##

OnMart Superstore

Author: Atef Bader **Created: 8/19/2021**
Date: 31 October 2021
Edited by: Dr. Dill and Dr. Arroyo
- Assigned points to each question
- Clarified each question
- Added bonus questions

Deliverables: - Submit on Canvas two files with the format of **Lastname_Assignment_5**: 1. Your IPYNB script that has your source code and output for the requirements listed below 2. Your HTML document that has your source code and output for the requirements listed below

High Level Description: The OnMart superstore is an online retailer that has a business model similar to Amazon, Walmart, and Target. The online superstore has several departments that are selling products in different categories: Electronics, Clothing, Grocery, Furniture, Sports, etc.

Logistic and Supply Chain Network: For its supply chain and its delivery network/vehicles, OnMart has several warehouses and distributions centers. Every warehouse supplies a number of distribution centers and every distribution center delivers packages in different zip codes. The logistics and supply chain network for OnMart has the following characteristics: 1. It delivers packages to customers distributed across 785 zip codes 2. It has 97 distribution centers that are located in 97 zip codes 3. It has 17 warehouses that are located in 17 zip codes 4. It serves customers in 8 cities located in different states in the US 5. Every serving facility (distribution center or warehouse) has a unique pair of latitude and longitude Even though OnMart has many warehouses that supply distribution centers in the different cities, not every city has warehouses; Nashville and Atlanta do not have warehouses. Currently, Nashville is being supplied by products shipped from Chicago warehouses and Atlanta is being supplied by products shipped from Miami warehouses. The following figure illustrates the structure of the OnMart delivery network: ![image.png](attachment:image.png)

The following is a sample of the delivery zip codes, warehouses, distribution centers, cities, and state:

![image.png](attachment:image.png)

Departments and Products OnMart has several departments that are selling products in different categories. The following is a sample of these departments and the different product categories they sell: ![image.png](attachment:image.png)

Customers and Purchases Customers place their orders online and the order might have products from the different categories in the different departments. For every order there will be a unique order number generated. When the customers place the order online, the customer will enter home address, delivery address, and shipping class.
The customer is provided with the following capabilities: - Purchase items. - Return purchased items. - Review and rate purchased items. - Some customers are connected with friends in a Social Network and can share product description of purchased items with friends in the social network.

Connection instructions - Connect to NU VPN - Connect to DSCC Postgres Server via psycog2 connection string (provided in code below) - Connect to **onmart** database (provided in code below) - Use the two tables: **transactions_log** and **logistics_supply_chain_network**

In [1]:
```
import pandas as pd
import datetime
from datetime import datetime, date, timedelta
import time
import numpy as np
import plotly
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import plotly.express as px
import psycopg2
import csv

from area import area

from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT
from IPython.display import display
```

In [2]:
```
# allow muliple output in one cell window
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Check versions:

In [3]:
```
from platform import python_version
print('python version installed     :', python_version(),'    ,needed: 3.7.7 or higher')
print('plotly version installed     :', plotly.__version__,'   ,needed: 4.14.3 or higher')
print('matplotlib version installed :', matplotlib.__version__,'    ,needed: 3.2.2 or high
print('numpy version installed      :', np.__version__,'    ,needed: 1.19.5 or higher')
```

```
python version installed     : 3.9.7     ,needed: 3.7.7 or higher
plotly version installed     : 5.6.0    ,needed: 4.14.3 or higher
matplotlib version installed : 3.4.3     ,needed: 3.2.2 or higher
numpy version installed      : 1.20.3     ,needed: 1.19.5 or higher
```

In [4]:
```
# Connect to onmart database on Postgres
```

```
db_connection = psycopg2.connect(host='129.105.248.26',dbname="onmart", user="amb6291")

cursor = db_connection.cursor()
```

In [5]:

```
# Get the column names for table transactions_log

cursor.execute("SELECT column_name \
                FROM INFORMATION_SCHEMA.COLUMNS \
                WHERE table_name = 'transactions_log';")

rows=cursor.fetchall()

rows
```

Out[5]:

```
[('customerid',),
 ('firstname',),
 ('lastname',),
 ('creditcardnumber',),
 ('orderid',),
 ('purchasedate',),
 ('expecteddeliverydate',),
 ('actualdeliverydate',),
 ('productid',),
 ('department',),
 ('category',),
 ('itempurchased',),
 ('quantity',),
 ('price',),
 ('shippingcost',),
 ('discount',),
 ('sales',),
 ('profit',),
 ('deliveryzipcode',),
 ('homezipcode',),
 ('segment',),
 ('orderpriority',),
 ('orderreturned',),
 ('rating',),
 ('reviewid',),
 ('friends',),
 ('sharedwith',)]
```

In [6]:

```
# Load the table transactions_log into a dataframe:
#dfTrans = pd.read_sql_query("select * from transactions_log", db_connection)
#dfTrans.head()
#dfTrans.info()
```

In [7]:

