STAT 574 HW 4

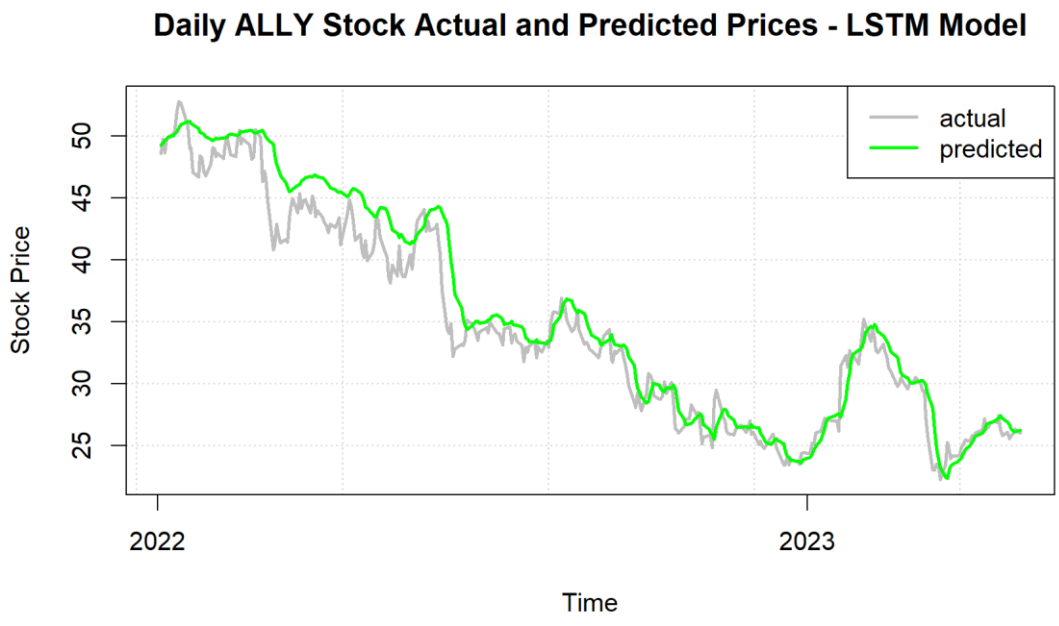
Derrick Edwards

Problem 1. From yahoofinance.com download the historical daily prices of some stocks other than Tesla and Amazon. Model daily high (or open, low, or closing) stock prices by training recurrent neural networks with LSTM and GRU methods. Plot the actual and predicted values and compute prediction accuracy within 10%, 15%, and 20% of the true values. Use R and Python.

R Output

LSTM Model

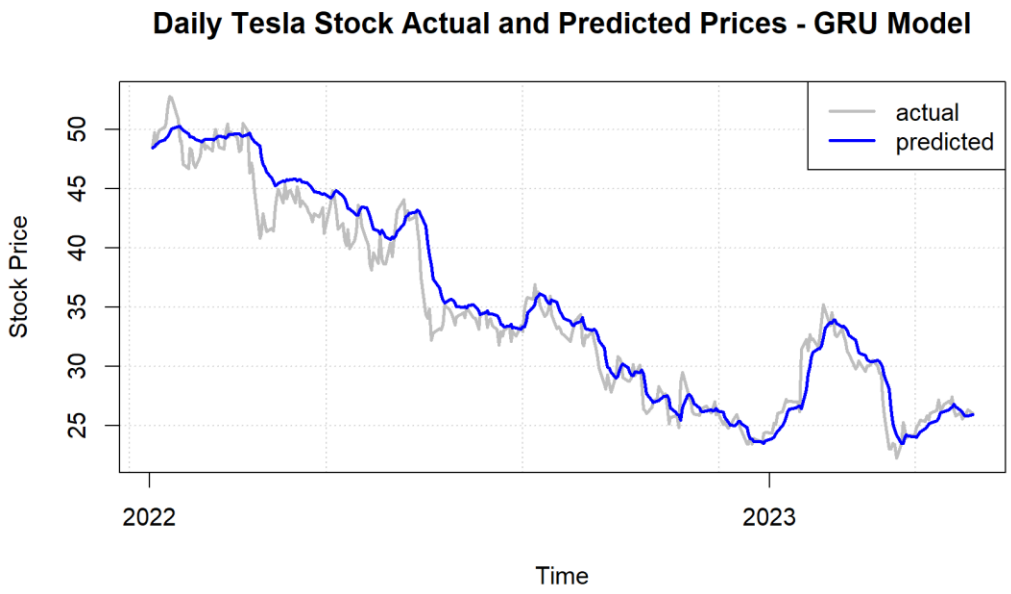
```
[1] "accuracy within 10%: 0.9099"
[1] "accuracy within 15%: 0.979"
[1] "accuracy within 20%: 0.985"
```



GRU Model

```
[1] "accuracy within 10%: 0.9279"
[1] "accuracy within 15%: 0.979"
```

```
[1] "accuracy within 20%: 0.994"
```



R Code

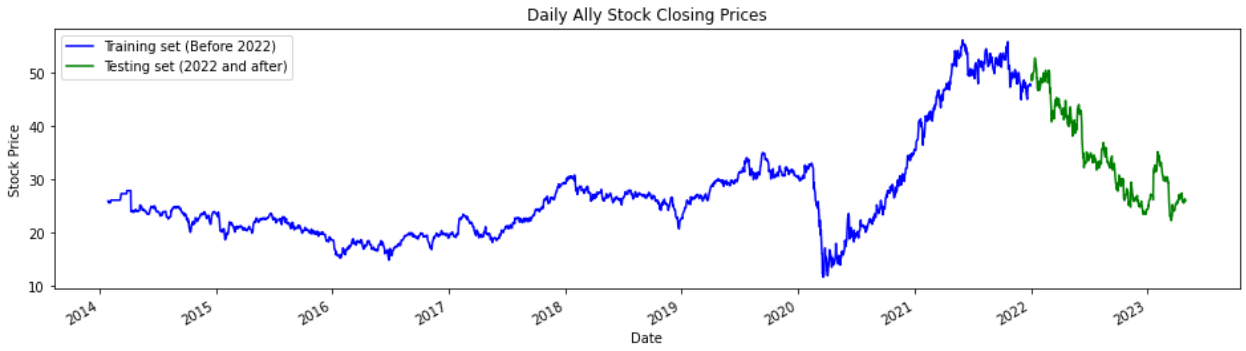
```
library(anytime)

ally_data <- read.csv("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA SETS/ALLY.csv")

ally_data$Date <- anydate(ally_data$Date)

#splitting data into testing and training sets
ally_data$Year<- as.numeric(format(as.Date(ally_data$Date, format="%Y-%m-%d"),"%Y"))
train.data<- ally_data[which(ally_data$Year<2022),1:2]
test.data<- ally_data[which(ally_data$Year>=2022),1:2]


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


ally_data
```

```
#scaling prices to fall in [0,1]
price<- ally_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(keras)
# library(tensorflow)

# install_keras()

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
epochs<- 5
for(i in 1:epochs){
 LSTM.model %>% fit(train.x, train.y, batch_size=32, epochs=5)
 LSTM.model %>% reset_states() #clears the hidden states in network after every batch
}
#predicting for testing data
pred.y<- LSTM.model %>% predict(test.x, batch_size=32)
```

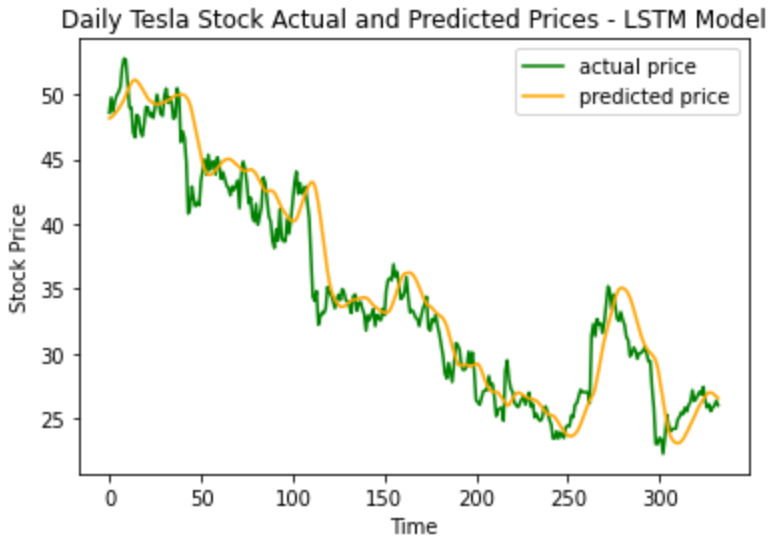
```
#rescaling test.y and pred.y back to the original scale
test.y.re<- test.y*(max(price)-min(price))+min(price)
pred.y.re<- pred.y*(max(price)-min(price))+min(price)
#computing prediction accuracy
accuracy10<- ifelse(abs(test.y.re-pred.y.re)<0.10*test.y.re,1,0)
accuracy15<- ifelse(abs(test.y.re-pred.y.re)<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)))
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$Date), test.y.re, type="l", lwd=2, col="gray",
main="Daily ALLY Stock Actual and Predicted Prices - LSTM Model",
xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="green")
legend("topright", c("actual", "predicted"), lty=1, lwd=2,
col=c("gray","green"))




#####################################################
#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
epochs<- 5
for(i in 1:epochs){
 GRU.model %>% fit(train.x, train.y, batch_size=32, epochs=5)
 GRU.model %>% reset_states()
}
#predicting for testing data

pred.y<- GRU.model %>% predict(test.x, batch_size=32)
```

```
#rescaling pred.y back to the original scale
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$Date), test.y.re, type="l", lwd=2, col="gray",
main="Daily Tesla Stock Actual and Predicted Prices - GRU Model",
xlab="Time", ylab="Stock Price", panel.first=grid())
lines(as.POSIXct(test.data$Date), pred.y.re, lwd=2, col="blue")
legend("topright", c("actual", "predicted"), lty=1, lwd=2,
col=c("gray","blue"))
```

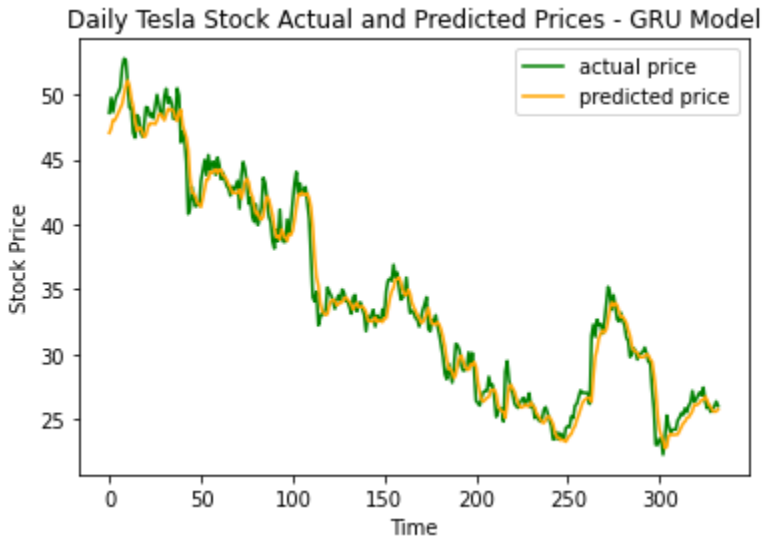
Python Output




```
accuracy within 10%: 0.8829
accuracy within 15%: 0.9429
accuracy within 20%: 0.976
```




```
accuracy within 10%: 0.973
accuracy within 15%: 0.991
accuracy within 20%: 1
```



Python Code

```python
# Problem 1
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

ally_data=pandas.read_csv(r"C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/ALLY.csv", index_col="Date", parse_dates=["Date"])


# plot ally daily closing prices

time_start = 2010
time_end = 2021
ally_data.loc[f"{time_start}":f"{time_end}", "Close"].plot(figsize=(16, 4),
color="blue", legend=True)
ally_data.loc[f"{time_end+1}":, "Close"].plot(figsize=(16, 4), color="green",
legend=True)
plt.legend([f"Training set (Before {time_end+1})", f"Testing set ({time_end+1}
and after)"])
plt.title("Daily Ally Stock Closing Prices")
plt.ylabel("Stock Price")
plt.show()


#rescaling data
ally_data["Close_sc"]=(ally_data["Close"]-
min(ally_data["Close"]))/(max(ally_data["Close"].values)-min(ally_data["Close"]))
train_set=ally_data.loc[f"{time_start}":f"{time_end}", "Close_sc"].values
test_set=ally_data.loc[f"{time_end+1}":, "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)

print(train_x)

#################################################################
#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=ally_data.loc[:,"Close_sc"][len(ally_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(ally_data["Close"].values)-
min(ally_data["Close"]))+min(ally_data["Close"])
test_y=test_y*(max(ally_data["Close"].values)-
min(ally_data["Close"]))+min(ally_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()

######################################################################
#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(ally_data["Close"].values)-
min(ally_data["Close"]))+min(ally_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()
```

Problem 2. Refer to the data from Problem 1. In Excel, create the variable "Shock", an indicator of whether the stock volume changed (increased or decreased) by more than 15% in one day. Train recurrent neural networks for binary classification, using the LSTM and GRU methods, and compute prediction accuracies. Use R and Python.

R Ouput

LTSM Model

```
0.70 0.8348348
```

GRU Model

```
0.65 0.8348348
```

R Code

```
library(anytime)
ally_data <- read.csv("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA SETS/ALLY.csv")

ally_data$Date <- anydate(ally_data$Date)

#splitting data into testing and training sets
```

```
ally_data$Year<- as.numeric(format(as.Date(ally_data$Date, format="%Y-%m-
%d"),"%Y"))
train.data<- ally_data[which(ally_data$Year<2022),1:2]
test.data<- ally_data[which(ally_data$Year>=2022),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,]<- ally_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,]<- ally_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(keras)
#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])),])
```

```
#####################################################################
#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_lstm(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)
10
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])),])
```

Python Output

```
accuracy  cut-off
0   0.831832     0.01
1   0.831832     0.02
2   0.831832     0.03
3   0.831832     0.04
4   0.831832     0.05
..       ...      ...
94  0.831832     0.95
95  0.831832     0.96
96  0.831832     0.97
97  0.831832     0.98
98  0.831832     0.99

[99 rows x 2 columns]
...
97  0.831832     0.98
98  0.831832     0.99
```

Python Code

```
# Problem 2

# binary RNN (recurrent nueral network)

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

ally_data=pandas.read_csv(r"C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/ALLY_shocks.csv",
                          index_col="Date", parse_dates=["Date"])


#splitting into training and testing sets
time_start = 2010
time_end = 2021
def train_test_split(time_start, time_end):
    train=ally_data.loc[f"{time_start}":f"{time_end}", "Shock"].values
    test=ally_data.loc[f"{time_end+1}":, "Shock"].values
    return train, test

train_set, test_set = train_test_split(time_start, time_end)

#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 = ally_data.loc[:,"Shock"][len(ally_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)

#####################################################################
#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_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)
```

Problem 3. Consider the data in the file "weather_description.csv" downloaded from kaggle.com. The file contains hourly weather conditions for several cities between 2012 and 2017. Select a city other than Los Angeles and Detroit, bin the weather conditions into 4-5 categories, whatever applies (e.g., "sunny", "cloudy", "foggy", "rainy", and "snowy"), and run the recurrent neural networks, applying the LSTM and GRU methods. Train the models for binary classifications, and choose the predicted category that corresponds to the largest predicted probability. Calculate the prediction accuracies for both methods. Use R and Python.

R Output

LTSM Model
0.5203

GRU Model
0.5205

R Code

```
# Problem 3
library(keras)
library(dplyr)

weather.data <- read.csv("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/weather_description.csv")

View(weather.data)

chicago <- weather.data$Chicago

chicago<- ifelse(chicago =="sky is clear", "clear",
```

```
        ifelse(chicago %in% c("broken clouds", "few clouds",
                    "overcast clouds","scattered clouds", "smoke"), "cloudy",
                    ifelse(chicago %in% c("heavy shower snow", "heavy snow",
                    "light shower snow", "light snow", "shower snow",
                    "snow"), "snow", "rain")))
weather.data$clear<- ifelse(chicago=="clear",1,0)
weather.data$cloudy<- ifelse(chicago=="cloudy",1,0)
weather.data$snow<- ifelse(chicago=="snow",1,0)
weather.data$rain<- ifelse(chicago=="rain",1,0)
weather.data$year<- format(as.Date(weather.data$datetime, 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<- 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)]

 #defining model architecture
 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=1)
  #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

accuracy<- function() {

test.y<- bind_cols(test.clear, test.cloudy, test.snow, test.rain)
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.clear, pred.prob.cloudy, pred.prob.snow,
pred.prob.cloudy)
colnames(pred.prob)<- 1:4
pred.class<- as.numeric(apply(pred.prob, 1, function(x)
colnames(pred.prob)[which.max(x)]))
15
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.rain<- (rnn.model('lstm', 'rain'))
test.rain<- list.rain[1]
pred.prob.rain<- list.rain[2]
```

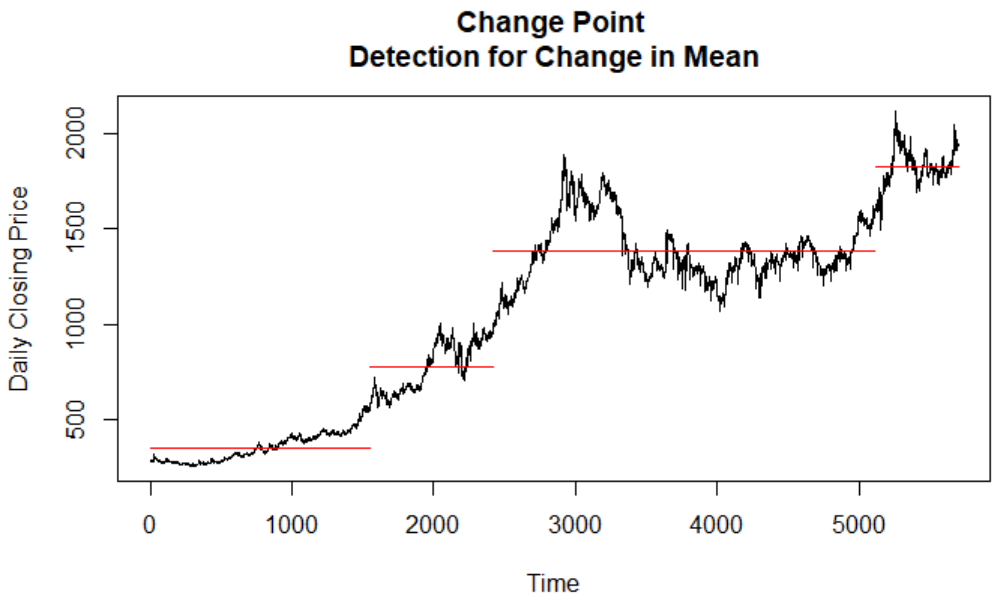
```
print(accuracy())


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

Python Output

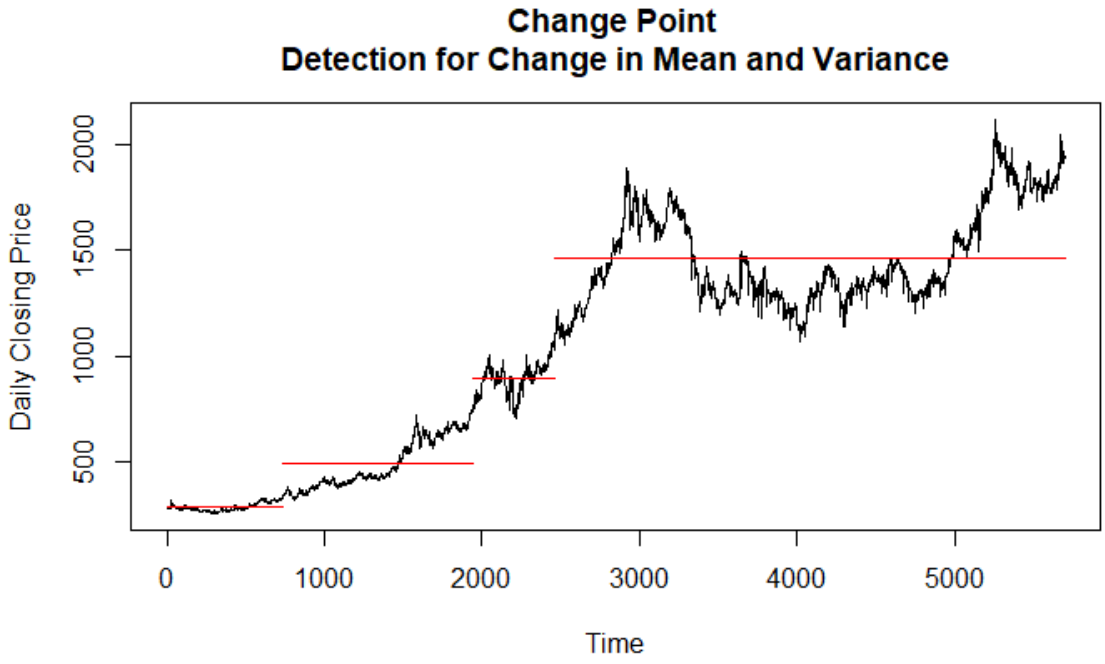
Problem 4. Consider the data in the file "commodity 2000-2022.csv" downloaded from kaggle.com. Choose any commodity except crude (Brent) oil and wheat (the remaining commodities are gold, natural gas, nickel, and palladium). Conduct the change-point detection analysis, assuming: (i) a change in mean, (ii) a change in variance, and (iii) a change in mean and variance. Display the results graphically. Use R.

R Output

```
Change Point Locations: 2418 1550 5114
```



Changepoint Locations : 731 1942 2460



R Code

```
comm.data <- read.csv("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/commodity 2000-2022.csv")

gold.data <- subset(comm.data, Symbol == "Gold")

library(changepoint)
#detection of change-points for change in mean
ansmean=cpt.mean(gold.data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(ansmean,cpt.col="red",ylab="Daily Closing Price", main="Change Point
Detection for Change in Mean")
print(ansmean)


#detection of change-points for change in mean and variance
ansmeanvar=cpt.meanvar(gold.data$Close, penalty="AIC", method="BinSeg", Q=3)
plot(ansmeanvar,cpt.col="red",ylab="Daily Closing Price", main="Change Point
Detection for Change in Mean and Variance")
print(ansmeanvar)
```

Problem 5. Using the data set from Problem 4, conduct the anomaly detection analysis. Visualize the results. Use R.

R Output

UNABLE TO PERFOM IN R

Problem 6. Choose a good book from the Project Gutenberg digital library. For the 25 most used words, display: (i) word count, (ii) bar graph, and (iii) word cloud. Use R.

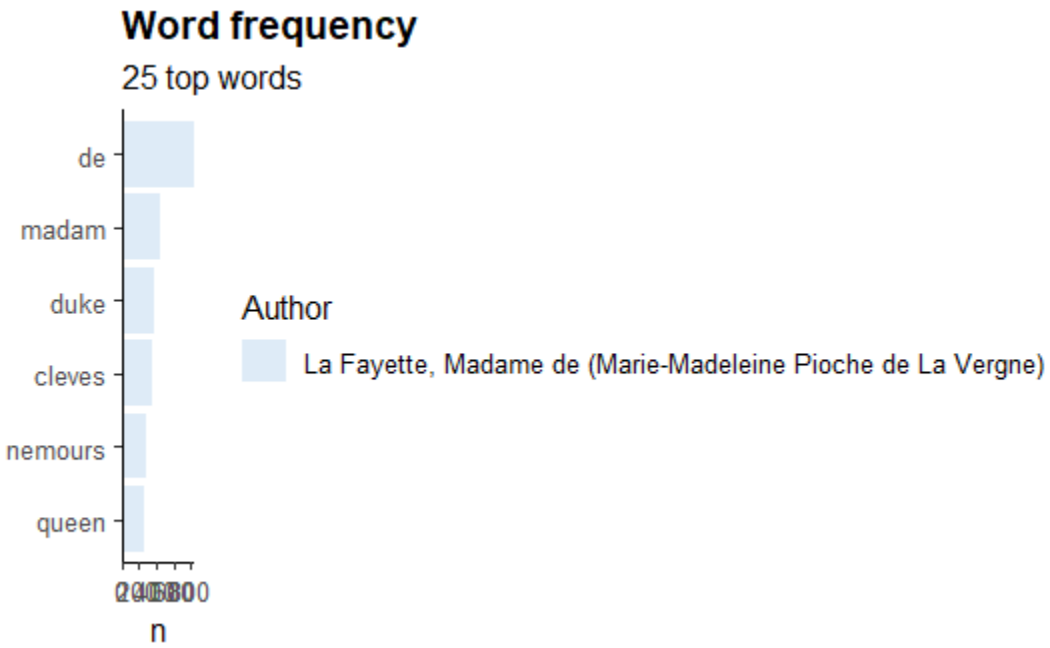
R Output

i. **Word Count**

	word	n
	<chr>	<int>
1	de	830
2	madam	457
3	duke	382
4	cleves	359
5	nemours	275
6	queen	252
7	love	180

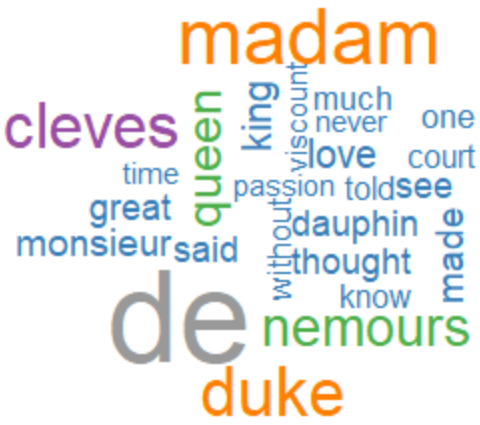
```
 8 king       177
 9 monsieur   170
10 made       156
11 said       147
12 great      146
13 see        145
14 thought    145
15 dauphin    141
16 one        131
17 much       126
18 know       124
19 court      122
20 without    122
21 told       115
22 passion    111
23 never      104
24 time       104
25 viscount   102
```

ii. **Bar Graph**



---

iii. **Word Clouds**



R Code

```
# Problem 6

# i. word count

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


book<- gutenberg_download(467, meta_fields="author")

#puts text into tibble format
book<- as_tibble(book) %>%
 mutate(document = row_number()) %>%
 select(-gutenberg_id)
#creates tokens (words)
#tokenization is the process of splitting text into tokens
tidy_book <- book %>%
```

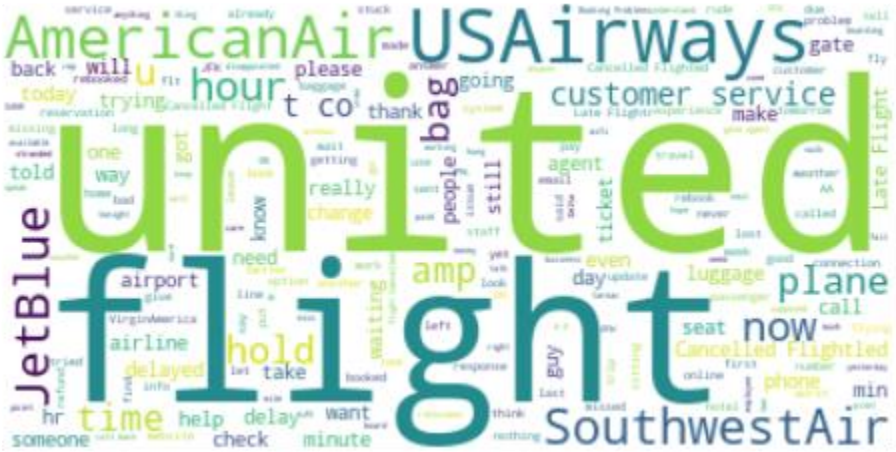
```
 unnest_tokens(word, text) %>%
 group_by(word) %>%
 filter(n() > 10) %>%
 ungroup()


#identifying and removing stopwords (prepositions, articles)
stopword <- as_tibble(stopwords::stopwords("en"))
stopword <- rename(stopword, word=value)
tb <- anti_join(tidy_book, stopword, by='word')

#calculating frequency for the top 25 words
word_count <- tb %>%
 count(word, sort = TRUE)
print(word_count, n=25)


# ii. bar graph

#plotting bar graph for 25 top words
tb %>%
 count(author, word, sort = TRUE) %>%
 filter(n >=232) %>%
 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()


# iii. Word Cloud

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

Problem 7. The file "USAirlinesTweets.csv" downloaded from Kaggle.com (https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment) contains Twitter data scraped from February 2015 and each tweet was classified as positive, negative, or neutral, followed by categorizing negative reasons (for example, "late flight", "lost luggage", or "rude service").

a) **Construct word clouds for positive and negative tweets. What airline gets the most positive tweets and what airline gets the most negative tweets?**

Positive – Jet Blue

Negative – United Airlines

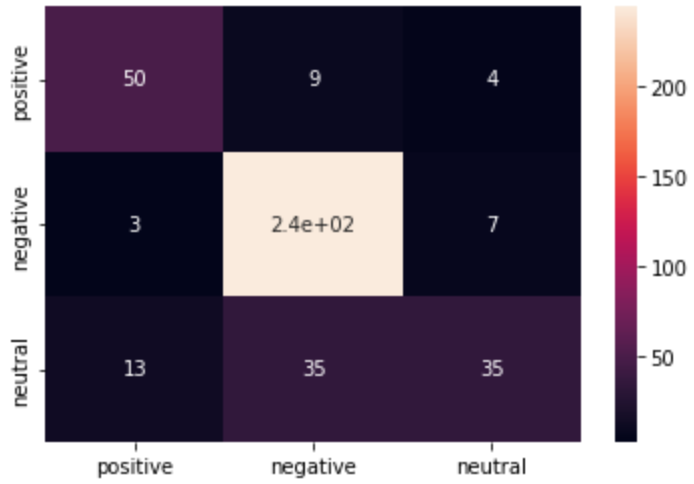



```
import pandas
import numpy
import seaborn
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud


data=pandas.read_csv(r"C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/USAirlinesTweets.csv",encoding='ISO-8859-1')
data.drop_duplicates(subset=['tweet'],keep='first',inplace=True)
text = " ".join([x for x in data.tweet])

#plotting wordclouds for positive news
text = "".join([x for x in data.tweet[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()


#plotting wordclouds for negative news
text = "".join([x for x in data.tweet[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()
```

b) Using all data entries, conduct the sentiment analysis by training the BERT model in Python and compute the accuracy of prediction of tweet sentiment (positive/negative/neutral). Test the model with your own mock tweets.

```
0.7357336065553605

[[ 50   9   4]
 [  3 244   7]
 [ 13  35  35]]
```




```
precision    recall  f1-score   support\n\n    positive       0.76      0.79
0.78        63\n    negative       0.85      0.96      0.90       254\n
neutral       0.76      0.42      0.54        83\n\n    accuracy
0.82       400\n   macro avg       0.79      0.73      0.74       400\nweighted
avg       0.82      0.82      0.81       400\n'

0.8225 - accuracy
```

All three statements were predicted as negative although the last statement was positive and the prediction accuracy is 82.25%.

```
# part a)
#!pip install wordcloud
import pandas
import numpy
import seaborn
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from wordcloud import WordCloud


data=pandas.read_csv(r"C:/Users/saedw/OneDrive/Desktop/STAT 574 Data
Mining/hw4STAT574S23/DATA SETS/USAirlinesTweets.csv",encoding='ISO-8859-1')
data.drop_duplicates(subset=['tweet'],keep='first',inplace=True)
text = " ".join([x for x in data.tweet])

#plotting wordclouds for positive news
```

```
text = "".join([x for x in data.tweet[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()


#plotting wordclouds for negative news
text = "".join([x for x in data.tweet[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()

# neutral
text = "".join([x for x in data.tweet[data.sentiment=='neutral']])

wordcloud = WordCloud(background_color='white').generate(text)

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

# part b)

#plotting bar graph for sentiments
# seaborn.countplot(data.sentiment)

#displaying frequency by sentiment
data['sentiment'].value_counts()

#training model
numpy.random.seed(5677934)
train, test = train_test_split(data,test_size = 0.2)

#!pip install simpletransformers
#!pip install torch
```

```
from simpletransformers.classification import ClassificationModel

# Create a TransformerModel
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 = pandas.DataFrame({
    'text': train['tweet'][:1500].replace(r'\n', ' ', regex=True),
    'label': train['label'][:1500]
})

eval_df = pandas.DataFrame({
    'text': test['tweet'][-400:].replace(r'\n', ' ', regex=True),
    'label': test['label'][-400:]
})

model.train_model(train_df)

# Problem 7 part b continued

#computing predicted sentiments for testing set
result, model_outputs, wrong_predictions = model.eval_model(eval_df)

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

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

#displaying confusion matrix (positive/negative/neutral)
import sklearn
confmatrix = sklearn.metrics.confusion_matrix(true, predicted)
print(confmatrix)
```

```
#displaying heatmap for confusion matrix
df_cm = pandas.DataFrame(confmatrix, ['positive','negative','neutral'],
['positive','negative','neutral'])

seaborn.heatmap(df_cm, annot=True)
plt.show()
#displaying performance metrics
sklearn.metrics.classification_report(true,predicted,target_names=['positive','ne
gative','neutral'])
#computing predicted accuracy
sklearn.metrics.accuracy_score(true,predicted)
#using the trained model to classify user-defined sentences
def classify(statement):
        result = model.predict([statement])
        pred_class = numpy.where(result[1][0] == numpy.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('My terminal was changed.')
classify('The airline misplaced my luggage.')
classify('The economy seats were very comfortable.')
```

Problem 8. The folder "WildAnimalsImages" contains images of caracals, cheetahs, lions, and tigers. The data are already split into training set (40 images of each type of animal) and testing set (3 images of each type of animal). Train a convolutional neural network on the training set, and use the trained model to classify the images in the testing set. Compute the prediction accuracy. Use R

R Output

Pred.class

```
[1] 2 0 0 1 2 2 2 2 2 3 1 1
Test.y
[1] 0 0 0 1 1 1 2 2 2 3 3 3

"accuracy= 0.5833"
```

R Code

```
# Problem 8


library(keras)
library(EBImage)


#preparing training set

# CARACALS ####################

setwd("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/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)
}

# CHEETAS ####################

setwd("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/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/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/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/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/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])
#permuting 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 set

# CARCALS ###############

setwd("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/WildAnimalsImages/test/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)
}

# CHEETAS #################

setwd("C:/Users/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/WildAnimalsImages/test/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/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/WildAnimalsImages/test/LIONS")
img.lions<- sample(dir());
test.lions<- list(NULL);
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/saedw/OneDrive/Desktop/STAT 574 Data Mining/hw4STAT574S23/DATA
SETS/WildAnimalsImages/test/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)


#building the 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)))
```