```
!pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-9.5.0-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (7.5 kB)
Downloading mysql_connector_python-9.5.0-cp312-cp312-manylinux_2_28_x86_64.whl (34.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 34.1/34.1 MB 7.0 MB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-9.5.0
```

```
from git import Repo
import os
import pandas as pd
import numpy as np
import mysql.connector
```

```
repo_url = "https://github.com/PhonePe/pulse.git"
clone_path = r"Projects/Phonepe_pulse/Miscellaneous"

if not os.path.exists(clone_path):
    os.makedirs(clone_path)

repo_path = os.path.join(clone_path, os.path.basename(repo_url).removesuffix('.git').title())

Repo.clone_from(repo_url, repo_path)

directory = os.path.join(repo_path, 'data')
print(directory)
```

```
Projects/Phonepe_pulse/Miscellaneous/Pulse/data
```

```
# Function to rename messy state names in a proper format
# Function to rename messy state names in a proper format

def rename(directory):
    for root, dirs, files in os.walk(directory):
        if 'state' in dirs:
            state_dir = os.path.join(root, 'state')
            for state_folder in os.listdir(state_dir):
                # rename the state folder
                old_path = os.path.join(state_dir, state_folder)
                new_path = os.path.join(state_dir, state_folder.title().replace('-', ' ').replace('&', 'and'))
                os.rename(old_path, new_path)
    print("Renamed all sub-directories successfully")

# Function to extract all paths that has sub-directory in the name of 'state'

def extract_paths(directory):
    path_list = []
    for root, dirs, files in os.walk(directory):
        if os.path.basename(root) == 'state':
            path_list.append(root.replace('\\', '/'))
    return path_list
```

```
rename(directory)
```

```
Renamed all sub-directories successfully
```

```
extract_paths(directory)
```

```
['Projects/Phonepe_pulse/Miscellaneous/Pulse/data/top/user/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/top/transaction/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/top/insurance/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/aggregated/user/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/aggregated/transaction/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/aggregated/insurance/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/user/hover/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/hover/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/insurance/country/india/state',
 'Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/insurance/hover/country/india/state']
```

```
state_directories = extract_paths(directory)
print("State directories found:", state_directories)
state_path = state_directories[4] # Correct index for aggregated transaction data
```

```
state_list = os.listdir(state_path)
agg_trans_dict = {
                  'State': [], 'Year': [], 'Quarter': [], 'Transaction_type': [],
                  'Transaction_count': [], 'Transaction_amount': []
                  }

print(f"Processing state path: {state_path}")

for state in state_list:
    year_path = os.path.join(state_path, state)
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}")

    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}")


        for quarter in quarter_list:
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}")
            df = pd.read_json(json_path)
            print(f"        DataFrame shape after reading JSON: {df.shape}")


            try:
                if 'data' in df and 'transactionData' in df['data']:
                    print(f"          Found 'transactionData' in {quarter}")
                    for transaction_data in df['data']['transactionData']:

                        type = transaction_data['name']
                        count = transaction_data['paymentInstruments'][0]['count']
                        amount = transaction_data['paymentInstruments'][0]['amount']

                        # Appending to agg_trans_dict

                        agg_trans_dict['State'].append(state)
                        agg_trans_dict['Year'].append(year)
                        agg_trans_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                        agg_trans_dict['Transaction_type'].append(type)
                        agg_trans_dict['Transaction_count'].append(count)
                        agg_trans_dict['Transaction_amount'].append(amount)
                    print(f"          Appended data for {len(df['data']['transactionData'])} transactions")
                else:
                    print(f"          'transactionData' not found in {quarter}")

            except Exception as e:
                print(f"          Error processing {quarter}: {e}")
                pass

agg_trans_df = pd.DataFrame(agg_trans_dict)
print(f"\nFinal agg_trans_df shape: {agg_trans_df.shape}")
```

```
            Appended data for 5 transactions
        Processing quarter file: 3.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 3.json
            Appended data for 5 transactions
        Processing quarter file: 4.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 4.json
            Appended data for 5 transactions
        Processing quarter file: 1.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 1.json
            Appended data for 5 transactions
    Processing year: 2021
        Processing quarter file: 2.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 2.json
            Appended data for 5 transactions
        Processing quarter file: 3.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 3.json
            Appended data for 5 transactions
        Processing quarter file: 4.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 4.json
            Appended data for 5 transactions
        Processing quarter file: 1.json
          DataFrame shape after reading JSON: (3, 4)
            Found 'transactionData' in 1.json
            Appended data for 5 transactions

Final agg_trans_df shape: (5034, 6)
```

```
import json

state_path = state_directories[3] # Correct index for aggregated user data
state_list = os.listdir(state_path)
agg_user_dict = {
                 'State': [], 'Year': [], 'Quarter': [], 'Brand': [],
                 'Transaction_count': [], 'Percentage': []
                 }

print(f"Processing state path: {state_path}")

for state in state_list:
    year_path = os.path.join(state_path, state)
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}")

    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}")


        for quarter in quarter_list:
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}")
            df = pd.read_json(json_path)
            print(f"        DataFrame shape after reading JSON: {df.shape}")


            try:
                # Extracting data from 'usersByDevice'
                if 'data' in df and 'usersByDevice' in df['data'] and df['data']['usersByDevice'] is not None:
                    print(f"          Found 'usersByDevice' in {quarter}")
                    for user_data in df['data']['usersByDevice']:
                        brand = user_data['brand']
                        count = user_data['count']
                        percentage = user_data['percentage']

                        # Appending to agg_user_dict
                        agg_user_dict['State'].append(state)
                        agg_user_dict['Year'].append(year)
                        agg_user_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                        agg_user_dict['Brand'].append(brand)
                        agg_user_dict['Transaction_count'].append(count)
                        agg_user_dict['Percentage'].append(percentage)
                    print(f"          Appended data for {len(df['data']['usersByDevice'])} users")
```

```
                else:
                    print(f"          'usersByDevice' not found or is None in {quarter}")

            except Exception as e:
                print(f"          Error processing {quarter}: {e}")
                pass

agg_user_df = pd.DataFrame(agg_user_dict)
print(f"\nFinal agg_user_df shape: {agg_user_df.shape}")
```

```
          Appended data for 11 users
      Processing quarter file: 4.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 4.json
          Appended data for 11 users
      Processing quarter file: 1.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 1.json
          Appended data for 11 users
    Processing year: 2020
      Processing quarter file: 2.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 2.json
          Appended data for 11 users
      Processing quarter file: 3.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 3.json
          Appended data for 11 users
      Processing quarter file: 4.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 4.json
          Appended data for 11 users
      Processing quarter file: 1.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 1.json
          Appended data for 11 users
    Processing year: 2023
      Processing quarter file: 2.json
        DataFrame shape after reading JSON: (2, 4)
          'usersByDevice' not found or is None in 2.json
      Processing quarter file: 3.json
        DataFrame shape after reading JSON: (2, 4)
          'usersByDevice' not found or is None in 3.json
      Processing quarter file: 4.json
        DataFrame shape after reading JSON: (2, 4)
          'usersByDevice' not found or is None in 4.json
      Processing quarter file: 1.json
        DataFrame shape after reading JSON: (2, 4)
          'usersByDevice' not found or is None in 1.json
    Processing year: 2021
      Processing quarter file: 2.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 2.json
          Appended data for 11 users
      Processing quarter file: 3.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 3.json
          Appended data for 11 users
      Processing quarter file: 4.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 4.json
          Appended data for 11 users
      Processing quarter file: 1.json
        DataFrame shape after reading JSON: (2, 4)
          Found 'usersByDevice' in 1.json
          Appended data for 11 users

Final agg_user_df shape: (6732, 6)
```

```
import os
print(os.listdir(directory))
```

```
['top', 'aggregated', 'map']
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment

state_directories = extract_paths(directory) # Ensure state_directories is available
state_path = state_directories[7] # Use the correct path for map transaction data
state_list = os.listdir(state_path)
```

```
map_trans_dict = {
                    'State': [], 'Year': [], 'Quarter': [], 'District': [],
                    'Transaction_count': [], 'Transaction_amount': []
                    }

print(f"--- Processing map transaction data from: {state_path} ---") # Added print

for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print

    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        if not os.path.isdir(quarter_path): continue # Skip if not a directory
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}") # Added print


        for quarter in quarter_list:
            if not quarter.endswith('.json'): continue # Skip non-json files
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}") # Added print
            try:
                with open(json_path, 'r') as f:
                    data = json.load(f) # Load JSON directly as dictionary
                # print(f"        Loaded JSON data keys: {data.keys()}") # Added print


                # Check if 'data' and 'districts' keys exist and are not None
                if 'data' in data and isinstance(data['data'], dict) and 'districts' in data['data'] and isinstance(data['data']['d
                    # print(f"          Found 'data' and 'districts' in {quarter}") # Added print
                    if not data['data']['districts']:
                        print(f"          'districts' list is empty in {quarter}") # Added print

                    for district_data in data['data']['districts']:
                        # print(f"            Processing district data: {district_data}") # Added print
                        # Check if necessary keys exist in district_data
                        if 'entityName' in district_data and 'metric' in district_data and isinstance(district_data['metric'], dict
                            name = district_data['entityName']
                            count = district_data['metric']['count']
                            amount = district_data['metric']['amount']

                            # Appending to map_trans_dict
                            map_trans_dict['State'].append(state)
                            map_trans_dict['Year'].append(year)
                            map_trans_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                            map_trans_dict['District'].append(name.title().replace(' And', ' and').replace('andaman', 'Andaman'))
                            map_trans_dict['Transaction_count'].append(count)
                            map_trans_dict['Transaction_amount'].append(amount)
                            # print(f"              Appended data for district: {name}") # Added print
                        else:
                            print(f"              Warning: Missing keys in district_data for {json_path}: {district_data.keys()}")
                else:
                    print(f"          Warning: 'data' or 'districts' not found or invalid in {json_path}. Keys in loaded data: {dat

            except Exception as e: # Catch specific exception and print it
                print(f"      Error processing {json_path}: {e}")


map_trans_df = pd.DataFrame(map_trans_dict)
print(f"\nFinal map_trans_df shape: {map_trans_df.shape}") # Added print
```

```
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 4.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 1.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
        Processing year: 2019
          Processing quarter file: 2.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 3.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 4.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 1.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
        Processing year: 2020
          Processing quarter file: 2.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 3.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 4.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 1.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
        Processing year: 2023
          Processing quarter file: 2.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 3.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 4.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 1.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
        Processing year: 2021
          Processing quarter file: 2.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 3.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 4.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho
          Processing quarter file: 1.json
            Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/transaction/ho

    Final map trans df shape: (0, 6)
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment


state_directories = extract_paths(directory) # Ensure state_directories is available
state_path = state_directories[6] # Use the correct path for map user data
state_list = os.listdir(state_path)
map_user_dict = {
                 'State': [], 'Year': [], 'Quarter': [], 'District': [],
                 'Registered_users': [], 'App_opens': []
                 }
print(f"--- Processing map user data from: {state_path} ---") # Added print


for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print


    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        if not os.path.isdir(quarter_path): continue # Skip if not a directory
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}") # Added print


        for quarter in quarter_list:
            if not quarter.endswith('.json'): continue # Skip non-json files
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}") # Added print
            try:
                with open(json_path, 'r') as f:
                    data = json.load(f) # Load JSON directly as dictionary
                # print(f"        Loaded JSON data keys: {data.keys()}") # Added print
```

```
                # Check if 'data' and 'districts' keys exist and are not None
                if 'data' in data and isinstance(data['data'], dict) and 'districts' in data['data'] and isinstance(data['data']['d
                    # print(f"          Found 'data' and 'districts' in {quarter}") # Added print
                    if not data['data']['districts']:
                        print(f"          'districts' list is empty in {quarter}") # Added print)

                    for district_data in data['data']['districts']:
                        # print(f"            Processing district data: {district_data}") # Added print
                         # Check if necessary keys exist in district_data
                        if 'name' in district_data and 'registeredUsers' in district_data and 'appOpens' in district_data:
                            name = district_data['name']
                            registered_users = district_data['registeredUsers']
                            app_opens = district_data['appOpens']

                            # Appending to map_user_dict
                            map_user_dict['State'].append(state)
                            map_user_dict['Year'].append(year)
                            map_user_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                            map_user_dict['District'].append(name.title().replace(' And', ' and').replace('andaman', 'Andaman'))
                            map_user_dict['Registered_users'].append(registered_users)
                            map_user_dict['App_opens'].append(app_opens)
                            # print(f"              Appended data for district: {name}") # Added print
                        else:
                            print(f"              Warning: Missing keys in district_data for {json_path}: {district_data.keys()}") 
                    else:
                        print(f"          Warning: 'data' or 'districts' not found or invalid in {json_path}. Keys in loaded data: {dat
                except Exception as e: # Catch specific exception and print it
                    print(f"      Error processing {json_path}: {e}")


map_user_df = pd.DataFrame(map_user_dict)
print(f"\nFinal map_user_df shape: {map_user_df.shape}") # Added print
```

```
    Processing year: 2021
      Processing quarter file: 2.json
          Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/user/hover/cou
      Processing quarter file: 3.json
          Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/user/hover/cou
      Processing quarter file: 4.json
          Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/user/hover/cou
      Processing quarter file: 1.json
          Warning: 'data' or 'districts' not found or invalid in Projects/Phonepe_pulse/Miscellaneous/Pulse/data/map/user/hover/cou

Final map_user_df shape: (0, 6)
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment

state_directories = extract_paths(directory) # Ensure state_directories is available
state_path = state_directories[1] # Use the correct path for top transaction district data
state_list = os.listdir(state_path)
top_trans_dist_dict = {
                        'State': [], 'Year': [], 'Quarter': [], 'District': [],
                        'Transaction_count': [], 'Transaction_amount': []
                        }
print(f"--- Processing top transaction district data from: {state_path} ---") # Added print


for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print

    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        if not os.path.isdir(quarter_path): continue # Skip if not a directory
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}") # Added print

        for quarter in quarter_list:
            if not quarter.endswith('.json'): continue # Skip non-json files
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}") # Added print
            try:
                with open(json_path, 'r') as f:
                    data = json.load(f) # Load JSON directly as dictionary
                # print(f"        Loaded JSON data keys: {data.keys()}") # Added print


                # Check if 'data' and 'districts' keys exist and are not None
                if 'data' in data and isinstance(data['data'], dict) and 'districts' in data['data'] and isinstance(data['data']['d
                    # print(f"          Found 'data' and 'districts' in {quarter}") # Added print
                    if not data['data']['districts']:
                        print(f"          'districts' list is empty in {quarter}") # Added print

                    for district_data in data['data']['districts']:
                        # print(f"            Processing district data: {district_data}") # Added print
                        # Check if necessary keys exist in district_data
                        if 'entityName' in district_data and 'metric' in district_data and isinstance(district_data['metric'], dict
                            name = district_data['entityName']
                            count = district_data['metric']['count']
                            amount = district_data['metric']['amount']

                            # Appending to top_trans_dist_dict
                            top_trans_dist_dict['State'].append(state)
                            top_trans_dist_dict['Year'].append(year)
                            top_trans_dist_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                            top_trans_dist_dict['District'].append(name.title().replace(' And', ' and').replace('andaman', 'Andaman
                            top_trans_dist_dict['Transaction_count'].append(count)
                            top_trans_dist_dict['Transaction_amount'].append(amount)
                            # print(f"              Appended data for district: {name}") # Added print
                        else:
                            print(f"              Warning: Missing keys in district_data for {json_path}: {district_data.keys()}") 
                else:
                    print(f"          Warning: 'data' or 'districts' not found or invalid in {json_path}. Keys in loaded data: {dat
            except Exception as e: # Catch specific exception and print it
                print(f"      Error processing {json_path}: {e}")
```

```
top_trans_dist_df = pd.DataFrame(top_trans_dist_dict)
print(f"\nFinal top_trans_dist_df shape: {top_trans_dist_df.shape}") # Added print
```

```
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
  Processing state: Andaman and Nicobar Islands
    Processing year: 2024
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2018
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2022
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2019
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json

Final top_trans_dist_df shape: (8296, 6)
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment

state_directories = extract_paths(directory) # Ensure state_directories is available
state_path = state_directories[1] # Use the correct path for top transaction pincode data
state_list = os.listdir(state_path)
top_trans_pin_dict = {
                        'State': [], 'Year': [], 'Quarter': [], 'Pincode': [],
                        'Transaction_count': [], 'Transaction_amount': []
                        }

print(f"--- Processing top transaction pincode data from: {state_path} ---") # Added print

for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print

    for year in year_list:
```

```
            quarter_path = os.path.join(year_path, year)
            if not os.path.isdir(quarter_path): continue # Skip if not a directory
            quarter_list = os.listdir(quarter_path)
            print(f"    Processing year: {year}") # Added print


            for quarter in quarter_list:
                if not quarter.endswith('.json'): continue # Skip non-json files
                json_path = os.path.join(quarter_path, quarter)
                print(f"      Processing quarter file: {quarter}") # Added print
                try:
                    with open(json_path, 'r') as f:
                        data = json.load(f) # Load JSON directly as dictionary
                    # print(f"        Loaded JSON data keys: {data.keys()}") # Added print

                    # Check if 'data' and 'pincodes' keys exist and are not None
                    if 'data' in data and isinstance(data['data'], dict) and 'pincodes' in data['data'] and isinstance(data['data']['pi
                        # print(f"          Found 'data' and 'pincodes' in {quarter}") # Added print
                        if not data['data']['pincodes']:
                             print(f"          'pincodes' list is empty in {quarter}") # Added print


                        for regional_data in data['data']['pincodes']:
                            # print(f"            Processing regional data: {regional_data}") # Added print
                            # Check if necessary keys exist in regional_data
                            if 'entityName' in regional_data and 'metric' in regional_data and isinstance(regional_data['metric'], dict
                                name = regional_data['entityName']
                                count = regional_data['metric']['count']
                                amount = regional_data['metric']['amount']

                                # Appending to top_trans_pin_dict
                                top_trans_pin_dict['State'].append(state)
                                top_trans_pin_dict['Year'].append(year)
                                top_trans_pin_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                                top_trans_pin_dict['Pincode'].append(name)
                                top_trans_pin_dict['Transaction_count'].append(count)
                                top_trans_pin_dict['Transaction_amount'].append(amount)
                                # print(f"              Appended data for pincode: {name}") # Added print
                            else:
                                 print(f"              Warning: Missing keys in regional_data for {json_path}: {regional_data.keys()}")
                    else:
                        print(f"          Warning: 'data' or 'pincodes' not found or invalid in {json_path}. Keys in loaded data: {data

                except Exception as e: # Catch specific exception and print it
                    print(f"      Error processing {json_path}: {e}")


top_trans_pin_df = pd.DataFrame(top_trans_pin_dict)
print(f"\nFinal top_trans_pin_df shape: {top_trans_pin_df.shape}") # Added print
```

```
    Processing year: 2022
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2019
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json

Final top_trans_pin_df shape: (9999, 6)
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment


state_directories = extract_paths(directory) # Ensure state_directories is available
state_path = state_directories[0] # Use the correct path for top user district data
state_list = os.listdir(state_path)
top_user_dist_dict = {
                        'State': [], 'Year': [], 'Quarter': [],
                        'District': [], 'Registered_users': []
                        }
print(f"--- Processing top user district data from: {state_path} ---") # Added print

for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print


    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        if not os.path.isdir(quarter_path): continue # Skip if not a directory
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}") # Added print


        for quarter in quarter_list:
            if not quarter.endswith('.json'): continue # Skip non-json files
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}") # Added print
            try:
                with open(json_path, 'r') as f:
                    data = json.load(f) # Load JSON directly as dictionary
                # print(f"        Loaded JSON data keys: {data.keys()}") # Added print


                # Check if 'data' and 'districts' keys exist and are not None
                if 'data' in data and isinstance(data['data'], dict) and 'districts' in data['data'] and isinstance(data['data']['d
                    # print(f"          Found 'data' and 'districts' in {quarter}") # Added print
                    if not data['data']['districts']:
                        print(f"          'districts' list is empty in {quarter}") # Added print)

                    for district_data in data['data']['districts']:
                        # print(f"            Processing district data: {district_data}") # Added print
                         # Check if necessary keys exist in district_data
                        if 'name' in district_data and 'registeredUsers' in district_data:
                            name = district_data['name']
                            registered_users = district_data['registeredUsers']
```

```
                            # Appending to top_user_dist_dict
                            top_user_dist_dict['State'].append(state)
                            top_user_dist_dict['Year'].append(year)
                            top_user_dist_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                            top_user_dist_dict['District'].append(name.title().replace(' And', ' and').replace('andaman', 'Andaman'
                            top_user_dist_dict['Registered_users'].append(registered_users)
                            # print(f"              Appended data for district: {name}") # Added print
                        else:
                            print(f"              Warning: Missing keys in district_data for {json_path}: {district_data.keys()}") 
                    else:
                        print(f"          Warning: 'data' or 'districts' not found or invalid in {json_path}. Keys in loaded data: {dat
                except Exception as e: # Catch specific exception and print it
                    print(f"      Error processing {json_path}: {e}")


top_user_dist_df = pd.DataFrame(top_user_dist_dict)
print(f"\nFinal top_user_dist_df shape: {top_user_dist_df.shape}") # Added print
```

```
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
  Processing state: Punjab
    Processing year: 2024
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2018
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2022
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2019
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
  Processing state: West Bengal
    Processing year: 2024
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
```

```
import json
import pandas as pd # Ensure pandas is imported if not already in the execution environment


state_directories = extract_paths(directory) # Ensure state_directories is available
```

```
state_path = state_directories[0] # Use the correct path for top user pincode data
state_list = os.listdir(state_path)
top_user_pin_dict = {
                        'State': [], 'Year': [], 'Quarter': [],
                        'Pincode': [], 'Registered_users': []
                        }

print(f"--- Processing top user pincode data from: {state_path} ---") # Added print


for state in state_list:
    year_path = os.path.join(state_path, state)
    if not os.path.isdir(year_path): continue # Skip if not a directory
    year_list = os.listdir(year_path)
    print(f"  Processing state: {state}") # Added print


    for year in year_list:
        quarter_path = os.path.join(year_path, year)
        if not os.path.isdir(quarter_path): continue # Skip if not a directory
        quarter_list = os.listdir(quarter_path)
        print(f"    Processing year: {year}") # Added print


        for quarter in quarter_list:
            if not quarter.endswith('.json'): continue # Skip non-json files
            json_path = os.path.join(quarter_path, quarter)
            print(f"      Processing quarter file: {quarter}") # Added print
            try:
                with open(json_path, 'r') as f:
                    data = json.load(f) # Load JSON directly as dictionary
                # print(f"        Loaded JSON data keys: {data.keys()}") # Added print


                # Check if 'data' and 'pincodes' keys exist and are not None
                if 'data' in data and isinstance(data['data'], dict) and 'pincodes' in data['data'] and isinstance(data['data']['pinc
                    # print(f"          Found 'data' and 'pincodes' in {quarter}") # Added print
                    if not data['data']['pincodes']:
                        print(f"          'pincodes' list is empty in {quarter}") # Added print)


                    for regional_data in data['data']['pincodes']:
                        # print(f"            Processing regional data: {regional_data}") # Added print
                        # Check if necessary keys exist in regional_data
                        if 'name' in regional_data and 'registeredUsers' in regional_data:
                            name = regional_data['name']
                            registered_users = regional_data['registeredUsers']

                            # Appending to top_user_pin_dict
                            top_user_pin_dict['State'].append(state)
                            top_user_pin_dict['Year'].append(year)
                            top_user_pin_dict['Quarter'].append(int(quarter.removesuffix('.json')))
                            top_user_pin_dict['Pincode'].append(name)
                            top_user_pin_dict['Registered_users'].append(registered_users)
                            # print(f"              Appended data for pincode: {name}") # Added print
                        else:
                             print(f"              Warning: Missing keys in regional_data for {json_path}: {regional_data.keys()}") #
                else:
                    print(f"          Warning: 'data' or 'pincodes' not found or invalid in {json_path}. Keys in loaded data: {data.k

            except Exception as e: # Catch specific exception and print it
                print(f"      Error processing {json_path}: {e}")


top_user_pin_df = pd.DataFrame(top_user_pin_dict)
print(f"\nFinal top_user_pin_df shape: {top_user_pin_df.shape}") # Added print
```

```
    Processing quarter file: 2.json
```

```
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
  Processing state: Andaman and Nicobar Islands
    Processing year: 2024
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2018
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2022
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2019
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2020
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2023
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json
    Processing year: 2021
      Processing quarter file: 2.json
      Processing quarter file: 3.json
      Processing quarter file: 4.json
      Processing quarter file: 1.json

Final top_user_pin_df shape: (10000, 5)
```

```
df_list = [df for df in globals() if isinstance(globals()[df], pd.core.frame.DataFrame) and df.endswith('_df')]

df_list
```

```
['agg_trans_df',
 'agg_user_df',
 'map_trans_df',
 'map_user_df',
 'top_trans_dist_df',
 'top_trans_pin_df',
 'top_user_dist_df',
 'top_user_pin_df']
```

```
def add_suffix_to_districts(df):
    if 'District' in df.columns and 'State' in df.columns:
        delhi_df = df[df['State'] == 'Delhi']

        districts_to_suffix = [d for d in delhi_df['District'].unique() if d != 'Shahdara']

        df.loc[(df['State'] == 'Delhi') & (df['District'].isin(districts_to_suffix)), 'District'] = df.loc[(df['State'] == 'Delhi')

    return df

for df_name in df_list:
    df = globals()[df_name]
    add_suffix_to_districts(df)
```

```
def add_region_column(df):
    state_groups = {
        'Northern Region': ['Jammu and Kashmir', 'Himachal Pradesh', 'Punjab', 'Chandigarh', 'Uttarakhand', 'Ladakh', 'Delhi', 'Har
        'Central Region': ['Uttar Pradesh', 'Madhya Pradesh', 'Chhattisgarh'],
```

```
        'Western Region': ['Rajasthan', 'Gujarat', 'Dadra and Nagar Haveli and Daman and Diu', 'Maharashtra'],
        'Eastern Region': ['Bihar', 'Jharkhand', 'Odisha', 'West Bengal', 'Sikkim'],
        'Southern Region': ['Andhra Pradesh', 'Telangana', 'Karnataka', 'Kerala', 'Tamil Nadu', 'Puducherry', 'Goa', 'Lakshadweep',
        'North-Eastern Region': ['Assam', 'Meghalaya', 'Manipur', 'Nagaland', 'Tripura', 'Arunachal Pradesh', 'Mizoram']
    }

    df['Region'] = df['State'].map({state: region for region, states in state_groups.items() for state in states})
    return df
```

```
for df_name in df_list:
    df = globals()[df_name]
    add_region_column(df)
```

```
for df_name in df_list:
    df = globals()[df_name]
    print(f"{df_name}:")
    print(f"Null count: \n{df.isnull().sum()}")
    print(f"Duplicated rows count: \n{df.duplicated().sum()}")
    print(df.shape)
    print("\n", 25 * "_", "\n")
```

```
Quarter               0
District              0
Transaction_count     0
Transaction_amount    0
Region                0
dtype: int64
Duplicated rows count:
0
(8296, 7)

 _________________________

top_trans_pin_df:
Null count:
State                 0
Year                  0
Quarter               0
Pincode               2
Transaction_count     0
Transaction_amount    0
Region                0
dtype: int64
Duplicated rows count:
0
(9999, 7)

 _________________________

top_user_dist_df:
Null count:
State               0
Year                0
Quarter             0
District            0
Registered_users    0
Region              0
dtype: int64
Duplicated rows count:
0
(8296, 6)

 _________________________

top_user_pin_df:
Null count:
State               0
Year                0
Quarter             0
Pincode             0
Registered_users    0
Region              0
dtype: int64
Duplicated rows count:
0
(10000, 6)

 _________________________
```

```
print('DATAFRAME INFO:\n')

for df_name in df_list:
    df = globals()[df_name]
    print(df_name + ':\n')
    df.info()
    print("\n", 45 * "_", "\n")
```

```
 _____

top_trans_pin_df:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   State               9999 non-null   object
 1   Year                9999 non-null   object
 2   Quarter             9999 non-null   int64
 3   Pincode             9997 non-null   object
 4   Transaction_count   9999 non-null   int64
 5   Transaction_amount  9999 non-null   float64
 6   Region              9999 non-null   object
dtypes: float64(1), int64(2), object(4)
memory usage: 546.9+ KB

 _____

top_user_dist_df:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8296 entries, 0 to 8295
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   State             8296 non-null   object
 1   Year              8296 non-null   object
 2   Quarter           8296 non-null   int64
 3   District          8296 non-null   object
 4   Registered_users  8296 non-null   int64
 5   Region            8296 non-null   object
dtypes: int64(2), object(4)
memory usage: 389.0+ KB

 _____

top_user_pin_df:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   State             10000 non-null  object
 1   Year              10000 non-null  object
 2   Quarter           10000 non-null  int64
 3   Pincode           10000 non-null  object
 4   Registered_users  10000 non-null  int64
 5   Region            10000 non-null  object
dtypes: int64(2), object(4)
memory usage: 468.9+ KB

 _____
```

```
top_trans_pin_df.dropna(axis = 'index', inplace = True)
top_trans_pin_df.isnull().sum()
```

	0
State	0
Year	0
Quarter	0
Pincode	0
Transaction_count	0
Transaction_amount	0
Region	0

dtype: int64

```
for df_name in df_list:
    df = globals()[df_name]
    if 'Year' in df.columns: # Check if 'Year' column exists
        df['Year'] = df['Year'].astype('int')
```

```
# Everything seems to be alright as far as dtypes and nullvalues are concerned so checking for outliers
# Function to check for outliers

def count_outliers(df):
    outliers = {}
    for col in df.select_dtypes(include=[np.number]).columns:
        if col in ['Transaction_count', 'Transaction_amount']:
            q1 = df[col].quantile(0.25)
            q3 = df[col].quantile(0.75)
            iqr = q3 - q1
            upper_bound = q3 + (1.5 * iqr)
            lower_bound = q1 - (1.5 * iqr)
            outliers[col] = len(df[(df[col] > upper_bound) | (df[col] < lower_bound)])
        else:
            continue
    return outliers
```

```
print('OUTLIER COUNT ACROSS DATAFRAMES:\n')

for df_name in df_list:
    df = globals()[df_name]
    outliers = count_outliers(df)
    if len(outliers) == 0:
        pass
    else:
        print(df_name, ":\n\n", outliers, "\n")
        print("\n", 55 * "_", "\n")
```

```
OUTLIER COUNT ACROSS DATAFRAMES:

agg_trans_df :

 {'Transaction_count': 936, 'Transaction_amount': 955}


 _____

agg_user_df :

 {'Transaction_count': 893}


 _____

map_trans_df :

 {'Transaction_count': 0, 'Transaction_amount': 0}


 _____

top_trans_dist_df :

 {'Transaction_count': 1144, 'Transaction_amount': 1027}
```

```
 ____________________________________________________

top_trans_pin_df :

 {'Transaction_count': 1411, 'Transaction_amount': 1366}


 ____________________________________________________
```

```
# Function to check for unique value counts and print if count less than 10;

def unique_value_count(df, exclude_cols=[]):
    for col in df.columns:
        if col in exclude_cols:
            continue
        unique_vals = df[col].nunique()
        print(f"{col}: {unique_vals} unique values")
        if unique_vals < 10:
            print(df[col].unique())
```

```
print('UNIQUE VALUE COUNT ACROSS DATAFRAMES; \n')

for df_name in df_list:
    df = globals()[df_name]
    print(df_name, ":\n")
    unique_value_count(df, exclude_cols = ['State', 'Year', 'Quarter', 'Percentage'])
    print("\n", 55 * "_", "\n")
```

```
 ____________________________________________________

map_user_df :

District: 0 unique values
[]
Registered_users: 0 unique values
[]
App_opens: 0 unique values
[]
Region: 0 unique values
[]

 ____________________________________________________

top_trans_dist_df :
```

```
Region: 6 unique values
['North-Eastern Region' 'Northern Region' 'Southern Region'
 'Eastern Region' 'Western Region' 'Central Region']


 ____________________________________________________
```

```
def save_dfs_as_csv(df_list):
    subfolder = 'Miscellaneous'
    if not os.path.exists(subfolder):
        os.makedirs(subfolder)

    for df_name in df_list:
        df = globals()[df_name]
        file_path = os.path.join(subfolder, df_name.replace('_df', '') + '.csv')
        df.to_csv(file_path, index=False)

# Calling function to execute

save_dfs_as_csv(df_list)
```

```
import sqlite3

# Connect to an SQLite database (creates the database file if it doesn't exist)
conn = sqlite3.connect('phonepe_pulse_data.db')

# Create a cursor object to execute SQL commands
cursor = conn.cursor()

print("Connected to SQLite database 'phonepe_pulse_data.db'")
```

```
Connected to SQLite database 'phonepe_pulse_data.db'
```

```
cursor.execute('''CREATE TABLE IF NOT EXISTS agg_trans (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    Transaction_type TEXT,
                    Transaction_count INTEGER,
                    Transaction_amount REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, Transaction_type, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS agg_user (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    Brand TEXT,
                    Transaction_count INTEGER,
                    Percentage REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, Brand, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS map_trans (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    District TEXT,
                    Transaction_count INTEGER,
                    Transaction_amount REAL,
                    Latitude REAL,
                    Longitude REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, District, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS map_user (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    District TEXT,
                    Registered_users INTEGER,
                    App_opens INTEGER,
                    Latitude REAL,
```

```
                    Longitude REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, District, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS top_trans_dist (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    District TEXT,
                    Transaction_count INTEGER,
                    Transaction_amount REAL,
                    Latitude REAL,
                    Longitude REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, District, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS top_trans_pin (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    Pincode TEXT,
                    Transaction_count INTEGER,
                    Transaction_amount REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, Pincode, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS top_user_dist (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    District TEXT,
                    Registered_users INTEGER,
                    Latitude REAL,
                    Longitude REAL,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, District, Region)
                )''')

cursor.execute('''CREATE TABLE IF NOT EXISTS top_user_pin (
                    State TEXT,
                    Year INTEGER,
                    Quarter INTEGER,
                    Pincode TEXT,
                    Registered_users INTEGER,
                    Region TEXT,
                    PRIMARY KEY (State, Year, Quarter, Pincode, Region)
                )''')
```

```
<sqlite3.Cursor at 0x7bfe58df7440>
```

```
def push_data_into_sqlite(conn, cursor, dfs):
    for df_name, df in dfs.items():
        table_name = df_name.replace('_df', '') # Assuming DataFrame names match table names
        # Create a list of column names from the DataFrame
        columns = df.columns.tolist()
        # Generate placeholders for the SQL query
        placeholders = ', '.join(['?'] * len(columns))
        # Construct the INSERT query
        query = f"INSERT OR IGNORE INTO {table_name} ({', '.join(columns)}) VALUES ({placeholders})"

        # Prepare data for insertion
        data_to_insert = [tuple(row) for row in df.values]

        # Execute the INSERT query for all rows
        cursor.executemany(query, data_to_insert)

        conn.commit()
    print("Data successfully pushed into SQLite tables")
```

```
# Mapping my_sql tables to pandas dataframes that we have created earlier

dfs = {
    'agg_trans': agg_trans_df,
```

```
    'agg_user': agg_user_df,
    'map_trans': map_trans_df,
    'map_user': map_user_df,
    'top_trans_dist': top_trans_dist_df,
    'top_trans_pin': top_trans_pin_df,
    'top_user_dist': top_user_dist_df,
    'top_user_pin': top_user_pin_df
}

# The table_columns dictionary is no longer needed by the push_data_into_sqlite function.
```

```
push_data_into_sqlite(conn, cursor, dfs)
```

```
Data successfully pushed into SQLite tables
```

```
# Get list of tables in database

cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
tables = cursor.fetchall()

# Loop through tables and get count of rows and columns in SQLite

for table in tables:
    table_name = table[0]
    cursor.execute(f"SELECT COUNT(*) FROM {table_name}")
    row_count = cursor.fetchone()[0]
    cursor.execute(f"PRAGMA table_info({table_name});")
    column_count = len(cursor.fetchall())

    # Check if shape of DataFrame matches count of rows and columns in table
    # Need to handle the case where the DataFrame might not exist or is not in the 'dfs' dictionary

    if table_name in dfs:
        df = dfs[table_name]
        # Note: The comparison of DataFrame shape and table shape might not be a perfect check
        # if there were issues during insertion (e.g., ignored rows due to primary key conflicts)
        # or if the DataFrame was modified after creation. A more thorough check would involve
        # comparing row counts after insertion and potentially checking data integrity.

        # Let's just print the counts for verification for now, as direct shape comparison might be misleading.
        print(f"Table '{table_name}': {row_count} rows, {column_count} columns. Corresponding DataFrame shape: {df.shape}")
    else:
        print(f"Table '{table_name}': {row_count} rows, {column_count} columns. No corresponding DataFrame in 'dfs'.")


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

```
Table 'agg_trans': 5034 rows, 7 columns. Corresponding DataFrame shape: (5034, 7)
Table 'agg_user': 6732 rows, 7 columns. Corresponding DataFrame shape: (6732, 7)
Table 'map_trans': 0 rows, 9 columns. Corresponding DataFrame shape: (0, 7)
Table 'map_user': 0 rows, 9 columns. Corresponding DataFrame shape: (0, 7)
Table 'top_trans_dist': 8296 rows, 9 columns. Corresponding DataFrame shape: (8296, 7)
Table 'top_trans_pin': 9997 rows, 7 columns. Corresponding DataFrame shape: (9997, 7)
Table 'top_user_dist': 8296 rows, 8 columns. Corresponding DataFrame shape: (8296, 6)
Table 'top_user_pin': 10000 rows, 6 columns. Corresponding DataFrame shape: (10000, 6)
```

```
# ============================================
# PHONEPE PULSE - PREMIUM DASHBOARD + GEMINI AI
# With Google Gemini API Integration
# ============================================

!pip install -q streamlit pyngrok streamlit-player streamlit-extras xlsxwriter openpyxl plotly seaborn altair google-generativeai

import os
os.makedirs('pages', exist_ok=True)
os.makedirs('Miscellaneous', exist_ok=True)

print("✅ All packages installed!\n")

# ============================================
# STUNNING HOME PAGE WITH GEMINI AI CHATBOT
# ============================================

with open('app.py', 'w') as f:
    f.write('''
```

```
    f.write(
import io
import pandas as pd
import streamlit as st
from streamlit_player import st_player
from streamlit_extras.metric_cards import style_metric_cards
from streamlit_extras.add_vertical_space import add_vertical_space
import google.generativeai as genai
import os # Import os to access environment variables

st.set_page_config(
    page_title='PhonePe Pulse AI',
    layout='wide',
    page_icon='💜',
    initial_sidebar_state='expanded'
)

# ULTRA PREMIUM CSS WITH GLASSMORPHISM
st.markdown("""
    <style>
    @import url('https://fonts.googleapis.com/css2?family=Inter:wght@300;400;500;600;700;800;900&display=swap');

    * {
        font-family: 'Inter', -apple-system, sans-serif;
    }

    /* Animated Gradient Background */
    .main {
        background: linear-gradient(-45deg, #667eea, #764ba2, #f093fb, #4facfe);
        background-size: 400% 400%;
        animation: gradientBG 15s ease infinite;
    }

    @keyframes gradientBG {
        0% { background-position: 0% 50%; }
        50% { background-position: 100% 50%; }
        100% { background-position: 0% 50%; }
    }

    [data-testid="stAppViewContainer"] {
        background: linear-gradient(-45deg, #667eea, #764ba2, #f093fb, #4facfe);
        background-size: 400% 400%;
        animation: gradientBG 15s ease infinite;
    }

    /* Glassmorphism Header */
    [data-testid="stHeader"] {
        background: rgba(255, 255, 255, 0.1);
        backdrop-filter: blur(10px);
        border-bottom: 1px solid rgba(255, 255, 255, 0.2);
    }

    /* Animated Title */
    .main-title {
        font-size: 4.5rem;
        font-weight: 900;
        text-align: center;
        background: linear-gradient(135deg, #ffffff 0%, #f0f9ff 50%, #ffffff 100%);
        background-size: 200% 200%;
        -webkit-background-clip: text;
        -webkit-text-fill-color: transparent;
        animation: shimmer 3s ease-in-out infinite;
        margin: 2rem 0;
        text-shadow: 0 0 40px rgba(255,255,255,0.5);
        letter-spacing: -2px;
    }

    @keyframes shimmer {
        0%, 100% { background-position: 0% 50%; }
        50% { background-position: 100% 50%; }
        100% { background-position: 0% 50%; }
    }

    .subtitle {
        text-align: center;
        font-size: 1.5rem;
        color: #ffffff;
        font-weight: 600;
```

```
        margin-bottom: 3rem;
        text-shadow: 0 2px 10px rgba(0,0,0,0.3);
        animation: fadeInUp 1s ease-out;
    }

    @keyframes fadeInUp {
        from { opacity: 0; transform: translateY(20px); }
        to { opacity: 1; transform: translateY(0); }
    }

    /* Premium Glassmorphism Cards */
    div[data-testid="metric-container"] {
        background: rgba(255, 255, 255, 0.15);
        backdrop-filter: blur(20px);
        border: 1px solid rgba(255, 255, 255, 0.3);
        border-radius: 24px;
        padding: 28px;
        box-shadow:
            0 8px 32px 0 rgba(31, 38, 135, 0.37),
            inset 0 1px 0 0 rgba(255, 255, 255, 0.5);
        transition: all 0.4s cubic-bezier(0.4, 0, 0.2, 1);
        animation: slideInUp 0.8s ease-out backwards;
        position: relative;
        overflow: hidden;
    }

    div[data-testid="metric-container"]::before {
        content: "";
        position: absolute;
        top: -50%;
        left: -50%;
        width: 200%;
        height: 200%;
        background: linear-gradient(45deg, transparent, rgba(255,255,255,0.1), transparent);
        transform: rotate(45deg);
        animation: shine 3s infinite;
    }

    @keyframes shine {
        0% { transform: translateX(-100%) translateY(-100%) rotate(45deg); }
        100% { transform: translateX(100%) translateY(100%) rotate(45deg); }
    }

    div[data-testid="metric-container"]:hover {
        transform: translateY(-12px) scale(1.03);
        box-shadow:
            0 20px 60px 0 rgba(31, 38, 135, 0.5),
            inset 0 1px 0 0 rgba(255, 255, 255, 0.7);
        border-color: rgba(255, 255, 255, 0.5);
    }

    @keyframes slideInUp {
        from { opacity: 0; transform: translateY(40px); }
        to { opacity: 1; transform: translateY(0); }
    }

    div[data-testid="metric-container"]:nth-child(1) { animation-delay: 0.1s; }
    div[data-testid="metric-container"]:nth-child(2) { animation-delay: 0.2s; }
    div[data-testid="metric-container"]:nth-child(3) { animation-delay: 0.3s; }

    div[data-testid="metric-container"] label {
        color: #ffffff !important;
        font-weight: 700;
        font-size: 1rem;
        text-transform: uppercase;
        letter-spacing: 1px;
    }

    div[data-testid="metric-container"] [data-testid="stMetricValue"] {
        color: #ffffff !important;
        font-size: 2.8rem !important;
        font-weight: 900;
        text-shadow: 0 2px 10px rgba(0,0,0,0.3);
    }

    div[data-testid="metric-container"] [data-testid="stMetricDelta"] {
        color: #34d399 !important;
```

```
        font-weight: 700;
        background: rgba(52, 211, 153, 0.2);
        padding: 4px 12px;
        border-radius: 20px;
    }

    /* Info Box Glassmorphism */
    .stAlert {
        background: rgba(255, 255, 255, 0.15);
        backdrop-filter: blur(20px);
        border: 1px solid rgba(255, 255, 255, 0.3);
        border-radius: 20px;
        color: #ffffff;
        box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
        animation: fadeIn 1s ease-out;
    }

    /* Headers with Glow */
    .stMarkdown h1, .stMarkdown h2, .stMarkdown h3 {
        color: #ffffff !important;
        font-weight: 800;
        text-shadow: 0 0 20px rgba(255,255,255,0.5);
    }

    .stMarkdown h2 {
        border-left: 5px solid rgba(255,255,255,0.8);
        padding-left: 20px;
        margin: 2rem 0 1rem 0;
        animation: slideInLeft 0.6s ease-out;
    }

    @keyframes slideInLeft {
        from { opacity: 0; transform: translateX(-30px); }
        to { opacity: 1; transform: translateX(0); }
    }

    .stMarkdown p {
        color: #ffffff;
        font-size: 1.1rem;
        line-height: 1.8;
        text-shadow: 0 1px 3px rgba(0,0,0,0.3);
    }

    /* Premium Buttons */
    .stButton button {
        background: rgba(255, 255, 255, 0.2);
        backdrop-filter: blur(10px);
        color: white;
        border: 2px solid rgba(255, 255, 255, 0.3);
        border-radius: 15px;
        padding: 14px 32px;
        font-weight: 700;
        font-size: 1rem;
        transition: all 0.3s ease;
        box-shadow: 0 4px 20px rgba(0,0,0,0.2);
        text-transform: uppercase;
        letter-spacing: 1px;
    }

    .stButton button:hover {
        background: rgba(255, 255, 255, 0.3);
        transform: translateY(-3px);
        box-shadow: 0 8px 30px rgba(0,0,0,0.3);
        border-color: rgba(255, 255, 255, 0.5);
    }

    /* Chat Container Glassmorphism */
    .chat-container {
        background: rgba(255, 255, 255, 0.1);
        backdrop-filter: blur(20px);
        border: 1px solid rgba(255, 255, 255, 0.3);
        border-radius: 20px;
        padding: 25px;
        margin: 20px 0;
        box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
    }

    /* Chat Messages */
```

```
        /* Chat Messages */
        .user-message {
            background: rgba(255, 255, 255, 0.2);
            backdrop-filter: blur(10px);
            border: 1px solid rgba(255, 255, 255, 0.3);
            border-radius: 15px;
            padding: 15px;
            margin: 10px 0;
            color: white;
            box-shadow: 0 4px 15px rgba(0,0,0,0.2);
        }

        .ai-message {
            background: rgba(102, 126, 234, 0.3);
            backdrop-filter: blur(10px);
            border: 1px solid rgba(255, 255, 255, 0.3);
            border-radius: 15px;
            padding: 15px;
            margin: 10px 0;
            color: white;
            box-shadow: 0 4px 15px rgba(0,0,0,0.2);
        }

        /* Sidebar Glassmorphism */
        [data-testid="stSidebar"] {
            background: rgba(255, 255, 255, 0.1);
            backdrop-filter: blur(20px);
            border-right: 1px solid rgba(255, 255, 255, 0.2);
        }

        /* Tabs */
        .stTabs [data-baseweb="tab-list"] {
            gap: 10px;
            background: rgba(255, 255, 255, 0.1);
            backdrop-filter: blur(10px);
            padding: 10px;
            border-radius: 20px;
        }

        .stTabs [data-baseweb="tab"] {
            background: rgba(255, 255, 255, 0.15);
            backdrop-filter: blur(10px);
            border-radius: 12px;
            color: white;
            font-weight: 700;
            border: 1px solid rgba(255, 255, 255, 0.3);
            transition: all 0.3s ease;
        }

        .stTabs [data-baseweb="tab"]:hover {
            background: rgba(255, 255, 255, 0.25);
            transform: translateY(-2px);
        }

        .stTabs [aria-selected="true"] {
            background: rgba(255, 255, 255, 0.3);
            border: 2px solid rgba(255, 255, 255, 0.5);
        }

        /* Text Input */
        .stTextInput input {
            background: rgba(255, 255, 255, 0.15);
            backdrop-filter: blur(10px);
            border: 1px solid rgba(255, 255, 255, 0.3);
            border-radius: 12px;
            color: white;
            font-size: 1rem;
        }

        .stTextInput input::placeholder {
            color: rgba(255, 255, 255, 0.6);
        }

        /* Hide Streamlit Branding */
        #MainMenu, footer, header { display: none !important; }

        /* Divider */
        hr {
```

```
            border: none;
            height: 2px;
            background: linear-gradient(90deg, transparent, rgba(255,255,255,0.5), transparent);
            margin: 3rem 0;
        }

        /* Video Player */
        .stVideo {
            border-radius: 25px;
            overflow: hidden;
            box-shadow: 0 20px 60px rgba(0,0,0,0.4);
            border: 2px solid rgba(255, 255, 255, 0.3);
        }
        </style>
        """, unsafe_allow_html=True)

    @st.cache_data
    def load_data():
        """Load all CSV files"""
        try:
            data = {
                'agg_trans_df': pd.read_csv(r'/content/Miscellaneous/agg_trans.csv'),
                'agg_user_df': pd.read_csv(r'/content/Miscellaneous/agg_user.csv'),
                'map_trans_df': pd.read_csv(r'/content/Miscellaneous/map_trans.csv'),
                'map_user_df': pd.read_csv(r'/content/Miscellaneous/map_user.csv'),
                'top_trans_dist_df': pd.read_csv(r'/content/Miscellaneous/top_trans_dist.csv'),
                'top_trans_pin_df': pd.read_csv(r'/content/Miscellaneous/top_trans_pin.csv'),
                'top_user_dist_df': pd.read_csv(r'/content/Miscellaneous/top_user_dist.csv'),
                'top_user_pin_df': pd.read_csv(r'/content/Miscellaneous/top_user_pin.csv')
            }

            for key in data:
                if 'Year' in data[key].columns:
                    data[key]['Year'] = data[key]['Year'].astype(str)

            return data
        except Exception as e:
            st.error(f"❌ Error: {e}")
            return None

    def query_gemini(question): # Removed api_key parameter
        """Query Google Gemini AI"""
        try:
            # Get API key from environment variable
            api_key = os.getenv('GOOGLE_API_KEY')
            if not api_key:
                return "Error: Google Gemini API key not found. Please set the GOOGLE_API_KEY environment variable."

            genai.configure(api_key=api_key)
            # Use a potentially more compatible model
            model = genai.GenerativeModel('gemini-pro-latest') # Changed model name

            prompt = f"""You are a helpful AI assistant specialized in digital payments and PhonePe data analytics.
            Provide concise, accurate, and helpful answers.

            Question: {question}

            Answer:"""

            response = model.generate_content(prompt)
            return response.text

        except Exception as e:
            return f"Error: {str(e)}"

    # Load data
    if 'agg_trans_df' not in st.session_state:
        data = load_data()
        if data:
            for key, value in data.items():
                st.session_state[key] = value
            st.session_state['states'] = st.session_state['agg_trans_df']['State'].unique()
            st.session_state['years'] = st.session_state['agg_trans_df']['Year'].unique()
            st.session_state['quarters'] = st.session_state['agg_trans_df']['Quarter'].unique()
        else:
            st.stop()

    agg_trans_df = st.session_state['agg_trans_df']
```

```
agg_trans_df = st.session_state['agg_trans_df']
map_user_df = st.session_state['map_user_df']
top_user_dist_df = st.session_state['top_user_dist_df']

# Initialize chat history
if 'chat_history' not in st.session_state:
    st.session_state.chat_history = []

# Animated Title
st.markdown('<h1 class="main-title">💜 PhonePe Pulse AI</h1>', unsafe_allow_html=True)
st.markdown('<p class="subtitle">AI-Powered Digital Payments Analytics Dashboard</p>', unsafe_allow_html=True)

add_vertical_space(2)

# Description
description = """PhonePe Pulse is a comprehensive data analytics platform providing deep insights into digital payments in India.
With over **30 crore registered users** and **2000 crore transactions**, PhonePe commands **46% UPI market share**.
Explore interactive visualizations, trends, and ask our **Google Gemini AI** assistant anything about digital payments!"""

st.info(description)

add_vertical_space(2)

# Video Player
st_player(url="https://www.youtube.com/watch?v=c_1H6vivsiA", height=480)

add_vertical_space(2)

# Key Metrics
st.markdown("## 📊 Real-Time Metrics")
col1, col2, col3 = st.columns(3)

total_reg_users = top_user_dist_df['Registered_users'].sum()
col1.metric('👫 REGISTERED USERS', f'{total_reg_users/100000000:.2f} Cr', '📈 +12.5%')

total_app_opens = map_user_df['App_opens'].sum()
col2.metric('📲 APP OPENS', f'{total_app_opens/100000000:.2f} Cr', '📈 +8.3%')

col3.metric('💳 TRANSACTIONS', '2000 Cr+', '📈 +15.2%')

add_vertical_space(3)

# AI CHATBOT SECTION
st.markdown("## 🤖 Google Gemini AI Assistant")

st.markdown('<div class="chat-container">', unsafe_allow_html=True)

# Removed API Key Input Field

# Chat Input
user_input = st.text_input(
    "Ask about digital payments, UPI trends, or PhonePe data:",
    placeholder="e.g., What are the latest UPI payment trends in India?",
    key="user_question"
)

col1, col2 = st.columns([1, 5])

with col1:
    send_button = st.button("🚀 Ask AI", use_container_width=True)

with col2:
    clear_button = st.button("🗑️ Clear Chat", use_container_width=True)

if clear_button:
    st.session_state.chat_history = []
    st.rerun()

if send_button and user_input:
    # Removed API key check here
    with st.spinner("🤔 Gemini AI is thinking..."):
        response = query_gemini(user_input) # Removed api_key argument
        st.session_state.chat_history.append({"role": "user", "content": user_input})
        st.session_state.chat_history.append({"role": "ai", "content": response})

# Display Chat History
if st.session_state.chat_history:
    st.markdown("### 💬 Conversation")
```

```
    for message in st.session_state.chat_history:
        if message["role"] == "user":
            st.markdown(f'<div class="user-message">👤 <strong>You:</strong><br>{message["content"]}</div>', unsafe_allow_html=True)
        else:
            st.markdown(f'<div class="ai-message">🤖 <strong>Gemini AI:</strong><br>{message["content"]}</div>', unsafe_allow_html=T
else:
    st.markdown("### 💡 Try asking:")
    st.markdown("""
    - What are the top digital payment trends in India?
    - Explain UPI and its impact on Indian payments
    - What is PhonePe and how does it work?
    - Compare digital payment methods in India
    - What are the security features of UPI?
    """)

st.markdown('</div>', unsafe_allow_html=True)

add_vertical_space(3)

# Navigation
st.markdown("---")
st.markdown("## 🧭 Explore Dashboard")
st.info("💡 Navigate to **Overview**, **Transactions**, **Users**, **Trends**, and **Comparisons** using the sidebar!")
''')

print("✅ Stunning Gemini AI-powered Home page created!")

# Create remaining pages (keeping your original code)
# Pages 1-5 code here...
# I will assume the code for pages 1-5 is already present in the notebook
# and only need to be written to files.

# Write other pages to files
# Write Overview page
with open('pages/1_📊_Overview.py', 'w') as f:
    f.write('''
import streamlit as st
import plotly.express as px
import json
import pandas as pd
from streamlit_extras.add_vertical_space import add_vertical_space

st.set_page_config(page_title='Overview', layout='wide', page_icon='📊')

if 'agg_trans_df' not in st.session_state:
    st.error("❌ Data not loaded. Please go to Home page first.")
    st.stop()

agg_trans = st.session_state["agg_trans_df"]
map_trans = st.session_state["map_trans_df"]
map_user = st.session_state["map_user_df"]

st.title(':blue[Overview]')
add_vertical_space(2)

# Chart 1: Transaction Breakdown by Type
st.subheader(":blue[Transaction Breakdown by Type]")
trans_type_count = agg_trans.groupby('Transaction_type')['Transaction_count'].sum()
total_trans_count = agg_trans['Transaction_count'].sum()
trans_type_perc = round(trans_type_count / total_trans_count * 100, 2).reset_index()

trans_type_fig = px.pie(
    trans_type_perc, names='Transaction_type', values='Transaction_count',
    hole=.65, hover_data={'Transaction_count': False}
)
trans_type_fig.update_layout(width=900, height=500)
st.plotly_chart(trans_type_fig, use_container_width=True)

add_vertical_space(2)

# Chart 2: Transaction Count by State
st.subheader(":blue[Transaction Count by State]")
trans_state = agg_trans.groupby('State')['Transaction_count'].sum().reset_index()
trans_state_sorted = trans_state.sort_values(by='Transaction_count', ascending=False).head(15)

trans_state_fig = px.bar(
    trans_state_sorted, x='Transaction_count', y='State', orientation='h',
```

```
    text='Transaction_count', text_auto='.2s',
    labels={'Transaction_count': "Transaction Count"}
)
trans_state_fig.update_layout(yaxis=dict(autorange="reversed"), width=900, height=500)
st.plotly_chart(trans_state_fig, use_container_width=True)

add_vertical_space(2)

# Chart 3: Transaction Count by District
st.subheader(":blue[Transaction Count by District]")
trans_district = map_trans.groupby(['State', 'District'])[['Transaction_count']].sum().reset_index()
trans_district_sorted = trans_district.sort_values(by='Transaction_count', ascending=False).head(15)

trans_district_fig = px.bar(
    trans_district_sorted, x='Transaction_count', y='District', orientation='h',
    text='Transaction_count', text_auto='.2s',
    labels={'Transaction_count': "Transaction Count"},
    hover_name='State', hover_data={'State': False, 'District': True}
)
trans_district_fig.update_layout(yaxis=dict(autorange="reversed"), width=900, height=500)
st.plotly_chart(trans_district_fig, use_container_width=True)

add_vertical_space(2)

# Chart 4: Registered User Count by State
st.subheader(':blue[Registered User Count by State]')
user_state = map_user.groupby('State')['Registered_users'].sum().reset_index()
user_state_sorted = user_state.sort_values(by='Registered_users', ascending=False).head(15)

user_state_fig = px.bar(
    user_state_sorted, x='Registered_users', y='State', orientation='h',
    text='Registered_users', text_auto='.2s',
    color='Registered_users', color_continuous_scale='Reds'
)
user_state_fig.update_layout(yaxis=dict(autorange="reversed"), height=600, width=900)
st.plotly_chart(user_state_fig, use_container_width=True)
''')

# Write Transaction page
with open('pages/2_💳_Transaction.py', 'w') as f:
    f.write('''
import streamlit as st
import plotly.express as px
from streamlit_extras.add_vertical_space import add_vertical_space

st.set_page_config(page_title='Transaction', layout='wide', page_icon='💳')

if 'agg_trans_df' not in st.session_state:
    st.error("❌ Data not loaded. Please go to Home page first.")
    st.stop()

agg_trans = trans_df = trans_df_2 = st.session_state["agg_trans_df"]
map_df = st.session_state["map_trans_df"]

states = st.session_state["states"]
years = st.session_state["years"]
quarters = st.session_state["quarters"]

st.title(':blue[Transaction]')
add_vertical_space(3)

# Section 1: Transaction amount breakdown
st.subheader(':blue[Transaction amount breakdown]')

col1, col2, col3 = st.columns([5, 3, 1])

state1 = col1.selectbox("State", states, key='state1')
year1 = col2.selectbox("Year", years, key='year1')
quarter_options = ["All"] + list(map(str, quarters))
quarter1 = col3.selectbox("Quarter", quarter_options, key='quarter1')

trans_df = trans_df[(trans_df["State"] == state1) & (trans_df["Year"] == year1)]

if quarter1 != 'All':
    trans_df = trans_df[(trans_df["Quarter"] == int(quarter1))]

trans_df = trans_df.sort_values("Transaction_amount", ascending=False).reset_index(drop=True)
```

```
suffix1 = " quarters" if quarter1 == 'All' else "st" if quarter1 == '1' else "nd" if quarter1 == '2' else "rd" if quarter1 == '3' els
title1 = f"Transaction details of {state1} for {quarter1.lower()}{suffix1} {'' if quarter1 == 'All' else 'quarter'} of {year1}"

fig1 = px.bar(
    trans_df, x="Transaction_type", y="Transaction_amount",
    color="Transaction_type",
    color_discrete_sequence=px.colors.qualitative.Plotly,
    title=title1,
    labels=dict(Transaction_amount='Transaction Amount', Transaction_type='Transaction Type'),
    hover_data={'Quarter': True}
)

fig1.update_layout(
    showlegend=False,
    title={'x': 0.5, 'xanchor': 'center', 'y': 0.9, 'yanchor': 'top'},
    width=900, height=500
)

fig1.update_traces(marker=dict(line=dict(width=1, color='DarkSlateGrey')))

st.plotly_chart(fig1, use_container_width=True)

expander1 = st.expander(label='Detailed view')
expander1.write(trans_df.loc[:, ['Quarter', 'Transaction_type', 'Transaction_amount']].reset_index(drop=True))

add_vertical_space(2)

# Section 2: Transaction Hotspots - Districts
st.subheader(':blue[Transaction Hotspots - Districts]')

year_col, quarter_col, buff = st.columns([1, 1, 4])

year2 = year_col.selectbox("Year", years, key='year2')
quarter2 = quarter_col.selectbox("Quarter", quarter_options, key='quarter2')

map_df = map_df[map_df["Year"] == year2]

if quarter2 != 'All':
    map_df = map_df[(map_df["Quarter"] == int(quarter2))]

suffix2 = " quarters" if quarter2 == 'All' else "st" if quarter2 == '1' else "nd" if quarter2 == '2' else "rd" if quarter2 == '3' els
title2 = f"Transaction hotspots for {quarter2.lower()}{suffix2} {'' if quarter2 == 'All' else 'quarter'} of {year2}"

fig2 = px.scatter_mapbox(
    map_df, lat="Latitude", lon="Longitude",
    size="Transaction_amount", hover_name="District",
    hover_data={"Transaction_count": True, "Transaction_amount": True, 'Quarter': True},
    title=title2,
    color_discrete_sequence=px.colors.sequential.Plotly3
)

fig2.update_layout(
    mapbox_style='carto-positron',
    mapbox_zoom=3.45, mapbox_center={"lat": 20.93684, "lon": 78.96288},
    geo=dict(scope='asia', projection_type='equirectangular'),
    title={'x': 0.5, 'xanchor': 'center', 'y': 0.04, 'yanchor': 'bottom', 'font': dict(color='black')},
    margin={"r": 0, "t": 0, "l": 0, "b": 0}, width=900, height=500
)

st.plotly_chart(fig2, use_container_width=True)

expander2 = st.expander(label='Detailed view')
expander2.write(map_df.loc[:, ['State', 'District', 'Quarter', 'Transaction_amount']].reset_index(drop=True))

add_vertical_space(2)

# Section 3: Breakdown by transaction count proportion
st.subheader(":blue[Breakdown by transaction count proportion]")

state_pie, year_pie, quarter_pie = st.columns([5, 3, 1])

state3 = state_pie.selectbox('State', options=states, key='state3')
year3 = year_pie.selectbox('Year', options=years, key='year3')
quarter3 = quarter_pie.selectbox('Quarter', options=quarter_options, key='quarter3')

filtered_trans = trans_df_2[(trans_df_2.State == state3) & (trans_df_2.Year == year3)]
```

```
    if quarter3 != 'All':
        filtered_trans = filtered_trans[filtered_trans.Quarter == int(quarter3)]

    fig3 = px.pie(
        filtered_trans, names='Transaction_type',
        values='Transaction_count', hole=.65
    )

    fig3.update_layout(width=900, height=500)

    st.plotly_chart(fig3, use_container_width=True)

    expander3 = st.expander(label='Detailed view')
    expander3.write(filtered_trans.loc[:, ['Quarter', 'Transaction_type', 'Transaction_count']].reset_index(drop=True))
    ''')

    # Write Users page
    with open('pages/3_👥_Users.py', 'w') as f:
        f.write('''
    import streamlit as st
    import plotly.express as px
    from streamlit_extras.add_vertical_space import add_vertical_space

    st.set_page_config(page_title='Users', layout='wide', page_icon='👥')

    if 'agg_trans_df' not in st.session_state:
        st.error("❌ Data not loaded. Please go to Home page first.")
        st.stop()

    st.title(':blue[Users Analysis]')
    add_vertical_space(2)

    # Add user-specific visualizations here
    # This is a placeholder - you can add your Users page content

    map_user = st.session_state["map_user_df"]
    top_user_dist = st.session_state["top_user_dist_df"]

    st.subheader(":blue[User Distribution Analysis]")

    col1, col2 = st.columns(2)

    with col1:
        user_state = map_user.groupby('State')['Registered_users'].sum().reset_index()
        user_state_sorted = user_state.sort_values(by='Registered_users', ascending=False).head(10)

        fig1 = px.bar(
            user_state_sorted, x='Registered_users', y='State', orientation='h',
            text='Registered_users', text_auto='.2s',
            title="Top 10 States by Registered Users"
        )
        fig1.update_layout(yaxis=dict(autorange="reversed"))
        st.plotly_chart(fig1, use_container_width=True)

    with col2:
        user_district = top_user_dist.groupby('District')['Registered_users'].sum().reset_index()
        user_district_sorted = user_district.sort_values(by='Registered_users', ascending=False).head(10)

        fig2 = px.bar(
            user_district_sorted, x='Registered_users', y='District', orientation='h',
            text='Registered_users', text_auto='.2s',
            title="Top 10 Districts by Registered Users"
        )
        fig2.update_layout(yaxis=dict(autorange="reversed"))
        st.plotly_chart(fig2, use_container_width=True)

    st.info("💡 This is a sample Users page. You can customize it with your specific user analytics.")
    ''')

    # Write Trend Analysis page
    with open('pages/4_📈_Trend_Analysis.py', 'w') as f:
        f.write('''
    import streamlit as st
    import plotly.express as px
    import altair as alt
    from streamlit_extras.add_vertical_space import add_vertical_space

    st.set_page_config(page_title='Trend Analysis', layout='wide', page_icon='📈')
```

```
st.set_page_config(page_title="Trend Analysis", layout="wide", page_icon="📈")

if 'agg_trans_df' not in st.session_state:
    st.error("❌ Data not loaded. Please go to Home page first.")
    st.stop()

map_trans = st.session_state['map_trans_df']
top_trans_dist = dist_trans = st.session_state["top_trans_dist_df"]
pin_trans = st.session_state['top_trans_pin_df']

def filter_top_trans_dist(top_trans_dist, year, quarter):
    filtered_top_trans_dist = top_trans_dist[top_trans_dist['Year'] == year]
    if quarter != 'All':
        filtered_top_trans_dist = filtered_top_trans_dist[filtered_top_trans_dist['Quarter'] == int(quarter)]
    return filtered_top_trans_dist

st.title(':blue[Trend Analysis]')
add_vertical_space(3)

# Section 1: Transaction Count and Amount - Trend over the years
st.subheader(':blue[Transaction Count and Amount - Trend over the years]')
add_vertical_space(1)

col1, col2, col3, col4 = st.columns([3, 4, 4, 2])

region1 = col1.selectbox('Region', map_trans["Region"].unique(), key='region1')
df = map_trans[map_trans['Region'] == region1]

state1 = col2.selectbox('State', df['State'].unique(), key='state1')
df = df[df['State'] == state1]

district1 = col3.selectbox('District', df['District'].unique(), key='district1')
df = df[df['District'] == district1]

year_options = ['All'] + [year for year in st.session_state['years']]
year1 = col4.selectbox('Year', year_options, key='year1')

title1 = f'Transaction count trend for {district1} district in {state1} across {str(year1).lower()} years'
title2 = f'Transaction amount trend for {district1} district in {state1} across {str(year1).lower()} years'

if year1 != 'All':
    df = df[df['Year'] == year1]
    title1 = f'Transaction count trend for {district1} district in {state1} during {year1}'
    title2 = f'Transaction amount trend for {district1} district in {state1} during {year1}'

fig1 = px.line(df, x='Quarter', y='Transaction_count', color='Year', title=title1)
fig1.update_xaxes(tickmode='array', tickvals=list(range(1, 5)))
fig1.update_layout(
    height=500, width=900,
    yaxis_title='Transaction Count',
    title={'x': 0.5, 'xanchor': 'center', 'y': 0.9, 'yanchor': 'bottom'}
)

fig2 = px.line(df, x='Quarter', y='Transaction_amount', color='Year', title=title2)
fig2.update_xaxes(tickmode='array', tickvals=list(range(1, 5)))
fig2.update_layout(
    height=500, width=900,
    yaxis_title='Transaction Amount',
    title={'x': 0.5, 'xanchor': 'center', 'y': 0.9, 'yanchor': 'bottom'}
)

tab1, tab2 = st.tabs(['🏅Transaction Count Trend', '💰Transaction Amount Trend'])

tab1.plotly_chart(fig1, use_container_width=True)
expander1 = tab1.expander('Detailed view')
expander1.write(df.loc[:, ['Region', 'District', 'Year', 'Quarter', 'Transaction_count']].reset_index(drop=True))

tab2.plotly_chart(fig2, use_container_width=True)
expander2 = tab2.expander('Detailed view')
expander2.write(df.loc[:, ['Region', 'District', 'Year', 'Quarter', 'Transaction_amount']].reset_index(drop=True))

add_vertical_space(2)

# Section 2: Top Districts
st.subheader(':blue[Transaction Count and Amount - Top Districts]')

col5, col6, col7 = st.columns([5, 3, 1])
```

```
state_options = ["All"] + list(st.session_state['states'])
year_options = st.session_state["years"]
quarter_options = ["All"] + list(map(str, st.session_state['quarters']))

state2 = col5.selectbox("State", state_options, key="state2")
year2 = col6.selectbox("Year", year_options, key="year2")
quarter2 = col7.selectbox("Quarter", quarter_options, key="quarter2")

if state2 != "All":
    top_trans_dist = top_trans_dist[top_trans_dist["State"] == state2]

top_trans_dist = top_trans_dist[top_trans_dist["Year"] == year2]

if quarter2 != "All":
    top_trans_dist = top_trans_dist[top_trans_dist["Quarter"] == int(quarter2)]

top_dist_grouped_1 = top_trans_dist.groupby("District")["Transaction_count"].sum().nlargest(10).index.tolist()
top_trans_dist_filtered_1 = top_trans_dist[top_trans_dist["District"].isin(top_dist_grouped_1)]

suffix1 = " quarters" if quarter2 == 'All' else "st" if quarter2 == '1' else "nd" if quarter2 == '2' else "rd" if quarter2 == '3' els
title3 = f"Top districts in {'India' if state2 == 'All' else state2} by Transaction count during {str(quarter2).lower()}{suffix1} {''

axis_format = '~s'

chart1 = alt.Chart(top_trans_dist_filtered_1, height=500, width=900).mark_bar(size=18).encode(
    x=alt.X("Transaction_count", title="Transaction Count", axis=alt.Axis(format=axis_format)),
    y=alt.Y("District", sort=top_dist_grouped_1, title=None),
    color="State",
    tooltip=["District", "State", "Year", "Quarter", "Transaction_count"]
).properties(
    title=alt.TitleParams(text=title3, align="center", anchor='middle', baseline="bottom")
).configure_axis(grid=False)

top_dist_grouped_2 = top_trans_dist.groupby("District")["Transaction_amount"].sum().nlargest(10).index.tolist()
top_trans_dist_filtered_2 = top_trans_dist[top_trans_dist["District"].isin(top_dist_grouped_2)]

title4 = f"Top districts in {'India' if state2 == 'All' else state2} by Transaction amount during {str(quarter2).lower()}{suffix1} {'

chart2 = alt.Chart(top_trans_dist_filtered_2, height=500, width=900).mark_bar(size=18).encode(
    x=alt.X("sum(Transaction_amount)", title="Transaction Amount", axis=alt.Axis(format=axis_format)),
    y=alt.Y("District", sort=top_dist_grouped_2, title=None),
    color="State",
    tooltip=["District", "State", "Year", "Quarter", "Transaction_amount"]
).properties(
    title=alt.TitleParams(text=title4, align="center", anchor='middle', baseline="bottom")
).configure_axis(grid=False)

tab3, tab4 = st.tabs(['🥇Transaction Count - Top Districts', '💰Transaction Amount - Top Districts'])

tab3.altair_chart(chart1, use_container_width=True)
expander3 = tab3.expander('Detailed view')
expander3.write(top_trans_dist_filtered_1.loc[:, ['State', 'District', 'Quarter', 'Transaction_count']].reset_index(drop=True))

tab4.altair_chart(chart2, use_container_width=True)
expander4 = tab4.expander('Detailed view')
expander4.write(top_trans_dist_filtered_2.loc[:, ['State', 'District', 'Quarter', 'Transaction_amount']].reset_index(drop=True))

add_vertical_space(2)

# Section 3: Other Key Trends
st.subheader(':blue[Other Key Trends over the years]')

col8, col9, col10 = st.columns([5, 3, 1])

trend3 = col8.selectbox(
    'Trend',
    ('Top 10 States by Transaction Volume', 'Top 10 Districts by Transaction Volume', 'Top 10 Pincodes by Transaction Volume'),
    key='trend3'
)

year3 = col9.selectbox('Year', st.session_state["years"], key='year3')
quarter3 = col10.selectbox('Quarter', quarter_options, key='quarter3')

filtered_dist_trans = filter_top_trans_dist(dist_trans, year3, quarter3)
filtered_pin_trans = filter_top_trans_dist(pin_trans, year3, quarter3)

filtered_top_states = filtered_dist_trans.groupby('State')['Transaction_amount'].sum().reset_index().sort_values('Transaction_amount'
filtered_top_districts = filtered_dist_trans.groupby('District')['Transaction_amount'].sum().reset_index().sort_values('Transaction_a
```

```
filtered_top_districts = filtered_dist_trans.groupby('District')['Transaction_amount'].sum().reset_index().sort_values('Transaction_a
filtered_top_pincodes = filtered_pin_trans.groupby('Pincode')['Transaction_amount'].sum().reset_index().sort_values('Transaction_amou
filtered_top_pincodes['Pincode'] = filtered_top_pincodes['Pincode'].astype(str)

suffix2 = " quarters" if quarter3 == 'All' else "st" if quarter3 == '1' else "nd" if quarter3 == '2' else "rd" if quarter3 == '3' els

title5 = f"Top 10 states by Transaction volume {'across' if quarter3 == 'All' else 'in'} {str(quarter3).lower()}{suffix2} {'' if quar
title6 = f"Top 10 districts by Transaction volume {'across' if quarter3 == 'All' else 'in'} {str(quarter3).lower()}{suffix2} {'' if q
title7 = f"Top 10 pincode locations by Transaction volume {'across' if quarter3 == 'All' else 'in'} {str(quarter3).lower()}{suffix2}

if trend3 == 'Top 10 States by Transaction Volume':
    chart3 = alt.Chart(filtered_top_states, height=500, width=900).mark_bar(size=18).encode(
        x=alt.X('Transaction_amount', axis=alt.Axis(format=axis_format), title="Transaction Amount"),
        y=alt.Y('State', sort='-x'),
        tooltip=['State', alt.Tooltip('Transaction_amount', format='.2f')]
    ).properties(title=alt.TitleParams(text=title5, align="center", anchor='middle'))
    data = filtered_top_states

elif trend3 == 'Top 10 Districts by Transaction Volume':
    chart3 = alt.Chart(filtered_top_districts, height=500, width=900).mark_bar(size=18).encode(
        x=alt.X('Transaction_amount', axis=alt.Axis(format=axis_format), title="Transaction Amount"),
        y=alt.Y('District', sort='-x'),
        tooltip=['District', alt.Tooltip('Transaction_amount', format='.2f')]
    ).properties(title=alt.TitleParams(text=title6, align="center", anchor='middle'))
    data = filtered_top_districts

else:
    chart3 = alt.Chart(filtered_top_pincodes, height=500, width=900).mark_bar(size=18).encode(
        x=alt.X('Transaction_amount', axis=alt.Axis(format=axis_format), title="Transaction Amount"),
        y=alt.Y('Pincode', sort='-x'),
        tooltip=['Pincode', alt.Tooltip('Transaction_amount', format='.2f')]
    ).properties(title=alt.TitleParams(text=title7, align="center", anchor='middle'))
    data = filtered_top_pincodes

st.altair_chart(chart3, use_container_width=True)

expander5 = st.expander('Detailed view')
expander5.dataframe(data.reset_index(drop=True))
''')

# Write Comparative Analysis page
with open('pages/5_⚖️_Comparative_Analysis.py', 'w') as f:
    f.write('''
import streamlit as st
import seaborn as sns
import pandas as pd
import plotly.express as px
from streamlit_extras.add_vertical_space import add_vertical_space
import matplotlib.pyplot as plt

st.set_page_config(page_title='Comparative Analysis', layout='wide', page_icon='⚖️')

if 'agg_trans_df' not in st.session_state:
    st.error("❌ Data not loaded. Please go to Home page first.")
    st.stop()

trans_df1 = trans_df2 = st.session_state['agg_trans_df'].copy()
user_df = st.session_state["agg_user_df"]

trans_df1["Transaction_amount(B)"] = trans_df1["Transaction_amount"] / 1e9
year_order = sorted(trans_df1["Year"].unique())
trans_df1["Year"] = pd.Categorical(trans_df1["Year"], categories=year_order, ordered=True)

trans_df2["Transaction_amount(B)"] = trans_df2["Transaction_amount"] / 1e9

quarter_options = ["All"] + list(map(str, st.session_state['quarters']))

st.title(':blue[Comparative Analysis]')
add_vertical_space(3)

# Section 1: Regionwise Transaction volume comparison
st.subheader(':blue[Regionwise Transaction volume comparison]')

fig1 = sns.catplot(
    x="Year", y="Transaction_amount",
    col="Region", data=trans_df1,
    kind="bar", errorbar=None,
    height=5, aspect=1.5, col_wrap=2,
```

```
        sharex=False
    )

    for ax in fig1.axes.flat:
        ax.set_yticklabels(['₹. {:,.0f}B'.format(y / 1e9) for y in ax.get_yticks()])
        ax.set_ylabel('Transaction Amount')

    sns.set_style("white")
    st.pyplot(fig1)

    add_vertical_space(2)

    # Section 2: Transaction breakdown by Transaction type
    st.subheader(':blue[Transaction breakdown by Transaction type]')

    col1, col2, col3 = st.columns([5, 3, 1])

    selected_states = col1.multiselect("Select state(s)", st.session_state['states'], key='selected_states')
    year1 = col2.selectbox("Year", st.session_state['years'], key='year1')
    quarter1 = col3.selectbox("Quarter", quarter_options, key='quarter1')

    trans_df1_filtered = trans_df1[(trans_df1["Year"] == year1)]

    if quarter1 != "All":
        trans_df1_filtered = trans_df1_filtered[(trans_df1_filtered["Quarter"] == int(quarter1))]

    suffix1 = " quarters" if quarter1 == 'All' else "st" if quarter1 == '1' else "nd" if quarter1 == '2' else "rd" if quarter1 == '3' els

    title1 = f"Transaction details comparison of the selected states for {str(quarter1).lower()}{suffix1} {'' if quarter1 == 'All' else '

    if len(selected_states) == 1:
        state_str = ''.join(selected_states)
        title1 = f"Transaction details of {state_str} for {str(quarter1).lower()}{suffix1} {'' if quarter1 == 'All' else 'quarter'} of {y

    if selected_states:
        trans_df1_filtered = trans_df1_filtered[trans_df1_filtered["State"].isin(selected_states)]
        trans_df1_filtered = trans_df1_filtered.sort_values("Transaction_count", ascending=False)

        fig2 = px.bar(
            trans_df1_filtered, x="Transaction_type", y="Transaction_count",
            color="State",
            color_discrete_sequence=px.colors.qualitative.Plotly,
            barmode='group',
            title=title1,
            labels=dict(Transaction_count='Transaction Count', Transaction_type='Transaction Type'),
            hover_data={'Quarter': True}
        )

        fig2.update_layout(
            width=900, height=550,
            title={'x': 0.5, 'xanchor': 'center', 'y': 0.9, 'yanchor': 'top'}
        )

        fig2.update_traces(marker=dict(line=dict(width=1, color='DarkSlateGrey')))

        st.plotly_chart(fig2, use_container_width=True)

    else:
        column, buffer = st.columns([5, 4])
        column.info("Please select at least one state to display the plot.")
        add_vertical_space(8)

    add_vertical_space(2)

    # Section 3: Transaction amount comparison - Quarterwise
    st.subheader(':blue[Transaction amount comparison - Quarterwise]')

    col4, col5, buff = st.columns([3, 2, 4])

    region2 = col4.selectbox('Region', trans_df2['Region'].unique(), key='region2')
    year2 = col5.selectbox('Year', st.session_state['years'], key='year2')

    filtered_df = trans_df2[(trans_df2['Region'] == region2) & (trans_df2['Year'] == year2)]

    filtered_df['Quarter'] = 'Quarter ' + filtered_df['Quarter'].astype(str)

    fig3 = px.pie(
```

```
    filtered_df, values='Transaction_amount(B)',
    names='Quarter', color='Quarter',
    title=f'Transaction amount Comparison of {region2} for the year {year2}'
)

fig3.update_layout(
    width=850, height=550,
    title={'x': 0.45, 'xanchor': 'center', 'y': 0.9, 'yanchor': 'top'}
)

fig3.update_traces(textposition='inside', textinfo='percent+label')

st.plotly_chart(fig3, use_container_width=True)

filtered_df['Year'] = filtered_df["Year"].astype(int)

expander1 = st.expander('Detailed view')
expander1.dataframe(
    filtered_df.groupby(['Year', 'Quarter']).agg({'Transaction_amount(B)': sum}).reset_index().sort_values(
        'Transaction_amount(B)', ascending=False
    ).loc[:, ['Quarter', 'Transaction_amount(B)']].reset_index(drop=True)
)
''')


print("\n" + "="*60)
print("✨ PREMIUM GEMINI AI DASHBOARD CREATED!")
print("="*60)

# Run with ngrok (kill existing tunnels first)
from pyngrok import ngrok
import subprocess
import time
from google.colab import userdata # Import userdata to access secrets

print("\n🔴 Cleaning up existing tunnels...")
os.system("pkill -9 ngrok 2>/dev/null")
time.sleep(3)
os.system("pkill -f streamlit 2>/dev/null")
time.sleep(2)

try:
    ngrok.kill()
except:
    pass

time.sleep(2)

print("\n🔐 Loading Gemini API key from Colab secrets...")
# Load the API key from Colab secrets
gemini_api_key = userdata.get('GOOGLE_API_KEY')

if not gemini_api_key:
    print("❌ Error: Google Gemini API key not found in Colab secrets.")
    print("Please add it to the secrets manager (🔑 icon on the left) with the name 'GOOGLE_API_KEY'.")
else:
    print("✅ API key loaded successfully!")
    print("\n🚀 Starting Streamlit...")
    # Pass the API key as an environment variable
    env = os.environ.copy()
    env['GOOGLE_API_KEY'] = gemini_api_key

    process = subprocess.Popen(
        ["streamlit", "run", "app.py", "--server.port", "8501", "--server.headless", "true"],
        stdout=subprocess.DEVNULL,
        stderr=subprocess.DEVNULL,
        env=env # Pass the environment variables
    )

    time.sleep(10) # Give Streamlit time to start

    # Check if Streamlit process is still running
    if process.poll() is not None:
        print("\n❌ Error: Streamlit app failed to start.")
        print("Check the Streamlit logs for details (e.g., in the output of previous cells if not suppressed).")
    else:
        print("🌐 Creating public URL...\n")
        try:
```

```
        public_url = ngrok.connect(8501)

        print("=" * 60)
        print("✅ SUCCESS! GEMINI AI DASHBOARD IS LIVE!")
        print("=" * 60)
        print(f"\n🔗 {public_url}")
        print("\n🎆 Features:")
        print("   • Stunning animated gradient UI")
        print("   • Google Gemini AI Chatbot (FREE!)")
        print("   • 5 interactive analytics pages")
        print("   • Real-time data insights")
        print("\n⚠️  Keep this cell running to keep the dashboard live!")
        print("=" * 60)
    except Exception as e:
        print(f"\n❌ Error creating ngrok tunnel: {e}")
        print("\n💡 Ensure you have a valid ngrok token and that no other processes are using port 8501.")
```

```
✅ All packages installed!

✅ Stunning Gemini AI-powered Home page created!

============================================================
✨ PREMIUM GEMINI AI DASHBOARD CREATED!
============================================================

🛑 Cleaning up existing tunnels...

🔐 Loading Gemini API key from Colab secrets...
✅ API key loaded successfully!

🚀 Starting Streamlit...
🌐 Creating public URL...

============================================================
✅ SUCCESS! GEMINI AI DASHBOARD IS LIVE!
============================================================

🔗 NgrokTunnel: "https://fanny-unradiative-noblemanly.ngrok-free.dev" -> "http://localhost:8501"

🎆 Features:
   • Stunning animated gradient UI
   • Google Gemini AI Chatbot (FREE!)
   • 5 interactive analytics pages
   • Real-time data insights

⚠️  Keep this cell running to keep the dashboard live!
============================================================
```

```
# Create a new Streamlit page file for the "About" section
about_page_content = """
import streamlit as st
from streamlit_extras.add_vertical_space import add_vertical_space
import os

st.set_page_config(page_title='About', layout='wide', page_icon='👤')

st.title(':blue[About Me]')
add_vertical_space(1)

# Define image paths
PROFILE_IMAGE_PATH = "/content/user.jpg"
COVER_IMAGE_PATH = "/content/cover photo.jpg"

# Cover Photo Section
if os.path.exists(COVER_IMAGE_PATH):
    st.image(COVER_IMAGE_PATH, use_container_width=True)
else:
    st.warning(f"Cover photo not found at {COVER_IMAGE_PATH}. Please upload it.")

add_vertical_space(1)

col1, col2 = st.columns([1, 3])

with col1:
    # Profile Photo
    if os.path.exists(PROFILE_IMAGE_PATH):
        st.image(PROFILE_IMAGE_PATH, caption="Laxman Rathod", width=150)
    else:
        st.warning(f"Profile photo not found at {PROFILE_IMAGE_PATH}. Please upload it.")
```

```
with col2:
    st.subheader("Laxman Rathod")
    st.markdown(f"- **LinkedIn:** [Laxman Rathod](https://www.linkedin.com/in/laxman-rathod-627264102)")
    st.markdown(f"- **GitHub:** [proglax](https://github.com/proglax)")
add_vertical_space(2)

st.subheader("📂 Project Repositories")
st.markdown("- [Walmart-time-series-and-online-Retail-Data-Analysis](https://github.com/ProgLax/Walmart-time-series-and-online-Reta
st.markdown("- [covid-19](https://github.com/ProgLax/covid-19)")
st.markdown("- [Netflix-prediction-engine](https://github.com/ProgLax/Netflix-prediction-engine-)")
st.markdown("- [Wallmart](https://github.com/ProgLax/Walmart-python-)")
st.markdown("- [paisa bazaar banking fraud analysis](https://github.com/ProgLax/paisabazaarbankingfraudanalysis)")
st.markdown("- [phonepe pulse](https://github.com/ProgLax/phonepe-pulse-data-project-)")


"""




with open('pages/6_👤_About.py', 'w') as f:
    f.write(about_page_content)

print("✅ 'About' page created in pages/6_👤_About.py")
```

```
✅ 'About' page created in pages/6_👤_About.py
```

```
Start coding or generate with AI.
```