```
# create a dataframe for transaction_log - note the limit to 5000 records
query = '''SELECT *
            FROM transactions_log
            LIMIT 5000;'''

cursor.execute(query)
data = cursor.fetchall()

colnames = [desc[0] for desc in cursor.description]
#create dataframe with column headers
customers = pd.DataFrame(data, columns = colnames)
customers.info()
customers.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   customerid            5000 non-null   object
 1   firstname             5000 non-null   object
 2   lastname              5000 non-null   object
 3   creditcardnumber      5000 non-null   object
 4   orderid               5000 non-null   object
 5   purchasedate          5000 non-null   object
 6   expecteddeliverydate  5000 non-null   object
 7   actualdeliverydate    5000 non-null   object
 8   productid             5000 non-null   object
 9   department            5000 non-null   object
 10  category              5000 non-null   object
 11  itempurchased         5000 non-null   object
 12  quantity              5000 non-null   int64
 13  price                 5000 non-null   float64
 14  shippingcost          5000 non-null   float64
 15  discount              5000 non-null   float64
 16  sales                 5000 non-null   float64
 17  profit                5000 non-null   float64
 18  deliveryzipcode       5000 non-null   object
 19  homezipcode           5000 non-null   object
 20  segment               5000 non-null   object
 21  orderpriority         5000 non-null   object
 22  orderreturned         5000 non-null   object
 23  rating                5000 non-null   int64
 24  reviewid              5000 non-null   object
 25  friends               5000 non-null   object
 26  sharedwith            5000 non-null   object
dtypes: float64(5), int64(2), object(20)
memory usage: 1.0+ MB
```

Out[7]:

	customerid	firstname	lastname	creditcardnumber	orderid	purchasedate	expecteddeliverydate
0	381-91-3338	Cullen	Armstrong	xxxx-xxxx-xxxx-4162	59b19d07-d5af-4320-9bf8-a2eeb84578d1	2018-01-09	2018-01-13
1	740-35-5564	Nola	Kshlerin	xxxx-xxxx-xxxx-7206	04f48338-f39a-41c5-bb8a-3f88c0470e4b	2019-03-01	2019-03-05
2	774-94-3073	Philip	Wilderman	xxxx-xxxx-xxxx-1184	77fec7d0-b391-460f-973c-5c85ac685b37	2020-06-13	2020-06-17
3	631-39-8872	Alessandro	Thiel	xxxx-xxxx-xxxx-7101	cb44de08-0bad-4ba0-862e-bbbfb863e72d	2018-01-09	2018-01-13
4	642-30-5778	Sofia	Rutherford	xxxx-xxxx-xxxx-8395	cf4b6c08-0024-4da6-9392-b57de5f993fd	2019-03-01	2019-03-05

5 rows × 27 columns

In [8]:
```
# check out customer

cursor.execute("SELECT * \
                FROM transactions_log \
                Where customerid = '004-81-0268';")

rows = cursor.fetchall()

rows
```

Out[8]:
```
[('004-81-0268',
  'Lydia',
  'Turner',
  'xxxx-xxxx-xxxx-5873',
  '5d585c1e-1df4-4eb2-8bc5-aba8dc173087',
  datetime.date(2019, 10, 8),
  datetime.date(2019, 10, 12),
  datetime.date(2019, 10, 12),
  'ELE-CO-2076-686',
  'Electronics',
  'Copiers',
  'HP Multipurpose Laser Scanner',
  2,
  540.26,
  24.2,
  20.51,
  1084.21,
  119.2631,
  '30309',
  '30309',
  'Consumer',
  'High',
  'No',
  5,
  '',
  '',
  '')]
```

In [9]:
```
# Get the column names for table logistics_supply_chain_network

cursor.execute("SELECT column_name \
                FROM INFORMATION_SCHEMA.COLUMNS \
                WHERE table_name = 'logistics_supply_chain_network';")

rows=cursor.fetchall()

rows
```

Out[9]:
```
[('zipcode',),
 ('city',),
 ('state',),
 ('latitude',),
 ('longitude',),
 ('facility_id',),
 ('distribution_center_id',),
 ('warehouse_id',),
 ('where_is',)]
```

In [10]:
```
# create dataframe for the supply chain network table
query = '''SELECT *
            FROM logistics_supply_chain_network;'''

cursor.execute(query)
```

```
data = cursor.fetchall()

colnames = [desc[0] for desc in cursor.description]
#create dataframe with column headers
centers = pd.DataFrame(data, columns = colnames).drop_duplicates()
centers.info()
centers.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 785 entries, 0 to 784
Data columns (total 9 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   zipcode                785 non-null    object 
 1   city                   785 non-null    object 
 2   state                  785 non-null    object 
 3   latitude               785 non-null    float64
 4   longitude              785 non-null    float64
 5   facility_id            785 non-null    object 
 6   distribution_center_id  785 non-null    object 
 7   warehouse_id           785 non-null    object 
 8   where_is               0 non-null      object 
dtypes: float64(2), object(7)
memory usage: 61.3+ KB
```

Out[10]:

	zipcode	city	state	latitude	longitude	facility_id	distribution_center_id	warehouse_id	where_i
0	60651	Chicago	IL	41.901485	-87.74055	Warehouse_1	DistributionCenter_1	Warehouse_1	Non
1	60697	Chicago	IL	41.811929	-87.68732		DistributionCenter_1	Warehouse_1	Non
2	60667	Chicago	IL	41.811929	-87.68732		DistributionCenter_1	Warehouse_1	Non
3	60694	Chicago	IL	41.811929	-87.68732		DistributionCenter_1	Warehouse_1	Non
4	60684	Chicago	IL	41.811929	-87.68732		DistributionCenter_1	Warehouse_1	Non

In [11]:

```
# check out zips
centers['zipcode'].nunique()
```

Out[11]:

```
785
```

In [12]:

```
# check out locations
centers['facility_id'].nunique()
centers['distribution_center_id'].nunique()
centers['warehouse_id'].nunique()
```

Out[12]:

```
119
```

Out[12]:

```
97
```

Out[12]:

```
22
```

In [13]:

```
# what states are there?
centers['state'].value_counts()
```

Out[13]:

```
NY    166
TX    123
GA    111
CA    100
FL     96
IL     85
MA     54
```

```
TN      44
NJ       1
MI       1
NC       1
PA       1
IA       1
VA       1
Name: state, dtype: int64
```

Complete all requirements listed below - Write your code in the cell below every requirement

In [14]:
```
# Uncomment and run this code when your transactions get "stuck"
db_connection.rollback()
```

Requirement 1 (5 points): - To see the delivery zip code with the most returns, count the total number of order returns per Delivery Zip Code. Show the top five zip codes with their count. **(Use SQL - Postgres)**

In [15]:
```
cursor.execute('''SELECT deliveryzipcode, count(*) AS ordersreturned
                            FROM transactions_log
                            WHERE orderreturned = 'Yes'
                            GROUP BY deliveryzipcode
                            ORDER BY ordersreturned DESC
                            LIMIT 5;''')
r1data = cursor.fetchall()

colnames = ['DeliveryZip', 'NumberOfReturns']
r1 = pd.DataFrame(r1data, columns = colnames).drop_duplicates()
r1.head()
```

Out[15]:

	DeliveryZip	NumberOfReturns
0	75202	1421
1	60660	1411
2	75210	1395
3	60649	1386
4	75218	1386

Requirement 2 (5 points): - Which product categories had the most returns? We want to know the number of returns by the category and the delivery zip code. Count the total number of orders for every product category in every Delivery Zip Code. The output should show the category, the zip code and the count; show the top 10. **(Use SQL - Postgres)**

In [16]:
```
cursor.execute('''SELECT category, deliveryzipcode, count(*) AS ordersreturned
                    FROM transactions_log
                    WHERE orderreturned = 'Yes'
                    GROUP BY category, deliveryzipcode
                    ORDER BY ordersreturned DESC
                    LIMIT 10;''')
```

```
r2data = cursor.fetchall()

colnames = ['ProductCategory', 'DeliveryZip', 'NumberOfReturns']
r2 = pd.DataFrame(r2data, columns = colnames).drop_duplicates()
r2.head(10)
```

Out[16]:

	ProductCategory	DeliveryZip	NumberOfReturns
0	Shorts	60643	146
1	Shorts	60614	143
2	Shorts	75202	141
3	TShirt	75212	136
4	TShirt	60660	136
5	TShirt	60646	133
6	Shorts	75210	131
7	TShirt	60678	130
8	TShirt	60626	130
9	Shorts	60603	129

Requirement 3 (5 points): Calculate the total sales per Delivery Zip Code; Use the **sales** column for your total sales value. Sort by total sales with the highest amount showing on top. **(Use SQL - Postgres)**

In [17]:

```
cursor.execute('''SELECT deliveryzipcode, SUM(sales) AS totalSales
                  FROM transactions_log
                  GROUP BY deliveryzipcode
                  ORDER BY totalSales DESC;''')

r3data = cursor.fetchall()

colnames = ['DeliveryZip', 'TotalSales']
r3 = pd.DataFrame(r3data, columns = colnames).drop_duplicates()
r3.head(10)
```

Out[17]:

	DeliveryZip	TotalSales
0	60636	6723517.19
1	75225	6663814.60
2	60679	6634994.33
3	75210	6629017.59
4	60646	6620039.21
5	75214	6569905.55
6	75224	6562457.31
7	60673	6536017.53
8	75223	6523289.58
9	60615	6509460.70

Requirement 4 (15 points): - **Part 1: (Use Python - SQL - Postgres)** - Use Seaborn to create a

product category side-by-side BoxPlot Chart.
- The boxplots will show the total sales (sum) for every delivery zip code on the (y-axis) for every product category (x-axis). - Suggest to create a dataframe with deliveryzipcode, category and sales and use this dataframe for the boxplots. - **Part 2:** - In a Markdown cell, explain the insights from the Boxplots
https://seaborn.pydata.org/generated/seaborn.boxplot.html

In [18]:
```
cursor.execute('''SELECT deliveryzipcode, category, sales
                    FROM transactions_log;''')

r4data = cursor.fetchall()

colnames = ['DeliveryZip', 'Category', 'Sales']
r4 = pd.DataFrame(r4data, columns = colnames).drop_duplicates()
r4.head(10)
```

Out[18]:

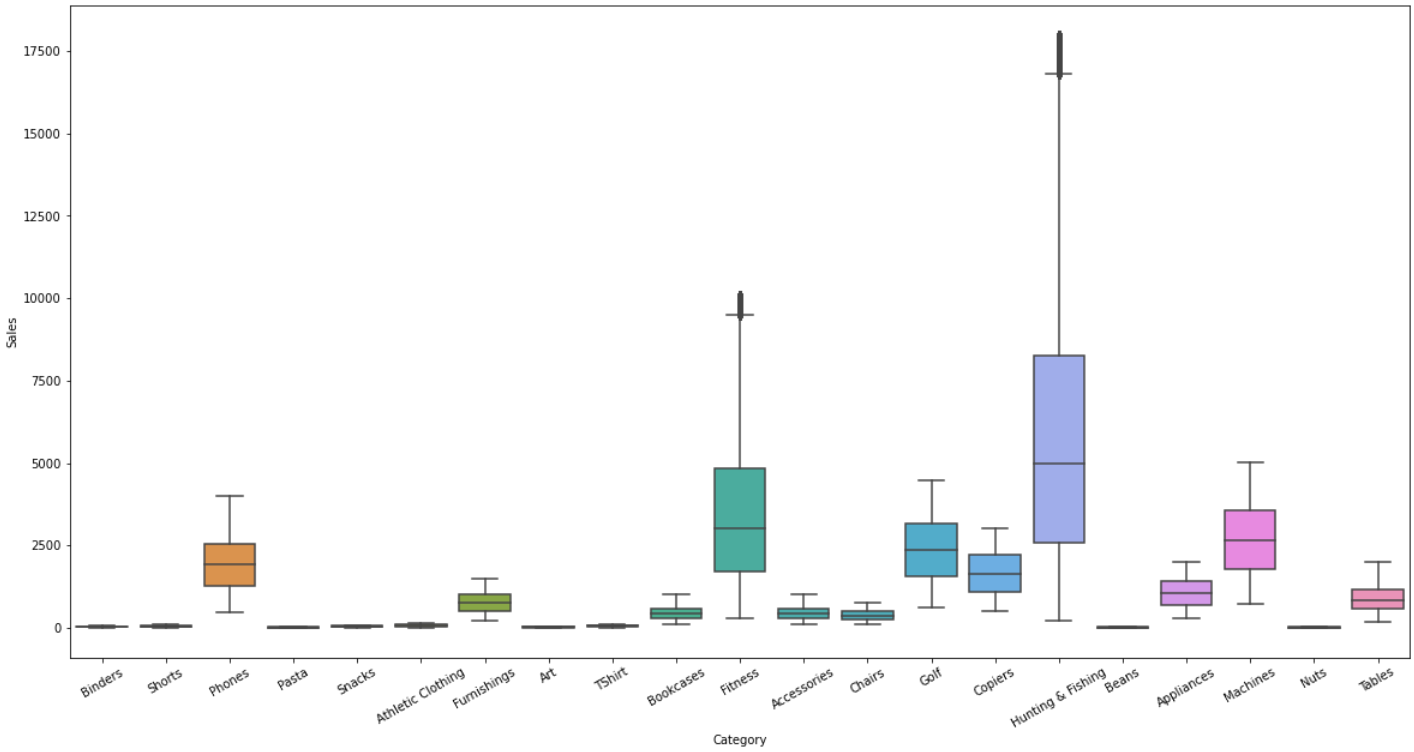
	DeliveryZip	Category	Sales
0	90027	Binders	45.79
1	90040	Shorts	33.07
2	2208	Shorts	67.61
3	75240	Phones	2356.40
4	33165	Pasta	6.88
5	60661	Snacks	41.60
6	33257	Athletic Clothing	42.70
7	2125	Furnishings	655.72
8	10257	Snacks	67.70
9	75380	Art	26.63

In [19]:
```
plt.figure(figsize=(20,10))
ax = sns.boxplot(x=r4['Category'], y=r4['Sales'])
ax.set_xticklabels(ax.get_xticklabels(),rotation=30)
plt.show()
```

Out[19]:
```
<Figure size 1440x720 with 0 Axes>
```

Out[19]:
```
[Text(0, 0, 'Binders'),
 Text(1, 0, 'Shorts'),
 Text(2, 0, 'Phones'),
 Text(3, 0, 'Pasta'),
 Text(4, 0, 'Snacks'),
 Text(5, 0, 'Athletic Clothing'),
 Text(6, 0, 'Furnishings'),
 Text(7, 0, 'Art'),
 Text(8, 0, 'TShirt'),
 Text(9, 0, 'Bookcases'),
 Text(10, 0, 'Fitness'),
 Text(11, 0, 'Accessories'),
 Text(12, 0, 'Chairs'),
 Text(13, 0, 'Golf'),
 Text(14, 0, 'Copiers'),
 Text(15, 0, 'Hunting & Fishing'),
 Text(16, 0, 'Beans'),
 Text(17, 0, 'Appliances'),
```

```
Text(18, 0, 'Machines'),
Text(19, 0, 'Nuts'),
Text(20, 0, 'Tables')]
```



The hunting and fishing category has a very wide variance with quite a few outliers. The fitness category also has a few outliers. This could make sense do to the high variance of cost in these types of products, as well as the lower consistency in these types of purchases.

Requirement 5 (15 points): - Calculate the total number of orders per product category per Home Zip Code per purchase month. Sort by home zip code, cateogry and month and show the first 12 records. **(Use Python - SQL - Postgres)**

You can use either Python or SQL to create the month field. If you want to try SQL, here is one of many sources of info: https://www.postgresqltutorial.com/postgresql-extract/

In [20]:
```
cursor.execute('''SELECT homezipcode, category, EXTRACT(MONTH FROM purchasedate) AS month,
                  FROM transactions_log
                  GROUP BY category, homezipcode, month
                  ORDER BY homezipcode, category, month
                  LIMIT 12;''')

r5data = cursor.fetchall()

colnames = ['HomeZip', 'Category', 'Month', 'Orders']
r5 = pd.DataFrame(r5data, columns = colnames).drop_duplicates()
r5['Month'] = r5['Month'].astype(int)
r5.head(12)
```

Out[20]:

	HomeZip	Category	Month	Orders
0	10001	Accessories	1	23
1	10001	Accessories	2	31
2	10001	Accessories	3	8
3	10001	Accessories	4	10

	HomeZip	Category	Month	Orders
4	10001	Accessories	5	8
5	10001	Accessories	6	22
6	10001	Accessories	7	15
7	10001	Accessories	8	11
8	10001	Accessories	9	14
9	10001	Accessories	10	17
10	10001	Accessories	11	34
11	10001	Accessories	12	13

Requirement 6 (15 points): - Use Seaborn (FacetGrid and scatterplot) to create six scatter plots, one for each delivery zipcode in the list of zipcodes [60623, 60663, 60609, 60660, 60615, 60622]. - Each zip code should appear in its own chart. - Plot the daily sales for these categories: Athletic Clothing, Fitness, Golf, Hunting & Fishing, Shorts - The colors in the plots should represent the categories. - **(Use Python - SQL - Postgres)**
Reference: https://seaborn.pydata.org/generated/seaborn.FacetGrid.html

In [21]:
```
zips = '60623', '60663', '60609', '60660', '60615', '60622'
cats = 'Athletic', 'Clothing', 'Fitness', 'Golf', 'Hunting & Fishing', 'Shorts'

cursor.execute('''SELECT deliveryzipcode, category, sales, purchasedate
                    FROM transactions_log
                    WHERE deliveryzipcode IN %s AND category IN %s
                    GROUP BY deliveryzipcode, category, sales, purchasedate;''', [zips, ca

r6data = cursor.fetchall()

colnames = ['DeliveryZip', 'Category', 'Sales', 'Date']
r6 = pd.DataFrame(r6data, columns = colnames).drop_duplicates()
r6.head(10)
```

Out[21]:

	DeliveryZip	Category	Sales	Date
0	60609	Fitness	508.81	2020-12-26
1	60609	Fitness	558.97	2019-01-08
2	60609	Fitness	615.18	2020-02-29
3	60609	Fitness	616.27	2020-02-26
4	60609	Fitness	617.26	2018-06-01
5	60609	Fitness	692.62	2019-07-11
6	60609	Fitness	703.01	2020-12-09
7	60609	Fitness	709.47	2019-09-23
8	60609	Fitness	710.85	2020-08-13
9	60609	Fitness	720.52	2020-02-11

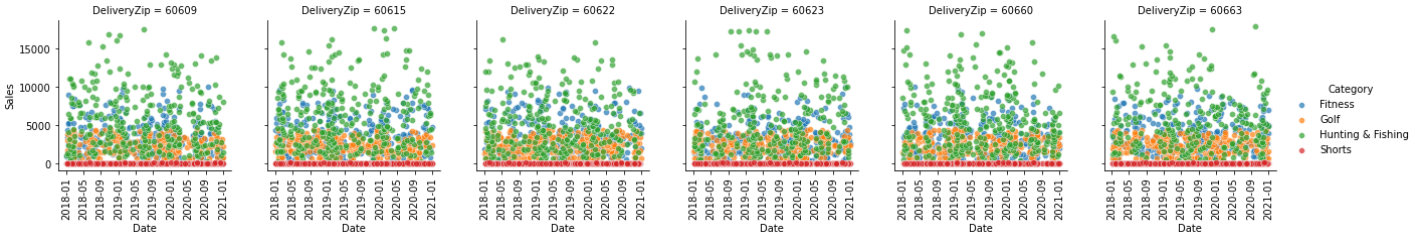
In [22]:
```
r6_plt = sns.FacetGrid(r6, col="DeliveryZip", hue="Category")
r6_plt.map(sns.scatterplot, "Date", "Sales", alpha=.7)
for axes in r6_plt.axes.flat:
```

```
    _ = axes.set_xticklabels(axes.get_xticklabels(), rotation=90)
r6_plt.add_legend()
```

Out[22]:
```
<seaborn.axisgrid.FacetGrid at 0x7feafc35ff40>

/var/folders/b7/zyw1_ftd2r95pj7r6z65m1gm0000gn/T/ipykernel_6349/3617415949.py:4: UserWarni
ng: FixedFormatter should only be used together with FixedLocator
  _ = axes.set_xticklabels(axes.get_xticklabels(), rotation=90)
```

Out[22]:
```
<seaborn.axisgrid.FacetGrid at 0x7feafc35ff40>
```



Requirement 7 (5 points): - List the top 10 distribution centers that have **highest** number of delayed deliveries? Show the distribution center ID, the city, the center's zip and the number of delayed deliveries. **(Use Python - SQL - Postgres)**

In [23]:
```
cursor.execute('''SELECT s.distribution_center_id, s.city, s.zipcode, count(*) AS deliveri
                    FROM transactions_log t, logistics_supply_chain_network s
                    WHERE t.deliveryzipcode = s.zipcode AND t.expecteddeliverydate <= t.ac
                        AND s.facility_id = s.distribution_center_id
                    GROUP BY s.distribution_center_id, s.city, s.zipcode
                    ORDER BY deliveries DESC
                    LIMIT 10;''')

r7data = cursor.fetchall()

colnames = ['DistributionID', 'City', 'Zipcode', 'DelayedDeliveries']
r7 = pd.DataFrame(r7data, columns = colnames).drop_duplicates()
r7.head(10)
```

Out[23]:

	DistributionID	City	Zipcode	DelayedDeliveries
0	DistributionCenter_78	Dallas	75224	6787
1	DistributionCenter_6	Chicago	60640	6742
2	DistributionCenter_5	Chicago	60631	6729
3	DistributionCenter_11	Chicago	60670	6724
4	DistributionCenter_1	Chicago	60644	6702
5	DistributionCenter_8	Chicago	60671	6685
6	DistributionCenter_79	Dallas	75211	6682
7	DistributionCenter_68	Dallas	75223	6680
8	DistributionCenter_2	Chicago	60647	6673
9	DistributionCenter_4	Chicago	60685	6659

Requirement 8 (15 points): - Find the most **influential product reviewer** for every product. - Note: the **influential product reviewer** is the customer who rated the product either 5 or 1, wrote a review for the product, has the highest number of friends, and shared it with the highest number of

friends. **(Use Python - SQL - Postgres)** - In the following example customer c3 is the most influential reviewer (list is sorted decending by the **count** of sharedwith and friends)

Example:

1. c3 : sharedwith=5, friends=10
2. c1 : sharedwith=4, friends=15
3. c2 : sharedwith=4, friends=10
4. c5 : sharedwith=3, friends=12
5. c4 : sharedwith=3, friends=10

Hints for Requirement 8

1. Select all your data
2. Parse the friends field to get a count and store in your dataframe
3. Parse the sharedwith field to get a count and store in your dataframe
4. Sort output by sharewith count and friends count

	customerid	lastname	friends	sharedwith	reviewid
689347	583-10-1717	Beier	538-06-4627;625-45-6218;146-74-4763;378-42-4105	378-42-4105	af334-4cbb
689348	446-06-0298	Lebsack	802-51-1018;170-40-2761	802-51-1018	838e4-a1a9
689349	244-23-1551	Zieme	521-24-1695;497-99-6785;752-05-3362;038-22-3876	521-24-1695	59e0b-b7b4
689350	651-69-6790	Schiller	684-79-1266;836-57-7366;553-62-3887	684-79-1266;553-62-3887	fbfe2-be4c

In [24]:

```
cursor.execute('''SELECT productid, customerid, lastname, friends, sharedwith, reviewid
                  FROM transactions_log
                  WHERE rating = 1 OR rating = 5;''')

r8data = cursor.fetchall()

colnames = ['ProductID','CustomerID', 'Lastname', 'Friends', 'SharedWith', 'ReviewID']
r8 = pd.DataFrame(r8data, columns = colnames).drop_duplicates()
r8.head()
```

Out[24]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID
0	CLO-SH-0896-667	677-18-9377	Lebsack	010-65-1397;606-83-6670;798-79-9490	010-65-1397;798-79-9490	6a0f0-7d0d
1	GRO-NU-9742-556	158-70-8620	Gutkowski	616-16-8774;616-53-4099;153-52-4746;310-79-0061	616-16-8774;616-53-4099	
2	SPO-HU-3553-052	800-70-6327	Harber	655-33-8752;681-60-2900;135-29-7385;300-92-8644	655-33-8752;681-60-2900;300-92-8644	
3	FUR-CH-1589-358	345-16-5244	Stroman	149-81-7782;121-48-7416;575-18-4699	575-18-4699	
4	OFF-BI-7934-050	490-31-5996	Beier	250-60-7420;737-10-6640	250-60-7420;737-10-6640	356d3-0117

In [25]:

```
#replace empty strings with Nan for counting
r8.replace(r'^\s*$', np.nan, inplace=True, regex=True)
r8.head()
```

Out[25]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID
0	CLO-SH-0896-667	677-18-9377	Lebsack	010-65-1397;606-83-6670;798-79-9490	010-65-1397;798-79-9490	6a0f0-7d0d
1	GRO-NU-9742-556	158-70-8620	Gutkowski	616-16-8774;616-53-4099;153-52-4746;310-79-0061	616-16-8774;616-53-4099	NaN
2	SPO-HU-3553-052	800-70-6327	Harber	655-33-8752;681-60-2900;135-29-7385;300-92-8644	655-33-8752;681-60-2900;300-92-8644	NaN
3	FUR-CH-1589-358	345-16-5244	Stroman	149-81-7782;121-48-7416;575-18-4699	575-18-4699	NaN
4	OFF-BI-7934-050	490-31-5996	Beier	250-60-7420;737-10-6640	250-60-7420;737-10-6640	356d3-0117

In [26]:

```
#drop na from reviewID
r8.dropna(subset=['ReviewID'], inplace=True)
r8.head()
```

Out[26]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID
0	CLO-SH-0896-667	677-18-9377	Lebsack	010-65-1397;606-83-6670;798-79-9490	010-65-1397;798-79-9490	6a0f0-7d0d
4	OFF-BI-7934-050	490-31-5996	Beier	250-60-7420;737-10-6640	250-60-7420;737-10-6640	356d3-0117
5	GRO-BE-8481-662	187-93-4439	Parisian	446-34-9989;280-84-4114;576-34-4585	NaN	e1ff7-6829
8	SPO-HU-8773-354	685-49-1541	Weber	271-13-7868;186-09-5798;015-44-8557;517-55-2785	271-13-7868;517-55-2785	48d41-19e8
9	FUR-FU-5598-872	365-38-8521	Lakin	795-61-9903;174-62-7233;269-29-5496;374-62-0942	795-61-9903;269-29-5496	d1e5e-927d

In [27]:

```
#count number of friends
r8['FriendCount'] = r8['Friends'].str.split(';').str.len()
r8.head()
```

Out[27]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount
0	CLO-SH-0896-667	677-18-9377	Lebsack	010-65-1397;606-83-6670;798-79-9490	010-65-1397;798-79-9490	6a0f0-7d0d	3.0
4	OFF-BI-7934-050	490-31-5996	Beier	250-60-7420;737-10-6640	250-60-7420;737-10-6640	356d3-0117	2.0
5	GRO-BE-8481-662	187-93-4439	Parisian	446-34-9989;280-84-4114;576-34-4585	NaN	e1ff7-6829	3.0
8	SPO-HU-8773-354	685-49-1541	Weber	271-13-7868;186-09-5798;015-44-8557;517-55-2785	271-13-7868;517-55-2785	48d41-19e8	4.0
9	FUR-FU-5598-872	365-38-8521	Lakin	795-61-9903;174-62-7233;269-29-5496;374-62-0942	795-61-9903;269-29-5496	d1e5e-927d	4.0

In [28]:

```
#count number of shares
r8['SharedCount'] = r8['SharedWith'].str.split(';').str.len()
```

```
r8.head()
```

Out[28]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount	SharedCount
0	CLO-SH-0896-667	677-18-9377	Lebsack	010-65-1397;606-83-6670;798-79-9490	010-65-1397;798-79-9490	6a0f0-7d0d	3.0	2.0
4	OFF-BI-7934-050	490-31-5996	Beier	250-60-7420;737-10-6640	250-60-7420;737-10-6640	356d3-0117	2.0	2.0
5	GRO-BE-8481-662	187-93-4439	Parisian	446-34-9989;280-84-4114;576-34-4585	NaN	e1ff7-6829	3.0	NaN
8	SPO-HU-8773-354	685-49-1541	Weber	271-13-7868;186-09-5798;015-44-8557;517-55-2785	271-13-7868;517-55-2785	48d41-19e8	4.0	2.0
9	FUR-FU-5598-872	365-38-8521	Lakin	795-61-9903;174-62-7233;269-29-5496;374-62-0942	795-61-9903;269-29-5496	d1e5e-927d	4.0	2.0

In [29]:

```
#identify how many customers per product
r8['ProductID'].value_counts()
```

Out[29]:

```
CLO-TS-7925-542    3
OFF-AR-6302-487    3
CLO-TS-5222-355    3
ELE-CO-8762-467    3
OFF-BI-3736-728    3
                  ..
CLO-SH-0045-552    1
SPO-HU-3608-156    1
GRO-NU-0323-734    1
CLO-TS-4362-740    1
GRO-BE-7602-986    1
Name: ProductID, Length: 687897, dtype: int64
```

In [30]:

```
#displays most influential BY PRODUCTID -- most influential for each product
r8_byprod = r8.sort_values(by=['ProductID', 'SharedCount', 'FriendCount'], ascending = Fal
r8_byprod.drop_duplicates(subset=['ProductID'], inplace=True)
r8_byprod.head()
```

Out[30]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount	SharedCount
1199657	SPO-HU-9991-442	582-03-3384	Anderson	685-81-8124	685-81-8124	5bf3c-3348	1.0	1.0
1289530	SPO-HU-9990-329	470-21-1875	Shanahan	NaN	NaN	59412-f665	NaN	NaN
537471	SPO-HU-9990-179	090-54-2477	Pollich	850-91-8763	850-91-8763	3f383-247d	1.0	1.0
899787	SPO-HU-9989-658	063-08-1414	O'Connell	836-43-2250;818-68-6207	836-43-2250;818-68-6207	5f8f9-393a	2.0	2.0

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount	SharedCount
1348575	SPO-HU-9988-882	000-69-9925	Wisozk	293-75-6386;265-27-0742;580-15-3101;599-80-1914	265-27-0742;580-15-3101;599-80-1914	0d9a8-fcbb	4.0	3.0

In [31]:
```
#confirm duplicates dropped
r8_byprod['ProductID'].value_counts()
```

Out[31]:
```
SPO-HU-9991-442    1
FUR-BO-0117-825    1
FUR-BO-0117-624    1
FUR-BO-0117-590    1
FUR-BO-0117-565    1
                  ..
OFF-AP-0030-121    1
OFF-AP-0029-588    1
OFF-AP-0029-285    1
OFF-AP-0029-232    1
CLO-SH-0000-050    1
Name: ProductID, Length: 687897, dtype: int64
```

In [32]:
```
#determine overall influencers
r8_overall = r8.sort_values(by=['SharedCount', 'FriendCount'], ascending = False)
r8_overall.head()
```

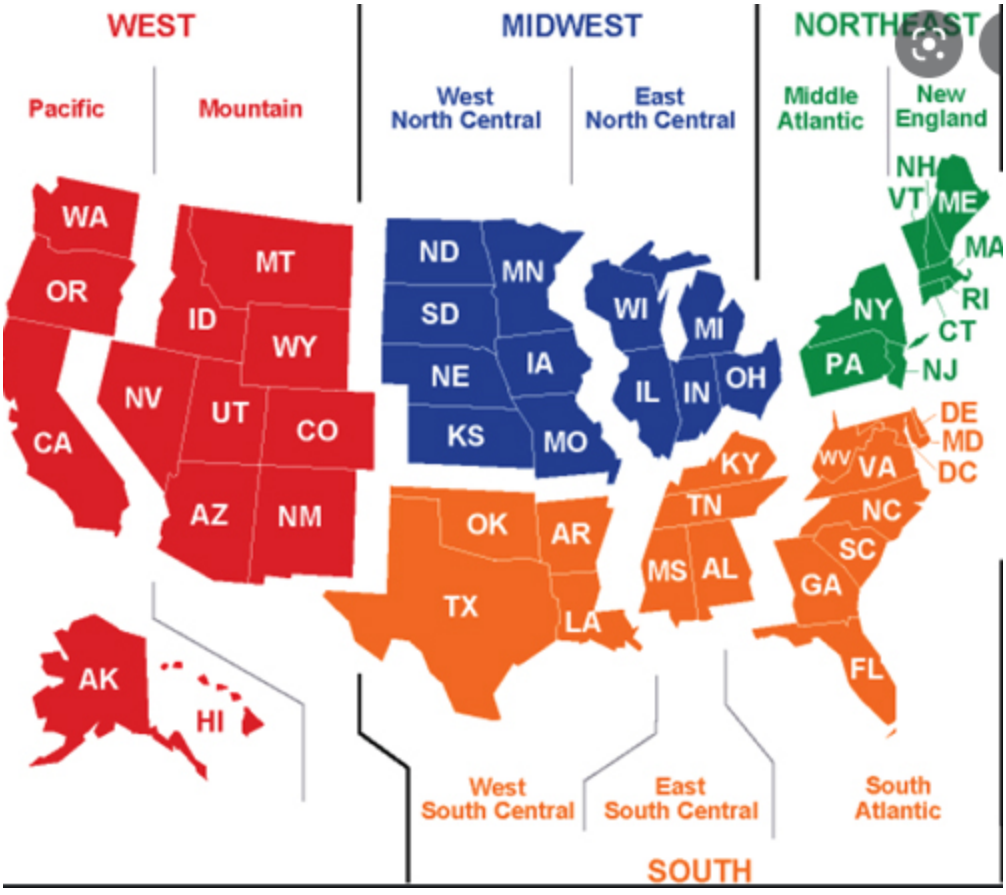
Out[32]:

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount	SharedCount
192	OFF-AR-5394-652	159-72-9047	Aufderhar	240-15-7206;446-08-2145;741-98-4525;001-39-2133	240-15-7206;446-08-2145;741-98-4525;001-39-2133	d9a14-419c	4.0	4.0
209	FUR-CH-1425-237	264-04-3286	Gibson	376-43-9955;517-24-3201;897-62-8223;296-16-5096	376-43-9955;517-24-3201;897-62-8223;296-16-5096	58cb3-8e53	4.0	4.0
263	CLO-SH-5025-531	283-48-1082	Heidenreich	851-28-4127;234-98-9086;162-87-5968;002-55-5549	851-28-4127;234-98-9086;162-87-5968;002-55-5549	99982-77ee	4.0	4.0
309	SPO-GO-7912-171	302-54-0525	Littel	051-56-6310;803-91-6500;845-22-2609;582-80-9634	051-56-6310;803-91-6500;845-22-2609;582-80-9634	89501-d0ae	4.0	4.0

	ProductID	CustomerID	Lastname	Friends	SharedWith	ReviewID	FriendCount	SharedCount
495	FUR-CH-3780-283	438-44-2573	Little	611-86-5322;890-45-8885;334-78-1687;773-99-8587	611-86-5322;890-45-8885;334-78-1687;773-99-8587	68494-3457	4.0	4.0

Bonus 1 (15 points): - OnMart has allocated the budget for the next quarter to expand its logistics and supply network by adding a new warehouse in the region that has the highest number of orders. Inspect the map below for the list of regions of United States and find the region that has the highest number of orders based on the delivery zip code and the zip code of the distribution center **(Use Python - SQL - Postgres)**

In [ ]:



In [ ]:

Bonus 2 (10 points): - To better plan for hiring hourly temporary workers on the peak-day of the week, OnMart has requested to find the busiest day of the week (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday or Sunday) that has the highest average for the number of order deliveries in every delivery zip code. Use the actual delivery date to determine the day of the week. **(Use Python - SQL - Postgres)**

In [ ]: