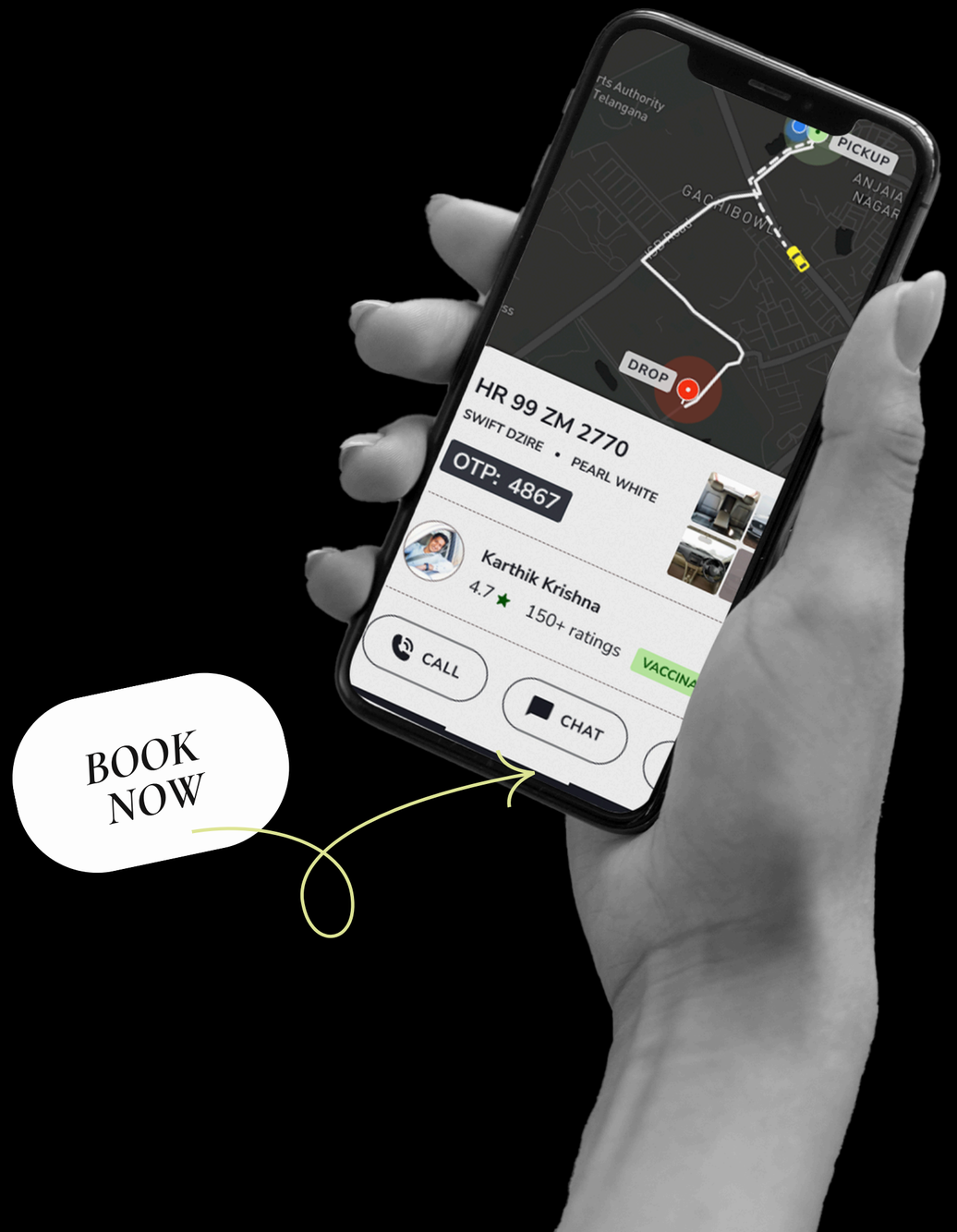
Uber

CAPSTONE PROJECT: UBER TRIPS ANALYTICS



PROJECT OVERVIEW

The project analyzes Uber green taxi trips in NYC from 2017 to 2020, examining patterns in pickups, drop-offs, trip distances, and revenue. The goal is to uncover trends, optimize operations, and support data-driven decisions for fleet management, pricing, and improving customer experience.

PART 1: ADVANCED SQL ANALYSIS – TAXI OPERATIONS INSIGHTS

Yearly Trip Trends: Calculate total trips per year and percentage change year-over-year.

```sql
WITH yearly_trips AS (
    SELECT
        YEAR(lpep_pickup_datetime) AS year,
        COUNT(*) AS total_trips
    FROM (
        SELECT lpep_pickup_datetime FROM 2017_trimmed
        UNION ALL
        SELECT lpep_pickup_datetime FROM 2018_trimmed
        UNION ALL
        SELECT lpep_pickup_datetime FROM 2019_trimmed
        UNION ALL
        SELECT lpep_pickup_datetime FROM 2020_trimmed
    ) t
    GROUP BY YEAR(lpep_pickup_datetime)
)
```

```sql
SELECT
    year,
    total_trips,
    ROUND(
        (total_trips - LAG(total_trips) OVER (ORDER BY year))
        / NULLIF(LAG(total_trips) OVER (ORDER BY year), 0) * 100,
        2
    ) AS percentage_change
FROM yearly_trips
WHERE year BETWEEN 2017 AND 2020
ORDER BY year;
```

year	total_trips	percentage_change
2017	100000	NULL
2018	99994	-0.01
2019	99997	0.00
2020	99998	0.00

Monthly Revenue Insights: Find monthly total revenue and average revenue per trip.

```sql
SELECT
    DATE_FORMAT(lpep_pickup_datetime, '%Y-%m') AS month,
    SUM(IFNULL(total_amount,0)) AS total_revenue,
    ROUND(AVG(IFNULL(total_amount,0)),2) AS avg_revenue_per_trip
FROM (
    SELECT lpep_pickup_datetime, total_amount FROM 2017_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime, total_amount FROM 2018_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime, total_amount FROM 2019_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime, total_amount FROM 2020_trimmed
) t
GROUP BY month
ORDER BY month;
```

month	total_revenue	avg_revenue_per_trip
2017-01	125343.88993702084	13.67
2017-02	121654.71995186806	13.87
2017-03	138186.66992497444	13.84
2017-04	129384.94998180866	14.03
2017-05	128265.8599050045	14.25
2017-06	123504.16987156868	14.75
2017-07	113452.92990589142	14.49
2017-08	107950.79000392556	14.55

Peak Day & Time Analysis: Identify the day of week and hour of day with the highest average trip volumes.

```sql
SELECT
    DAYNAME(lpep_pickup_datetime) AS day_of_week,
    HOUR(lpep_pickup_datetime) AS hour_of_day,
    COUNT(*) AS trip_count
FROM (
    SELECT lpep_pickup_datetime FROM 2017_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime FROM 2018_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime FROM 2019_trimmed
    UNION ALL
    SELECT lpep_pickup_datetime FROM 2020_trimmed
) t
GROUP BY day_of_week, hour_of_day
ORDER BY trip_count DESC
LIMIT 1;
```

day_of_week	hour_of_day	trip_count
Friday	18	4676

Borough Trip Distribution: Find the percentage of trips starting in each pickup_borough.

```sql
SELECT
    tz.Borough AS pickup_borough,
    COUNT(*) AS trip_count,
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS percentage_share
FROM (
    SELECT PULocationID FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID FROM 2020_trimmed
) trips
JOIN taxi_zones tz
    ON trips.PULocationID = tz.LocationID
GROUP BY tz.Borough
ORDER BY percentage_share DESC;
```

pickup_borough	trip_count	percentage_share
Manhattan	135913	33.98
Brooklyn	123234	30.81
Queens	112139	28.03
Bronx	27650	6.91
Unknown	843	0.21
Staten Island	212	0.05
EWR	9	0.00

Top Pickup-Dropoff Routes: Identify the top 10 most frequent pickup_zone to dropoff_zone combinations.

```sql
SELECT
    CONCAT(pu.Zone, ' → ', do.Zone) AS route,
    COUNT(*) AS trip_count
FROM (
    SELECT PULocationID, DOLocationID FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID FROM 2020_trimmed
) t
JOIN taxi_zones pu ON t.PULocationID = pu.LocationID
JOIN taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY route
ORDER BY trip_count DESC
LIMIT 10;
```

route	trip_count
East Harlem South → East Harlem North	4594
Astoria → Astoria	4545
East Harlem North → East Harlem South	3830
Central Harlem → Central Harlem North	3789
Forest Hills → Forest Hills	3358
East Harlem North → East Harlem North	3056
Elmhurst → Jackson Heights	3012
Central Harlem North → Central Harlem North	3001
Central Harlem → Central Harlem	2934
Central Harlem → East Harlem North	2809

Passenger Load Patterns: Find the average passenger_count for trips in each borough.

```sql
SELECT
    pu.Borough AS pickup_borough,
    ROUND(AVG(t.passenger_count), 2) AS avg_passenger_count
FROM (
    SELECT PULocationID, passenger_count FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID, passenger_count FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID, passenger_count FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID, passenger_count FROM 2020_trimmed
) t
JOIN taxi_zones pu ON t.PULocationID = pu.LocationID
GROUP BY pu.Borough
ORDER BY avg_passenger_count DESC;
```

pickup_borough	avg_passenger_count
Staten Island	1.47
Queens	1.4
Brooklyn	1.32
Manhattan	1.29
Bronx	1.28
Unknown	1.23
EWR	1.22

Trip Distance Distribution: Determine the average trip_distance for each pickup_borough and dropoff_borough.

```sql
SELECT
    pu.Borough AS pickup_borough,
    do.Borough AS dropoff_borough,
    ROUND(AVG(t.trip_distance), 2) AS avg_trip_distance
FROM (
    SELECT PULocationID, DOLocationID, trip_distance FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, trip_distance FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, trip_distance FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, trip_distance FROM 2020_trimmed
) t
JOIN taxi_zones pu ON t.PULocationID = pu.LocationID
JOIN taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY pu.Borough, do.Borough
ORDER BY avg_trip_distance DESC;
```

pickup_borough	dropoff_borough	avg_trip_distance
Unknown	Queens	764.2
EWR	Queens	37.12
Bronx	Staten Island	32.83
Staten Island	Bronx	32.46
Staten Island	Queens	28.64
Bronx	EWR	28.22
Queens	EWR	27.99
Queens	Staten Island	27.1
Manhattan	EWR	25.44

Payment Method Analysis: Calculate the percentage share of each payment_type and their average fare_amount.

```sql
SELECT
    t.payment_type,
    COUNT(*) AS trip_count,
    ROUND(COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (), 2) AS percentage_share,
    ROUND(AVG(t.fare_amount), 2) AS avg_fare
FROM (
    SELECT payment_type, fare_amount FROM 2017_trimmed
    UNION ALL
    SELECT payment_type, fare_amount FROM 2018_trimmed
    UNION ALL
    SELECT payment_type, fare_amount FROM 2019_trimmed
    UNION ALL
    SELECT payment_type, fare_amount FROM 2020_trimmed
) t
GROUP BY t.payment_type
ORDER BY percentage_share DESC;
```

payment_type	trip_count	percentage_share	avg_fare
1	199110	49.78	14.64
2	160690	40.17	10.72
NULL	37523	9.38	27.53
3	1856	0.46	2.67
4	798	0.20	4.1
5	23	0.01	7.09

High-Tip Routes: Find the top 5 pickup-dropoff combinations with the highest average tip_amount.

```sql
SELECT
    CONCAT(pu.Zone, ' → ', do.Zone) AS route,
    ROUND(AVG(t.tip_amount), 2) AS avg_tip
FROM (
    SELECT PULocationID, DOLocationID, tip_amount FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, tip_amount FROM 2020_trimmed
) t
JOIN taxi_zones pu ON t.PULocationID = pu.LocationID
JOIN taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY route
ORDER BY avg_tip DESC
LIMIT 5;
```

route	avg_tip
Laurelton → Bloomfield/Emerson Hill	150
NV → Brooklyn Heights	68.12
Steinway → NA	62
Jamaica → Marine Park/Floyd Bennett Field	36.16
Flushing Meadows-Corona Park → Newark Airport	33.51

Revenue by Borough Pairs: Calculate total_amount earned for each pickup_borough to dropoff_borough pair.

```sql
SELECT
    pu.Borough AS pickup_borough,
    do.Borough AS dropoff_borough,
    ROUND(SUM(t.total_amount), 2) AS total_revenue
FROM (
    SELECT PULocationID, DOLocationID, total_amount FROM 2017_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM 2018_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM 2019_trimmed
    UNION ALL
    SELECT PULocationID, DOLocationID, total_amount FROM 2020_trimmed
) t
JOIN taxi_zones pu ON t.PULocationID = pu.LocationID
JOIN taxi_zones do ON t.DOLocationID = do.LocationID
GROUP BY pu.Borough, do.Borough
ORDER BY total_revenue DESC;
```

pickup_borough	dropoff_borough	total_revenue
Manhattan	Manhattan	1501830.59
Brooklyn	Brooklyn	1449567.31
Queens	Queens	1291154.01
Brooklyn	Manhattan	517324.39
Bronx	Bronx	313641.04
Queens	Manhattan	255855.94
Brooklyn	Queens	237483.19
Manhattan	Bronx	208710.71
Queens	Brooklyn	178985.56

PART 2: INTERACTIVE DASHBOARDS – POWER BI

uber Trips Analytics | Trip & Revenue Overview

Filter Panel

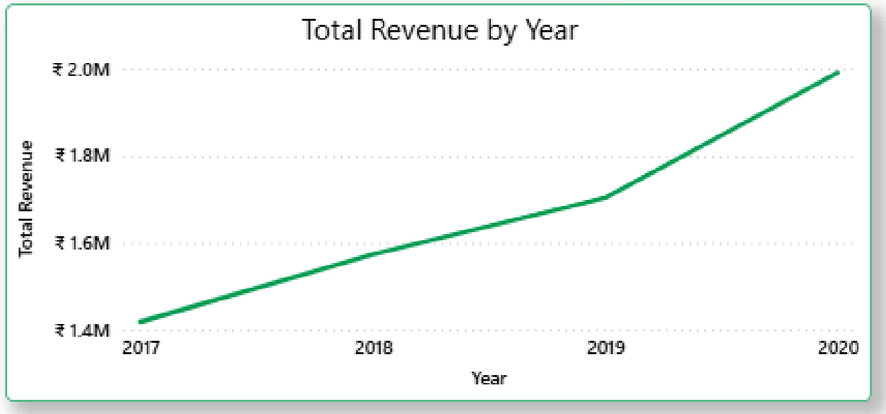
Zone
All

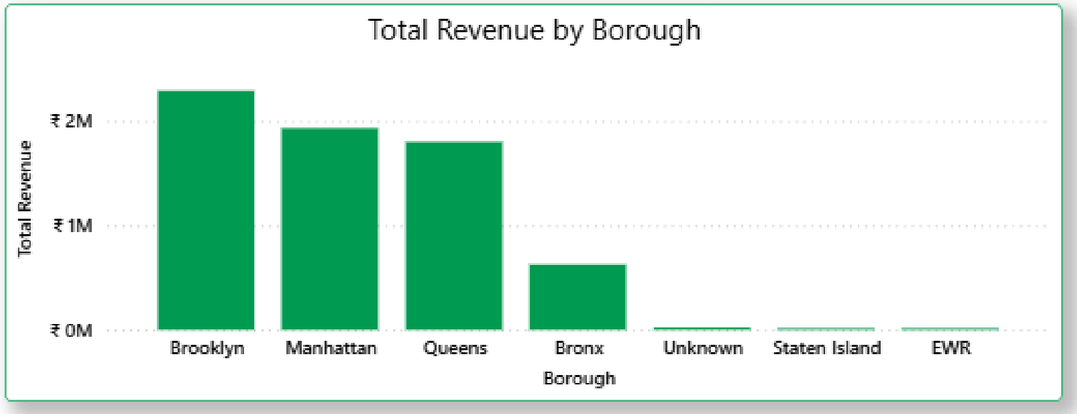
LocationID
All

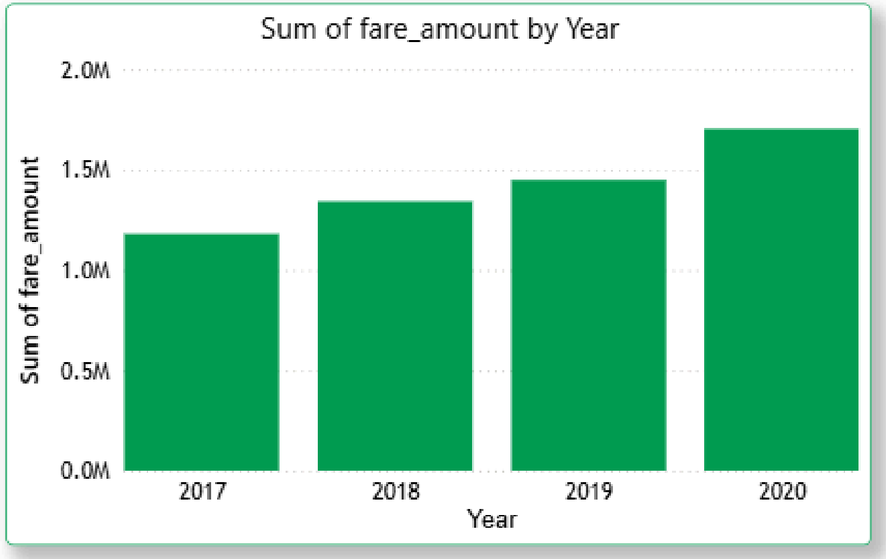
