Assignment 6

setup environment (must run first)

```
%python
orders_df = spark.read.json("/FileStore/tables/orders.json")
orders_df.cache() # Cache data for faster reuse
stores_df = spark.read.json("/FileStore/tables/stores.json")
stores_df.cache() # Cache data for faster reuse
users_df = spark.read.json("/FileStore/tables/users.json")
users_df.cache() # Cache data for faster reuse
own_df = spark.read.json("/FileStore/tables/own.json")
own_df.cache() # Cache data for faster reuse
workfor_df = spark.read.json("/FileStore/tables/workfor.json")
workfor_df.cache() # Cache data for faster reuse
products_df = spark.read.json("/FileStore/tables/products.json")
products_df.cache() # Cache data for faster reuse
stockedby_df = spark.read.json("/FileStore/tables/stockedby.json")
stockedby_df.cache() # Cache data for faster reuse
vehicles_df = spark.read.json("/FileStore/tables/vehicles.json")
vehicles_df.cache() # Cache data for faster reuse

orders_df.createOrReplaceTempView("orders")
stores_df.createOrReplaceTempView("stores")
users_df.createOrReplaceTempView("users")
own_df.createOrReplaceTempView("own")
workfor_df.createOrReplaceTempView("workfor")
products_df.createOrReplaceTempView("products")
stockedby_df.createOrReplaceTempView("stockedby")
vehicles_df.createOrReplaceTempView("vehicles")
```

Solution

```
from pyspark.sql import functions as F
from pyspark.sql.functions import *

#1.A
orders_df.printSchema()

#1.B time_fulfilled data type is:
# time_fulfilled data type is string.
```

```
root
 |-- customer_id: string (nullable = true)
 |-- items: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- item_id: string (nullable = true)
 |    |    |-- product_id: string (nullable = true)
 |    |    |-- qty: long (nullable = true)
 |    |    |-- selling_price: double (nullable = true)
 |-- order_id: string (nullable = true)
 |-- pickup_time: string (nullable = true)
 |-- shopper_id: string (nullable = true)
 |-- store_id: string (nullable = true)
 |-- time_fulfilled: string (nullable = true)
 |-- time_placed: string (nullable = true)
 |-- total_price: double (nullable = true)
 |-- vehicle: struct (nullable = true)
 |    |-- license_plate: string (nullable = true)
 |    |-- state: string (nullable = true)
```

```
%sql
--1.C
DESCRIBE users;

--1.D phones data type is:
-- Phones data type is an array containing string data type for kind and number.
```

	col_name	data_type	comment
1	capacity	bigint	null
2	email	string	null
3	kind	array<string>	null
4	name	struct<first:string,last:string>	null
5	phones	array<struct<kind:string,number:string>>	null
6	user_id	string	null

Showing all 6 rows.

```
#2.A DF
display(orders_df.where("customer_id = 'JVN1X'").select("order_id"))
```

	order_id
1	8Y0WZ
2	M6FIH
3	K43RY
4	DWCA6
5	C24A3
6	DSC4O
7	L5KX5
8	C07AY
9	4RL11

Showing all 9 rows.

```
%sql
--2.A SQL
SELECT orders.order_id
FROM orders
WHERE customer_id = 'JVN1X';
```

	order_id
1	8Y0WZ
2	M6FIH
3	K43RY
4	DWCA6
5	C24A3
6	DSC4O
7	L5KX5
8	C07AY
9	4RL11

Showing all 9 rows.

```
#2.B DF
display(products_df.where("category LIKE '%&%'").groupby("category").count())
```

	category	count
1	Condiments, Spice, & Bake	35
2	Canned Goods & Soups	34
3	Paper, Cleaning, & Home	37
4	Fruits & Vegetables	35
5	Meat & Seafood	35
6	Breakfast & Cereal	30
7	Grains, Pasta, & Sides	35
8	Cookies, Snacks, & Candy	30
9	Bread & Bakery	35
10	Personal Care & Health	35
11	Dairy, Eggs, & Cheese	30

Showing all 11 rows.

```
%sql
--2.B SQL
SELECT category, count(*)
FROM products
WHERE category LIKE '%&%'
GROUP BY category;
```

	category	count(1)
1	Condiments, Spice, & Bake	35
2	Canned Goods & Soups	34
3	Paper, Cleaning, & Home	37
4	Fruits & Vegetables	35
5	Meat & Seafood	35
6	Breakfast & Cereal	30
7	Grains, Pasta, & Sides	35
8	Cookies, Snacks, & Candy	30
9	Bread & Bakery	35
10	Personal Care & Health	35
11	Dairy, Eggs, & Cheese	30

Showing all 11 rows.

```
#2.C DF
display(orders_df.where("time_fulfilled IS NOT
NULL").groupby("store_id").agg(F.count("*").alias("fulfilled_orders")).sort("fulfilled_orders",
ascending=False).select("store_id").head(5))
```

	store_id
1	1RMXY
2	2TM62
3	70GOX
4	17KE2
5	49TNX

Showing all 5 rows.

```
%sql
--2.C SQL
SELECT store_id
FROM orders
WHERE time_fulfilled IS NOT NULL
GROUP BY store_id
ORDER BY count(*) DESC
LIMIT 5;
```

	store_id
1	1RMXY
2	2TM62
3	70GOX
4	17KE2
5	49TNX

Showing all 5 rows.

```
#2.D DF
display(stores_df.withColumn("popular_category",
explode(stores_df.categories)).groupby("popular_category").count().sort("count",
ascending=False).select("popular_category").head(1))
```

	popular_category
1	Canned Goods & Soups

Showing all 1 rows.

```
%sql
--2.D SQL
SELECT popular_category
FROM stores AS s LATERAL VIEW explode(s.categories) AS popular_category
GROUP BY popular_category
ORDER BY count(*) DESC
LIMIT 1;
```

	popular_category
1	Canned Goods & Soups

Showing all 1 rows.

```
#2.E DF
display(orders_df.filter(('2020-05-01' <= orders_df.time_fulfilled) & (orders_df.time_fulfilled <
'2020-06-01')).join(users_df, orders_df.shopper_id ==
users_df.user_id).sort("time_fulfilled").select("name").head(5))
```

	name
1	{"first": "Mar", "last": "Phillips"}
2	{"first": "Ste", "last": "Nichols"}
3	{"first": "Elizabeth", "last": "Robinson"}
4	{"first": "James", "last": "Beltran"}
5	{"first": "Jose", "last": "Jenkins"}

Showing all 5 rows.

```
%sql
--2.E SQL
SELECT users.name
FROM orders, users
WHERE orders.shopper_id = users.user_id AND '2020-05-01' <= orders.time_fulfilled AND
orders.time_fulfilled < '2020-06-01'
ORDER BY orders.time_fulfilled
LIMIT 5;
```

	name
1	{"first": "Mar", "last": "Phillips"}
2	{"first": "Ste", "last": "Nichols"}
3	{"first": "Elizabeth", "last": "Robinson"}
4	{"first": "James", "last": "Beltran"}
5	{"first": "Jose", "last": "Jenkins"}

Showing all 5 rows.

```
#2.F DF
price = orders_df.groupby("customer_id").agg(F.sum("total_price").alias("sum")).filter(col("sum")
> 650)
vehicle =
own_df.groupby("customer_id").agg(F.count("license_plate").alias("count")).filter(col("count") >
1)
result = users_df.join(price, users_df.user_id == price.customer_id).join(vehicle,
users_df.user_id == vehicle.customer_id).sort("email").select("name", "email")
display(result)
```

	name	email
1	{"first": "Carl", "last": "Deleon"}	Deleon.carl28@gmail.com
2	{"first": "April", "last": "White"}	april83@gmail.com
3	{"first": "Bro", "last": "Copeland"}	bro_copeland@yahoo.com
4	{"first": "James", "last": "Adams"}	james5135@gmail.com
5	{"first": "Jill", "last": "Nielsen"}	nielsen9521@gmail.com
6	{"first": "Amy", "last": "Smith"}	smithAmy@gmail.com
7	{"first": "Drew", "last": "Smith"}	smith_Drew60@gmail.com

Showing all 7 rows.

```
%sql
--2.F SQL
WITH
price as (SELECT orders.customer_id as id FROM orders GROUP BY customer_id HAVING
sum(orders.total_price) > 650),
vehicle as (SELECT own.customer_id from own GROUP BY customer_id HAVING count(DISTINCT
own.license_plate) > 1)
SELECT name, email
FROM users, price, vehicle
WHERE users.user_id = price.id and price.id = vehicle.customer_id
ORDER BY users.email;
```

	name	email
1	{"first": "Carl", "last": "Deleon"}	Deleon.carl28@gmail.com
2	{"first": "April", "last": "White"}	april83@gmail.com
3	{"first": "Bro", "last": "Copeland"}	bro_copeland@yahoo.com
4	{"first": "James", "last": "Adams"}	james5135@gmail.com
5	{"first": "Jill", "last": "Nielsen"}	nielsen9521@gmail.com
6	{"first": "Amy", "last": "Smith"}	smithAmy@gmail.com
7	{"first": "Drew", "last": "Smith"}	smith_Drew60@gmail.com

Showing all 7 rows.

```
%sql
--2.G SQL
WITH
info (SELECT o.store_id, count(*) AS rnk
      FROM products, orders AS o LATERAL VIEW explode(o.items) AS item
      WHERE item.product_id = products.product_id
      GROUP BY o.store_id, products.category),
top_rank (SELECT info.store_id, max(info.rnk) as top_rnk
          FROM info
          GROUP BY info.store_id),
top_info (SELECT o.store_id as id, products.category, avg(item.selling_price) AS
avg_selling_price, count(*) AS rnk
          FROM products, orders AS o LATERAL VIEW explode(o.items) AS item
          WHERE item.product_id = products.product_id
          GROUP BY o.store_id, products.category)
SELECT stores.store_id, stores.name, top_info.category, top_info.avg_selling_price
FROM top_rank, top_info, stores
WHERE top_rank.store_id = top_info.id and top_rank.top_rnk = top_info.rnk and stores.store_id =
top_info.id
ORDER BY top_info.avg_selling_price DESC
LIMIT 10;
```

	store_id	name	category	avg_selling_price
1	VRGID	Maay Convenient Inc	Meat & Seafood	27.785
2	ZDSRP	Border Station	Pet Care	22.588333333333335
3	KMJVY	Beasley Enterprises Inc	Pet Care	19.21875
4	6QSXS	6-Twelve Convenient-Mart Inc	Pet Care	18.99909090909091
5	NAX4O	Irving Oil Corp	Pet Care	18.49
6	NJEIB	Super Quik Inc	Pet Care	18.38
7	70GOX	Spaceway Oil CO	Baby Care	17.71588235294118
8	UAU1O	Mapco	Pet Care	17.67875
9	C7KJY	Plaid Pantries Inc	Pet Care	17.6075
10	NT8S6	Beasley Enterprises Inc	Pet Care	17.50090909090909

Showing all 10 rows.

```
#2.H DF
rdd = products_df.where("category LIKE '%&%'").rdd
ans = rdd.mapValues(lambda x: 1).reduceByKey(lambda x, y: x+y).collect()
display(ans)
```

	_1	_2
1	Grains, Pasta, & Sides	35
2	Breakfast & Cereal	30
3	Fruits & Vegetables	35
4	Bread & Bakery	35
5	Paper, Cleaning, & Home	37
6	Cookies, Snacks, & Candy	30
7	Dairy, Eggs, & Cheese	30
8	Personal Care & Health	35
9	Canned Goods & Soups	34
10	Meat & Seafood	35
11	Condiments, Spice, & Bake	35

Showing all 11 rows.