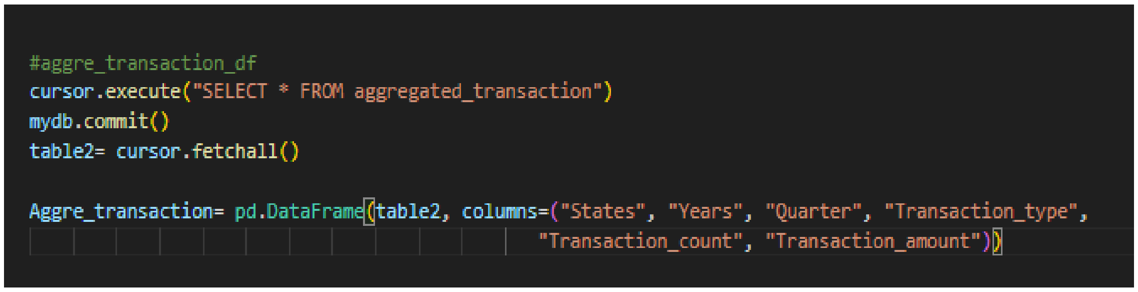


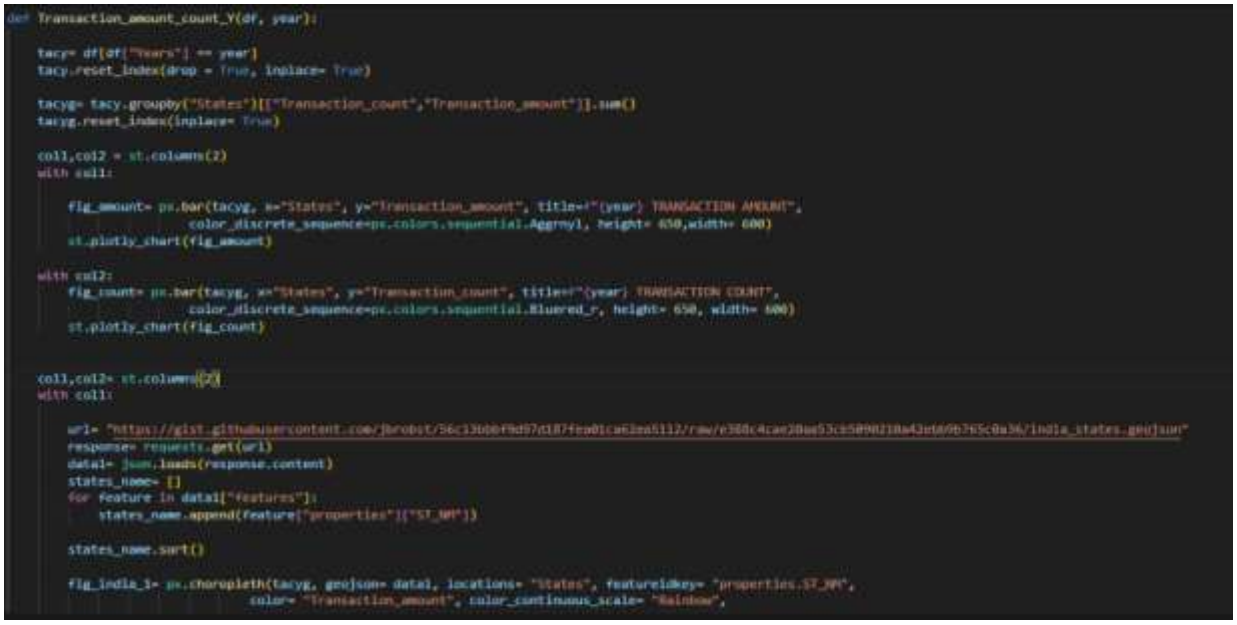
```
#aggre_transaction_df
cursor.execute("SELECT * FROM aggregated_transaction")
mydb.commit()
table2= cursor.fetchall()

Aggre_transaction= pd.DataFrame(table2, columns=("States", "Years", "Quarter", "Transaction_type",
                                                  "Transaction_count", "Transaction_amount"))
```

1. `cursor.execute("SELECT * FROM aggregated_transaction")`: Executes a SQL query to select all columns (`*`) from the table named "aggregated_transaction".

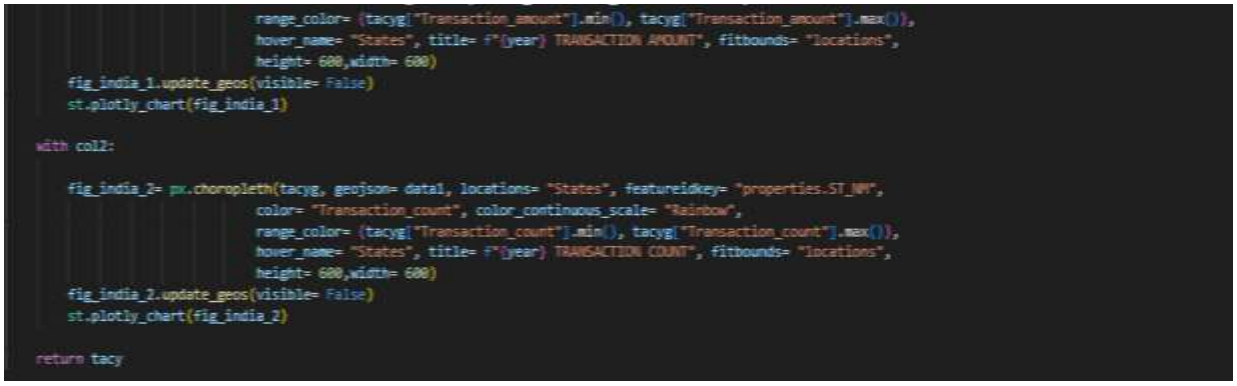
2. `mydb.commit()`: Commits the transaction. This ensures that any changes made by the SQL query are finalized in the database.

3. `table2 = cursor.fetchall()`: Fetches all the rows returned by the SQL query and stores them in the variable `table2`.

4. `Aggre_transaction = pd.DataFrame(table2, columns=("States", "Years", "Quarter", "Transaction_type", "Transaction_count", "Transaction_amount"))`: Creates a DataFrame named `Aggre_transaction` from the fetched data (`table2`). It specifies the column names as "States", "Years", "Quarter", "Transaction_type", "Transaction_count", and "Transaction_amount".



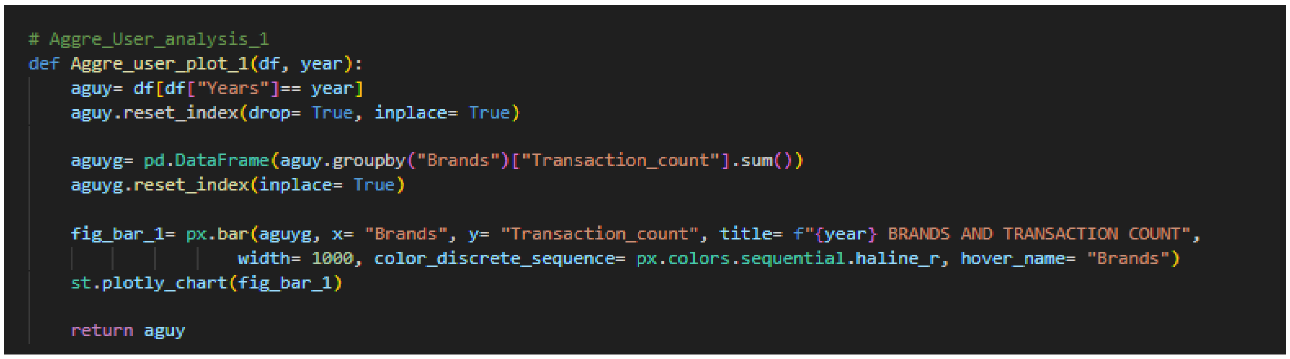
```
def Transaction_amount_count_Y(df, year):

    tacy= df[df["Years"] == year]
    tacy.reset_index(drop = True, inplace= True)

    tacyg= tacy.groupby("States")[["Transaction_count","Transaction_amount"]].sum()
    tacyg.reset_index(inplace= True)

    col1,col2 = st.columns(2)
    with col1:

        fig_amount= px.bar(tacyg, x="States", y="Transaction_amount", title=f"{year} TRANSACTION AMOUNT",
                           color_discrete_sequence=px.colors.sequential.Aggrnyl, height= 650,width= 600)
        st.plotly_chart(fig_amount)

    with col2:
        fig_count= px.bar(tacyg, x="States", y="Transaction_count", title=f"{year} TRANSACTION COUNT",
                          color_discrete_sequence=px.colors.sequential.Bluered_r, height= 650, width= 600)
        st.plotly_chart(fig_count)


    col1,col2= st.columns(2)
    with col1:

        url= "https://gist.githubusercontent.com/jbrobst/56c13bbbf9d97d187fea01ca62ea5112/raw/e388c4cae20aa53cb5090210a42ebb9b765c0a36/india_states.geojson"
        response= requests.get(url)
        data1= json.loads(response.content)
        states_name= []
        for feature in data1["features"]:
            states_name.append(feature["properties"]["ST_NM"])

        states_name.sort()

        fig_india_1= px.choropleth(tacyg, geojson= data1, locations= "States", featureidkey= "properties.ST_NM",
                                   color= "Transaction_amount", color_continuous_scale= "Rainbow",
```



```
                                   range_color= (tacyg["Transaction_amount"].min(), tacyg["Transaction_amount"].max()),
                                   hover_name= "States", title= f"{year} TRANSACTION AMOUNT", fitbounds= "locations",
                                   height= 600,width= 600)
        fig_india_1.update_geos(visible= False)
        st.plotly_chart(fig_india_1)

    with col2:

        fig_india_2= px.choropleth(tacyg, geojson= data1, locations= "States", featureidkey= "properties.ST_NM",
                                   color= "Transaction_count", color_continuous_scale= "Rainbow",
                                   range_color= (tacyg["Transaction_count"].min(), tacyg["Transaction_count"].max()),
                                   hover_name= "States", title= f"{year} TRANSACTION COUNT", fitbounds= "locations",
                                   height= 600,width= 600)
        fig_india_2.update_geos(visible= False)
        st.plotly_chart(fig_india_2)

    return tacy
```

1. **`Transaction_amount_count_Y(df, year)`**: This function takes a DataFrame **`df`** containing transaction data and a **`year`** as input. It filters the data for the specified year, groups the data by states, calculates the sum of transaction counts and amounts for each state, and then generates bar charts and choropleth maps for transaction amounts and counts respectively. Finally, it returns the filtered DataFrame for the specified year.

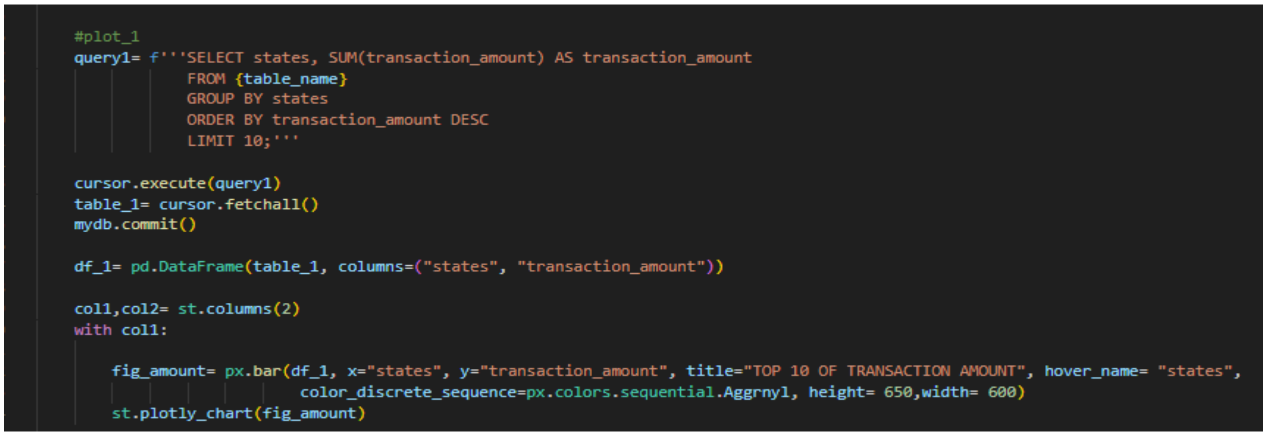
2. **`Transaction_amount_count_Y_Q(df, quarter)`**: Similar to the first function, this one takes a DataFrame **`df`** and a **`quarter`** as input. It filters the data for the specified quarter, groups the data by states, calculates the sum of transaction counts and amounts for each state, and then generates bar charts and choropleth maps for transaction amounts and counts respectively. Finally, it returns the filtered DataFrame for the specified quarter.



```python
# Aggre_User_analysis_1
def Aggre_user_plot_1(df, year):
    aguy= df[df["Years"]== year]
    aguy.reset_index(drop= True, inplace= True)

    aguyg= pd.DataFrame(aguy.groupby("Brands")["Transaction_count"].sum())
    aguyg.reset_index(inplace= True)

    fig_bar_1= px.bar(aguyg, x= "Brands", y= "Transaction_count", title= f"{year} BRANDS AND TRANSACTION COUNT",
                        width= 1000, color_discrete_sequence= px.colors.sequential.haline_r, hover_name= "Brands")
    st.plotly_chart(fig_bar_1)

    return aguy
```

- **Input Parameters**:
    - **`df`**: This parameter represents the DataFrame containing user transaction data.
    - **`year`**: This parameter specifies the year for which you want to analyze user transactions.
- **Functionality**:
    - It filters the DataFrame **`df`** to only include data for the specified **`year`**.
    - Groups the filtered data by the "Brands" column and calculates the sum of transaction counts for each brand.
    - Creates a bar plot using Plotly (**`px.bar`**) showing the transaction counts for each brand.
    - The title of the plot includes the specified **`year`**.
    - Finally, it returns the filtered DataFrame **`aguy`**.
- **Visualization**:
    - The function generates a bar plot where each bar represents a brand, and the height of the bar represents the total transaction count for that brand in the specified year.
    - The colors of the bars are chosen from the **`px.colors.sequential.haline_r`** color sequence.
    - The hover tooltip displays the name of the brand when hovering over a bar.
- **Output**:

- The function returns the filtered DataFrame `aguy`.



```python
#plot_1
query1= f'''SELECT states, SUM(transaction_amount) AS transaction_amount
            FROM {table_name}
            GROUP BY states
            ORDER BY transaction_amount DESC
            LIMIT 10;'''

cursor.execute(query1)
table_1= cursor.fetchall()
mydb.commit()

df_1= pd.DataFrame(table_1, columns=("states", "transaction_amount"))

col1,col2= st.columns(2)
with col1:

    fig_amount= px.bar(df_1, x="states", y="transaction_amount", title="TOP 10 OF TRANSACTION AMOUNT", hover_name= "states",
                       color_discrete_sequence=px.colors.sequential.Aggrnyl, height= 650,width= 600)
    st.plotly_chart(fig_amount)
```

1. **SQL Query**: You're constructing a SQL query using an f-string (`f''' ... '''`) where you're selecting the "states" column and calculating the sum of "transaction_amount" for each state from the table specified by `table_name`. The results are grouped by states, sorted in descending order based on transaction amount, and limited to the top 10 states.

2. **Executing Query**: You're executing the constructed SQL query using `cursor.execute(query1)`.

3. **Fetching Data**: After executing the query, you're fetching all the rows returned by the query using `cursor.fetchall()`.

4. **Creating DataFrame**: You're creating a pandas DataFrame (`df_1`) from the fetched data, specifying the column names as "states" and "transaction_amount".

5. **Visualization**: You're using Plotly (`px.bar`) to create a bar chart (`fig_amount`) to visualize the top 10 states with the highest transaction amounts. The chart shows the transaction amounts for each state, with the states on the x-axis and transaction amounts on the y-axis. The chart is displayed using `st.plotly_chart()`