In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Top and Bottom Performing Cities based on number of trips

In [2]:
```
fact_trips = pd.read_csv("fact_trips.csv")
dim_city = pd.read_csv("dim_city.csv")
```

In [3]:
```
fact_trips.head()
```

Out[3]:

	trip_id	date	city_id	passenger_type	distance_travelled(km)	fare_amount	passenger_rating	driver_rating
0	TRPLUC240113d55de2fb	2024-01-13	UP01	repeated	11	158	5	5
1	TRPVAD240129a3b6dba8	2024-01-29	GJ02	repeated	7	74	5	5
2	TRPCOI240107a42430fb	2024-01-07	TN01	repeated	11	155	8	8
3	TRPKOC240325d7601389	2024-03-25	KL01	repeated	36	427	9	10
4	TRPVIS2406027be97166	2024-06-02	AP01	new	17	265	8	8

In [4]:
```
fact_trips.shape
```

Out[4]:
```
(425903, 8)
```

In [5]:
```
fact_trips.isnull().sum()
```

Out[5]:
```
trip_id                  0
date                     0
city_id                  0
passenger_type           0
distance_travelled(km)   0
fare_amount              0
passenger_rating         0
driver_rating            0
dtype: int64
```

In [6]:
```
fact_trips.describe()
```

Out[6]:

	distance_travelled(km)	fare_amount	passenger_rating	driver_rating
count	425903.000000	425903.000000	425903.00000	425903.000000
mean	19.127172	254.020495	7.66104	7.830947
std	9.491735	159.638784	1.45533	1.442553
min	5.000000	58.000000	1.00000	1.000000
25%	12.000000	135.000000	7.00000	7.000000
50%	17.000000	199.000000	8.00000	8.000000
75%	25.000000	337.000000	9.00000	9.000000
max	45.000000	802.000000	10.00000	10.000000

In [7]:
```
# Group by city_id and count the number of trips:

citywise_trip_count = fact_trips.groupby("city_id", as_index = False).agg(total_trips = ("trip_id","count"))
citywise_trip_count
```

Out[7]:

	city_id	total_trips
0	AP01	28366
1	CH01	38981
2	GJ01	54843
3	GJ02	32026
4	KA01	16238
5	KL01	50702
6	MP01	42456
7	RJ01	76888
8	TN01	21104
9	UP01	64299

In [8]:
```
# Identifying top 3 and bottom 3 by total trips:
# Top 3:
```

```
top_3_cities = citywise_trip_count.nlargest(3, "total_trips")
top_3_cities
```

Out[8]:

	city_id	total_trips
7	RJ01	76888
9	UP01	64299
2	GJ01	54843

In [9]:
```
# Bottom 3:

bottom_3_cities = citywise_trip_count.nsmallest(3, "total_trips")
bottom_3_cities
```

Out[9]:

	city_id	total_trips
4	KA01	16238
8	TN01	21104
0	AP01	28366

In [10]:
```
# How to get the city name from the city_id:
# Join function:

citywise_trip_count = citywise_trip_count.merge(dim_city, on= "city_id", how = "left")
citywise_trip_count
```

Out[10]:

	city_id	total_trips	city_name
0	AP01	28366	Visakhapatnam
1	CH01	38981	Chandigarh
2	GJ01	54843	Surat
3	GJ02	32026	Vadodara
4	KA01	16238	Mysore
5	KL01	50702	Kochi
6	MP01	42456	Indore
7	RJ01	76888	Jaipur
8	TN01	21104	Coimbatore
9	UP01	64299	Lucknow

In [11]:
```
# Top 3 cities by total_trips:

top_3_cities = citywise_trip_count.nlargest(3, "total_trips")
top_3_cities
```

Out[11]:

	city_id	total_trips	city_name
7	RJ01	76888	Jaipur
9	UP01	64299	Lucknow
2	GJ01	54843	Surat

In [12]:
```
# Reorder columns to place city_id and city_name first
citywise_trip_count = citywise_trip_count[["city_id", "city_name", "total_trips"]]
```

In [13]:
```
citywise_trip_count
```

Out[13]:

	city_id	city_name	total_trips
0	AP01	Visakhapatnam	28366
1	CH01	Chandigarh	38981
2	GJ01	Surat	54843
3	GJ02	Vadodara	32026
4	KA01	Mysore	16238
5	KL01	Kochi	50702
6	MP01	Indore	42456
7	RJ01	Jaipur	76888
8	TN01	Coimbatore	21104
9	UP01	Lucknow	64299

In [14]:
```
# Top 3 cities:
```

```
top_3_cities = citywise_trip_count.nlargest(3, "total_trips")
top_3_cities
```

Out[14]:

	city_id	city_name	total_trips
7	RJ01	Jaipur	76888
9	UP01	Lucknow	64299
2	GJ01	Surat	54843

In [15]:
```
# Bottom 3 cities:

bottom_3_cities = citywise_trip_count.nsmallest(3, "total_trips")
bottom_3_cities
```

Out[15]:

	city_id	city_name	total_trips
4	KA01	Mysore	16238
8	TN01	Coimbatore	21104
0	AP01	Visakhapatnam	28366

Key takeaways:

1. first - Join function
2. second - group by- counts fn
3. reorder
4. result

Average Fare per trip by city

In [16]:
```
# Group by city_id to calculate total fare, total trips, and total distance:

city_metric = fact_trips.groupby("city_id", as_index= False).agg(total_fare = ("fare_amount", "sum"),
               total_trips = ("trip_id", "count"),
               total_distance = ("distance_travelled(km)", "sum"))

city_metric
```

Out[16]:

	city_id	total_fare	total_trips	total_distance
0	AP01	8018282	28366	639765
1	CH01	11058401	38981	916783
2	GJ01	6431599	54843	603122
3	GJ02	3797200	32026	368867
4	KA01	4054745	16238	267877
5	KL01	16997596	50702	1220167
6	MP01	7635228	42456	700629
7	RJ01	37207497	76888	2308418
8	TN01	3523992	21104	316121
9	UP01	9463551	64299	804571

In [17]:
```
# Calculate average fare per trip and average trip distance:

city_metric["average_fare_per_trip"] = city_metric["total_fare"]/city_metric["total_trips"]
city_metric["average_trip_distance"] = city_metric["total_distance"]/city_metric["total_trips"]
```

In [18]:
```
city_metric
```

Out[18]:

	city_id	total_fare	total_trips	total_distance	average_fare_per_trip	average_trip_distance
0	AP01	8018282	28366	639765	282.672284	22.553938
1	CH01	11058401	38981	916783	283.686950	23.518714
2	GJ01	6431599	54843	603122	117.272925	10.997247
3	GJ02	3797200	32026	368867	118.566165	11.517736
4	KA01	4054745	16238	267877	249.707168	16.496921
5	KL01	16997596	50702	1220167	335.245079	24.065461
6	MP01	7635228	42456	700629	179.838609	16.502473
7	RJ01	37207497	76888	2308418	483.918128	30.023125
8	TN01	3523992	21104	316121	166.982183	14.979198
9	UP01	9463551	64299	804571	147.180376	12.512963

In [19]:
```
# Merge with dim_city to get city names:

city_metric = city_metric.merge(dim_city, on = "city_id", how = "left")
```

In [20]:
```
# Reorder the columns

city_metric = city_metric[["city_id","city_name","average_fare_per_trip", "average_trip_distance"]]
```

In [21]:
```
city_metric
```

Out[21]:

	city_id	city_name	average_fare_per_trip	average_trip_distance
0	AP01	Visakhapatnam	282.672284	22.553938
1	CH01	Chandigarh	283.686950	23.518714
2	GJ01	Surat	117.272925	10.997247
3	GJ02	Vadodara	118.566165	11.517736
4	KA01	Mysore	249.707168	16.496921
5	KL01	Kochi	335.245079	24.065461
6	MP01	Indore	179.838609	16.502473
7	RJ01	Jaipur	483.918128	30.023125
8	TN01	Coimbatore	166.982183	14.979198
9	UP01	Lucknow	147.180376	12.512963

In [22]:
```
# Highest Average fare trip:

highest_avg_fare_city = city_metric.loc[city_metric["average_fare_per_trip"].idxmax()]
```

In [23]:
```
highest_avg_fare_city
```

Out[23]:
```
city_id                     RJ01
city_name                 Jaipur
average_fare_per_trip    483.918128
average_trip_distance     30.023125
Name: 7, dtype: object
```

In [24]:
```
# Smallest Average fare trip:

lowest_avg_per_trip = city_metric.loc[city_metric["average_fare_per_trip"].idxmin()]
lowest_avg_per_trip
```

Out[24]:
```
city_id                     GJ01
city_name                 Surat
average_fare_per_trip    117.272925
average_trip_distance     10.997247
Name: 2, dtype: object
```

Avearge rating by city and passenger type

In [25]:
```
## Calculate the average passenger and driver ratings for each city, segmented by passenger type.
## Identify the cities with highest and lowest average ratings
```

In [26]:
```
city_ratings = fact_trips.groupby(["city_id","passenger_type"], as_index = False).agg(avg_passenger_rating = ("
                        avg_driver_rating = ("driver_rating","mean"))

city_ratings
```

Out[26]:

	city_id	passenger_type	avg_passenger_rating	avg_driver_rating
0	AP01	new	8.976151	8.979995
1	AP01	repeated	7.989628	8.992701
2	CH01	new	8.489158	7.992120
3	CH01	repeated	7.493798	7.472824
4	GJ01	new	7.984173	6.994925
5	GJ01	repeated	5.995511	6.479441
6	GJ02	new	7.979263	7.004147
7	GJ02	repeated	5.978629	6.481072
8	KA01	new	8.982964	8.982878
9	KA01	repeated	7.978495	8.965767
10	KL01	new	8.987394	8.985350
11	KL01	repeated	8.003665	8.989830
12	MP01	new	8.485837	7.970800
13	MP01	repeated	7.473961	7.477404
14	RJ01	new	8.985018	8.988246
15	RJ01	repeated	7.991042	8.984790
16	TN01	new	8.485788	7.990604
17	TN01	repeated	7.475457	7.480778
18	UP01	new	7.977429	6.990406
19	UP01	repeated	5.985741	6.491663

In [27]:
```
# Merge city_name:

city_ratings = city_ratings.merge(dim_city, on = "city_id", how = "left")
city_ratings
```

Out[27]:

	city_id	passenger_type	avg_passenger_rating	avg_driver_rating	city_name
0	AP01	new	8.976151	8.979995	Visakhapatnam
1	AP01	repeated	7.989628	8.992701	Visakhapatnam
2	CH01	new	8.489158	7.992120	Chandigarh
3	CH01	repeated	7.493798	7.472824	Chandigarh
4	GJ01	new	7.984173	6.994925	Surat
5	GJ01	repeated	5.995511	6.479441	Surat
6	GJ02	new	7.979263	7.004147	Vadodara
7	GJ02	repeated	5.978629	6.481072	Vadodara
8	KA01	new	8.982964	8.982878	Mysore
9	KA01	repeated	7.978495	8.965767	Mysore
10	KL01	new	8.987394	8.985350	Kochi
11	KL01	repeated	8.003665	8.989830	Kochi
12	MP01	new	8.485837	7.970800	Indore
13	MP01	repeated	7.473961	7.477404	Indore
14	RJ01	new	8.985018	8.988246	Jaipur
15	RJ01	repeated	7.991042	8.984790	Jaipur
16	TN01	new	8.485788	7.990604	Coimbatore
17	TN01	repeated	7.475457	7.480778	Coimbatore
18	UP01	new	7.977429	6.990406	Lucknow
19	UP01	repeated	5.985741	6.491663	Lucknow

In [28]:
```
# reorder:

city_ratings = city_ratings[["city_id", "city_name", "passenger_type", "avg_passenger_rating", "avg_driver_rati
city_ratings
```

Out[28]:

	city_id	city_name	passenger_type	avg_passenger_rating	avg_driver_rating
0	AP01	Visakhapatnam	new	8.976151	8.979995
1	AP01	Visakhapatnam	repeated	7.989628	8.992701
2	CH01	Chandigarh	new	8.489158	7.992120
3	CH01	Chandigarh	repeated	7.493798	7.472824
4	GJ01	Surat	new	7.984173	6.994925
5	GJ01	Surat	repeated	5.995511	6.479441
6	GJ02	Vadodara	new	7.979263	7.004147
7	GJ02	Vadodara	repeated	5.978629	6.481072
8	KA01	Mysore	new	8.982964	8.982878
9	KA01	Mysore	repeated	7.978495	8.965767
10	KL01	Kochi	new	8.987394	8.985350
11	KL01	Kochi	repeated	8.003665	8.989830
12	MP01	Indore	new	8.485837	7.970800
13	MP01	Indore	repeated	7.473961	7.477404
14	RJ01	Jaipur	new	8.985018	8.988246
15	RJ01	Jaipur	repeated	7.991042	8.984790
16	TN01	Coimbatore	new	8.485788	7.990604
17	TN01	Coimbatore	repeated	7.475457	7.480778
18	UP01	Lucknow	new	7.977429	6.990406
19	UP01	Lucknow	repeated	5.985741	6.491663

In [29]:
```
# Identify cities with highest and lowest average passenger ratings

highest_avg_passenger_rating = city_ratings.loc[city_ratings["avg_passenger_rating"].idxmax()]
lowest_avg_passenger_rating = city_ratings.loc[city_ratings["avg_driver_rating"].idxmin()]

print("Highest average passenger rating:\n", highest_avg_passenger_rating)
print("Lowest average passenger rating:\n", lowest_avg_passenger_rating)
```

```
Highest average passenger rating:
 city_id                 KL01
city_name                Kochi
passenger_type             new
avg_passenger_rating    8.987394
avg_driver_rating        8.98535
Name: 10, dtype: object
Lowest average passenger rating:
 city_id                 GJ01
city_name                Surat
passenger_type          repeated
avg_passenger_rating    5.995511
avg_driver_rating       6.479441
Name: 5, dtype: object
```

In [30]:
```
# Identify cities with highest and lowest average driver rating:

highest_avg_driver_rating = city_ratings.loc[city_ratings["avg_driver_rating"].idxmax()]
lowest_avg_driver_rating = city_ratings.loc[city_ratings["avg_driver_rating"].idxmin()]

print("Highest average driver rating\n", highest_avg_driver_rating)
print("Lowest average driver rating\n", lowest_avg_driver_rating)
```

```
Highest average driver rating
 city_id                      AP01
city_name              Visakhapatnam
passenger_type              repeated
avg_passenger_rating        7.989628
avg_driver_rating           8.992701
Name: 1, dtype: object
Lowest average driver rating
 city_id                 GJ01
city_name                Surat
passenger_type          repeated
avg_passenger_rating    5.995511
avg_driver_rating       6.479441
Name: 5, dtype: object
```

Peak and Low Demand Months by city

In [31]:
```
## For each city, identify the month with the highest total trips (peak demand) and the month with the lowest t
## (low demand). This analysis will help Goodcabs understand seasonal patterns and adjust resources accordingly
```

In [32]:
```
dim_date = pd.read_csv("dim_date.csv")
```

```
dim_date
```

Out[32]:

	date	start_of_month	month_name	day_type
0	2024-01-01	2024-01-01	January	Weekday
1	2024-01-02	2024-01-01	January	Weekday
2	2024-01-03	2024-01-01	January	Weekday
3	2024-01-04	2024-01-01	January	Weekday
4	2024-01-05	2024-01-01	January	Weekday
...	...	...	...	...
177	2024-06-26	2024-06-01	June	Weekday
178	2024-06-27	2024-06-01	June	Weekday
179	2024-06-28	2024-06-01	June	Weekday
180	2024-06-29	2024-06-01	June	Weekend
181	2024-06-30	2024-06-01	June	Weekend

