Передбачення майбутніх продажів

Надані щоденні історичні дані про продажі. Завдання - передбачити загальну кількість продукції, що продається у кожному магазині, для тестового набору. Зауважте, що список магазинів та товарів дещо змінюється щомісяця. Створення надійної моделі, яка може впоратися з такими ситуаціями, є частиною виклику.

Крок 1: Визначимо проблему та розіб'ємо на прості кроки:

1. Кінцева ціль -> Передбачення продажу товарів у кожному магазині на наступний місяць
2. Примітки -> список магазинів та товарів дещо змінюється щомісяця

Тут я використовую популярні бібліотеки для завантаження даних.

In [1]:
```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

path = "/kaggle/input/competitive-data-science-predict-future-sales/"

items = pd.read_csv(path+'/items.csv')
item_cats = pd.read_csv(path+'/item_categories.csv')
shops = pd.read_csv(path+'/shops.csv')
sales = pd.read_csv(path+'/sales_train.csv')
test = pd.read_csv(path+'/test.csv')
submission = pd.read_csv(path+'/sample_submission.csv')

print("Data set loaded successfully.")
```

```
Data set loaded successfully.
```

Крок 2: Тут я використовую декілька різних команд, щоб перелічити стовпці у фреймі даних.

- DataFrame - це структура таблиці, що містить список стовпців із даними

In [2]:
```
print(items.info())
print('Items : \n\t'+'\n\t'.join(list(items)))
print('ItemsCatagories : \n\t'+'\n\t'.join(list(item_cats.columns.value
print('Shops : \n\t'+'\n\t'.join(shops.columns.tolist()))
print('Sales : \n\t'+'\n\t'.join(sales.columns.tolist()))
## you will get above data set along with row data of sales only in rea
## based on those, Usually we have to create our training and test data
## Here they giving us and test data set where we can directly use and
print('TestSet : \n\t'+'\n\t'.join(list(test)))
print('Output : \n\t'+'\n\t'.join(list(submission)))

sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22170 entries, 0 to 22169
Data columns (total 3 columns):
item_name           22170 non-null object
item_id             22170 non-null int64
item_category_id    22170 non-null int64
dtypes: int64(2), object(1)
memory usage: 519.7+ KB
None
Items :
        item_name
        item_id
        item_category_id
ItemsCatagories :
        item_category_name
        item_category_id
Shops :
        shop_name
        shop_id
Sales :
        date
        date_block_num
        shop_id
        item_id
        item_price
        item_cnt_day
TestSet :
        ID
        shop_id
        item_id
Output :
        ID
        item_cnt_month
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2935849 entries, 0 to 2935848
Data columns (total 6 columns):
date              object
date_block_num    int64
shop_id           int64
item_id           int64
item_price        float64
item_cnt_day      float64
```

```
dtypes: float64(2), int64(3), object(1)
memory usage: 134.4+ MB
```

Тепер ми маємо чітке уявлення про те, про що ці дані. Щоб отримати більше уявлення про дані, ми спробуємо візуалізувати ці дані.

Крок 3: Візуалізація даних. Спочатку спробуйте візуалізувати деякі випадкові вибірки, витягнуті з даних використовуючи різні методи, які ми можемо використовувати для візуалізації даних табличним способом.

In [3]:
```
print("Items")
print(items.head(2))
print("\nItem Catagerios")
print(item_cats.tail(2))
print("\nShops")
print(shops.sample(n=2))
print("\nTraining Data Set")
print(sales.sample(n=3,random_state=1))
print("\nTest Data Set")
print(test.sample(n=3,random_state=1))
```

```
Items
                                                item_name  item_id  \
0          ! ВО ВЛАСТИ НАВАЖДЕНИЯ (ПЛАСТ.)         D        0
1  !ABBYY FineReader 12 Professional Edition Full...        1

   item_category_id
0                40
1                76

Item Catagerios
           item_category_name  item_category_id
82  Чистые носители (штучные)                82
83           Элементы питания                83

Shops
                      shop_name  shop_id
26  Москва ТЦ "Ареал" (Беляево)       26
31      Москва ТЦ "Семеновский"       31

Training Data Set
                date  date_block_num  shop_id  item_id  item_price  \
651498   12.07.2013              6       37    11691       149.0
460637   18.05.2013              4       25     4302       649.0
1696749  18.05.2014             16       47     1306       299.0

         item_cnt_day
651498            1.0
460637            1.0
1696749           1.0

Test Data Set
            ID  shop_id  item_id
100999  100999       19    19049
41385    41385       28      961
129419  129419       47     7878
```

Переглянувши цей набір даних, ми можемо катарагізувати ці дані на метадані та ефективні дані. Отже, назви магазинів та назви предметів нас не дуже хвилюють. У нас може бути магазин і товар, поєднані id і дані про продажі для подальшого аналізу.

Кінцева мета - передбачити продажі, тому ми можемо ігнорувати назви продуктів. нас цікавить кількість предметів у часовому ряду дат. І ціна також може бути фактором продажу.

Отже, спробуйте побудувати деякі релевантні дані. Перш ніж будувати щось, краще скласти уявлення про межі набору даних.

Як ми бачимо, простий спосіб вирішити це - використовувати дані про продажі та спробувати згрупувати та узагальнити їх. Для зручності ми розділимо стовпець дати на рік та місяць

In [4]:
```
from datetime import datetime
sales['year'] = pd.to_datetime(sales['date']).dt.strftime('%Y')
sales['month'] = sales.date.apply(lambda x: datetime.strptime(x,'%d.%m.%

sales.head(2)
```

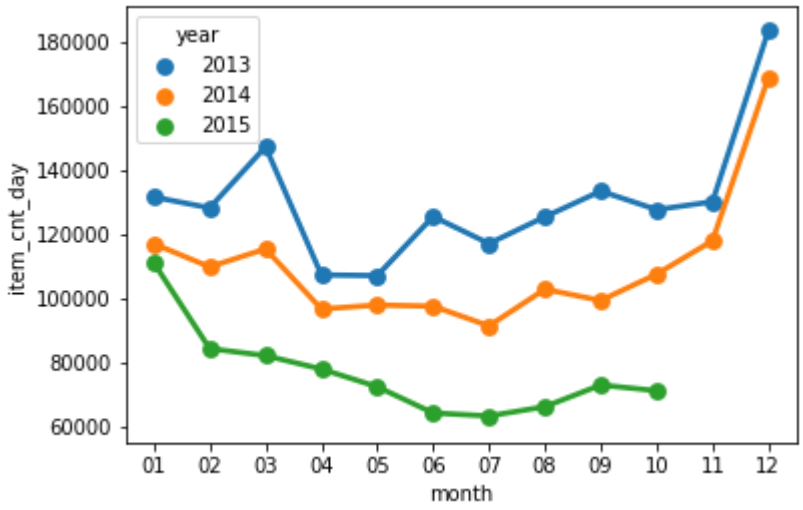
Out[4]:

	date	date_block_num	shop_id	item_id	item_price	item_cnt_day	year	month
0	02.01.2013	0	59	22154	999.0	1.0	2013	01
1	03.01.2013	0	25	2552	899.0	1.0	2013	01

Давайте спробуємо побудувати графіки продажів на кожен рік, щоб зрозуміти інформацію про сезонні дані

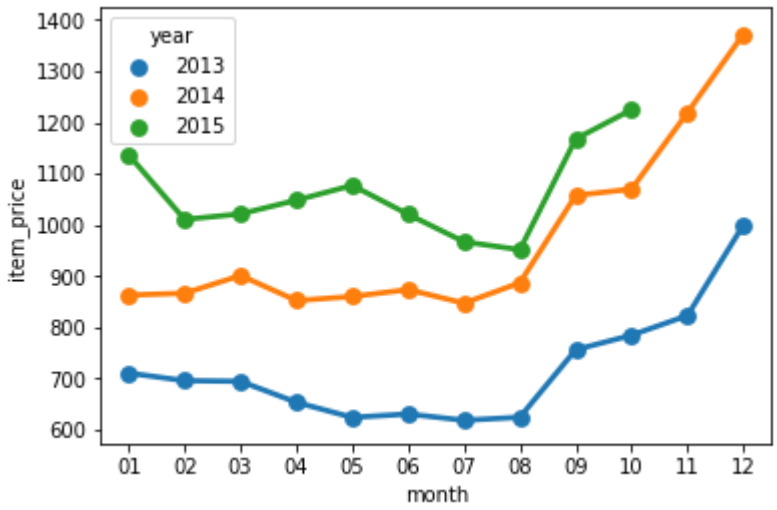
In [5]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#will make your plot outputs appear and be stored within the notebook.
%matplotlib inline

grouped = pd.DataFrame(sales.groupby(['year','month'])['item_cnt_day'].s
sns.pointplot(x='month', y='item_cnt_day', hue='year', data=grouped)
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f878fc4eb38>
```



In [6]:
```
#Price
grouped_price = pd.DataFrame(sales.groupby(['year','month'])['item_price
sns.pointplot(x='month', y='item_price', hue='year', data=grouped_price
```

Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x7f87a3d182e8>

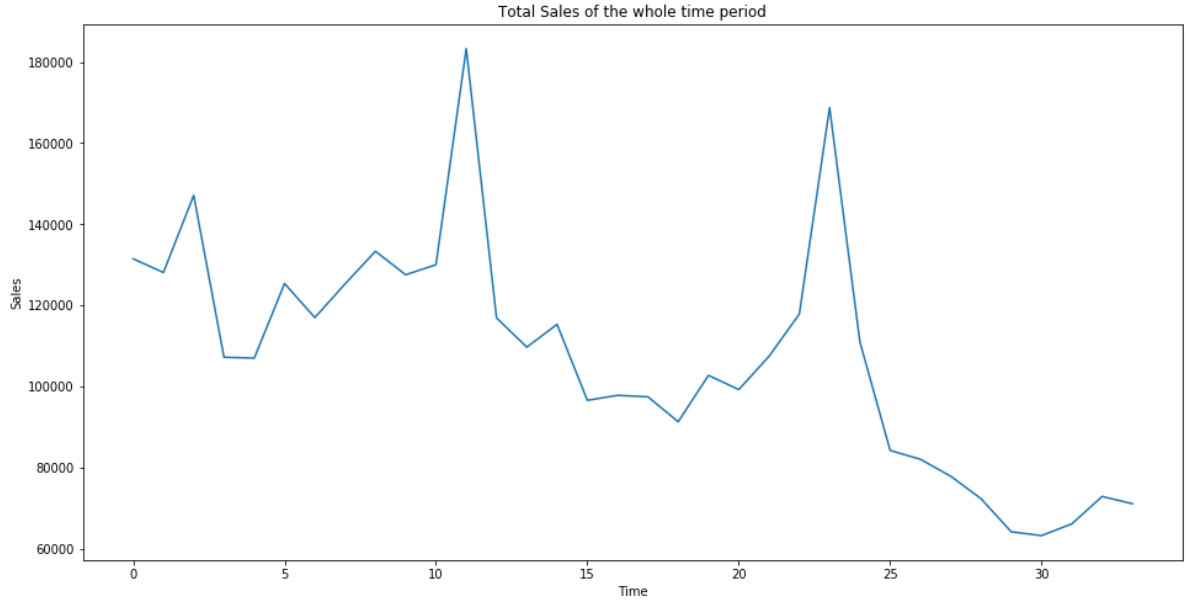


Побачивши цей графік, ми можемо це побачити

1. Останні два місяці року що маємо більше продажів.
2. В 2015 році ми очікуємо збільшення продажів.

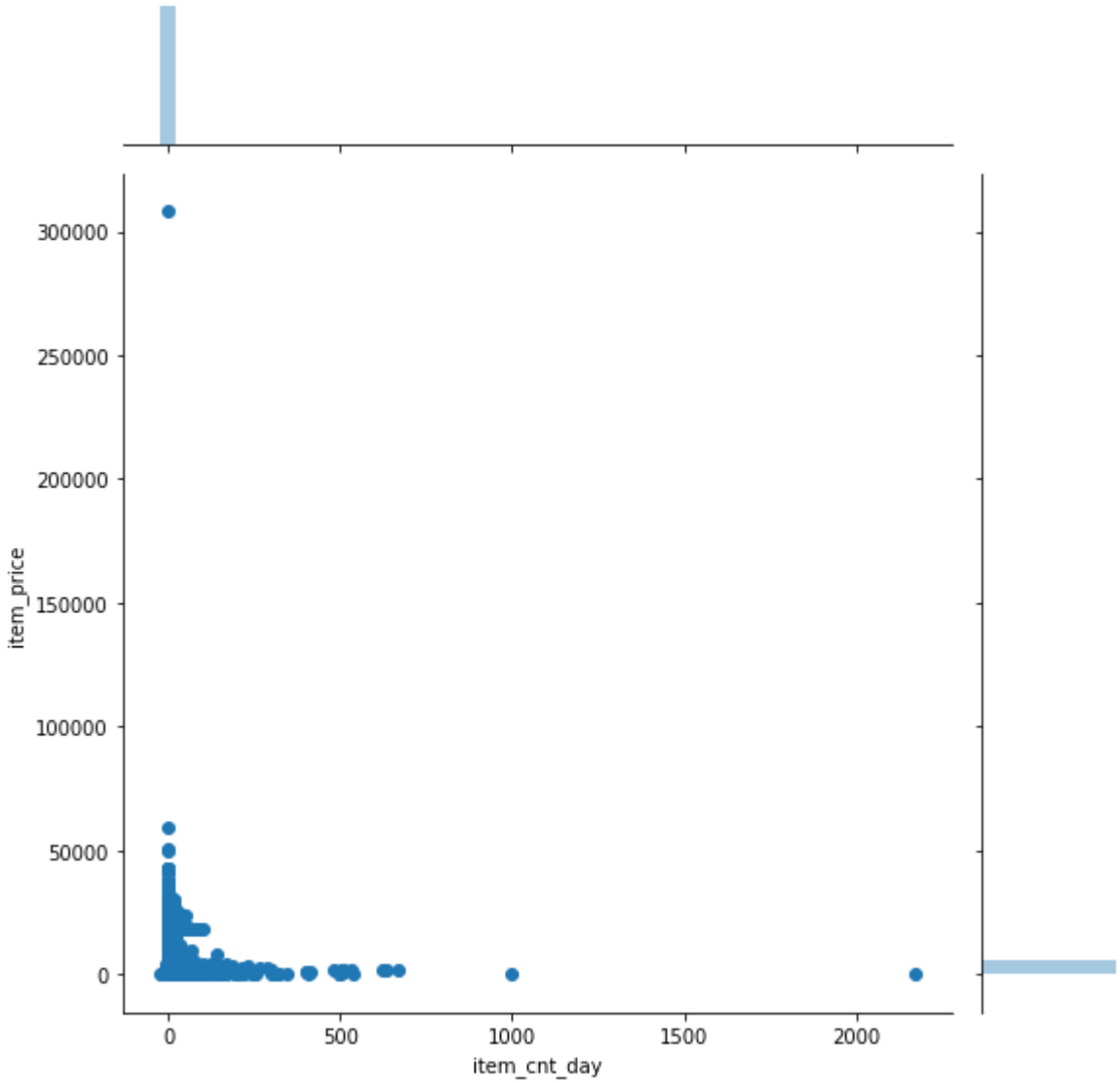
Спробуємо намалювати загальний обсяг продажів разом з лінійним періодом місяця.

In [7]:
```
ts=sales.groupby(["date_block_num"])["item_cnt_day"].sum()
ts.astype('float')
plt.figure(figsize=(16,8))
plt.title('Total Sales of the whole time period')
plt.xlabel('Time')
plt.ylabel('Sales')
plt.plot(ts);
```



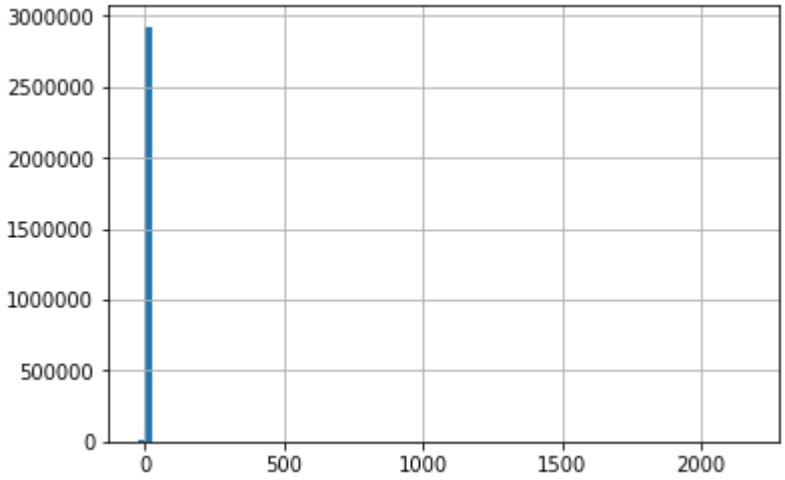
Перевіримо розподіл для знаходження разбросу

In [8]:
```
sns.jointplot(x="item_cnt_day", y="item_price", data=sales, height=8)
plt.show()
```



In [9]:
```
sales.item_cnt_day.hist(bins=100)
sales.item_cnt_day.describe()
```

Out[9]:
```
count    2.935849e+06
mean     1.242641e+00
std      2.618834e+00
min     -2.200000e+01
25%      1.000000e+00
50%      1.000000e+00
75%      1.000000e+00
max      2.169000e+03
Name: item_cnt_day, dtype: float64
```



Бачимо, що item_cnt_day"> 125 и <0, "item_price"> = 75000 мы можемо розглядати, як статистичну похибку. На етапі очистки даних мы видалимо ці елементи.

Крок 4: Очищення даних

Фільтруємо некоректні дані. Наприклад:

1. Ціна предмета дорівнює 0
2. Дані, які не вказані у тестовому наборі
3. Видалимо статистичні похибки

In [10]:
```
print('Data set size before remove item price 0 cleaning:', sales.shape
sales = sales.query('item_price > 0')
print('Data set size after remove item price 0 cleaning:', sales.shape)
```

```
Data set size before remove item price 0 cleaning: (2935849, 8)
Data set size after remove item price 0 cleaning: (2935848, 8)
```

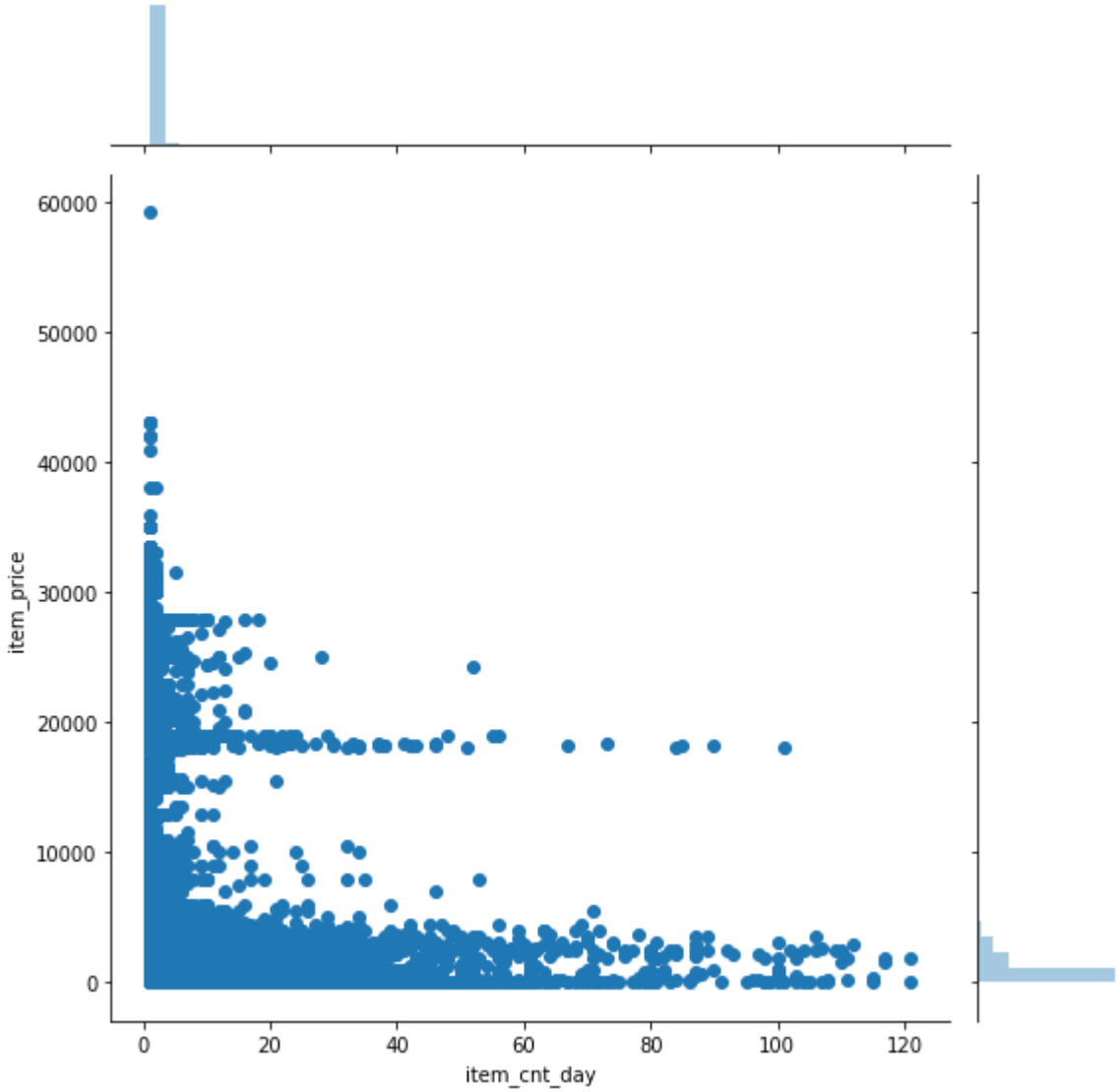
In [11]:
```
print('Data set size before filter valid:', sales.shape)
# Only shops that exist in test set.
sales = sales[sales['shop_id'].isin(test['shop_id'].unique())]
# Only items that exist in test set.
sales = sales[sales['item_id'].isin(test['item_id'].unique())]
print('Data set size after filter valid:', sales.shape)
```

```
Data set size before filter valid: (2935848, 8)
Data set size after filter valid: (1224439, 8)
```

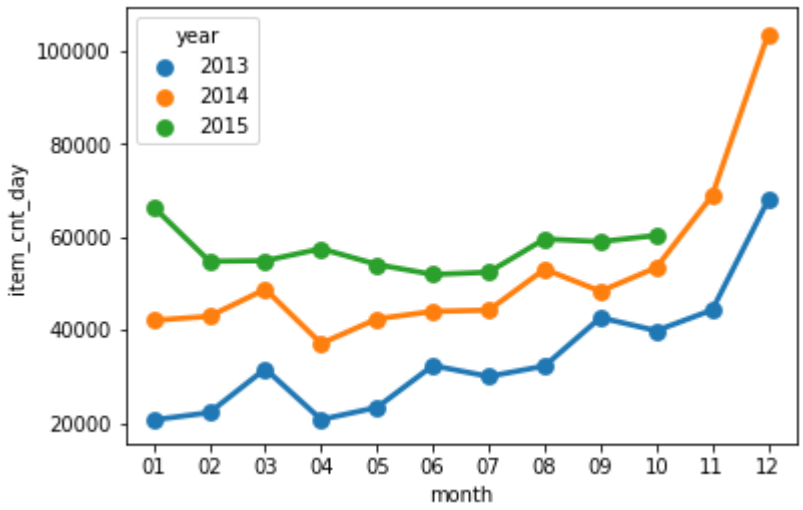
In [12]:
```
print('Data set size before remove outliers:', sales.shape)
sales = sales.query('item_cnt_day >= 0 and item_cnt_day <= 125 and item_
print('Data set size after remove outliers:', sales.shape)
```

```
Data set size before remove outliers: (1224439, 8)
Data set size after remove outliers: (1221451, 8)
```

In [13]:
```
#After cleaning plot
sns.jointplot(x="item_cnt_day", y="item_price", data=sales, height=8)
plt.show()

cleaned = pd.DataFrame(sales.groupby(['year','month'])['item_cnt_day'].s
sns.pointplot(x='month', y='item_cnt_day', hue='year', data=cleaned)
```



Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f879a5559e8>



Крок 5: Попередня обробка даних. Визначимо особливості. Це означає, що вибираємо лише необхідні функції та створюємо належний набір даних для обробки. Нам потрібно з'ясувати, які особливості впливатимуть на продажі

1. Ціна
2. Місяць
3. Рік
4. Категоряі товару

Виходячи з наведених вище функцій, продажі можуть бути різними. Отже, ми будемо тримати лише зацікавлені стовпці та відкидати інші.

In [14]:
```
# Aggregate to monthly level the sales
monthly_sales=sales.groupby(["date_block_num","shop_id","item_id"])[
    "date_block_num","date","item_price","item_cnt_day"].agg({"date_blo

monthly_sales.head(5)
```

Out[14]:

			date_block_num		date	item_price	item_cnt
			mean	min	max	mean	
date_block_num	shop_id	item_id					
0	2	33	0	05.01.2013	05.01.2013	499.0	
		482	0	16.01.2013	16.01.2013	3300.0	
		491	0	09.01.2013	09.01.2013	600.0	
		839	0	22.01.2013	22.01.2013	3300.0	
		1007	0	11.01.2013	25.01.2013	449.0	

Тренування датасету

Ми будемо використовувати алгоритм LSTM (Long Short Term Memory) для моделювання даних часового ряду. Модель LSTM засвоїть функцію, яка відображає послідовність минулих спостережень як вхід до вихідного спостереження.

Для цього підходу нам потрібно підготувати наш набір даних із послідовністю введення та виводу.

Напр .: Скажімо, у нас щомісячні середні продажі як,

[10, 20, 30, 40, 50, 60, 70, 80, 90]

Ми можемо розділити послідовність на кілька шаблонів вводу / виводу, званих зразками, де три вхідні кроки використовуються як вхідні дані, а один часовий крок використовується як вихід для однокрокового прогнозування, яке вивчається.

```
          X         y
10, 20, 30        40
20, 30, 40        50
30, 40, 50        60
```

Наш стовпець 'date_block_num' буде індексом послідовності, значення - продажі.

In [15]:
```
sales_data_flat = monthly_sales.item_cnt_day.apply(list).reset_index()
#Keep only the test data of valid
sales_data_flat = pd.merge(test,sales_data_flat,on = ['item_id','shop_i
#fill na with 0
sales_data_flat.fillna(0,inplace = True)
sales_data_flat.drop(['shop_id','item_id'],inplace = True, axis = 1)
sales_data_flat.head(20)
```

Out[15]:

	ID	date_block_num	sum
0	0	20.0	1.0
1	0	22.0	1.0
2	0	23.0	2.0
3	0	24.0	2.0
4	0	28.0	1.0
5	0	29.0	1.0
6	0	30.0	1.0
7	0	31.0	3.0
8	0	32.0	1.0
9	1	0.0	0.0
10	2	28.0	3.0
11	2	29.0	2.0
12	2	31.0	1.0
13	2	32.0	3.0
14	2	33.0	1.0
15	3	31.0	1.0
16	4	0.0	0.0
17	5	20.0	2.0
18	5	23.0	3.0
19	5	24.0	1.0

In [16]:
```
#We will create pivot table.
# Rows = each shop+item code
# Columns will be out time sequence
pivoted_sales = sales_data_flat.pivot_table(index='ID', columns='date_b
pivoted_sales.head(20)
```

Out[16]:

date_block_num	0.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	...	24.0	25.0	26.0	27.0	28.0
ID																
0	0	0	0	0	0	0	0	0	0	0	...	2	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	3
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	3
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	1	0	0
8	0	0	0	0	0	0	0	0	0	0	...	9	2	3	2	2
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	...	4	3	6	2	6
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
12	0	0	1	0	1	0	0	0	0	0	...	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	3	3	1	0	0
14	0	0	0	0	0	0	1	5	4	4	...	7	1	3	1	1
15	0	0	0	0	0	0	0	0	0	0	...	4	1	7	1	6
16	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	...	0	0	0	9	5
19	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0

20 rows × 34 columns

Крок 6 : Розподільне навчання, валідація та перевірка даних.

In [17]:
```
# X we will keep all columns execpt the last one
X_train = np.expand_dims(pivoted_sales.values[:,:-1],axis = 2)
# the last column is our prediction
y_train = pivoted_sales.values[:,-1:]

# for test we keep all the columns execpt the first one
X_test = np.expand_dims(pivoted_sales.values[:,1:],axis = 2)

# lets have a look on the shape
print(X_train.shape,y_train.shape,X_test.shape)
```

```
(214200, 33, 1) (214200, 1) (214200, 33, 1)
```

In [18]:
```
from keras.models import Sequential
from keras.layers import LSTM,Dense,Dropout
from keras.models import load_model, Model

# our defining sales model
sales_model = Sequential()
sales_model.add(LSTM(units = 64,input_shape = (33,1)))
#sales_model.add(LSTM(units = 64,activation='relu'))
sales_model.add(Dropout(0.5))
sales_model.add(Dense(1))

sales_model.compile(loss = 'mse',optimizer = 'adam', metrics = ['mean_s
sales_model.summary()
```

```
Using TensorFlow backend.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 64)                16896
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 16,961
Trainable params: 16,961
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
sales_model.fit(X_train,y_train,batch_size = 4096,epochs = 10)
```

```
Epoch 1/10
214200/214200 [==============================] - 31s 146us/step - los
s: 5.8036 - mean_squared_error: 5.8036
Epoch 2/10
214200/214200 [==============================] - 32s 151us/step - los
s: 5.5572 - mean_squared_error: 5.5572
Epoch 3/10
214200/214200 [==============================] - 31s 147us/step - los
s: 5.3642 - mean_squared_error: 5.3642
Epoch 4/10
214200/214200 [==============================] - 31s 147us/step - los
s: 5.2544 - mean_squared_error: 5.2544
Epoch 5/10
214200/214200 [==============================] - 32s 151us/step - los
s: 5.1882 - mean_squared_error: 5.1882
Epoch 6/10
214200/214200 [==============================] - 32s 148us/step - los
s: 5.1647 - mean_squared_error: 5.1647
Epoch 7/10
214200/214200 [==============================] - 33s 155us/step - los
s: 5.0580 - mean_squared_error: 5.0580
Epoch 8/10
214200/214200 [==============================] - 33s 152us/step - los
s: 5.0365 - mean_squared_error: 5.0365
Epoch 9/10
214200/214200 [==============================] - 33s 155us/step - los
s: 5.0078 - mean_squared_error: 5.0078
Epoch 10/10
214200/214200 [==============================] - 31s 147us/step - los
s: 4.9830 - mean_squared_error: 4.9830
```

Out[19]:
```
<keras.callbacks.callbacks.History at 0x7f879c218dd8>
```

In [20]:
```
submission_output = sales_model.predict(X_test)
# creating dataframe with required columns
submission = pd.DataFrame({'ID':test['ID'],'item_cnt_month':submission_
# creating csv file from dataframe
#submission.to_csv('submission.csv',index = False)
submission.to_csv('submission_stacked.csv',index = False)
submission.head()
```

Out[20]:

	ID	item_cnt_month
0	0	0.448359
1	1	0.112115
2	2	0.795173
3	3	0.164107
4	4	0.112115