Explain Video Link Q1 & Q2: https://drive.google.com/file/d/1zu-Yk6b57rPoFyw8g3H165IiiBgAwN-S/view (https://drive.google.com/file/d/1zu-Yk6b57rPoFyw8g3H165IiiBgAwN-S/view)

Explain Video Link Q3 & Q4: https://drive.google.com/file/d/1gwI6rBtY4PIWR9yt8BE0JJjFFjhoqNMM/view (https://drive.google.com/file/d/1gwI6rBtY4PIWR9yt8BE0JJjFFjhoqNMM/view)

Name: LIU,HONGYANG 17201091/1

Q1:

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)

import libraries

In [1]:

```
#import libraries
import requests
import pandas as pd
import time
```

web crawling

In [2]:

```
# user-agent
url_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_3) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/80.0.3987.122 Safari/537.36'

headers = {'User-Agent':url_agent}

Name=["customer","transaction","product"]

# WWW websites
url="https://beijing01.github.io/DataWarehouse/"

content=[]

for i in Name:

    #execute the crawlling task every 1 seconds to avoid banning by the server

    time.sleep(1)

    try:
        # build url
        webpage =url+i+".html"

        resp = requests.get(webpage,headers=headers)

        df = pd.read_html(resp.text)

        content.append(df[0])

    except:
        print("The website doesn't work!",url+i)
```

data preprocessing

In [3]:

```
data=content[0]
customer = {'customer_id':data.iloc[:,1],'DOB':data.iloc[:,2],'Gender':data.iloc
[:,3],
            'City_id':data.iloc[:,4],'City':data.iloc[:,5]}
customer = pd.DataFrame(customer)
```

In [4]:

```
#customer
customer.head()
```

Out[4]:

	customer_id	DOB	Gender	City_id	City
0	268408	02-01-1970	M	4.0	Tianjin
1	269696	07-01-1970	F	8.0	Chaohu
2	268159	08-01-1970	F	8.0	Chaohu
3	270181	10-01-1970	F	2.0	Chongqing
4	268073	11-01-1970	M	1.0	Beijing

In [5]:

```
data=content[1]

transactions = {'transaction_id':data.iloc[:,1],'customer_id':data.iloc[:,2],'tr
ansaction_date':data.iloc[:,3],
                'prod_cat_code':data.iloc[:,4],'prod_subcat_code':data.iloc[:,5
],
                'Quantity':data.iloc[:,6],'Price':data.iloc[:,7],'Tax':data.iloc
[:,8],
                'Total_amount':data.iloc[:,9],'Store_type':data.iloc[:,10]}
transactions = pd.DataFrame(transactions)
```

In [6]:

```
transactions.head()
```

Out[6]:

	transaction_id	customer_id	transaction_date	prod_cat_code	prod_subcat_code	Quantity
0	80712190438	270351	28-02-2014	1	1	-5
1	29258453508	270384	27-02-2014	5	3	-5
2	51750724947	273420	24-02-2014	6	5	-2
3	93274880719	271509	24-02-2014	11	6	-3
4	51750724947	273420	23-02-2014	6	5	-2

In [7]:

```
#transaction
transactions.head()
```

Out[7]:

	transaction_id	customer_id	transaction_date	prod_cat_code	prod_subcat_code	Quantity
0	80712190438	270351	28-02-2014	1	1	-5
1	29258453508	270384	27-02-2014	5	3	-5
2	51750724947	273420	24-02-2014	6	5	-2
3	93274880719	271509	24-02-2014	11	6	-3
4	51750724947	273420	23-02-2014	6	5	-2

In [8]:

```
data=content[2]

product={'pro_cat_code':data.iloc[:,1],'pro_cat':data.iloc[:,2],
         'prod_subcat_code':data.iloc[:,3],'prod_subcat':data.iloc[:,4]}

product = pd.DataFrame(product)
```

In [9]:

```
#product
product.head()
```

Out[9]:

	pro_cat_code	pro_cat	prod_subcat_code	prod_subcat
0	1	Clothing	4	Mens
1	1	Clothing	1	Women
2	1	Clothing	3	Kids
3	2	Footwear	1	Mens
4	2	Footwear	3	Women

save to csv file

In [10]:

```
# store the data to csv format
customer.to_csv("customer.csv")
transactions.to_csv("transactions.csv")
product.to_csv("product.csv")
```

Q2:

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

In [11]:

```
# check the data sets generated in first step
#customer
customer.head()
```

Out[11]:

	customer_id	DOB	Gender	City_id	City
0	268408	02-01-1970	M	4.0	Tianjin
1	269696	07-01-1970	F	8.0	Chaohu
2	268159	08-01-1970	F	8.0	Chaohu
3	270181	10-01-1970	F	2.0	Chongqing
4	268073	11-01-1970	M	1.0	Beijing

In [12]:

```
#transaction
transactions.head()
```

Out[12]:

	transaction_id	customer_id	transaction_date	prod_cat_code	prod_subcat_code	Quantity
0	80712190438	270351	28-02-2014	1	1	-5
1	29258453508	270384	27-02-2014	5	3	-5
2	51750724947	273420	24-02-2014	6	5	-2
3	93274880719	271509	24-02-2014	11	6	-3
4	51750724947	273420	23-02-2014	6	5	-2