182 rows × 4 columns

In [33]:
```
fact_trips
```

Out[33]:

	trip_id	date	city_id	passenger_type	distance_travelled(km)	fare_amount	passenger_rating	driver_rating
0	TRPLUC240113d55de2fb	2024-01-13	UP01	repeated	11	158	5	5
1	TRPVAD240129a3b6dba8	2024-01-29	GJ02	repeated	7	74	5	5
2	TRPCOI240107a42430fb	2024-01-07	TN01	repeated	11	155	8	8
3	TRPKOC240325d7601389	2024-03-25	KL01	repeated	36	427	9	10
4	TRPVIS2406027be97166	2024-06-02	AP01	new	17	265	8	8
...	...	...	...	...	...	...	...	...
425898	TRPLUC2403180b02b4d0	2024-03-18	UP01	repeated	12	134	5	5
425899	TRPVAD2401032679e669	2024-01-03	GJ02	repeated	12	114	7	5
425900	TRPJAI24022578e10280	2024-02-25	RJ01	repeated	26	479	7	10
425901	TRPJAI240401e297ad20	2024-04-01	RJ01	repeated	27	361	7	10
425902	TRPJAI24021056f0e951	2024-02-10	RJ01	new	37	666	9	10

425903 rows × 8 columns

In [34]:
```
# Merge fact_trips with dim_date to get month information

fact_trips = fact_trips.merge(dim_date, on ="date", how = "left")
fact_trips
```

Out[34]:

	trip_id	date	city_id	passenger_type	distance_travelled(km)	fare_amount	passenger_rating	driver_rating	start_of
0	TRPLUC240113d55de2fb	2024-01-13	UP01	repeated	11	158	5	5	202
1	TRPVAD240129a3b6dba8	2024-01-29	GJ02	repeated	7	74	5	5	202
2	TRPCOI240107a42430fb	2024-01-07	TN01	repeated	11	155	8	8	202
3	TRPKOC240325d7601389	2024-03-25	KL01	repeated	36	427	9	10	202
4	TRPVIS2406027be97166	2024-06-02	AP01	new	17	265	8	8	202
...	...	...	...	...	...	...	...	...	
425898	TRPLUC2403180b02b4d0	2024-03-18	UP01	repeated	12	134	5	5	202
425899	TRPVAD2401032679e669	2024-01-03	GJ02	repeated	12	114	7	5	202
425900	TRPJAI24022578e10280	2024-02-25	RJ01	repeated	26	479	7	10	202
425901	TRPJAI240401e297ad20	2024-04-01	RJ01	repeated	27	361	7	10	202
425902	TRPJAI24021056f0e951	2024-02-10	RJ01	new	37	666	9	10	202

425903 rows × 11 columns

In [35]:
```
# Group by city_id and month to calculate total trips

city_month_trips = fact_trips.groupby(["city_id","month_name"], as_index = False).agg(total_trips = ("trip_id",
city_month_trips
```

Out[35]:

	city_id	month_name	total_trips
0	AP01	April	4938
1	AP01	February	4793
2	AP01	January	4468
3	AP01	June	4478
4	AP01	March	4877
5	AP01	May	4812
6	CH01	April	5566
7	CH01	February	7387
8	CH01	January	6810
9	CH01	June	6029
10	CH01	March	6569
11	CH01	May	6620
12	GJ01	April	9831
13	GJ01	February	9069
14	GJ01	January	8358
15	GJ01	June	8544
16	GJ01	March	9267
17	GJ01	May	9774
18	GJ02	April	5941
19	GJ02	February	5228
20	GJ02	January	4775
21	GJ02	June	4685
22	GJ02	March	5598
23	GJ02	May	5799
24	KA01	April	2603
25	KA01	February	2668
26	KA01	January	2485
27	KA01	June	2842
28	KA01	March	2633
29	KA01	May	3007
30	KL01	April	9762

31	KL01	February	7688
32	KL01	January	7344
33	KL01	June	6399
34	KL01	March	9495
35	KL01	May	10014
36	MP01	April	7415
37	MP01	February	7210
38	MP01	January	6737
39	MP01	June	6288
40	MP01	March	7019
41	MP01	May	7787
42	RJ01	April	11406
43	RJ01	February	15872
44	RJ01	January	14976
45	RJ01	June	9842
46	RJ01	March	13317
47	RJ01	May	11475
48	TN01	April	3661
49	TN01	February	3404
50	TN01	January	3651
51	TN01	June	3158
52	TN01	March	3680
53	TN01	May	3550
54	UP01	April	10212
55	UP01	February	12060
56	UP01	January	10858
57	UP01	June	10240
58	UP01	March	11224
59	UP01	May	9705

In [36]:
```
# Merge with dim_city to get city names:

city_month_trips = city_month_trips.merge(dim_city, on="city_id", how = "left")
city_month_trips
```

Out[36]:

	city_id	month_name	total_trips	city_name
0	AP01	April	4938	Visakhapatnam
1	AP01	February	4793	Visakhapatnam
2	AP01	January	4468	Visakhapatnam
3	AP01	June	4478	Visakhapatnam
4	AP01	March	4877	Visakhapatnam
5	AP01	May	4812	Visakhapatnam
6	CH01	April	5566	Chandigarh
7	CH01	February	7387	Chandigarh
8	CH01	January	6810	Chandigarh
9	CH01	June	6029	Chandigarh
10	CH01	March	6569	Chandigarh
11	CH01	May	6620	Chandigarh
12	GJ01	April	9831	Surat
13	GJ01	February	9069	Surat
14	GJ01	January	8358	Surat
15	GJ01	June	8544	Surat
16	GJ01	March	9267	Surat
17	GJ01	May	9774	Surat
18	GJ02	April	5941	Vadodara
19	GJ02	February	5228	Vadodara

20	GJ02	January	4775	Vadodara
21	GJ02	June	4685	Vadodara
22	GJ02	March	5598	Vadodara
23	GJ02	May	5799	Vadodara
24	KA01	April	2603	Mysore
25	KA01	February	2668	Mysore
26	KA01	January	2485	Mysore
27	KA01	June	2842	Mysore
28	KA01	March	2633	Mysore
29	KA01	May	3007	Mysore
30	KL01	April	9762	Kochi
31	KL01	February	7688	Kochi
32	KL01	January	7344	Kochi
33	KL01	June	6399	Kochi
34	KL01	March	9495	Kochi
35	KL01	May	10014	Kochi
36	MP01	April	7415	Indore
37	MP01	February	7210	Indore
38	MP01	January	6737	Indore
39	MP01	June	6288	Indore
40	MP01	March	7019	Indore
41	MP01	May	7787	Indore
42	RJ01	April	11406	Jaipur
43	RJ01	February	15872	Jaipur
44	RJ01	January	14976	Jaipur
45	RJ01	June	9842	Jaipur
46	RJ01	March	13317	Jaipur
47	RJ01	May	11475	Jaipur
48	TN01	April	3661	Coimbatore
49	TN01	February	3404	Coimbatore
50	TN01	January	3651	Coimbatore
51	TN01	June	3158	Coimbatore
52	TN01	March	3680	Coimbatore
53	TN01	May	3550	Coimbatore
54	UP01	April	10212	Lucknow
55	UP01	February	12060	Lucknow
56	UP01	January	10858	Lucknow
57	UP01	June	10240	Lucknow
58	UP01	March	11224	Lucknow
59	UP01	May	9705	Lucknow

In [37]:
```
# Identify the month with highest and lowest total trips for each city:

peak_demand_months = city_month_trips.loc[city_month_trips.groupby('city_id')['total_trips'].idxmax()]
low_demand_months = city_month_trips.loc[city_month_trips.groupby('city_id')['total_trips'].idxmin()]
```

In [38]:
```
# Reorder columns for clarity
peak_demand_months = peak_demand_months[["city_id", "city_name", "month_name", "total_trips"]]
low_demand_months = low_demand_months[["city_id", "city_name", "month_name", "total_trips"]]
```

In [39]:
```
# Display results

peak_demand_months
```

Out[39]:

	city_id	city_name	month_name	total_trips
0	AP01	Visakhapatnam	April	4938
7	CH01	Chandigarh	February	7387
12	GJ01	Surat	April	9831
18	GJ02	Vadodara	April	5941
29	KA01	Mysore	May	3007
35	KL01	Kochi	May	10014
41	MP01	Indore	May	7787
43	RJ01	Jaipur	February	15872
52	TN01	Coimbatore	March	3680
55	UP01	Lucknow	February	12060

In [40]:
```
low_demand_months
```

Out[40]:

	city_id	city_name	month_name	total_trips
2	AP01	Visakhapatnam	January	4468
6	CH01	Chandigarh	April	5566
14	GJ01	Surat	January	8358
21	GJ02	Vadodara	June	4685
26	KA01	Mysore	January	2485
33	KL01	Kochi	June	6399
39	MP01	Indore	June	6288
45	RJ01	Jaipur	June	9842
51	TN01	Coimbatore	June	3158
59	UP01	Lucknow	May	9705

Weekend vs Weekday Trip Demand City

In [41]:
```
## Compare the total trips taken on weekdays versus weekends for each city over the six month period.
## Identify cities with strong preference for either weekend or weekday trips to understand demand variations.
```

In [42]:
```
# Group by city_id, day_type to calculate total trips:

city_day_type_trips = fact_trips.groupby(["city_id", "day_type"], as_index=False).agg(
    total_trips=("trip_id", "count")  # Count of trips for each city, day_type, and month
)

city_day_type_trips
```

Out[42]:

	city_id	day_type	total_trips
0	AP01	Weekday	15100
1	AP01	Weekend	13266
2	CH01	Weekday	19914
3	CH01	Weekend	19067
4	GJ01	Weekday	37793
5	GJ01	Weekend	17050
6	GJ02	Weekday	20310
7	GJ02	Weekend	11716
8	KA01	Weekday	6424
9	KA01	Weekend	9814
10	KL01	Weekday	22915
11	KL01	Weekend	27787
12	MP01	Weekday	21198
13	MP01	Weekend	21258
14	RJ01	Weekday	32491
15	RJ01	Weekend	44397
16	TN01	Weekday	12576
17	TN01	Weekend	8528
18	UP01	Weekday	49617
19	UP01	Weekend	14682

In [43]:
```
# Join dim_city to get city_name:

city_day_type_trips = city_day_type_trips.merge(dim_city, on = "city_id", how = "left")
city_day_type_trips
```

Out[43]:

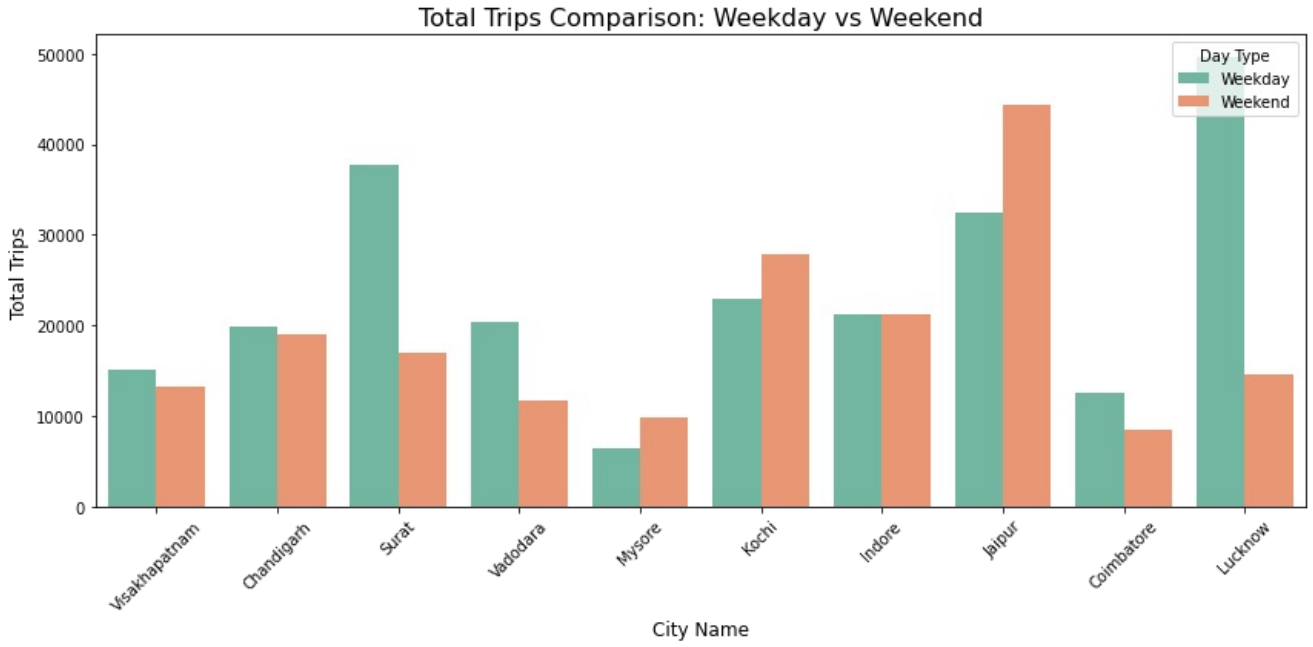
	city_id	day_type	total_trips	city_name
0	AP01	Weekday	15100	Visakhapatnam
1	AP01	Weekend	13266	Visakhapatnam
2	CH01	Weekday	19914	Chandigarh
3	CH01	Weekend	19067	Chandigarh
4	GJ01	Weekday	37793	Surat
5	GJ01	Weekend	17050	Surat
6	GJ02	Weekday	20310	Vadodara
7	GJ02	Weekend	11716	Vadodara
8	KA01	Weekday	6424	Mysore
9	KA01	Weekend	9814	Mysore
10	KL01	Weekday	22915	Kochi
11	KL01	Weekend	27787	Kochi
12	MP01	Weekday	21198	Indore
13	MP01	Weekend	21258	Indore
14	RJ01	Weekday	32491	Jaipur
15	RJ01	Weekend	44397	Jaipur
16	TN01	Weekday	12576	Coimbatore
17	TN01	Weekend	8528	Coimbatore
18	UP01	Weekday	49617	Lucknow
19	UP01	Weekend	14682	Lucknow

In [44]:
```
# Reorder:

city_day_type_trips = city_day_type_trips[["city_id","city_name","day_type","total_trips"]]
city_day_type_trips
```

Out[44]:

	city_id	city_name	day_type	total_trips
0	AP01	Visakhapatnam	Weekday	15100
1	AP01	Visakhapatnam	Weekend	13266
2	CH01	Chandigarh	Weekday	19914
3	CH01	Chandigarh	Weekend	19067
4	GJ01	Surat	Weekday	37793
5	GJ01	Surat	Weekend	17050
6	GJ02	Vadodara	Weekday	20310
7	GJ02	Vadodara	Weekend	11716
8	KA01	Mysore	Weekday	6424
9	KA01	Mysore	Weekend	9814
10	KL01	Kochi	Weekday	22915
11	KL01	Kochi	Weekend	27787
12	MP01	Indore	Weekday	21198
13	MP01	Indore	Weekend	21258
14	RJ01	Jaipur	Weekday	32491
15	RJ01	Jaipur	Weekend	44397
16	TN01	Coimbatore	Weekday	12576
17	TN01	Coimbatore	Weekend	8528
18	UP01	Lucknow	Weekday	49617
19	UP01	Lucknow	Weekend	14682

In [45]:
```
# # Create the bar plot :


plt.figure(figsize=(12, 6))
sns.barplot(data=city_day_type_trips, x="city_name", y="total_trips", hue="day_type", palette="Set2")

# Add titles and labels
plt.title("Total Trips Comparison: Weekday vs Weekend", fontsize=16)
plt.xlabel("City Name", fontsize=12)
plt.ylabel("Total Trips", fontsize=12)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability

# Display the legend
plt.legend(title="Day Type", loc="upper right")

# Show the plot
plt.tight_layout()
plt.show()
```



Repeat passenger frequency and city contribution analysis

In [46]:
```
## Analyse the frequency of trips taken by repeat passengers in each city (% of repeat passengers taking 2 trip
## Identify which cities contribute the most to higher trip frequencies among repeat passengers, and examine if
## distinguishable patterns between torism focused and business focused cities
```

In [47]:
```
dim_repeat_trip_distribution = pd.read_csv("dim_repeat_trip_distribution.csv")
dim_repeat_trip_distribution
```

Out[47]:

	month	city_id	trip_count	repeat_passenger_count
0	2024-01-01	AP01	10-Trips	7
1	2024-01-01	AP01	2-Trips	352
2	2024-01-01	AP01	3-Trips	158
3	2024-01-01	AP01	4-Trips	53
4	2024-01-01	AP01	5-Trips	38
...	...	...	...	...
535	2024-06-01	UP01	5-Trips	272
536	2024-06-01	UP01	6-Trips	272
537	2024-06-01	UP01	7-Trips	246
538	2024-06-01	UP01	8-Trips	83
539	2024-06-01	UP01	9-Trips	19

540 rows × 4 columns

In [48]:
```
# Merge with dim_city:

dim_repeat_trip_distribution = dim_repeat_trip_distribution.merge(dim_city, on = "city_id", how = "left")
dim_repeat_trip_distribution
```

Out[48]:

	month	city_id	trip_count	repeat_passenger_count	city_name
0	2024-01-01	AP01	10-Trips	7	Visakhapatnam
1	2024-01-01	AP01	2-Trips	352	Visakhapatnam
2	2024-01-01	AP01	3-Trips	158	Visakhapatnam
3	2024-01-01	AP01	4-Trips	53	Visakhapatnam
4	2024-01-01	AP01	5-Trips	38	Visakhapatnam
...	...	...	...	...	...
535	2024-06-01	UP01	5-Trips	272	Lucknow
536	2024-06-01	UP01	6-Trips	272	Lucknow
537	2024-06-01	UP01	7-Trips	246	Lucknow
538	2024-06-01	UP01	8-Trips	83	Lucknow
539	2024-06-01	UP01	9-Trips	19	Lucknow

540 rows × 5 columns

In [49]:
```
# Reorder:

dim_repeat_trip_distribution = dim_repeat_trip_distribution[["month","city_id","city_name","trip_count","repeat
dim_repeat_trip_distribution
```

Out[49]:

	month	city_id	city_name	trip_count	repeat_passenger_count
0	2024-01-01	AP01	Visakhapatnam	10-Trips	7
1	2024-01-01	AP01	Visakhapatnam	2-Trips	352
2	2024-01-01	AP01	Visakhapatnam	3-Trips	158
3	2024-01-01	AP01	Visakhapatnam	4-Trips	53
4	2024-01-01	AP01	Visakhapatnam	5-Trips	38
...	...	...	...	...	...
535	2024-06-01	UP01	Lucknow	5-Trips	272
536	2024-06-01	UP01	Lucknow	6-Trips	272
537	2024-06-01	UP01	Lucknow	7-Trips	246
538	2024-06-01	UP01	Lucknow	8-Trips	83
539	2024-06-01	UP01	Lucknow	9-Trips	19

540 rows × 5 columns

In [50]:
```
# Total repeat passangers:

total_repeat_passengers = dim_repeat_trip_distribution.groupby("city_id", as_index= False).agg(total_repeat_pas
total_repeat_passengers
```

Out[50]:

	city_id	total_repeat_passengers
0	AP01	5108
1	CH01	5070
2	GJ01	8638
3	GJ02	4346
4	KA01	1477
5	KL01	7626
6	MP01	7216
7	RJ01	9682
8	TN01	2551
9	UP01	9597

In [51]:
```
# Merge with dim_city:

total_repeat_passengers = total_repeat_passengers.merge(dim_city, on="city_id", how = "left")
total_repeat_passengers
```

Out[51]:

	city_id	total_repeat_passengers	city_name
0	AP01	5108	Visakhapatnam
1	CH01	5070	Chandigarh
2	GJ01	8638	Surat
3	GJ02	4346	Vadodara
4	KA01	1477	Mysore
5	KL01	7626	Kochi
6	MP01	7216	Indore
7	RJ01	9682	Jaipur
8	TN01	2551	Coimbatore
9	UP01	9597	Lucknow

In [52]:
```
# Reorder:

total_repeat_passengers = total_repeat_passengers[["city_id","city_name","total_repeat_passengers"]]
total_repeat_passengers
```

Out[52]:

	city_id	city_name	total_repeat_passengers
0	AP01	Visakhapatnam	5108
1	CH01	Chandigarh	5070
2	GJ01	Surat	8638
3	GJ02	Vadodara	4346
4	KA01	Mysore	1477
5	KL01	Kochi	7626
6	MP01	Indore	7216
7	RJ01	Jaipur	9682
8	TN01	Coimbatore	2551
9	UP01	Lucknow	9597

In [53]:
```
# Calculate the percentage of repeat passengers for each trip count in each city:

total_repeat_passengers["percentage_repeat_passengers"] = (dim_repeat_trip_distribution["repeat_passenger_count
total_repeat_passengers
```

```
C:\Users\Aditi\AppData\Local\Temp\ipykernel_29496\4186577568.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  total_repeat_passengers["percentage_repeat_passengers"] = (dim_repeat_trip_distribution["repeat_passenger_cou
nt"]/total_repeat_passengers["total_repeat_passengers"]*100)
```

Out[53]:

	city_id	city_name	total_repeat_passengers	percentage_repeat_passengers
0	AP01	Visakhapatnam	5108	0.137040
1	CH01	Chandigarh	5070	6.942801
2	GJ01	Surat	8638	1.829127
3	GJ02	Vadodara	4346	1.219512
4	KA01	Mysore	1477	2.572783
5	KL01	Kochi	7626	0.183582
6	MP01	Indore	7216	0.138581
7	RJ01	Jaipur	9682	0.113613
8	TN01	Coimbatore	2551	0.274402
9	UP01	Lucknow	9597	0.145879

In [54]:
```
# Sort the data by city and trip_count for visualization
dim_repeat_trip_distribution = dim_repeat_trip_distribution.sort_values(by=["city_name", "trip_count"])
dim_repeat_trip_distribution
```

Out[54]:

	month	city_id	city_name	trip_count	repeat_passenger_count
9	2024-01-01	CH01	Chandigarh	10-Trips	14
99	2024-02-01	CH01	Chandigarh	10-Trips	13
189	2024-03-01	CH01	Chandigarh	10-Trips	19
279	2024-04-01	CH01	Chandigarh	10-Trips	12
369	2024-05-01	CH01	Chandigarh	10-Trips	15
...	...	...	...	...	...
98	2024-02-01	AP01	Visakhapatnam	9-Trips	7
188	2024-03-01	AP01	Visakhapatnam	9-Trips	10
278	2024-04-01	AP01	Visakhapatnam	9-Trips	9
368	2024-05-01	AP01	Visakhapatnam	9-Trips	6
458	2024-06-01	AP01	Visakhapatnam	9-Trips	6

540 rows × 5 columns

In [55]:
```
# Stripping the trip count column:

dim_repeat_trip_distribution["trip_count"] = dim_repeat_trip_distribution["trip_count"].str.extract(r"(\d+)").a
dim_repeat_trip_distribution
```

Out[55]:

	month	city_id	city_name	trip_count	repeat_passenger_count
9	2024-01-01	CH01	Chandigarh	10	14
99	2024-02-01	CH01	Chandigarh	10	13
189	2024-03-01	CH01	Chandigarh	10	19
279	2024-04-01	CH01	Chandigarh	10	12
369	2024-05-01	CH01	Chandigarh	10	15
...	...	...	...	...	...
98	2024-02-01	AP01	Visakhapatnam	9	7
188	2024-03-01	AP01	Visakhapatnam	9	10
278	2024-04-01	AP01	Visakhapatnam	9	9
368	2024-05-01	AP01	Visakhapatnam	9	6
458	2024-06-01	AP01	Visakhapatnam	9	6

540 rows × 5 columns

In [56]:
```
# Filter cities with higher trip frequencies (e.g., 5+ trips)
high_trip_frequencies = dim_repeat_trip_distribution[dim_repeat_trip_distribution["trip_count"] >= 5]
high_trip_frequencies
```

Out[56]:

	month	city_id	city_name	trip_count	repeat_passenger_count
9	2024-01-01	CH01	Chandigarh	10	14
99	2024-02-01	CH01	Chandigarh	10	13
189	2024-03-01	CH01	Chandigarh	10	19
279	2024-04-01	CH01	Chandigarh	10	12
369	2024-05-01	CH01	Chandigarh	10	15
...	...	...	...	...	...
98	2024-02-01	AP01	Visakhapatnam	9	7
188	2024-03-01	AP01	Visakhapatnam	9	10
278	2024-04-01	AP01	Visakhapatnam	9	9
368	2024-05-01	AP01	Visakhapatnam	9	6
458	2024-06-01	AP01	Visakhapatnam	9	6

360 rows × 5 columns

In [57]:
```
# Identify cities contributing most to higher trip frequencies:

city_high_freq_contribution = high_trip_frequencies.groupby("city_name").agg(
                        total_high_freq_passengers = ("repeat_passenger_count","sum"))

city_high_freq_contribution
```

Out[57]:

city_name	total_high_freq_passengers
Chandigarh	1658
Coimbatore	1490
Indore	2134
Jaipur	1647
Kochi	1233
Lucknow	5698
Mysore	208
Surat	5133
Vadodara	2583
Visakhapatnam	705

In [58]:
```
# Percentage :

city_high_freq_contribution["high_frequency_percentage"] = (city_high_freq_contribution["total_high_freq_passen

city_high_freq_contribution
```

Out[58]:

city_name	total_high_freq_passengers	high_frequency_percentage
Chandigarh	1658	7.372493
Coimbatore	1490	6.625461
Indore	2134	9.489084
Jaipur	1647	7.323580
Kochi	1233	5.482680
Lucknow	5698	25.336831
Mysore	208	0.924897
Surat	5133	22.824492
Vadodara	2583	11.485615
Visakhapatnam	705	3.134866

In [59]:
```
# Sort cities by contribution to high frequencies
sorted_cities = city_high_freq_contribution.sort_values(by="high_frequency_percentage", ascending=False)
print("\nCities contributing most to higher trip frequencies:")
print(sorted_cities)
```

```
Cities contributing most to higher trip frequencies:
              total_high_freq_passengers  high_frequency_percentage
city_name
Lucknow                             5698                  25.336831
Surat                               5133                  22.824492
Vadodara                            2583                  11.485615
Indore                              2134                   9.489084
Chandigarh                          1658                   7.372493
Jaipur                              1647                   7.323580
Coimbatore                          1490                   6.625461
Kochi                               1233                   5.482680
Visakhapatnam                        705                   3.134866
Mysore                               208                   0.924897
```

Monthly Target achievement Analysis for Key Metrics

In [60]:
```
#For each city, evaluate monthly performance against targets for total trips, new passengers, and average passe
# targets db. Determine if each metric met, exceeded or missed the target, and calculate the percentage differe
#Identfy any consistent patterns in target achievement, particularly across tourism vs. business focused cities
```

In [61]:
```
import pandas as pd

# Load actual performance and target data
fact_passenger_summary = pd.read_csv("fact_passenger_summary.csv")
monthly_target_trips = pd.read_csv("monthly_target_trips.csv")
monthly_target_new_passengers = pd.read_csv("monthly_target_new_passengers.csv")
city_target_passenger_rating = pd.read_csv("city_target_passenger_rating.csv")
```

In [62]:
```
fact_passenger_summary.head(3)
```

Out[62]:

	month	city_id	new_passengers	repeat_passengers	total_passengers
0	2024-01-01	AP01	2513	650	3163
1	2024-01-01	CH01	3920	720	4640
2	2024-01-01	GJ01	2432	1184	3616

In [63]:
```
monthly_target_trips.head(3)
```

Out[63]:

	month	city_id	total_target_trips
0	2024-03-01	MP01	7000
1	2024-05-01	KA01	2500
2	2024-04-01	UP01	11000

In [64]:
```
monthly_target_new_passengers.head(3)
```

Out[64]:

	month	city_id	target_new_passengers
0	2024-05-01	GJ01	1500
1	2024-05-01	GJ02	1500
2	2024-03-01	GJ01	2000

In [65]:
```
city_target_passenger_rating.head(3)
```

Out[65]:

	city_id	target_avg_passenger_rating
0	CH01	8.00
1	UP01	7.25
2	AP01	8.50

In [66]:
```
# Merge targets with actual performance data:

performance_data = fact_passenger_summary.merge(monthly_target_trips, on=["city_id","month"], how = "left").mer
                   monthly_target_new_passengers, on = ["city_id","month"], how = "left").merge(
                   city_target_passenger_rating, on = "city_id", how = "left").merge(
                   dim_city, on = "city_id", how="left")
```

In [67]:
```
performance_data
```

Out[67]:

	month	city_id	new_passengers	repeat_passengers	total_passengers	total_target_trips	target_new_passengers	target_avg_passenger_ra
0	2024-01-01	AP01	2513	650	3163	4500	2500	8
1	2024-01-01	CH01	3920	720	4640	7000	4000	8
2	2024-01-01	GJ01	2432	1184	3616	9000	2000	7

3	2024-01-01	GJ02	2089	544	2633	6000	1800	7
4	2024-01-01	KA01	1957	172	2129	2000	2000	8
5	2024-01-01	KL01	4865	795	5660	7500	5000	8
6	2024-01-01	MP01	2843	1033	3876	7000	2700	8
7	2024-01-01	RJ01	10423	1422	11845	13000	12000	8
8	2024-01-01	TN01	1822	392	2214	3500	1500	8
9	2024-01-01	UP01	3465	1431	4896	13000	3200	7
10	2024-02-01	AP01	2380	790	3170	4500	2500	8
11	2024-02-01	CH01	4104	853	4957	7000	4000	8
12	2024-02-01	GJ01	2254	1313	3567	9000	2000	7
13	2024-02-01	GJ02	2146	610	2756	6000	1800	7
14	2024-02-01	KA01	2107	183	2290	2000	2000	8
15	2024-02-01	KL01	4367	1005	5372	7500	5000	8
16	2024-02-01	MP01	2878	1103	3981	7000	2700	8
17	2024-02-01	RJ01	10789	1661	12450	13000	12000	8
18	2024-02-01	TN01	1647	346	1993	3500	1500	8
19	2024-02-01	UP01	3529	1659	5188	13000	3200	7
20	2024-03-01	AP01	2170	923	3093	4500	2500	8
21	2024-03-01	CH01	3228	872	4100	7000	4000	8
22	2024-03-01	GJ01	1946	1494	3440	9000	2000	7
23	2024-03-01	GJ02	1763	759	2522	6000	1800	7
24	2024-03-01	KA01	1986	208	2194	2000	2000	8
25	2024-03-01	KL01	4865	1348	6213	7500	5000	8
26	2024-03-01	MP01	2742	1091	3833	7000	2700	8
27	2024-03-01	RJ01	7417	1840	9257	13000	12000	8
28	2024-03-01	TN01	1538	427	1965	3500	1500	8
29	2024-03-01	UP01	3159	1622	4781	13000	3200	7
30	2024-04-01	AP01	1845	992	2837	5000	2000	8
31	2024-04-01	CH01	2496	789	3285	6000	3000	8
32	2024-04-01	GJ01	1843	1551	3394	10000	1500	7
33	2024-04-01	GJ02	1637	862	2499	6500	1500	7
34	2024-04-01	KA01	1836	236	2072	2500	2000	8
35	2024-04-01	KL01	4939	1576	6515	9000	4000	8
36	2024-04-01	MP01	2351	1295	3646	7500	2000	8
37	2024-04-01	RJ01	6120	1736	7856	9500	6000	8
	2024-							

38	04-01	TN01	1242	480	1722	3500	1000	8
39	2024-04-01	UP01	2311	1496	3807	11000	2000	7
40	2024-05-01	AP01	1939	951	2890	5000	2000	8
41	2024-05-01	CH01	2730	969	3699	6000	3000	8
42	2024-05-01	GJ01	1611	1606	3217	10000	1500	7
43	2024-05-01	GJ02	1388	868	2256	6500	1500	7
44	2024-05-01	KA01	1921	349	2270	2500	2000	8
45	2024-05-01	KL01	4369	1853	6222	9000	4000	8
46	2024-05-01	MP01	2028	1563	3591	7500	2000	8
47	2024-05-01	RJ01	5332	1842	7174	9500	6000	8
48	2024-05-01	TN01	1039	504	1543	3500	1000	8
49	2024-05-01	UP01	1825	1662	3487	11000	2000	7
50	2024-06-01	AP01	1900	802	2702	5000	2000	8
51	2024-06-01	CH01	2430	867	3297	6000	3000	8
52	2024-06-01	GJ01	1540	1490	3030	10000	1500	7
53	2024-06-01	GJ02	1104	703	1807	6500	1500	7
54	2024-06-01	KA01	1874	329	2203	2500	2000	8
55	2024-06-01	KL01	3011	1049	4060	9000	4000	8
56	2024-06-01	MP01	2021	1131	3152	7500	2000	8
57	2024-06-01	RJ01	5775	1181	6956	9500	6000	8
58	2024-06-01	TN01	1226	402	1628	3500	1000	8
59	2024-06-01	UP01	1971	1727	3698	11000	2000	7

In [68]:
```
# Calculate performance metrics for total trips:

performance_data["total_trips_status"] = performance_data.apply(lambda row : "Exceeded" if row["total_passenger
                                        else "Met" if row["total_passengers"] == row["total_target_trips"]
                                        else "Missed" , axis = 1)

performance_data["total_trips_diff_percent"] = ((performance_data["total_passengers"] - performance_data["total
                                                performance_data["total_target_trips"]*100)


# Calculate performance metrics for new passengers:

performance_data["new_passangers_status"] = performance_data.apply(lambda row : "Exceeded" if row["new_passenge
                                                else "Met" if row["new_passengers"] == row["target_new_passenge

performance_data["new_passengers_diff_percent"] = (
    (performance_data["new_passengers"] - performance_data["target_new_passengers"])
    / performance_data["target_new_passengers"] * 100
)

# Calculate performance metrics for average passenger rating
performance_data["avg_rating_status"] = performance_data.apply(
    lambda row: "Exceeded" if row["total_passengers"] > row["target_avg_passenger_rating"] else
                "Met" if row["total_passengers"] == row["target_avg_passenger_rating"] else "Missed", axis=1
)
performance_data["avg_rating_diff_percent"] = (
    (performance_data["total_passengers"] - performance_data["target_avg_passenger_rating"])
    / performance_data["target_avg_passenger_rating"] * 100
)
```

In [69]:
```
performance_data
```

Out[69]:

	month	city_id	new_passengers	repeat_passengers	total_passengers	total_target_trips	target_new_passengers	target_avg_passenger_rat
0	2024-01-01	AP01	2513	650	3163	4500	2500	8
1	2024-01-01	CH01	3920	720	4640	7000	4000	8
2	2024-01-01	GJ01	2432	1184	3616	9000	2000	7
3	2024-01-01	GJ02	2089	544	2633	6000	1800	7
4	2024-01-01	KA01	1957	172	2129	2000	2000	8
5	2024-01-01	KL01	4865	795	5660	7500	5000	8
6	2024-01-01	MP01	2843	1033	3876	7000	2700	8
7	2024-01-01	RJ01	10423	1422	11845	13000	12000	8
8	2024-01-01	TN01	1822	392	2214	3500	1500	8
9	2024-01-01	UP01	3465	1431	4896	13000	3200	7
10	2024-02-01	AP01	2380	790	3170	4500	2500	8
11	2024-02-01	CH01	4104	853	4957	7000	4000	8
12	2024-02-01	GJ01	2254	1313	3567	9000	2000	7
13	2024-02-01	GJ02	2146	610	2756	6000	1800	7
14	2024-02-01	KA01	2107	183	2290	2000	2000	8
15	2024-02-01	KL01	4367	1005	5372	7500	5000	8
16	2024-02-01	MP01	2878	1103	3981	7000	2700	8
17	2024-02-01	RJ01	10789	1661	12450	13000	12000	8
18	2024-02-01	TN01	1647	346	1993	3500	1500	8
19	2024-02-01	UP01	3529	1659	5188	13000	3200	7
20	2024-03-01	AP01	2170	923	3093	4500	2500	8
21	2024-03-01	CH01	3228	872	4100	7000	4000	8
22	2024-03-01	GJ01	1946	1494	3440	9000	2000	7
23	2024-03-01	GJ02	1763	759	2522	6000	1800	7
24	2024-03-01	KA01	1986	208	2194	2000	2000	8
25	2024-03-01	KL01	4865	1348	6213	7500	5000	8
26	2024-03-01	MP01	2742	1091	3833	7000	2700	8
27	2024-03-01	RJ01	7417	1840	9257	13000	12000	8
28	2024-03-01	TN01	1538	427	1965	3500	1500	8
29	2024-03-01	UP01	3159	1622	4781	13000	3200	7
30	2024-04-01	AP01	1845	992	2837	5000	2000	8
31	2024-04-01	CH01	2496	789	3285	6000	3000	8
32	2024-04-01	GJ01	1843	1551	3394	10000	1500	7
33	2024-04-01	GJ02	1637	862	2499	6500	1500	7
34	2024-04-01	KA01	1836	236	2072	2500	2000	8

35	2024-04-01	KL01	4939	1576	6515	9000	4000	8
36	2024-04-01	MP01	2351	1295	3646	7500	2000	8
37	2024-04-01	RJ01	6120	1736	7856	9500	6000	8
38	2024-04-01	TN01	1242	480	1722	3500	1000	8
39	2024-04-01	UP01	2311	1496	3807	11000	2000	7
40	2024-05-01	AP01	1939	951	2890	5000	2000	8
41	2024-05-01	CH01	2730	969	3699	6000	3000	8
42	2024-05-01	GJ01	1611	1606	3217	10000	1500	7
43	2024-05-01	GJ02	1388	868	2256	6500	1500	7
44	2024-05-01	KA01	1921	349	2270	2500	2000	8
45	2024-05-01	KL01	4369	1853	6222	9000	4000	8
46	2024-05-01	MP01	2028	1563	3591	7500	2000	8
47	2024-05-01	RJ01	5332	1842	7174	9500	6000	8
48	2024-05-01	TN01	1039	504	1543	3500	1000	8
49	2024-05-01	UP01	1825	1662	3487	11000	2000	7
50	2024-06-01	AP01	1900	802	2702	5000	2000	8
51	2024-06-01	CH01	2430	867	3297	6000	3000	8
52	2024-06-01	GJ01	1540	1490	3030	10000	1500	7
53	2024-06-01	GJ02	1104	703	1807	6500	1500	7
54	2024-06-01	KA01	1874	329	2203	2500	2000	8
55	2024-06-01	KL01	3011	1049	4060	9000	4000	8
56	2024-06-01	MP01	2021	1131	3152	7500	2000	8
57	2024-06-01	RJ01	5775	1181	6956	9500	6000	8
58	2024-06-01	TN01	1226	402	1628	3500	1000	8
59	2024-06-01	UP01	1971	1727	3698	11000	2000	7

In [70]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the overall figure style
sns.set_theme(style="whitegrid")

# Create separate dataframes for each metric to visualize their statuses
metrics = ["total_trips_status", "new_passangers_status", "avg_rating_status"]
titles = ["Total Trips Performance", "New Passengers Performance", "Average Rating Performance"]

fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)

# Loop through each metric and create a bar plot
for i, metric in enumerate(metrics):
    # Count the occurrences of Missed, Met, Exceeded for each city
    status_counts = performance_data.groupby(["city_name", metric]).size().reset_index(name="count")

    # Pivot the data for easier plotting
    pivoted = status_counts.pivot(index="city_name", columns=metric, values="count").fillna(0)

    # Ensure the pivoted columns include only the relevant statuses dynamically
    available_columns = [col for col in ["Missed", "Met", "Exceeded"] if col in pivoted.columns]
    pivoted = pivoted[available_columns].sort_values(by=available_columns[-1], ascending=False)
```

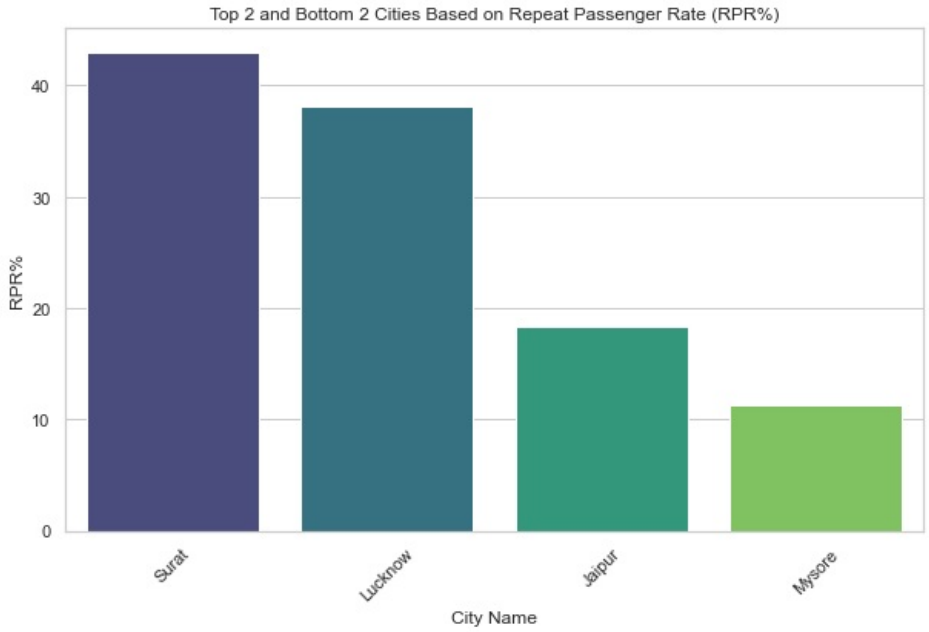
```
    # Plot the data
    pivoted.plot(kind="bar", stacked=True, ax=axes[i], color=["red", "gray", "green"][:len(available_columns)],
    axes[i].set_title(titles[i])
    axes[i].set_xlabel("City Name")
    axes[i].set_ylabel("Count")
    axes[i].legend(title="Status", loc="upper right")
    axes[i].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

Highest and Lowest Repeat Passenger Rate (RPR%) by City and Month

In [71]:

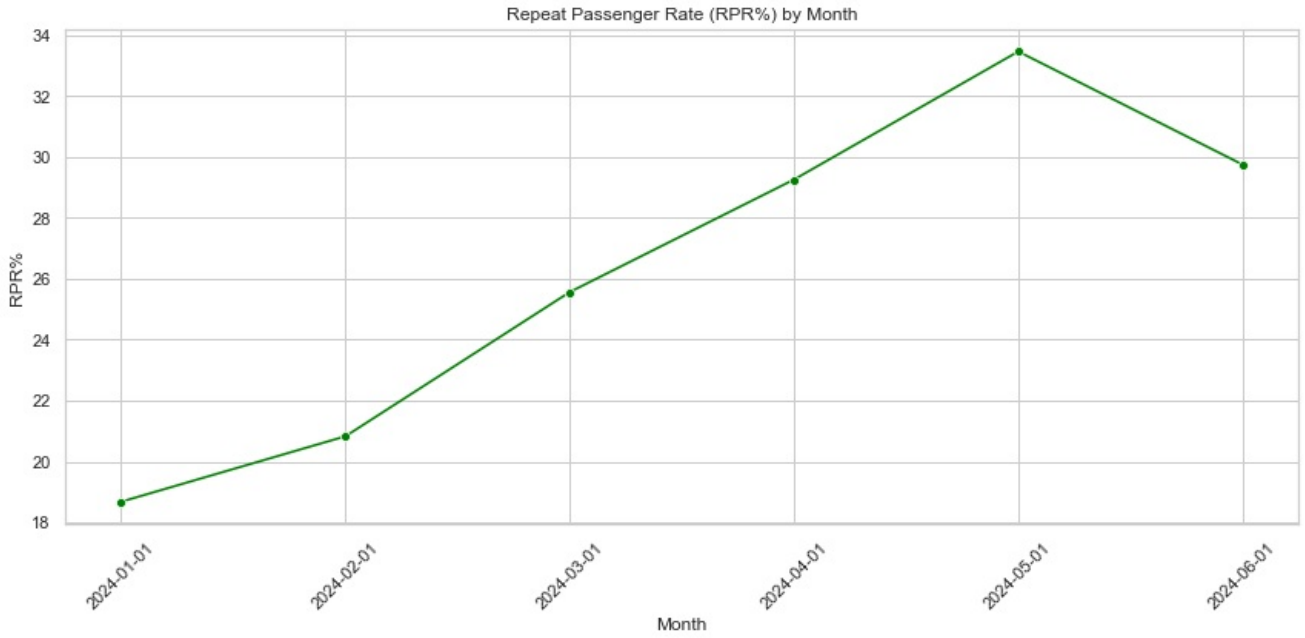
```
# Analyse the repeat passenger rate (RPR%) for each city across six months period. Identify top 2 and bottom 2
#based on their RPR% to determine which locations have the strongest and the weakest rates


# Calculate RPR% for each city:

performance_data["RPR%"] = (performance_data["repeat_passengers"]/performance_data["total_passengers"])*100

# Sort the data by RPR%
sorted_data = performance_data.groupby(["city_id","city_name"])["RPR%"].mean().reset_index()
sorted_data = sorted_data.sort_values(by = "RPR%", ascending = False)

# Identify the top 2 and bottom 2 cities
top_2_cities = sorted_data.head(2)
bottom_2_cities = sorted_data.tail(2)

# Combine the results
result = pd.concat([top_2_cities,bottom_2_cities]).reset_index(drop=True)

print("Top 2 and Bottom 2 Cities based on RPR%:")
print(result)

plt.figure(figsize=(10, 6))
sns.barplot(x="city_name", y="RPR%", data=result, palette="viridis")
plt.title("Top 2 and Bottom 2 Cities Based on Repeat Passenger Rate (RPR%)")
plt.xlabel("City Name")
plt.ylabel("RPR%")
plt.xticks(rotation=45)
plt.show()
```

```
Top 2 and Bottom 2 Cities based on RPR%:
  city_id city_name       RPR%
0    GJ01     Surat  42.963123
1    UP01   Lucknow  38.131873
2    RJ01    Jaipur  18.329207
3    KA01    Mysore  11.208195
```



In [72]:
```
# Analyse the RPR% by month across all cities and identify the months with the highest and lowest repeat passen
# This will help to pinpoint any seasonal patterns or months with higher repeat passenger loyalty.


# Calculate RPR% for each month
monthly_rpr = performance_data.groupby("month")[["repeat_passengers", "total_passengers"]].sum().reset_index()
monthly_rpr["RPR%"] = (monthly_rpr["repeat_passengers"] / monthly_rpr["total_passengers"]) * 100

# Identify months with highest and lowest RPR%
highest_rpr = monthly_rpr.loc[monthly_rpr["RPR%"].idxmax()]
lowest_rpr = monthly_rpr.loc[monthly_rpr["RPR%"].idxmin()]

print("Month with the Highest RPR%:")
print(highest_rpr)
print("\nMonth with the Lowest RPR%:")
print(lowest_rpr)

# Visualize RPR% by month
plt.figure(figsize=(12, 6))
sns.lineplot(data=monthly_rpr, x="month", y="RPR%", marker="o", color="green")
plt.title("Repeat Passenger Rate (RPR%) by Month")
plt.xlabel("Month")
plt.ylabel("RPR%")
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Month with the Highest RPR%:
month                2024-05-01
repeat_passengers         12167
total_passengers          36349
RPR%                  33.472723
Name: 4, dtype: object

Month with the Lowest RPR%:
month                2024-01-01
repeat_passengers          8343
total_passengers          44672
RPR%                  18.676128
Name: 0, dtype: object
```



In [ ]:

In [ ]:

In [ ]: