DASTGYR CASE STUDY

MUHAMMAD HAMZA NAVIWALA

GOOGLE DRIVE LINK:
https://drive.google.com/drive/folders/1fb2PduvqECH_3mudzoGaEOX66QudEg9y?usp=sharing

TASK 1: CHOOSING THE TOP 5 AGENTS FOR LEADERSHIP ROLE

To select the top five agents, I had to examine the data from two tables. The first was a **user's** table, where I could see how many referrals each agent had. The other table was the **Agent order** table, which contained information on each agent's orders. These were the two criteria I had to consider to identify the top five agents. My initial objective was to construct these two SQL tables using MySQL workbench.

This statement below was used to create the user table:

```
CREATE TABLE users (
    id  INT NOT NULL,
    full_name CHAR(50),
    gender CHAR(1),
    is_active ENUM('TRUE', 'FALSE') NOT NULL DEFAULT 'FALSE',
    is_verify ENUM('TRUE', 'FALSE') NOT NULL DEFAULT 'FALSE',
    created_at  char(10),
    refer_by INT,
    refer_at char(30),
    ref_code INT,
    PRIMARY KEY (id)
);
```

The statement below is used to create the sales agent order table

```
CREATE TABLE sales_agent_orders (
            id INT NOT NULL,
            order_id INT NOT NULL,
            agent_user_id INT NOT NULL,
            PRIMARY KEY (id)
);
```

This statement below was to populate the tables created above with the data present in the CSV files provided to me.

```
LOAD DATA LOCAL INFILE "D:/Dastgyr_data/Users.csv"
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

After the data had been entered into both the tables I started analyzing it.

This table was created to contain the data which showed each agent's total referrals

```
CREATE TABLE each_agent_referrals (
    agent_user_id INT NOT NULL,
    total_referals INT
);
```

This table was created to contain the data which showed each agent's total orders

```
CREATE TABLE each_agent_orders (
    agent_user_id INT NOT NULL,
    total_orders INT
);
```

This statement was used to populate each_agent_referrals table

```
INSERT INTO each_agent_referrals
SELECT refer_by, count(refer_by) as total_referals
FROM users
WHERE refer_by != 0
GROUP BY refer_by
ORDER BY total_referals DESC;
```

This statement was used to populate each_agent_orders table

```
INSERT INTO each_agent_orders
SELECT agent_user_id, count(agent_user_id) as total_orders
FROM sales_agent_orders
-- WHERE refer_by != 0
GROUP BY agent_user_id
ORDER BY total_orders DESC;
```

My reasoning for selecting the top five agents was that whoever had the most orders referrals and onboarding referrals would be chosen. I used the query below to complete this final step. It displays four columns: agent id, total referrals, total order, and referrals and orders added together. I used the join command to join two tables together.

```
SELECT orders.agent_user_id, orders.total_orders, referals.total_referals, SUM(orders.total_orders+referals.total_referals)as total
FROM each_agent_orders as orders
LEFT JOIN each_agent_referrals as referals
ON orders.agent_user_id = referals.agent_user_id
GROUP BY orders.agent_user_id
ORDER BY total DESC;
```

This is how the output showed after the above query was executed.

	agent_user_id	total_orders	total_referals	total
	16781	275	819	1094
	20282	348	556	904
	18871	277	565	842
	17205	171	455	626
	46908	177	338	515
	37402	211	252	463
	46913	132	319	451
	11310	134	266	400
	22515	186	211	397
	44736	132	228	360
	30836	157	171	328

I then exported this table and used it for visualization using Mi8criosoft excel which I will talk about later.

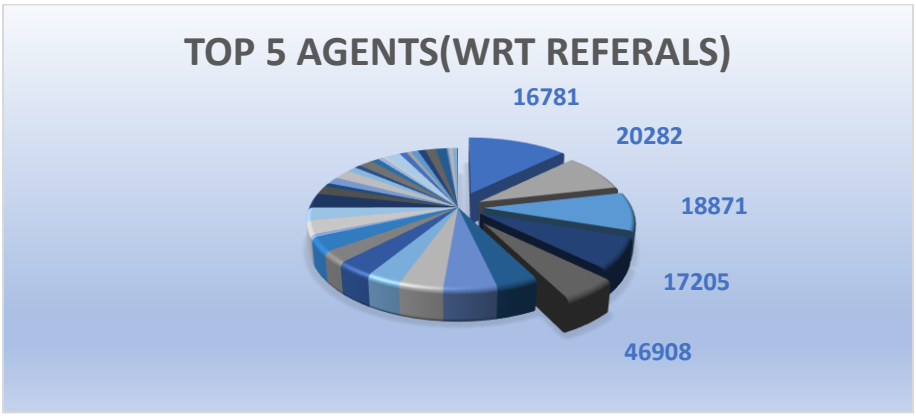
I also used python to extract the relevant information from the CSV files. I used Google Colaboratory for this task. To work with the CSV files I used the pandas' library function to read the CSV file and stored the data in the data frame. I used the similar approach that I used in SQL.

```
[ ] df2 = pd.read_csv('/content/drive/MyDrive/Dastgyr_case_study/Dastgyr_data/Users.csv', index_col=0)
    df2

[ ] temp2 = df2['Refer By'].value_counts()
    temp2.to_frame()

[ ] df3 = pd.read_csv('/content/drive/MyDrive/Dastgyr_case_study/Dastgyr_data/Sales Agents Orders.csv', index_col=0)

[ ] temp2 = df3['agent_user_id'].value_counts()
    temp2.to_frame()
```

The piece of code above shows that after reading the CSV files I used an aggregate function value_counts() to find each agent's total referrals and total orders.

To better understand my findings, I used visualization to explain them clearly.

The graph below shows the top five agents with the most referrals on signup



The chart below shows top five agents with most referrals while ordering.

The chart below shows top five agents with most order and signup referrals. These are the agents which I will choose for the lead role in the team

To answer the final question should we have different teams for onboarding and order booking? According to me, both these roles are interconnected so dividing them into two different teams will complicate the situation. At this point, it would be better to go with the same team as this will give more responsibility to the lead member and encourage others to try and increase their referrals on both onboarding and order placing.

TASK 2: Finding the most prevalent reasons for order cancellation.

For this task two CSV files were used, the cancellation reasons list and the cancellation reasons. Here also I used the create table statements and then loaded the data from the CSV file to these tables.

```
CREATE TABLE cancellation_reasons_list (
            order_id INT NOT NULL,
            cancellation_reason_id INT NOT NULL,
            cancel_reason_text char(100)
);

CREATE TABLE cancellation_reasons (
            id INT NOT NULL,
            reason_name char(100),
            complaint_id INT NOT NULL,
            PRIMARY KEY (id)
);

LOAD DATA LOCAL INFILE "D:/Dastgyr_data/Cancellation Reasons List.csv"
INTO TABLE cancellation_reasons_list
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

The cancellation reasons list table had a problem. The column **reason_text** was empty. This was due to the fact that there were some predetermined reasons in another table, each with its own unique id. Only rows with an id of 63 or 57 had their reason text column populated. Because the explanation for these two ids was defined as other in the preconfigured table, a particular reason was provided in this table. I intended to fill the entire **reason_text** table.

So I did a left join on tables **cancellation_reasons_list** and **cancellation_reasons**. The query I used is given below.

```
SELECT c1.order_id, c1.cancellation_reason_id, c1.cancel_reason_text, c2.reason_name
FROM cancellation_reasons_list as c1
LEFT JOIN cancellation_reasons as c2
ON c1.cancellation_reason_id = c2.id;
```

After I executed the query, I received the following output which I stored in a new table.

order_id	cancellation_reason_id	cancel_reason_text	reason_name
188354	49		Driver got late
188354	58		Customer Does Not Have Cash
188352	58		Customer Does Not Have Cash
188348	30		Retailer's Location is Incorrect
188345	58		Customer Does Not Have Cash
188343	63	shop closed	Other
188333	53		Late Delivery

I then created a new column in this new table using the following query

```
ALTER TABLE cancellation_reasons_list_updated ADD COLUMN cancellation_reason CHAR(100);
```

After that I used the following query to fill the new column. I used concat function to merge text of cancel_reason_text and reason_name into the new column.

```
UPDATE cancellation_reasons_list_updated
SET cancellation_reason = concat(cancel_reason_text, reason_name);
```

This is how my new table looks. I deleted the two columns cancel_reason_text and reason_name as there was no need of these columns.

order_id	cancellation_reason_id	cancellation_reason
188424	60	Customer Has Already Purchased It From Some...
188420	59	Shop Closed And Customer Not Responding
188401	51	Some SKUs were missing
188401	53	Late Delivery
188396	53	Late Delivery
188367	30	Retailer's Location is Incorrect
188366	30	Retailer's Location is Incorrect

After that I grouped the reasons to see occurrence of each reason using the following query.

```
SELECT cancellation_reason, count(order_id) as total
FROM cancellation_reasons_list_updated
GROUP BY cancellation_reason
ORDER BY total DESC;
```

The output looks something like this.

	cancellation_reason	total
▶	SKU's NA	1042
	Late Delivery	605
	Shop Close Other	472
	no service area Other	434
	cash issue Other	325
	Shop Closed And Customer Not Responding	270
	Driver got late	209

As you can see the same problems were being treated as separate one because it was written in different text. To solve this issue I exported my new table as CSV and used python to further solve the situation

The following piece of code was used to read the csv file.

```
df1 = pd.read_csv('/content/drive/MyDrive/Dastgyr_case_study/Dastgyr_data/Cancellation Reasons List Updated.csv', index_col=0)
df1
```

Let me take an example to explain what the code below does. In the cancellation_reason column there are two rows. One says shop closed and the other says shop is closed. Although both of them mean the same thing, they will be treated as different. So to cater this issue what I did is whenever there is a word shop in a sentence it will replace the entire sentence to shop closed. So the shop is closed turns into shop closed.

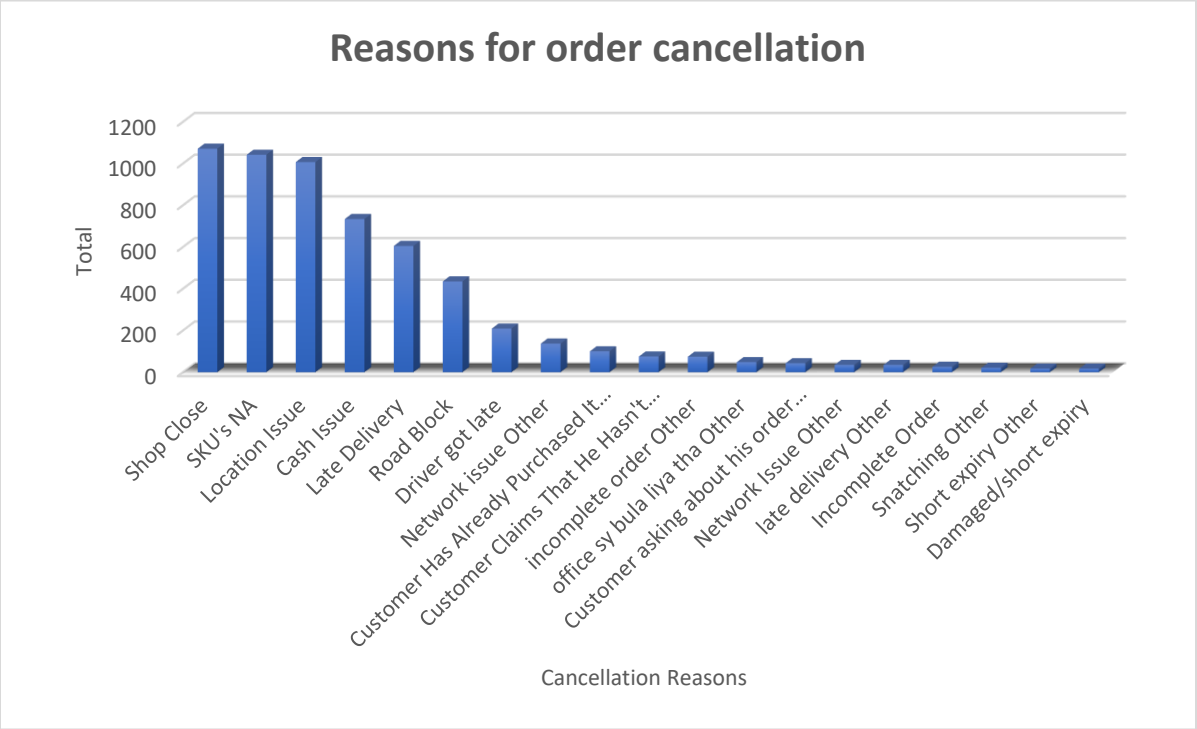
```
df1['cancellation_reason'] = df1.cancellation_reason.fillna('')
df1.loc[df1['cancellation_reason'].str.contains('shop', case=False), 'cancellation_reason'] = 'Shop Close'
df1.loc[df1['cancellation_reason'].str.contains('cash', case=False), 'cancellation_reason'] = 'Cash Issue'
df1.loc[df1['cancellation_reason'].str.contains('payment', case=False), 'cancellation_reason'] = 'Cash Issue'
df1.loc[df1['cancellation_reason'].str.contains('location', case=False), 'cancellation_reason'] = 'Location Issue'
df1.loc[df1['cancellation_reason'].str.contains('area', case=False), 'cancellation_reason'] = 'Location Issue'
df1.loc[df1['cancellation_reason'].str.contains('road', case=False), 'cancellation_reason'] = 'Road Block'
```

```
# temp1 = df1.loc[df1['cancellation_reason_id'] == 63]
temp2 = df1['cancellation_reason'].value_counts()
temp2.to_frame()
```

After executing the above statement I received the following output which is much refined and has catered to the different text problems.

index	cancellation_reason
Shop Close	1071
SKU's NA	1042
Location Issue	1007
Late Delivery	881
Cash Issue	734
Road Block	435
Network issue Other	137

More than 70% of the cancellations are due to top 5 reasons



The major reasons for order cancellation can be seen above.

The first one is that shop is closed when the order was being delivered. The solution could be that a could be decided upon when ordering or the user could be informed on call that he would be receiving his order within a few hours.

The second issue is due to the item being out of stock. One solution could be to find high-demand products and recommend their alternatives if there are any or ask them to pre-order the product and supply them when the inventory comes back in stock.

The third issue can be solved by creating a pickup point near such remote locations. The delivery is dropped at the pickup point where it is collected by the customer.

Late delivery could be due to remote location reasons or a lesser number of drivers. This could be achieved by hiring more drivers.

TASK 3: Finding in-demand SKUs.

This task was quite difficult as relevant data was missing but I tried to find it from the given data. I used the SKU table for this task. The first was to create the table and load the values

The statement below was used to create SKU table

```
CREATE TABLE sku(
    id int not null,
    product_id int not null,
    price int,
    discount int,
    stock int,
    is_active ENUM('TRUE', 'FALSE') NOT NULL DEFAULT 'FALSE',
    is_stock ENUM('TRUE', 'FALSE') NOT NULL DEFAULT 'FALSE',
    weight int,
    unit int,
    all_types int,
    is_deal ENUM('TRUE', 'FALSE') NOT NULL DEFAULT 'FALSE'

);
```

The data is then loaded into the table

```
LOAD DATA LOCAL INFILE "D:/Dastgyr_data/SKU.csv"
INTO TABLE sku
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

The first table below tells how many times a product was in stock and the second table tells how many times the product was out of stock.

```
CREATE table true_sku(
    product_id int not null,
    total_true int
);

CREATE table false_sku(
    product_id int not null,
    total_false int
);
```

This is how entered data in the above two tables

```
INSERT INTO false_sku
SELECT product_id, count(product_id) as total_false
FROM sku
WHERE is_stock != true
Group By product_id;

INSERT INTO true_sku
SELECT product_id, count(product_id) as total_true
FROM sku
WHERE is_stock = true
Group By product_id;
```

Lastly executed the below query to see the difference in how many times it was in stock and how many times it was out of stock.

```
SELECT ts.product_id, ts.total_true, fs.total_false, (ts.total_true - fs.total_false) as difference
FROM true_sku as ts
LEFT JOIN false_sku as fs
ON ts.product_id = fs.product_id
ORDER BY difference ASC;
```

These are some of the in-demand products.

	product_id	total_true	total_false	difference
	88	1	4	-3
	89	5	8	-3
	144	5	7	-2
	459	1	3	-2
	369	2	4	-2
	157	4	5	-1
	252	6	7	-1
	412	4	5	-1
	365	2	3	-1
	310	1	2	-1
	137	2	3	-1

Result 19

I did a similar approach using python as well and received similar results.

```
[ ] df4 = pd.read_csv('/content/drive/MyDrive/Dastgyr_case_study/Dastgyr_data/SKU.csv', index_col=0)
    df5 = df4[(df4['Is Stock'] == True) & (df4['Is Active'] == True)]
    df6 = df4[(df4['Is Stock'] == False) & (df4['Is Active'] == True)]
```

```
    t2 = df5['Product ID'].value_counts()
    t2.to_frame()
```

+ Code + Text

```
[ ] t3 = df6['Product ID'].value_counts()
    t3.to_frame()
```

```
    pd.merge(t2, t3, left_index=True, right_index=True)
```