UNIVERSITY OF MALAYA

EXAMINATION FOR THE DEGREE OF MASTER OF DATA SCIENCE

ACADEMIC SESSION 2019/2020 : SEMESTER II

WQD7005 : Data Mining

Name of Candidate: Lew Teck Wei

Matric Number: 17021682

June 2020

INSTRUCTIONS TO CANDIDATES :

Answer **ALL** questions (50 marks).

(This question paper consists of 5 questions on 3 printed pages)

Mini-assignment (50 marks)

Instructions: Work individually, submission via Spectrum.

1. You are required to make a user-agent that will crawl the WWW (your familiar domain) to produce dataset of a particular website.
   - the web site can be as simple as a list of webpages and what other pages they link to
   - the output does not need to be in XHTML (or HTML) form
     a multi-stage approach (e.g. produce the xhtml or html in csv format)

   Answer:
   Scrape the following date from WWW using the library and code as below:

   ➔ refer to video presentation https://youtu.be/lqMMceTyH5A for better understanding or refer to the presentation slide at github https://github.com/L-venLewTeckWei/Final-Exam-Q1-and-Q2-Lew-Teck-Wei-17021682

MBB

	date	close
0	2020-03-09	8.239999771118164
1	2020-03-06	8.5
2	2020-03-05	8.529999732971191
3	2020-03-04	8.470000267028809
4	2020-03-03	8.40999984741211
...	...	...
1237	2015-03-16	9.09000015258789
1238	2015-03-13	9.100000381469727
1239	2015-03-12	9.119999885559082
1240	2015-03-11	9.09000015258789
1241	2015-03-10	9.229999542236328

[1242 rows x 2 columns]

CIMB

	date	close
0	2020-03-09	4.199999809265137
1	2020-03-06	4.440000057220459
2	2020-03-05	4.5
3	2020-03-04	4.539999961853027
4	2020-03-03	4.5
...	...	...
1238	2015-03-16	5.909999847412109
1239	2015-03-13	5.920000076293945
1240	2015-03-12	5.849999904632568
1241	2015-03-11	5.800000190734863
1242	2015-03-10	5.949999809265137

[1243 rows x 2 columns]

RHB

	date	close
0	2020-03-09	5.46999979019165
1	2020-03-06	5.699999809265137
2	2020-03-05	5.710000038146973
3	2020-03-04	5.710000038146973
4	2020-03-03	5.639999866485596
...	...	...
1238	2015-03-16	7.265110015869141
1239	2015-03-13	7.199659824371338
1240	2015-03-12	7.237060070037842
1241	2015-03-11	7.349259853363037
1242	2015-03-10	7.386660099029541

[1243 rows x 2 columns]

KLCI

	date	close
0	2020-03-09	1424.1600341796875
1	2020-03-06	1483.0999755859375
2	2020-03-05	1491.030029296875
3	2020-03-04	1489.949951171875
4	2020-03-03	1478.6400146484375
...	...	...
1221	2015-03-16	1780.5400390625
1222	2015-03-13	1781.75
1223	2015-03-12	1786.8699951171875
1224	2015-03-11	1778.1600341796875
1225	2015-03-10	1789.72998046875

[1226 rows x 2 columns]

DJI

	date	close
0	2020-03-09	23851.01953125
1	2020-03-06	25864.779296875
2	2020-03-05	26121.279296875
3	2020-03-04	27090.859375
4	2020-03-03	25917.41015625
...	...	...
1254	2015-03-16	17977.419921875
1255	2015-03-13	17749.310546875
1256	2015-03-12	17895.220703125
1257	2015-03-11	17635.390625
1258	2015-03-10	17662.939453125

[1259 rows x 2 columns]

SNP

	date	close
0	2020-03-09	2746.56005859375
1	2020-03-06	2972.3701171875
2	2020-03-05	3023.93994140625
3	2020-03-04	3130.1201171875
4	2020-03-03	3003.3701171875
...	...	...
1254	2015-03-16	2081.18994140625
1255	2015-03-13	2053.39990234375
1256	2015-03-12	2065.949951171875
1257	2015-03-11	2040.239990234375
1258	2015-03-10	2044.1600341796875

[1259 rows x 2 columns]

Step 1 : Data Scraping – crawl the trading date and closing price of the following Index and Stocks

- Maybank –MBB
- RHB
- KLCI – Kuala Lumpur Composite Index
- DJI – Dow Jone Index
- SNP - S&P 500 Index

```
import requests
import bs4
import json
import datetime
import pandas as pd
page = requests.get("https://finance.yahoo.com/quote/MBBM.KL/history?period1=1425945600&period2=1583798400&interval=1d&filter

from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())

listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        MBB = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                MBB = MBB.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(MBB)
```

```
import requests
import bs4
import json
import datetime
import pandas as pd
page = requests.get("https://finance.yahoo.com/quote/COMM.KL/history?period1=1425945600&period2=1583798400&interval=1d&filter

from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())


listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        CIMB = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                CIMB = CIMB.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(CIMB)

CIMB.to_csv(r"C:/Users/L-ven Lew/Desktop/UM/Semester 4 UM/WQD 7005 Data Mining/Final Exam/Final Exam Question 1/CIMB.csv",ind
```

```
import requests
import bs4
import json
import datetime
import pandas as pd
page = requests.get("https://finance.yahoo.com/quote/RHBC.KL/history?period1=1425945600&period2=1583798400&interval=1d&filter

from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())


listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        RHB = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                RHB = RHB.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(RHB)

RHB.to_csv(r"C:/Users/L-ven Lew/Desktop/UM/Semester 4 UM/WQD 7005 Data Mining/Final Exam/Final Exam Question 1/RHB.csv",index
```

