In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In [2]:
```
data=pd.read_csv('train.csv')
```

In [3]:
```
data.head(5)
```

Out[3]:

	Period_No	Facility_No	Facility_Category	City_Zip_Code	Operational_Region_Coverage_Ar
0	1	324	c1	977	1
1	1	10	c3	0	
2	1	99	c3	0	1
3	1	95	c3	17	
4	1	128	c3	17	

In [4]:
```
data.describe()
```

Out[4]:

	Period_No	Facility_No	City_Zip_Code	Operational_Region_Coverage_Area	Bil
count	321437.000000	321437.000000	321437.000000	321437.000000	3
mean	50.054129	420.232991	271.276987	418.082445	
std	28.779770	1575.034508	403.842530	2662.500716	
min	1.000000	3.000000	0.000000	1.000000	
25%	25.000000	36.000000	0.000000	13.000000	
50%	50.000000	78.000000	17.000000	48.000000	
75%	75.000000	132.000000	582.000000	140.000000	
max	100.000000	12390.000000	977.000000	23595.000000	

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 321437 entries, 0 to 321436
Data columns (total 13 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Period_No                         321437 non-null  int64  
 1   Facility_No                       321437 non-null  int64  
 2   Facility_Category                 321437 non-null  object 
 3   City_Zip_Code                     321437 non-null  int64  
 4   Operational_Region_Coverage_Area  321437 non-null  int64  
 5   Billing_Amount                    321437 non-null  float64
 6   Labelled_Price                    321437 non-null  float64
 7   Custom_Promoted                   321437 non-null  int64  
 8   Promoted                          321437 non-null  int64  
 9   Search_Promotions                 321437 non-null  int64  
 10  Orders_Count                      321437 non-null  int64  
 11  Course                            321437 non-null  object 
 12  Flavour_Profile                   321437 non-null  object 
dtypes: float64(2), int64(8), object(3)
memory usage: 31.9+ MB
```

In [6]:
```
data.columns=[col.lower().replace("_"," ") for col in data.columns]
```

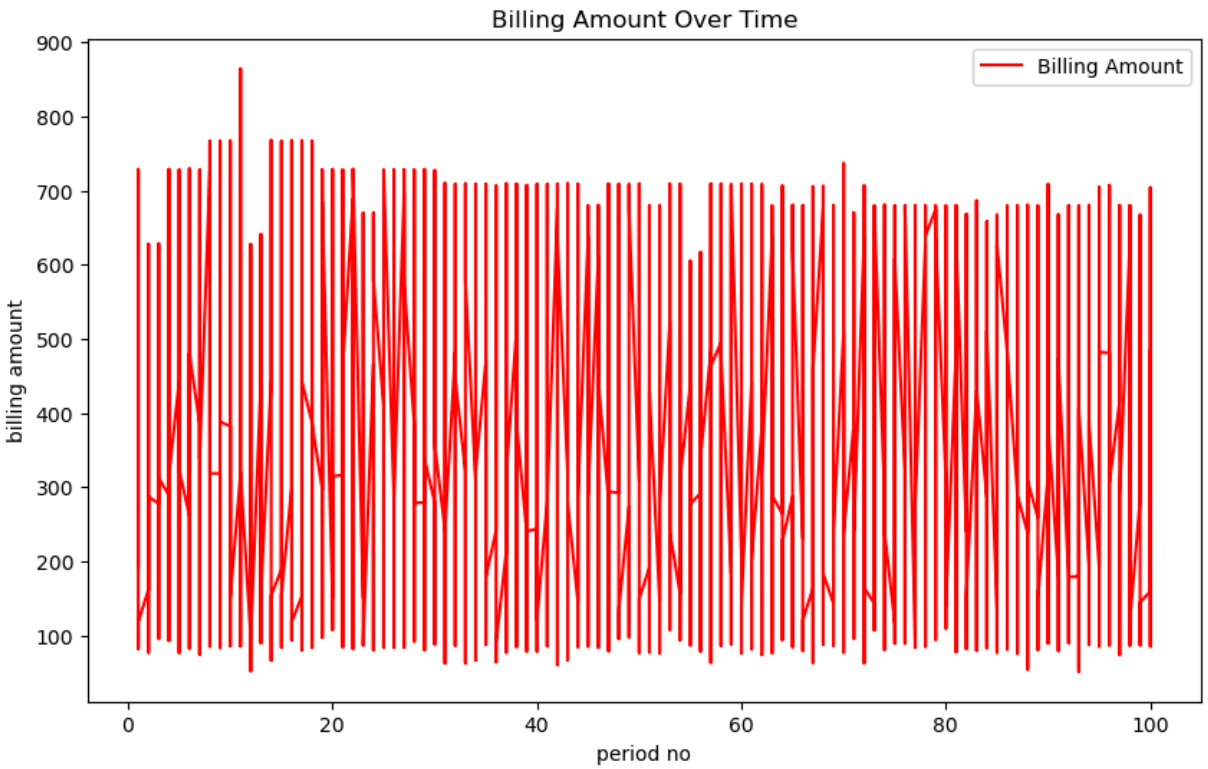
In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 321437 entries, 0 to 321436
Data columns (total 13 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   period no                         321437 non-null  int64  
 1   facility no                       321437 non-null  int64  
 2   facility category                 321437 non-null  object 
 3   city zip code                     321437 non-null  int64  
 4   operational region coverage area  321437 non-null  int64  
 5   billing amount                    321437 non-null  float64
 6   labelled price                    321437 non-null  float64
 7   custom promoted                   321437 non-null  int64  
 8   promoted                          321437 non-null  int64  
 9   search promotions                 321437 non-null  int64  
 10  orders count                      321437 non-null  int64  
 11  course                            321437 non-null  object 
 12  flavour profile                   321437 non-null  object 
dtypes: float64(2), int64(8), object(3)
memory usage: 31.9+ MB
```

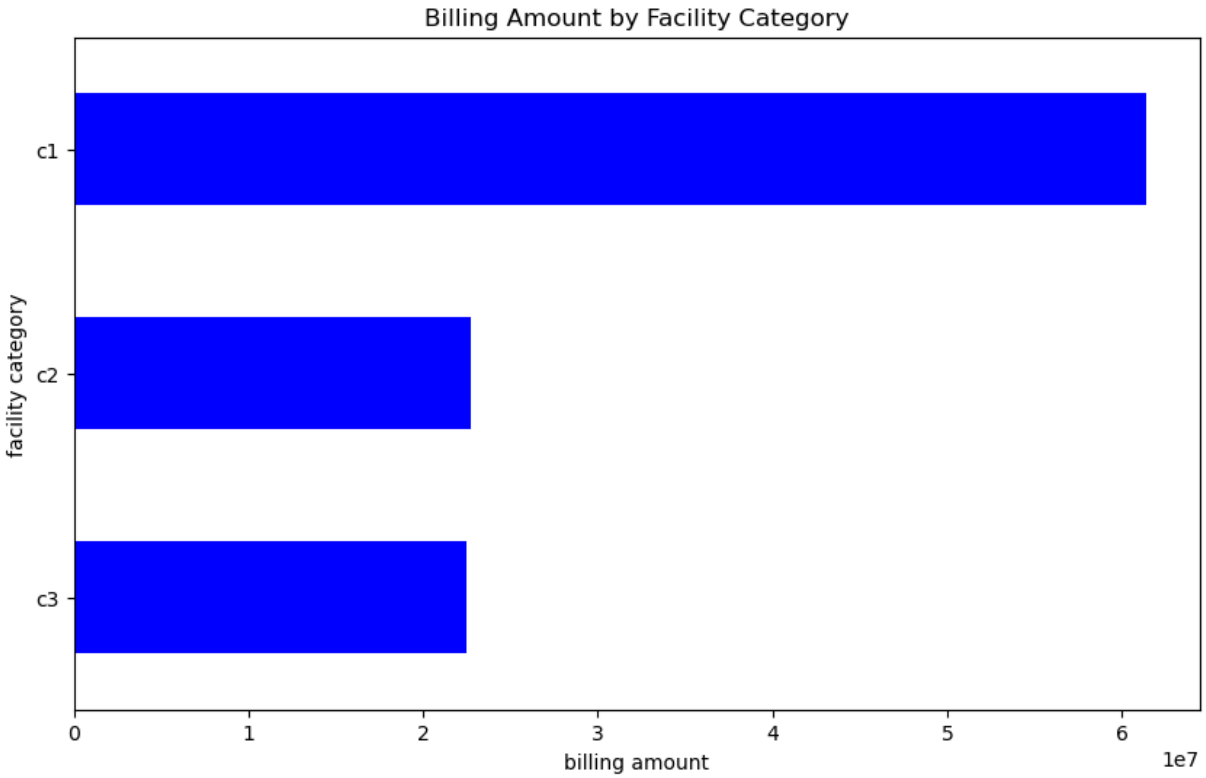
In [8]:
```
data.isnull().sum()
```

Out[8]:
```
period no                          0
facility no                        0
facility category                  0
city zip code                      0
operational region coverage area   0
billing amount                     0
labelled price                     0
custom promoted                    0
promoted                           0
search promotions                  0
orders count                       0
course                             0
flavour profile                    0
dtype: int64
```

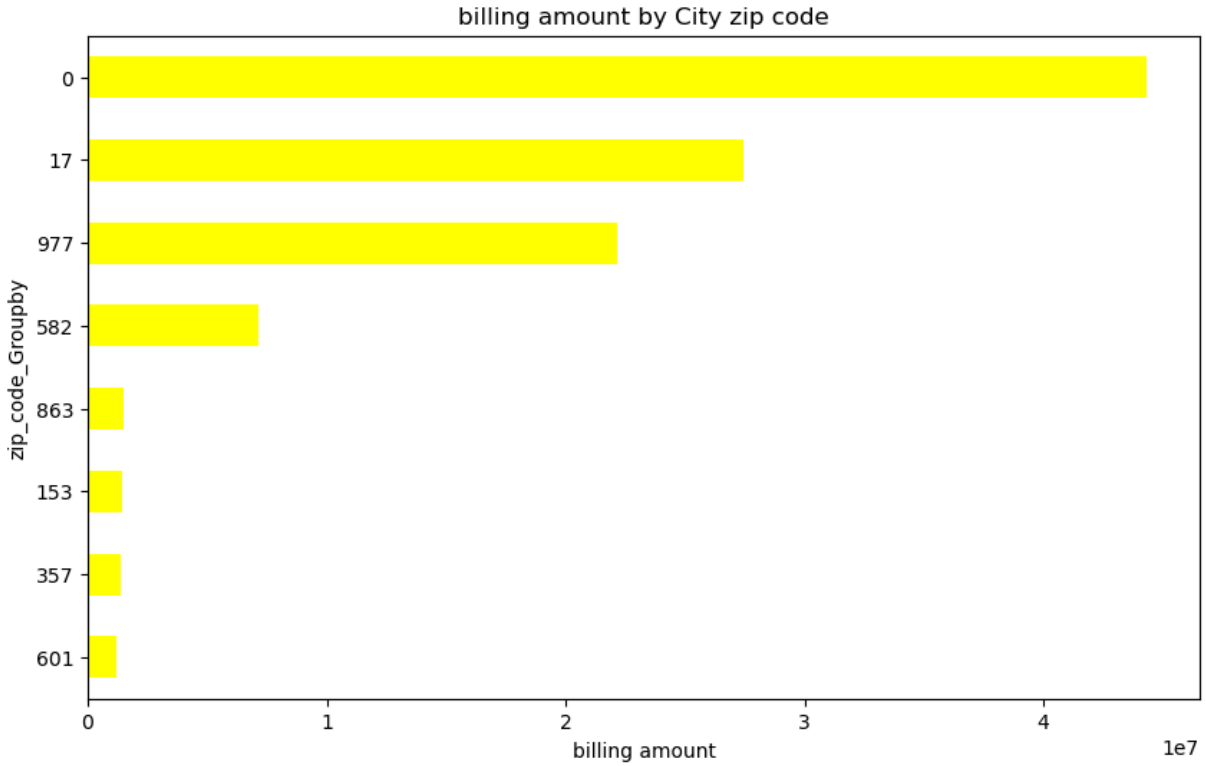
In [9]:
```
plt.figure(figsize=(10,6))
plt.plot(data['period no'],data['billing amount'],label='Billing Amount',color='red
plt.xlabel('period no')
plt.ylabel('billing amount')
plt.title('Billing Amount Over Time')
plt.legend()
plt.show()
```



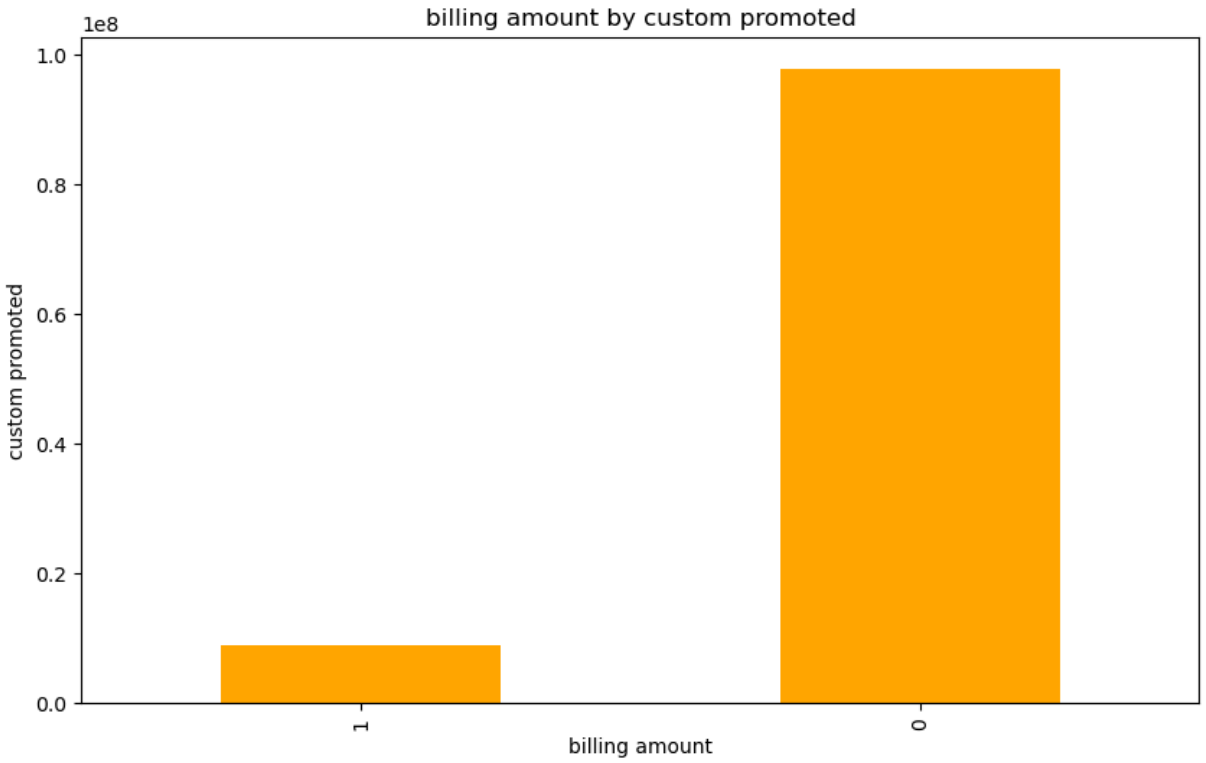
In [10]:
```
facility_groupbed=data.groupby('facility category')['billing amount'].sum()

plt.figure(figsize=(10,6))
facility_groupbed.sort_values().plot(kind='barh',color='blue')
plt.xlabel('billing amount')
plt.ylabel('facility category')
plt.title('Billing Amount by Facility Category')
plt.show()
```



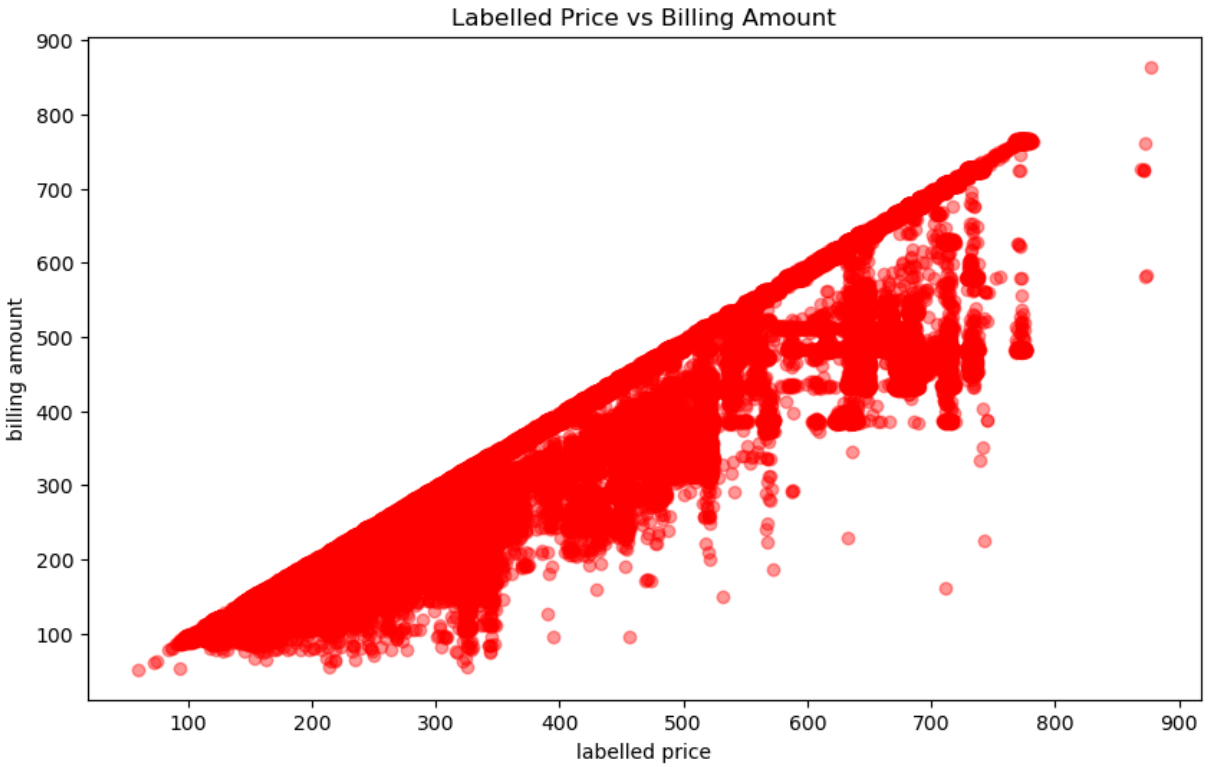
In [12]:
```
zip_code_Groupby=data.groupby('city zip code')['billing amount'].sum()
plt.figure(figsize=(10,6))
zip_code_Groupby.sort_values().plot(kind='barh',color='Yellow')
plt.ylabel('zip_code_Groupby')
plt.xlabel('billing amount')
plt.title('billing amount by City zip code')
plt.show()
```



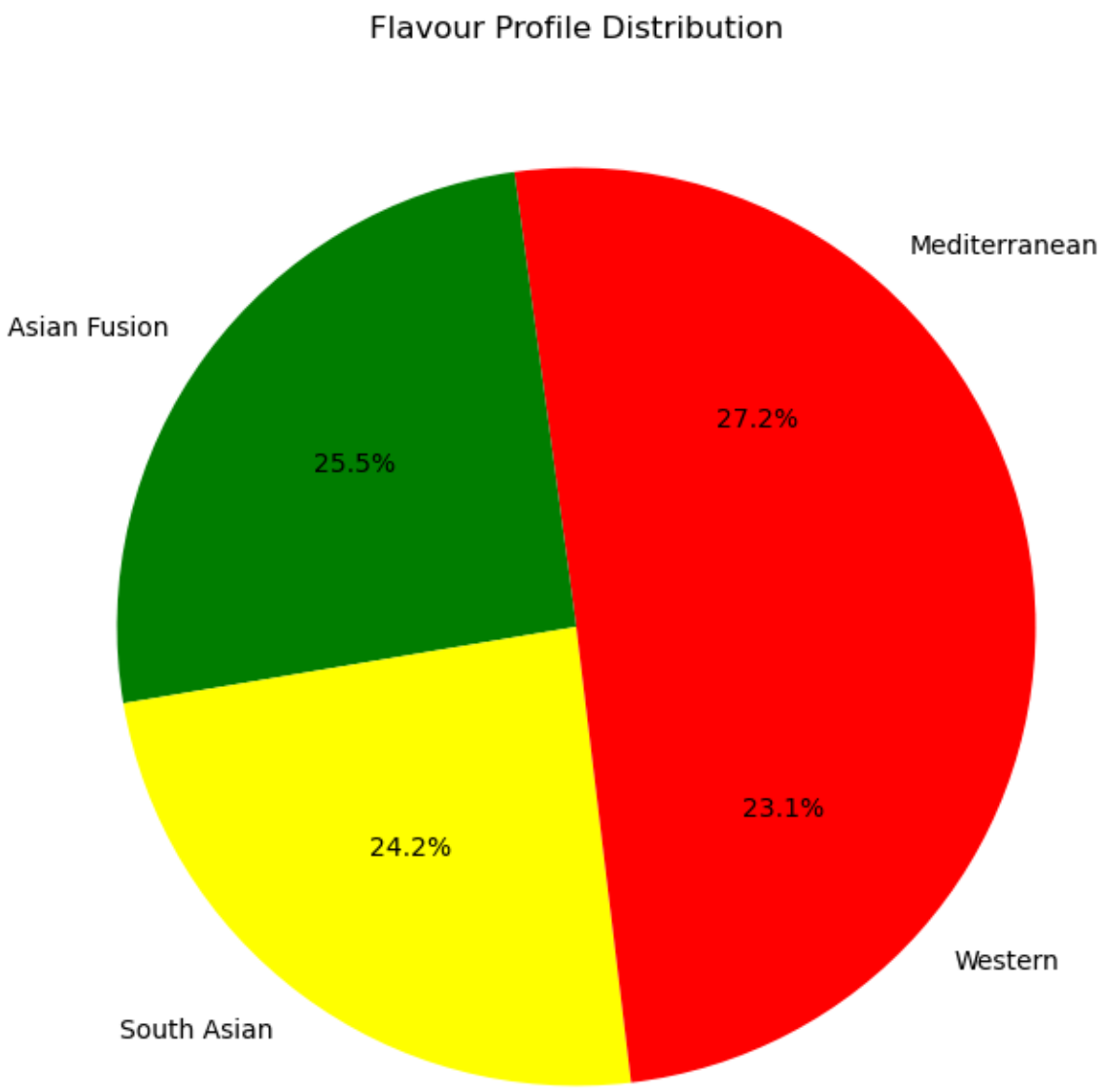
In [13]:
```
custom_promoted_groupby=data.groupby('custom promoted')['billing amount'].sum()
plt.figure(figsize=(10,6))
custom_promoted_groupby.sort_values().plot(kind='bar',color='orange')
plt.ylabel('custom promoted')
plt.xlabel('billing amount')
plt.title('billing amount by custom promoted')
plt.show()
```



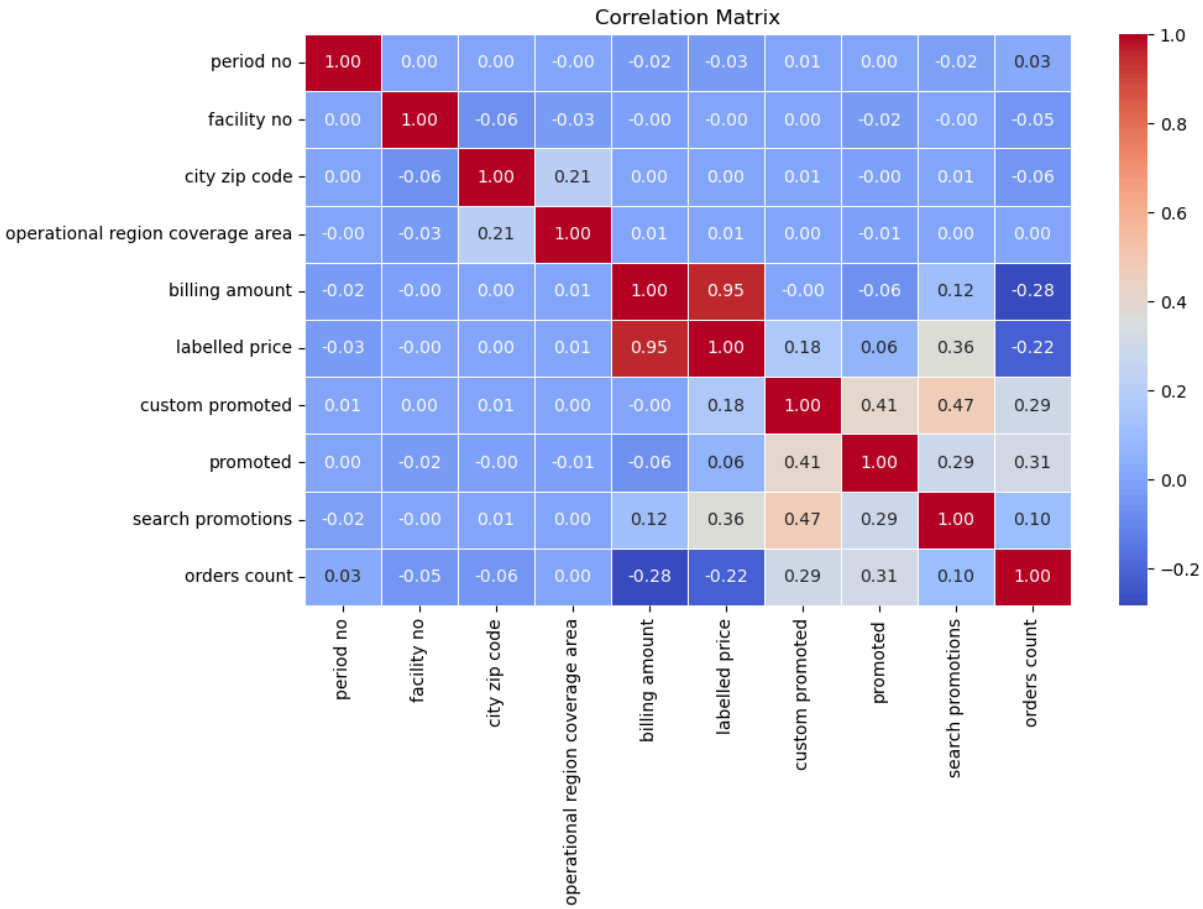
In [14]:
```
plt.figure(figsize=(10,6))
plt.scatter(data['labelled price'],data['billing amount'],alpha=0.4,color='red')
plt.title('Labelled Price vs Billing Amount')
plt.xlabel('labelled price')
plt.ylabel('billing amount')
plt.show()
```



In [15]:
```
flavour_profile_counts=data['flavour profile'].value_counts()
plt.figure(figsize=(8,8))
flavour_profile_counts.plot(kind='pie',autopct='%1.1f%%',colors=['red','green','yel
plt.title('Flavour Profile Distribution')
plt.ylabel('')
plt.show()
```

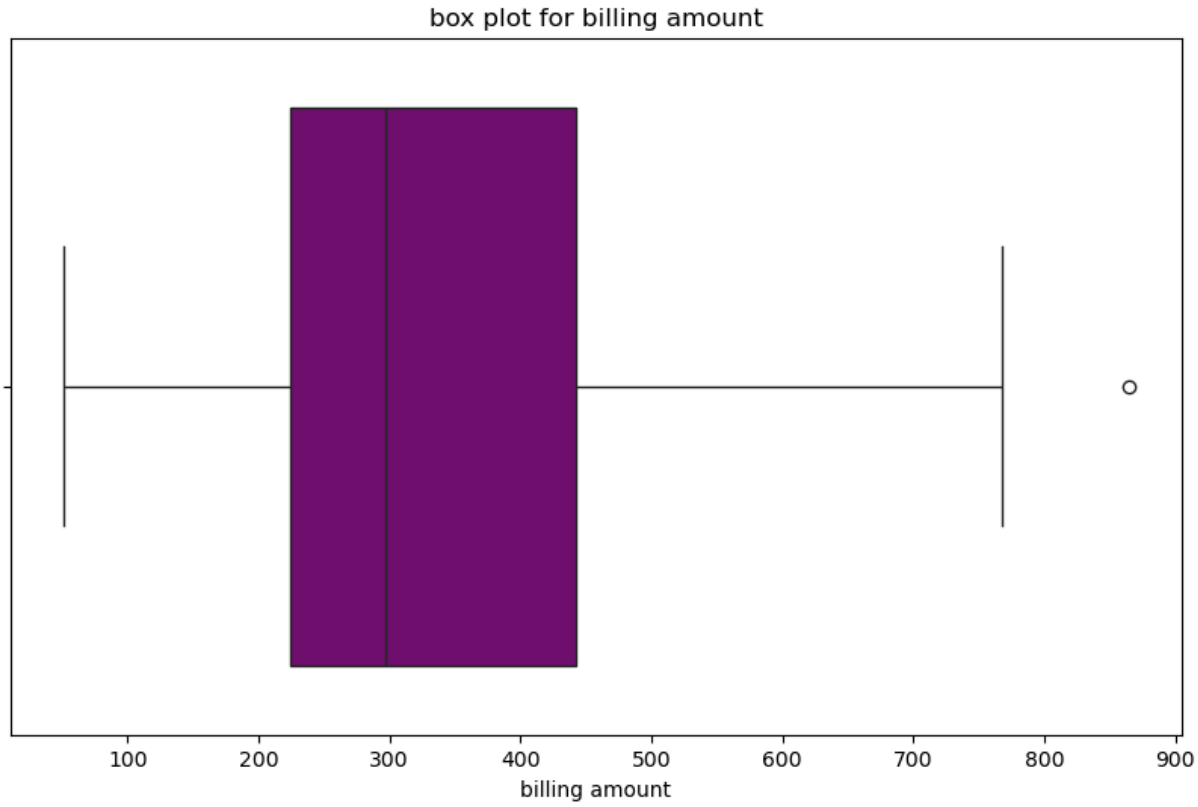


In [16]:
```
numeric_data = data.select_dtypes(include=['float64', 'int64'])

corr_matrix = numeric_data.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title("Correlation Matrix")
plt.show()

numeric_data.info()
```

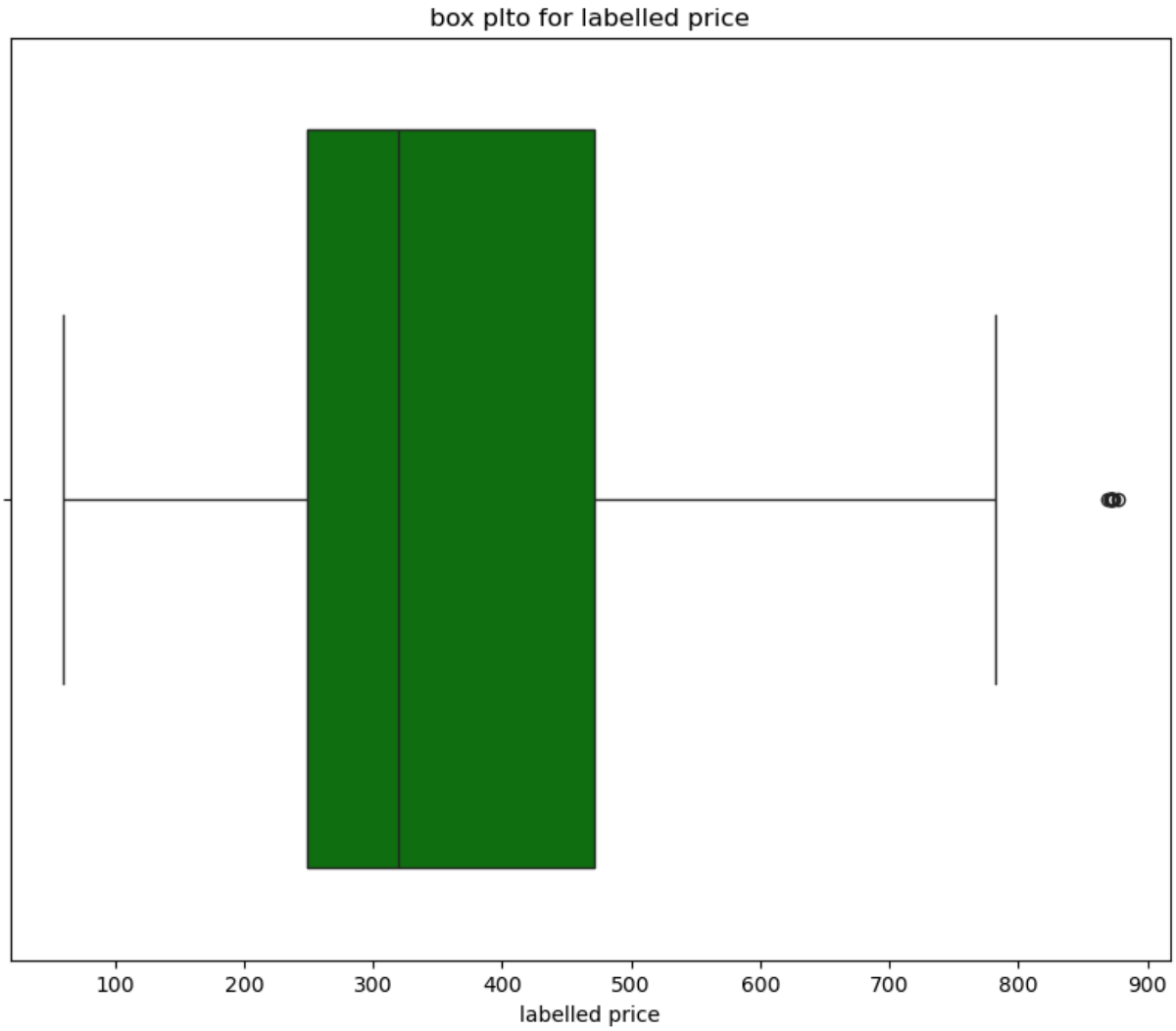



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 321437 entries, 0 to 321436
Data columns (total 10 columns):
 #   Column                           Non-Null Count   Dtype  
---  ------                           --------------   -----  
 0   period no                        321437 non-null  int64  
 1   facility no                      321437 non-null  int64  
 2   city zip code                    321437 non-null  int64  
 3   operational region coverage area 321437 non-null  int64  
 4   billing amount                   321437 non-null  float64
 5   labelled price                   321437 non-null  float64
 6   custom promoted                  321437 non-null  int64  
 7   promoted                         321437 non-null  int64  
 8   search promotions                321437 non-null  int64  
 9   orders count                     321437 non-null  int64  
dtypes: float64(2), int64(8)
memory usage: 24.5 MB
```

In [17]:
```
plt.figure(figsize=(10,6))
sns.boxplot(x=data['billing amount'],color='purple')
plt.title('box plot for billing amount')
plt.show()
```

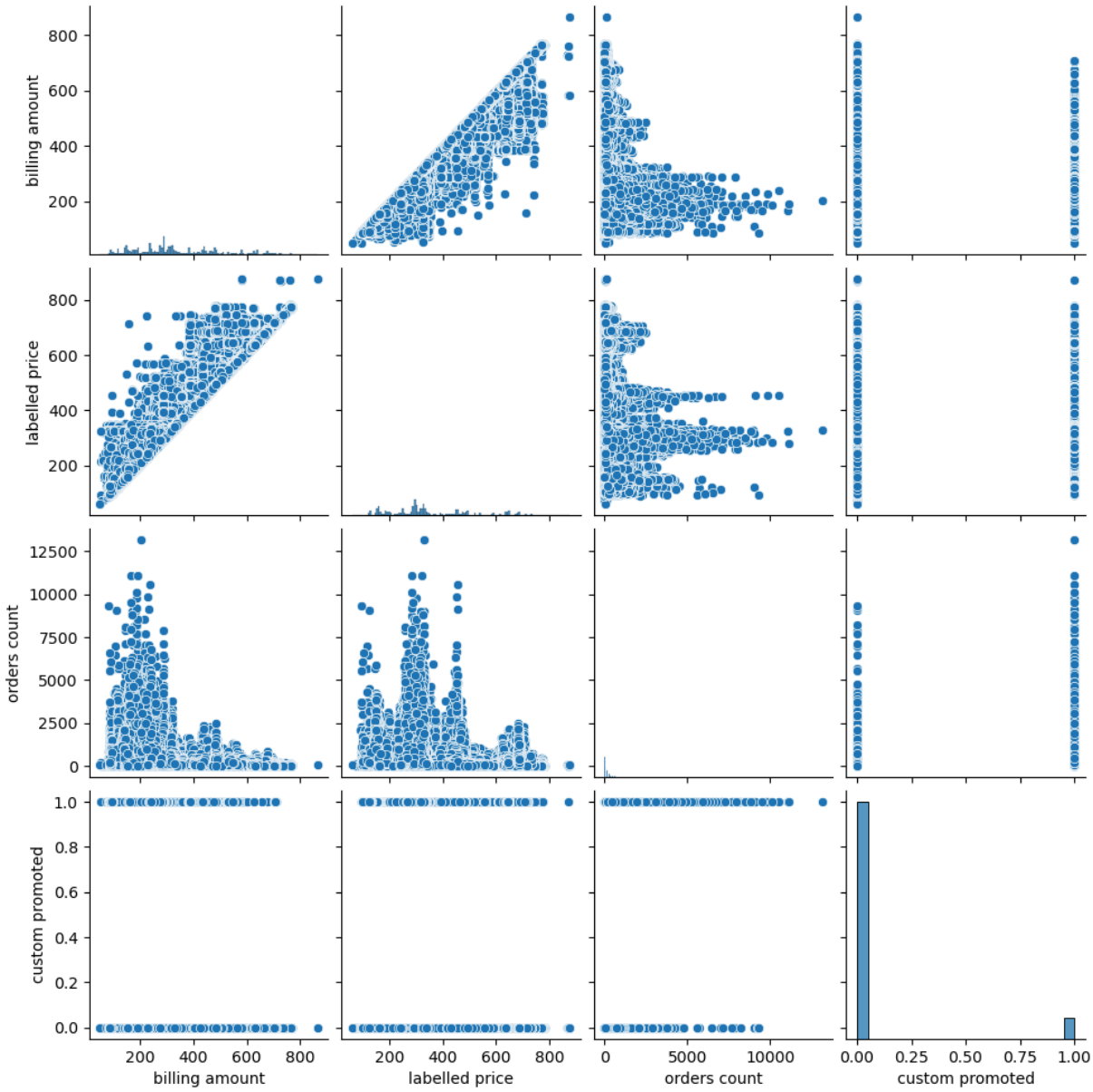


In [18]:
```
plt.figure(figsize=(10,8))
sns.boxplot(x=data['labelled price'],color='green')
plt.title('box plto for labelled price')
plt.show()
```

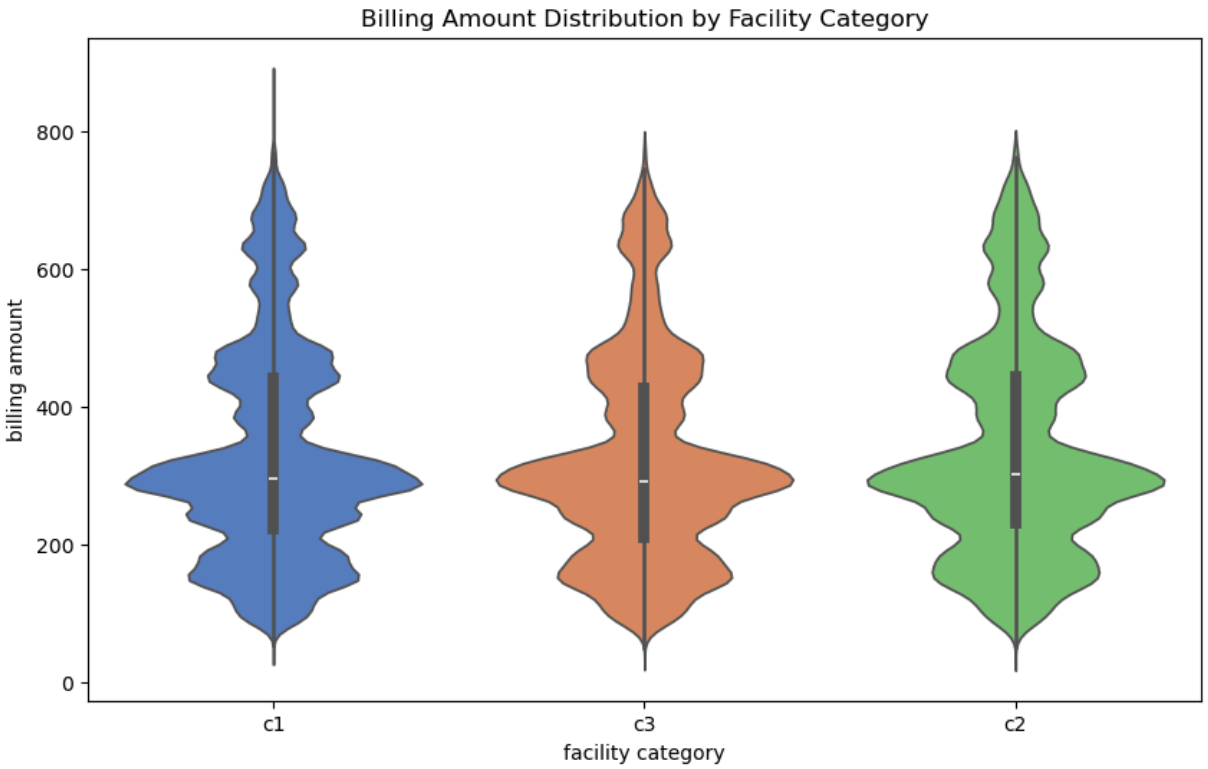


In [ ]:

In [19]:
```
# Pairplot for selected numerical columns
sns.pairplot(data[['billing amount', 'labelled price', 'orders count', 'custom prom
plt.show()
```



In [20]:
```
plt.figure(figsize=(10,6))
sns.violinplot(y='billing amount',x='facility category',hue='facility category',dat
plt.title('Billing Amount Distribution by Facility Category')
plt.show()
```



In [21]:
```
data.describe()
```

Out[21]:

	period no	facility no	city zip code	operational region coverage area	billing amount	labelle
count	321437.000000	321437.000000	321437.000000	321437.000000	321437.000000	321437.
mean	50.054129	420.232991	271.276987	418.082445	331.620562	362.
std	28.779770	1575.034508	403.842530	2662.500716	154.058035	163.
min	1.000000	3.000000	0.000000	1.000000	51.211374	59.
25%	25.000000	36.000000	0.000000	13.000000	223.793489	249.
50%	50.000000	78.000000	17.000000	48.000000	297.266165	318.
75%	75.000000	132.000000	582.000000	140.000000	441.947052	471.
max	100.000000	12390.000000	977.000000	23595.000000	864.305642	876.

In [22]:
```
data.describe(include=['object'])
```

Out[22]:

	facility category	course	flavour profile
count	321437	321437	321437
unique	3	14	4
top	c1	Smoothies & Juices	Mediterranean
freq	184775	88363	87284

In [23]:
```
from scipy.stats import chi2_contingency
contigency_table=pd.crosstab(data['facility category'],data['promoted'])
```

In [24]:
```
chi2, p, _, _ =chi2_contingency(contigency_table)
print(f"Chi-Squre Statistic :{chi2},p-value:{p}")
```

```
Chi-Squre Statistic :232.52361509073802,p-value:3.2220963797502046e-51
```

In [25]:
```
contigency_table
```

Out[25]:

promoted	0	1
facility category		
c1	163822	20953
c2	59621	7395
c3	63203	6443

In [26]:
```
from scipy.stats import f_oneway
categories =[data[data['facility category']==category]['billing amount'] for catego
anova_result = f_oneway(*categories)
print(f"ANOVA result: F-statistic = {anova_result.statistic}, p-value = {anova_resu
```

```
ANOVA result: F-statistic = 178.13982937523988, p-value = 4.761014109448446e-78
```

In [27]:
```
train_data=pd.read_csv('train.csv')
```

In [28]:
```
test_data=pd.read_csv('test.csv')
```

In [29]:
```
# Standardize column names
train_data.columns = train_data.columns.str.strip().str.lower().str.replace(" ", "_
test_data.columns = test_data.columns.str.strip().str.lower().str.replace(" ", "_")

# Check for 'orders_count' column
required_column = 'orders_count'

if required_column not in train_data.columns:
    raise KeyError(f"'{required_column}' is missing in train_data.")
if required_column not in test_data.columns:
    print(f"'{required_column}' is missing in test_data. Filling with default value
    test_data[required_column] = 0  # Default value; adjust as needed

# Convert 'orders_count' to numeric
```

```
train_data[required_column] = pd.to_numeric(train_data[required_column], errors='co
test_data[required_column] = pd.to_numeric(test_data[required_column], errors='coer

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')
train_data[required_column] = imputer.fit_transform(train_data[[required_column]])
test_data[required_column] = imputer.transform(test_data[[required_column]])

# Print success message
print(f"'{required_column}' column successfully processed.")
```

```
'orders_count' is missing in test_data. Filling with default value 0.
'orders_count' column successfully processed.
```

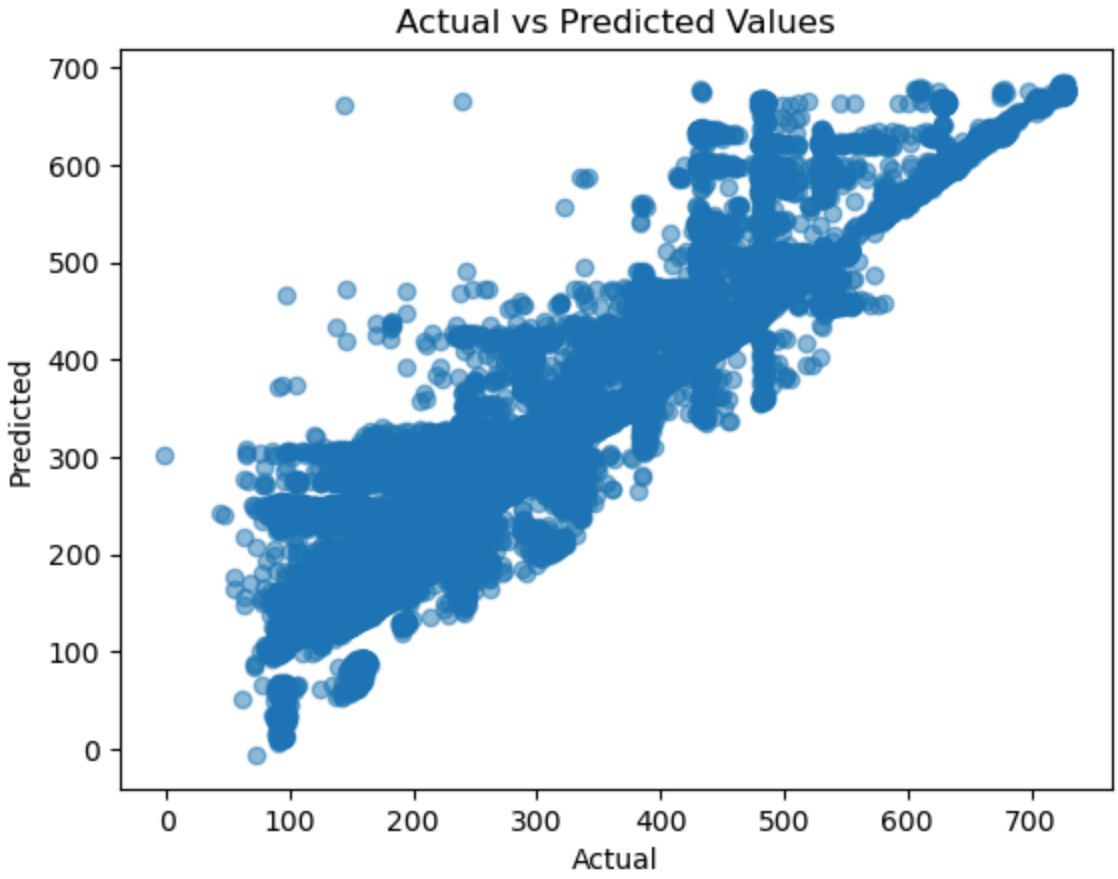
In [30]:

```
# Standardize column names for train_data and test_data
train_data.columns = train_data.columns.str.strip().str.lower().str.replace(" ", "_
test_data.columns = test_data.columns.str.strip().str.lower().str.replace(" ", "_")

# Check column names
print("Columns in train_data:", train_data.columns)
print("Columns in test_data:", test_data.columns)

# Ensure 'course' column exists
course_column_name = 'course'  # Adjust if the column name is different

if course_column_name not in train_data.columns or course_column_name not in test_d
    raise KeyError(f"'{course_column_name}' column not found in datasets. Please ch

# Fix orders_count if numeric but stored as string
train_data['orders_count'] = pd.to_numeric(train_data['orders_count'], errors='coer
test_data['orders_count'] = pd.to_numeric(test_data['orders_count'], errors='coerce

# Handle missing values using SimpleImputer
imputer = SimpleImputer(strategy='mean')  # Replace NaN with the mean
train_data['orders_count'] = imputer.fit_transform(train_data[['orders_count']])
test_data['orders_count'] = imputer.transform(test_data[['orders_count']])

# One-hot encode the 'course' column
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Fit-transform for train_data and transform for test_data
course_encoded_train = ohe.fit_transform(train_data[[course_column_name]])
course_encoded_test = ohe.transform(test_data[[course_column_name]])

# Convert encoded arrays to DataFrame
course_columns = ohe.get_feature_names_out([course_column_name])
course_encoded_train_df = pd.DataFrame(course_encoded_train, columns=course_columns
course_encoded_test_df = pd.DataFrame(course_encoded_test, columns=course_columns,

# Concatenate the encoded data with original DataFrame
train_data = pd.concat([train_data.drop(columns=[course_column_name]), course_encod
test_data = pd.concat([test_data.drop(columns=[course_column_name]), course_encoded

# Feature columns after preprocessing
feature_columns = ['orders_count', 'labelled_price', 'custom_promoted'] + list(cour
```

```
# Select features and target for training and testing
X_train = train_data[feature_columns]
y_train = train_data['billing_amount']

X_test = test_data[feature_columns]
y_test = test_data['billing_amount']

# Ensure no NaN values remain
if X_train.isna().any().any() or X_test.isna().any().any():
    raise ValueError("NaN values detected after preprocessing.")

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

```
Columns in train_data: Index(['period_no', 'facility_no', 'facility_category', 'city
_zip_code',
       'operational_region_coverage_area', 'billing_amount', 'labelled_price',
       'custom_promoted', 'promoted', 'search_promotions', 'orders_count',
       'course', 'flavour_profile'],
      dtype='object')
Columns in test_data: Index(['period_no', 'facility_no', 'facility_category', 'city_
zip_code',
       'operational_region_coverage_area', 'billing_amount', 'labelled_price',
       'custom_promoted', 'promoted', 'search_promotions', 'course',
       'flavour_profile', 'orders_count'],
      dtype='object')
Mean Squared Error: 1528.3190109914624
```

In [31]:
```
# Plot actual vs predicted
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted Values")
plt.show()
```



In [ ]:

In [32]:
```
# Assuming your target variable ('billing_amount') is converted into categorical cl
# For instance, create binary classes (high/low billing amount) or use predefined c
# Example: Convert 'billing_amount' into binary labels
threshold = train_data['billing_amount'].median()  # Using median as a threshold
train_data['billing_class'] = (train_data['billing_amount'] > threshold).astype(int
test_data['billing_class'] = (test_data['billing_amount'] > threshold).astype(int)

# Define the feature columns and target
feature_columns = ['orders_count', 'labelled_price', 'custom_promoted', 'promoted']
X_train = train_data[feature_columns]
y_train = train_data['billing_class']

X_test = test_data[feature_columns]
y_test = test_data['billing_class']

# Fit a classification model (e.g., Logistic Regression)
from sklearn.linear_model import LogisticRegression

classifier = LogisticRegression()
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Compute metrics
accuracy = accuracy_score(y_test, y_pred)
```

```
classification_rep = classification_report(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Print metrics
print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.9545354450749556
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96     61912
           1       0.95      0.96      0.95     56158

    accuracy                           0.95    118070
   macro avg       0.95      0.95      0.95    118070
weighted avg       0.95      0.95      0.95    118070

Confusion Matrix:
[[58961  2951]
 [ 2417 53741]]
```

In [ ]:

In [ ]:

In [ ]: