
model_lstm.add(LSTM(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
```

```
model_lstm.add(Dense(units=1))

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

test_x, test_y = split_sequence(inputs)
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

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 = pd.DataFrame({'accuracy': accuracy, 'cut-off':cutoff})
max_accuracy = max(accuracy)
optimal=df[df['accuracy']==max_accuracy]
print(optimal)
# Fitting GRU Architecture

model_gru = Sequential()
model_gru.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps, features)))
model_gru.add(Dense(units=1))

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

pred_prob = model_gru.predict(test_x)

cutoff=[]
accuracy=[]
for i in range(99):
    tp=0
```

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

```
  super().__init__(**kwargs)
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.5266
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.4353
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.4260
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.4176
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 0.3981
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
    accuracy  cut-off
0   0.954092     0.01
1   0.954092     0.02
2   0.954092     0.03
3   0.954092     0.04
4   0.954092     0.05
..       ...      ...
74  0.954092     0.75
75  0.954092     0.76
76  0.954092     0.77
77  0.954092     0.78
78  0.954092     0.79
```

```
  super().__init__(**kwargs)
93/93 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - loss: 2.4998
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 1s 16ms/step - loss: 0.6285
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 16ms/step - loss: 0.4338
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.4110
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 18ms/step - loss: 0.4252
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step
    accuracy  cut-off
0   0.954092     0.01
1   0.954092     0.02
2   0.954092     0.03
3   0.954092     0.04
4   0.954092     0.05
..       ...      ...
80  0.954092     0.81
81  0.954092     0.82
82  0.954092     0.83
83  0.954092     0.84
84  0.954092     0.85
```

R

```
# STAT 574 HW4 Problem 2

library(readr)
library(dplyr)
library(keras3)

cvx_data =
read.csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/CVX_historical_dat
a_shock.csv",
header=T, sep=",")
```

```
# Splitting data into training and testing sets.

cvx_data$Year = as.numeric(format(as.Date(cvx_data$Date, format="%m/%d/%Y"),
"%Y"))
train_data = cvx_data[which(cvx_data$Year<2022), 1:2]
test_data = cvx_data[which(cvx_data$Year>=2022), 1:2]

nsteps = 60
train_matrix = matrix(nrow=nrow(train_data)-nsteps, ncol=nsteps+1)
for (i in 1:(nrow(train_data)-nsteps)) {
    train_matrix[i,] = cvx_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,] =
cvx_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

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")

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) {
```

```r
    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])),])

# Fitting GRU model

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")

gru_biclass %>% fit(train_x, train_y, batch_size=32, epochs=5)

# Computing prediction accuracy for testing data.

pred_prob_gru = gru_biclass %>% predict(test_x)
match_gru = cbind(test_y, pred_prob_gru)
tp_gru = matrix(NA, nrow=nrow(match_gru), ncol=99)
tn_gru = matrix(NA, nrow=nrow(match_gru), ncol=99)

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

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

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

```
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 9s 53ms/step - loss: 0.4590
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - loss: 0.4235
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 5s 56ms/step - loss: 0.4255
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - loss: 0.4224
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 5s 54ms/step - loss: 0.4137
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 66ms/step
```

```
      [,1]      [,2]
 [1,] 0.01 0.9540918
 [2,] 0.02 0.9540918
 [3,] 0.03 0.9540918
 [4,] 0.04 0.9540918
 [5,] 0.05 0.9540918
 [6,] 0.06 0.9540918
 [7,] 0.07 0.9540918
 [8,] 0.08 0.9540918
 [9,] 0.09 0.9540918
[10,] 0.10 0.9540918
[11,] 0.11 0.9540918
[12,] 0.12 0.9540918
[13,] 0.13 0.9540918
[14,] 0.14 0.9540918
[15,] 0.15 0.9540918
[16,] 0.16 0.9540918
[17,] 0.17 0.9540918
[18,] 0.18 0.9540918
[19,] 0.19 0.9540918
[20,] 0.20 0.9540918
[21,] 0.21 0.9540918
```

```
[22,] 0.22 0.9540918
[23,] 0.23 0.9540918
[24,] 0.24 0.9540918
[25,] 0.25 0.9540918
[26,] 0.26 0.9540918
[27,] 0.27 0.9540918
[28,] 0.28 0.9540918
[29,] 0.29 0.9540918
[30,] 0.30 0.9540918
[31,] 0.31 0.9540918
[32,] 0.32 0.9540918
[33,] 0.33 0.9540918
[34,] 0.34 0.9540918
[35,] 0.35 0.9540918
[36,] 0.36 0.9540918
[37,] 0.37 0.9540918
[38,] 0.38 0.9540918
[39,] 0.39 0.9540918
[40,] 0.40 0.9540918
[41,] 0.41 0.9540918
[42,] 0.42 0.9540918
[43,] 0.43 0.9540918
[44,] 0.44 0.9540918
[45,] 0.45 0.9540918
[46,] 0.46 0.9540918
[47,] 0.47 0.9540918
[48,] 0.48 0.9540918
[49,] 0.49 0.9540918
[50,] 0.50 0.9540918
```

```
[51,] 0.51 0.9540918
[52,] 0.52 0.9540918
[53,] 0.53 0.9540918
[54,] 0.54 0.9540918
[55,] 0.55 0.9540918
[56,] 0.56 0.9540918
[57,] 0.57 0.9540918
[58,] 0.58 0.9540918
[59,] 0.59 0.9540918
[60,] 0.60 0.9540918
[61,] 0.61 0.9540918
[62,] 0.62 0.9540918
[63,] 0.63 0.9540918
[64,] 0.64 0.9540918
[65,] 0.65 0.9540918
[66,] 0.66 0.9540918
[67,] 0.67 0.9540918
[68,] 0.68 0.9540918
[69,] 0.69 0.9540918
[70,] 0.70 0.9540918
[71,] 0.71 0.9540918
[72,] 0.72 0.9540918
[73,] 0.73 0.9540918
[74,] 0.74 0.9540918
[75,] 0.75 0.9540918
[76,] 0.76 0.9540918
[77,] 0.77 0.9540918
[78,] 0.78 0.9540918
[79,] 0.79 0.9540918
```

```
Epoch 1/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 5s 23ms/step - loss: 0.4926
Epoch 2/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - loss: 0.4268
Epoch 3/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - loss: 0.4262
Epoch 4/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 22ms/step - loss: 0.4215
Epoch 5/5
93/93 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - loss: 0.4116
16/16 ━━━━━━━━━━━━━━━━━━━━ 1s 27ms/step
```

```
       [,1]      [,2]
 [1,] 0.01 0.9540918
 [2,] 0.02 0.9540918
 [3,] 0.03 0.9540918
 [4,] 0.04 0.9540918
 [5,] 0.05 0.9540918
 [6,] 0.06 0.9540918
 [7,] 0.07 0.9540918
 [8,] 0.08 0.9540918
 [9,] 0.09 0.9540918
[10,] 0.10 0.9540918
[11,] 0.11 0.9540918
[12,] 0.12 0.9540918
[13,] 0.13 0.9540918
[14,] 0.14 0.9540918
[15,] 0.15 0.9540918
[16,] 0.16 0.9540918
[17,] 0.17 0.9540918
[18,] 0.18 0.9540918
[19,] 0.19 0.9540918
[20,] 0.20 0.9540918
[21,] 0.21 0.9540918
[22,] 0.22 0.9540918
```

```
[23,] 0.23 0.9540918
[24,] 0.24 0.9540918
[25,] 0.25 0.9540918
[26,] 0.26 0.9540918
[27,] 0.27 0.9540918
[28,] 0.28 0.9540918
[29,] 0.29 0.9540918
[30,] 0.30 0.9540918
[31,] 0.31 0.9540918
[32,] 0.32 0.9540918
[33,] 0.33 0.9540918
[34,] 0.34 0.9540918
[35,] 0.35 0.9540918
[36,] 0.36 0.9540918
[37,] 0.37 0.9540918
[38,] 0.38 0.9540918
[39,] 0.39 0.9540918
[40,] 0.40 0.9540918
[41,] 0.41 0.9540918
[42,] 0.42 0.9540918
[43,] 0.43 0.9540918
[44,] 0.44 0.9540918
[45,] 0.45 0.9540918
[46,] 0.46 0.9540918
[47,] 0.47 0.9540918
[48,] 0.48 0.9540918
[49,] 0.49 0.9540918
[50,] 0.50 0.9540918
[51,] 0.51 0.9540918
```

```
[52,] 0.52 0.9540918
[53,] 0.53 0.9540918
[54,] 0.54 0.9540918
[55,] 0.55 0.9540918
[56,] 0.56 0.9540918
[57,] 0.57 0.9540918
[58,] 0.58 0.9540918
[59,] 0.59 0.9540918
[60,] 0.60 0.9540918
[61,] 0.61 0.9540918
[62,] 0.62 0.9540918
[63,] 0.63 0.9540918
[64,] 0.64 0.9540918
[65,] 0.65 0.9540918
[66,] 0.66 0.9540918
[67,] 0.67 0.9540918
[68,] 0.68 0.9540918
[69,] 0.69 0.9540918
[70,] 0.70 0.9540918
[71,] 0.71 0.9540918
[72,] 0.72 0.9540918
[73,] 0.73 0.9540918
[74,] 0.74 0.9540918
[75,] 0.75 0.9540918
[76,] 0.76 0.9540918
[77,] 0.77 0.9540918
[78,] 0.78 0.9540918
[79,] 0.79 0.9540918
```

