Ecommerce_python-sql

[1]:
```
import pandas as pd
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('customers.csv', 'customers'),
    ('orders.csv', 'orders'),
    ('sellers.csv', 'sellers'),
    ('products.csv', 'products'),
    ('geolocation.csv', 'geolocation'),
    ('payments.csv', 'payments'),
    ('order_items.csv', 'order_items') # Added payments.csv for specific␣
 ↪handling
]

# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',  # Replace with your external MySQL host
    user='root',       # Replace with your MySQL username
    password='root',   # Replace with your MySQL password
    database='E_commerce'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:/Users/gnave/OneDrive/Desktop/E-commerce'  # Update this path␣
 ↪to where your files

def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Replace NaN with None to handle SQL NULL
    df = df.where(pd.notnull(df), None)

    # Debugging: Check for NaN values
    print(f"Processing {csv_file}")
    print(f"NaN values before replacement:\n{df.isnull().sum()}\n")

    # Clean column names
    df.columns = [col.replace(' ', '_').replace('-', '_').replace('.', '_') for␣
 ↪col in df.columns]

    # Generate the CREATE TABLE statement with appropriate data types
    columns = ', '.join([f'`{col}` {get_sql_type(df[col].dtype)}' for col in df.
 ↪columns])
    create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}`␣
 ↪({columns})'
    cursor.execute(create_table_query)

    # Insert DataFrame data into the MySQL table
    for _, row in df.iterrows():
        # Convert row to tuple and handle NaN/None explicitly
        values = tuple(None if pd.isna(x) else x for x in row)
        sql = f"INSERT INTO `{table_name}` ({', '.join(['`' + col + '`' for col␣
 ↪in df.columns])}) VALUES ({', '.join(['%s'] * len(row))})"
        cursor.execute(sql, values)

    # Commit the transaction for the current CSV file
    conn.commit()

# Close the connection
conn.close()
```

```
Processing customers.csv
NaN values before replacement:
customer_id                 0
customer_unique_id          0
customer_zip_code_prefix    0
```

```
customer_city                  0
customer_state                 0
dtype: int64

Processing  orders.csv
NaN values before replacement:
order_id                          0
customer_id                       0
order_status                      0
order_purchase_timestamp          0
order_approved_at               160
order_delivered_carrier_date   1783
order_delivered_customer_date  2965
order_estimated_delivery_date     0
dtype: int64

Processing sellers.csv
NaN values before replacement:
seller_id                  0
seller_zip_code_prefix     0
seller_city                0
seller_state               0
dtype: int64

Processing  products.csv
NaN values before replacement:
product_id                      0
product category              610
product_name_length           610
product_description_length    610
product_photos_qty            610
product_weight_g                2
product_length_cm               2
product_height_cm               2
product_width_cm                2
dtype: int64

Processing  geolocation.csv
NaN values before replacement:
geolocation_zip_code_prefix     0
geolocation_lat                 0
geolocation_lng                 0
geolocation_city                0
geolocation_state               0
dtype: int64

Processing  payments.csv
NaN values before replacement:
```

```
order_id                0
payment_sequential      0
payment_type            0
payment_installments    0
payment_value           0
dtype: int64

Processing  order_items.csv
NaN values before replacement:
order_id               0
order_item_id          0
product_id             0
seller_id              0
shipping_limit_date    0
price                  0
freight_value          0
dtype: int64
```

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mysql.connector


db = mysql.connector.connect(host = "localhost",
                            username = "root",
                            password = "root",
                            database = "E_commerce")

cur = db.cursor()
```

1 List all unique cities where customers are located.

[4]:
```python
query = """ select distinct customer_city from customers """

cur.execute(query)

data = cur.fetchall()

df = pd.DataFrame(data)
df.head()
```

[4]:
```
                       0
0                 franca
1  sao bernardo do campo
```

```
2                  sao paulo
3            mogi das cruzes
4                   campinas
```

2 Count the number of orders placed in 2017

[8]:
```
query="""select count(order_id) from orders
where order_purchase_timestamp =2017"""
cur.execute(query)
data=cur.fetchall()
"Total orders placed in 2017 is ", data[0][0]
```

[8]:
```
('Total orders placed in 2017 is ', 180404)
```

3 Find the total sales per category

[11]:
```
query = """ select upper(products.product_category) category,
round(sum(payments.payment_value),2) sales
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category
"""

cur.execute(query)

data = cur.fetchall()
df
```

[11]:
```
                         Category        Sales
0                       PERFUMERY  18242591.76
1           FURNITURE DECORATION  51486350.10
2                       TELEPHONY  17527753.83
3                  BED TABLE BATH  61651932.16
4                      AUTOMOTIVE  30682595.92
..                            ...          ...
69                 CDS MUSIC DVDS     43179.48
70                     LA CUISINE    104887.08
71  FASHION CHILDREN'S CLOTHING       28284.12
72                       PC GAMER     78279.48
73         INSURANCE AND SERVICES     11682.36

[74 rows x 2 columns]
```

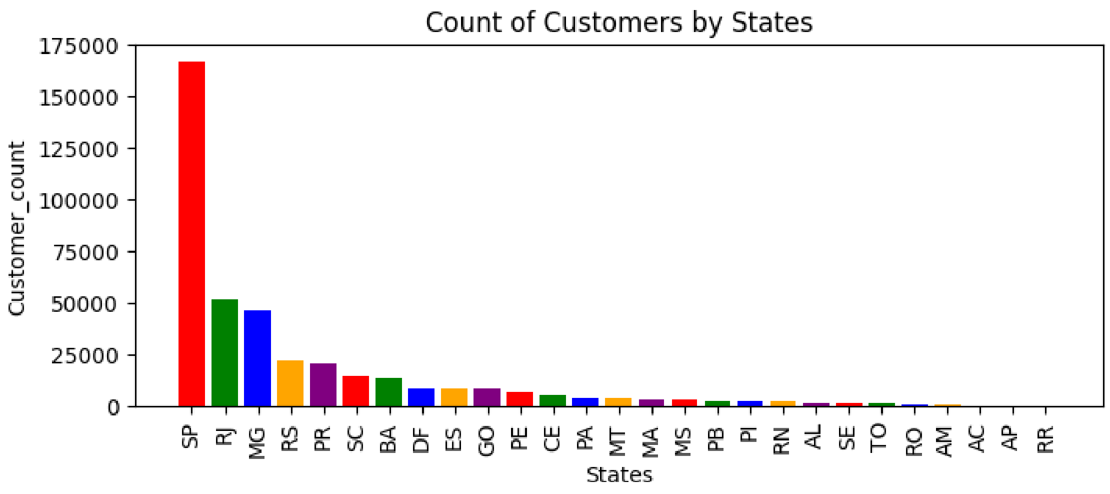
4 Calculate the percentage of orders that were paid in installments.

[14]:
```
query = """ select ((sum(case when payment_installments >= 1 then 1
else 0 end))/count(*))*100 from payments
"""

cur.execute(query)

data = cur.fetchall()

"the percentage of orders that were paid in installments is", data[0][0]
```

[14]:
```
('the percentage of orders that were paid in installments is',
 Decimal('99.9981'))
```

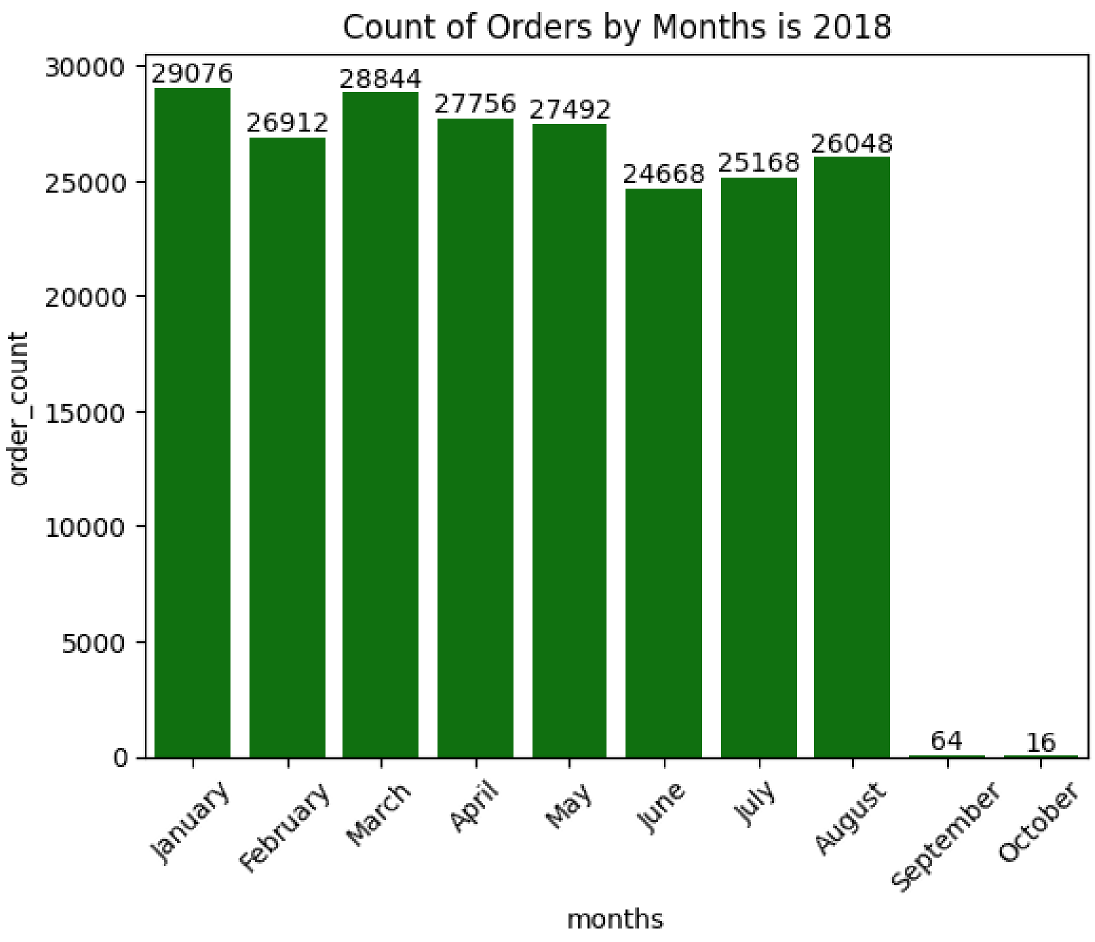
5 Count the number of customers from each state

[20]:
```
query = """ select customer_state ,count(customer_id)
from customers group by customer_state
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["state", "customer_count" ])
df = df.sort_values(by = "customer_count", ascending= False)

plt.figure(figsize = (8,3))
colors = ['red', 'green', 'blue', 'orange', 'purple']
plt.bar(df["state"], df["customer_count"], color=colors)
plt.xticks(rotation = 90)
plt.xlabel("States")
plt.ylabel("Customer_count")
plt.title("Count of Customers by States")
plt.show()
```



6 Calculate the number of orders per month in 2018.

[34]:
```
query = """ select monthname(order_purchase_timestamp) months, count(order_id)␣
  ↪order_count
from orders where year(order_purchase_timestamp) = 2018
group by months
"""

cur.execute(query)

data = cur.fetchall()
df = pd.DataFrame(data, columns = ["months", "order_count"])
o = ["January",␣
  ↪"February","March","April","May","June","July","August","September","October"]

ax = sns.barplot(x = df["months"],y =  df["order_count"], data = df, order = o,␣
  ↪color = "green")
plt.xticks(rotation = 45)
ax.bar_label(ax.containers[0])
plt.title("Count of Orders by Months is 2018")

plt.show()
```



7 Find the average number of products per order, grouped by customer city.

[36]:
```
query = """with count_per_order as
(select orders.order_id, orders.customer_id, count(order_items.order_id) as oc
from orders join order_items
on orders.order_id = order_items.order_id
group by orders.order_id, orders.customer_id)

select customers.customer_city, round(avg(count_per_order.oc),2) average_orders
from customers join count_per_order
on customers.customer_id = count_per_order.customer_id
group by customers.customer_city order by average_orders desc
"""

cur.execute(query)
```

```
data =  cur.fetchall()
df = pd.DataFrame(data,columns = ["customer city", "average products/order"])
df.head(10)
```

[36]:
```
          customer city average products/order
0        padre carvalho                  84.00
1           celso ramos                  78.00
2                 datas                  72.00
3         candido godoi                  72.00
4        matias olimpio                  60.00
5            cidelandia                  48.00
6            curralinho                  48.00
7               picarra                  48.00
8    morro de sao paulo                  48.00
9       teixeira soares                  48.00
```

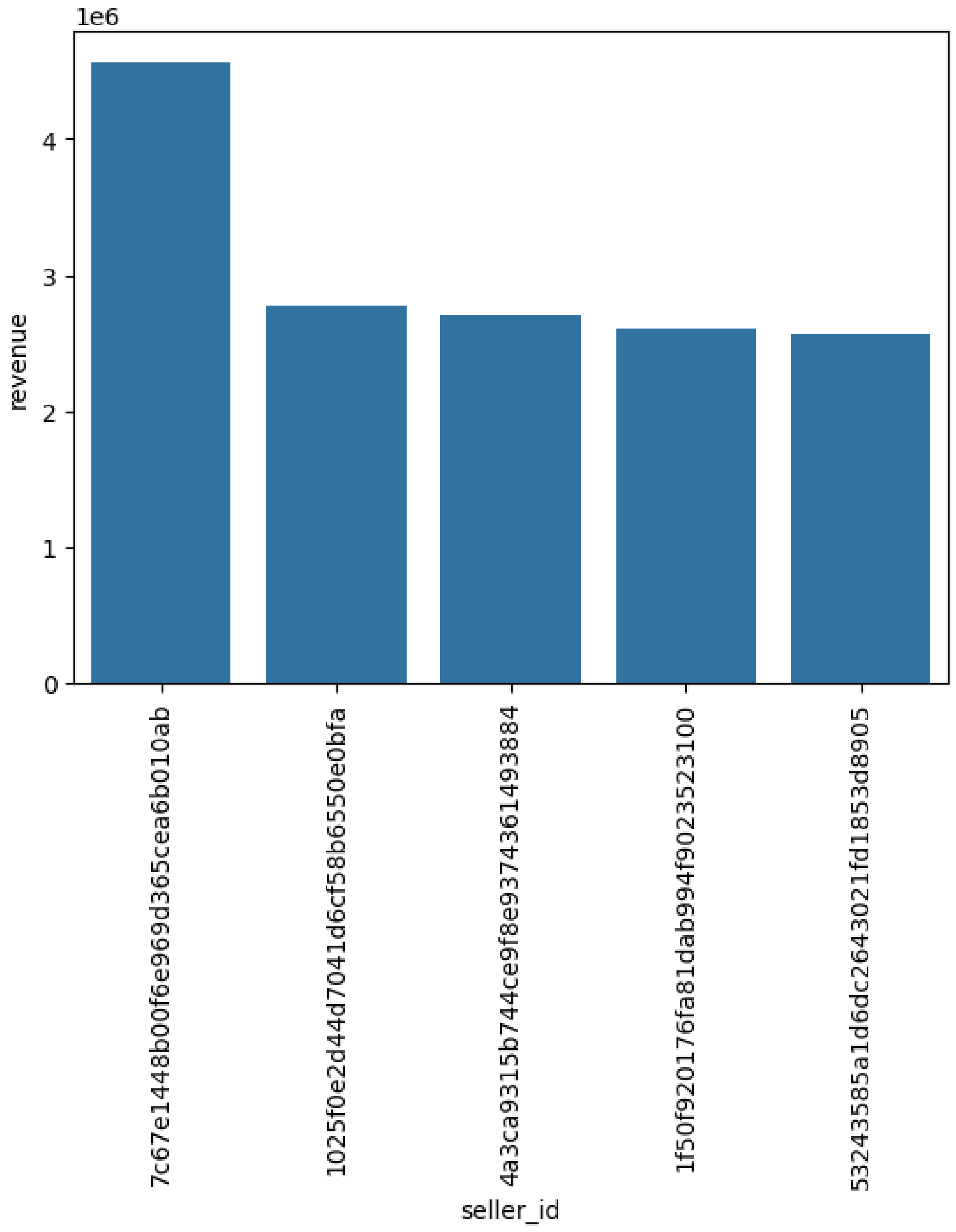
8 Calculate the percentage of total revenue contributed by each product category.

[37]:
```
query = """select upper(products.product_category) category,
round((sum(payments.payment_value)/(select sum(payment_value) from␣
 ↪payments))*100,2) sales_percentage
from products join order_items
on products.product_id = order_items.product_id
join payments
on payments.order_id = order_items.order_id
group by category order by sales_percentage desc"""


cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data,columns = ["Category", "percentage distribution"])
df.head()
```

[37]:
```
               Category  percentage distribution
0        BED TABLE BATH                   128.37
1         HEALTH BEAUTY                   124.23
2  COMPUTER ACCESSORIES                   118.83
3  FURNITURE DECORATION                   107.20
4       WATCHES PRESENT                   107.13
```

9 Calculate the total revenue generated by each seller, and rank them by revenue.

[40]:
```
query = """ select *, dense_rank() over(order by revenue desc) as rn from
(select order_items.seller_id, sum(payments.payment_value)
revenue from order_items join payments
on order_items.order_id = payments.order_id
group by order_items.seller_id) as a """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data, columns = ["seller_id", "revenue", "rank"])
df = df.head()
sns.barplot(x = "seller_id", y = "revenue", data = df)
plt.xticks(rotation = 90)
plt.show()
```


1e6
4
3
2
1
0
revenue
7c67e1448b00f6e969d365cea6b010ab
1025f0e2d44d7041d6cf58b6550e0bfa
4a3ca9315b744ce9f8e9374361493884
1f50f920176fa81dab994f9023523100
53243585a1d6dc2643021fd1853d8905
seller_id

10 Calculate the cumulative sales per month for each year.

[41]:
```
query = """select years, months , payment, sum(payment)
over(order by years, months)  cumulative_sales  from
(select year(orders.order_purchase_timestamp) as years,
month(orders.order_purchase_timestamp) as months,
round(sum(payments.payment_value),2)  as  payment  from  orders  join  payments
on  orders.order_id  =  payments.order_id
group by years, months order by years, months) as a
"""
cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(data)
df
```

[41]:
```
       0   1            2             3
0   2016   9      3026.88  3.026880e+03
1   2016  10    709085.76  7.121126e+05
2   2016  12       235.44  7.123481e+05
3   2017   1   1661856.48  2.374205e+06
4   2017   2   3502896.11  5.877101e+06
5   2017   3   5398363.19  1.127546e+07
6   2017   4   5013456.35  1.628892e+07
7   2017   5   7115025.84  2.340395e+07
8   2017   6   6135316.56  2.953926e+07
9   2017   7   7108595.03  3.664786e+07
10  2017   8   8092755.84  4.474061e+07
11  2017   9   8733149.40  5.347376e+07
12  2017  10   9356134.56  6.282990e+07
13  2017  11  14338593.60  7.716849e+07
14  2017  12  10540817.76  8.770931e+07
15  2018   1  13380050.15  1.010894e+08
16  2018   2  11909560.08  1.129989e+08
17  2018   3  13915825.44  1.269147e+08
18  2018   4  13929425.77  1.408442e+08
19  2018   5  13847785.81  1.546920e+08
20  2018   6  12286566.01  1.669785e+08
21  2018   7  12798488.99  1.797770e+08
22  2018   8  12269103.86  1.920461e+08
23  2018   9     53274.48  1.920994e+08
24  2018  10      7076.04  1.921065e+08
```