ry-cost-and-profitability-analysis

July 7, 2024

Below is the process we can follow for the task of Food Delivery Cost and Profitability Analysis:

1. Start by gathering comprehensive data related to all aspects of food delivery operations.
2. Clean the dataset for inconsistencies, missing values, or irrelevant information.
3. Extract relevant features that could impact cost and profitability.
4. Break down the costs associated with each order, including fixed costs (like packaging) and variable costs (like delivery fees and discounts).
5. Determine the revenue generated from each order, focusing on commission fees and the order value before discounts.
6. For each order, calculate the profit by subtracting the total costs from the revenue. Analyze the distribution of profitability across all orders to identify trends.
7. Based on the cost and profitability analysis, develop strategic recommendations aimed at enhancing profitability.
8. Use the data to simulate the financial impact of proposed changes, such as adjusting discount or commission rates.

[24]:
```python
import pandas as pd

food_orders = pd.read_csv("food_orders_new_delhi.csv")
food_orders.head()
```

[24]:
```
   Order ID Customer ID Restaurant ID  Order Date and Time  \
0         1       C8270         R2924  2024-02-01 01:11:52
1         2       C1860         R2054  2024-02-02 22:11:04
2         3       C6390         R2870  2024-01-31 05:54:35
3         4       C6191         R2642  2024-01-16 22:52:49
4         5       C6734         R2799  2024-01-29 01:19:30

  Delivery Date and Time  Order Value  Delivery Fee    Payment Method  \
0    2024-02-01 02:39:52         1914             0       Credit Card
1    2024-02-02 22:46:04          986            40    Digital Wallet
2    2024-01-31 06:52:35          937            30  Cash on Delivery
3    2024-01-16 23:38:49         1463            50  Cash on Delivery
4    2024-01-29 02:48:30         1992            30  Cash on Delivery

  Discounts and Offers  Commission Fee  Payment Processing Fee  \
0           5% on App             150                      47
1                 10%             198                      23
2        15% New User             195                      45
```

```
3                NaN            146                        27
4          50 off Promo         130                        50

   Refunds/Chargebacks
0                    0
1                    0
2                    0
3                    0
4                    0
```

[19]:
```
# see column name
food_orders.columns
```

[19]:
```
Index(['Order ID', 'Customer ID', 'Restaurant ID', 'Order Date and Time',
       'Delivery Date and Time', 'Order Value', 'Delivery Fee',
       'Payment Method', 'Discounts and Offers', 'Commission Fee',
       'Payment Processing Fee', 'Refunds/Chargebacks'],
      dtype='object')
```

[20]:
```
#check null
food_orders.isnull().sum()
```

[20]:
```
Order ID                    0
Customer ID                 0
Restaurant ID               0
Order Date and Time         0
Delivery Date and Time      0
Order Value                 0
Delivery Fee                0
Payment Method              0
Discounts and Offers      185
Commission Fee              0
Payment Processing Fee      0
Refunds/Chargebacks         0
dtype: int64
```

[8]:
```
food_orders.shape
```

[8]:
```
(1000, 12)
```

[9]:
```
food_orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
```

```
 0   Order ID                1000 non-null   int64
 1   Customer ID             1000 non-null   object
 2   Restaurant ID           1000 non-null   object
 3   Order Date and Time     1000 non-null   object
 4   Delivery Date and Time  1000 non-null   object
 5   Order Value             1000 non-null   int64
 6   Delivery Fee            1000 non-null   int64
 7   Payment Method          1000 non-null   object
 8   Discounts and Offers    815 non-null    object
 9   Commission Fee          1000 non-null   int64
 10  Payment Processing Fee  1000 non-null   int64
 11  Refunds/Chargebacks     1000 non-null   int64
dtypes: int64(6), object(6)
memory usage: 93.9+ KB
```

[21]:
```
food_orders['Discounts and Offers'].fillna( 0.0 ,inplace= True)
```

```
C:\Users\Ashraf Sakil\AppData\Local\Temp\ipykernel_16204\2155935680.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  food_orders['Discounts and Offers'].fillna( 0.0 ,inplace= True)
```

[11]:
```
#Again check
food_orders.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Order ID                1000 non-null   int64
 1   Customer ID             1000 non-null   object
 2   Restaurant ID           1000 non-null   object
 3   Order Date and Time     1000 non-null   object
 4   Delivery Date and Time  1000 non-null   object
 5   Order Value             1000 non-null   int64
 6   Delivery Fee            1000 non-null   int64
 7   Payment Method          1000 non-null   object
 8   Discounts and Offers    1000 non-null   object
 9   Commission Fee          1000 non-null   int64
 10  Payment Processing Fee  1000 non-null   int64
 11  Refunds/Chargebacks     1000 non-null   int64
dtypes: int64(6), object(6)
memory usage: 93.9+ KB
```

[31]:
```
food_orders['Discounts and Offers'].value_counts()
```

[31]:
```
Discounts and Offers
10%               233
50 off Promo      201
15% New User      198
5% on App         183
Name: count, dtype: int64
```

The dataset contains 1,000 entries and 12 columns, with no missing values in any of the columns. Now, we need to perform some data cleaning and preparation. Below are the necessary cleaning steps we need to take:

Convert "Order Date and Time" and "Delivery Date and Time" to a datetime format. Convert "Discounts and Offers" to a consistent numeric value (if applicable) or calculate the discount amounts. Ensure all monetary values are in a suitable format for calculations.

[26]:
```
from datetime import datetime
#convert date and time columns to datetime
food_orders['Order Date and Time']=pd.to_datetime(food_orders['Order Date and␣
 ↪Time'])
food_orders['Delivery Date and Time']=pd.to_datetime(food_orders['Delivery Date␣
 ↪and Time'])
```

[39]:
```
# first, let's create a function to extract numeric values from the 'Discounts␣
 ↪and Offers' string
def extract_discount(discount_str):
    if pd.isna(discount_str):
        return 0.0
    if 'off' in discount_str:
        try:
            return float(discount_str.split(' ')[0])
        except ValueError:
            return 0.0
    elif '%' in discount_str:
        try:
            return float(discount_str.split('%')[0])
        except ValueError:
            return 0.0
    else:
        return 0.0
# apply the function to create a new 'Discount Value' column
```

```
food_orders['Discount Percentage'] = food_orders['Discounts and Offers'].
↪apply(lambda x: extract_discount(x))
```

[41]:
```
# for percentage discounts, calculate the discount amount based on the order␣
↪value
food_orders['Discount Amount'] = food_orders.apply(lambda x: (x['Order Value']␣
↪* x['Discount Percentage'] / 100)
                                                    if x['Discount Percentage']␣
↪> 1
                                                    else x['Discount␣
↪Percentage'], axis=1)
```

[42]:
```
# adjust 'Discount Amount' for fixed discounts directly specified in the␣
↪'Discounts and Offers' column
food_orders['Discount Amount'] = food_orders.apply(lambda x: x['Discount␣
↪Amount'] if x['Discount Percentage'] <= 1
                                                    else x['Order Value'] *␣
↪x['Discount Percentage'] / 100, axis=1)
```

[36]:
```
food_orders['Discount Amount']
```

[36]:
```
0       95.70
1       98.60
2      140.55
3        0.00
4        0.00
        …
995     41.25
996      0.00
997      0.00
998    212.10
999    248.55
Name: Discount Amount, Length: 1000, dtype: float64
```

[43]:
```
print(food_orders[['Order Value', 'Discounts and Offers', 'Discount␣
↪Percentage', 'Discount Amount']].head(), food_orders.dtypes)
```

```
   Order Value Discounts and Offers  Discount Percentage  Discount Amount
0         1914             5% on App                  5.0            95.70
1          986                   10%                 10.0            98.60
2          937         15% New User                  15.0           140.55
3         1463                   NaN                  0.0             0.00
4         1992         50 off Promo                  50.0           996.00 Order
ID                           int64
Customer ID                      object
Restaurant ID                    object
Order Date and Time      datetime64[ns]
```

```
Delivery Date and Time    datetime64[ns]
Order Value                        int64
Delivery Fee                       int64
Payment Method                    object
Discounts and Offers              object
Commission Fee                     int64
Payment Processing Fee             int64
Refunds/Chargebacks                int64
Discount Percentage              float64
Discount Amount                  float64
dtype: object
```

Cost and Profitability Analysis For the cost analysis, we'll consider the following costs associated with each order:

1. Delivery Fee: The fee charged for delivering the order.
2. Payment Processing Fee: The fee for processing the payment.
3. Discount Amount: The discount provided on the order.

```
[44]: # calculate total costs and revenue per order
food_orders['Total Costs'] = food_orders['Delivery Fee'] + food_orders['Payment␣
 ↪Processing Fee'] + food_orders['Discount Amount']
food_orders['Revenue'] = food_orders['Commission Fee']
food_orders['Profit'] = food_orders['Revenue'] - food_orders['Total Costs']
```

```
[47]: # aggregate data to get overall metrics
total_orders = food_orders.shape[0]
total_revenue = food_orders['Revenue'].sum()
total_costs = food_orders['Total Costs'].sum()
total_profit = food_orders['Profit'].sum()
```

```
[48]: overall_metrics = {
    "Total Orders": total_orders,
    "Total Revenue": total_revenue,
    "Total Costs": total_costs,
    "Total Profit": total_profit
}

print(overall_metrics)
```

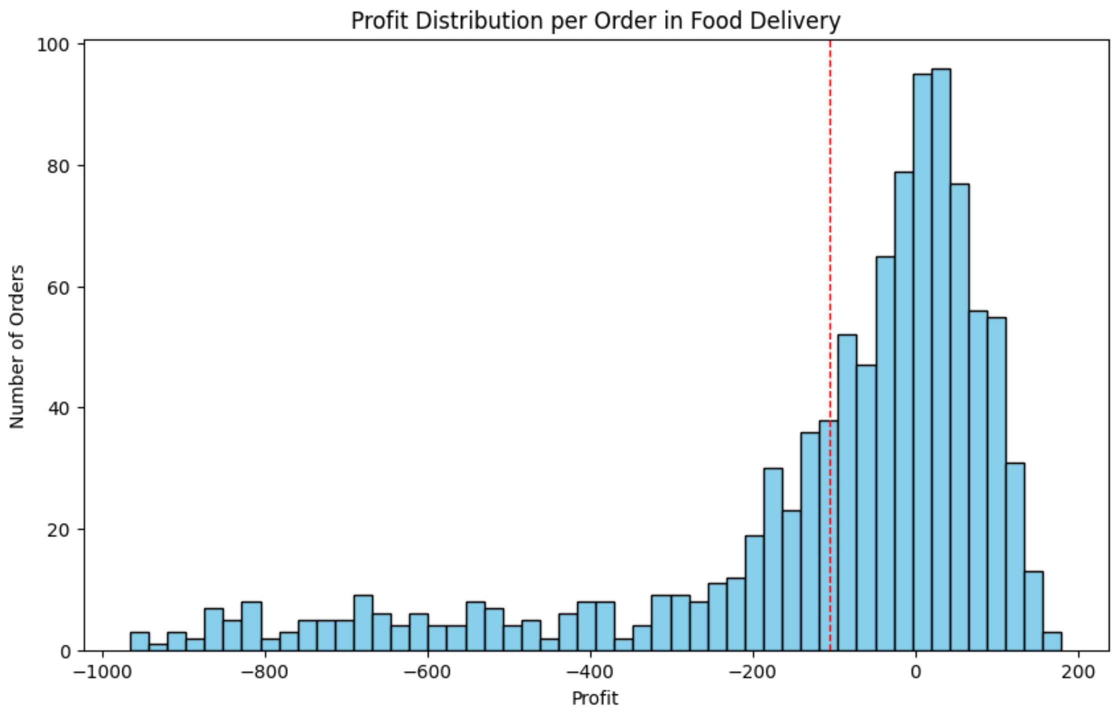
```
{'Total Orders': 1000, 'Total Revenue': 126990, 'Total Costs': 232709.85, 'Total
Profit': -105719.85}
```

Based on the analysis, here are the overall metrics for the food delivery operations:

1. Total Orders: 1,000
2. Total Revenue (from Commission Fees): 126,990 INR
3. Total Costs: 232,709.85 INR (including delivery fees, payment processing fees, and discounts)
4. Total Profit: -105,719.85 INR

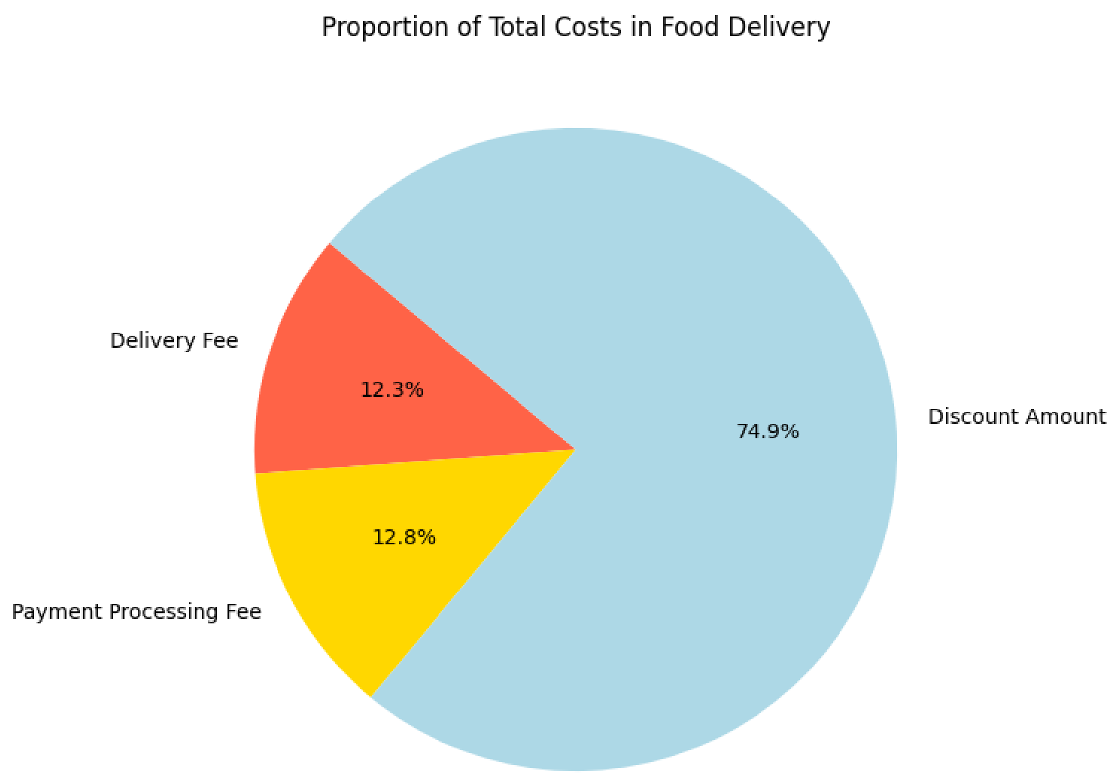
The analysis indicates that the total costs associated with the food delivery operations exceed the total revenue generated from commission fees, resulting in a net loss. It suggests that the current commission rates, delivery fees, and discount strategies might not be sustainable for profitability

[49]:
```python
import matplotlib.pyplot as plt

# histogram of profits per order
plt.figure(figsize=(10, 6))
plt.hist(food_orders['Profit'], bins=50, color='skyblue', edgecolor='black')
plt.title('Profit Distribution per Order in Food Delivery')
plt.xlabel('Profit')
plt.ylabel('Number of Orders')
plt.axvline(food_orders['Profit'].mean(), color='red', linestyle='dashed',␣
 ↪linewidth=1)
plt.show()
```



The histogram shows a wide distribution of profit per order, with a noticeable number of orders resulting in a loss (profits below 0). The red dashed line indicates the average profit, which is in the negative territory, highlighting the overall loss-making situation

[50]:
```python
# pie chart for the proportion of total costs
costs_breakdown = food_orders[['Delivery Fee', 'Payment Processing Fee',␣
 ↪'Discount Amount']].sum()
plt.figure(figsize=(7, 7))
plt.pie(costs_breakdown, labels=costs_breakdown.index, autopct='%1.1f%%',␣
 ↪startangle=140, colors=['tomato', 'gold', 'lightblue'])
plt.title('Proportion of Total Costs in Food Delivery')
plt.show()
```



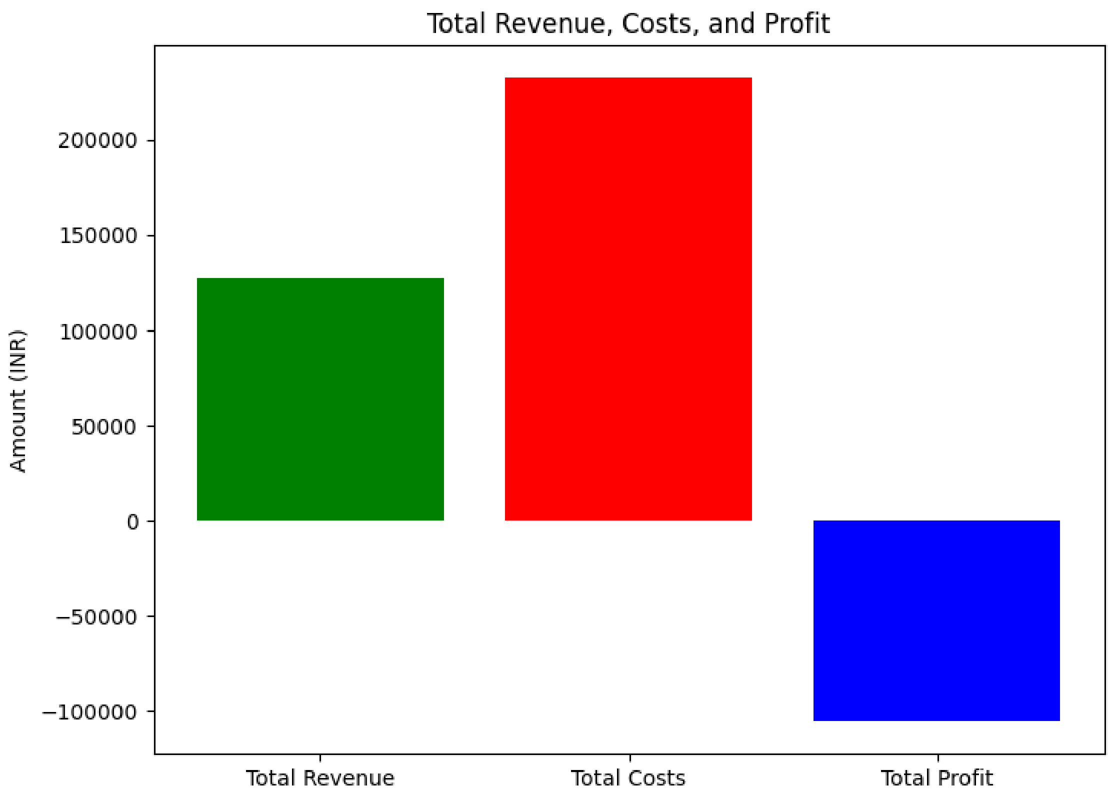
The pie chart illustrates the breakdown of total costs into delivery fees, payment processing fees, and discount amounts. Discounts constitute a significant portion of the costs, suggesting that promotional strategies might be heavily impacting overall profitability.

[51]:
```
costs_breakdown
```

[51]:
```
Delivery Fee               28620.00
Payment Processing Fee     29832.00
Discount Amount           174257.85
dtype: float64
```

[52]:
```
# bar chart for total revenue, costs, and profit
totals = ['Total Revenue', 'Total Costs', 'Total Profit']
values = [total_revenue, total_costs, total_profit]
```

```python
plt.figure(figsize=(8, 6))
plt.bar(totals, values, color=['green', 'red', 'blue'])
plt.title('Total Revenue, Costs, and Profit')
plt.ylabel('Amount (INR)')
plt.show()
```



The bar chart compares total revenue, total costs, and total profit. It visually represents the gap between revenue and costs, clearly showing that the costs surpass the revenue, leading to a total loss. A New Strategy for Profits

From the analysis so far we understood that the discounts on food orders are resulting in huge losses. Now, we need to find a new strategy for profitability. We need to find a sweet spot for offering discounts and charging commissions. To find a sweet spot for commission and discount percentages, we can analyze the characteristics of profitable orders more deeply. Specifically, we need to look for:

A new average commission percentage based on profitable orders. A new average discount percentage for profitable orders, that could serve as a guideline for what level of discount still allows for profitability.

[53]:
```python
# filter the dataset for profitable orders
profitable_orders = food_orders[food_orders['Profit'] > 0]
```

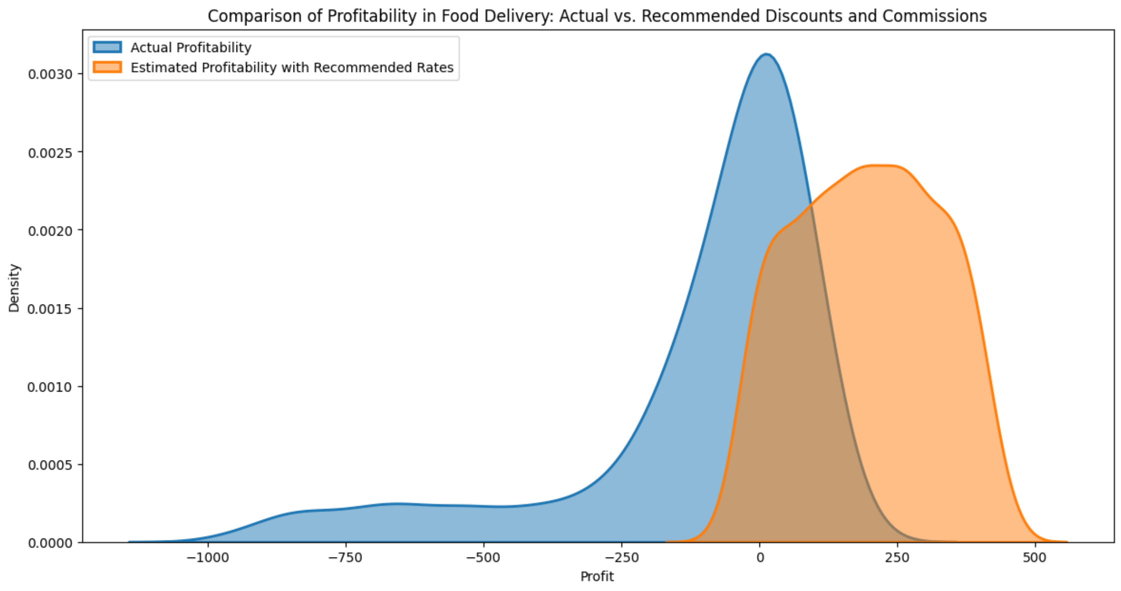
```
# calculate the average commission percentage for profitable orders
profitable_orders['Commission Percentage'] = (profitable_orders['Commission␣
↪Fee'] / profitable_orders['Order Value']) * 100

# calculate the average discount percentage for profitable orders
profitable_orders['Effective Discount Percentage'] =␣
↪(profitable_orders['Discount Amount'] / profitable_orders['Order Value']) *␣
↪100

# calculate the new averages
new_avg_commission_percentage = profitable_orders['Commission Percentage'].
↪mean()
new_avg_discount_percentage = profitable_orders['Effective Discount␣
↪Percentage'].mean()

print(new_avg_commission_percentage, new_avg_discount_percentage)
```

```
30.508436145149435 5.867469879518072

C:\Users\Ashraf Sakil\AppData\Local\Temp\ipykernel_16204\2426516542.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  profitable_orders['Commission Percentage'] = (profitable_orders['Commission
Fee'] / profitable_orders['Order Value']) * 100
C:\Users\Ashraf Sakil\AppData\Local\Temp\ipykernel_16204\2426516542.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  profitable_orders['Effective Discount Percentage'] =
(profitable_orders['Discount Amount'] / profitable_orders['Order Value']) * 100
```

[56]:
```
# simulate profitability with recommended discounts and commissions
recommended_commission_percentage = 30.0  # 30%
recommended_discount_percentage = 6.0     # 6%

# calculate the simulated commission fee and discount amount using recommended␣
↪percentages
food_orders['Simulated Commission Fee'] = food_orders['Order Value'] *␣
↪(recommended_commission_percentage / 100)
```

```python
food_orders['Simulated Discount Amount'] = food_orders['Order Value'] *␣
 ↪(recommended_discount_percentage / 100)

# recalculate total costs and profit with simulated values
food_orders['Simulated Total Costs'] = (food_orders['Delivery Fee'] +
                                        food_orders['Payment Processing Fee'] +
                                        food_orders['Simulated Discount␣
 ↪Amount'])

food_orders['Simulated Profit'] = (food_orders['Simulated Commission Fee'] -
                                   food_orders['Simulated Total Costs'])


print('total cost simulated :',  food_orders['Simulated Total Costs'].sum())
print('toala profit simulated', food_orders['Simulated Profit'].sum())

# visualizing the comparison
import seaborn as sns

plt.figure(figsize=(14, 7))

# actual profitability
sns.kdeplot(food_orders['Profit'], label='Actual Profitability', fill=True,␣
 ↪alpha=0.5, linewidth=2)

# simulated profitability
sns.kdeplot(food_orders['Simulated Profit'], label='Estimated Profitability␣
 ↪with Recommended Rates', fill=True, alpha=0.5, linewidth=2)

plt.title('Comparison of Profitability in Food Delivery: Actual vs. Recommended␣
 ↪Discounts and Commissions')
plt.xlabel('Profit')
plt.ylabel('Density')
plt.legend(loc='upper left')
plt.show()
```

```
total cost simulated : 121690.14
toala profit simulated 194500.56
```



The visualization compares the distribution of profitability per order using actual discounts and commissions versus the simulated scenario with recommended discounts (6%) and commissions (30%).

The actual profitability distribution shows a mix, with a significant portion of orders resulting in losses (profit < 0) and a broad spread of profit levels for orders. The simulated scenario suggests a shift towards higher profitability per order. The distribution is more skewed towards positive profit, indicating that the recommended adjustments could lead to a higher proportion of profitable orders.

Summary

So, this is how you can analyze the cost and profitability of a food delivery company. Food Delivery Cost and Profitability Analysis involves examining all the costs associated with delivering food orders, from direct expenses like delivery fees and packaging to indirect expenses like discounts offered to customers and commission fees paid by restaurants. By juxtaposing these costs against the revenue generated (primarily through order values and commission fees), the analysis aims to provide insights into how profitable the food delivery service is on a per-order basis.