Problem 3

Python

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

seattle_weather =
pd.read_csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/w
eather_description.csv",
index_col = "datetime", parse_dates=["datetime"])
seattle_weather['Seattle'].value_counts()
code_conditions = {"sky is clear": "clear", "light rain": "rain", "overcast
clouds": "cloudy",
                   "broken clouds": "cloudy", "mist": "fog", "scattered clouds":
"cloudy",
                   "few clouds": "cloudy", "moderate rain": "rain", "light
intensity drizzle": "rain",
                   "fog": "fog", "haze": "fog", "heavy intensity rain": "cloudy",
"smoke": "cloudy",
                   "light snow": "snow", "light intensity shower rain": "rain",
                   "proximity thunderstorm": "rain", "very heavy rain": "rain",
"drizzle": "rain",
                   "thunderstorm": "rain", "thunderstorm with light rain":
"rain",
                   "heavy intensity drizzle": "rain", "heavy snow": "snow",
"shower rain": "rain",
                   "thunderstorm with heavy rain": "rain", "thunderstorm with
rain": "rain",
                   "light shower snow": "snow", "squalls": "rain", "heavy
intensity shower rain": "rain"}

seattle_weather['Seattle'] = seattle_weather['Seattle'].map(code_conditions)
seattle_weather = pd.get_dummies(seattle_weather['Seattle'])
# LSTM Models

# Rain

time_start = 2012
time_end = 2016
def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "rain"].values
    test = seattle_weather.loc[f"{time_end+1}":, "rain"].values
```

```python
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

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

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_rain = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_rain = fitted_model.predict(test_x)
# Fog Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "fog"].values
    test = seattle_weather.loc[f"{time_end+1}":, "fog"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

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

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_fog = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_fog = fitted_model.predict(test_x)
# Cloudy Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "cloudy"].values
    test = seattle_weather.loc[f"{time_end+1}":, "cloudy"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)
```

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

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

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_cloudy = split_sequence(test_set)
test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)

pred_prob_cloudy = fitted_model.predict(test_x)
# Snow Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "snow"].values
    test = seattle_weather.loc[f"{time_end+1}":, "snow"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

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

fitted_model.compile(loss="binary_crossentropy")
```

```
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
test_x, test_snow = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_snow = fitted_model.predict(test_x)
# Clear Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "clear"].values
    test = seattle_weather.loc[f"{time_end+1}":, "clear"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

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

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)
test_x, test_clear = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_clear = fitted_model.predict(test_x)
# Computing prediction accuracy for LSTM models

pred_prob_all = np.concatenate((pred_prob_rain, pred_prob_fog, pred_prob_clear,
pred_prob_cloudy, pred_prob_snow), axis=1)
pred_prob_all = pd.DataFrame(pred_prob_all)
```

```python
pred_class = pred_prob_all.idxmax(axis=1)

test_all = np.c_[test_rain, test_fog, test_clear, test_cloudy, test_snow]
test_all = pd.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))
```

```
0.7094
```

```python
# GRU Models

# Rain Model

time_start = 2012
time_end = 2016

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "rain"].values
    test = seattle_weather.loc[f"{time_end+1}":, "rain"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)
```

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

fitted_model = Sequential()
fitted_model.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_rain = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_rain = fitted_model.predict(test_x)
# Fog Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "fog"].values
    test = seattle_weather.loc[f"{time_end+1}":, "fog"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

fitted_model = Sequential()
fitted_model.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))
```

```
fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_fog = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_fog = fitted_model.predict(test_x)
# Cloudy Model


def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "cloudy"].values
    test = seattle_weather.loc[f"{time_end+1}":, "cloudy"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

fitted_model = Sequential()
fitted_model.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_fog = split_sequence(test_set)

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

```python
pred_prob_fog = fitted_model.predict(test_x)
# Snow Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "snow"].values
    test = seattle_weather.loc[f"{time_end+1}":, "snow"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

fitted_model = Sequential()
fitted_model.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_fog = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_fog = fitted_model.predict(test_x)
# Clear Model

def train_test_split(time_start, time_end):
    train = seattle_weather.loc[f"{time_start}":f"{time_end}", "clear"].values
    test = seattle_weather.loc[f"{time_end+1}":, "clear"].values
    return train, test
```

```
train_set, test_set = train_test_split(time_start, time_end)

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 np.array(x), np.array(y)

train_x, train_y = split_sequence(train_set)

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

fitted_model = Sequential()
fitted_model.add(GRU(units=6, activation="sigmoid", input_shape=(nsteps,
features)))
fitted_model.add(Dense(units=1, activation="sigmoid"))

fitted_model.compile(loss="binary_crossentropy")
fitted_model.fit(train_x, train_y, epochs=5, batch_size=32)

test_x, test_fog = split_sequence(test_set)

test_x = test_x.reshape(test_x.shape[0], test_x.shape[1], features)
pred_prob_fog = fitted_model.predict(test_x)
# Computing prediction accuracy for GRU model.

pred_prob_all = np.concatenate((pred_prob_rain, pred_prob_fog, pred_prob_cloudy,
pred_prob_snow, pred_prob_clear), axis=1)
pred_prob_all = pd.DataFrame(pred_prob_all)
pred_class = pred_prob_all.idxmax(axis=1)

test_all = np.c_[test_rain, test_fog, test_cloudy, test_snow, test_clear]
test_all = pd.DataFrame(test_all)
true_class = test_all.idxmax(axis=1)

match_gru = []
for i in range(len(pred_class)):
    if pred_class[i] == true_class[i]:
        match_gru.append(1)
```

```
    else:
        match_gru.append(0)

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

```
0.6338
```

R

```
# STAT 574 HW4 Problem 3

library(readr)
library(keras3)

weather_data =
read.csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/weat
her_description.csv")
table(DT <- weather_data$Seattle)

DT = ifelse(DT=="sky is clear", "clear", ifelse(DT %in% c("broken clouds", "few
clouds", "overcast clouds", "scattered clouds", "smoke"), "cloudy",
ifelse(DT %in% c("heavy shower snow", "heavy snow", "light shower snow", "light
snow", "shower snow", "snow"), "snow",
ifelse(DT %in% c("fog", "haze", "mist"), "fog", "rain"))))

weather_data$clear = ifelse(DT=="clear", 1, 0)
weather_data$cloudy = ifelse(DT=="cloudy",1,0)
weather_data$rain = ifelse(DT=="rain",1,0)
weather_data$snow = ifelse(DT=="snow",1,0)
weather_data$fog = ifelse(DT=="fog",1,0)
weather_data$year = format(as.Date(weather_data$datetime, format="%m/%d/%y"),
"%Y")

rnn_model = function(modelname, varname) {
    train_data = weather_data[which(weather_data$year<2017), varname]
    test_data = weather_data[which(weather_data$year==2017), 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,] = weather_data[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,] =
weather_data[(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)]

    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')

    fitted_model %>% fit(train_x, train_y, batch_size=32, epochs=1)
    pred_prob = fitted_model %>% predict(test_x)
    return(list(test_y, pred_prob))
}

accuracy = function() {
    test_y = bind_cols(test_clear, test_cloudy, test_snow, test_fog, test_rain)
    colnames(test_y) = 1:5
    true_class = as.numeric(apply(test_y, 1, function(x)
    colnames(test_y)[which.max(x)]))
    pred_prob = bind_cols(pred_prob_clear, pred_prob_cloudy, pred_prob_snow,
pred_prob_fog, pred_prob_rain)
    colnames(pred_prob) = 1:5
    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_clear = rnn_model('lstm', 'clear')
```

```
test_clear = list_clear[1]
pred_prob_clear = list_clear[2]

list_cloudy = rnn_model('lstm', 'cloudy')
test_cloudy = list_cloudy[1]
pred_prob_cloudy = list_cloudy[2]

list_snow = rnn_model('lstm', 'snow')
test_snow = list_snow[1]
pred_prob_snow = list_snow[2]

list_fog = rnn_model('lstm', 'fog')
test_fog = list_fog[1]
pred_prob_fog = list_fog[2]

list_rain = rnn_model('lstm', 'rain')
test_rain = list_rain[1]
pred_prob_rain = list_rain[2]
```

```
## [1] 0.71638
```

```
# Running GRU Binary Classification Models

list_clear = rnn_model('gru', 'clear')
test_clear = list_clear[1]
pred_prob_clear = list_clear[2]

list_cloudy = rnn_model('gru', 'cloudy')
test_cloudy = list_cloudy[1]
pred_prob_cloudy = list_cloudy[2]

list_snow = rnn_model('gru', 'snow')
test_snow = list_snow[1]
pred_prob_snow = list_snow[2]

list_fog = rnn_model('gru', 'fog')
test_fog = list_fog[1]
pred_prob_fog = list_fog[2]

list_rain = rnn_model('gru', 'rain')
test_rain = list_rain[1]
pred_prob_rain = list_rain[2]
```

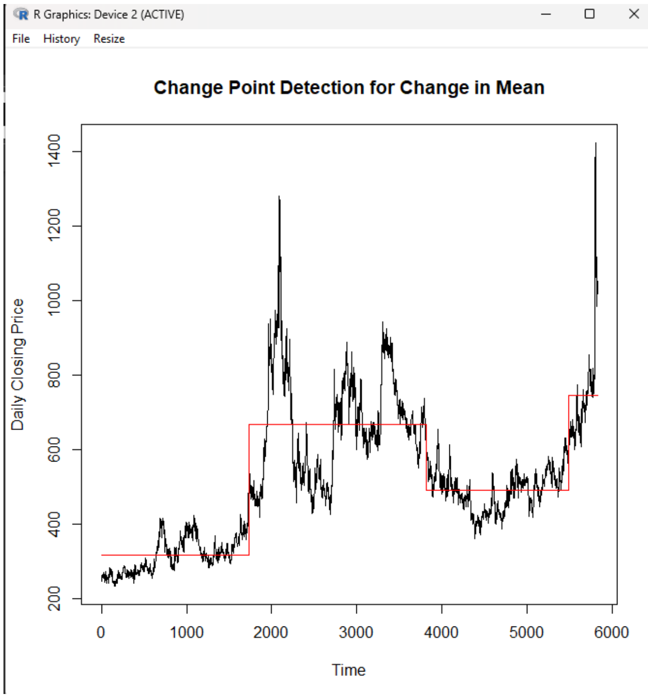
```
## [1] 0.7105
```

Problem 4

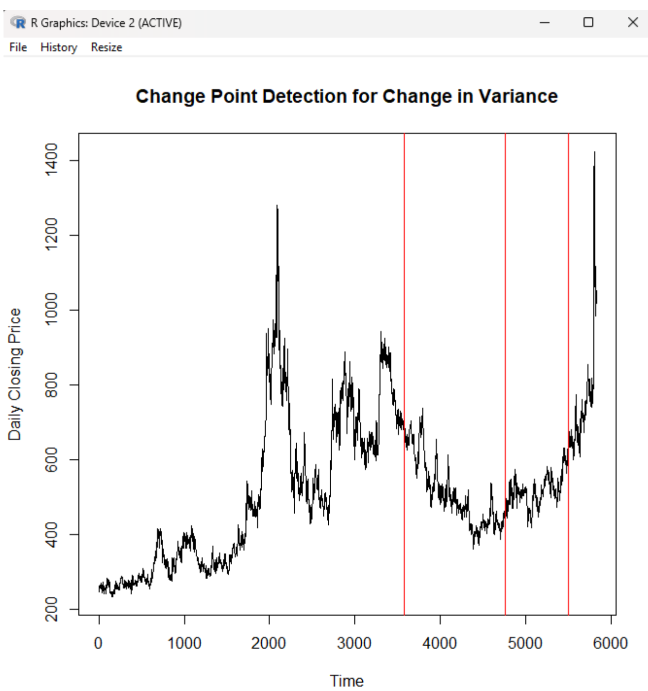
R

```r
# STAT 574 HW4 Problem 4

install.packages("changepoint")
library(readr)
library(changepoint)

gold_data =
read.csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/whea
t_data.csv",
header=T, sep=",")

# Detection of change points for change in mean.

mean_det = cpt.mean(gold_data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(mean_det, cpt.col="red", ylab="Daily Closing Price", main="Change Point
Detection for Change in Mean")
paste("Change Point Locations: ", paste(mean_det@cpts, collapse=", "))

# Detection of change points for change in variance.

var_det = cpt.var(gold_data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(var_det, cpt.col="red", ylab="Daily Closing Price", main="Change Point
Detection for Change in Variance")
paste("Change Point Locations: ", paste(var_det@cpts, collapse=", "))

