In [ ]:
```
# Created by: Jess Gallo
# Created: 05/17/2022
# Last Modified: 05/17/2022
# Description: Shopify Fall 2022 Data Science Intern Challenge
# Problem:
```

In [1]:
```
# Imports
import pandas as pd
from matplotlib import pyplot
import numpy as np
import seaborn as sns
```

In [2]:
```
# CSV
filename = "./Downloads/2019 Winter Data Science Intern Challenge Data Set - Shee
names = ['order_id', 'shop_id', 'user_id', 'order_amount', 'total_items', 'paymer
dataSetCsv = pd.read_csv(filename)
dataset = pd.DataFrame(dataSetCsv)
```

In [3]:
```
dataset.head(10)
```

Out[3]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
0	1	53	746	224	2	cash	2017-03-13 12:36:56
1	2	92	925	90	1	cash	2017-03-03 17:38:52
2	3	44	861	144	1	cash	2017-03-14 4:23:56
3	4	18	935	156	1	credit_card	2017-03-26 12:43:37
4	5	18	883	156	1	credit_card	2017-03-01 4:35:11
5	6	58	882	138	1	credit_card	2017-03-14 15:25:01
6	7	87	915	149	1	cash	2017-03-01 21:37:57
7	8	22	761	292	2	cash	2017-03-08 2:05:38
8	9	64	914	266	2	debit	2017-03-17 20:56:50
9	10	52	788	146	1	credit_card	2017-03-30 21:08:26

In [4]:
```
# Any NULL Values?
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   order_id        5000 non-null   int64 
 1   shop_id         5000 non-null   int64 
 2   user_id         5000 non-null   int64 
 3   order_amount    5000 non-null   int64 
 4   total_items     5000 non-null   int64 
 5   payment_method  5000 non-null   object
 6   created_at      5000 non-null   object
dtypes: int64(5), object(2)
memory usage: 273.6+ KB
```

In [ ]:
```
# It seems that the original way the average order value (AOV) was calculated
# was by taking the mean of the order_amount column, which gave the very high
# amount
```

In [ ]:
```
# Understanding What We Need

# The AOV is an average dollar spend when a customer places an order. Knowing
# this amount helps a business understand their marketing and pricing
# strategies

# To calculate AOV: AOV = total_revenue/number_of_orders
```

In [5]:
```
AOV = dataset['order_amount'].mean()
print('The original Average Order Values is: ', AOV)
```

```
The original Average Order Values is:  3145.128
```

In [ ]:
```
# a) Think about what could be going wrong with our calculation. Think about
#    a better way to evaluate this data.
```

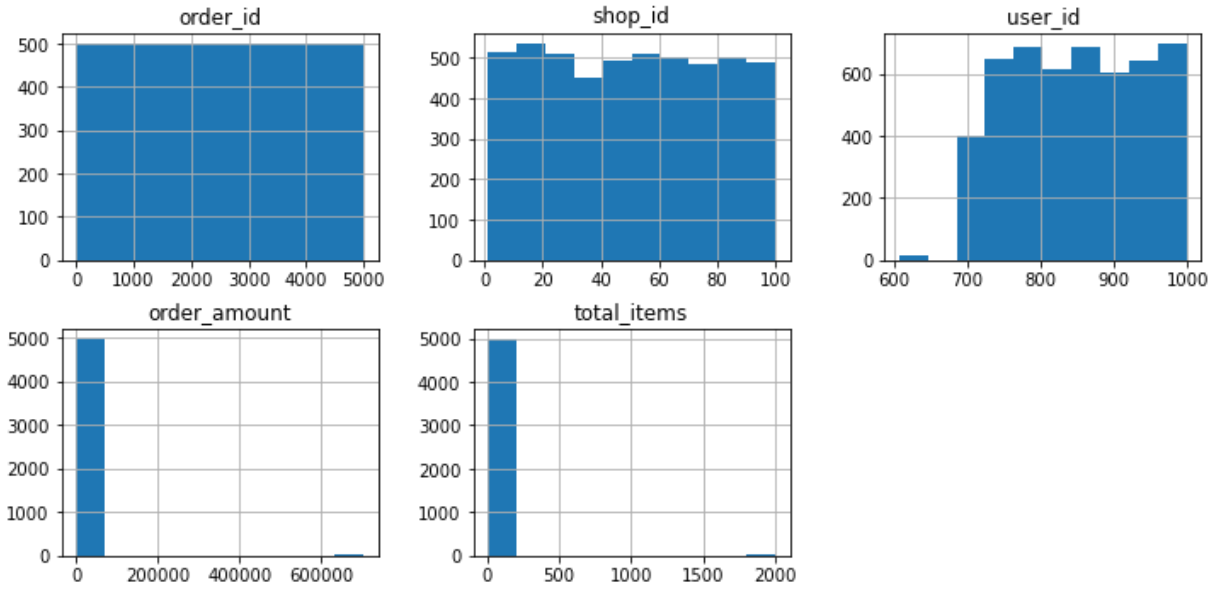
In [6]:
```
dataset.describe()
```

Out[6]:

	order_id	shop_id	user_id	order_amount	total_items
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.00000
mean	2500.500000	50.078800	849.092400	3145.128000	8.78720
std	1443.520003	29.006118	87.798982	41282.539349	116.32032
min	1.000000	1.000000	607.000000	90.000000	1.00000
25%	1250.750000	24.000000	775.000000	163.000000	1.00000
50%	2500.500000	50.000000	849.000000	284.000000	2.00000
75%	3750.250000	75.000000	925.000000	390.000000	3.00000
max	5000.000000	100.000000	999.000000	704000.000000	2000.00000

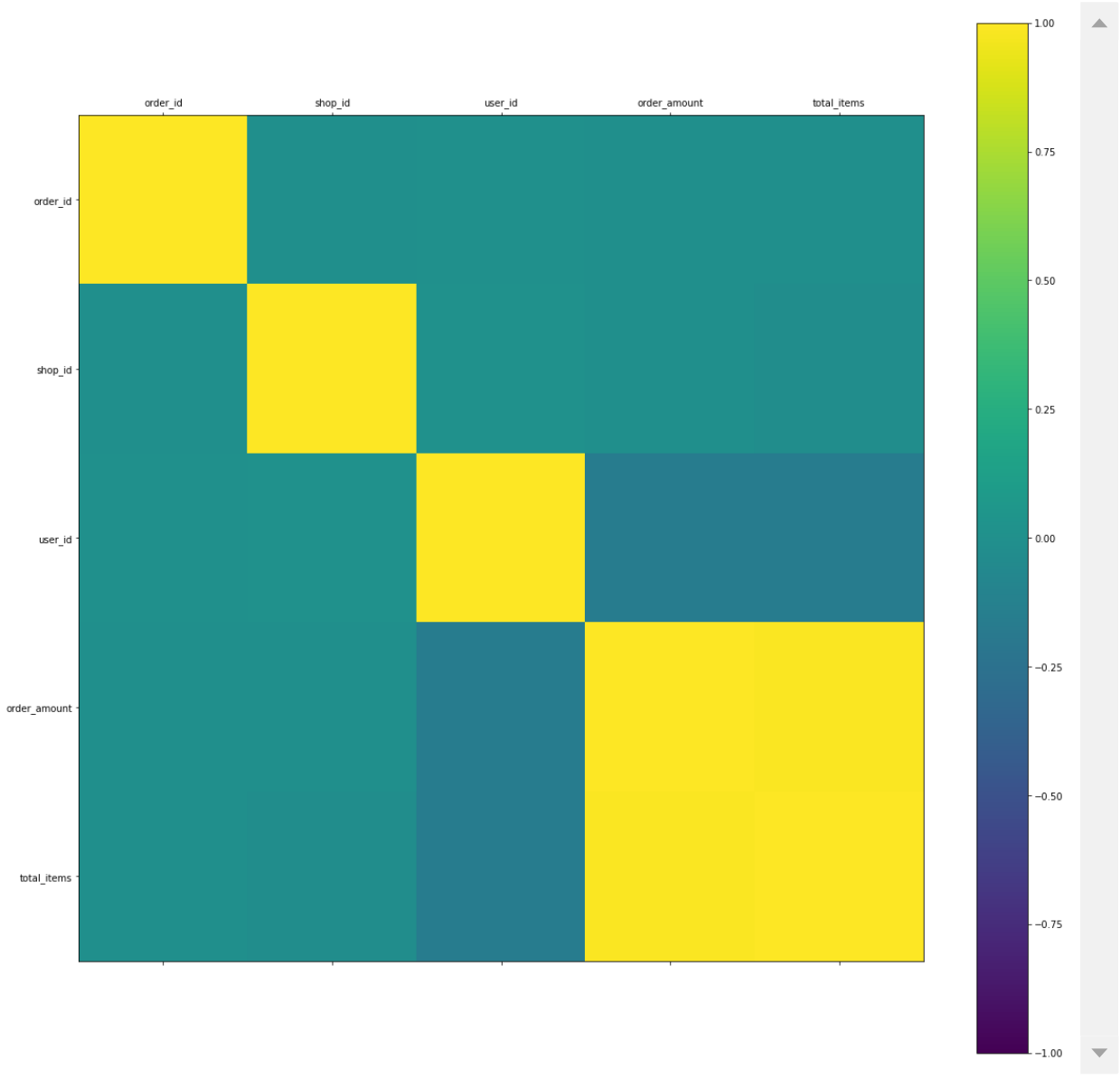
In [ ]:
```
# We need to focus on the order_amount column, since that is what is used
# to calculate the AOV. As you can see from the table above, the data seems
# a bit unbalanced since the min value is 90, 25% of orders are below 163,
# 50% of orders are below 284, and 75% of orders are below 390, which are all
# normal, but as you can see, the max value for orders is 704000, which is
# very high. On further inspection of the table, if we look at total_items,
# we can see that the max total items is 2000, which means there are large
# purchase orders here. having those large orders is imbalancing the dataset
# and not allowing for an accurate mean of the order amount
```

In [7]:
```
# histogram
dataset.hist(layout= (4,3),figsize=(12,12))
pyplot.show()
```



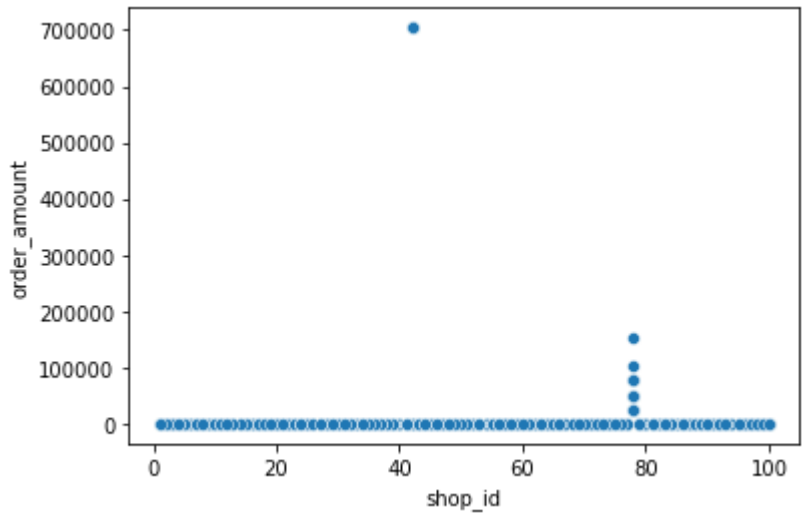
In [8]:
```
# ------------------------
# Correlation Matrix Plot |
# ------------------------
names = ['order_id', 'shop_id', 'user_id', 'order_amount', 'total_items']
correlations = dataset.corr()
fig = pyplot.figure(figsize=(20, 20))
ax = fig.add_subplot(111)  # subplot grid parameters 1x1 grid, 1st subplot
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)  # creates colorbar on axes
ticks = np.arange(0,5,1)  # returns evenly spaced values within a given interval
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
print("\nCorrelation Matrix:")
pyplot.show()  # displays plot
```

Correlation Matrix:



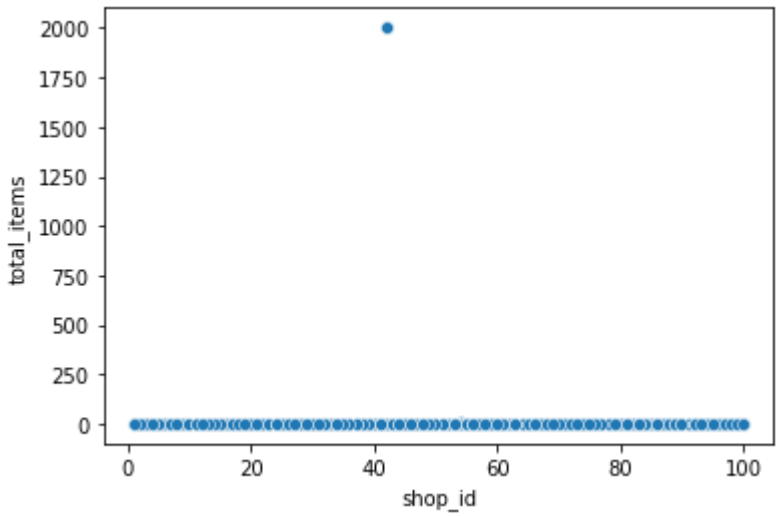
In [9]:
```
sns.scatterplot(x="shop_id", y="order_amount",data=dataset)
```

Out[9]: <AxesSubplot:xlabel='shop_id', ylabel='order_amount'>



In [10]:
```
sns.scatterplot(x="shop_id", y="total_items",data=dataset)
```

Out[10]: <AxesSubplot:xlabel='shop_id', ylabel='total_items'>



In [ ]:
```
# It seems that a particular store has the largest order amounts and total
# items
```

In [11]:
```
dataset.nlargest(20, 'order_amount', keep='all')
# This shows us the largest order_amounts are coming from the stores
# 42 and 78
```

Out[11]:

	order_id	shop_id	user_id	order_amount	total_items	payment_method	created_at
15	16	42	607	704000	2000	credit_card	2017-03-07 4:00:00
60	61	42	607	704000	2000	credit_card	2017-03-04 4:00:00
520	521	42	607	704000	2000	credit_card	2017-03-02 4:00:00
1104	1105	42	607	704000	2000	credit_card	2017-03-24 4:00:00
1362	1363	42	607	704000	2000	credit_card	2017-03-15 4:00:00
1436	1437	42	607	704000	2000	credit_card	2017-03-11 4:00:00
1562	1563	42	607	704000	2000	credit_card	2017-03-19 4:00:00
1602	1603	42	607	704000	2000	credit_card	2017-03-17 4:00:00
2153	2154	42	607	704000	2000	credit_card	2017-03-12 4:00:00
2297	2298	42	607	704000	2000	credit_card	2017-03-07 4:00:00
2835	2836	42	607	704000	2000	credit_card	2017-03-28 4:00:00
2969	2970	42	607	704000	2000	credit_card	2017-03-28 4:00:00
3332	3333	42	607	704000	2000	credit_card	2017-03-24 4:00:00
4056	4057	42	607	704000	2000	credit_card	2017-03-28 4:00:00
4646	4647	42	607	704000	2000	credit_card	2017-03-02 4:00:00
4868	4869	42	607	704000	2000	credit_card	2017-03-22 4:00:00
4882	4883	42	607	704000	2000	credit_card	2017-03-25 4:00:00
691	692	78	878	154350	6	debit	2017-03-27 22:51:43
2492	2493	78	834	102900	4	debit	2017-03-04 4:37:34
1259	1260	78	775	77175	3	credit_card	2017-03-27 9:27:20
2564	2565	78	915	77175	3	debit	2017-03-25 1:19:35
2690	2691	78	962	77175	3	debit	2017-03-22 7:33:25
2906	2907	78	817	77175	3	debit	2017-03-16 3:45:46
3403	3404	78	928	77175	3	debit	2017-03-16 9:45:05
3724	3725	78	766	77175	3	credit_card	2017-03-16 14:13:26
4192	4193	78	787	77175	3	credit_card	2017-03-18 9:25:32
4420	4421	78	969	77175	3	debit	2017-03-09 15:21:35
4715	4716	78	818	77175	3	debit	2017-03-05 5:10:44

In [ ]:
```
'''
The problem with this calculation is that there are 2 stores that have orders
in bulk, which is making the dataset imbalanced to calculate the average
order value. Because the average order value takes the total revenue and
divides that by number of orders, it gives a very high number due to these
bulk values.
'''
```

In [ ]:
```
# (b) What metric would you report for this dataset?
'''
There are a few ways we could go about fixing it:
(1) We could remove the large bulk orders, but I wouldn't do that since that
data is still important, even though it skews our AOC results.
(2) We could also keep the same metric of AOV, but calculate it by individual
stores
(3) We could change the metric completly to median, which would order the
order_amount and take the middle number. This would be my pick.

I choose to use the median instead of these other options since the mean is
only used when the dataset is symetrical. Since we have an unbalanced,
asymetrical dataset which has outliers that are very high, the median is the
better option. Our data here is too skewed to use the mean.
'''
```

In [12]:
```
# (c) What is the new value?

median = dataset['order_amount'].median()
print('The median is: ', median)
```

```
The median is:  284.0
```

In [ ]:
```
# Now we have a better value that more accurately reflects the dataset
```