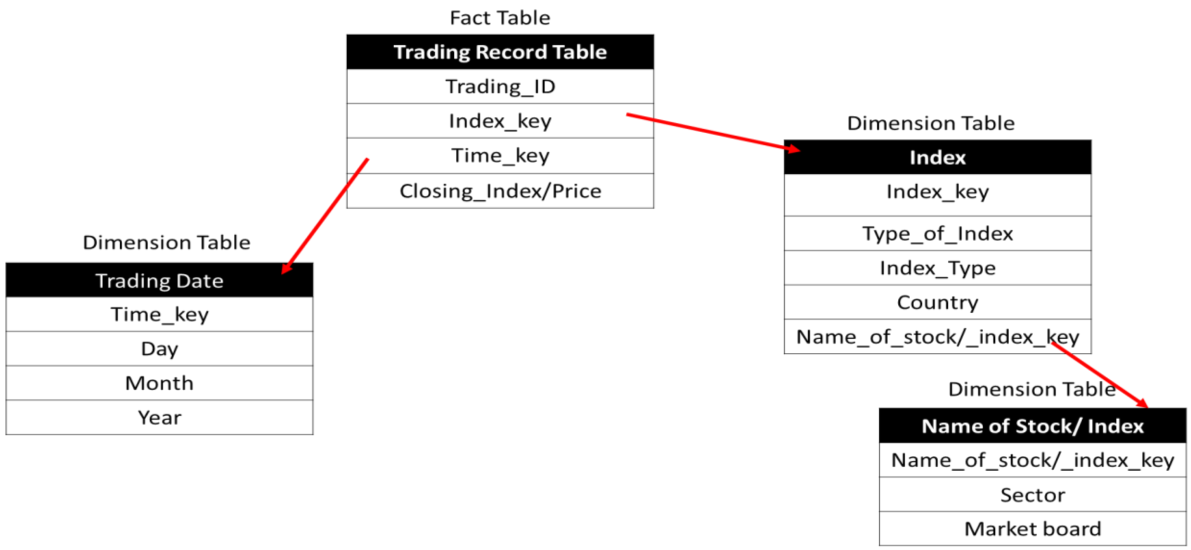
```
page = requests.get("https://finance.yahoo.com/quote/%5EKLSE%3FP%3D%5EKLSE/history?period1=1425945600&period2=1583798400&inte
page.status_code

from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())


listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        KLCI = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                KLCI = KLCI.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(KLCI)
KLCI.to_csv(r'C:/Users/L-ven Lew/Desktop/UM/Semester 4 UM/WQD 7005 Data Mining/Final Exam/Final Exam Question 1/KLCI3.csv',in
KLCI
```

```
import requests
import bs4
import json
import datetime
import pandas as pd
page = requests.get("https://finance.yahoo.com/quote/%5EDJI/history?period1=1425945600&period2=1583798400&interval=1d&filter=

from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())

listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        DJI = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                DJI = DJI.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(DJI)

DJI.to_csv(r"C:/Users/L-ven Lew/Desktop/UM/Semester 4 UM/WQD 7005 Data Mining/Final Exam/Final Exam Question 1/DJI.csv",index
```

```
import requests
import bs4
import json
import datetime
import pandas as pd
page = requests.get("https://finance.yahoo.com/quote/%5EGSPC/history?period1=1425945600&period2=1583798400&interval=1d&filter
from bs4 import BeautifulSoup
soup = BeautifulSoup(page.content, 'html.parser')
#print(soup.prettify())

listScript = soup.find_all("script")
for script in listScript:
    txtScript = script.string
    if type(txtScript) is bs4.element.NavigableString and txtScript.find('HistoricalPriceStore') != -1:
        SNP = pd.DataFrame(columns=['date','close'])
        txtInfo = txtScript[txtScript.find('HistoricalPriceStore'):txtScript.find('}],"isPending":false,"firstTradeDate":')]
        txtInfo = "{\"" + txtInfo + "}]}}"
        objJson = json.loads(txtInfo)
        for price in objJson['HistoricalPriceStore']['prices']:
            txtDate = ""
            txtClose = ""
            for attr,val in price.items():
                if attr == "date" and val != None:
                    txtDate = datetime.datetime.fromtimestamp(val).strftime('%Y-%m-%d')
                if attr == "close" and val != None:
                    txtClose = str(val)
            if txtDate != "" and txtClose != "":
                SNP = SNP.append({"date":txtDate, "close":txtClose}, ignore_index=True)
        print(SNP)

SNP.to_csv(r"C:/Users/L-ven Lew/Desktop/UM/Semester 4 UM/WQD 7005 Data Mining/Final Exam/Final Exam Question 1/SNP.csv",index
SNP
```

(10 marks)

2. Draw snowflake schema diagram for the above dataset. Justify your attributes to be selected in the respective dimensions.



Index and Name of Stock/ Index dimension

Index dimension table contains Type_of_Index, Index_Type, Country and Name_of_stock/_index_key, the reason that I selected the attributes is because if I read the attributes (dataset) from this schema, only these attributes will be read, I will have a glance of what type_of_index,Index_type, Country and Name_of_stock/_index being stored in this disk. Sector and Market Board attribute are not important attribute to be read so I store it in a separate dimension namely Name of Stock/ Index dimension with the given key to retrieve the data only when it needed. In addition, it requires low disk storage and it will be faster when it is queried.

Trading Date Dimension

Trading date dimension store the day, month and year data. The reason is to ensure data storage and integrity issue when the format of the date is changed, one the date format is changed in this dimension, trading date will be consistent / standardized across all the reads.

Submissions:

Sample of SQL queries

```
# SELECT Index.Trading_ID, Type_of_Index, Country, Name of stock/ index
#    from Index_key.Index
#    join Name of stock/ index
#    on Name_of_stock/_index_key.Trading_ID = Index.Trading_ID limit 3;

DF[['Trading_Date', 'Type_of_Index', 'Country','Name of Stock/ Index','Sector','Market board']].head(3)
```

	Trading_Date	Type_of_Index	Country	Name of Stock/ Index	Sector	Market board
0	2020-03-09	Malaysia Stock	MY	MBB	Financial	Main Board
1	2020-03-06	Malaysia Stock	MY	MBB	Financial	Main Board
2	2020-03-05	Malaysia Stock	MY	MBB	Financial	Main Board

```
# SELECT index.Trading_ID, Type_of_Index, Country
#    from Index_key.index
#    join Name of Stock/ Index where Market board = "Main Board"
#    on Name_of_stock/_index_key.Trading_ID = index.Trading_ID limit 3;

DF[DF['Market board']== 'Main Board']
```

	Trading_Date	Closing_Index/Price	Type_of_Index	Sector	Market board	Month_Year	Name of Stock/ Index	Monthly_Average	Country	Index_Type
0	2020-03-09	8.24000	Malaysia Stock	Financial	Main Board	2020-03	MBB	8.423333	MY	Stock
1	2020-03-06	8.50000	Malaysia Stock	Financial	Main Board	2020-03	MBB	8.423333	MY	Stock
2	2020-03-05	8.53000	Malaysia Stock	Financial	Main Board	2020-03	MBB	8.423333	MY	Stock

3. You are required to write code to create a decision tree (DT) model using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:

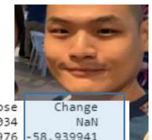
You may refer to video presentation https://youtu.be/VlT2MEblMK8 (video 1) and https://youtu.be/ar-cHPhOyow (video 2) for better understanding or refer to the presentation slide at github

- Importing required libraries

```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn import tree
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics
```

- Loading Data

i. Perform the code below to create a feature for the Change of price/ index for the stocks and index as follows:



a. Maybank –MBB
b. RHB
c. KLCI – Kuala Lumpur Composite Index
d. DJI – Dow Jone Index
e. SNP - S&P 500 Index

```python
MBB1 = MBB.iloc[:, 0:2]
CIMB1= CIMB.iloc[:, 0:2]
RHB1 = RHB.iloc[:, 0:2]
KLCI1 = KLCI.iloc[:, 0:2]
DJI1 = DJI.iloc[:, 0:2]
SNP1 = SNP.iloc[:, 0:2]

for i in range(1, len(MBB1)):
    MBB1.loc[i, 'Change'] = MBB1.loc[i-1, 'close'] - MBB1.loc[i, 'close']
print(MBB1)

for i in range(1, len(CIMB1)):
    CIMB1.loc[i, 'Change'] = CIMB1.loc[i-1, 'close'] - CIMB1.loc[i, 'close']
print(CIMB1)

for i in range(1, len(RHB1)):
    RHB1.loc[i, 'Change'] = RHB1.loc[i-1, 'close'] - RHB1.loc[i, 'close']
print(CIMB1)

for i in range(1, len(KLCI1)):
    KLCI1.loc[i, 'Change'] = KLCI1.loc[i-1, 'close'] - KLCI1.loc[i, 'close']
print(KLCI1)

for i in range(1, len(DJI1)):
    DJI1.loc[i, 'Change'] = DJI1.loc[i-1, 'close'] - DJI1.loc[i, 'close']
print(DJI1)

for i in range(1, len(SNP1)):
    SNP1.loc[i, 'Change'] = SNP1.loc[i-1, 'close'] - SNP1.loc[i, 'close']
print(SNP1)
```

```
MBB        date  close    Change
0    2020-03-09   8.24       NaN
1    2020-03-06   8.50 -0.260000
2    2020-03-05   8.53 -0.030000
3    2020-03-04   8.47  0.059999
4    2020-03-03   8.41  0.060000
...         ...    ...       ...
1237 2015-03-16   9.09  0.070000
1238 2015-03-13   9.10 -0.010000
1239 2015-03-12   9.12 -0.020000
1240 2015-03-11   9.09  0.030000
1241 2015-03-10   9.23 -0.139999

[1242 rows x 3 columns]
CIMB       date  close Change
0    2020-03-09   4.20    NaN
1    2020-03-06   4.44  -0.24
2    2020-03-05   4.50  -0.06
3    2020-03-04   4.54  -0.04
4    2020-03-03   4.50   0.04
...         ...    ...    ...
1238 2015-03-16   5.91   0.05
1239 2015-03-13   5.92  -0.01
1240 2015-03-12   5.85   0.07
1241 2015-03-11   5.80   0.05
1242 2015-03-10   5.95  -0.15

[1243 rows x 3 columns]
RHB        date  close Change
0    2020-03-09   4.20    NaN
1    2020-03-06   4.44  -0.24
2    2020-03-05   4.50  -0.06
3    2020-03-04   4.54  -0.04
4    2020-03-03   4.50   0.04
...         ...    ...    ...
1238 2015-03-16   5.91   0.05
1239 2015-03-13   5.92  -0.01
1240 2015-03-12   5.85   0.07
1241 2015-03-11   5.80   0.05
1242 2015-03-10   5.95  -0.15

[1243 rows x 3 columns]
```

```
KLCI       date        close     Change
0    2020-03-09  1424.160034        NaN
1    2020-03-06  1483.099976 -58.939941
2    2020-03-05  1491.030029  -7.930054
3    2020-03-04  1489.949951   1.080078
4    2020-03-03  1478.640015  11.309937
...         ...          ...        ...
1221 2015-03-16  1780.540039   7.329956
1222 2015-03-13  1781.750000  -1.209961
1223 2015-03-12  1786.869995  -5.119995
1224 2015-03-11  1778.160034   8.709961
1225 2015-03-10  1789.729980 -11.569946

[1226 rows x 3 columns]
DJI        date         close       Change
0    2020-03-09  23851.019531          NaN
1    2020-03-06  25864.779297 -2013.759766
2    2020-03-05  26121.279297  -256.500000
3    2020-03-04  27090.859375  -969.580078
4    2020-03-03  25917.410156  1173.449219
...         ...           ...          ...
1254 2015-03-16  17977.419922  -128.339844
1255 2015-03-13  17749.310547   228.109375
1256 2015-03-12  17895.220703  -145.910156
1257 2015-03-11  17635.390625   259.830078
1258 2015-03-10  17662.939453   -27.548828

[1259 rows x 3 columns]
SNP        date        close      Change
0    2020-03-09  2746.560059         NaN
1    2020-03-06  2972.370117 -225.810059
2    2020-03-05  3023.939941  -51.569824
3    2020-03-04  3130.120117 -106.180176
4    2020-03-03  3003.370117  126.750000
...         ...          ...         ...
1254 2015-03-16  2081.189941   -6.909912
1255 2015-03-13  2053.399902   27.790039
1256 2015-03-12  2065.949951  -12.550049
1257 2015-03-11  2040.239990   25.709961
1258 2015-03-10  2044.160034   -3.920044

[1259 rows x 3 columns]
```

Step 2 : Loading the dataset (Continued from previous slide)

ii. Perform the following code to create a " movement" column. The value of the features as follows:

a. No Change
b. Rise
c. Drop

```python
def f(row):
    if row['Change'] > 0:
        val = 'Rise'
    elif row['Change'] < 0:
        val = 'Drop'
    else:
        val = 'No Change'
    return val

MBB1['MBB_Movement'] = MBB1.apply(f, axis=1)
CIMB1['CIMB1_Movement'] = CIMB1.apply(f, axis=1)
RHB1['RHB1_Movement'] = RHB1.apply(f, axis=1)
KLCI1['KLCI1_Movement'] = KLCI1.apply(f, axis=1)
DJI1['DJI1_Movement'] = DJI1.apply(f, axis=1)
SNP1['SNP1_Movement'] = SNP1.apply(f, axis=1)
```

```
MBB         date  close    Change MBB_Movement
0     2020-03-09   8.24       NaN    No Change
1     2020-03-06   8.50 -0.260000         Drop
2     2020-03-05   8.53 -0.030000         Drop
3     2020-03-04   8.47  0.059999         Rise
4     2020-03-03   8.41  0.060000         Rise
...          ...    ...       ...          ...
1237  2015-03-16   9.09  0.070000         Rise
1238  2015-03-13   9.10 -0.010000         Drop
1239  2015-03-12   9.12 -0.020000         Drop
1240  2015-03-11   9.09  0.030000         Rise
1241  2015-03-10   9.23 -0.139999         Drop

[1242 rows x 4 columns]
CIMB        date  close  Change CIMB1_Movement
0     2020-03-09   4.20     NaN      No Change
1     2020-03-06   4.44   -0.24           Drop
2     2020-03-05   4.50   -0.06           Drop
3     2020-03-04   4.54   -0.04           Drop
4     2020-03-03   4.50    0.04           Rise
...          ...    ...     ...            ...
1238  2015-03-16   5.91    0.05           Rise
1239  2015-03-13   5.92   -0.01           Drop
1240  2015-03-12   5.85    0.07           Rise
1241  2015-03-11   5.80    0.05           Rise
1242  2015-03-10   5.95   -0.15           Drop

[1243 rows x 4 columns]
RHB         date    close   Change RHB1_Movement
0     2020-03-09  5.47000      NaN     No Change
1     2020-03-06  5.70000 -0.23000          Drop
2     2020-03-05  5.71000 -0.01000          Drop
3     2020-03-04  5.71000  0.00000     No Change
4     2020-03-03  5.64000  0.07000          Rise
...          ...      ...      ...           ...
1238  2015-03-16  7.26511  0.03740          Rise
1239  2015-03-13  7.19966  0.06545          Rise
1240  2015-03-12  7.23706 -0.03740          Drop
1241  2015-03-11  7.34926 -0.11220          Drop
1242  2015-03-10  7.38666 -0.03740          Drop

[1243 rows x 4 columns]
```

```
KLCI        date        close     Change KLCI1_Movement
0     2020-03-09  1424.160034        NaN      No Change
1     2020-03-06  1483.099976 -58.939941           Drop
2     2020-03-05  1491.030029  -7.930054           Drop
3     2020-03-04  1489.949951   1.080078           Rise
4     2020-03-03  1478.640015  11.309937           Rise
...          ...          ...        ...            ...
1221  2015-03-16  1780.540039   7.329956           Rise
1222  2015-03-13  1781.750000  -1.209961           Drop
1223  2015-03-12  1786.869995  -5.119995           Drop
1224  2015-03-11  1778.160034   8.709961           Rise
1225  2015-03-10  1789.729980 -11.569946           Drop

[1226 rows x 4 columns]
DJI         date         close       Change DJI1_Movement
0     2020-03-09  23851.019531          NaN     No Change
1     2020-03-06  25864.779297 -2013.759766          Drop
2     2020-03-05  26121.279297  -256.500000          Drop
3     2020-03-04  27090.859375  -969.580078          Drop
4     2020-03-03  25917.410156  1173.449219          Rise
...          ...           ...          ...           ...
1254  2015-03-16  17977.419922  -128.339844          Drop
1255  2015-03-13  17749.310547   228.109375          Rise
1256  2015-03-12  17895.220703  -145.910156          Drop
1257  2015-03-11  17635.390625   259.830078          Rise
1258  2015-03-10  17662.939453   -27.548828          Drop

[1259 rows x 4 columns]
SNP         date        close      Change SNP1_Movement
0     2020-03-09  2746.560059         NaN     No Change
1     2020-03-06  2972.370117 -225.810059          Drop
2     2020-03-05  3023.939941  -51.569824          Drop
3     2020-03-04  3130.120117 -106.180176          Drop
4     2020-03-03  3003.370117  126.750000          Rise
...          ...          ...         ...           ...
1254  2015-03-16  2081.189941   -6.909912          Drop
1255  2015-03-13  2053.399902   27.790039          Rise
1256  2015-03-12  2065.949951  -12.550049          Drop
1257  2015-03-11  2040.239990   25.709961          Rise
1258  2015-03-10  2044.160034   -3.920044          Drop

[1259 rows x 4 columns]
```

Step 2 : Loading the dataset (Continued from previous slide)

iv. Perform the following code to drop the unnecessary features:

```python
MBB2 = MBB1.drop(['close', 'Change'], axis=1)
CIMB2 = CIMB1.drop(['close', 'Change'], axis=1)
RHB2 = RHB1.drop(['close', 'Change'], axis=1)
KLCI2 = KLCI1.drop(['close', 'Change'], axis=1)
DJI2 = DJI1.drop(['close', 'Change'], axis=1)
SNP2 = SNP1.drop(['close', 'Change'], axis=1)
```

```
MBB2        date MBB_Movement
0     2020-03-09    No Change
1     2020-03-06         Drop
2     2020-03-05         Drop
3     2020-03-04         Rise
4     2020-03-03         Rise
...          ...          ...
1237  2015-03-16         Rise
1238  2015-03-13         Drop
1239  2015-03-12         Drop
1240  2015-03-11         Rise
1241  2015-03-10         Drop

[1242 rows x 2 columns]
CIMB2       date CIMB1_Movement
0     2020-03-09      No Change
1     2020-03-06           Drop
2     2020-03-05           Drop
3     2020-03-04           Drop
4     2020-03-03           Rise
...          ...            ...
1238  2015-03-16           Rise
1239  2015-03-13           Drop
1240  2015-03-12           Rise
1241  2015-03-11           Rise
1242  2015-03-10           Drop

[1243 rows x 2 columns]
RHB2        date RHB1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04     No Change
4     2020-03-03          Rise
...          ...           ...
1238  2015-03-16          Rise
1239  2015-03-13          Rise
1240  2015-03-12          Drop
1241  2015-03-11          Drop
1242  2015-03-10          Drop

[1243 rows x 2 columns]
```

```
KLCI2       date KLCI1_Movement
0     2020-03-09      No Change
1     2020-03-06           Drop
2     2020-03-05           Drop
3     2020-03-04           Rise
4     2020-03-03           Rise
...          ...            ...
1221  2015-03-16           Rise
1222  2015-03-13           Drop
1223  2015-03-12           Drop
1224  2015-03-11           Rise
1225  2015-03-10           Drop

[1226 rows x 2 columns]
DJI2        date DJI1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04          Drop
4     2020-03-03          Rise
...          ...           ...
1254  2015-03-16          Drop
1255  2015-03-13          Rise
1256  2015-03-12          Drop
1257  2015-03-11          Rise
1258  2015-03-10          Drop

[1259 rows x 2 columns]
SNP2        date SNP1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04          Drop
4     2020-03-03          Rise
...          ...           ...
1254  2015-03-16          Drop
1255  2015-03-13          Rise
1256  2015-03-12          Drop
1257  2015-03-11          Rise
1258  2015-03-10          Drop

[1259 rows x 2 columns]
```

Step 2 : Loading the dataset (Continued from previous slide)

v. Perform the following code to combine the required data then clean them

```
MergeD = pd.merge(KLCI1, DJI1,
                  how="left", on=["date"])

MergeD1 = pd.merge(MergeD,SNP1,
                   how="left", on=["date"])

MergeD2 = pd.merge(MergeD1,MBB2,
                   how="left", on=["date"])

MergeD3 = pd.merge(MergeD2, RHB2,
                   how="left", on=["date"])

Tree_Decision_Dataset = pd.merge(MergeD3, CIMB2,
                   how="left", on=["date"])
```

```
Tree_Decision_Dataset1 = Tree_Decision_Dataset.dropna(how='any',axis=0)
print(Tree_Decision_Dataset1.isnull().sum())
print(Tree_Decision_Dataset1)
```

```
date              0
close_x           0
Change_x          0
KLCI1_Movement    0
close_y           0
Change_y          0
DJI1_Movement     0
close             0
Change            0
SNP1_Movement     0
MBB_Movement      0
RHB1_Movement     0
CIMB1_Movement    0
dtype: int64
```

```
            date     close_x   Change_x KLCI1_Movement       close_y  \
1     2020-03-06  1483.099976 -58.939941           Drop  25864.779297
2     2020-03-05  1491.030029  -7.930054           Drop  26121.279297
3     2020-03-04  1489.949951   1.080078           Rise  27090.859375
4     2020-03-03  1478.640015  11.309937           Rise  25917.410156
5     2020-03-02  1466.939941  11.700073           Rise  26703.320312
...          ...          ...        ...            ...           ...
1221  2015-03-16  1780.540039   7.329956           Rise  17977.419922
1222  2015-03-13  1781.750000  -1.209961           Drop  17749.310547
1223  2015-03-12  1786.869995  -5.119995           Drop  17895.220703
1224  2015-03-11  1778.160034   8.709961           Rise  17635.390625
1225  2015-03-10  1789.729980 -11.569946           Drop  17662.939453

         Change_y DJI1_Movement        close      Change SNP1_Movement
1    -2013.759766          Drop  2972.370117 -225.810059          Drop
2     -256.500000          Drop  3023.939941  -51.569824          Drop
3     -969.580078          Drop  3130.120117 -106.180176          Drop
4     1173.449219          Rise  3003.370117  126.750000          Rise
5     -785.910156          Drop  3090.229980  -86.859863          Drop
...           ...           ...          ...         ...           ...
1221  -128.339844          Drop  2081.189941   -6.909912          Drop
1222   228.109375          Rise  2053.399902   27.790039          Rise
1223  -145.910156          Drop  2065.949951  -12.550049          Drop
1224   259.830078          Rise  2040.239990   25.709961          Rise
1225   -27.548828          Drop  2044.160034   -3.920044          Drop

     MBB_Movement RHB1_Movement CIMB1_Movement
1            Drop          Drop           Drop
2            Drop          Drop           Drop
3            Rise     No Change           Drop
4            Rise          Rise           Rise
5            Rise          Rise           Drop
...           ...           ...            ...
1221         Rise          Rise           Rise
1222         Drop          Rise           Drop
1223         Drop          Drop           Rise
1224         Rise          Drop           Rise
1225         Drop          Drop           Drop

[1192 rows x 13 columns]
```

- Feature Selection
- Splitting Data
- Building Decision Tree Model
- Evaluating Model

Step 3 to step 6 of Question 3
MayBank Stock price movement vs daily change of KLCI, DJI & SNP

Step 3. Feature

KLCI1_Movement

```
feature_cols = ['Change_x']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.MBB_Movement # Target variable
```

DJI1_Movement

```
feature_cols = ['Change_y']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.MBB_Movement # Target variable
```

SNP1_Movement

```
feature_cols = ['Change']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.MBB_Movement # Target variable
```

Step 4. Split the data

```
# Split dataset into training set and test set
# 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Step 5. Building the decision tree model

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Step 6. Evaluating Model

```
accuracy = metrics.accuracy_score(y_test,y_pred)
accuracy
```

0.5446927374301676 KLCI1_Movement

0.4329608938547486 DJI1_Movement

0.3659217877094972 SNP1_Movement

Step 3 to step 6 of Question 3
CIMB Stock price movement vs daily change of KLCI, DJI & SNP

Step 3. Feature

KLCI1_Movement

```
feature_cols = ['Change_x']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.CIMB1_Movement # Target variable
```

DJI1_Movement

```
feature_cols = ['Change_y']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.CIMB1_Movement # Target variable
```

SNP1_Movement

```
feature_cols = ['Change']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.CIMB1_Movement # Target variable
```

Step 4. Split the data

```
# Split dataset into training set and test set
# 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

Step 5. Building the decision tree model

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Step 6. Evaluating Model

```
accuracy = metrics.accuracy_score(y_test,y_pred)
accuracy
```

0.5195530726256983 KLCI1_Movement

0.4329608938547486 DJI1_Movement

0.40502793296089384 SNP1_Movement

Step 3 to step 6 of Question 3

RHB Stock price movement vs daily change of KLCI, DJI & SNP

Step 3. Feature

KLCI1_Movement

```
feature_cols = ['Change_x']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.RHB1_Movement # Target variable
```

DJI1_Movement

```
feature_cols = ['Change_y']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.RHB1_Movement # Target variable
```

SNP1_Movement

```
feature_cols = ['Change']
X = Tree_Decision_Dataset1[feature_cols] # Features
y = Tree_Decision_Dataset1.RHB1_Movement # Target variable
```

Step 4. Split the data

```
# Split dataset into training set and test set
# 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

Step 5. Building the decision tree model

```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

Step 6. Evaluating Model

```
accuracy = metrics.accuracy_score(y_test,y_pred)
accuracy
```

0.4581005586592179 KLCI1_Movement

0.3575418994413408 DJI1_Movement

0.4022346368715084 SNP1_Movement

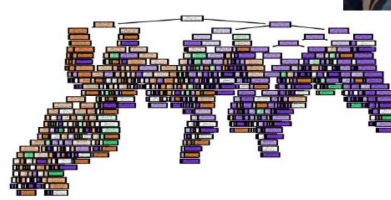
- Visualizing Decision Trees

Step 7 : Visualizing Decision Trees by performing the code below:

MayBank Stock price movement vs daily change of KLCI, DJI & SNP

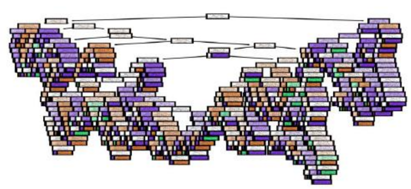
KLCI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_x']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```



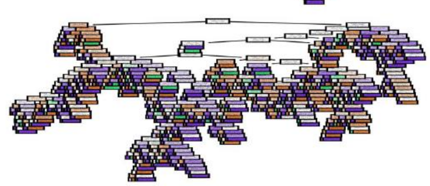
DJI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_y']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```



SNP1_Movement

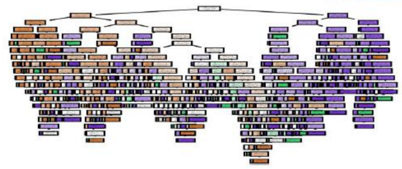
```
from matplotlib import pyplot as plt
fn=['Change']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename.png')
```



Step 7 : Visualizing Decision Trees by performing the code below: CIMB Stock price movement vs daily change of KLCI, DJI & SNP

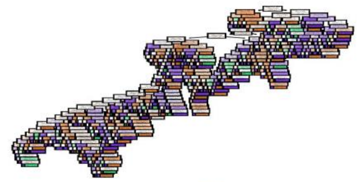
KLCI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_x']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename4.png')

tree.plot_tree(clf) # alternative
```



DJI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_y']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename5.png')

tree.plot_tree(clf) # alternative
```



SNP1_Movement

```
from matplotlib import pyplot as plt
fn=['Change']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename6.png')
```

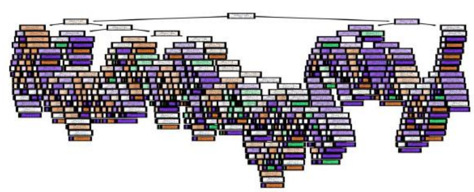
Step 7 : Visualizing Decision Trees by performing the code below: RHB Stock price movement vs daily change of KLCI, DJI & SNP

KLCI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_x']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename7.png')
tree.plot_tree(clf) # alternative
```

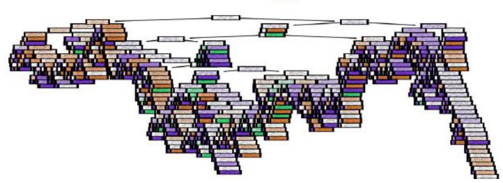
DJI1_Movement

```
from matplotlib import pyplot as plt
fn=['Change_y']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename8.png')
tree.plot_tree(clf) # alternative
```



SNP1_Movement

```
from matplotlib import pyplot as plt
fn=['Change']
cn=['Rise', 'Drop', 'No Change']
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('imagename9.png')
tree.plot_tree(clf) # alternative
```



Conclusion for final exam question 3

- The accuracy of tree decision for all Maybank, CIMB and RHB share in term of their respective movement against KLCI, DJI and SNP are not more than 50%
- Thus, the level of purity is very low, presenting a messy tree decision chart

(10 marks)

4. You are required to write code to find frequent itemsets using the above dataset (Question 1). In order to achieve the task, you are going to cover the following steps:
   - Importing required libraries

Step 1 : Importing required libraries

```
import pyfpgrowth
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
import numpy as np
```

```
# Training the Model
## Generating Frequent Itemsets
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

- Creating a list from dataset (Question 1)

Step 2 : Creating a list from dataset (Question 1)

i. Perform the code below to transform the data (on the left) into the required dataset as follows:

MBB2

```
      date       MBB_Movement
0     2020-03-09    No Change
1     2020-03-06         Drop
2     2020-03-05         Drop
3     2020-03-04         Rise
4     2020-03-03         Rise
...          ...          ...
1237  2015-03-16         Rise
1238  2015-03-13         Drop
1239  2015-03-12         Drop
1240  2015-03-11         Rise
1241  2015-03-10         Drop

[1242 rows x 2 columns]
```

KLCI2

```
      date       KLCI1_Movement
0     2020-03-09      No Change
1     2020-03-06           Drop
2     2020-03-05           Drop
3     2020-03-04           Rise
4     2020-03-03           Rise
...          ...            ...
1221  2015-03-16           Rise
1222  2015-03-13           Drop
1223  2015-03-12           Drop
1224  2015-03-11           Rise
1225  2015-03-10           Drop

[1226 rows x 2 columns]
```

CIMB2

```
      date       CIMB1_Movement
0     2020-03-09      No Change
1     2020-03-06           Drop
2     2020-03-05           Drop
3     2020-03-04           Drop
4     2020-03-03           Rise
...          ...            ...
1238  2015-03-16           Rise
1239  2015-03-13           Drop
1240  2015-03-12           Rise
1241  2015-03-11           Rise
1242  2015-03-10           Drop

[1243 rows x 2 columns]
```

DJI2

```
      date       DJI1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04          Drop
4     2020-03-03          Rise
...          ...           ...
1254  2015-03-16          Drop
1255  2015-03-13          Rise
1256  2015-03-12          Drop
1257  2015-03-11          Rise
1258  2015-03-10          Drop

[1259 rows x 2 columns]
```

RHB2

```
      date       RHB1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04     No Change
4     2020-03-03          Rise
...          ...           ...
1238  2015-03-16          Rise
1239  2015-03-13          Rise
1240  2015-03-12          Drop
1241  2015-03-11          Drop
1242  2015-03-10          Drop

[1243 rows x 2 columns]
```

SNP2

```
      date       SNP1_Movement
0     2020-03-09     No Change
1     2020-03-06          Drop
2     2020-03-05          Drop
3     2020-03-04          Drop
4     2020-03-03          Rise
...          ...           ...
1254  2015-03-16          Drop
1255  2015-03-13          Rise
1256  2015-03-12          Drop
1257  2015-03-11          Rise
1258  2015-03-10          Drop

[1259 rows x 2 columns]
```

```
TD_Dataset = pd.merge(MBB2, CIMB2,
                      how="left", on=["date"])
```

```
TD_Dataset1 = pd.merge(KLCI2, TD_Dataset,
                       how="left", on=["date"])
```

```
TD_Dataset2 = pd.merge(TD_Dataset1, RHB2,
                       how="left", on=["date"])
```

```
TD_Dataset3 = pd.merge(TD_Dataset2, DJI2,
                       how="left", on=["date"])
```

```
TD_Dataset4 = pd.merge(TD_Dataset3, SNP2,
                       how="left", on=["date"])
TD_Dataset4
```

	date	KLCI1_Movement	MBB_Movement	CIMB1_Movement	RHB1_Movement	DJI1_Movement	SNP1_Movement
0	2020-03-09	No Change	No Change	No Change	No Change	No Change	No Change
1	2020-03-06	Drop	Drop	Drop	Drop	Drop	Drop
2	2020-03-05	Drop	Drop	Drop	Drop	Drop	Drop
3	2020-03-04	Rise	Rise	Drop	No Change	Drop	Drop
4	2020-03-03	Rise	Rise	Rise	Rise	Rise	Rise
...	...	...	...	...	...	...	...
1221	2015-03-16	Rise	Rise	Rise	Rise	Drop	Drop
1222	2015-03-13	Drop	Drop	Drop	Rise	Rise	Rise
1223	2015-03-12	Drop	Drop	Rise	Drop	Drop	Drop
1224	2015-03-11	Rise	Rise	Rise	Drop	Rise	Rise
1225	2015-03-10	Drop	Drop	Drop	Drop	Drop	Drop

1226 rows × 7 columns

Step 2 : Creating a list from dataset (Question 1) (continued from previous slide)

ii. Perform the code below to change the value of every index/price movement into the required dataset as follows:

```
TD_Dataset = pd.merge(MBB2, CIMB2,
                      how="left", on=["date"])
```

```
TD_Dataset1 = pd.merge(KLCI2, TD_Dataset,
                       how="left", on=["date"])
```

```
TD_Dataset2 = pd.merge(TD_Dataset1, RHB2,
                       how="left", on=["date"])
```

```
TD_Dataset3 = pd.merge(TD_Dataset2, DJI2,
                       how="left", on=["date"])
```

```
TD_Dataset4 = pd.merge(TD_Dataset3, SNP2,
                       how="left", on=["date"])
TD_Dataset4
```

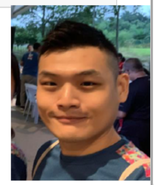
	date	KLCI1_Movement	MBB_Movement	CIMB1_Movement	RHB1_Movement	DJI1_Movement	SNP1_Movement
0	2020-03-09	No Change	No Change	No Change	No Change	No Change	No Change
1	2020-03-06	Drop	Drop	Drop	Drop	Drop	Drop
2	2020-03-05	Drop	Drop	Drop	Drop	Drop	Drop
3	2020-03-04	Rise	Rise	Drop	No Change	Drop	Drop
4	2020-03-03	Rise	Rise	Rise	Rise	Rise	Rise
...	...	...	...	...	...	...	...
1221	2015-03-16	Rise	Rise	Rise	Rise	Drop	Drop
1222	2015-03-13	Drop	Drop	Drop	Rise	Rise	Rise
1223	2015-03-12	Drop	Drop	Rise	Drop	Drop	Drop
1224	2015-03-11	Rise	Rise	Rise	Drop	Rise	Rise
1225	2015-03-10	Drop	Drop	Drop	Drop	Drop	Drop

1226 rows × 7 columns

```
TD_Dataset5 = TD_Dataset4.replace(to_replace={"KLCI1_Movement": {"No Change": "KLCI:No_Change", "Drop": "KLCI:Drop","Rise": "
                                  "MBB_Movement": {"No Change": "MBB:No_Change", "Drop": "MBB:Drop","Rise": "MBB:Rise"},
                                  "CIMB1_Movement": {"No Change": "CIMB:No_Change", "Drop": "CIMB:Drop","Rise": "CIMB:Rise"},
                                  "RHB1_Movement": {"No Change": "RHB:No_Change", "Drop": "RHB:Drop","Rise": "RHB:Rise"},
                                  "DJI1_Movement": {"No Change": "DJI:No_Change", "Drop": "DJI:Drop","Rise": "DJI:Rise"},
                                  "SNP1_Movement": {"No Change": "SNP:No_Change", "Drop": "SNP:Drop","Rise": "SNP:Rise"}})
```

```
print(TD_Dataset5)
```

```
           date  KLCI1_Movement    MBB_Movement  CIMB1_Movement   RHB1_Movement  \
0    2020-03-09  KLCI:No_Change   MBB:No_Change  CIMB:No_Change   RHB:No_Change
1    2020-03-06       KLCI:Drop        MBB:Drop       CIMB:Drop        RHB:Drop
2    2020-03-05       KLCI:Drop        MBB:Drop       CIMB:Drop        RHB:Drop
3    2020-03-04       KLCI:Rise        MBB:Rise       CIMB:Drop   RHB:No_Change
4    2020-03-03       KLCI:Rise        MBB:Rise       CIMB:Rise        RHB:Rise
...         ...             ...             ...             ...             ...
1221 2015-03-16       KLCI:Rise        MBB:Rise       CIMB:Rise        RHB:Rise
1222 2015-03-13       KLCI:Drop        MBB:Drop       CIMB:Drop        RHB:Rise
1223 2015-03-12       KLCI:Drop        MBB:Drop       CIMB:Rise        RHB:Drop
1224 2015-03-11       KLCI:Rise        MBB:Rise       CIMB:Rise        RHB:Drop
1225 2015-03-10       KLCI:Drop        MBB:Drop       CIMB:Drop        RHB:Drop

      DJI1_Movement  SNP1_Movement
0     DJI:No_Change  SNP:No_Change
1          DJI:Drop       SNP:Drop
2          DJI:Drop       SNP:Drop
3          DJI:Drop       SNP:Drop
4          DJI:Rise       SNP:Rise
...             ...            ...
1221       DJI:Drop       SNP:Drop
1222       DJI:Rise       SNP:Rise
1223       DJI:Drop       SNP:Drop
1224       DJI:Rise       SNP:Rise
1225       DJI:Drop       SNP:Drop

[1226 rows x 7 columns]
```



```
TD_Dataset5.isnull().sum()
TD_Dataset6 = TD_Dataset5.dropna(how='any',axis=0)
TD_Dataset6.isnull().sum()
```

```
date              0
KLCI1_Movement    0
MBB_Movement      0
CIMB1_Movement    0
RHB1_Movement     0
DJI1_Movement     0
SNP1_Movement     0
dtype: int64
```

Ensure no empty value

Step 2 : Creating a list from dataset (Question 1) *(continued from previous slide)*

iii. Perform the code to preprocess the dataset as follows:

a. Change the trading date to "string" type of data

```
TD_Dataset6['date'] = TD_Dataset6['date'].astype('str')
TD_Dataset6
```

b. Feature selection

```
KLCI = TD_Dataset6.loc[:,['date','KLCI1_Movement']]
MBB = TD_Dataset6.loc[:,['date','MBB_Movement']]
CIMB = TD_Dataset6.loc[:,['date','CIMB1_Movement']]
RHB = TD_Dataset6.loc[:,['date','RHB1_Movement']]
DJI = TD_Dataset6.loc[:,['date','DJI1_Movement']]
SNP = TD_Dataset6.loc[:,['date','SNP1_Movement']]
```

c. Rename the column name

```
KLCI.rename(columns={'date':'Trading_Date',
                     'KLCI1_Movement':'Movement'},inplace=True)
MBB.rename(columns={'date':'Trading_Date',
                    'MBB_Movement':'Movement'},inplace=True)
CIMB.rename(columns={'date':'Trading_Date',
                     'CIMB1_Movement':'Movement'},inplace=True)
RHB.rename(columns={'date':'Trading_Date',
                    'RHB1_Movement':'Movement'},inplace=True)
DJI.rename(columns={'date':'Trading_Date',
                    'DJI1_Movement':'Movement'},inplace=True)
SNP.rename(columns={'date':'Trading_Date',
                    'SNP1_Movement':'Movement'},inplace=True)
```

d. Combine the dataset and create a frequency column

```
DATASET_AM = pd.concat([MBB, CIMB, RHB, KLCI, DJI, SNP], ignore_index=True)
DATASET_AM.sort_values(by=['Trading_Date'], inplace=True)
DATASET_AM["Frequency"] = 1
DATASET_AM
```

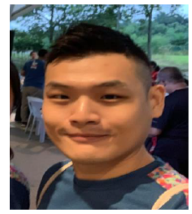
	Trading_Date	Movement	Frequency
3578	2015-03-10	RHB:Drop	1
5964	2015-03-10	DJI:Drop	1
4771	2015-03-10	KLCI:Drop	1
2385	2015-03-10	CIMB:Drop	1
1192	2015-03-10	MBB:Drop	1
...	...	...	...
3579	2020-03-09	KLCI:No_Change	1
4772	2020-03-09	DJI:No_Change	1
5965	2020-03-09	SNP:No_Change	1
2386	2020-03-09	RHB:No_Change	1
0	2020-03-09	MBB:No_Change	1

7158 rows × 3 columns

- Convert list to dataframe with boolean values

Step 3 :Convert list to data frame with boolean values

i. Perform code below to create the “Basket” of dataset with Boolean values :



```
MyBasket= (DATASET_AM.groupby(['Trading_Date','Movement'])['Frequency'].sum()
           .unstack().reset_index().fillna(0).set_index('Trading_Date'))

def my_encode_units(x):
    if x <=0:
        return 0
    if x >=1:
        return 1

my_basket_sets = MyBasket.applymap(my_encode_units)
my_basket_sets
```

Movement	CIMB:Drop	CIMB:No_Change	CIMB:Rise	DJI:Drop	DJI:No_Change	DJI:Rise	KLCI:Drop	KLCI:No_Change	KLCI:Rise	MBB:Drop	MBB:No_
Trading_Date											
2015-03-10	1	0	0	1	0	0	1	0	0	1	
2015-03-11	0	0	1	0	0	1	0	0	1	0	
2015-03-12	0	0	1	1	0	0	1	0	0	1	
2015-03-13	1	0	0	0	0	1	1	0	0	1	
2015-03-16	0	0	1	1	0	0	0	0	1	0	
...	...	...	...	...	...	...	...	...	...	...	
2020-03-03	0	0	1	0	0	1	0	0	1	0	
2020-03-04	1	0	0	1	0	0	0	0	1	0	
2020-03-05	1	0	0	1	0	0	1	0	0	1	
2020-03-06	1	0	0	1	0	0	1	0	0	1	
2020-03-09	0	1	0	0	1	0	0	1	0	0	

1193 rows x 18 columns

- Find frequently occurring itemsets using Apriori Algorithm

Step 4 : Find frequently occurring itemsets using Apriori Algorithm

i. Perform code below to create the “Basket” of dataset :

```
my_frequent_itemsets = apriori(my_basket_sets, min_support = 0.07, use_colnames = True)
my_frequent_itemsets
```

	support	itemsets
0	0.433361	(CIMB:Drop)
1	0.131601	(CIMB:No_Change)
2	0.435038	(CIMB:Rise)
3	0.455993	(DJI:Drop)
4	0.541492	(DJI:Rise)
...	...	...
240	0.087175	(DJI:Drop, KLCI:Drop, RHB:Drop, MBB:Drop, SNP:...
241	0.072087	(DJI:Drop, MBB:Rise, RHB:Rise, SNP:Drop, KLCI:...
242	0.087175	(DJI:Rise, KLCI:Drop, RHB:Drop, MBB:Drop, SNP:...
243	0.116513	(DJI:Rise, MBB:Rise, RHB:Rise, SNP:Rise, KLCI:...
244	0.090528	(DJI:Rise, MBB:Rise, RHB:Rise, CIMB:Rise, SNP:...

245 rows × 2 columns

- Find frequently occurring itemsets using F-P Growth

Step 5 : Find frequently occurring itemsets using F-P Growth

i. Perform code below to create the listing of "Basket" for the dataset and the patterns frequency :

```
grouped_df = DATASET_AM.groupby(by = ['Trading_Date'])

transactions = []
for group, pdf in grouped_df:
    transactions.append(pdf['Movement'].values.tolist())

transactions
```

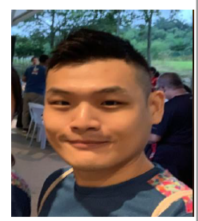
```
[['RHB:Drop', 'DJI:Drop', 'KLCI:Drop', 'CIMB:Drop', 'MBB:Drop', 'SNP:Drop'],
 ['SNP:Rise', 'CIMB:Rise', 'RHB:Drop', 'MBB:Rise', 'DJI:Rise', 'KLCI:Rise'],
 ['DJI:Drop', 'RHB:Drop', 'CIMB:Rise', 'SNP:Drop', 'KLCI:Drop', 'MBB:Drop'],
 ['KLCI:Drop', 'RHB:Rise', 'CIMB:Drop', 'DJI:Rise', 'MBB:Drop', 'SNP:Rise'],
 ['RHB:Rise', 'MBB:Rise', 'SNP:Drop', 'KLCI:Rise', 'CIMB:Rise', 'DJI:Drop'],
 ['KLCI:Rise',
  'RHB:Drop',
  'CIMB:Drop',
  'SNP:Rise',
  'DJI:Rise',
  'MBB:No_Change'],
 ['SNP:Drop', 'CIMB:Rise', 'DJI:Drop', 'KLCI:Rise', 'MBB:Rise', 'RHB:Rise'],
 ['CIMB:No_Change',
  'MBB:Drop',
  'SNP:Rise',
  'RHB:Drop',
  'DJI:Rise',
  'KLCI:Drop'],
 ['DJI:Drop', 'SNP:Drop', 'CIMB:Drop', 'MBB:Drop', 'KLCI:Drop', 'RHB:Rise'],
```

```
patterns = pyfpgrowth.find_frequent_patterns(transactions, 300)
patterns
```

```
{('RHB:Drop',): 507,
 ('KLCI:Drop', 'RHB:Drop'): 345,
 ('CIMB:Drop', 'MBB:Drop'): 300,
 ('KLCI:Drop', 'MBB:Drop'): 387,
 ('CIMB:Drop',): 517,
 ('CIMB:Drop', 'KLCI:Drop'): 389,
 ('CIMB:Rise', 'MBB:Rise'): 313,
 ('CIMB:Rise', 'KLCI:Rise'): 377,
 ('RHB:Rise', 'SNP:Rise'): 301,
 ('DJI:Rise', 'RHB:Rise'): 307,
 ('KLCI:Rise', 'RHB:Rise'): 360,
 ('DJI:Rise', 'MBB:Rise'): 300,
 ('MBB:Rise', 'SNP:Rise'): 305,
 ('KLCI:Rise', 'MBB:Rise'): 391,
 ('DJI:Drop',): 544,
 ('DJI:Drop', 'SNP:Drop'): 474,
 ('SNP:Drop',): 545,
 ('KLCI:Rise', 'SNP:Rise'): 337,
 ('DJI:Rise', 'KLCI:Rise', 'SNP:Rise'): 307,
 ('DJI:Rise', 'KLCI:Rise'): 341,
 ('DJI:Rise', 'KLCI:Drop'): 305,
 ('KLCI:Drop', 'SNP:Rise'): 309,
 ('SNP:Rise',): 646,
 ('DJI:Rise',): 646}
```

- Mine the Association Rules

Step 6 : Mine the Association Rules using Apriori Algorithm & F-P Growth



i. Perform code below to calculate support, confidence and lift value for the respective combination of items (mining the association rule) :

```
my_rules = association_rules(my_frequent_itemsets, metric = 'lift', min_threshold=1)
```

```
my_rules.head()
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(DJI:Drop)	(CIMB:Drop)	0.455993	0.433361	0.199497	0.437500	1.009550	0.001887	1.007358
1	(CIMB:Drop)	(DJI:Drop)	0.433361	0.455993	0.199497	0.460348	1.009550	0.001887	1.008070
2	(KLCI:Drop)	(CIMB:Drop)	0.504610	0.433361	0.326069	0.646179	1.491087	0.107390	1.601486
3	(CIMB:Drop)	(KLCI:Drop)	0.433361	0.504610	0.326069	0.752418	1.491087	0.107390	2.000910
4	(MBB:Drop)	(CIMB:Drop)	0.431685	0.433361	0.251467	0.582524	1.344200	0.064391	1.357297

```
my_rules[(my_rules['lift']>=3.5)&
        (my_rules['confidence']>=0.7)]
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
1403	(MBB:Rise, CIMB:Rise, SNP:Drop)	(DJI:Drop, KLCI:Rise)	0.108131	0.207041	0.080469	0.744186	3.594388	0.058082	3.099749
1605	(DJI:Drop, MBB:Rise, RHB:Rise)	(KLCI:Rise, SNP:Drop)	0.096396	0.210394	0.072087	0.747826	3.554408	0.051806	3.131196
1611	(MBB:Rise, RHB:Rise, SNP:Drop)	(DJI:Drop, KLCI:Rise)	0.094719	0.207041	0.072087	0.761062	3.675898	0.052476	3.318680

ii. Perform code below to mine the combination of items that have 0.8 threshold set for confidence (mining the association rule) :

```
rules = pyfpgrowth.generate_association_rules(patterns, 0.80)
rules
```

```
{('DJI:Drop',): (('SNP:Drop',), 0.8713235294117647),
 ('SNP:Drop',): (('DJI:Drop',), 0.8697247706422019),
 ('DJI:Rise', 'KLCI:Rise'): (('SNP:Rise',), 0.9002932551319648),
 ('KLCI:Rise', 'SNP:Rise'): (('DJI:Rise',), 0.9109792284866469)}
```

Conclusion for final exam question 4

- Using Apriori required multiple scans of the database to check the support count of each item and itemsets. When the database is huge, this will cost a significant amount of disk I/O and computing power. Therefore the FP-Growth algorithm is created

(10 marks)

The student is expected to submit answers to each question individually, and submit the document in PDF format. The student can include online materials, screenshots, videos and/or codes (ipynb format) to support your answer