Improving B2B Courier Charges Accuracy: A Python-Based Analysis

In today's dynamic e-commerce landscape, efficient order delivery stands as a linchpin for business prosperity. However, managing charges accrued from courier companies poses a significant challenge for B2B businesses, especially amidst high order volumes. Discrepancies between estimated and actual charges for the same invoice emerge as a pressing real-time issue, underscoring the need for a systematic solution. The B2B Courier Charges Accuracy Analysis endeavors to tackle the challenge of ensuring precise billing in B2B courier transactions. This entails assessing the alignment between charged fees and expected charges derived from predefined parameters and tariffs. By identifying and mitigating discrepancies, businesses aim to streamline billing processes, enhance financial integrity, and optimize operational efficiency.

Problem Statement:
The B2B Courier Charges Accuracy Analysis seeks to address the challenge of ensuring the precision of fees levied by courier companies for the transportation of goods in B2B transactions. The primary objective is to ascertain that businesses are billed accurately for the services rendered by courier partners.

Key Objectives:

1. **Assessment of Charge Accuracy:** Conduct a thorough examination of charged fees vis-à-vis the expected charges derived from predefined parameters and tariffs. Identify and analyze any deviations or discrepancies to gauge the accuracy of billing practices.
2. **Identification of Factors Impacting Accuracy:** Explore various factors contributing to discrepancies in charge accuracy, encompassing aspects such as weight calculations, shipment types, delivery areas, and tariff application. Gain insights into potential sources of errors and inefficiencies in the billing process.
3. **Development of Analytical Solutions:** Utilize Python-based analytical tools and techniques to develop solutions aimed at enhancing charge accuracy in B2B courier transactions. Leverage data analysis, visualization, and modeling to streamline billing processes and mitigate discrepancies.

Skills Utilization:
The analysis of B2B courier charges accuracy presents a multifaceted problem that necessitates the application of a wide array of data-related skills. From data preprocessing and manipulation to statistical analysis and machine learning, this endeavor provides an opportunity to leverage diverse skill sets in working with data.

By delving into the realm of B2B courier charges accuracy analysis, businesses can unlock insights that drive operational efficiency and financial integrity. Through the application of Python-based analytical methodologies, stakeholders can not only address existing discrepancies but also implement proactive measures to ensure precision in future billing processes. This article serves as a guide to navigate the complexities of charge accuracy analysis and forge a path towards optimized B2B courier operations.

In [...

```
import pandas as pd

order_report = pd.read_csv('C:/Users/anike/OneDrive/Desktop/Projects/Machine Learning/courier/Order Report.csv')
sku_master = pd.read_csv('C:/Users/anike/OneDrive/Desktop/Projects/Machine Learning/courier/SKU Master.csv')
pincode_mapping = pd.read_csv('C:/Users/anike/OneDrive/Desktop/Projects/Machine Learning/courier/pincodes.csv')
courier_invoice = pd.read_csv('C:/Users/anike/OneDrive/Desktop/Projects/Machine Learning/courier/Invoice.csv')
courier_company_rates = pd.read_csv('C:/Users/anike/OneDrive/Desktop/Projects/Machine Learning/courier/Courier Company

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 [14]:
```
# 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 [15]:
```
# 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 [16]:
```
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
```

The 'ExternOrderNo' is nothing but 'Order Id' in other datasets.

In [17]:
```
# Rename the "ExternOrderNo" column to "Order ID" in the merged_data DataFrame
merged_data = merged_data.rename(columns={'ExternOrderNo': 'Order ID'})
```

Let's combine the data from the courier invoice and pincode mapping datasets.

In [18]:
```
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
```

Integrating Courier Invoice Data with Pincode Mapping: A Step-by-Step Explanation

In the provided code snippet, we execute a series of steps to merge the courier invoice and pincode mapping datasets. Here's a detailed breakdown of the process:

1. **Extraction of Unique Customer Pin Codes:** We initiate by extracting the unique customer pin codes from the pincode mapping dataset. This operation culminates in the creation of a new DataFrame, termed as "abc_courier", which serves as a repository for this pin code information.
2. **Selection of Relevant Columns from Courier Invoice Dataset:** Next, we meticulously choose specific columns from the courier invoice dataset. The selected columns include "Order ID", "Customer Pincode", and "Type of Shipment". This curated subset of data is encapsulated within a new DataFrame named "courier_abc".
3. **Merging DataFrames based on Customer Pincode:** Subsequently, we proceed to merge the "courier_abc" DataFrame with the "abc_courier" DataFrame. This merge operation hinges on the "Customer Pincode" column, effectively establishing an association between customer pin codes and their corresponding orders along with shipment types. The resulting merged DataFrame is denoted as "pincodes".

By meticulously executing these steps, we facilitate the integration of courier invoice data with pincode mapping, thereby enabling comprehensive insights into order details vis-à-vis customer locations and shipment types.

Enhancing Data Integration and Weight Calculations in Courier Operations

To further streamline our courier operations, we embark on merging the pin codes with the main dataframe, followed by computing weight in kilograms and determining weight slabs. Let's delve into each step with comprehensive elaboration:

Merging Pin Codes with Main DataFrame
Utilizing the merge() function, we seamlessly integrate the pin codes into our main dataset, denoted as merged_data. This merge operation ensures that each order in our dataset is associated with its respective pin code, facilitating enhanced geographical insights and logistical management.

In [19]:
```
merged2 = merged_data.merge(pincodes, on='Order ID')
```

Computing Weight in Kilograms:

To accurately represent the weight of each order, we calculate the weight in kilograms by dividing the 'Weight (g)' column in the merged2 DataFrame by 1000. This transformation ensures consistency and compatibility with standard weight measurements in logistics.

In [20]:
```
merged2['Weights (Kgs)'] = merged2['Weight (g)'] / 1000
```

Determining Weight Slabs:

Weight slabs are crucial for accurate pricing and logistical planning. We define a custom function, weight_slab(), to categorize weights into appropriate slabs. This function evaluates the decimal part of the weight and rounds it up to the nearest half or whole kilogram, depending on predefined criteria.

In [21]:
```
def weight_slab(weight):
    # Determine the decimal part of the weight
    decimal_part = round(weight % 1, 1)

    # Apply rules to categorize weights into slabs
    if decimal_part == 0.0:
        return weight
    elif decimal_part > 0.5:
        return int(weight) + 1.0
    else:
        return int(weight) + 0.5

# Apply the weight_slab function to create 'Weight Slab (KG)' column
merged2['Weight Slab (KG)'] = merged2['Weights (Kgs)'].apply(weight_slab)
```

Understanding the Weight Slab Determination Function for Shipment Logistics

Function Explanation:
The `weight_slab()` function serves as a pivotal tool for categorizing shipments into appropriate weight slabs based on their weight. Let's dissect its functionality in detail:

Calculation of Remainder:
The function initiates by computing the remainder of the weight divided by 1, with rounding to one decimal place. This step is crucial as it isolates the fractional part of the weight, enabling precise determination of the weight slab.

Handling Different Scenarios:

1. **Remainder Equals 0.0 (Multiple of 1 KG):** If the remainder is precisely 0.0, it indicates that the weight aligns perfectly with a multiple of 1 KG. In such cases, the function straightforwardly returns the weight without any adjustments.
2. **Remainder Greater Than 0.5 (Next Half KG Slab):** When the remainder surpasses 0.5, it signifies that the weight exceeds the next half-KG slab. To accommodate this scenario, the function rounds the weight to the nearest integer and then adds 1.0 to denote advancement to the next heavier slab.
3. **Remainder Less Than or Equal to 0.5 (Current Half-KG Slab):** Conversely, if the remainder is less than or equal to 0.5, it indicates that the weight falls within the current half-KG bracket. In this scenario, the function rounds the weight to the nearest integer and appends 0.5 to signify its position within the current weight slab.

Application:
The `weight_slab()` function encapsulates these intricate rules to precisely categorize shipments into appropriate weight slabs. This categorization is vital for accurate pricing, logistical planning, and optimizing resource allocation in courier operations.

By comprehending the inner workings of this function, stakeholders gain valuable insights into how shipment weights are systematically organized and managed, fostering efficiency and precision in logistical endeavors.

In [22]:
```
courier_invoice = courier_invoice.rename(columns={'Zone': 'Delivery Zone Charged by Courier Company'})
merged2 = merged2.rename(columns={'Zone': 'Delivery Zone As Per ABC'})
merged2 = merged2.rename(columns={'Weight Slab (KG)': 'Weight Slab As Per ABC'})
```

In [24]:
```
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_additional)
    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_additional + rto_additional))
    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
```

Comprehensive Calculation of Expected Charges Based on ABC's Tariffs

Detailed Process Explanation:
The provided code orchestrates a meticulous calculation of expected charges for shipments, leveraging ABC's tariffs as a benchmark. Let's delve into the intricacies of this process step by step:

Iterative Row-wise Calculation:

1. **Iterating Through DataFrame Rows:** The code initiates by traversing through each row of the 'merged2' DataFrame, enabling granular computation of charges for individual shipments.
2. **Retrieval of Tariff Rates and Parameters:** At each iteration, the pertinent rates and parameters from ABC's tariff structure are retrieved. These encompass critical elements such as fixed charges, surcharges per weight tier, and specific considerations for forward and RTO (Return to Origin) shipments, contingent upon the delivery area.
3. **Determination of Weight Slabs:** One pivotal aspect of the calculation involves determining the weight slab applicable to each shipment. This step ensures accurate application of charges based on weight categories.

Consideration of Shipment Type:

1. **Handling 'Forward Charges' Shipments:** For shipments designated as 'Forward Charges', the code dynamically calculates any additional weight beyond the basic weight slab (typically 0.5 KG). Subsequently, the corresponding additional charges are computed and incorporated into the expected charges.
2. **Incorporating 'Forward and RTO Charges' Shipments:** In scenarios involving 'Forward and RTO Charges' shipments, the calculation process extends to include additional charges pertaining to terminal and RTO components. This comprehensive approach ensures meticulous consideration of all relevant factors influencing the final charges.

Storage of Calculated Charges:

1. **Storage in DataFrame:** The culminated expected charges, meticulously computed for each shipment, are meticulously stored in the "Expected charges according to ABC" column of the 'merged2' DataFrame. This strategic storage facilitates seamless comparison between the expected charges and the charges billed by the courier company, enabling rigorous analysis of charge accuracy.

By executing these elaborate calculations and systematic comparisons, stakeholders gain invaluable insights into the accuracy and adherence of the courier company's charges to ABC's tariff structure. This comprehensive approach underscores a commitment to precision and transparency in logistical operations.

In [25]:
```
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.)
0                 117.9
1                 117.9
2                 117.9
3                 117.9
4                 117.9
```

In [26]:
```
df_diff = merged_output
df_diff['Difference (Rs.)'] = df_diff['Billing Amount (Rs.)'] - df_diff['Expected Charge as per ABC']

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
```

Delivery Zone Analysis:

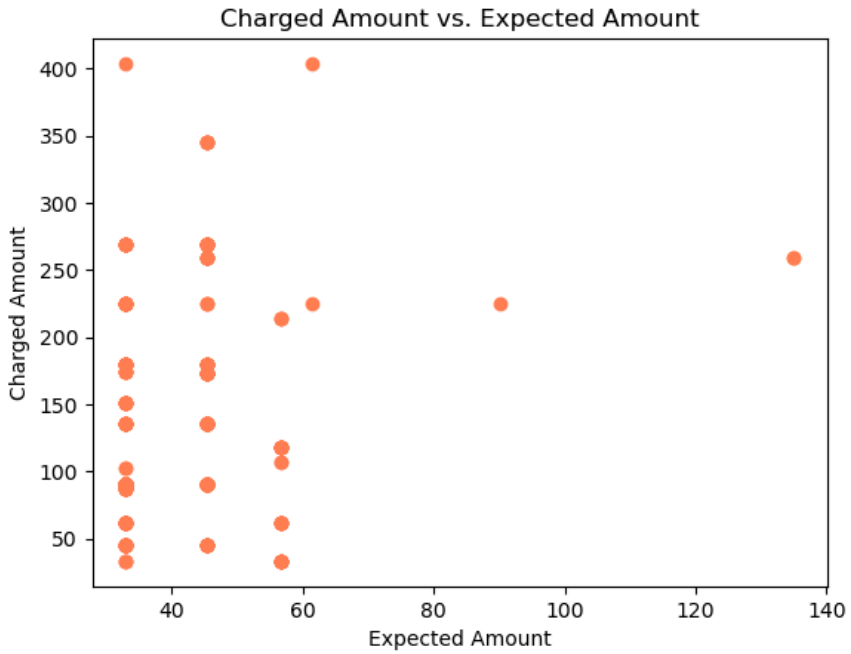
This count plot visualizes the distribution of orders across different delivery zones. Each bar represents a delivery zone, and the height of the bar corresponds to the number of orders associated with that zone. This visualization provides insights into the distribution of orders among various delivery zones, helping you identify zones with higher or lower order volumes.

In [31]:
```
import seaborn as sns

# Plotting a count plot of orders across different delivery zones
sns.countplot(x='Delivery Zone As Per ABC', data=merged_output, palette='viridis')
plt.title('Distribution of Orders Across Delivery Zones')
plt.xlabel('Delivery Zone')
plt.ylabel('Number of Orders')
plt.show()
```

Comparison of Charged Amount vs. Expected Amount:

This scatter plot compares the charged amount with the expected amount for each data point. Each point on the plot represents an order, with its position determined by the expected amount on the x-axis and the charged amount on the y-axis. By observing the distribution of points, you can assess how closely the charged amount aligns with the expected amount. This helps in identifying instances of overcharging or undercharging.

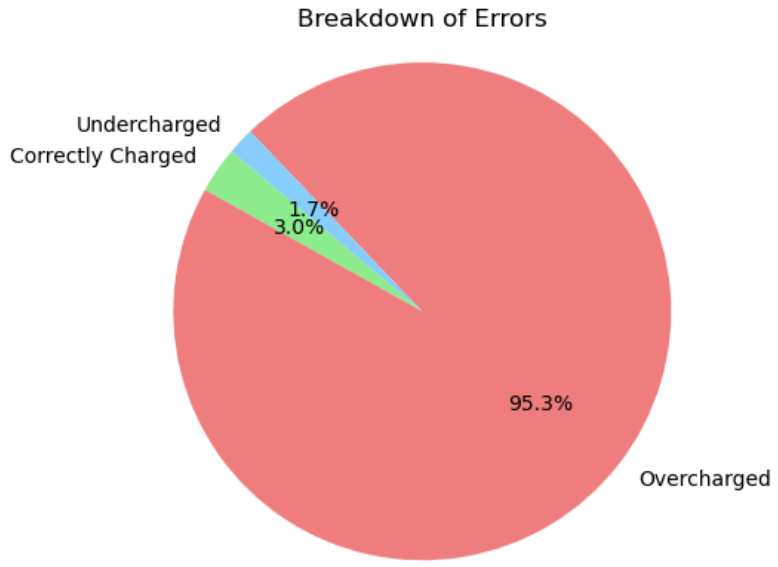
In [33]:
```
# Plotting a scatter plot of charged amount vs. expected amount
plt.scatter(merged_output['Expected Charge as per ABC'], merged_output['Billing Amount (Rs.)'], color='coral')
plt.title('Charged Amount vs. Expected Amount')
plt.xlabel('Expected Amount')
plt.ylabel('Charged Amount')
plt.show()
```



Breakdown of Errors:

This pie chart provides a breakdown of errors in the charging process. It categorizes orders into three groups: correctly charged, overcharged, and undercharged. The size of each slice of the pie represents the proportion of orders falling into that category. This visualization allows you to quickly grasp the distribution of errors and assess the overall accuracy of the charging process.

In [34]:
```
# Creating a pie chart to visualize the breakdown of errors
labels = ['Correctly Charged', 'Overcharged', 'Undercharged']
sizes = [total_correctly_charged, total_overcharged, total_undercharged]
colors = ['lightgreen', 'lightcoral', 'lightskyblue']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title('Breakdown of Errors')
plt.axis('equal')
plt.show()
```



Analyzing the Accuracy of B2B Courier Charges:

In evaluating the accuracy of B2B courier charges, it's imperative to juxtapose the charged prices with the expected prices derived from meticulous calculations based on predefined tariffs and parameters. Here's a concise summary of how this comparison unfolds:

1. **Charged Prices:** These denote the actual prices billed by the courier company for the services rendered. They represent the financial transactions recorded in the company's billing system.

2. **Expected Prices:** Derived from comprehensive calculations based on ABC's tariffs and parameters, the expected prices serve as a benchmark against which the charged prices are evaluated. These prices are systematically computed for each shipment, factoring in variables such as weight slabs, shipment types, and delivery areas.

Evaluation Criteria:
To assess the accuracy of B2B courier charges, the following criteria are typically considered:

- **Match between Charged and Expected Prices:** A direct comparison is made between the charged prices and the expected prices derived from the tariff calculations. Any discrepancies between these two sets of prices are scrutinized to ascertain the degree of accuracy in the billing process.
- **Consistency Across Shipments:** The consistency of charged prices with expected prices is evaluated across a range of shipments, spanning different weight categories, shipment types, and delivery areas. Consistency signifies adherence to established tariff structures and parameters.
- **Identification of Deviations:** Deviations or discrepancies between charged and expected prices are meticulously identified and analyzed. These deviations could stem from various factors such as errors in billing, inaccuracies in tariff application, or discrepancies in weight calculations.

The accuracy of B2B courier charges hinges on the alignment between the charged prices and the expected prices derived from predefined tariffs and parameters. By systematically comparing these two sets of prices and scrutinizing any deviations, stakeholders can assess the efficacy and reliability of the courier company's billing practices. This analysis serves as a vital tool for ensuring transparency, accountability, and trustworthiness in B2B courier transactions.

In [27]:
```
# 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.)'].sum())
amount_undercharged = df_new[df_new['Difference (Rs.)'] < 0]['Difference (Rs.)'].sum()
amount_correctly_charged = df_new[df_new['Difference (Rs.)'] == 0]['Expected Charge as per ABC'].sum()

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

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
```

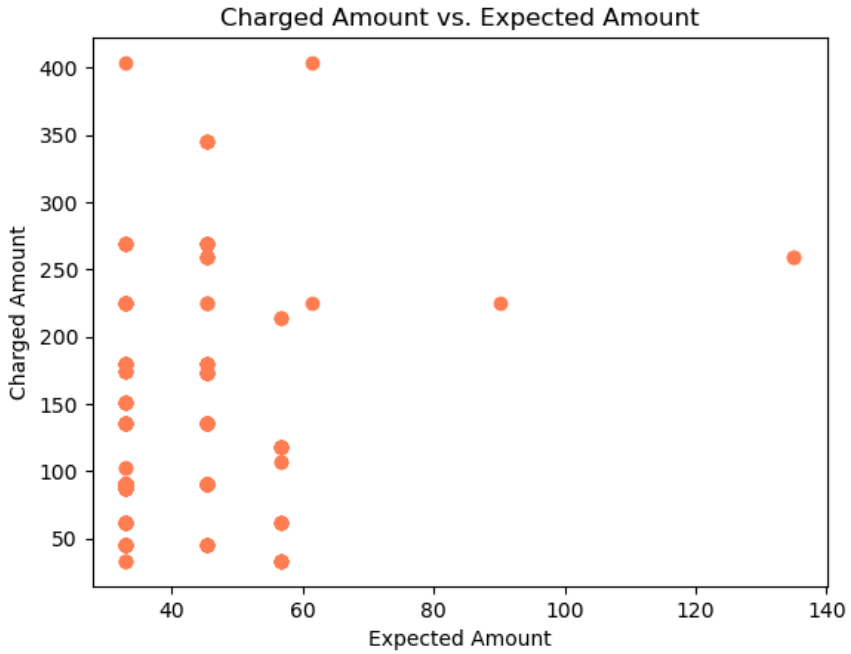
In [28]:
```
import plotly.graph_objects as go
fig = go.Figure(data=go.Pie(labels=df_summary['Description'],
                            values=df_summary['Count'],
                            textinfo='label+percent',
                            hole=0.4))
fig.update_layout(title='Proportion')

fig.show()
```

Total Orders

Total Orders v

In [30]:
```
# Plotting a scatter plot of charged amount vs. expected amount
plt.scatter(merged_output['Expected Charge as per ABC'], merged_output['Billing Amount (Rs.)'], color='coral')
plt.title('Charged Amount vs. Expected Amount')
```

```
plt.xlabel('Expected Amount')
plt.ylabel('Charged Amount')
plt.show()
```



Summary

The B2B Courier Charges Accuracy Analysis delves into the intricate realm of ensuring precise billing accuracy within B2B courier transactions, a critical facet of modern business operations. In the fast-paced landscape of e-commerce, where timely order fulfillment is paramount, businesses rely heavily on courier companies to deliver their products to customers efficiently. However, managing the charges incurred from these courier services can pose significant challenges, particularly when dealing with high order volumes and complex pricing structures.

At the heart of this analysis lies the need to assess the alignment between the fees charged by courier companies and the expected charges derived from predefined parameters and tariffs. The primary objective is to ascertain that businesses are accurately billed for the services provided by their courier partners. Discrepancies between estimated charges and actual billing amounts for the same invoice often emerge as a pervasive issue, highlighting the critical importance of addressing billing accuracy in real-time.

To tackle this challenge, businesses embark on a multifaceted analytical journey that involves data integration, meticulous calculation, and comprehensive analysis. Through the integration of disparate datasets containing order reports, courier invoices, and pincode mappings, businesses consolidate crucial information necessary for accurate billing assessments. Leveraging Python-based analytical tools and techniques, stakeholders manipulate and analyze this data to derive meaningful insights into billing accuracy.

One fundamental aspect of this analysis involves the calculation of weight slabs, a pivotal factor in determining shipping costs. By converting weights from grams to kilograms and applying predefined rules for weight slab categorization, businesses can accurately assess shipping charges based on weight categories. Additionally, stakeholders develop analytical models to compute expected charges based on predetermined tariff structures, factoring in variables such as weight slabs, shipment types, and delivery areas.

Through meticulous comparison and analysis of charged fees versus expected charges, stakeholders gain valuable insights into the accuracy of billing practices. Discrepancies are identified, scrutinized, and addressed, enabling businesses to streamline billing processes, enhance financial integrity, and optimize operational efficiency. This iterative process of analysis and improvement fosters a culture of continuous optimization, ensuring that businesses remain agile and responsive in an ever-evolving business landscape.

In summary, the B2B Courier Charges Accuracy Analysis serves as a cornerstone in the quest for operational excellence and financial transparency within the realm of B2B courier transactions. Through data-driven insights and analytical rigor, businesses can navigate the complexities of billing accuracy with confidence, ultimately driving sustainable growth and success in the competitive marketplace.