Projet 4_Analyse de ventes

Parcours Data Analyst

Xuefei ZHANG_2021

Readme

Ce projet est composé principalement de 3 parties: traitement et nettoyage, analyse univariée, et analyse bivariée.

Partie 1 se déroule comme suit:

- Import des librairies
- Import des CSV et création de 3 dataframes originaux: products, customers, transactions
- Création du dataframe principal: trans_prod_cus
- Nettoyage et traitement du dataframe principal
- Enrichissement du dataframe principal: création de nouvelles colonnes (âge, year, month, period…) pour faciliter l’exploitation et l’analyse

<u>Partie 2 consiste à :</u>

Réaliser des analyses univariées pour certaines variables du dataframe:

- Price
- Catégorie (categ)
- Age

Réaliser des analyses bivariées (qualitative et quantitative) entre les variables du dataframe:

- Age VS taille du panier moyen (nombre d’articles par session)
- Age VS Fréquence d’achat (nombre d'achats)
- Catégorie VS Sexe
- Age VS Montant d'achat
- Catégorie VS Montant d’achat
- Age VS Catégorie
- Age VS Montant d’achat

Sommaire

1. Dataframe principale et nettoyage

- Constitution de dataframe principale
- Nettoyage de données
- Traitement de données manquantes

Dataframes originaux :

customers , products , transactions

In [786]:

```
customers = pd.read_csv("customers.csv")

print(customers.describe(include="all"))
print(customers.isnull().values.any())
customers.head()
```

```
       client_id  sex        birth
count       8623 8623  8623.000000
unique      8623    2          NaN
top       c_1565    f          NaN
freq           1 4491          NaN
mean         NaN  NaN  1978.280877
std          NaN  NaN    16.919535
min          NaN  NaN  1929.000000
25%          NaN  NaN  1966.000000
50%          NaN  NaN  1979.000000
75%          NaN  NaN  1992.000000
max          NaN  NaN  2004.000000
False
```

Out[786]:

	client_id	sex	birth
0	c_4410	f	1967
1	c_7839	f	1975
2	c_1699	f	1984
3	c_5961	f	1962
4	c_5320	m	1943

In [787]:

```
products= pd.read_csv("products.csv")
print(products.describe(include="all"))
print(products.isnull().values.any())
print(products.info())
products.sort_values(by= 'price', ascending= True).head()
```

```
       id_prod        price        categ
count     3287  3287.000000  3287.000000
unique    3287          NaN          NaN
top       1_50          NaN          NaN
freq         1          NaN          NaN
mean       NaN    21.856641     0.370246
std        NaN    29.847908     0.615387
min        NaN    -1.000000     0.000000
25%        NaN     6.990000     0.000000
50%        NaN    13.060000     0.000000
75%        NaN    22.990000     1.000000
max        NaN   300.000000     2.000000
False
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3287 entries, 0 to 3286
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id_prod  3287 non-null   object
 1   price    3287 non-null   float64
 2   categ    3287 non-null   int64
dtypes: float64(1), int64(1), object(1)
memory usage: 77.2+ KB
None
```

Out[787]:

	id_prod	price	categ
731	T_0	-1.00	0
2355	0_202	0.62	0
2272	0_528	0.62	0
370	0_120	0.66	0
1211	0_1844	0.77	0

the product with id_prod = T_0 has its price negative, to clean

In [1096]:

```
transactions= pd.read_csv("transactions.csv")
print(transactions.describe(include="all"), sep='\n')
print(products.isnull().values.any())
print(transactions.info())
transactions.head()
```

```
       id_prod                             date session_id client_id
count   337016                           337016     337016    337016
unique    3266                           336855     169195      8602
top      1_369  test_2021-03-01 02:30:02.237413        s_0    c_1609
freq      1081                               13        200     12855
False
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 337016 entries, 0 to 337015
Data columns (total 4 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     337016 non-null  object
 1   date        337016 non-null  object
 2   session_id  337016 non-null  object
 3   client_id   337016 non-null  object
dtypes: object(4)
memory usage: 10.3+ MB
None
```

Out[1096]:

	id_prod	date	session_id	client_id
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242

Repérer et traiter les valeurs manquantes et aberrantes - products

In [6]:

```
products = products[products.id_prod != "T_0"]
print(products.info())
products.sort_values(by= 'price', ascending= True).head()

# the product with negative price is droped from the dataframe
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3286 entries, 0 to 3286
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   id_prod  3286 non-null   object
 1   price    3286 non-null   float64
 2   categ    3286 non-null   int64
dtypes: float64(1), int64(1), object(1)
memory usage: 102.7+ KB
None
```

Out[6]:

	id_prod	price	categ
2272	0_528	0.62	0
2355	0_202	0.62	0
370	0_120	0.66	0
1211	0_1844	0.77	0
1530	0_1620	0.80	0

Le produit avec id_prod = T_0 est enlevé du dataframe

Repérer et traiter valeurs manquantes et aberrantes - transactions

In [13]:

```
# Select all duplicate rows based on values of [date] column

date_duplicate = transactions[transactions.duplicated(['date'])]

print(date_duplicate.describe(include="all"))

print("Duplicate rows based on the column 'date' are:")
date_duplicate.head()
```

```
       id_prod                          date session_id client_id
count     161                           161        161       161
unique      1                            37          1         2
top       T_0  test_2021-03-01 02:30:02.237413        s_0      ct_0
freq      161                            12        161        84
Duplicate rows based on the column 'date' are:
```

Out[13]:

	id_prod	date	session_id	client_id
27161	T_0	test_2021-03-01 02:30:02.237434	s_0	ct_0
34387	T_0	test_2021-03-01 02:30:02.237443	s_0	ct_0
48425	T_0	test_2021-03-01 02:30:02.237443	s_0	ct_1
54813	T_0	test_2021-03-01 02:30:02.237412	s_0	ct_1
56373	T_0	test_2021-03-01 02:30:02.237446	s_0	ct_0

In [16]:

```
#To drop duplicate rows in [date] column

transac=transactions[transactions.id_prod != "T_0"]
print(transac.describe())
transac.head()

transac2 = transac[transac.session_id != "s_0"]
print(transac2.describe())

# => no difference in case we deduplicate by id_prod or by session_id

transac.head()
```

```
       id_prod                        date session_id client_id
count   336816                      336816     336816    336816
unique    3265                      336816     169194      8600
top      1_369  2022-02-28 22:55:33.742567   s_118668    c_1609
freq      1081                           1         14     12855
       id_prod                        date session_id client_id
count   336816                      336816     336816    336816
unique    3265                      336816     169194      8600
top      1_369  2022-02-28 22:55:33.742567   s_118668    c_1609
freq      1081                           1         14     12855
```

Out[16]:

	id_prod	date	session_id	client_id
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242

it's clear that all these duplicated rows have some points in common: id_prod = T_0, date started by 'test_', session_id = 's=0'

Combiner 3 dataframes dans un seul: trans_prod_cus

In [19]:

```
# to merge 3 dataframes into trans_prod_cus via primary keys

trans_prod_cus = pd.merge(trans_prod, trans_cus, on = ['date', 'id_prod', 'session_id', 'client_id'])
print(trans_prod_cus.info())
trans_prod_cus.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 336816 entries, 0 to 336815
Data columns (total 8 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     336816 non-null  object
 1   date        336816 non-null  object
 2   session_id  336816 non-null  object
 3   client_id   336816 non-null  object
 4   price       336713 non-null  float64
 5   categ       336713 non-null  float64
 6   sex         336816 non-null  object
 7   birth       336816 non-null  int64
dtypes: float64(2), int64(1), object(5)
memory usage: 23.1+ MB
None
```

Out[19]:

	id_prod	date	session_id	client_id	price	categ	sex	birth
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450	4.99	0.0	f	1977
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277	65.75	2.0	f	2000
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270	10.71	1.0	f	1979
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597	4.20	0.0	m	1963
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242	8.99	0.0	f	1980

Conversion de types de data et création de nouvelles colonnes

In [21]:

```python
#convert the date and birth columns from object into datetime

trans_prod_cus['date'] = pd.to_datetime(trans_prod_cus.date)
trans_prod_cus['birth'] = pd.to_datetime(trans_prod_cus['birth'], format = '%Y', errors

print(trans_prod_cus.info())
trans_prod_cus.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 336816 entries, 0 to 336815
Data columns (total 8 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   id_prod     336816 non-null  object
 1   date        336816 non-null  datetime64[ns]
 2   session_id  336816 non-null  object
 3   client_id   336816 non-null  object
 4   price       336713 non-null  float64
 5   categ       336713 non-null  float64
 6   sex         336816 non-null  object
 7   birth       336816 non-null  datetime64[ns]
dtypes: datetime64[ns](2), float64(2), object(4)
memory usage: 23.1+ MB
None
```

Out[21]:

	id_prod	date	session_id	client_id	price	categ	sex	
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450	4.99	0.0	f	197
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277	65.75	2.0	f	200
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270	10.71	1.0	f	197
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597	4.20	0.0	m	196
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242	8.99	0.0	f	198

In [26]:

```python
trans_prod_cus['year-month-day'] = pd.to_datetime(trans_prod_cus ['date']).dt.date
trans_prod_cus['year'] = pd.to_datetime(trans_prod_cus ['date']).dt.year
trans_prod_cus['month'] = pd.to_datetime(trans_prod_cus ['date']).dt.month
trans_prod_cus["period"] = trans_prod_cus["year"].astype(str)  + trans_prod_cus["month"].astype(str)
trans_prod_cus["period"] = pd.to_numeric(trans_prod_cus["period"])

print(trans_prod_cus.info())
print(trans_prod_cus.isnull().values.any())
trans_prod_cus.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 336816 entries, 0 to 336815
Data columns (total 13 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   id_prod         336816 non-null  object
 1   date            336816 non-null  datetime64[ns]
 2   session_id      336816 non-null  object
 3   client_id       336816 non-null  object
 4   price           336713 non-null  float64
 5   categ           336713 non-null  float64
 6   sex             336816 non-null  object
 7   birth           336816 non-null  datetime64[ns]
 8   age             336816 non-null  int64
 9   year-month-day  336816 non-null  object
 10  year            336816 non-null  int64
 11  month           336816 non-null  int64
 12  period          336816 non-null  int64
dtypes: datetime64[ns](2), float64(2), int64(4), object(5)
memory usage: 36.0+ MB
None
True
```

Out[26]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	year-month-day	year	month	period
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450	4.99	0.0	f	1977-01-01	44	2021-04-10	2021	4	20214
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277	65.75	2.0	f	2000-01-01	21	2022-02-03	2022	2	20222
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270	10.71	1.0	f	1979-01-01	42	2021-09-23	2021	9	20219
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597	4.20	0.0	m	1963-01-01	58	2021-10-17	2021	10	202110
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242	8.99	0.0	f	1980-01-01	41	2021-07-17	2021	7	20217

Repérer et traiter les valeurs null

In [28]:
```
pricenull = pd.isnull(trans_prod_cus['price'])
categnull = pd.isnull(trans_prod_cus['categ'])
print(trans_prod_cus[pricenull].shape)
print(trans_prod_cus[categnull].shape)
```

```
(103, 13)
(103, 13)
```

Repérer des null dans les colonnes price et categ

=> then we find that the product with price and categ = NaN is product 0_2245.

given the id_prod starts always by the product categ, we can identify that product 0_2245 has categ 0.

In [29]:
```
trans_prod_cus[pricenull]
```

Out[29]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	year-month-day	year	month	period
6231	0_2245	2021-06-17 03:03:12.668129	s_49705	c_1533	NaN	NaN	m	1972-01-01	49	2021-06-17	2021	6	20216
10797	0_2245	2021-06-16 05:53:01.627491	s_49323	c_7954	NaN	NaN	m	1973-01-01	48	2021-06-16	2021	6	20216
14045	0_2245	2021-11-24 17:35:59.911427	s_124474	c_5120	NaN	NaN	f	1975-01-01	46	2021-11-24	2021	11	202111
17480	0_2245	2022-02-28 18:08:49.875709	s_172304	c_4964	NaN	NaN	f	1982-01-01	39	2022-02-28	2022	2	20222
21071	0_2245	2021-03-01 00:09:29.301897	s_3	c_580	NaN	NaN	m	1988-01-01	33	2021-03-01	2021	3	20213
21754	0_2245	2022-02-11 09:05:43.952857	s_163405	c_1098	NaN	NaN	m	1986-01-01	35	2022-02-11	2022	2	20222
22672	0_2245	2021-10-19 00:28:01.920054	s_106841	c_3953	NaN	NaN	f	1984-01-01	37	2021-10-19	2021	10	202110
24576	0_2245	2022-02-25 00:08:08.736068	s_170426	c_6236	NaN	NaN	f	1976-01-01	45	2022-02-25	2022	2	20222
30874	0_2245	2021-08-22 08:51:27.564509	s_79102	c_6752	NaN	NaN	m	1987-01-01	34	2021-08-22	2021	8	20218
31330	0_2245	2021-05-12 03:36:34.586221	s_33316	c_6205	NaN	NaN	f	1984-01-01	37	2021-05-12	2021	5	20215
34893	0_2245	2021-10-15 09:31:31.539354	s_105069	c_4188	NaN	NaN	f	1935-01-01	86	2021-10-15	2021	10	202110

À remarquer que pour tous ces valeurs null de colonne categ et price, id_prod est le même : 0_2245

=> Il faut trouver la categ et le price de ce produit dans dataframe products, et puis imputation

Traitement des valeurs manquantes dans colonnes price et categ - Imputation

In [31]:
```python
# price in array of products categ 0
price0 = products.loc[products['categ']== 0 , 'price' ].values
price0
```

Out[31]: array([19.99, 5.13, 17.99, ..., 17.14, 11.22, 25.16])

In [33]:
```python
import statistics

print("Average price for products in categ 0:", round(statistics.mean(price0),2))
print("Median price for products in categ 0:", round(statistics.median(price0),2))
print("Mode of price for products in categ 0:", statistics.mode(price0))
```

```
Average price for products in categ 0: 11.73
Median price for products in categ 0: 10.32
Mode of price for products in categ 0: 4.99
```

In [25]:
```python
# to fill NaN in trans_prod_cus

trans_prod_cus.loc[trans_prod_cus['id_prod'] == "0_2245", ['price'] ] =  trans_prod_cus['price'].fillna(10.32)
trans_prod_cus.loc[trans_prod_cus['id_prod'] == "0_2245", ['categ'] ] =  trans_prod_cus['categ'].fillna(0)

print('after fillna, the number of null values in dataframe is:', trans_prod_cus.isnull().sum().sum())
trans_prod_cus.head()
```

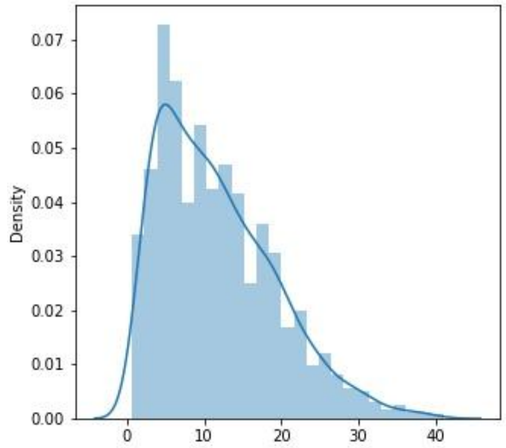
after fillna, the number of null values in dataframe is: 0

Out[25]:

	id_prod	date	session_id	client_id	price	categ	sex	birth	age	year-month-day	year	month	period
0	0_1483	2021-04-10 18:37:28.723910	s_18746	c_4450	4.99	0.0	f	1977-01-01	44	2021-04-10	2021	4	20214
1	2_226	2022-02-03 01:55:53.276402	s_159142	c_277	65.75	2.0	f	2000-01-01	21	2022-02-03	2022	2	20222
2	1_374	2021-09-23 15:13:46.938559	s_94290	c_4270	10.71	1.0	f	1979-01-01	42	2021-09-23	2021	9	20219
3	0_2186	2021-10-17 03:27:18.783634	s_105936	c_4597	4.20	0.0	m	1963-01-01	58	2021-10-17	2021	10	202110
4	0_1351	2021-07-17 20:34:25.800563	s_63642	c_1242	8.99	0.0	f	1980-01-01	41	2021-07-17	2021	7	20217

In dataframe 'products', there is NOT a product with id_prod = 0_2245

- but it's sure that it is in categ 0
- thus for price, need to decide among mean/mode/median, which to use in categ 0



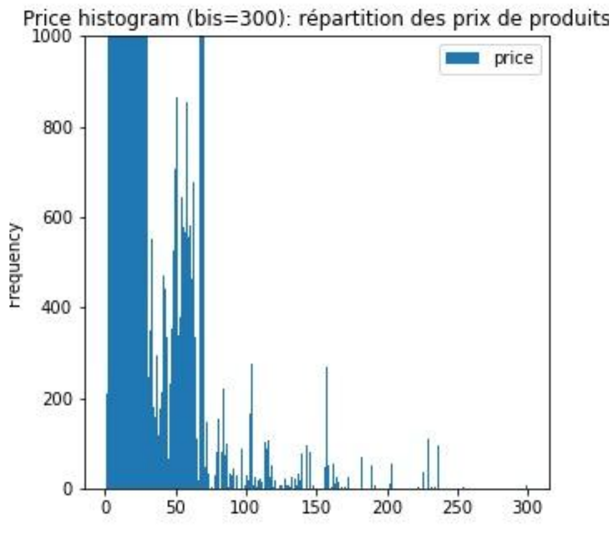
Based on this histogram, it's appropriate to use **median** of price in categ0

Analyse univariée

- Price
- Catégorie (categ)
- Age

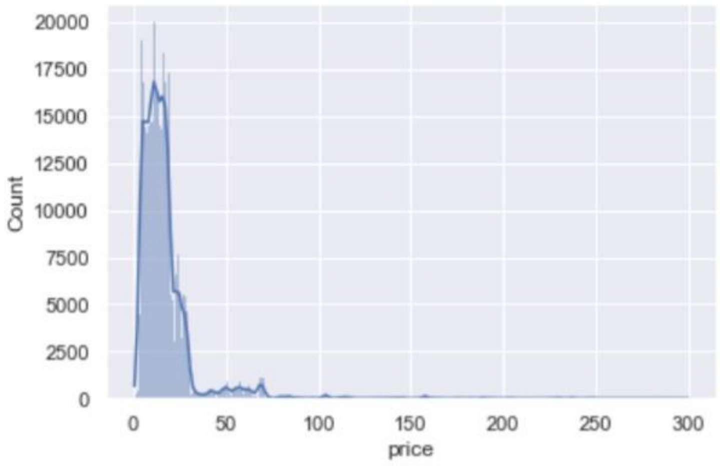
Univariée - price - histogramme et kde

In [119]:
```
trans_prod_cus.plot(kind="hist",
            y="price",
            figsize=(5,5),
          bins= 300,
            ylim=(0,1000))

plt.savefig('uni_categ_hist.jpg')
```



Répartition des prix de tous produits avec Kde

```
sns.set(style="darkgrid")
sns.histplot(data=trans_prod_cus, x="price",  bins=300, kde = True)
plt.show()
plt.savefig('uni_price_hist_kde.jpg')
```



Mesure d’asymétrie

Left-skewed: price se concentre fortement dans l’ éventail 0 - 30

Univariée - price - courbe Lorenz et indice Gini

In [136]:
```
depenses2 = trans_prod_cus['price'].values
n2 = len(depenses2)
lorenz2 = np.cumsum(np.sort(depenses2)) / depenses2.sum()
lorenz2 = np.append([0], lorenz2)     # La courbe de Lorenz commence à 0

fig, ax = plt.subplots(figsize=[5,5])
plt.axes().axis('equal')
xaxis = np.linspace(0-1/n,1+1/n, n+1)
plt.title("Lorenz - concentration des prix")
plt.plot(xaxis, lorenz2, drawstyle='steps-post', color = "darkblue" )
plt.plot(xaxis, xaxis, color = "blue")

plt.savefig("lorenz_courbe_price2.jpg")
```

In [138]:
```
AUC2 = (lorenz2.sum() -lorenz2[-1]/2 -lorenz2[0]/2)/n2
S2 = 0.5 - AUC2
gini2 = 2*S2
round(gini2,4)
```

Out[138]: 0.3921

indice de Gini = 39.21%:

40% du montant de chiffres d'affaires viennent de 60% (100%-40%) des clients.

Selon la loi de Pareto (20/80), qui est la normalité, ici 40/60 est moins concentré que 20/80

=> une concentration pas très forte et proche égalitaire.

Univariée - catégorie (categ) - pie chart et boxplot

```
]: uni_categ= trans_prod_cus[['categ','price']].groupby(by=["categ"])
   uni_categ.head()
   sumunicateg= uni_categ.sum()
   display("mean of price by categ", uni_categ.mean())
   display("sum of price by categ", uni_categ.sum())
   display("median by categ:", uni_categ.median())
```

'mean of price by categ'

	price
categ	
0.0	10.646668
1.0	20.480106
2.0	75.174949

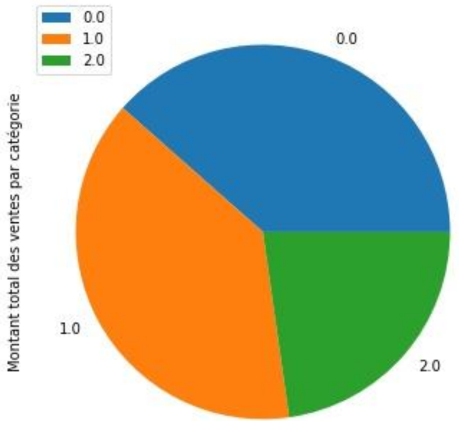
'sum of price by categ'

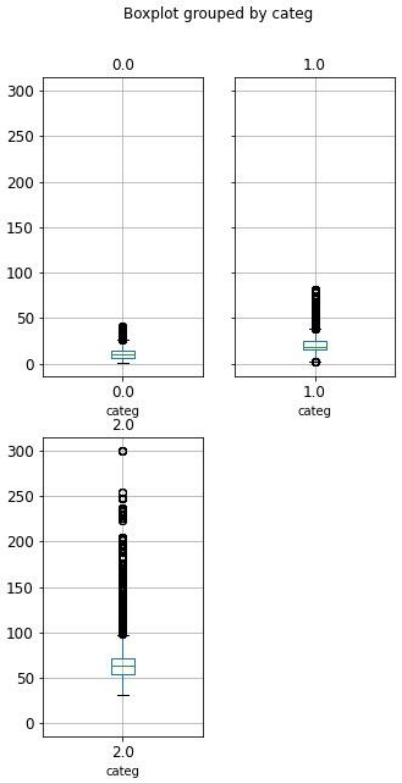
	price
categ	
0.0	2.230786e+06
1.0	2.247384e+06
2.0	1.319471e+06

'median by categ:'

	price
categ	
0.0	9.99
1.0	19.08
2.0	62.83

```
]: plot = sumunicateg.plot.pie(subplots=True,  figsize=(10, 6))
   plt.ylabel("Montant total des ventes par catégorie")
   plt.savefig('sum_uni_categ_pie.jpg')
```

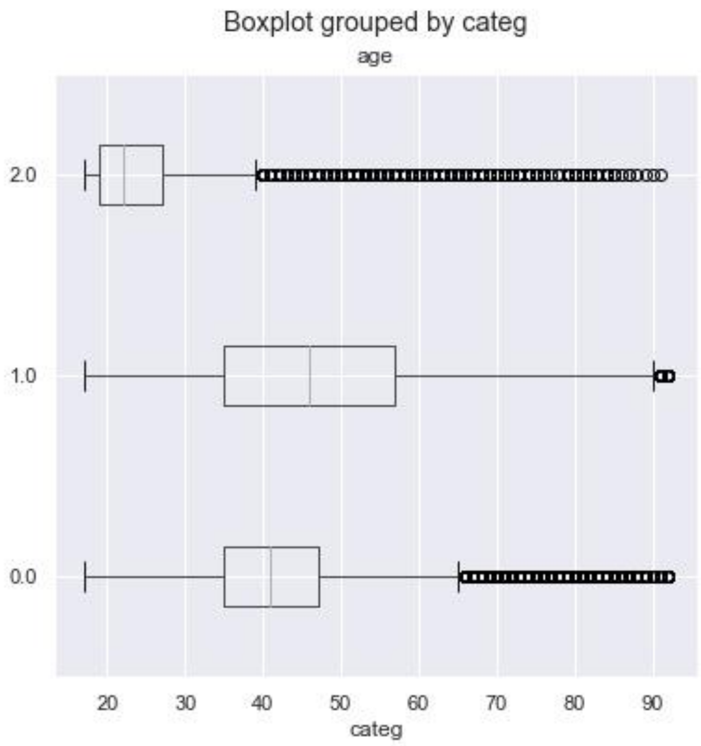




- **Categ 2 apporte relativement moins de CA par rapport à categ 0 et 1;**
- **Prix de produits (moyenne et mediane) pour categ 0 augmentent en fonction de catégorie (0 - 1 - 2)**

Univariée - âge vs categ

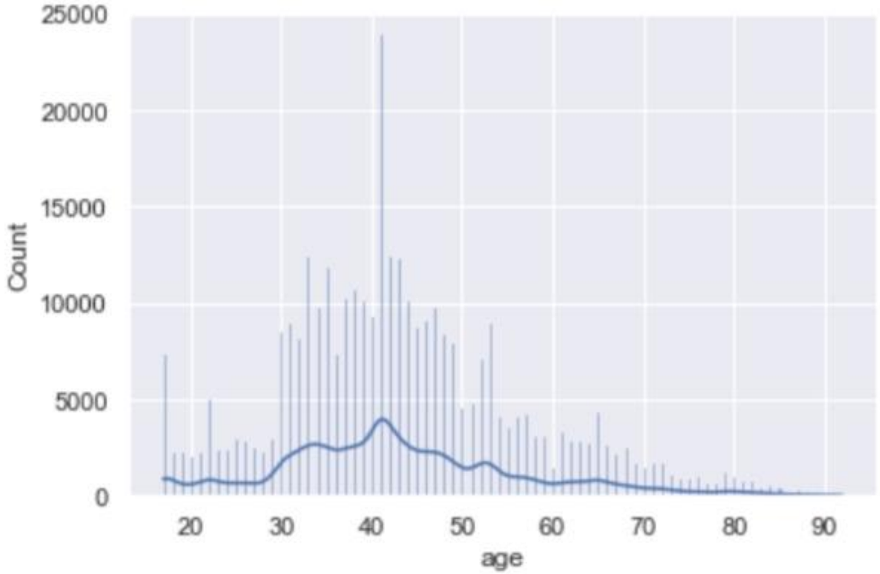
```
trans_prod_cus.boxplot(column = 'age', by = 'categ', vert= False,  figsize=(6,6))
```



Outliers des âges ne sont pas des valeurs mauvaises, vu qu'en réalité les produits sont accessibles par la population de tout âge.

Univariée - âge - histogramme

```
sns.set(style="darkgrid")
sns.histplot(data=trans_prod_cus, x="age",  bins=300, kde = True)
plt.show()
plt.savefig('uni_age_hist_kde.jpg')
```



Densité de l'âge:
La plupart des clients sont de tranche d'âge 30 - 55

Analyse bivariée

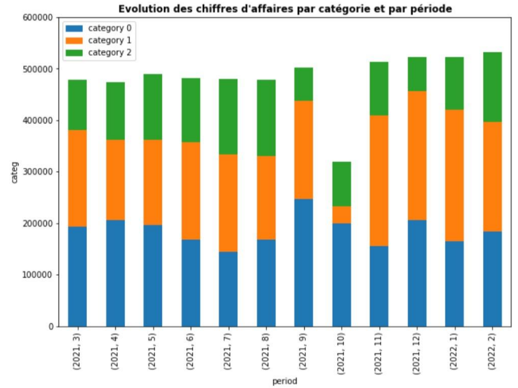
- Age VS taille du panier moyen (nombre d’articles par session)
- Age VS Fréquence d’achat (nombre d'achats)
- Catégorie VS Sexe
- Age VS Montant d'achat
- Catégorie VS Montant d’achat
- Age VS Catégorie
- Age VS Montant d’achat

Evolution CA au fil du temps _série temporelle

```
periode=pd.pivot_table(trans_prod_cus, index=['year', 'month'], columns=['categ'], values=['price'], aggfunc='sum')
periode.head()
```

Out[34]:

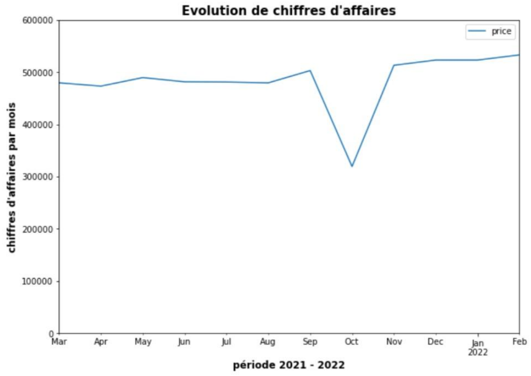
		price		
	categ	**0.0**	**1.0**	**2.0**
year	**month**			
2021	**3**	193659.26	186974.17	98771.48
	4	205304.15	156138.35	111682.70
	5	196197.52	165893.40	127359.59
	6	167958.58	189162.04	124209.56
	7	144753.20	188523.27	147663.47



In [37]:

```
plt.figure();
periode.plot.bar(figsize=(10,7), ylim= (0, 600000), stacked= True);
plt.xlabel('period');
plt.ylabel('categ') ;

colors = {'category 0':'C0', 'category 1':'C1','category 2':'C2'}
labels = list(colors.keys())
handles = [plt.Rectangle((0,0),1,1, color=colors[label]) for label in labels]
plt.legend(handles, labels)
plt.title("Evolution des chiffres d'affaires par catégorie et par période", fontsize= 12, fontweight='bold')
plt.show()
plt.savefig('Evolution des CA.png')
```

```
<Figure size 432x288 with 0 Axes>
```

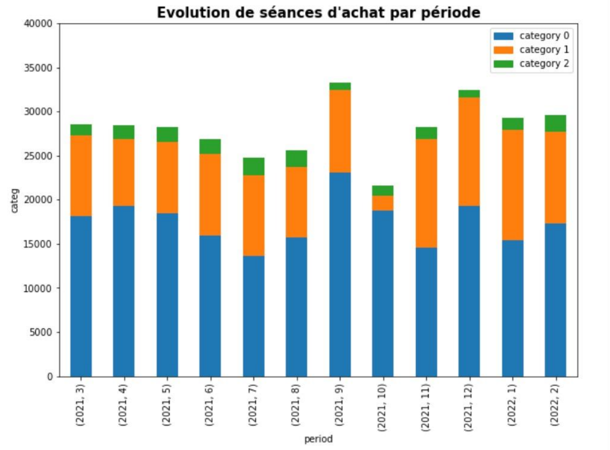


Evolution de séances au fil du temps_série temporelle

```
periode2=pd.pivot_table(trans_prod_cus, index=['year', 'month'], columns=['categ'], values=['session_id'], aggfunc='count')
periode2.head()
print(periode2.shape)
periode2.head()
```

(12, 3)

		session_id		
	categ	0.0	1.0	2.0
year	month			
2021	3	18119	9134	1315
	4	19335	7579	1501
	5	18485	8107	1653
	6	15886	9264	1669
	7	13569	9169	1978

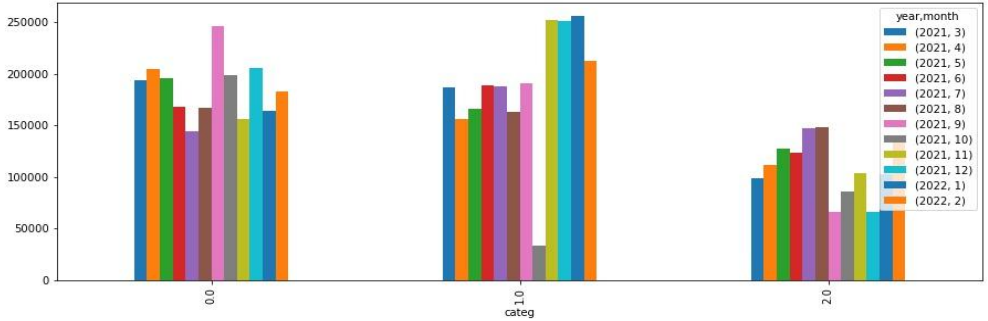


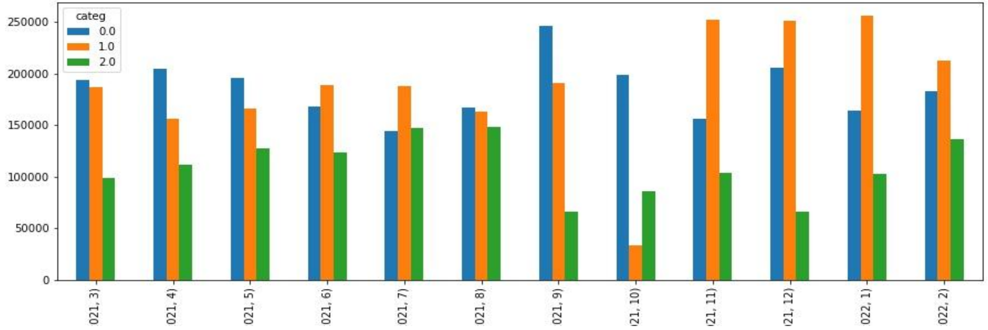
A remarquer une anomalie:
Les <u>chiffres d'affaires</u> et les <u>séances d'achat</u> pour les produits de **catégorie 1** en **octobre** 2021 sont tous relativement moins par rapport aux mois précédents et suivants

Que s’est-il passé avec les données d’octobre 2021?

```
month_pvt = trans_prod_cus.pivot_table(columns = ['year','month'], index = ['categ'], values= "price", aggfunc='sum')
month_pvt.head()
```

year	2021										2022
month	3	4	5	6	7	8	9	10	11	12	1
categ											
0.0	193659.26	205304.15	196197.52	167958.58	144753.20	167770.70	246388.05	199290.18	155946.98	206048.68	164214.
1.0	186974.17	156138.35	165893.40	189162.04	188523.27	162991.38	190613.78	$\boxed{33762.32}$	252910.39	251026.75	256267.
2.0	98771.48	111682.70	127359.59	124209.56	147663.47	148635.99	65893.29	86179.70	104136.00	65934.49	102524.



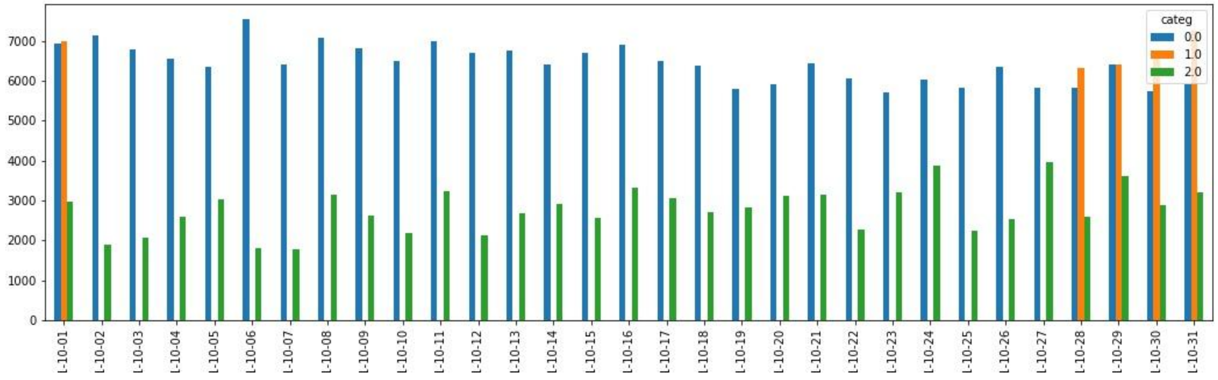


Données du mois octobre 2021

In [76]:
```python
month10_pvt = month10.pivot_table(index = ['year-month-day'], columns = ['categ'], values= "price", aggfunc='sum')
print(month10_pvt.sum())
print(month10_pvt.head())

month10_pvt.plot(kind="bar",
            figsize=(18,5),
            stacked= False)

plt.savefig("month10.jpg")
```

```
categ
0.0    199290.18
1.0     33762.32
2.0     86179.70
dtype: float64
```

categ	0.0	1.0	2.0
year-month-day			
2021-10-01	6947.51	7003.79	2958.06
2021-10-02	7138.02	NaN	1895.13
2021-10-03	6783.58	NaN	2060.49
2021-10-04	6551.25	NaN	2600.09
2021-10-05	6357.91	NaN	3032.55
2021-10-06	7543.59	NaN	1798.12
2021-10-07	6404.01	NaN	1787.07
2021-10-08	7069.53	NaN	3137.82
2021-10-09	6808.69	NaN	2616.67
2021-10-10	6487.99	NaN	2188.68
2021-10-11	7005.40	NaN	3225.16
2021-10-12	6703.98	NaN	2118.19

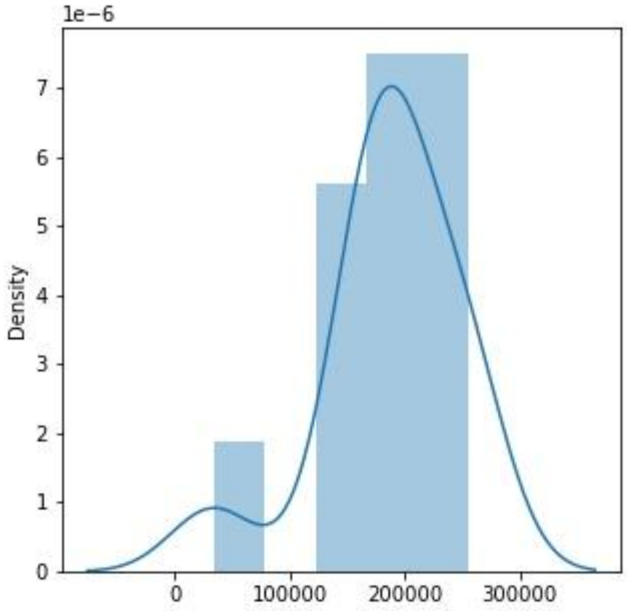


Données manquantes sur categ 1 pour la période de 02 à 27 oct 2021

Chercher la normalité: données de categ 1 de tous les mois

In [69]:
```
categ1 = month_pvt2[1].values
print(categ1)

import statistics
print('                                    ')
print('Statistics for products of category 1:       ')
print("mean:", statistics.mean(categ1))
print("median:", statistics.median(categ1))
print("mode:", statistics.mode(categ1))

# statistics (mean, median, mode) by month
```

```
[186974.16999999 156138.35      165893.4       189162.03999999
 188523.26999999 162991.38      190613.77999999  33762.32
 252910.38999998 251026.74999998 256267.91999998 213120.63999999]

Statistics for products of category 1:
mean: 187282.03416665728
median: 188842.65499999246
mode: 186974.16999999323
```



 # tendance centrale: distribution unimodale et fortement centré autour de 188,000

Quelle valeur à imputer pour oct 2021?

In [31]:
```
month_pvt2 = trans_prod_cus.pivot_table(columns = ['categ'], index = ['year', 'month'], values= "price", aggfunc= 'sum' )
month_pvt2

# price amount subtotal of products in every categ / distribution de chiffres d'affaires par catégorie de produit -- bar chart
```

Out[31]:

	categ	0.0	1.0	2.0
year	month			
2021	3	193659.26	186974.17	98771.48
	4	205304.15	156138.35	111682.70
	5	196197.52	165893.40	127359.59
	6	167958.58	189162.04	124209.56
	7	144753.20	188523.27	147663.47
	8	167770.70	162991.38	148635.99
	9	246388.05	190613.78	65893.29
	10	199290.18	33762.32	86179.70
	11	155946.98	252910.39	104136.00
	12	206048.68	251026.75	65934.49
2022	1	164214.27	256267.92	102524.72
	2	183254.04	213120.64	136479.72

moyenne de prix de categ1 au courant de 12 mois	**187282**
total de prix de categ1(1 oct + 28-31 oct)	**33762**
moyenne de prix pour 02-27 oct	=(187282 - 33762)/26 = **5905**

=> dans la table croisée de month10_pvt, il reste à imputer les manquantes de categ1 oct par **5905**

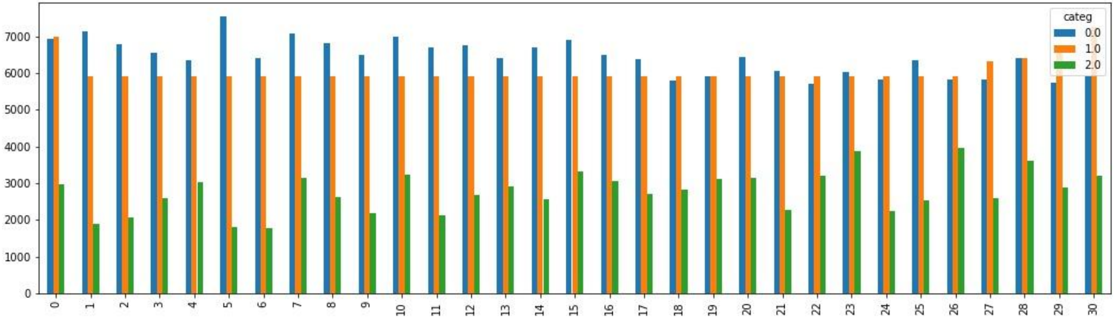
Imputation des données manquantes oct 2021

```
month10_pvt.reset_index(inplace = True)
month10_pvt[1].fillna(5905, inplace= True)
print(month10_pvt.sum())
month10_pvt.head()
```

```
categ
0.0    199290.18
1.0    187292.32
2.0     86179.70
dtype: float64
```

75]:

categ	0.0	1.0	2.0
year-month-day			
2021-10-01	6947.51	7003.79	2958.06
2021-10-02	7138.02	5905.00	1895.13
2021-10-03	6783.58	5905.00	2060.49
2021-10-04	6551.25	5905.00	2600.09
2021-10-05	6357.91	5905.00	3032.55



Data octobre après imputation

Analyse bivariée -
Âge vs nombre d’articles par séance (taille du panier moyen)

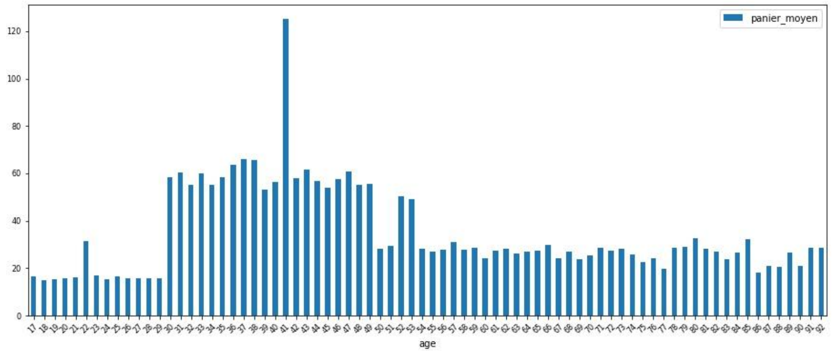
In [78]:
```
age_panier = trans_prod_cus[['age', 'id_prod']].groupby(by=['age']).count()
age_panier.reset_index(inplace =True)
#age_panier.head()

# how many products brought by clients of same age
```

In [79]:
```
customers['birth'] = pd.to_datetime(customers['birth'], format = '%Y', errors ='coerce')
customers['age'] = customers['birth'].apply(calculate_age)

age_pop =customers[['age', 'client_id']].groupby('age').count()
age_pop.reset_index(inplace =True)
#age_pop.head()

# pop of same age
```

In [80]:
```
taille_panier = pd.merge(age_panier, age_pop, on = 'age')
taille_panier['panier_moyen'] = round(taille_panier['id_prod']/taille_panier['client_id'], 2
taille_panier.head()
```

Out[80]:

	age	id_prod	client_id	panier_moyen
0	17	7348	440	16.70
1	18	2182	146	14.95
2	19	2224	146	15.23
3	20	2032	129	15.75
4	21	2175	136	15.99



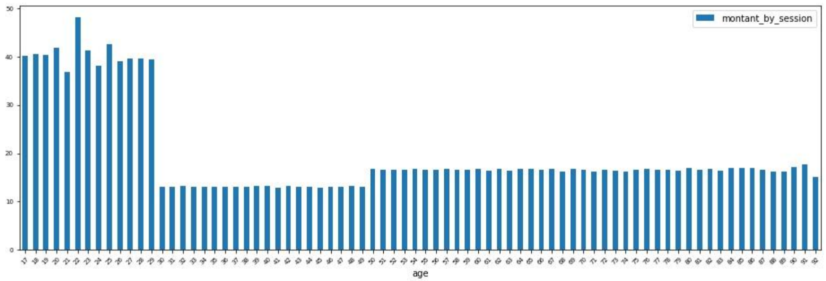
Analyse bivariée - Âge vs montant d'achats par séance

In [96]:

```
age_session = trans_prod_cus[['age', 'session_id']].groupby(by=['age']).count()
age_session= age_session.rename(columns= {'session_id': 'total_session' })
age_session.head()

age_price = trans_prod_cus[['age', 'price']].groupby(by=['age']).sum()
age_price= age_price.rename(columns= {'price': 'total_montant' })
age_price.head()

age_montant = age_session.merge(age_price, how='left', on='age').merge(age_pop, on = 'age', how= 'left')

age_montant['montant_by_session']=round(age_montant['total_montant']/age_montant['total_session'], 2)
age_montant['mean_session'] = round(age_montant['total_session']/age_montant['client_id'],2)

age_montant.head()
```

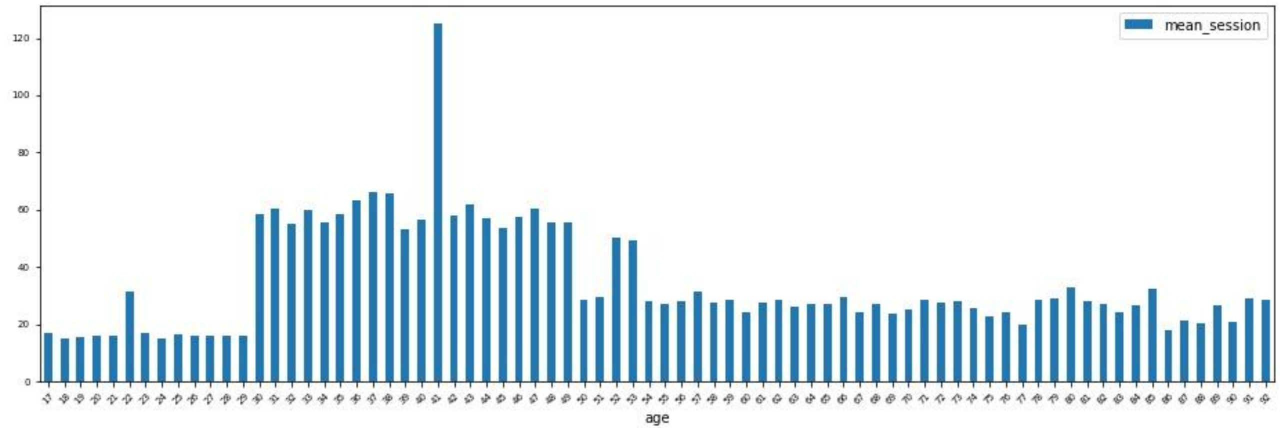
Out[96]:

	age	total_session	total_montant	client_id	montant_by_session	mean_session
0	17	7348	295387.98	440	40.20	16.70
1	18	2182	88461.39	146	40.54	14.95
2	19	2224	89920.34	146	40.43	15.23
3	20	2032	84881.48	129	41.77	15.75
4	21	2175	80110.24	136	36.83	15.99



montant d’achats par séance
(montant by session)

Analyse bivariée - Âge vs fréquence d'achats



fréquence d’achats (mean session)

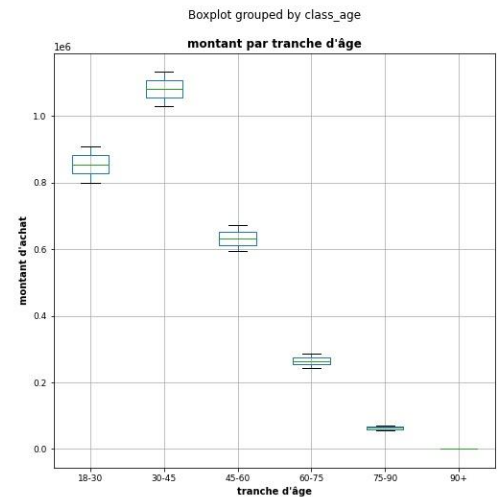
=> Clients de l’âge 30 à 53 connaissent des fréquences les plus élevées

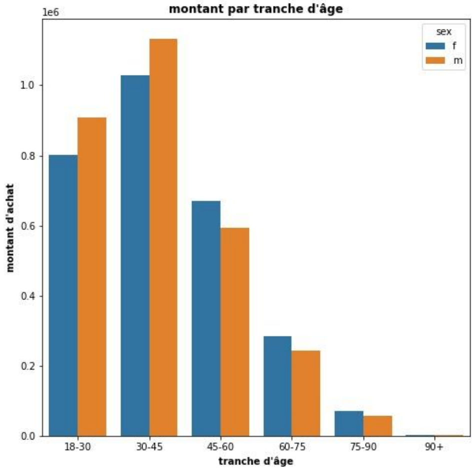
Analyse bivariée - montant d'achat vs âge et sexe

```
def class_age(x):
    if x <= 30:
        return '18-30'

    elif x > 30 and x <= 45:
        return '30-45'

    elif x > 45 and x <= 60:
        return '45-60'

    elif x > 60 and x <= 75:
        return '60-75'

    elif x > 75 and x <= 90:
        return '75-90'

    elif x > 90:
        return '90+'

trans_prod_cus['class_age'] = trans_prod_cus['age'].apply(lambda x: class_age(x))
agesex_pvt = trans_prod_cus.pivot_table(index = ['class_age', 'sex'],  values= "price"
agesex_pvt.head()
```

Out[121]:

		price
class_age	sex	
18-30	f	800863.59
	m	908630.70
30-45	f	1029260.12
	m	1132994.96
45-60	f	671398.73



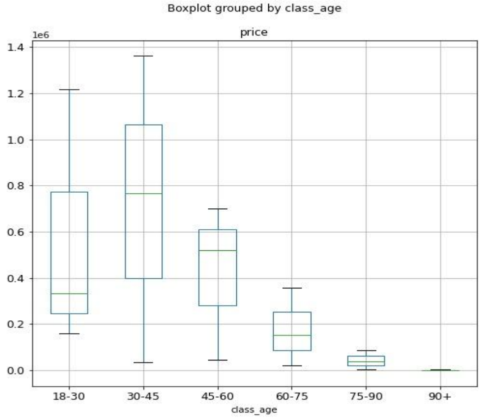


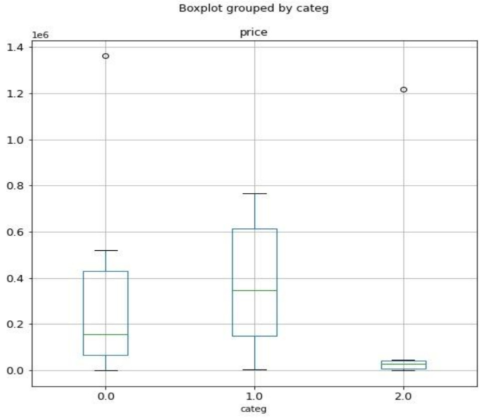
Analyse bivariée -
Âge vs montant, categ vs montant, Âge vs montant by categ

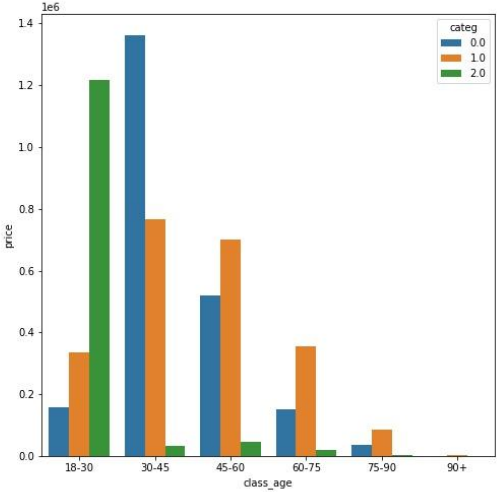
```
trans_prod_cus['class_age'] = trans_prod_cus['age'].apply(lambda x: class_age(x))

agecateg_pvt = trans_prod_cus.pivot_table(index = ['class_age', 'categ'],   values= 'price', aggfunc=sum)
agecateg_pvt.head()
```

[13]:

		price
class_age	**categ**	
18-30	**0.0**	1.594341e+05
	1.0	3.342012e+05
	2.0	1.215859e+06
30-45	**0.0**	1.362260e+06





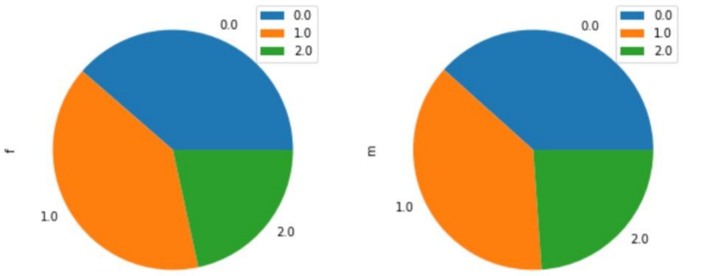


Analyse bivariée - sexe vs categ

In [100]:
```
categ_sex = trans_prod_cus.pivot_table(index = ['categ'], columns = ['sex'], values= "price", aggfunc=sum)
categ_sex
```
Out[100]:

sex	f	m
categ		
0.0	1104085.17	1126700.44
1.0	1137856.49	1109527.92
2.0	617799.31	701671.40

In [102]:
```
plot = categ_sex.plot.pie(subplots=True,  figsize=(10, 6))
plt.savefig('categsex.jpg')
```

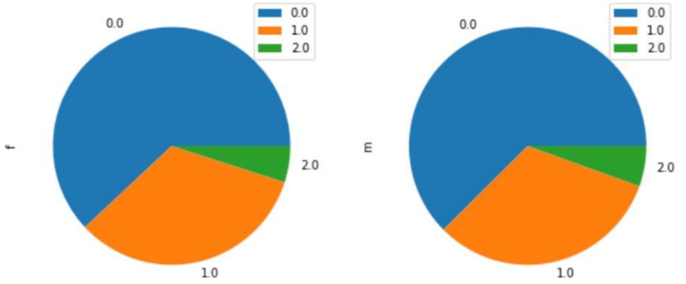


Par le montant d'achat

In [103]:
```
categ_sex2 = trans_prod_cus.pivot_table(index = ['categ'], columns = ['sex'], values= "id_prod", aggfunc='count')
categ_sex2
```
Out[103]:

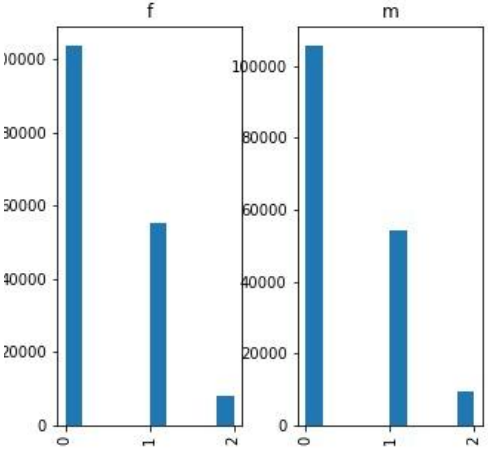
sex	f	m
categ		
0.0	103846	105683
1.0	55469	54266
2.0	8260	9292

In [104]:
```
plot = categ_sex2.plot.pie(subplots=True, figsize=(10, 6))
plt.savefig('categsex2.jpg')
```

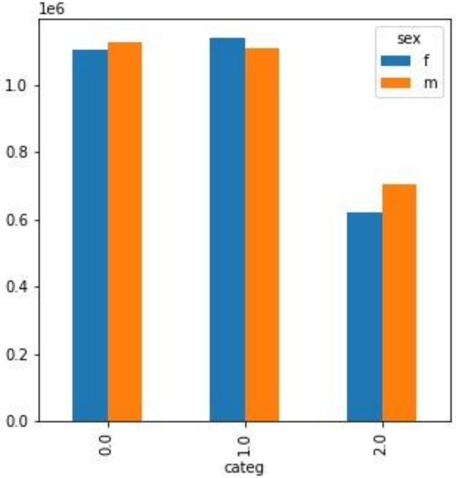


Par le nombres de produits achetés (quantité d'achat)

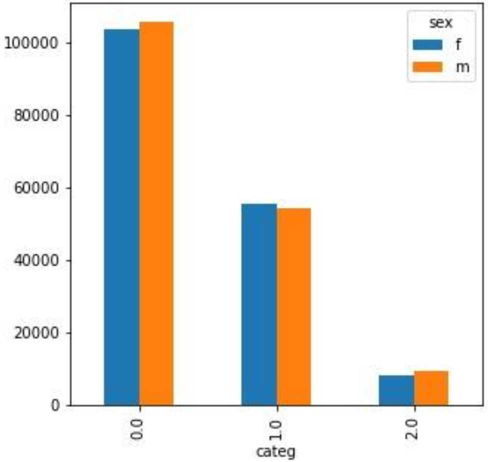
Analyse bivariée - sex vs categ



Par montant d’achat



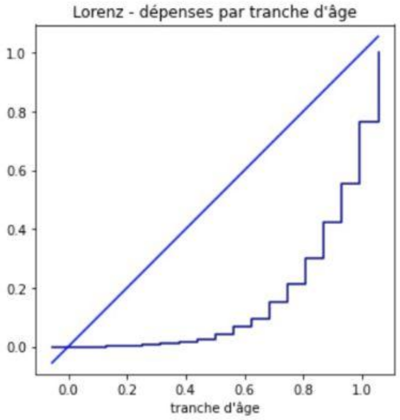
Par nombre de produits achetés
(quantité d’achat)



Courbe Lorenz et indice Gini

- concentration des **dépenses** en fonction de **tranche d'âge**

In [134]:
```
depenses = agecateg_pvt['price'].values      #sous-totaux des dépenses par age (all people of same age) et par categ

n = len(depenses)
lorenz = np.cumsum(np.sort(depenses)) / depenses.sum()
lorenz = np.append([0], lorenz)    # La courbe de Lorenz commence à 0

fig, ax = plt.subplots(figsize=[5,5])
plt.axes().axis('equal')
xaxis = np.linspace(0-1/n,1+1/n, n+1)   #Il y a un segment de taille n pour chaque individu, plus 1 segment supplémentaire d'ordonnée 0. Le premier segment
plt.xlabel("tranche d'âge")
plt.title("Lorenz - dépenses par tranche d'âge")
plt.plot(xaxis, lorenz, drawstyle='steps-post', color = "darkblue" )
plt.plot(xaxis, xaxis, color = "blue")

plt.savefig("lorenz_courbe.jpg")
```




```
AUC = (lorenz.sum() -lorenz[-1]/2 -lorenz[0]/2)/n      # Surface sous la courbe de Lorenz.
S = 0.5 - AUC        # surface entre la première bissectrice et le courbe de Lorenz
gini = 2*S
gini
```

```
0.6480633354879031
```

indice de Gini = 64%

=> au moins 64% du montant de chiffres d'affaires de clients de tranches d’âge 30-45 et 45-60 soit 36% de toute la population clientèle

Q & A

Merci

@Xuefei ZHANG 2021