In [13]:

```
#product
product.head()
```

Out[13]:

	pro_cat_code	pro_cat	prod_subcat_code	prod_subcat
0	1	Clothing	4	Mens
1	1	Clothing	1	Women
2	1	Clothing	3	Kids
3	2	Footwear	1	Mens
4	2	Footwear	3	Women

The snowflake schema should meet the three requirements

- 1 build **fact table** surrended by **dimension tables**
- 2 reduce data redundancy
- 3 Normalized data sturcture

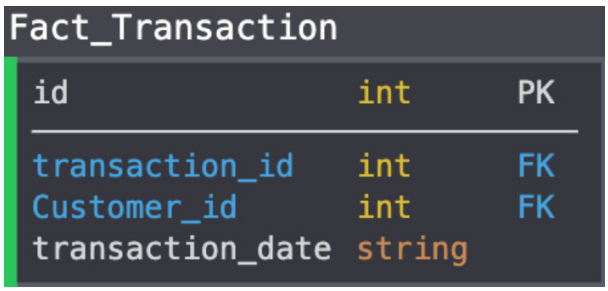
We will draw the the entity relationship diagram to represent the snowflake schema

In [14]:

```
#draw the fact table to represent the transaction and customer information
Fact_Transaction=transactions[["transaction_id","customer_id","transaction_date"
]]
Fact_Transaction.head(3)
```

Out[14]:

	transaction_id	customer_id	transaction_date
0	80712190438	270351	28-02-2014
1	29258453508	270384	27-02-2014
2	51750724947	273420	24-02-2014

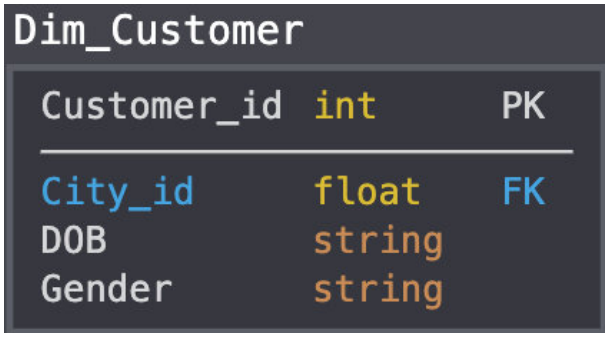


In [15]:

```
#draw customer dim table to represent customer information
Dim_Customer=customer[["customer_id","DOB","Gender","City_id"]]
Dim_Customer.head(3)
```

Out[15]:

	customer_id	DOB	Gender	City_id
0	268408	02-01-1970	M	4.0
1	269696	07-01-1970	F	8.0
2	268159	08-01-1970	F	8.0



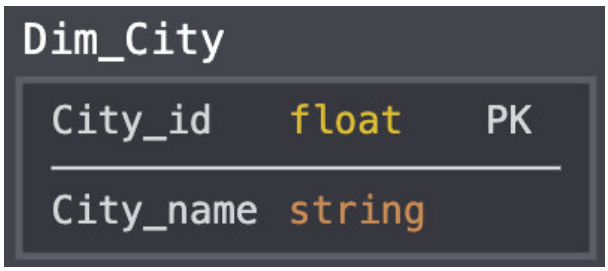
In [16]:

```
#draw city dim table

Dim_City=customer[["City_id","City"]]
Dim_City.head(3)
```

Out[16]:

	City_id	City
0	4.0	Tianjin
1	8.0	Chaohu
2	8.0	Chaohu



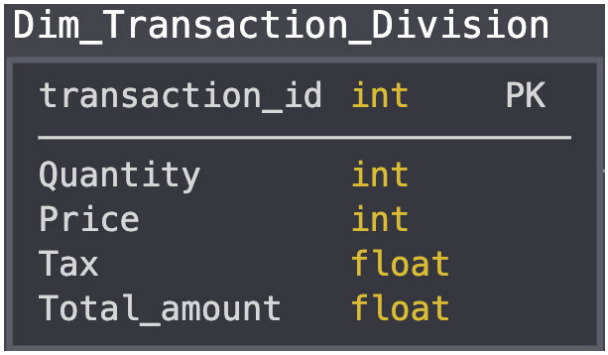
In [17]:

```
#draw Transaction_Division dim table to represent the goods, number, price  in e
very transaction

Dim_Transaction_Division=transactions[["transaction_id","Quantity","Price","Tax"
,"Total_amount"]]
Dim_Transaction_Division.head(3)
```

Out[17]:

	transaction_id	Quantity	Price	Tax	Total_amount
0	80712190438	-5	-772	405.300	-4265.300
1	29258453508	-5	-1497	785.925	-8270.925
2	51750724947	-2	-791	166.110	-1748.110

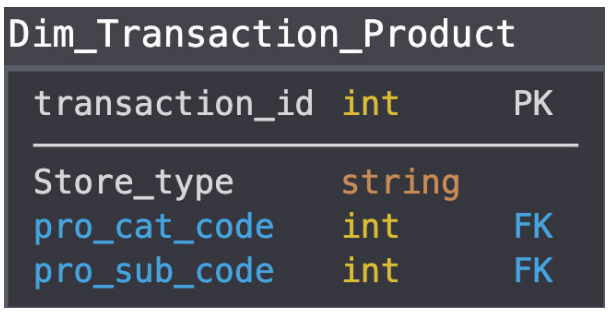


In [18]:

```
# draw the transaction product table
Dim_Transaction_Product=transactions[["transaction_id","Store_type","prod_cat_code","prod_subcat_code"]]
Dim_Transaction_Product.head(3)
```

Out[18]:

	transaction_id	Store_type	prod_cat_code	prod_subcat_code
0	80712190438	e-Shop	1	1
1	29258453508	e-Shop	5	3
2	51750724947	TeleShop	6	5

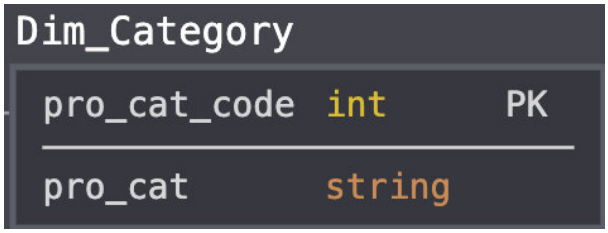


In [19]:

```
# draw the product category table
Dim_Category=product[["pro_cat_code","pro_cat"]]
Dim_Category.head(3)
```

Out[19]:

	pro_cat_code	pro_cat
0	1	Clothing
1	1	Clothing
2	1	Clothing

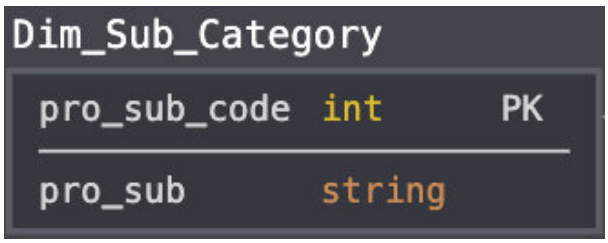


In [20]:

```
#draw the product subcategory table
Dim_Sub_Category=product[["prod_subcat_code","prod_subcat"]]
Dim_Sub_Category.head(3)
```

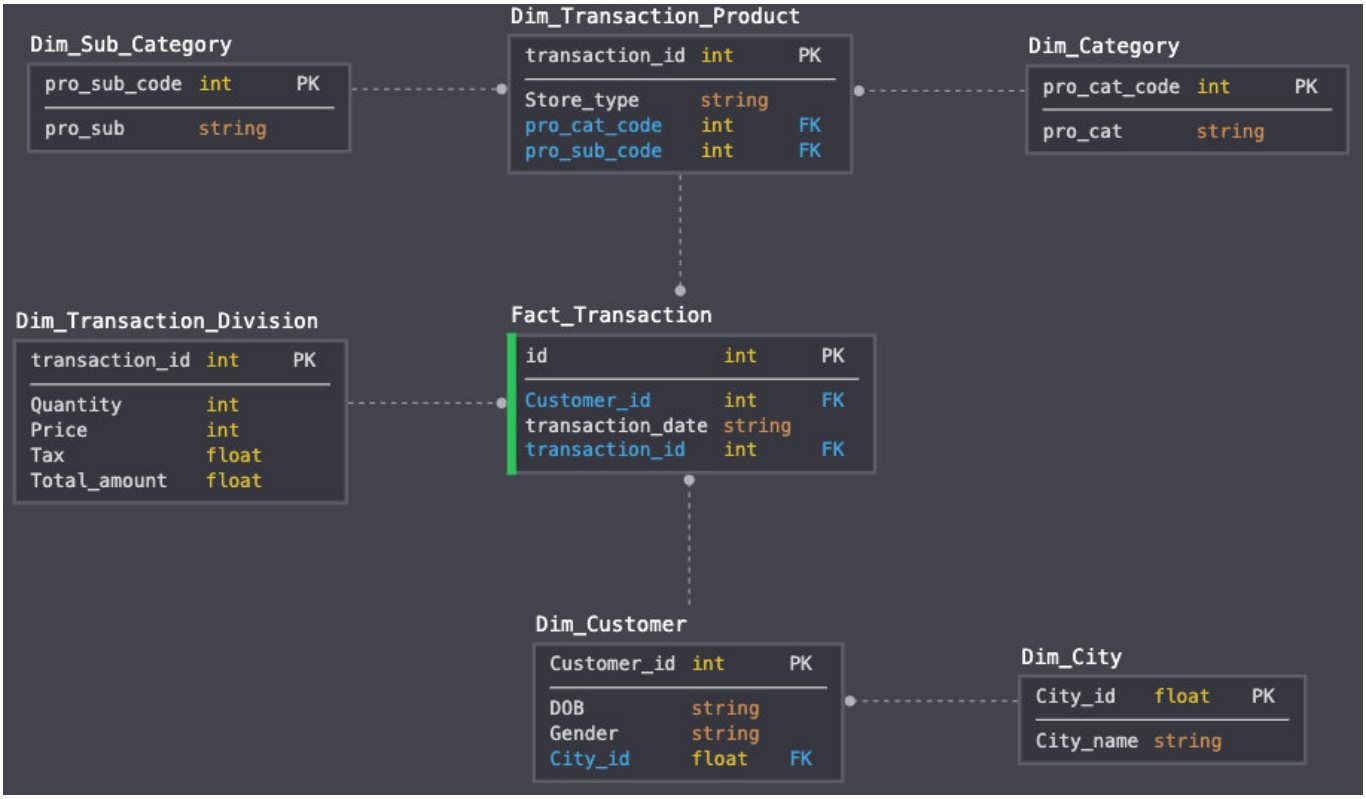
Out[20]:

	prod_subcat_code	prod_subcat
0	4	Mens
1	1	Women
2	3	Kids



Draw snowflake schema diagram for the above dataset

- PK: Primary Key
- FK: Foreign Key



Q3:

1. 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:
   - Importing required libraries
   - Loading Data
   - Feature Selection
   - Splitting Data
   - Building Decision Tree Model
   - Evaluating Model
   - Visualizing Decision Trees

Objective: we decide to build a supervised learning model to predict the consumers' age group based on the transactions data

Importing required libraries

In [21]:

```
import requests
import pandas as pd
import time
from sklearn import tree
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import classification_report, accuracy_score
from sklearn import metrics
import numpy as np
```

Loading Data

In [22]:

```
customer=pd.read_csv("customer.csv",header=0,index_col=0)
transactions=pd.read_csv("transactions.csv",header=0,index_col=0)
product=pd.read_csv("product.csv",header=0,index_col=0)
```

In [23]:

```
# we need to merge the data sets
```

In [24]:

```
#1 merge "product" and "transactions" table and create the table "prod_tran"
product=product.rename(columns={"pro_cat_code":"prod_cat_code"})
prod_tran = pd.merge(left=transactions, right=product,on=["prod_cat_code","prod_
subcat_code"],how="left")
```

In [25]:

```
#2 merge "prod_tran" and "customer" table and create the table "df"
df= pd.merge(left=prod_tran, right=customer,right_on="customer_id", left_on="cus
tomer_id", how="left")
```

In [26]:

```
df =df.drop_duplicates()
df.head()
```

Out[26]:

	transaction_id	customer_id	transaction_date	prod_cat_code	prod_subcat_code	Quantity
0	80712190438	270351	28-02-2014	1	1	-5
1	29258453508	270384	27-02-2014	5	3	-5
2	51750724947	273420	24-02-2014	6	5	-2
3	93274880719	271509	24-02-2014	11	6	-3
4	51750724947	273420	23-02-2014	6	5	-2

In [27]:

```
len(df)
```

Out[27]:

```
23040
```

Feature Selection

In [28]:

```
#calculate the customers's age using tran_date(transaction date) and DOB attribu
tes(Date or birth)

df['transaction_date'] = pd.to_datetime(df['transaction_date'], errors='coerce')
df.insert(loc=3, column='Tran_year', value= df.transaction_date.dt.year)


df['DOB'] = pd.to_datetime(df['DOB'], errors='coerce')
df.insert(loc=4, column='Birth_year', value= df.DOB.dt.year)



df['Tran_year']=df['Tran_year'].astype(int)
df['Birth_year']=df['Birth_year'].astype(int)

df["age"]=df['Tran_year'] -df['Birth_year']

df= df.drop(['Tran_year','Birth_year'],axis=1)
```

In [29]:

```
# build age_category column based on age

df["age_category"] = df["age"]
df["age_category"] = df["age_category"].apply(lambda x: "young aduld" if x<=25 e
lse "aduld")
```

In [30]:

```
# the columns that we have created

df[["age","age_category"]].head()
```

Out[30]:

	age	age_category
0	33	aduld
1	41	aduld
2	22	young aduld
3	33	aduld
4	22	young aduld

In [31]:

```
# Feature Selection
Features=["Gender","prod_cat_code","Store_type","prod_subcat_code","Total_amoun
t"]
Target=["age_category"]
```

Splitting Data

In [32]:

```
# format the data sets
X= pd.get_dummies(df[Features])
Y= pd.get_dummies(df[Target])

# train datsets 70% ; test datasets 30%
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3,random_s
tate=30)
print(X_train.shape)
print(y_test.shape)
```

```
(16128, 9)
(6912, 2)
```

Building Decision Tree Model

In [33]:

```
decision_tree = DecisionTreeClassifier(random_state=30)

decisiontree = decision_tree.fit(X_train,y_train)
```

Evaluating Model

In [34]:

```
predicted=decisiontree.predict(X_test)
accuracy = metrics.accuracy_score(y_test, predicted)
precision = metrics.precision_score(y_test, predicted, average='weighted')
recall = metrics.recall_score(y_test, predicted, average='weighted')
f1_score=metrics.f1_score(y_test, predicted, average='weighted')
```

In [35]:

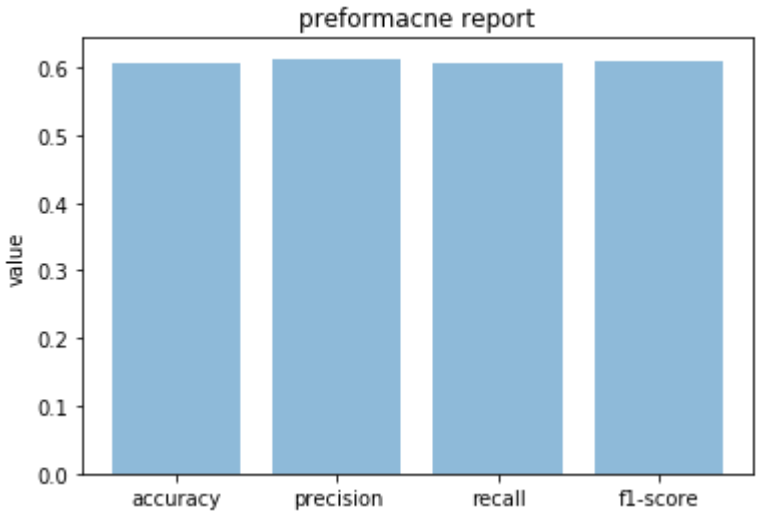
```
print("performance of accuracy:",accuracy)
print("performance of precision:",precision)
print("performance of recall:",recall)
print("performance of f1_score:",f1_score)

objects = ("accuracy","precision","recall","f1-score")
y_pos = np.arange(len(objects))
performance = [accuracy,precision,recall,f1_score]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('value')
plt.title('preformacne report')

plt.show()
```

```
performance of accuracy: 0.6076388888888888
performance of precision: 0.6132605524318323
performance of recall: 0.6076388888888888
performance of f1_score: 0.6104071139119456
```



In [36]:

```
X_test.iloc[:10]
```

Out[36]:

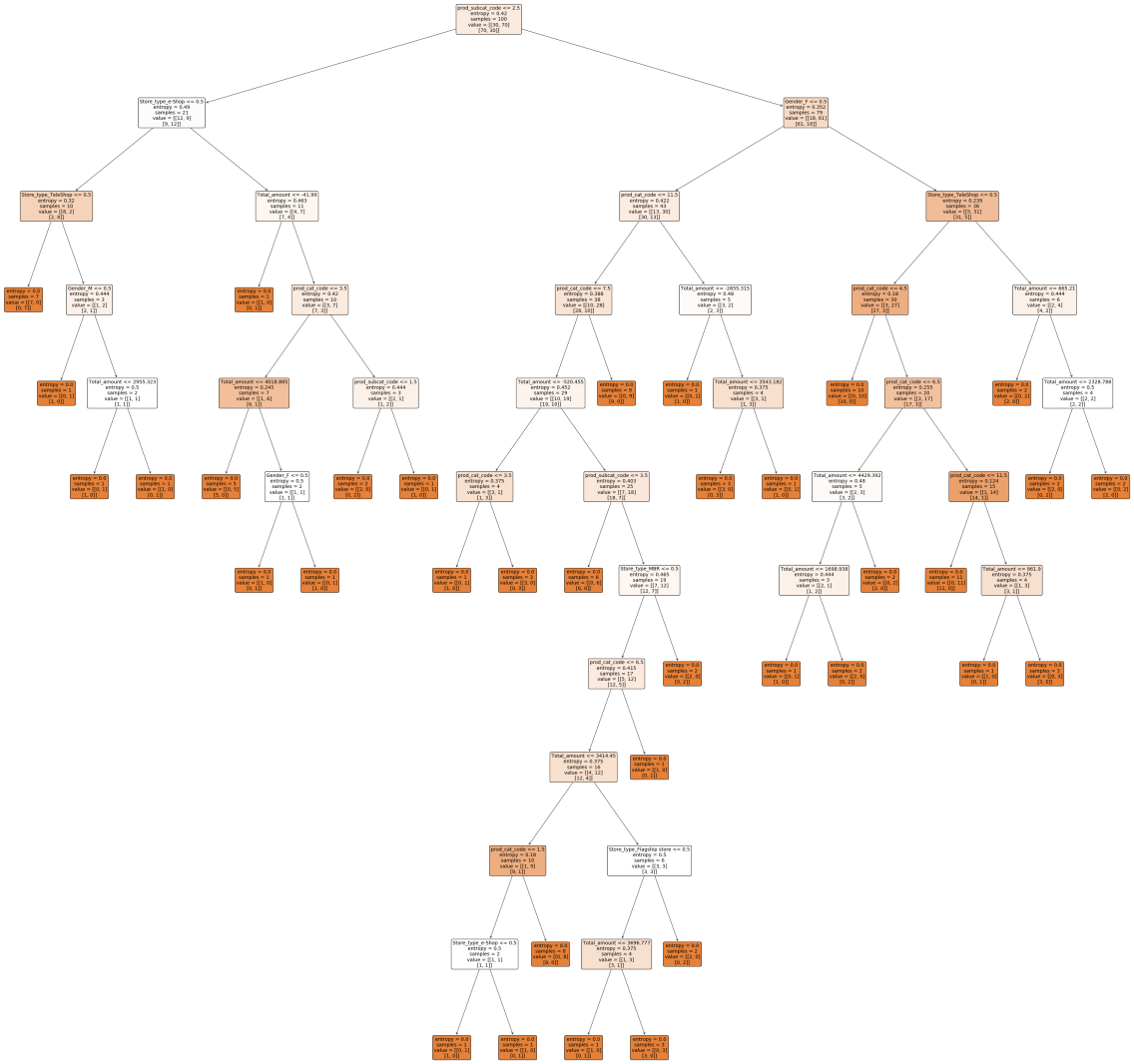
	prod_cat_code	prod_subcat_code	Total_amount	Gender_F	Gender_M	Store_type_Flags
21092	6	5	4038.775	1	0	
21465	1	2	3656.445	1	0	
12490	9	3	7226.700	1	0	
3879	7	5	-3348.150	1	0	
14124	1	4	1339.260	0	1	
721	11	5	2994.550	1	0	
10283	10	5	742.560	1	0	
13094	6	5	8160.425	0	1	
4190	6	5	1403.350	0	1	
3594	3	1	-508.300	1	0	

Visualizing Decision Trees

In [37]:

```
from sklearn.tree import plot_tree
decision_tree = DecisionTreeClassifier() #max_depth is maximum number of levels
 in the tree
decision_tree.fit(X_test.iloc[:100],y_test.iloc[:100])

plt.figure(figsize=(60,60))
a = plot_tree(decision_tree,
              feature_names=X.columns,
              class_names=Y.columns,
              filled=True,
              rounded=True,
              fontsize=14)
```



Q4:

1. 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
   - Creating a list from dataset (Question 1)
   - Convert list to dataframe with boolean values
   - Find frequently occurring itemsets using Apriori Algorithm
   - Find frequently occurring itemsets using F-P Growth
   - Mine the Association Rules

Importing required libraries

In [38]:

```
import pandas as pd
import numpy as np
from pandas import DataFrame
import seaborn as sns
from mlxtend.frequent_patterns import apriori, association_rules
import pyfpgrowth
```

Creating a list from dataset

In [39]:

```
# view data
itemsets = df[["customer_id","Store_type"]]
itemsets.head()
```

Out[39]:

	customer_id	Store_type
0	270351	e-Shop
1	270384	e-Shop
2	273420	TeleShop
3	271509	e-Shop
4	273420	TeleShop

In [40]:

```
itemsets.head()
```

Out[40]:

	customer_id	Store_type
0	270351	e-Shop
1	270384	e-Shop
2	273420	TeleShop
3	271509	e-Shop
4	273420	TeleShop

In [41]:

```
itemsets=itemsets.drop_duplicates(['customer_id','Store_type'],keep='first')
```

Creating a list from dataset

In [42]:

```
# list for Apriori Algorithm
relationship=(itemsets.groupby(['customer_id','Store_type']).
              size().unstack().reset_index().fillna(0).set_index('customer_id'))
relationship.iloc[0:10,:]
```

Out[42]:

Store_type	Flagship store	MBR	TeleShop	e-Shop
customer_id				
266783	0.0	0.0	1.0	1.0
266784	1.0	0.0	1.0	1.0
266785	1.0	0.0	1.0	1.0
266788	1.0	1.0	0.0	1.0
266794	1.0	1.0	1.0	1.0
266799	1.0	0.0	0.0	1.0
266803	0.0	0.0	0.0	1.0
266804	0.0	0.0	0.0	1.0
266805	0.0	0.0	0.0	1.0
266806	1.0	0.0	0.0	1.0

In [43]:

```
# list for F-P Growth
itemsets2= itemsets.copy()
itemsets2['Store_type'] = itemsets2['Store_type'].apply(lambda x:[x])

df_group = itemsets2.groupby(['customer_id'])['Store_type'].apply(sum).reset_ind
ex()

df_group
```

Out[43]:

	customer_id	Store_type
0	266783	[e-Shop, TeleShop]
1	266784	[Flagship store, e-Shop, TeleShop]
2	266785	[TeleShop, Flagship store, e-Shop]
3	266788	[e-Shop, MBR, Flagship store]
4	266794	[e-Shop, TeleShop, Flagship store, MBR]
5	266799	[Flagship store, e-Shop]
6	266803	[e-Shop]
7	266804	[e-Shop]
8	266805	[e-Shop]
9	266806	[e-Shop, Flagship store]
10	266807	[e-Shop, MBR, Flagship store]
11	266809	[MBR, e-Shop, TeleShop]
12	266810	[e-Shop, TeleShop]
13	266812	[TeleShop, MBR]
14	266813	[MBR]
15	266814	[MBR, Flagship store, e-Shop, TeleShop]
16	266815	[TeleShop, e-Shop, MBR]
17	266816	[e-Shop, TeleShop, Flagship store]
18	266817	[MBR, e-Shop, TeleShop]
19	266818	[e-Shop, TeleShop, MBR]
20	266819	[Flagship store, e-Shop]
21	266820	[Flagship store, e-Shop]
22	266821	[TeleShop, MBR]
23	266822	[e-Shop, TeleShop, Flagship store]
24	266823	[Flagship store, TeleShop]
25	266824	[MBR, TeleShop, e-Shop]
26	266825	[MBR, e-Shop]
27	266827	[TeleShop, Flagship store, e-Shop]
28	266829	[MBR, e-Shop]
29	266830	[MBR, TeleShop, e-Shop]
...	...	...
5476	275222	[MBR, e-Shop]
5477	275224	[e-Shop, Flagship store, MBR]
5478	275225	[MBR, e-Shop, TeleShop]
5479	275226	[TeleShop, Flagship store, MBR, e-Shop]
5480	275227	[MBR, e-Shop, TeleShop]

	customer_id	Store_type
5481	275228	[TeleShop]
5482	275229	[e-Shop, TeleShop]
5483	275230	[e-Shop, MBR]
5484	275231	[e-Shop, TeleShop, MBR]
5485	275232	[e-Shop]
5486	275233	[e-Shop, Flagship store]
5487	275236	[e-Shop]
5488	275237	[Flagship store]
5489	275241	[Flagship store, e-Shop]
5490	275242	[e-Shop]
5491	275243	[TeleShop, e-Shop]
5492	275244	[TeleShop, Flagship store, MBR]
5493	275245	[MBR, e-Shop, Flagship store]
5494	275246	[e-Shop, TeleShop, Flagship store]
5495	275247	[Flagship store, TeleShop, e-Shop]
5496	275249	[Flagship store, TeleShop, e-Shop]
5497	275250	[e-Shop, MBR]
5498	275251	[MBR, e-Shop]
5499	275252	[TeleShop, e-Shop, Flagship store, MBR]
5500	275255	[e-Shop]
5501	275257	[e-Shop, MBR]
5502	275261	[MBR, e-Shop, TeleShop]
5503	275262	[Flagship store, MBR]
5504	275264	[e-Shop, TeleShop]
5505	275265	[Flagship store, e-Shop, TeleShop]

5506 rows × 2 columns

Convert list to dataframe with boolean values

In [44]:

```
relationship=(itemsets.groupby(['customer_id','Store_type']).
              size().unstack().reset_index().fillna(0).set_index('customer_id'))
.replace({0.0:False,1.0:True})
relationship.iloc[0:10,:]
```

Out[44]:

Store_type	Flagship store	MBR	TeleShop	e-Shop
customer_id				
266783	False	False	True	True
266784	True	False	True	True
266785	True	False	True	True
266788	True	True	False	True
266794	True	True	True	True
266799	True	False	False	True
266803	False	False	False	True
266804	False	False	False	True
266805	False	False	False	True
266806	True	False	False	True

Find frequently occurring itemsets using Apriori Algorithm

In [45]:

```
frequent=apriori(relationship,min_support=0.05,use_colnames=True)

frequent.sort_values(by='support',ascending=False).head(10)
```

Out[45]:

	support	itemsets
3	0.792953	(e-Shop)
1	0.544860	(MBR)
0	0.534145	(Flagship store)
2	0.528878	(TeleShop)
8	0.424446	(MBR, e-Shop)
9	0.408645	(TeleShop, e-Shop)
6	0.407374	(e-Shop, Flagship store)
7	0.286233	(TeleShop, MBR)
4	0.283509	(MBR, Flagship store)
5	0.271704	(TeleShop, Flagship store)

Find frequently occurring itemsets using F-P Growth

In [46]:

```
df_group.Store_type.head()
```

Out[46]:

```
0                          [e-Shop, TeleShop]
1          [Flagship store, e-Shop, TeleShop]
2          [TeleShop, Flagship store, e-Shop]
3               [e-Shop, MBR, Flagship store]
4    [e-Shop, TeleShop, Flagship store, MBR]
Name: Store_type, dtype: object
```

In [47]:

```
import pyfpgrowth

FP_patterns = pyfpgrowth.find_frequent_patterns(df_group.Store_type, 2)
```

In [48]:

```
print("Patterns")
FP_patterns
```

```
Patterns
```

Out[48]:

```
{('Flagship store', 'MBR', 'TeleShop'): 794,
 ('Flagship store', 'MBR', 'TeleShop', 'e-Shop'): 618,
 ('Flagship store', 'TeleShop', 'e-Shop'): 1136,
 ('MBR', 'TeleShop'): 1576,
 ('MBR', 'TeleShop', 'e-Shop'): 1233,
 ('TeleShop', 'e-Shop'): 2250,
 ('Flagship store', 'MBR'): 1561,
 ('Flagship store', 'MBR', 'e-Shop'): 1205,
 ('Flagship store', 'e-Shop'): 2243,
 ('MBR',): 3000,
 ('MBR', 'e-Shop'): 2337,
 ('e-Shop',): 4366}
```

Mine the Association Rules

- **using F-P growth frequency results to generate rules**

In [49]:

```
rules = pyfpgrowth.generate_association_rules(FP_patterns, 0.6)

print("Rules")
rules
```

```
Rules
```

Out[49]:

```
{('Flagship store', 'MBR', 'TeleShop'): (('e-Shop',), 0.778337531486
1462),
 ('MBR', 'TeleShop'): (('e-Shop',), 0.7823604060913706),
 ('Flagship store', 'MBR'): (('e-Shop',), 0.7719410634208841),
 ('MBR',): (('e-Shop',), 0.779)}
```

- **using Apriori Algorithm result to generate rules**

In [50]:

```
rules=association_rules(frequent,metric='lift',min_threshold=0.8)
#The results have been sorted using lift attribute. Lift is the confidence divid
ed by expected confidence.
rules.sort_values("lift",ascending=False)
```

Out[50]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
34	(MBR)	(TeleShop, e-Shop)	0.544860	0.408645	0.223938	0.411000	1.005763	0.00
31	(TeleShop, e-Shop)	(MBR)	0.408645	0.544860	0.223938	0.548000	1.005763	0.00
38	(TeleShop, e-Shop, Flagship store)	(MBR)	0.206320	0.544860	0.112241	0.544014	0.998447	-0.00
47	(MBR)	(TeleShop, e-Shop, Flagship store)	0.544860	0.206320	0.112241	0.206000	0.998447	-0.00
33	(TeleShop)	(MBR, e-Shop)	0.528878	0.424446	0.223938	0.423420	0.997583	-0.00
32	(MBR, e-Shop)	(TeleShop)	0.424446	0.528878	0.223938	0.527599	0.997583	-0.00
6	(TeleShop)	(MBR)	0.528878	0.544860	0.286233	0.541209	0.993299	-0.00
7	(MBR)	(TeleShop)	0.544860	0.528878	0.286233	0.525333	0.993299	-0.00
35	(e-Shop)	(TeleShop, MBR)	0.792953	0.286233	0.223938	0.282410	0.986641	-0.00
30	(TeleShop, MBR)	(e-Shop)	0.286233	0.792953	0.223938	0.782360	0.986641	-0.00
20	(e-Shop, Flagship store)	(MBR)	0.407374	0.544860	0.218852	0.537227	0.985990	-0.00
21	(MBR)	(e-Shop, Flagship store)	0.544860	0.407374	0.218852	0.401667	0.985990	-0.00
9	(e-Shop)	(MBR)	0.792953	0.544860	0.424446	0.535273	0.982404	-0.00
8	(MBR)	(e-Shop)	0.544860	0.792953	0.424446	0.779000	0.982404	-0.00
48	(e-Shop)	(TeleShop, MBR, Flagship store)	0.792953	0.144206	0.112241	0.141548	0.981568	-0.00
37	(TeleShop, MBR, Flagship store)	(e-Shop)	0.144206	0.792953	0.112241	0.778338	0.981568	-0.00
11	(e-Shop)	(TeleShop)	0.792953	0.528878	0.408645	0.515346	0.974414	-0.01
10	(TeleShop)	(e-Shop)	0.528878	0.792953	0.408645	0.772665	0.974414	-0.01
1	(Flagship store)	(MBR)	0.534145	0.544860	0.283509	0.530772	0.974143	-0.00
0	(MBR)	(Flagship store)	0.544860	0.534145	0.283509	0.520333	0.974143	-0.00
16	(MBR)	(TeleShop, Flagship store)	0.544860	0.271704	0.144206	0.264667	0.974101	-0.00

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
13	(TeleShop, Flagship store)	(MBR)	0.271704	0.544860	0.144206	0.530749	0.974101	-0.0(
22	(e-Shop)	(MBR, Flagship store)	0.792953	0.283509	0.218852	0.275996	0.973501	-0.0(
19	(MBR, Flagship store)	(e-Shop)	0.283509	0.792953	0.218852	0.771941	0.973501	-0.0(
43	(MBR, e-Shop)	(TeleShop, Flagship store)	0.424446	0.271704	0.112241	0.264442	0.973272	-0.0(
42	(TeleShop, Flagship store)	(MBR, e-Shop)	0.271704	0.424446	0.112241	0.413102	0.973272	-0.0(
46	(TeleShop)	(MBR, e-Shop, Flagship store)	0.528878	0.218852	0.112241	0.212225	0.969720	-0.0(
39	(MBR, e-Shop, Flagship store)	(TeleShop)	0.218852	0.528878	0.112241	0.512863	0.969720	-0.0(
44	(MBR, Flagship store)	(TeleShop, e-Shop)	0.283509	0.408645	0.112241	0.395900	0.968811	-0.0(
41	(TeleShop, e-Shop)	(MBR, Flagship store)	0.408645	0.283509	0.112241	0.274667	0.968811	-0.0(
18	(MBR, e-Shop)	(Flagship store)	0.424446	0.534145	0.218852	0.515618	0.965316	-0.0(
23	(Flagship store)	(MBR, e-Shop)	0.534145	0.424446	0.218852	0.409725	0.965316	-0.0(
45	(e-Shop, Flagship store)	(TeleShop, MBR)	0.407374	0.286233	0.112241	0.275524	0.962585	-0.0(
40	(TeleShop, MBR)	(e-Shop, Flagship store)	0.286233	0.407374	0.112241	0.392132	0.962585	-0.0(
5	(Flagship store)	(e-Shop)	0.534145	0.792953	0.407374	0.762666	0.961804	-0.01
4	(e-Shop)	(Flagship store)	0.792953	0.534145	0.407374	0.513743	0.961804	-0.01
2	(TeleShop)	(Flagship store)	0.528878	0.534145	0.271704	0.513736	0.961793	-0.01
3	(Flagship store)	(TeleShop)	0.534145	0.528878	0.271704	0.508671	0.961793	-0.01
15	(TeleShop)	(MBR, Flagship store)	0.528878	0.283509	0.144206	0.272665	0.961751	-0.0(
14	(MBR, Flagship store)	(TeleShop)	0.283509	0.528878	0.144206	0.508648	0.961751	-0.0(

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leve
25	(TeleShop, Flagship store)	(e-Shop)	0.271704	0.792953	0.206320	0.759358	0.957633	-0.0(
28	(e-Shop)	(TeleShop, Flagship store)	0.792953	0.271704	0.206320	0.260192	0.957633	-0.0(
27	(TeleShop)	(e-Shop, Flagship store)	0.528878	0.407374	0.206320	0.390110	0.957622	-0.0(
26	(e-Shop, Flagship store)	(TeleShop)	0.407374	0.528878	0.206320	0.506465	0.957622	-0.0(
29	(Flagship store)	(TeleShop, e-Shop)	0.534145	0.408645	0.206320	0.386263	0.945229	-0.01
24	(TeleShop, e-Shop)	(Flagship store)	0.408645	0.534145	0.206320	0.504889	0.945229	-0.01
17	(Flagship store)	(TeleShop, MBR)	0.534145	0.286233	0.144206	0.269976	0.943204	-0.0(
12	(TeleShop, MBR)	(Flagship store)	0.286233	0.534145	0.144206	0.503807	0.943204	-0.0(
49	(Flagship store)	(TeleShop, MBR, e-Shop)	0.534145	0.223938	0.112241	0.210133	0.938354	-0.0(
36	(TeleShop, MBR, e-Shop)	(Flagship store)	0.223938	0.534145	0.112241	0.501217	0.938354	-0.0(

In [ ]: