Order Success Rate Analysis Project

December 22, 2024

[1]:
```
import pandas as pd
import mysql.connector
import matplotlib.pyplot as plt
from sqlalchemy import create_engine

pd.set_option('display.float_format', '{:,.2f}'.format)
```

[50]:
```
# Connect to MySQL database
db_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="0000"
)

cursor = db_connection.cursor()



create_database=""" CREATE DATABASE Talbat
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

"""


cursor.execute(create_database)

# Commit the transaction
db_connection.commit()




cursor.close()
db_connection.close()
```

[51]:
```
# Step 1: Create a connection to the database using SQLAlchemy
db_url = 'mysql+mysqlconnector://root:0000@localhost/Talbat'
engine = create_engine(db_url)

xls = pd.ExcelFile('Talbat Case Study Dataset.xlsx', engine='openpyxl')
for i in xls.sheet_names[0:3]:


# Step 2: Read the Excel file into a pandas DataFrame
    df=pd.read_excel('Talbat Case Study Dataset.xlsx',sheet_name=i,␣
 ↪engine='openpyxl')

    df = df.where(pd.notna(df), None)



# Step 3: Insert the DataFrame into the database table in bulk


    if i =="fct_order":
        i="orders"
    df.to_sql(i, con=engine, if_exists='replace', index=False, method='multi',␣
 ↪chunksize=1000)



engine.dispose()

print("Data imported successfully!")
```

```
Data imported successfully!
```

[4]:
```
# Establish MySQL Connection
db_connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="0000",
    database="Talbat"
)

cursor = db_connection.cursor()

# Function to execute SQL queries and return results as a DataFrame
def execute_query(query):
    cursor.execute(query)
    result = cursor.fetchall()
    columns = [col[0] for col in cursor.description]
```

```
    return pd.DataFrame(result, columns=columns)
```

[8]:
```
overview_query = "SELECT * FROM orders LIMIT 10;"  # Retrieve first 10 rows for␣
↪overview
df_overview = execute_query(overview_query)
df_overview
```

[8]:

	order_id	order_time	vendor_id	analytical_customer_id	\
0	544145594	2021-09-01 10:41:53	609508	6zWSp9yqnEQuq4N6ykRgbbdQ	
1	544478492	2021-09-01 18:17:52	602440	TtmFFw26sQR26nBE2RykhxQA	
2	544358032	2021-09-01 15:56:56	709071	7pB4MC2aBPRdajLH2wPvIvAA	
3	544366924	2021-09-01 16:09:09	607670	AFedsf0e-lSZer_X0eR4VE-g	
4	544372362	2021-09-01 16:16:32	616421	JDYqF36aULSEaWsz604DuuRw	
5	544150522	2021-09-01 10:51:28	614199	7JA09h4C9NTbCYXV4EFncRWQ	
6	544376183	2021-09-01 16:21:37	739898	4RIyq_6K1vRHKS1n6SDQK1Lg	
7	544467338	2021-09-01 18:06:40	735754	uJkW3IHmcxT5m8qhHNHLfIpg	
8	545200868	2021-09-02 17:08:46	748514	8wxoxuXWxxTvWXj2XDO3irLA	
9	545186719	2021-09-02 16:52:50	735735	VcKa651esrQXedBe1p78LQhA	

	is_acquisition	is_successful	reason	sub_reason	owner	delivery_arrangement	\
0	0	1	None	None	None	TGO	
1	0	1	None	None	None	TGO	
2	0	1	None	None	None	TGO	
3	0	1	None	None	None	TGO	
4	0	1	None	None	None	TGO	
5	0	1	None	None	None	TGO	
6	0	1	None	None	None	TGO	
7	0	1	None	None	None	TGO	
8	0	1	None	None	None	TGO	
9	0	1	None	None	None	TGO	

	…	is_affordable_freedelivery	is_affordable_item	is_affordable_gem	\
0	…	0	0	0	
1	…	0	0	0	
2	…	0	0	0	
3	…	0	0	0	
4	…	0	0	0	
5	…	0	0	0	
6	…	0	0	0	
7	…	0	1	0	
8	…	0	0	0	
9	…	0	0	0	

	is_affordable_restaurant	is_affordable_voucher	is_affordable	\
0	0	0	0	
1	0	0	0	
2	0	0	0	

```
3                         0                       0              0
4                         0                       0              0
5                         0                       0              0
6                         0                       0              0
7                         0                       0              1
8                         0                       0              0
9                         0                       0              0

   affordability_amt_total  city_cluster   city              zone
0                     None         Cairo  Cairo       Mohandiseen
1                     None         Cairo  Cairo  Cairo_city_centre
2                     None         Cairo  Cairo     Tagamoa 5 east
3                     None         Cairo  Cairo  Cairo_city_centre
4                     None         Cairo  Cairo             Maadi
5                     None         Cairo  Cairo  Cairo_city_centre
6                     None         Cairo  Cairo        Tagammoa 5
7                    38.00         Cairo  Cairo             Maadi
8                     None         Cairo  Cairo      El rehab city
9                     None         Cairo  Cairo              Giza

[10 rows x 31 columns]
```

[ ]:

1 1. Overall Success Rate Analysis

[16]:
```
success_rate_query = "SELECT AVG(is_successful) AS overall_success_rate FROM␣
↪orders;"
overall_success_rate = execute_query(success_rate_query)
print(f"\nOverall Success Rate:␣
↪{overall_success_rate['overall_success_rate'][0] * 100:.2f}%")
```

```
Overall Success Rate: 83.55%
```

2 2. Reason Analysis for Unsuccessful Orders

[19]:
```
reason_analysis_query = """
    SELECT reason, COUNT(*) AS count, ROUND(COUNT(*) * 100 / (SELECT COUNT(*)␣
↪FROM orders WHERE is_successful = 0), 2) AS percentage
    FROM orders
    WHERE is_successful = 0
    GROUP BY reason
```

```
    ORDER BY count DESC;
"""
reason_analysis = execute_query(reason_analysis_query)
print("\nReason Analysis for Unsuccessful Orders:")
print(reason_analysis)
```

```
Reason Analysis for Unsuccessful Orders:
                            reason  count percentage
0                   ONLINE_PAYMENT   7394      37.50
1            CUSTOMER_MISTAKE_ERROR   6161      31.25
2                      FRAUD_PRANK   2096      10.63
3                 ITEM_UNAVAILABLE    637       3.23
4               VENDOR_NO_RESPONSE    460       2.33
5                    LATE_DELIVERY    385       1.95
6                       NO_COURIER    347       1.76
7      ADDRESS_INCOMPLETE_MISSTATED    272       1.38
8           UNABLE_TO_FIND_CUSTOMER    257       1.30
9                  VENDOR_TOO_BUSY    215       1.09
10                   VENDOR_CLOSED    214       1.09
11                   REASON_UNKNOWN    183       0.93
12                TECHNICAL_PROBLEM    164       0.83
13               VENDOR_UNREACHABLE    158       0.80
14            OUTSIDE_DELIVERY_AREA    138       0.70
15            MENU_ACCOUNT_SETTINGS    131       0.66
16                 DUPLICATE_ORDER     97       0.49
17              VOUCHER_NOT_APPLIED     88       0.45
18            DELIVERY_ETA_TOO_LONG     77       0.39
19  ORDER_MODIFICATION_NOT_POSSIBLE     74       0.38
20            FOOD_QUALITY_SPILLAGE     59       0.30
21      WRONG_ORDER_ITEMS_DELIVERED     38       0.19
22         UNPROFESSIONAL_BEHAVIOUR     30       0.15
23         CONTENT_WRONG_MISLEADING     16       0.08
24                      BAD_WEATHER      9       0.05
25                 COURIER_ACCIDENT      9       0.05
26                  NEVER_DELIVERED      2       0.01
27              COURIER_UNREACHABLE      2       0.01
28            OUTSIDE_SERVICE_HOURS      1       0.01
29        CARD_READER_NOT_AVAILABLE      1       0.01
30            PLATFORM_BAD_LOCATION      1       0.01
```

Observation: The primary reasons for failure are:

- **ONLINE_PAYMENT**: 37.5%
- **CUSTOMER_MISTAKE_ERROR**: 31.25%
- **FRAUD_PRANK**: 10.63%

Recommendation:

- **ONLINE_PAYMENT**: Optimize the payment process, provide alternative payment methods, and add real-time assistance for failed payments.
- **CUSTOMER_MISTAKE_ERROR**: Enhance UI/UX for clarity, add order review steps, and educate users with tooltips or FAQs.
- **FRAUD_PRANK**: Strengthen fraud detection mechanisms and consider penalty systems for prank orders.

3 3. Time-based Analysis

3.1 Group by hour and calculate success rate

[9]:
```
hourly_success_rate_query = """
    SELECT
        EXTRACT(HOUR FROM order_time) AS hour_of_day,
        AVG(is_successful)*100 AS average_success_rate
    FROM
        orders
    GROUP BY
        hour_of_day;
"""
hourly_success_rate = execute_query(hourly_success_rate_query)
hourly_success_rate["average_success_rate"]=pd.
 ↪to_numeric(hourly_success_rate["average_success_rate"])
hourly_success_rate
```

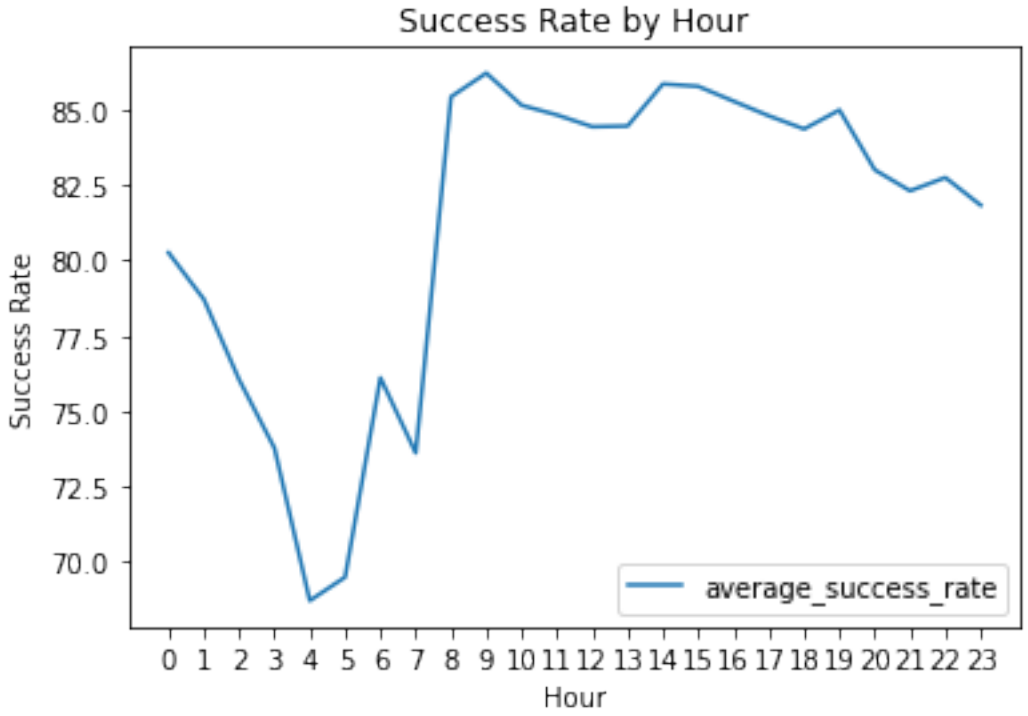
[9]:

	hour_of_day	average_success_rate
0	0	80.25
1	1	78.71
2	2	76.04
3	3	73.78
4	4	68.72
5	5	69.50
6	6	76.11
7	7	73.62
8	8	85.42
9	9	86.21
10	10	85.14
11	11	84.82
12	12	84.43
13	13	84.45
14	14	85.84
15	15	85.77
16	16	85.27
17	17	84.79
18	18	84.35

19	19	84.98
20	20	83.00
21	21	82.30
22	22	82.74
23	23	81.82

3.2 Visualization

[10]:
```
hourly_success_rate.plot(x="hour_of_day",y="average_success_rate",kind="line")
plt.xlabel("Hour")
plt.ylabel("Success Rate")
plt.title("Success Rate by Hour")
plt.xticks(range(24))
plt.show()
```



[23]:
```
hourly_late_night_reasons_query = """
    SELECT
    reason,
    COUNT(*) AS count_of_reasons,
    Round((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM orders WHERE EXTRACT(HOUR␣
 ↪FROM order_time) BETWEEN 2 AND 6  and reason is not null),2) AS␣
 ↪percentage_of_total_reasons
```

```
FROM
    orders
WHERE
    EXTRACT(HOUR FROM order_time) BETWEEN 2 AND 6 and reason is not null
GROUP BY
    reason
ORDER BY
    count_of_reasons DESC;
"""

# More descriptive variable name
hourly_late_night_reason_counts_and_percentages =␣
 ↪execute_query(hourly_late_night_reasons_query)

# Print the results
print("\nHourly Last Night Reason Counts and Percentages:")
hourly_late_night_reason_counts_and_percentages[["count_of_reasons"␣
 ↪,"percentage_of_total_reasons"]]= \
    hourly_late_night_reason_counts_and_percentages[["count_of_reasons"␣
 ↪,"percentage_of_total_reasons"]].astype(float)
hourly_late_night_reason_counts_and_percentages
```

```
Hourly Last Night Reason Counts and Percentages:
```

[23]:

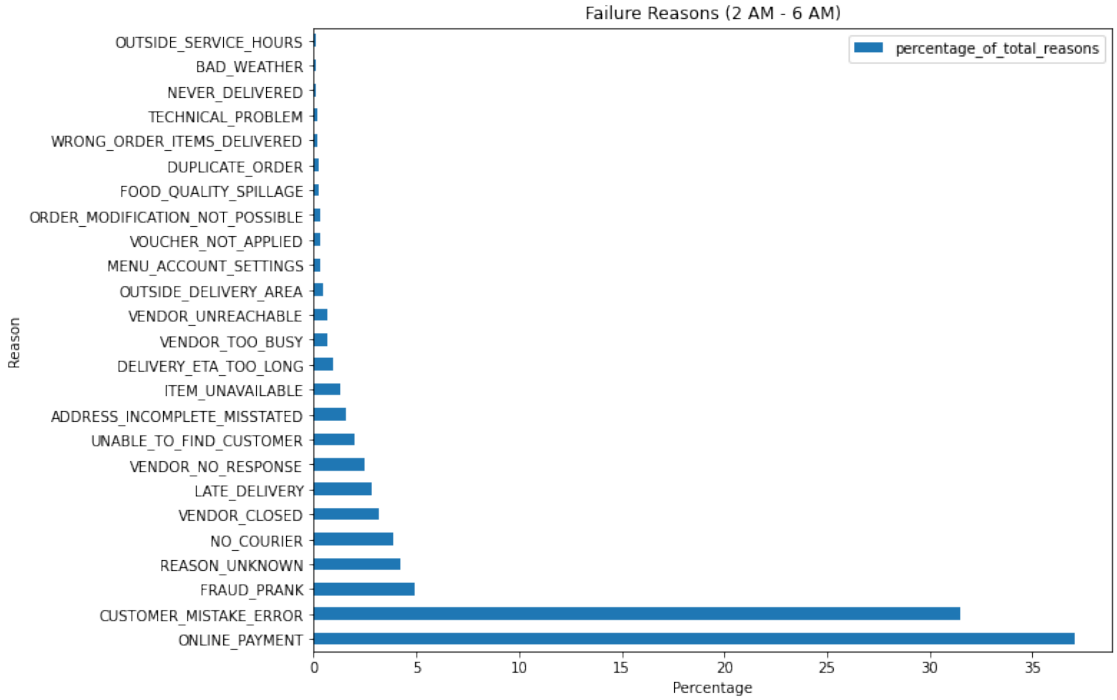
	reason	count_of_reasons	\
0	ONLINE_PAYMENT	428.00	
1	CUSTOMER_MISTAKE_ERROR	364.00	
2	FRAUD_PRANK	57.00	
3	REASON_UNKNOWN	49.00	
4	NO_COURIER	45.00	
5	VENDOR_CLOSED	37.00	
6	LATE_DELIVERY	33.00	
7	VENDOR_NO_RESPONSE	29.00	
8	UNABLE_TO_FIND_CUSTOMER	23.00	
9	ADDRESS_INCOMPLETE_MISSTATED	18.00	
10	ITEM_UNAVAILABLE	15.00	
11	DELIVERY_ETA_TOO_LONG	11.00	
12	VENDOR_TOO_BUSY	8.00	
13	VENDOR_UNREACHABLE	8.00	
14	OUTSIDE_DELIVERY_AREA	5.00	
15	MENU_ACCOUNT_SETTINGS	4.00	
16	VOUCHER_NOT_APPLIED	4.00	
17	ORDER_MODIFICATION_NOT_POSSIBLE	4.00	
18	FOOD_QUALITY_SPILLAGE	3.00	
19	DUPLICATE_ORDER	3.00	
20	WRONG_ORDER_ITEMS_DELIVERED	2.00	

```
21                TECHNICAL_PROBLEM              2.00
22                  NEVER_DELIVERED              1.00
23                      BAD_WEATHER              1.00
24           OUTSIDE_SERVICE_HOURS               1.00

    percentage_of_total_reasons
0                         37.06
1                         31.52
2                          4.94
3                          4.24
4                          3.90
5                          3.20
6                          2.86
7                          2.51
8                          1.99
9                          1.56
10                         1.30
11                         0.95
12                         0.69
13                         0.69
14                         0.43
15                         0.35
16                         0.35
17                         0.35
18                         0.26
19                         0.26
20                         0.17
21                         0.17
22                         0.09
23                         0.09
24                         0.09
```

[ ]:

[22]:
```
# Bar chart for reasons (2 AM to 6 AM)
hourly_late_night_reason_counts_and_percentages.
 ↪plot(x="reason",y="percentage_of_total_reasons",kind='barh', figsize=(10, 8))
plt.title('Failure Reasons (2 AM - 6 AM)')
plt.xlabel('Percentage')
plt.ylabel('Reason')
plt.show()
```



[25]:
```
hourly_morning_reasons_query = """
    SELECT
    reason,
    COUNT(*) AS count_of_reasons,
    Round((COUNT(*) * 100.0) / (SELECT COUNT(*) FROM orders WHERE EXTRACT(HOUR␣
 ↪FROM order_time) BETWEEN 9 AND 20 and reason is not null),2) AS␣
 ↪percentage_of_total_reasons
FROM
    orders
WHERE
    EXTRACT(HOUR FROM order_time) BETWEEN 9 AND 20 and reason is not null
GROUP BY
    reason
ORDER BY
    count_of_reasons DESC;
"""

hourly_morning_reason_counts_and_percentages =␣
 ↪execute_query(hourly_morning_reasons_query)


print("\nHourly Morning Reason Counts and Percentages:")
hourly_morning_reason_counts_and_percentages[["count_of_reasons"␣
 ↪,"percentage_of_total_reasons"]]= \
```

```
    hourly_morning_reason_counts_and_percentages[["count_of_reasons"␣
↪,"percentage_of_total_reasons"]].astype(float)
hourly_morning_reason_counts_and_percentages
```

Hourly Morning Reason Counts and Percentages:

[25]:

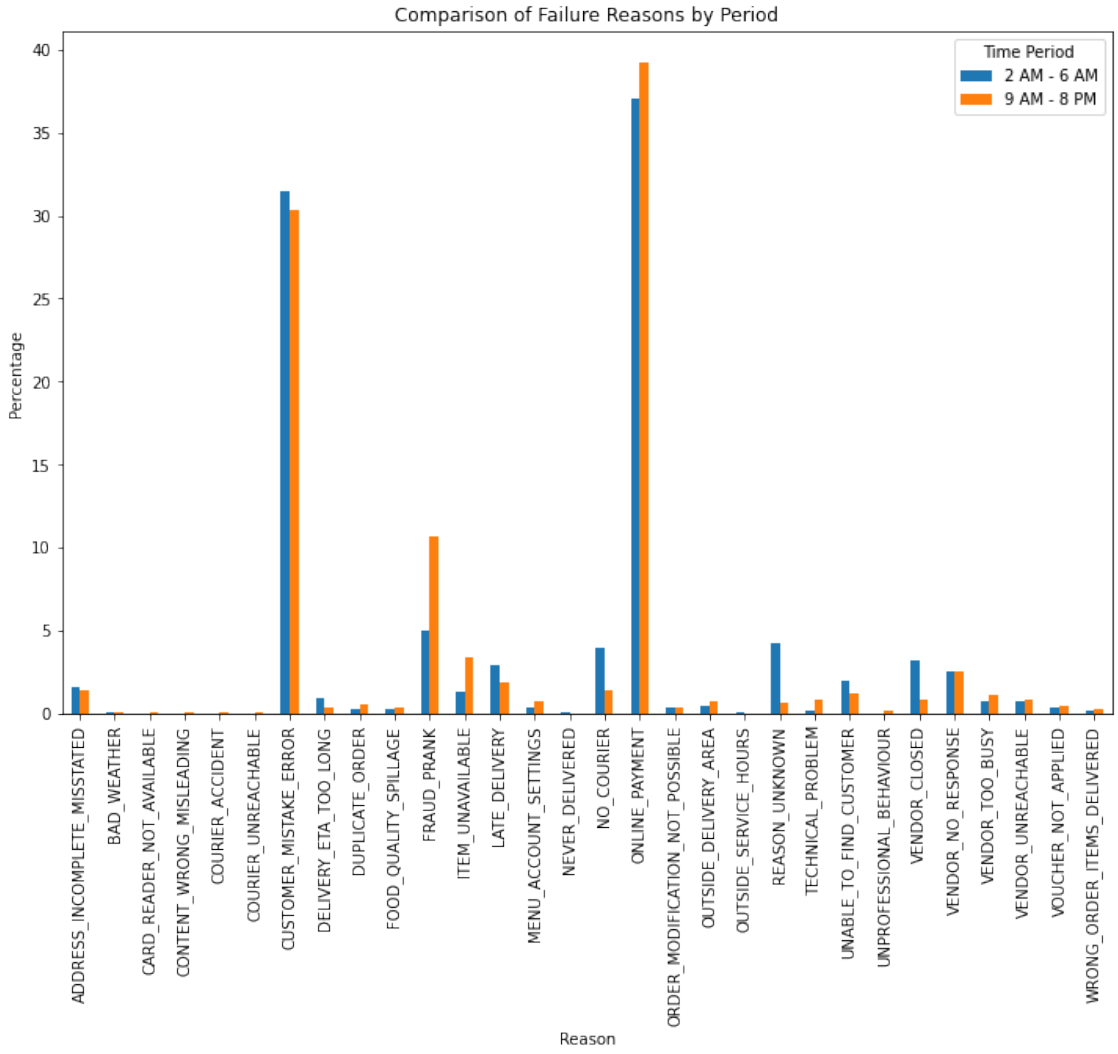
	reason	count_of_reasons	percentage_of_total_reasons
0	ONLINE_PAYMENT	4,898.00	39.20
1	CUSTOMER_MISTAKE_ERROR	3,794.00	30.36
2	FRAUD_PRANK	1,327.00	10.62
3	ITEM_UNAVAILABLE	421.00	3.37
4	VENDOR_NO_RESPONSE	311.00	2.49
5	LATE_DELIVERY	236.00	1.89
6	ADDRESS_INCOMPLETE_MISSTATED	175.00	1.40
7	NO_COURIER	172.00	1.38
8	UNABLE_TO_FIND_CUSTOMER	152.00	1.22
9	VENDOR_TOO_BUSY	143.00	
10	TECHNICAL_PROBLEM	106.00	
11	VENDOR_UNREACHABLE	99.00	
12	VENDOR_CLOSED	96.00	
13	OUTSIDE_DELIVERY_AREA	92.00	
14	MENU_ACCOUNT_SETTINGS	84.00	
15	REASON_UNKNOWN	79.00	
16	DUPLICATE_ORDER	67.00	
17	VOUCHER_NOT_APPLIED	55.00	
18	ORDER_MODIFICATION_NOT_POSSIBLE	42.00	
19	FOOD_QUALITY_SPILLAGE	41.00	
20	DELIVERY_ETA_TOO_LONG	37.00	
21	WRONG_ORDER_ITEMS_DELIVERED	28.00	
22	UNPROFESSIONAL_BEHAVIOUR	16.00	
23	CONTENT_WRONG_MISLEADING	10.00	
24	COURIER_ACCIDENT	7.00	
25	BAD_WEATHER	5.00	
26	COURIER_UNREACHABLE	1.00	
27	CARD_READER_NOT_AVAILABLE	1.00	

```
9                       1.14
10                      0.85
11                      0.79
12                      0.77
13                      0.74
14                      0.67
15                      0.63
16                      0.54
17                      0.44
18                      0.34
19                      0.33
20                      0.30
21                      0.22
22                      0.13
23                      0.08
24                      0.06
25                      0.04
26                      0.01
27                      0.01
```

[ ]:

[35]:
```
reasons_comparison = pd.DataFrame({
    '2 AM - 6 AM': hourly_late_night_reason_counts_and_percentages.
 ↪set_index("reason")["percentage_of_total_reasons"],
    '9 AM - 8 PM': hourly_morning_reason_counts_and_percentages.
 ↪set_index("reason")["percentage_of_total_reasons"]
}).fillna(0)
reasons_comparison
# reasons_comparison = hourly_morning_reason_counts_and_percentages.
 ↪merge(hourly_late_night_reason_counts_and_percentages, on='reason')

reasons_comparison.plot(kind='bar', figsize=(12, 8))
plt.title('Comparison of Failure Reasons by Period')
plt.xlabel('Reason')
plt.ylabel('Percentage')
plt.legend(title='Time Period')
plt.show()
```



[ ]:

Observation:

- Success rates are lowest during late-night hours (2 AM–6 AM) and highest during the day (8 AM–8 PM).

Observation: - If we exclude the top three reasons for failure, we observe distinct differences in the types of problems occurring at night compared to the morning.

- During the morning (9 AM to 8 PM), issues like ITEM_UNAVAILABLE, VENDOR_NO_RESPONSE, and LATE_DELIVERY are more prevalent. These problems are likely caused by the overwhelming volume of orders during peak hours.
- At night (2 AM to 6 AM), unique challenges such as REASON_UNKNOWN, NO_COURIER, and VENDOR_CLOSED are more frequent. These issues may stem from reduced operational capacity and limited resources available during late hours.

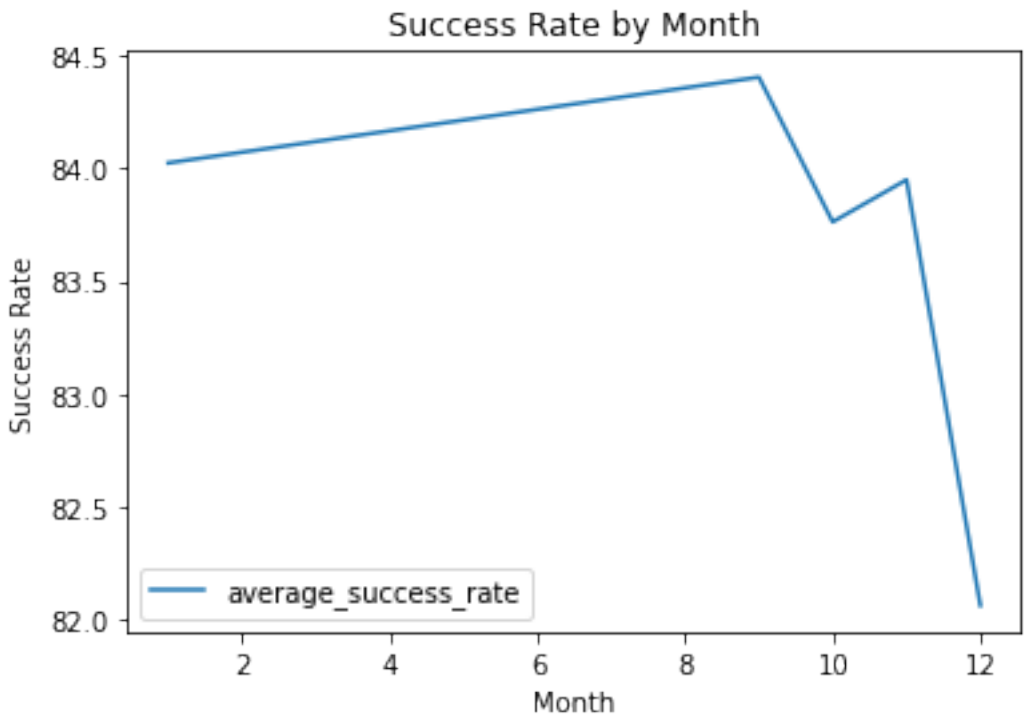
3.3 Group by month and calculate success rate

[36]:
```
month_success_rate_query = """
    SELECT
        EXTRACT(MONTH FROM order_time) AS month,
        AVG(is_successful)*100 AS average_success_rate
    FROM
        orders
    GROUP BY
        month;
"""
monthly_success_rate = execute_query(month_success_rate_query)
monthly_success_rate["average_success_rate"]=pd.
 ↪to_numeric(monthly_success_rate["average_success_rate"])
monthly_success_rate
```

[36]:

	month	average_success_rate
0	1	84.02
1	9	84.40
2	10	83.76
3	11	83.95
4	12	82.06

[38]:
```
monthly_success_rate.plot(x="month",y="average_success_rate",kind="line")
plt.xlabel("Month")
plt.ylabel("Success Rate")
plt.title("Success Rate by Month")
plt.show()
```



4 4. Geographical Analysis

4.1 Success rate by City

[20]:
```
geo_success_rate_query = """
    SELECT City, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY City;
"""
geo_success_rate = execute_query(geo_success_rate_query)
print("\nGeographical Success Rate Analysis:")
print(geo_success_rate)
```

```
Geographical Success Rate Analysis:
                    City success_rate
0                   None       0.8345
1   Al Mahallah Al Kubra       0.6885
2             Alexandria       0.8410
3                 Assiut       0.6619
4                  Banha       0.7743
5                  Cairo       0.8430
```

```
6              Damanhour       0.7982
7               Damietta       0.8421
8          Damietta New       0.9100
9              El Gouna       0.8000
10             Hurghada       0.8517
11             Ismailia       0.9023
12             Mansoura       0.8332
13                Minya       0.9355
14          North Coast       0.8438
15            Port Said       0.7425
16           Ras El Bar       0.8923
17      Shebeen El Koom       0.8667
18                 Suez       0.8133
19                Suhag       0.7820
20               Talkha       0.8468
21                Tanta       0.8252
22              Zagazig       0.7768
```

4.2 Success rate by City Cluster

[37]:
```
cluster_success_rate_query = """
    SELECT city_cluster, AVG(is_successful)*100 AS success_rate
    FROM orders
    GROUP BY city_cluster;
"""
geo_success_rate = execute_query(cluster_success_rate_query)
print("\n City Cluster Success Rate Analysis:")
print(geo_success_rate)
```

```
 City Cluster Success Rate Analysis:
  city_cluster success_rate
1         Alex      84.0971
2        Cairo      84.3031
3        Delta      82.0965
4          ESM      74.4385
```

4.3 ESM Order Failure Insights

[44]:
```
esm_failure_reasons_query = """
    SELECT reason, ROUND(COUNT(*) * 100 / (SELECT COUNT(*) FROM orders where␣
 ↪city_cluster="ESM" and is_successful=0), 2) AS percentage
    FROM orders
    where city_cluster="ESM"
    GROUP BY reason;
```

```
"""
esm_failure_reasons_df = execute_query(esm_failure_reasons_query)
print("\nESM Failure Reason Analysis:")
print(esm_failure_reasons_df)
```

```
ESM Failure Reason Analysis:
                            reason percentage
1      ADDRESS_INCOMPLETE_MISSTATED       0.21
2            CUSTOMER_MISTAKE_ERROR      15.13
3             DELIVERY_ETA_TOO_LONG       0.07
4                   DUPLICATE_ORDER       0.14
5             FOOD_QUALITY_SPILLAGE       0.21
6                       FRAUD_PRANK      41.98
7                  ITEM_UNAVAILABLE       1.19
8                     LATE_DELIVERY       0.49
9            MENU_ACCOUNT_SETTINGS       0.21
10                   ONLINE_PAYMENT      35.01
11  ORDER_MODIFICATION_NOT_POSSIBLE       0.49
12           OUTSIDE_DELIVERY_AREA       0.28
13                  REASON_UNKNOWN       1.67
14               TECHNICAL_PROBLEM       0.42
15          UNABLE_TO_FIND_CUSTOMER       0.28
16                    VENDOR_CLOSED       0.35
17               VENDOR_NO_RESPONSE       0.49
18                  VENDOR_TOO_BUSY       0.49
19               VENDOR_UNREACHABLE       0.07
20              VOUCHER_NOT_APPLIED       0.77
21      WRONG_ORDER_ITEMS_DELIVERED       0.07
```

[ ]:

Observation:

- **Cities** like **Minya** (94%) and **Damietta New** (91%) show high success rates, while others like **Assiut** (66%) and Al Mahallah Al Kubra (69%) lag behind.
- Success rates vary significantly by cluster (e.g., **Cairo**: 84%, **ESM**: 74%).-
- The primary reason for order failures in ESM is FRAUD_PRANK

5 5. Payment Method Analysis

[9]:
```
payment_method_analysis_query = """
    SELECT payment_method, AVG(is_successful)*100 AS success_rate
    FROM orders
    GROUP BY payment_method;
"""
```

```
payment_method_analysis = execute_query(payment_method_analysis_query)
print("\nPayment Method Success Rates:")
print(payment_method_analysis)
```

```
Payment Method Success Rates:
     payment_method success_rate
0              Cash      87.8689
1     Online - Card      73.8172
2    Online - Mixed      49.5096
3  Online - Talabat      83.4792
```

Observation:

- Success rates are highest for **Cash** (88%) and lowest for **Online - Mixed** (50%).

[ ]:

6 6. Platform-based Analysis

[11]:
```
platform_analysis_query = """
    SELECT platform, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY platform;
"""
platform_analysis = execute_query(platform_analysis_query)
print("\nPlatform-Based Success Rate Analysis:")
print(platform_analysis)
```

[11]:
```
      platform success_rate
0      android       0.8348
1  desktop web       0.8834
2          iOS       0.8352
3   mobile web       0.8623
```

Observation:

- Desktop web (88%) performs better than Android (83%) and iOS (84%).

[15]:
```
orders_by_platform_query = """
    SELECT platform, count(order_id) AS total_orders, Round(count(order_id)*100/
 ↪(select count(*) from orders),2) as Percent
    FROM orders
    GROUP BY platform
    order by total_orders desc;
"""
```

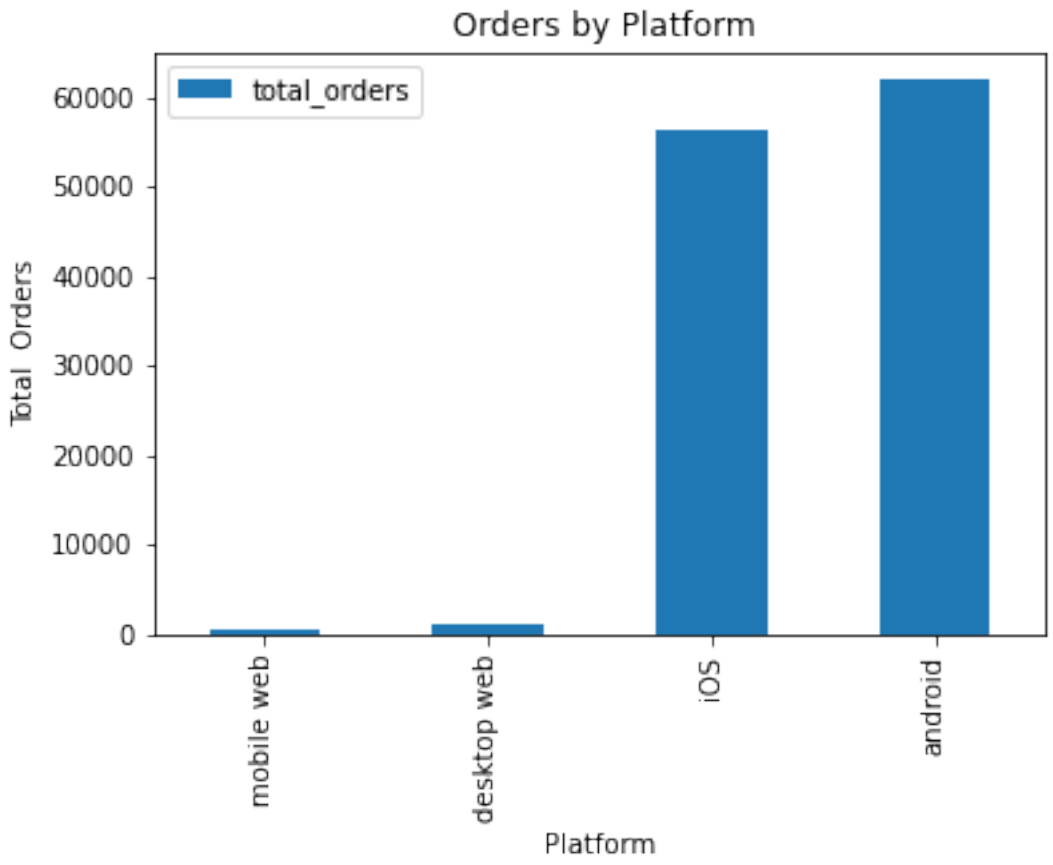
```
orders_per_platform_df = execute_query(orders_by_platform_query)

print("\nPlatform Order Counts:")
print(orders_per_platform_df)
```

```
Platform Order Counts:
      platform  total_orders Percent
0   mobile web           581    0.48
1  desktop web           986    0.82
2          iOS         56454   47.09
3      android         61864   51.60
```

[19]:
```
orders_per_platform_df.iloc[:,0:2].plot(x='platform', y='total_orders',␣
↪kind='bar')
plt.title(' Orders by Platform')
plt.xlabel('Platform')
plt.ylabel('Total  Orders')
```

[19]:
```
Text(0, 0.5, 'Total  Orders')
```



observation : Most of our orders originate from mobile apps (Android and iOS)

7 7. Product/Service-based Analysis

7.1 Success rate by vertical_class

[25]:
```
product_analysis_query = """
    SELECT vertical_class, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY vertical_class
    order by success_rate desc;
"""
product_analysis = execute_query(product_analysis_query)
print("\nProduct/Service-Based Success Rate Analysis:")
print(product_analysis)
```

```
Product/Service-Based Success Rate Analysis:
  vertical_class success_rate
0       non-food       0.8400
1           food       0.8349
```

7.2 Success rate by vertical

[27]:
```
product_analysis_query = """
    SELECT vertical_class,vertical, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY vertical_class,vertical
    order by vertical_class,success_rate desc;
"""
product_analysis = execute_query(product_analysis_query)
print("\nProduct/Service-Based Success Rate Analysis:")
print(product_analysis)
```

```
Product/Service-Based Success Rate Analysis:
  vertical_class     vertical success_rate
0           food         food       0.8349
1       non-food      flowers       1.0000
2       non-food     pet shop       0.8750
3       non-food      grocery       0.8408
4       non-food    cosmetics       0.8148
5       non-food     pharmacy       0.7558
6       non-food  electronics       0.7037
```

[49]:
```
vertical_total_orders_query = """
SELECT vertical_class,
       vertical,
       COUNT(*) AS total_orders,
       ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM orders), 2) AS␣
 ↪percentage_of_total_orders
FROM orders
GROUP BY vertical_class, vertical
ORDER BY vertical_class, total_orders DESC;
"""

# Execute Query and Store Results
vertical_total_orders_df = execute_query(vertical_total_orders_query)

# Print the DataFrame
print("\nProduct/Service-Based Success Rate Analysis:")
print(vertical_total_orders_df)
```

```
Product/Service-Based Success Rate Analysis:
  vertical_class     vertical  total_orders percentage_of_total_orders
0           food         food        105057                       87.63
1       non-food      grocery         14670                       12.24
2       non-food     pharmacy            86                        0.07
3       non-food  electronics            27                        0.02
4       non-food    cosmetics            27                        0.02
5       non-food     pet shop            16                        0.01
6       non-food      flowers             2                        0.00
```

Observation:

- Success rates are higher for **pet shop** (88%) and **grocery** (84%), but lower for **electronics** (70%) and **pharmacy** (76%)

[ ]:
```

```

8 8. Delivery Time and Distance Analysis

8.1 Delivery delay and distance analysis

[34]:
```
delivery_delay_query = """
    SELECT order_delay, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY order_delay;
"""
dropoff_distance_query = """
```

```
    SELECT dropoff_distance_manhattan, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY dropoff_distance_manhattan;
"""

delivery_delay_analysis = execute_query(delivery_delay_query)
dropoff_distance_analysis = execute_query(dropoff_distance_query)

print("\nDelivery Delay Analysis:")
print(delivery_delay_analysis)
print("\nDropoff Distance Analysis:")
print(dropoff_distance_analysis)
```

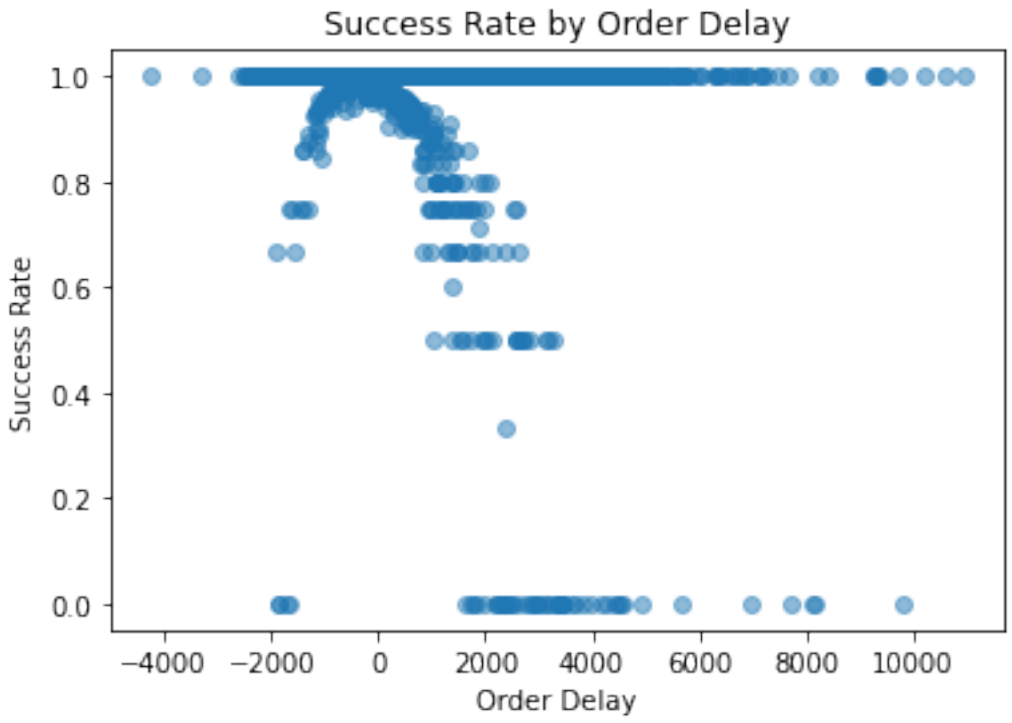
```
Delivery Delay Analysis:
      order_delay success_rate
0             nan       0.5608
1       -4,238.00       1.0000
2       -3,324.00       1.0000
3       -2,610.00       1.0000
4       -2,532.00       1.0000
…             …             …
4584     9,732.00       1.0000
4585     9,824.00       0.0000
4586    10,222.00       1.0000
4587    10,630.00       1.0000
4588    10,939.00       1.0000

[4589 rows x 2 columns]

Dropoff Distance Analysis:
      dropoff_distance_manhattan success_rate
0                            nan       0.6157
1                           0.00       0.0160
2                           1.00       1.0000
3                           2.00       1.0000
4                           3.00       0.8000
…                              …            …
6732                   55,926.00       1.0000
6733                   66,173.00       1.0000
6734                  203,530.00       1.0000
6735               13,240,805.00       0.0000
6736               16,364,616.00       1.0000

[6737 rows x 2 columns]
```

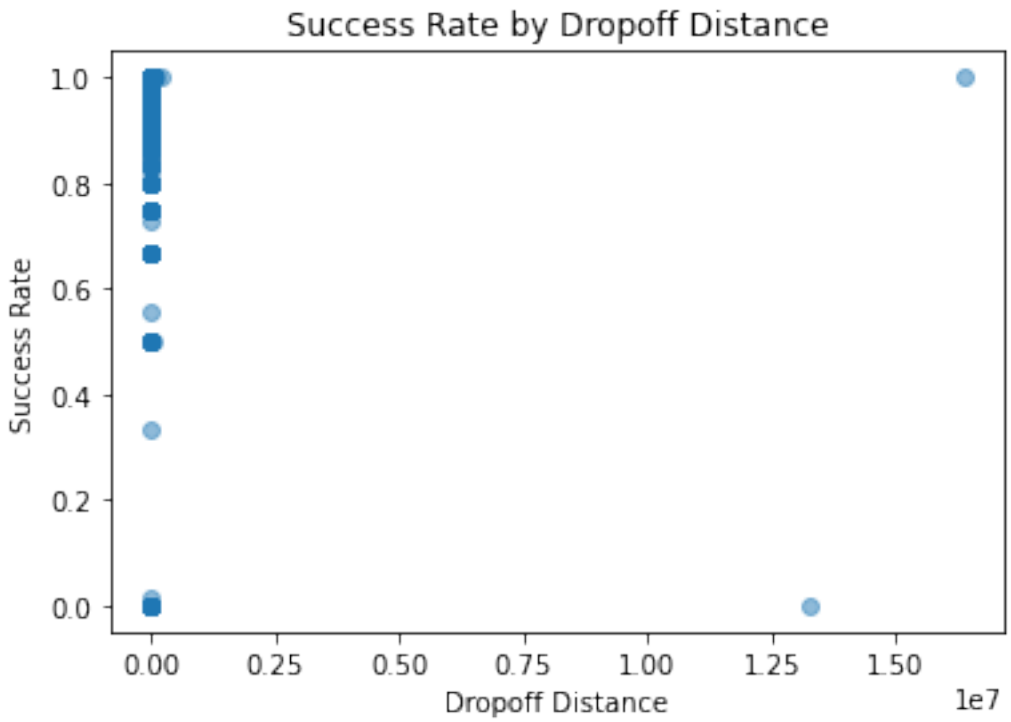
[35]:
```
# Delivery delay plot
plt.scatter(delivery_delay_analysis['order_delay'],␣
 ↪delivery_delay_analysis['success_rate'], alpha=0.5)
plt.xlabel("Order Delay")
plt.ylabel("Success Rate")
plt.title("Success Rate by Order Delay")
plt.show()
```



[ ]:
```

```

[36]:
```
plt.scatter(dropoff_distance_analysis['dropoff_distance_manhattan'],␣
 ↪dropoff_distance_analysis['success_rate'], alpha=0.5)
plt.xlabel("Dropoff Distance")
plt.ylabel("Success Rate")
plt.title("Success Rate by Dropoff Distance")
plt.show()
```



Observation

Delivery Delay: * **Short Delays, High Success:** Most orders with short delays are successful. * **Longer Delays, Lower Success:** As delivery delays increase, success rates tend to decrease.

Dropoff Distance: * **Short Distances, High Success:** Short-distance deliveries have high success rates. * **Longer Distances, Varied Success:** Longer distances can lead to more variability in success rates, likely due to factors like traffic, weather, and logistical challenges.

Overall, timely delivery and shorter distances are crucial for ensuring order success.

9 9. Customer Segmentation

9.1 Success rate for acquisition vs retention

[38]:
```
customer_segmentation_query = """
    SELECT is_acquisition, AVG(is_successful) AS success_rate
    FROM orders
    GROUP BY is_acquisition;
"""
customer_segmentation_success_rate = execute_query(customer_segmentation_query)
print("\nCustomer Segmentation Success Rates:")
print(customer_segmentation_success_rate)
```

```
Customer Segmentation Success Rates:
   is_acquisition success_rate
0               0       0.8302
1               1       1.0000
```

Observation:

- Retained customers (False: 83%) perform worse than acquired customers (True: 100%).

[ ]:

10 10. Sub-reason Analysis

[44]:
```
sub_reason_analysis_query = """
    SELECT sub_reason, COUNT(*) AS count,
    Round(count(*)*100/(SELECT  COUNT(*)
    FROM orders
    WHERE is_successful = 0 and sub_reason is not null),2) as percent
    FROM orders
    WHERE is_successful = 0 and sub_reason is not null
    GROUP BY sub_reason
    ORDER BY count DESC;
"""
sub_reason_analysis = execute_query(sub_reason_analysis_query)
print("\nSub-Reason Analysis:")
print(sub_reason_analysis)
```

```
Sub-Reason Analysis:
                                    sub_reason  count percent
0  Canceled by Customer - Before vendor accepted   5607   67.17
1                                 Fraud_SHIELD   2036   24.39
2              Changed mind - Mistake / Error    552    6.61
3                     Change Mind - No Reason     95    1.14
4                                   Fraud_SSC     57    0.68
5                     Unknown (NCR, RPS, POS)      1    0.01
```

Observation:

- Major sub-reasons include Canceled by Customer (e.g., before vendor acceptance).

[ ]:

11 11. Combined Analysis: Success Rate by Month and Hour

11.1 Group by month and hour, then calculate success rate

[41]:
```
month_hour_success_rate_query = """
    SELECT
    MONTH(order_time) AS month,
    HOUR(order_time) AS hour,
    COUNT(*) AS total_orders,

    AVG(is_successful)*100 AS success_rate
FROM
    orders
GROUP BY
    month, hour
ORDER BY
    month, hour;
"""
month_hour_success_rate = execute_query(month_hour_success_rate_query)
month_hour_success_rate_pivot = month_hour_success_rate.pivot(index='hour',␣
↪columns='month', values='success_rate')
print(month_hour_success_rate)
```

```
     month  hour  total_orders success_rate
0        1     0            76      63.1579
1        1     1            60      70.0000
2        1     2            38      71.0526
3        1     3            21      76.1905
4        1     4             7      85.7143
..     …   …           …           …
115     12    19          2155      84.3155
116     12    20          2168      81.5498
117     12    21          2166      78.6242
118     12    22          1885      79.7347
119     12    23          1546      79.4955

[120 rows x 4 columns]
```

[77]:
```
month_hour_success_rate_pivot
```

[77]:
```
month        1        9        10       11       12
hour
0       63.1579  82.4449  81.5169  78.4512  79.0759
1       70.0000  79.8670  79.6158  79.9517  75.8155
2       71.0526  75.9843  77.1870  75.2515  75.9777
3       76.1905  77.9710  64.0138  76.2712  76.9231
4       85.7143  68.0000  71.5385  58.5938  78.1250
```

```
5       33.3333  69.3182  77.9221  58.8235  71.6667
6      100.0000  73.9130  76.3636  81.3953  70.5882
7        0.0000  72.0930  85.7143  73.5294  68.0000
8      100.0000  84.2623  85.0144  86.1199  85.6631
9       95.4545  88.5312  85.4730  83.6299  87.1747
10      95.4545  87.5949  82.7503  84.9938  84.7607
11      89.0625  87.5627  83.5916  84.3350  83.6988
12      85.3659  85.9937  85.8438  86.8293  79.1480
13      87.0968  85.9306  85.7322  86.2438  80.0000
14      86.2385  84.6640  86.1810  86.2239  86.2428
15      91.0569  87.5000  85.5637  84.8544  84.9545
16      86.5672  86.2390  85.2054  85.6756  83.9738
17      81.0127  85.6952  82.9186  86.0060  84.8407
18      87.1795  84.9279  84.7770  84.0919  83.4192
19      85.2713  85.8921  84.8348  84.8628  84.3155
20      88.5496  82.9787  82.9907  84.2079  81.5498
21      84.3972  83.2195  83.1833  84.3949  78.6242
22      87.3874  84.2214  83.8574  82.8396  79.7347
23      77.3723  82.4225  83.9881  81.5190  79.4955
```

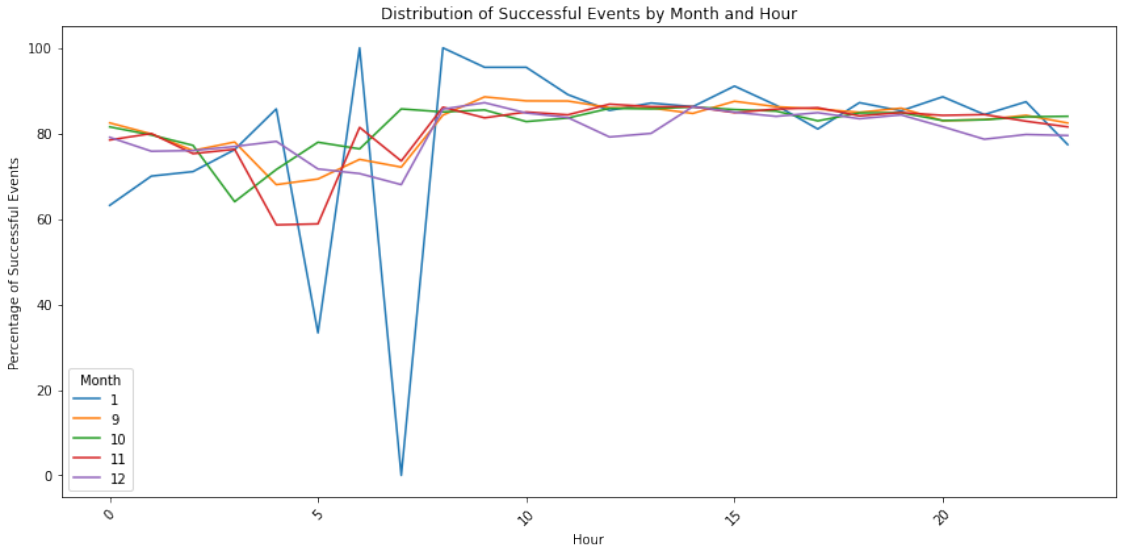
[ ]:

11.2 Plot percentages by month and hour

[42]:
```
# Plot success rate by month and hour
#  Convert column names (month) to numeric
month_hour_success_rate_pivot.columns = pd.
 ↪to_numeric(month_hour_success_rate_pivot.columns, errors='coerce')

# Convert all data to numeric (handle errors gracefully)
month_hour_success_rate_pivot = month_hour_success_rate_pivot.apply(pd.
 ↪to_numeric, errors='coerce')


month_hour_success_rate_pivot.plot(kind="line", figsize=(12, 6))
plt.xlabel("Hour")
plt.ylabel("Percentage of Successful Events")
plt.title("Distribution of Successful Events by Month and Hour")
plt.legend(title="Month")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



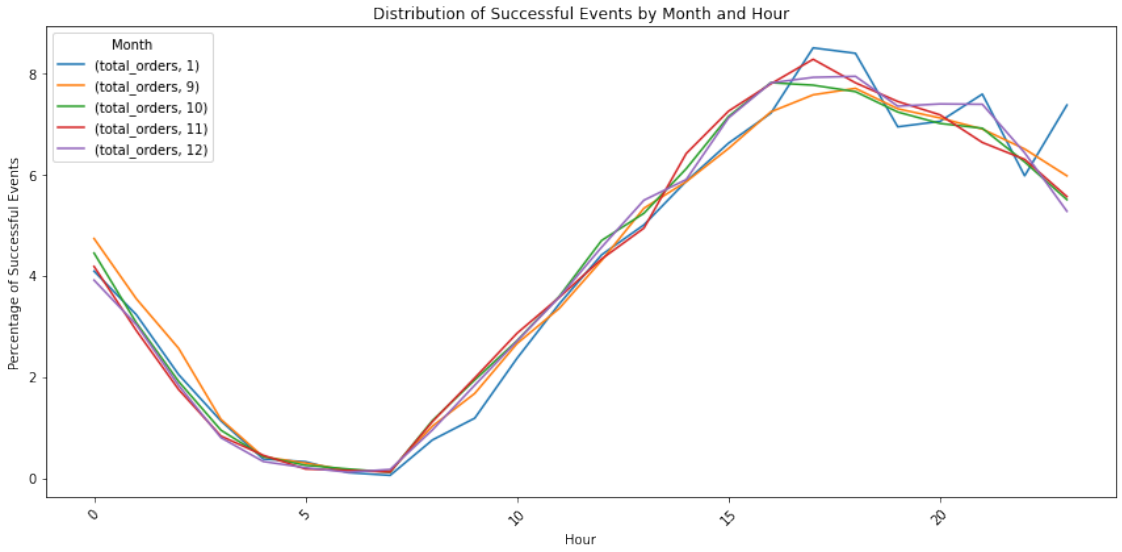
[ ]:

11.3 Percentage Distribution of Hourly Orders by Month

[100]:
```
month_hour_percent_of_orders_pivot = month_hour_success_rate.iloc[:,0:3].
 ↪set_index(["month","hour"]).groupby(level="month").apply(lambda x: (x / x.
 ↪sum()) * 100)
month_hour_percent_of_orders_pivot_unstacked =␣
 ↪month_hour_percent_of_orders_pivot.unstack(level="month")




month_hour_percent_of_orders_pivot_unstacked.plot(kind="line", figsize=(12, 6))
plt.xlabel("Hour")
plt.ylabel("Percentage of Orders Per Day")
plt.title("Distribution of  order in percentage by Month and Hour")
plt.legend(title="Month")
plt.xticks(rotation=45)
# plt.ylim(bottom=70)
plt.tight_layout()
plt.show()
```



[ ]:

11.4 Recommendations Based on the Analysis

11.4.1 Overall Success Rate Analysis

- **Observation:** The overall success rate is 83.55%.
- **Recommendation:** Aim to improve the success rate by focusing on the major failure reasons, particularly online payment failures and customer mistakes. Enhancing UI/UX for payment and offering alternative payment methods might help reduce these failures.

11.4.2 Reason Analysis for Unsuccessful Orders

- **Observation:** The main reasons for failure are "Online Payment," "Customer Mistake Error," and "Fraud Prank."
- **Recommendation:**
- **Online Payment:** Improve the payment process to minimize failures and offer real-time assistance for payment issues.
- **Customer Mistake Error:** Enhance UI clarity with better order review options and include tooltips or FAQs to help users understand the process.
- **Fraud Prank:** Invest in stronger fraud detection systems and penalize prank orders to deter malicious activity.

11.4.3 Time-Based Analysis

- **Observation:** Success rates are lower during late-night hours (2 AM - 6 AM) and higher during the day (9 AM - 8 PM).

- **Recommendation:**
- **Late-night Orders (2 AM - 6 AM):** Address operational challenges during these hours, such as reducing issues with vendors being closed or lack of couriers. Implement backup systems for such cases.
- **Peak Hours (9 AM - 8 PM):** During peak hours, focus on reducing issues like "Item Unavailable," "Vendor No Response," and "Late Delivery" by improving vendor reliability and stock management.

11.4.4 Geographical Analysis

- **Observation:** There are significant variations in success rates by city, with some cities like Minya (94%) performing better than others like Assiut (66%).
- **Recommendation:** Focus on improving success rates in cities with lower performance by improving vendor and logistical support, particularly in areas like Assiut and Al Mahallah Al Kubra.

11.4.5 Payment Method Analysis

- **Observation:** Cash payments have a higher success rate (88%) compared to online payment methods.
- **Recommendation:** Investigate the issues with online payments, especially "Online - Mixed" payments, and find ways to streamline or improve these methods. Offering more reliable online payment solutions could increase overall success rates.

11.4.6 Platform-Based Analysis

- **Observation:** Desktop web orders have a higher success rate than mobile platforms.
- **Recommendation:** Focus on enhancing the mobile app experience (especially Android and iOS) to improve success rates, considering that most orders come from mobile platforms.

11.4.7 Product/Service-Based Analysis

- **Observation:** Success rates for "Pet Shop" and "Grocery" are higher compared to "Electronics" and "Pharmacy."
- **Recommendation:** Investigate the challenges faced by vendors in the "Electronics" and "Pharmacy" verticals. These might be related to stock management or delivery issues. Ensuring better product availability and reliability could improve success rates in these categories.

11.4.8 Delivery Time and Distance Analysis

- **Observation:** Delivery delays and distances might correlate with lower success rates.
- **Recommendation:** Optimize delivery times and reduce delays by improving logistics. Assess delivery routes and consider introducing faster delivery options for high-priority areas.

[ ]: