B2B Courier Charges Accuracy Analysis

In today’s fast-paced e-commerce industry, fast and efficient order delivery is crucial to business success. To ensure seamless order fulfilment, businesses often partner with courier companies to ship their products to customers. However, managing the charges collected by these courier companies can be difficult, especially when dealing with a high volume of orders. It is one of the real-time problems B2B businesses experience when their estimated charges for the same invoice don’t match.

In [1]:
```
#Let’s start this task by importing the necessary Python libraries and the datas
import pandas as pd

order_report = pd.read_csv('Order Report.csv')
sku_master = pd.read_csv('SKU Master.csv')
pincode_mapping = pd.read_csv('pincodes.csv')
courier_invoice = pd.read_csv('Invoice.csv')
courier_company_rates = pd.read_csv('Courier Company - Rates.csv')

print("Order Report:")
print(order_report.head())
print("\nSKU Master:")
print(sku_master.head())
print("\nPincode Mapping:")
print(pincode_mapping.head())
print("\nCourier Invoice:")
print(courier_invoice.head())
print("\nCourier Company rates:")
print(courier_company_rates.head())
```

```
Order Report:
   ExternOrderNo            SKU  Order Qty  Unnamed: 3  Unnamed: 4
0     2001827036  8904223818706        1.0         NaN         NaN
1     2001827036  8904223819093        1.0         NaN         NaN
2     2001827036  8904223819109        1.0         NaN         NaN
3     2001827036  8904223818430        1.0         NaN         NaN
4     2001827036  8904223819277        1.0         NaN         NaN

SKU Master:
             SKU  Weight (g)  Unnamed: 2  Unnamed: 3  Unnamed: 4
0  8904223815682         210         NaN         NaN         NaN
1  8904223815859         165         NaN         NaN         NaN
2  8904223815866         113         NaN         NaN         NaN
3  8904223815873          65         NaN         NaN         NaN
4  8904223816214         120         NaN         NaN         NaN

Pincode Mapping:
   Warehouse Pincode  Customer Pincode Zone  Unnamed: 3  Unnamed: 4
0             121003            507101    d         NaN         NaN
1             121003            486886    d         NaN         NaN
2             121003            532484    d         NaN         NaN
3             121003            143001    b         NaN         NaN
4             121003            515591    d         NaN         NaN

Courier Invoice:
        AWB Code    Order ID  Charged Weight  Warehouse Pincode  \
0  1091117222124  2001806232            1.30             121003
1  1091117222194  2001806273            1.00             121003
2  1091117222931  2001806408            2.50             121003
3  1091117223244  2001806458            1.00             121003
4  1091117229345  2001807012            0.15             121003

   Customer Pincode Zone Type of Shipment  Billing Amount (Rs.)
0            507101    d  Forward charges                 135.0
1            486886    d  Forward charges                  90.2
2            532484    d  Forward charges                 224.6
3            143001    b  Forward charges                  61.3
4            515591    d  Forward charges                  45.4

Courier Company rates:
   fwd_a_fixed  fwd_a_additional  fwd_b_fixed  fwd_b_additional  fwd_c_fixed  \
0         29.5              23.6           33              28.3         40.1

   fwd_c_additional  fwd_d_fixed  fwd_d_additional  fwd_e_fixed  \
0              38.9         45.4              44.8         56.6

   fwd_e_additional  rto_a_fixed  rto_a_additional  rto_b_fixed  \
0              55.5         13.6              23.6         20.5

   rto_b_additional  rto_c_fixed  rto_c_additional  rto_d_fixed  \
0              28.3         31.9              38.9         41.3

   rto_d_additional  rto_e_fixed  rto_e_additional
0              44.8         50.7              55.5
```

In [2]:

```
#Now let’s have a look if any of the data contains missing values:
# Check for missing values
print("\nMissing values in Website Order Report:")
print(order_report.isnull().sum())
print("\nMissing values in SKU Master:")
```

```
print(sku_master.isnull().sum())
print("\nMissing values in Pincode Mapping:")
print(pincode_mapping.isnull().sum())
print("\nMissing values in Courier Invoice:")
print(courier_invoice.isnull().sum())
print("\nMissing values in courier company rates:")
print(courier_company_rates.isnull().sum())
```

```
Missing values in Website Order Report:
ExternOrderNo      0
SKU                0
Order Qty          0
Unnamed: 3       400
Unnamed: 4       400
dtype: int64

Missing values in SKU Master:
SKU            0
Weight (g)     0
Unnamed: 2    66
Unnamed: 3    66
Unnamed: 4    66
dtype: int64

Missing values in Pincode Mapping:
Warehouse Pincode      0
Customer Pincode       0
Zone                   0
Unnamed: 3           124
Unnamed: 4           124
dtype: int64

Missing values in Courier Invoice:
AWB Code                0
Order ID                0
Charged Weight          0
Warehouse Pincode       0
Customer Pincode        0
Zone                    0
Type of Shipment        0
Billing Amount (Rs.)    0
dtype: int64

Missing values in courier company rates:
fwd_a_fixed         0
fwd_a_additional    0
fwd_b_fixed         0
fwd_b_additional    0
fwd_c_fixed         0
fwd_c_additional    0
fwd_d_fixed         0
fwd_d_additional    0
fwd_e_fixed         0
fwd_e_additional    0
rto_a_fixed         0
rto_a_additional    0
rto_b_fixed         0
rto_b_additional    0
rto_c_fixed         0
rto_c_additional    0
rto_d_fixed         0
rto_d_additional    0
rto_e_fixed         0
rto_e_additional    0
dtype: int64
```

In [3]:
```
#Now let’s clean the data:
# Remove unnamed columns from the Website Order Report DataFrame
```

```
order_report = order_report.drop(columns=['Unnamed: 3', 'Unnamed: 4'])

# Remove unnamed columns from the SKU Master DataFrame
sku_master = sku_master.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'])

# Remove unnamed columns from the Pincode Mapping DataFrame
pincode_mapping = pincode_mapping.drop(columns=['Unnamed: 3', 'Unnamed: 4'])
```

In [4]:
```
#Now let’s merge the order report and SKU master datasets according to the commo
# Merge the Order Report and SKU Master based on SKU
merged_data = pd.merge(order_report, sku_master, on='SKU')
print(merged_data.head())
```

```
   ExternOrderNo            SKU  Order Qty  Weight (g)
0     2001827036  8904223818706        1.0         127
1     2001821995  8904223818706        1.0         127
2     2001819252  8904223818706        1.0         127
3     2001816996  8904223818706        1.0         127
4     2001814580  8904223818706        1.0         127
```

In [5]:
```
# Rename the "ExternOrderNo" column to "Order ID" in the merged_data DataFrame
merged_data = merged_data.rename(columns={'ExternOrderNo': 'Order ID'})
print(merged_data.head())
```

```
     Order ID            SKU  Order Qty  Weight (g)
0  2001827036  8904223818706        1.0         127
1  2001821995  8904223818706        1.0         127
2  2001819252  8904223818706        1.0         127
3  2001816996  8904223818706        1.0         127
4  2001814580  8904223818706        1.0         127
```

In [6]:
```
#Now let’s merge the courier invoice and pincode mapping dataset:
abc_courier = pincode_mapping.drop_duplicates(subset=['Customer Pincode'])
courier_abc= courier_invoice[['Order ID', 'Customer Pincode','Type of Shipment']
pincodes= courier_abc.merge(abc_courier,on='Customer Pincode')
print(pincodes.head())
```

```
     Order ID  Customer Pincode Type of Shipment  Warehouse Pincode Zone
0  2001806232            507101  Forward charges             121003    d
1  2001806273            486886  Forward charges             121003    d
2  2001806408            532484  Forward charges             121003    d
3  2001806458            143001  Forward charges             121003    b
4  2001807012            515591  Forward charges             121003    d
```

Below is how the above code works:

We first extract the unique customer pin codes from the pincode mapping dataset and create a new DataFrame called “abc_courier” to store this information. We then select specific columns (“Order ID”, “Customer Pincode”, “Type of Shipment”) from the courier_invoice dataset and create a new DataFrame called “courier_abc” to store this subset of data. We then merge the ‘courier_abc’ DataFrame with the ‘abc_courier’ DataFrame based on the ‘Customer Pincode’ column. This merge operation helps us associate customer pin codes with their respective orders and shipping types. The resulting DataFrame is named ‘pincodes’.

In [7]:
```
#Now let’s merge the pin codes with the main dataframe:
merged2 = merged_data.merge(pincodes, on='Order ID')
```

In [8]:
```
#Now let’s calculate the weight in kilograms by dividing the ‘Weight (g)’ column
merged2['Weights (Kgs)'] = merged2['Weight (g)'] / 1000
```

In [9]:
```
#Now let’s calculate the weight slabs:
def weight_slab(weight):
    i = round(weight % 1, 1)
    if i == 0.0:
        return weight
    elif i > 0.5:
        return int(weight) + 1.0
    else:
        return int(weight) + 0.5

merged2['Weight Slab (KG)'] = merged2['Weights (Kgs)'].apply(weight_slab)
courier_invoice['Weight Slab Charged by Courier Company']=(courier_invoice['Char
```

The weight_slab() function is defined to determine the weight slab based on the weight of the shipment. It takes the input weight and applies certain conditions to calculate the weight slab. Below is how it works:

The function first calculates the remainder of the weight divided by 1 and rounds it to one decimal place. If the remainder is 0.0, it means the weight is a multiple of 1 KG, and the function returns the weight as it is. If the remainder is greater than 0.5, it means that the weight exceeds the next half KG slab. In this case, the function rounds the weight to the nearest integer and adds 1.0 to it, which represents the next heavier slab. If the remainder is less than or equal to 0.5, it means the weight falls into the current half-KG bracket. In this case, the function rounds the weight to the nearest integer and adds 0.5 to it, which represents the current weight slab.

In [10]:
```
#Now let’s rename the columns to prepare the desired dataframe:
courier_invoice = courier_invoice.rename(columns={'Zone': 'Delivery Zone Chargec
merged2 = merged2.rename(columns={'Zone': 'Delivery Zone As Per ABC'})
merged2 = merged2.rename(columns={'Weight Slab (KG)': 'Weight Slab As Per ABC'})
```

In [11]:
```
#Now let’s calculate the expected charges:
total_expected_charge = []

for _, row in merged2.iterrows():
    fwd_category = 'fwd_' + row['Delivery Zone As Per ABC']
    fwd_fixed = courier_company_rates.at[0, fwd_category + '_fixed']
    fwd_additional = courier_company_rates.at[0, fwd_category + '_additional']
    rto_category = 'rto_' + row['Delivery Zone As Per ABC']
    rto_fixed = courier_company_rates.at[0, rto_category + '_fixed']
    rto_additional = courier_company_rates.at[0, rto_category + '_additional']

    weight_slab = row['Weight Slab As Per ABC']

    if row['Type of Shipment'] == 'Forward charges':
        additional_weight = max(0, (weight_slab - 0.5) / 0.5)
        total_expected_charge.append(fwd_fixed + additional_weight * fwd_additic
    elif row['Type of Shipment'] == 'Forward and RTO charges':
        additional_weight = max(0, (weight_slab - 0.5) / 0.5)
        total_expected_charge.append(fwd_fixed + additional_weight * (fwd_additi
    else:
```

```
        total_expected_charge.append(0)

merged2['Expected Charge as per ABC'] = total_expected_charge
print(merged2.head())
```

```
     Order ID            SKU  Order Qty  Weight (g)  Customer Pincode  \
0  2001827036  8904223818706        1.0         127            173213
1  2001827036  8904223819093        1.0         150            173213
2  2001827036  8904223819109        1.0         100            173213
3  2001827036  8904223818430        1.0         165            173213
4  2001827036  8904223819277        1.0         350            173213

  Type of Shipment  Warehouse Pincode Delivery Zone As Per ABC  Weights (Kgs)  \
0  Forward charges             121003                        e          0.127
1  Forward charges             121003                        e          0.150
2  Forward charges             121003                        e          0.100
3  Forward charges             121003                        e          0.165
4  Forward charges             121003                        e          0.350

   Weight Slab As Per ABC  Expected Charge as per ABC
0                     0.5                        56.6
1                     0.5                        56.6
2                     0.5                        56.6
3                     0.5                        56.6
4                     0.5                        56.6
```

Below is how the above code works:

In this code, we loop through each row of the 'merged2' DataFrame to calculate the expected charges based on ABC's tariffs. We retrieve the necessary rates and parameters, such as fixed charges and surcharges per weight tier for forward and RTO shipments, based on the delivery area. We then determine the weight of the slab for each row. If the shipment type is 'Forward Charges', we calculate the additional weight beyond the basic weight slab (0.5 KG) and apply the corresponding additional charges. For "Forward and RTO Charges" shipments, we consider additional charges for term and RTO components. Finally, we store the calculated expected charges in the "Expected charges according to ABC" column of the "merged2" DataFrame. This allows us to compare the expected charges with the charges billed to analyze the accuracy of the courier company's charges.

In [12]:
```
#Now let's merge it with the courier invoice to display the final dataframe:
merged_output = merged2.merge(courier_invoice, on='Order ID')
print(merged_output.head())
```

```
      Order ID            SKU  Order Qty  Weight (g)  Customer Pincode_x  \
0  2001827036  8904223818706        1.0         127              173213
1  2001827036  8904223819093        1.0         150              173213
2  2001827036  8904223819109        1.0         100              173213
3  2001827036  8904223818430        1.0         165              173213
4  2001827036  8904223819277        1.0         350              173213

  Type of Shipment_x  Warehouse Pincode_x Delivery Zone As Per ABC  \
0    Forward charges               121003                        e
1    Forward charges               121003                        e
2    Forward charges               121003                        e
3    Forward charges               121003                        e
4    Forward charges               121003                        e

   Weights (Kgs)  Weight Slab As Per ABC  Expected Charge as per ABC  \
0          0.127                     0.5                        56.6
1          0.150                     0.5                        56.6
2          0.100                     0.5                        56.6
3          0.165                     0.5                        56.6
4          0.350                     0.5                        56.6

        AWB Code  Charged Weight  Warehouse Pincode_y  Customer Pincode_y  \
0  1091122418320             1.6               121003              173213
1  1091122418320             1.6               121003              173213
2  1091122418320             1.6               121003              173213
3  1091122418320             1.6               121003              173213
4  1091122418320             1.6               121003              173213

  Delivery Zone Charged by Courier Company Type of Shipment_y  \
0                                        b    Forward charges
1                                        b    Forward charges
2                                        b    Forward charges
3                                        b    Forward charges
4                                        b    Forward charges

   Billing Amount (Rs.)  Weight Slab Charged by Courier Company
0                 117.9                                     2.0
1                 117.9                                     2.0
2                 117.9                                     2.0
3                 117.9                                     2.0
4                 117.9                                     2.0
```

In [13]:
```
#Now let’s calculate the differences in charges and expected charges for each or
df_diff = merged_output
df_diff['Difference (Rs.)'] = df_diff['Billing Amount (Rs.)'] - df_diff['Expecte

df_new = df_diff[['Order ID', 'Difference (Rs.)', 'Expected Charge as per ABC']]

print(df_new.head())
```

```
     Order ID  Difference (Rs.)  Expected Charge as per ABC
0  2001827036              61.3                        56.6
1  2001827036              61.3                        56.6
2  2001827036              61.3                        56.6
3  2001827036              61.3                        56.6
4  2001827036              61.3                        56.6
```

In [14]:
```
#Now let’s summarize the accuracy of B2B courier charges based on the charged pr
# Calculate the total orders in each category
total_correctly_charged = len(df_new[df_new['Difference (Rs.)'] == 0])
```

```
total_overcharged = len(df_new[df_new['Difference (Rs.)'] > 0])
total_undercharged = len(df_new[df_new['Difference (Rs.)'] < 0])

# Calculate the total amount in each category
amount_overcharged = abs(df_new[df_new['Difference (Rs.)'] > 0]['Difference (Rs.
amount_undercharged = df_new[df_new['Difference (Rs.)'] < 0]['Difference (Rs.)']
amount_correctly_charged = df_new[df_new['Difference (Rs.)'] == 0]['Expected Cha

# Create a new DataFrame for the summary
summary_data = {'Description': ['Total Orders where ABC has been correctly charg
                                'Total Orders where ABC has been overcharged',
                                'Total Orders where ABC has been undercharged'],
                'Count': [total_correctly_charged, total_overcharged, total_unde
                'Amount (Rs.)': [amount_correctly_charged, amount_overcharged, a

df_summary = pd.DataFrame(summary_data)

print(df_summary)
```

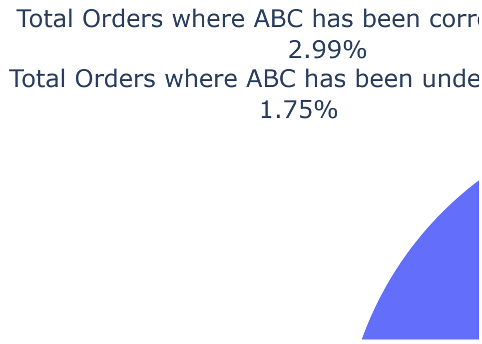
```
                                  Description  Count  Amount (Rs.)
0  Total Orders where ABC has been correctly charged     12         507.6
1        Total Orders where ABC has been overcharged    382       33750.5
2       Total Orders where ABC has been undercharged      7        -165.2
```

In [15]:
```
#We can also visualize the proportion of errors as shown below:
import plotly.graph_objects as go

fig = go.Figure(data=go.Pie(labels=df_summary['Description'],
                            values=df_summary['Count'],
                            textinfo='label+percent',
                            hole=0.4))

fig.update_layout(
    title={
        'text': 'Proportion',
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    }
)

fig.show()
```


Total Orders where ABC has been corr
2.99%
Total Orders where ABC has been unde
1.75%

```
In [ ]:
```