```
import pandas as pd

# Load the datasets
city_target_passenger_rating = pd.read_csv('city_target_passenger_rating.csv')
dim_city = pd.read_csv('dim_city.csv')
dim_date = pd.read_csv('dim_date.csv')
dim_repeat_trip_distribution = pd.read_csv('dim_repeat_trip_distribution.csv')
fact_passenger_summary = pd.read_csv('fact_passenger_summary.csv')
fact_trips = pd.read_csv('fact_trips.csv')
monthly_target_new_passengers = pd.read_csv('monthly_target_new_passengers.csv')
monthly_target_trips = pd.read_csv('monthly_target_trips.csv')
```

Top and Bottom Cities

```
# Group by city_id and sum the total trips
total_trips_by_city = fact_trips.groupby('city_id')['trip_id'].count().reset_index()
total_trips_by_city.columns = ['city_id', 'total_trips']


# Merge with dim_city to get city names
total_trips_with_names = total_trips_by_city.merge(dim_city, on='city_id', how='left')


# Sort to get the top and bottom cities
top_cities = total_trips_with_names.sort_values(by='total_trips', ascending=False).head(3)
bottom_cities = total_trips_with_names.sort_values(by='total_trips', ascending=True).head(3)

# Display the results
print("Top 3 Cities by Total Trips:")
print(top_cities[['city_name', 'total_trips']])

print("\nBottom 3 Cities by Total Trips:")
print(bottom_cities[['city_name', 'total_trips']])
```

```
Top 3 Cities by Total Trips:
  city_name  total_trips
7    Jaipur        76888
9   Lucknow        64299
2     Surat        54843

Bottom 3 Cities by Total Trips:
       city_name  total_trips
4         Mysore        16238
8     Coimbatore        21104
0  Visakhapatnam        28366
```

Compare Against Monthly Targets

```
# Merge with monthly_target_trips to get target trips
target_trips_by_city = total_trips_with_names.merge(monthly_target_trips, on='city_id', how='left')

# You can now calculate the difference between actual trips and target trips
target_trips_by_city['trip_difference'] = target_trips_by_city['total_trips'] - target_trips_by_city['total_target_trips']

# Display the cities with the largest positive and negative differences
print("\nCities with the Largest Positive Trip Difference (Exceeded Targets):")
print(target_trips_by_city[target_trips_by_city['trip_difference'] > 0].sort_values(by='trip_difference', ascending=False).head(3))

print("\nCities with the Largest Negative Trip Difference (Fell Short of Targets):")
print(target_trips_by_city[target_trips_by_city['trip_difference'] < 0].sort_values(by='trip_difference', ascending=True).head(3))
```

```

Cities with the Largest Positive Trip Difference (Exceeded Targets):
   city_id  total_trips city_name       month  total_target_trips  \
43    RJ01        76888    Jaipur  2024-05-01                9500   
45    RJ01        76888    Jaipur  2024-06-01                9500   
46    RJ01        76888    Jaipur  2024-04-01                9500   

    trip_difference  
43            67388  
45            67388  
46            67388  

Cities with the Largest Negative Trip Difference (Fell Short of Targets):
Empty DataFrame
Columns: [city_id, total_trips, city_name, month, total_target_trips, trip_difference]
Index: []
```

Average Per Trip and Average Trip Distance

```
# Merge fact_trips with dim_city on city_id
merged_data = pd.merge(fact_trips, dim_city, on='city_id')

# Calculate average fare per trip and average distance per city
avg_fare_per_city = merged_data.groupby('city_name').agg(average_fare_per_trip=('fare_amount', 'mean'),average_trip_distance=('distance_

# Calculate price per kilometer
avg_fare_per_city['price_per_km'] = avg_fare_per_city['average_fare_per_trip'] / avg_fare_per_city['average_trip_distance']

# Sort by price per km to identify the highest and lowest pricing efficiencies
sorted_avg_fare = avg_fare_per_city.sort_values(by='price_per_km', ascending=False)

# Display the results
print("Top 3 Cities by Price per Km:")
print(sorted_avg_fare.head(3))

print("\nBottom 3 Cities by Price per Km:")
print(sorted_avg_fare.tail(3))
```

```
Top 3 Cities by Price per Km:
  city_name  average_fare_per_trip  average_trip_distance  price_per_km
3    Jaipur             483.918128              30.023125     16.118180
6    Mysore             249.707168              16.496921     15.136593
4     Kochi             335.245079              24.065461     13.930549

Bottom 3 Cities by Price per Km:
  city_name  average_fare_per_trip  average_trip_distance  price_per_km
2    Indore             179.838609              16.502473     10.897676
7     Surat             117.272925              10.997247     10.663844
8  Vadodara             118.566165              11.517736     10.294225
```

Average Passenger and Driver Ratings for Each City, Segmented by Passenger Type

```
# Calculate average passenger and driver ratings, segmented by passenger type (new vs repeat)
avg_ratings_by_city = merged_data.groupby(['city_name', 'passenger_type']).agg(
    average_passenger_rating=('passenger_rating', 'mean'),
    average_driver_rating=('driver_rating', 'mean')
).reset_index()

# Identify cities with the highest and lowest ratings
# For average passenger rating
top_passenger_ratings = avg_ratings_by_city.sort_values(by='average_passenger_rating', ascending=False).head(3)
bottom_passenger_ratings = avg_ratings_by_city.sort_values(by='average_passenger_rating', ascending=True).head(3)

# For average driver rating
top_driver_ratings = avg_ratings_by_city.sort_values(by='average_driver_rating', ascending=False).head(3)
bottom_driver_ratings = avg_ratings_by_city.sort_values(by='average_driver_rating', ascending=True).head(3)

# Display results
print("Top 3 Cities by Average Passenger Rating:")
print(top_passenger_ratings[['city_name', 'passenger_type', 'average_passenger_rating']])

print("\nBottom 3 Cities by Average Passenger Rating:")
print(bottom_passenger_ratings[['city_name', 'passenger_type', 'average_passenger_rating']])

print("\nTop 3 Cities by Average Driver Rating:")
print(top_driver_ratings[['city_name', 'passenger_type', 'average_driver_rating']])

print("\nBottom 3 Cities by Average Driver Rating:")
print(bottom_driver_ratings[['city_name', 'passenger_type', 'average_driver_rating']])
```

```
Top 3 Cities by Average Passenger Rating:
   city_name passenger_type  average_passenger_rating
8      Kochi            new                  8.987394
6     Jaipur            new                  8.985018
12    Mysore            new                  8.982964

Bottom 3 Cities by Average Passenger Rating:
   city_name passenger_type  average_passenger_rating
17  Vadodara       repeated                  5.978629
11   Lucknow       repeated                  5.985741
15     Surat       repeated                  5.995511

Top 3 Cities by Average Driver Rating:
```

```
         city_name passenger_type  average_driver_rating
19  Visakhapatnam       repeated               8.992701
9           Kochi       repeated               8.989830
6          Jaipur            new               8.988246

Bottom 3 Cities by Average Driver Rating:
   city_name passenger_type  average_driver_rating
15     Surat       repeated               6.479441
17  Vadodara       repeated               6.481072
11   Lucknow       repeated               6.491663
```

Month with the Highest Total Trips (peak demand) and Lowest Total Trips (low demand) for Each City

```
# Merge the datasets
merged_data = pd.merge(merged_data, dim_date, on='date')

# Calculate total trips by city and month
total_trips_by_city_month = merged_data.groupby(['city_name', 'month_name']).agg(
    total_trips=('trip_id', 'count')
).reset_index()

# Identify peak (highest total trips) and low demand (lowest total trips) months for each city
peak_months = total_trips_by_city_month.loc[total_trips_by_city_month.groupby('city_name')['total_trips'].idxmax()]
low_months = total_trips_by_city_month.loc[total_trips_by_city_month.groupby('city_name')['total_trips'].idxmin()]

# Display the peak and low months for each city
print("Peak Demand Months (Highest Total Trips) for Each City:")
print(peak_months[['city_name', 'month_name', 'total_trips']])

print("\nLow Demand Months (Lowest Total Trips) for Each City:")
print(low_months[['city_name', 'month_name', 'total_trips']])
```

```
Peak Demand Months (Highest Total Trips) for Each City:
        city_name month_name  total_trips
1      Chandigarh   February         7387
10     Coimbatore      March         3680
17         Indore        May         7787
19         Jaipur   February        15872
29          Kochi        May        10014
31        Lucknow   February        12060
41         Mysore        May         3007
42          Surat      April         9831
48       Vadodara      April         5941
54  Visakhapatnam      April         4938

Low Demand Months (Lowest Total Trips) for Each City:
        city_name month_name  total_trips
0      Chandigarh      April         5566
9      Coimbatore       June         3158
15         Indore       June         6288
21         Jaipur       June         9842
27          Kochi       June         6399
35        Lucknow        May         9705
38         Mysore    January         2485
44          Surat    January         8358
51       Vadodara       June         4685
56  Visakhapatnam    January         4468
```

Repeat Passenger Rate (RPR%)

```
# Calculate RPR% for each city and month
fact_passenger_summary['RPR%'] = (fact_passenger_summary['repeat_passengers'] / fact_passenger_summary['total_passengers']) * 100

# Aggregate RPR% over the six-month period for each city
city_rpr = fact_passenger_summary.groupby('city_id').agg(
    avg_rpr=('RPR%', 'mean')
).reset_index()

# Add city names
city_rpr = pd.merge(city_rpr, dim_city, on='city_id')

# Identify top 2 and bottom 2 cities by average RPR%
top_2_cities = city_rpr.nlargest(2, 'avg_rpr')
bottom_2_cities = city_rpr.nsmallest(2, 'avg_rpr')

# Display results
print("Top 2 Cities with Highest RPR%:")
print(top_2_cities)

print("\nBottom 2 Cities with Lowest RPR%:")
print(bottom_2_cities)
```

```
Top 2 Cities with Highest RPR%:
  city_id    avg_rpr city_name
2    GJ01  42.963123     Surat
9    UP01  38.131873   Lucknow

Bottom 2 Cities with Lowest RPR%:
  city_id    avg_rpr city_name
4    KA01  11.208195    Mysore
7    RJ01  18.329207    Jaipur
```

Repeat Passenger Rate (RPR%) by Month Across All Cities

```
# Calculate RPR% for each city and month
fact_passenger_summary['RPR%'] = (fact_passenger_summary['repeat_passengers'] / fact_passenger_summary['total_passengers']) * 100

# Aggregate RPR% by month across all cities
monthly_rpr = fact_passenger_summary.groupby('month').agg(
    avg_rpr=('RPR%', 'mean')
).reset_index()

# Identify the months with the highest and lowest RPR%
highest_rpr_month = monthly_rpr.nlargest(1, 'avg_rpr')
lowest_rpr_month = monthly_rpr.nsmallest(1, 'avg_rpr')

# Display results
print("Month with Highest RPR%:")
print(highest_rpr_month)

print("\nMonth with Lowest RPR%:")
print(lowest_rpr_month)
```

```
Month with Highest RPR%:
        month   avg_rpr
4  01-05-2024  34.21841

Month with Lowest RPR%:
        month    avg_rpr
0  01-01-2024  19.718611
```

Impact of Tourism Seasons and Local Events on Demand Patterns

```
# Merge to add event and day_type info
trip_data = pd.merge(fact_trips, dim_date, on='date')

# Aggregate trips by event type and city
event_analysis = trip_data.groupby(['city_id', 'day_type', 'month_name']).agg(
    total_trips=('trip_id', 'count'),
    avg_fare=('fare_amount', 'mean')
).reset_index()

# Compare trips during event days vs regular days
event_vs_regular = trip_data.groupby(['day_type']).agg(
    total_trips=('trip_id', 'count'),
    avg_fare=('fare_amount', 'mean')
)

print(event_vs_regular)
```

```
          total_trips    avg_fare
day_type
Weekday        238338  199.009822
Weekend        187565  323.922310
```

Identifying High-Traffic Zones

```
# Identify top pick-up/drop-off locations by city
location_analysis = fact_trips.groupby(['city_id']).agg(
    total_trips=('trip_id', 'count'),
    avg_fare=('fare_amount', 'mean')
).reset_index()
```

```
# Filter high-traffic zones
high_traffic = location_analysis[location_analysis['total_trips'] > location_analysis['total_trips'].mean()]

print(high_traffic)
```

```
  city_id  total_trips    avg_fare
2    GJ01        54843  117.272925
5    KL01        50702  335.245079
7    RJ01        76888  483.918128
9    UP01        64299  147.180376
```

Start coding or generate with AI.