# Detection of Change Points for change in mean and variance.

mean_var_det = cpt.meanvar(gold_data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(mean_var_det, cpt.col="red", ylab="Daily Closing Price",
main="Change Point Detection for Change in Mean and Variance")
paste("Change Point Locations: ", paste(mean_var_det@cpts, collapse=", "))
```

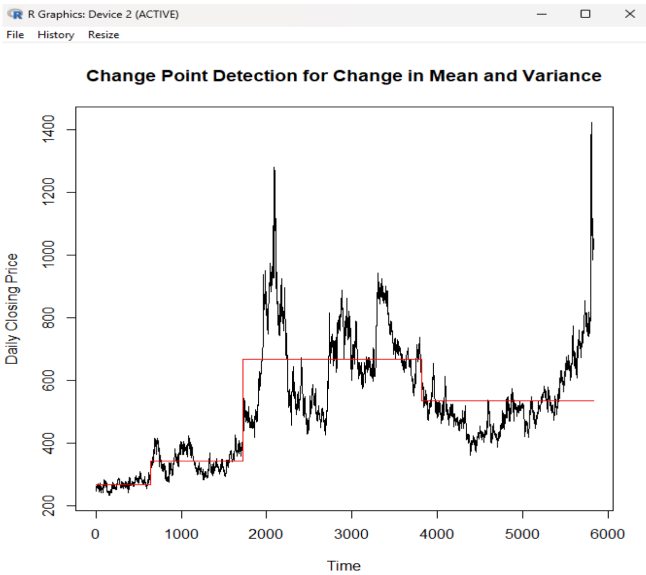
Change in Mean



Change in Variance



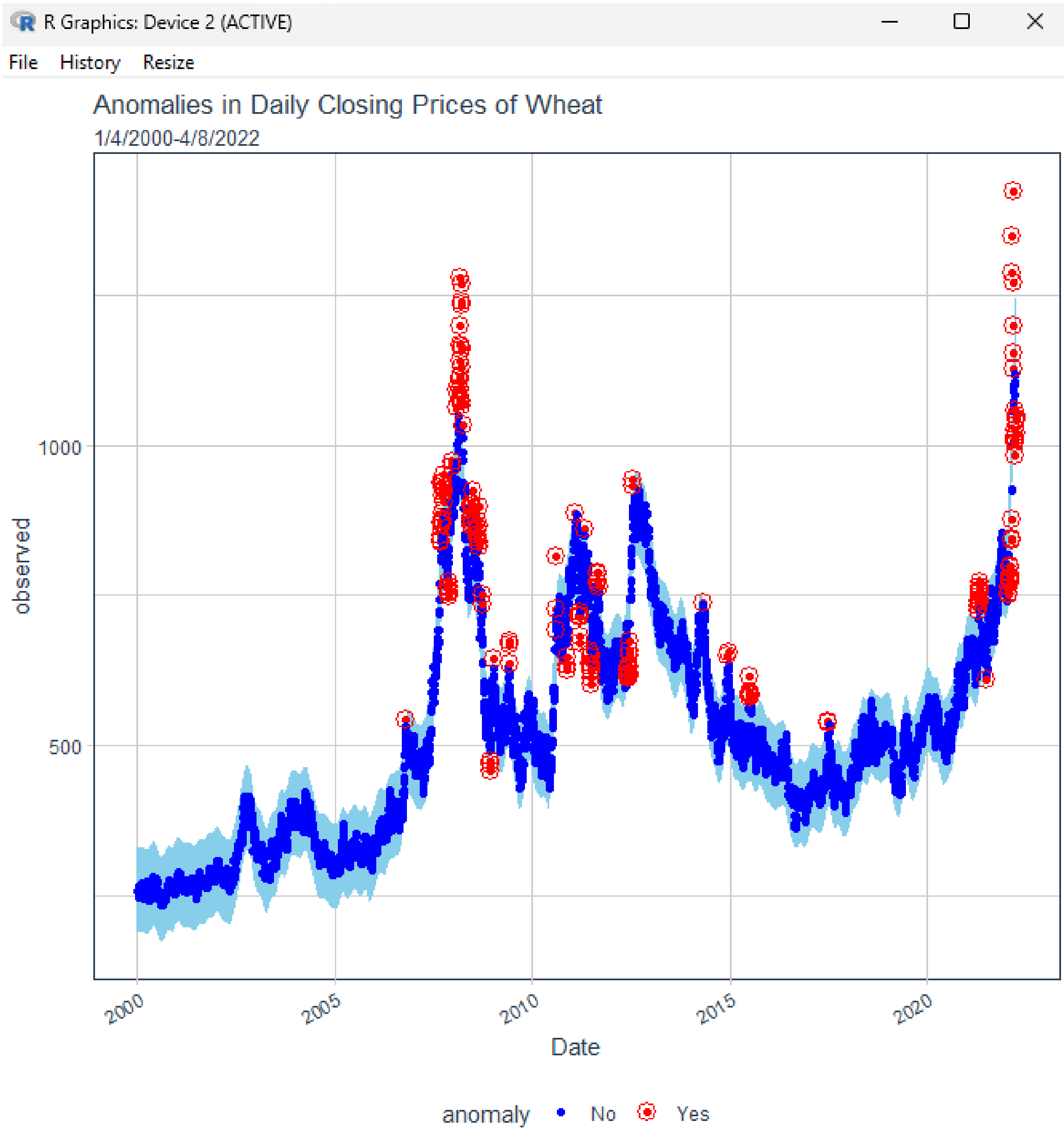
Change in Mean and Variance



Problem 5

R

```
# install.packages("tibbletime")
# install.packages("anomalize")
# install.packages("tidyverse")
library(readr)
library(tibbletime)
library(anomalize)
library(tidyverse)

wheat_data =
read.csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/whea
t_data.csv",
header=T, sep=",")

wheat_data$Date = as.Date(wheat_data$Date, "%m/%d/%Y")

wheat_data_tbl = as_tbl_time(wheat_data, Date)

print(wheat_data_tbl %>% time_decompose(Close, method="stl") %>%
anomalize(remainder,
method="iqr") %>% time_recompose() %>% plot_anomalies(time_recomposed=T,
color_no='blue',
color_yes='red', fill_ribbon='skyblue', size_circles=4) +
labs(title="Anomalies in Daily Closing Prices of Wheat", subtitle="1/4/2000-
4/8/2022"))
```



Problem 6

R

```
# install.packages("BiocManager")
# BiocManager::install("EBImage")
library(keras3)
library(EBImage)
```

```r
# Preparing Data

# Caracals

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAnimalsImages/train/CARACALS")
img_caracals = sample(dir());
train_caracals = list(NULL);
for (i in 1:length(img_caracals)) {
    train_caracals[[i]] = readImage(img_caracals[i])
    train_caracals[[i]] = resize(train_caracals[[i]], 100, 100)
}

# Cheetahs

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAnimalsImages/train/CHEETAHS")
img_cheetahs = sample(dir());
train_cheetahs = list(NULL);
for (i in 1:length(img_cheetahs)) {
    train_cheetahs[[i]] = readImage(img_cheetahs[i])
    train_cheetahs[[i]] = resize(train_cheetahs[[i]], 100, 100)
}

# Lions

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAnimalsImages/train/LIONS")
img_lions = sample(dir());
train_lions = list(NULL);
for (i in 1:length(img_lions)) {
    train_lions[[i]] = readImage(img_lions[i])
    train_lions[[i]] = resize(train_lions[[i]], 100, 100)
}

# Tigers

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAnimalsImages/train/TIGERS")
img_tigers = sample(dir());
train_tigers = list(NULL);
for (i in 1:length(img_tigers)) {
    train_tigers[[i]] = readImage(img_tigers[i])
    train_tigers[[i]] = resize(train_tigers[[i]], 100, 100)
}
```

```
train_pool = c(train_caracals[1:40], train_cheetahs[1:40], train_lions[1:40],
train_tigers[1:40])

# Permutation of image dimensions.

train = aperm(combine(train_pool), c(4,1,2,3))

# Creating image labels.

train_y = c(rep(0,40), rep(1,40), rep(2,40), rep(3,40))
train_lab = to_categorical(train_y)

# Preparing testing data.

# Caracals

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAni
malsImages/train/CARACALS")
img_caracals = sample(dir())
test_caracals = list(NULL)

for (i in 1:length(img_caracals)) {
    test_caracals[[i]] = readImage(img_caracals[i])
    test_caracals[[i]] = resize(test_caracals[[i]], 100, 100)
}

# Cheetahs

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAni
malsImages/train/CHEETAHS")
img_cheetahs = sample(dir())
test_cheetahs = list(NULL)

for (i in 1:length(img_cheetahs)) {
    test_cheetahs[[i]] = readImage(img_cheetahs[i])
    test_cheetahs[[i]] = resize(test_cheetahs[[i]], 100, 100)
}

# Lions

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAni
malsImages/train/LIONS")
img_lions = sample(dir())
test_lions = list(NULL)
```

```r
for (i in 1:length(img_lions)) {
    test_lions[[i]] = readImage(img_lions[i])
    test_lions[[i]] = resize(test_lions[[i]], 100, 100)
}

# Tigers

setwd("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/WildAni
malsImages/train/TIGERS")
img_tigers = sample(dir())
test_tigers = list(NULL)

for (i in 1:length(img_tigers)) {
    test_tigers[[i]] = readImage(img_tigers[i])
    test_tigers[[i]] = resize(test_tigers[[i]], 100, 100)
}

test_pool = c(test_caracals[1:3], test_cheetahs[1:3], test_lions[1:3],
test_tigers[1:3])
test = aperm(combine(test_pool), c(4,1,2,3))
test_y = c(rep(0,3), rep(1,3), rep(2,3), rep(3,3))
test_lab = to_categorical(test_y)

# Fitting CNN Architecture Model.

model_cnn = keras_model_sequential()

model_cnn %>% layer_conv_2d(filters=40, kernel_size=c(3,3),
activation="relu", input_shape=c(100,100,3)) %>%
layer_conv_2d(filters=40, kernel_size=c(3,3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(3,3)) %>% layer_dropout(rate=0.25) %>%
layer_conv_2d(filters=80, kernel_size=c(3,3), activation="relu") %>%
layer_conv_2d(filters=80, kernel_size=c(3,3), activation="relu") %>%
layer_max_pooling_2d(pool_size=c(3,3)) %>% layer_dropout(rate=0.35) %>%
layer_flatten() %>% layer_dense(units=256, activation="relu") %>%
layer_dropout(rate=0.25) %>% layer_dense(units=4, activation="softmax") %>%
compile(loss="categorical_crossentropy", optimizer=optimizer_adam(),
metrics=c("accuracy"))

history = model_cnn %>% fit(train, train_lab, epochs=50, batch_size=40,
validation_split=0.2)

# Computing prediction accuracy for testing set.
```

```
model_cnn %>% evaluate(test, test_lab)

pred_class = as.array(model_cnn %>% predict(test) %>% k_argmax())
print(pred_class)
print(test_y)

print(paste("accuracy= ", round(1-mean(test_y!=pred_class), digits=4)))
```

```
## [1] "accuracy= 0.9167"
```

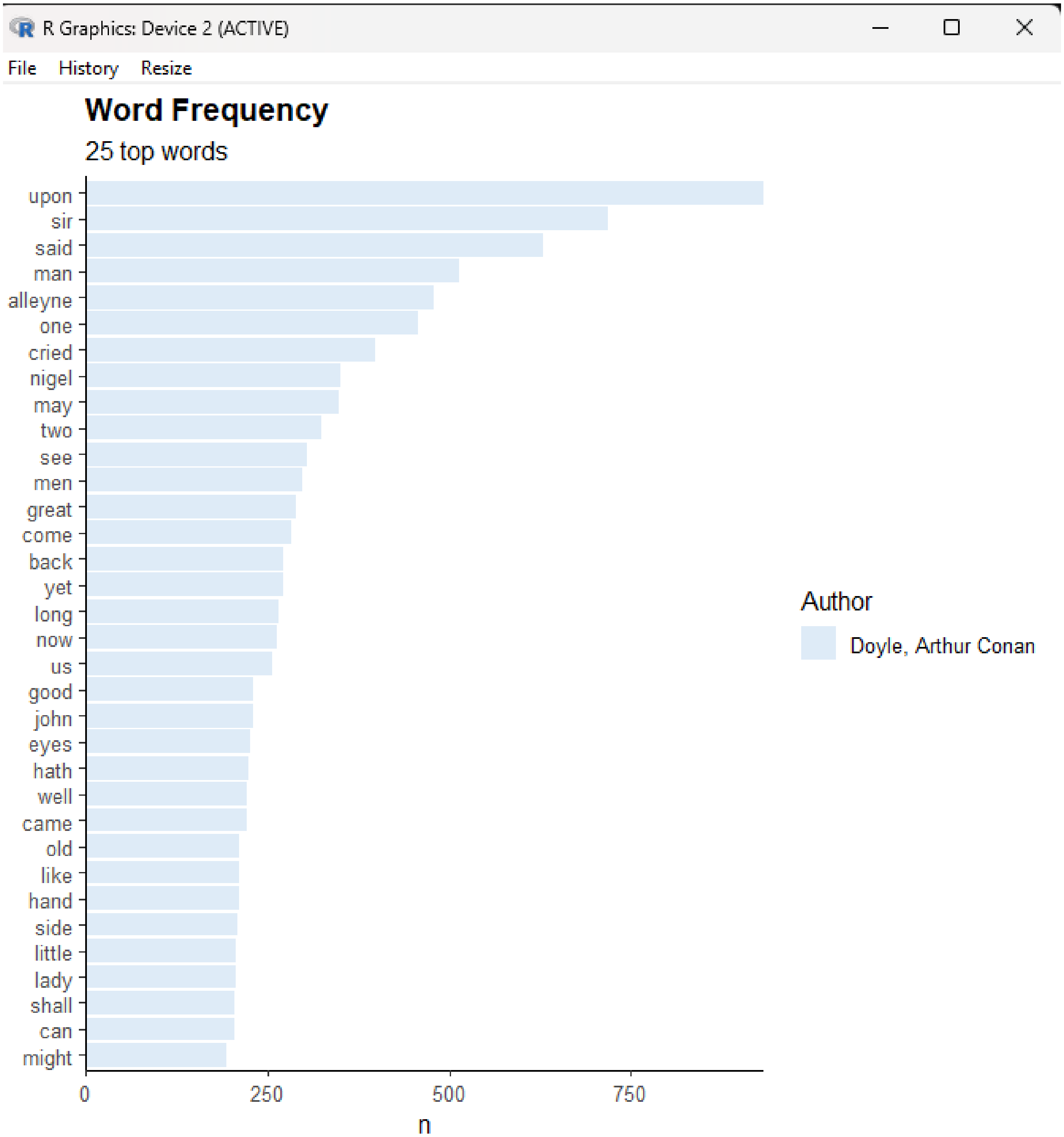
Problem 7

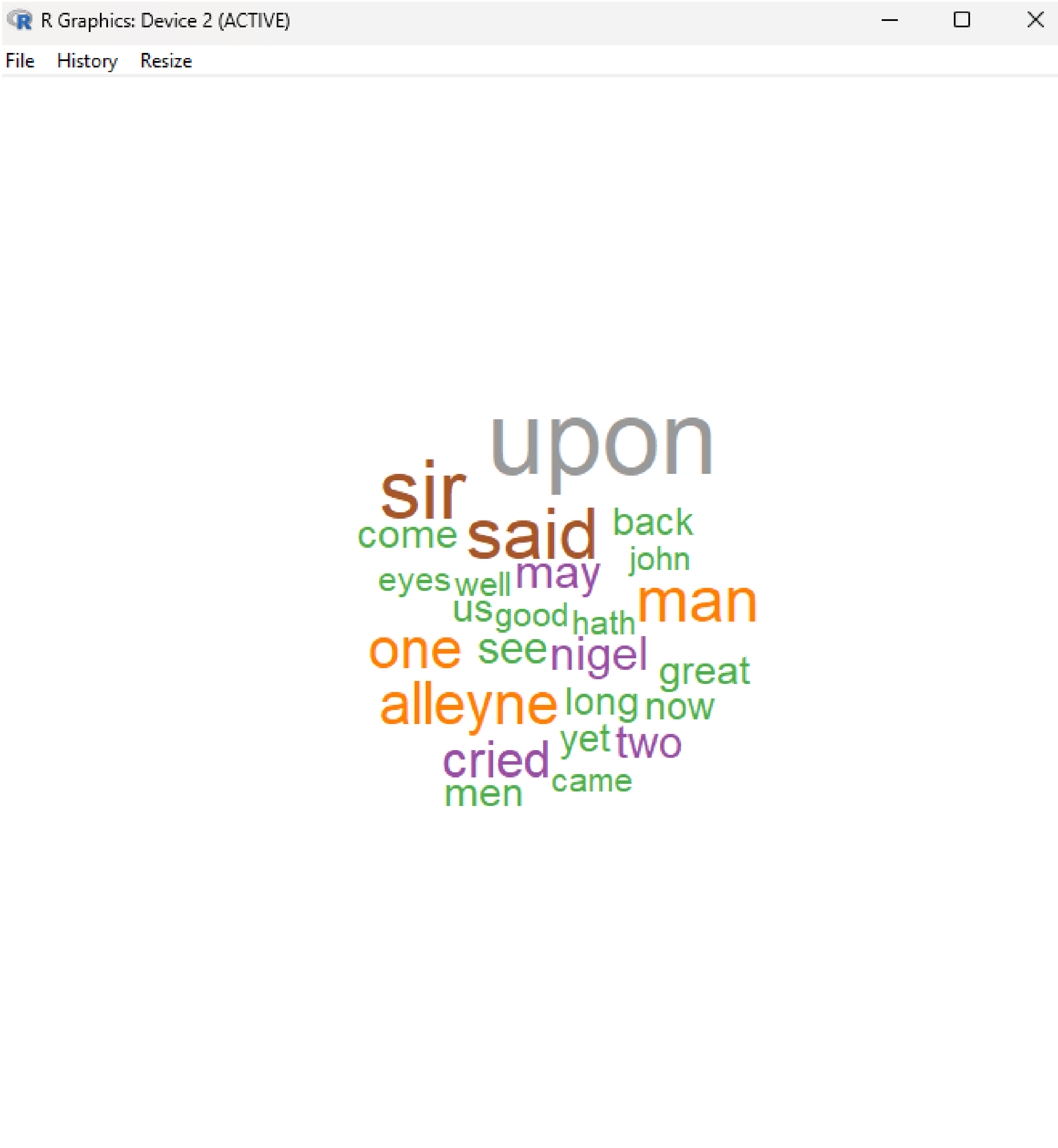
R

```
library(readr)
library(dplyr)
library(gutenbergr)
library(stringr)
library(tidytext)
library(stopwords)
library(tibble)
library(wordcloud)
library(ggplot2)

# Book selected: The White Company by Sir Arthur Conan Doyle

book_sel = gutenberg_download(903, meta_fields="author")

book_sel = as_tibble(book_sel %>%
mutate(document=row_number()) %>%
select(-gutenberg_id))

tidy_book = book_sel %>% unnest_tokens(word, text) %>%
group_by(word) %>% filter(n()>10) %>%
ungroup()

# Identifying and dropping stopwords from text.

stopword = as_tibble(stopwords::stopwords("en"))
stopword = rename(stopword, word=value)
tb = anti_join(tidy_book, stopword, by='word')
```

```
word_count = tb %>% count(word, sort=T)
print(word_count)

# Displaying top 25 words.

tb %>%
count(author, word, sort=T) %>%
filter(n>=190) %>%
mutate(word=reorder(word,n)) %>%
ggplot(aes(word,n)) +
geom_col(aes(fill=author)) +
xlab(NULL) +
scale_y_continuous(expand=c(0, 0)) +
coord_flip() +
theme_classic(base_size=12) +
labs(fill="Author", title="Word Frequency", subtitle="25 top words")+
theme(plot.title = element_text(lineheight=.8, face="bold"))+
scale_fill_brewer()

# Plotting word cloud.

tb %>%
count(word) %>%
with(wordcloud(word, n, max.words=25, colors=brewer.pal(10, "Set1")))
```

```
# A tibble: 25 × 2
   word        n
   <chr>   <int>
 1 upon      931
 2 sir       719
 3 said      629
 4 man       514
 5 alleyne   480
 6 one       458
 7 cried     398
 8 nigel     350
 9 may       348
10 two       325
# ℹ 15 more rows
```

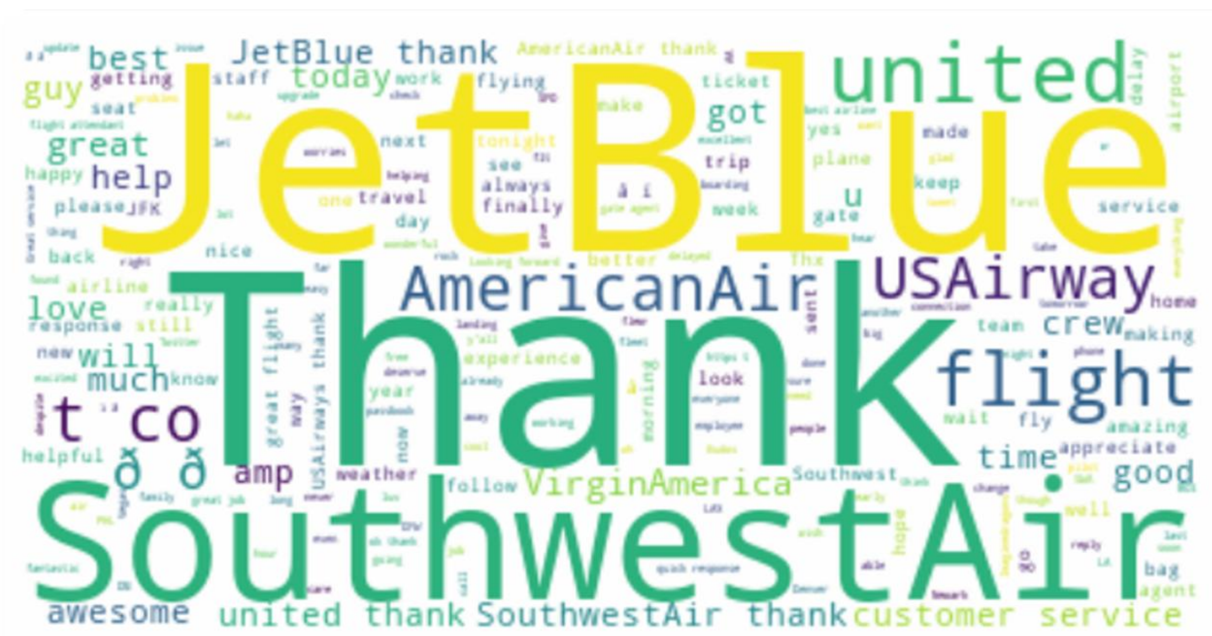

R Graphics: Device 2 (ACTIVE)
File
History
Resize
Word Frequency
25 top words
upon
sir
said
man
alleyne
one
cried
nigel
may
two
see
men
great
come
back
yet
long
now
us
good
john
eyes
hath
well
came
old
like
hand
side
little
lady
shall
can
might
0
250
500
750
n
Author
Doyle, Arthur Conan



Problem 8

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
```

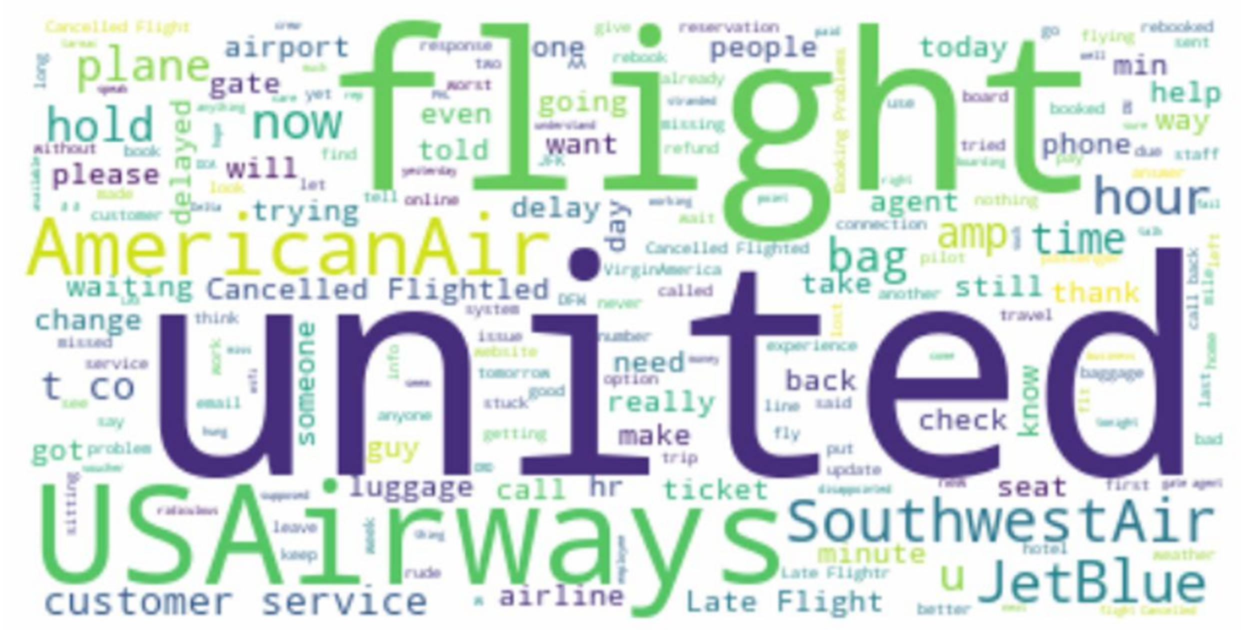
```
from wordcloud import WordCloud
from simpletransformers.classification import ClassificationModel

air_data =
pd.read_csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/U
SAirlinesTweets.csv")
air_data.drop_duplicates(subset=['tweet'], keep='first', inplace=True)
text = " ".join([x for x in air_data.tweet[air_data.sentiment == 'positive']])

# Positive Tweets

# Plotting wordclouds for positive news

text = " ".join([x for x in air_data.tweet[air_data.sentiment == 'positive']])
wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Jet Blue has the most Positive Tweets

```
# Negative Tweets
```

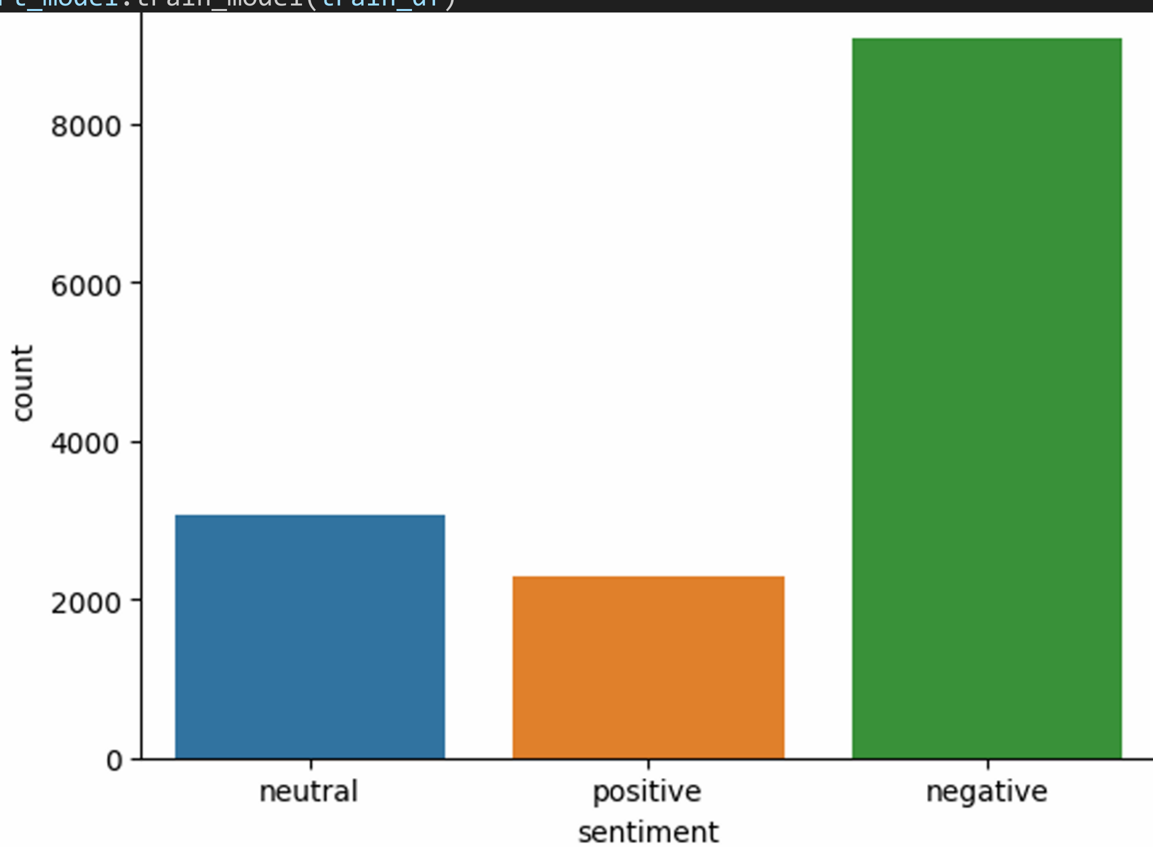
```
# Plotting wordcloud for negative tweets

text = " ".join([x for x in air_data.tweet[air_data.sentiment == 'negative']])
wordcloud = WordCloud(background_color='white').generate(text)

plt.figure(figsize=(8,6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



United Airlines has the most Negative Tweets

```
# Plotting bar graph for sentiments.

sns.countplot(x = air_data['sentiment'])
air_data['sentiment'].value_counts()

# Training model

train, test = train_test_split(air_data, test_size=0.2, random_state=576574)

bert_model = ClassificationModel('bert', 'bert-base-cased', num_labels=3,
        args={'reprocess_input_data': True, 'overwrite_output_dir': True},
        use_cuda=False)

def making_label(st):
    if (st=='positive'):
```

```
        return 0
    elif (st=='neutral'):
        return 2
    else:
        return 1

train['label'] = train['sentiment'].apply(making_label)
test['label'] = test['sentiment'].apply(making_label)

train_df = pd.DataFrame({
    'text': train['tweet'][:6000].replace(r'\n', '', regex=True),
    'label': train['label'][:6000]
})

eval_df = pd.DataFrame({
    'text': train['tweet'][-1000:].replace(r'\n', '', regex=True),
    'label': test['label'][-1000:]
})
bert_model.train_model(train_df)
```



```
sentiment
negative    9080
neutral     3057
positive    2290
Name: count, dtype: int64
```

```
(750, 0.5900795254607996)
```

Multinomial Logistic Regression

```
# Multinomial Logistic Regression Analysis for tweets.

from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import re, string
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
tweet_df =
pd.read_csv("C:/Users/coryg/OneDrive/Desktop/STAT_574_Data_Mining/hw4STAT574S25/U
SAirlinesTweets.csv")
tweet_df = tweet_df[['sentiment', 'tweet']].dropna()

def preprocessing_tweets(tweets):
    tweets = tweets.lower()
    tweets = re.sub(r"http\S+|www\S+|https\S+", '', tweets, flags=re.MULTILINE)
    tweets = re.sub(r'\@[\w]+|\#', '', tweets)
    tweets = tweets.translate(str.maketrans('', '', string.punctuation))
    return tweets

tweet_df['cleaned_tweets'] = tweet_df['tweet'].apply(preprocessing_tweets)

# Text Vectorization to extract top 50 words from tweets

vectorization = CountVectorizer(max_features=50, stop_words='english')
```

```
X_mat = vectorization.fit_transform(tweet_df['cleaned_tweets'])
X_df = pd.DataFrame(X_mat.toarray(),
columns=vectorization.get_feature_names_out())
y = tweet_df['sentiment']

labeler = LabelEncoder()
y_encode = labeler.fit_transform(y)
```

```
# Split into 80% training and 20% testing sets and run the model.

X_train, X_test, y_train, y_test = train_test_split(X_df, y_encode,
test_size=0.2,
                                                    random_state=5720255)

multi_logistic = LogisticRegression(multi_class='multinomial', solver='lbfgs',
                                    max_iter=900, random_state=987022)
multi_logistic.fit(X_train, y_train)
```

```
# Words most associated with Positive Tweets

coefs = pd.DataFrame(multi_logistic.coef_,
columns=vectorization.get_feature_names_out())
coefs.index = labeler.classes_

top_pos = coefs.loc['positive'].sort_values(ascending=False).head(10)
top_neg = coefs.loc['negative'].sort_values(ascending=False).head(10)

print("Positive tweets associated with the following words:")
print(top_pos)
print("\nNegative tweets associated with the following words:")
print(top_neg)
```

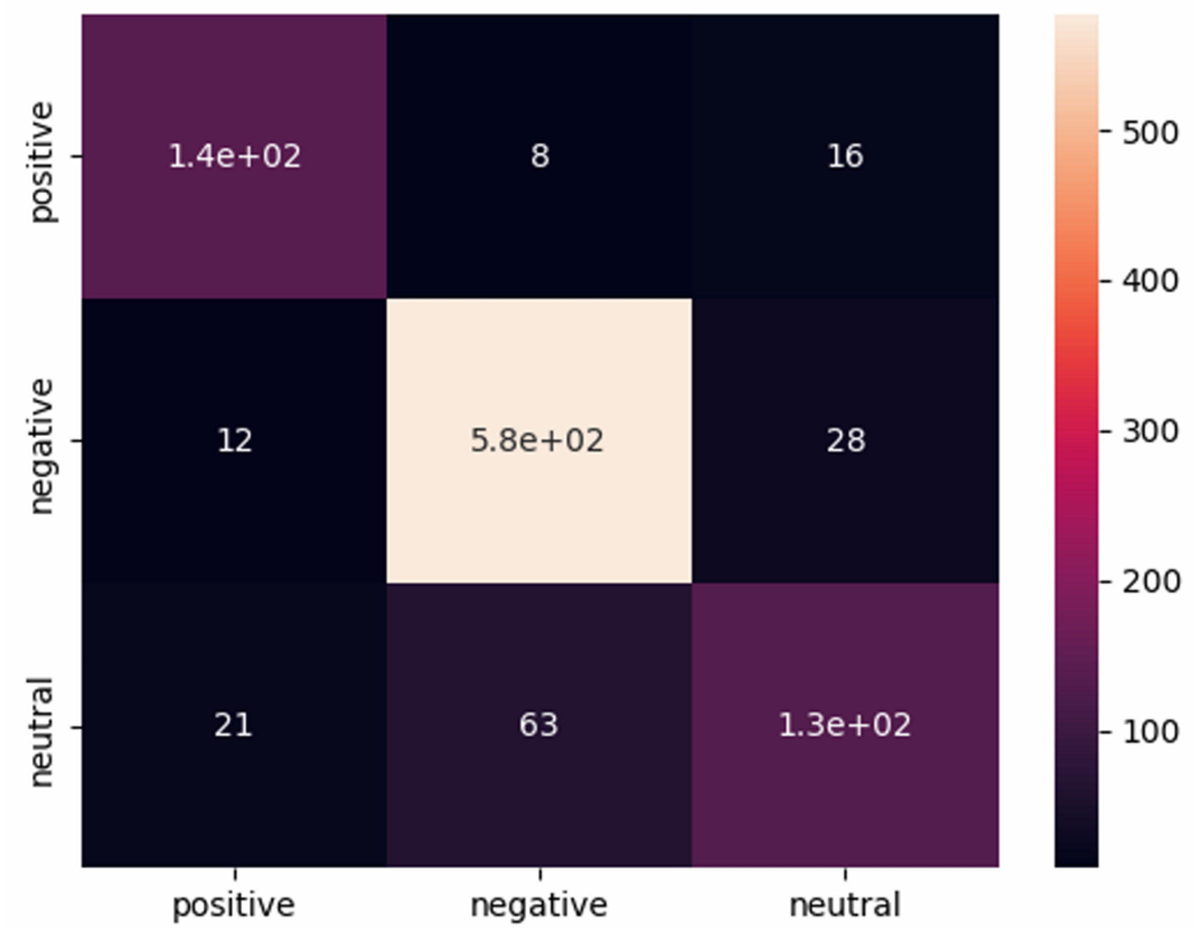
```
Positive tweets associated with the following words:
thank       2.010026
great       1.656956
thanks      1.540171
good        0.899002
airline     0.605125
guys        0.456910
got         0.424672
really      0.392620
flying      0.349597
home        0.331437
Name: positive, dtype: float64

Negative tweets associated with the following words:
hours         1.339060
hold          1.234786
delayed       1.215004
hour          1.163527
delay         0.990636
waiting       0.865747
bag           0.842824
cancelled     0.809081
phone         0.794711
dont          0.785270
Name: negative, dtype: float64
```

```
# Computing prediction accuracy on testing set.

y_pred = multi_logistic.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
print("accuracy: ", round(accuracy, 4))
print("confusion matrix: \n", confusion)
```

```
•   accuracy:  0.68
    confusion matrix:
     [[1735   33   51]
     [ 543   41   43]
     [ 256   11  215]]
```

```
# Computing predicted sentiments for testing set.

result, model_outputs, wrong_predictions = bert_model.eval_model(eval_df)
lst = []
for arr in model_outputs:
    lst.append(np.argmax(arr))

truth = eval_df['label'].tolist()
predicted = lst

# Displaying confusion matrix.

conf_mat = confusion_matrix(truth, predicted)
print(conf_mat)

# Displaying heatmap for confusion matrix.

df_cm = pd.DataFrame(conf_mat, ['positive', 'negative', 'neutral'], ['positive',
'negative', 'neutral'])
sns.heatmap(df_cm, annot=True)
plt.show()
```

```
[[141   8  16]
 [ 12 577  28]
 [ 21  63 134]]
```




```
# Displaying performance metrics

print(sklearn.metrics.classification_report(truth, predicted,
target_names=['positive', 'negative', 'neutral']))
print(sklearn.metrics.accuracy_score(truth, predicted))
```

```
: '              precision    recall  f1-score   support\n\n    positive       0.81
  0.85      0.83       165\n    negative       0.89      0.94      0.91       617\n
  neutral       0.75      0.61      0.68       218\n\n    accuracy
  0.85      1000\n   macro avg       0.82      0.80      0.81      1000\nweighted av
  g       0.85      0.85      0.85      1000\n'
```

```
0.852
```

```
# Using trained model to classify user-defined sentences.

def classify(statement):
    result = bert_model.predict([statement])
    pred_class = np.where(result[1][0] == np.amax(result[1][0]))
    pred_class = int(pred_class[0])
```

```
    sentiment_dict = {0: "Positive", 1: "Negative", 2: "Neutral"}
    print(sentiment_dict[pred_class])
    return

classify("Riding on a plane today.”)
classify("The seats were cramped in my flight.”)
classify("The crew were friendly and attentive.")
```

```
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.34s/it]
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  3.08it/s]
neutral
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.22s/it]
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  2.89it/s]
negative
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.04s/it]
100%|████████████████████████████████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  3.13it/s]
positive
```

```
# Training a Bert based model on negative tweets and computing prediction
accuracy
# for reasons of negative tweets. Testing on mock tweets.

negative_tweets = air_data[air_data.sentiment == 'negative']
negative_tweets = negative_tweets.drop('sentiment', axis=1)
negative_tweets['negativereason'].value_counts()

negative_reasons = negative_tweets['negativereason'].unique()

train, test = train_test_split(negative_tweets, test_size=0.2,
random_state=432648)

neg_bert = ClassificationModel('bert', 'bert-base-cased', num_labels=10,
                               args={'reprocess_input_data': True,
'overwrite_output_dir':True},
                               use_cuda=False)

def making_label(st):
    if (st == "Bad Flight"):
        return 0
    if (st == "Can't Tell"):
```

```
        return 1
    if (st == "Late Flight"):
        return 2
    if (st == "Customer Service Issue"):
        return 3
    if (st == "Flight Booking Problems"):
        return 4
    if (st == "Lost Luggage"):
        return 5
    if (st == "Flight Attendant Complaints"):
        return 6
    if (st == "Cancelled Flight"):
        return 7
    if (st == "Damaged Luggage"):
        return 8
    if (st == "longlines"):
        return 9

train['label'] = train['negativereason'].apply(making_label)
test['label'] = test['negativereason'].apply(making_label)

train_df = pd.DataFrame({
    'text': train['tweet'][:3500].replace(r'\n', ' ', regex=True),
    'label': train['label'][:3500]
})

eval_df = pd.DataFrame({
    'text': test['tweet'][-900:].replace(r'\n', ' ', regex=True),
    'label': test['label'][-900]
})

neg_bert.train_model(train_df)
```

```
(438, 1.5106357800770023)
```

```
# Computing predicted sentiments for testing set.

result, model_outputs, wrong_predictions = neg_bert.eval_model(eval_df)

lst = []
for arr in model_outputs:
    lst.append(np.argmax(arr))

truth = eval_df['label'].tolist()
```

```python
predicted = lst

# Computing predicted accuracy

print(accuracy_score(truth, predicted))

def classify(statement):
    result = neg_bert.predict([statement])
    pred_class = np.where(result[1][0] == np.amax(result[1][0]))
    pred_class = int(pred_class[0])
    sentiment_dict = {0: "Bad Flight", 1: "Can't Tell", 2: "Late Flight", 3:
"Customer Service Issue",
                      4: "Flight Booking Problems", 5: "Lost Luggage", 6: "Flight
Attendant Complaints",
                      7: "Cancelled Flight", 8: "Damaged Luggage", 9:
"longlines"}
    print(sentiment_dict[pred_class])
    return

classify('The staff were rude and disrespectful.')
classify('The flight was late.')
classify('My luggage got lost and was found damaged.')
```

```
0.6344444444444445
```

```
100%|██████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.93s/it]
100%|██████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  2.84it/s]
Customer Service Issue
100%|██████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.43s/it]
100%|██████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  2.90it/s]
Late Flight
100%|██████████████████████████████████████████████████
█████| 1/1 [00:09<00:00,  9.41s/it]
100%|██████████████████████████████████████████████████
█████| 1/1 [00:00<00:00,  3.09it/s]
Lost Luggage
```