Year
All

MonthName
All

₹ 6.69M	₹ 16.72	75.00%	400K
Total Revenue	Avg Fare per Trip	YoY Growth %	Total Trips







MonthName	2017	2018	2019	2020	Total
Jan	₹ 1,24,950.44	₹ 1,27,201.30	₹ 1,69,648.16	₹ 4,78,201.29	₹ 9,00,001.19
Feb	₹ 1,21,389.22	₹ 1,23,065.36	₹ 1,57,315.27	₹ 4,22,828.91	₹ 8,24,598.76
Mar	₹ 1,37,769.97	₹ 1,41,626.58	₹ 1,64,189.56	₹ 2,12,282.00	₹ 6,55,868.11
Oct	₹ 1,14,509.82	₹ 1,34,962.49	₹ 1,46,416.33	₹ 1,25,900.08	₹ 5,21,788.72
Dec	₹ 1,07,476.15	₹ 1,27,536.54	₹ 1,40,937.68	₹ 1,15,541.89	₹ 4,91,492.26
Sep	₹ 1,10,510.88	₹ 1,25,367.46	₹ 1,37,687.76	₹ 1,15,903.05	₹ 4,89,469.15
Nov	₹ 1,03,544.37	₹ 1,26,591.44	₹ 1,35,991.41	₹ 1,17,054.75	₹ 4,83,181.97
May	₹ 1,27,880.21	₹ 1,46,847.45	₹ 1,30,152.57	₹ 72,120.12	₹ 4,77,000.35
Jul	₹ 1,12,473.58	₹ 1,24,712.69	₹ 1,36,234.43	₹ 1,00,385.97	₹ 4,73,806.67
Aug	₹ 1,06,705.04	₹ 1,22,370.98	₹ 1,35,143.03	₹ 1,09,258.81	₹ 4,73,477.86
Jun	₹ 1,22,709.17	₹ 1,35,533.65	₹ 1,22,636.74	₹ 81,831.03	₹ 4,62,710.59
Apr	₹ 1,29,002.30	₹ 1,37,691.71	₹ 1,27,680.08	₹ 39,627.20	₹ 4,34,001.29
Total	₹ 14,18,921.15	₹ 15,73,507.65	₹ 17,04,033.02	₹ 19,90,935.10	₹ 66,87,396.92

uber Trips Analytics | Time Based Demand Analysis

2017 | 2018 | 2019 | 2020

LocationID: All

PickupHour: All

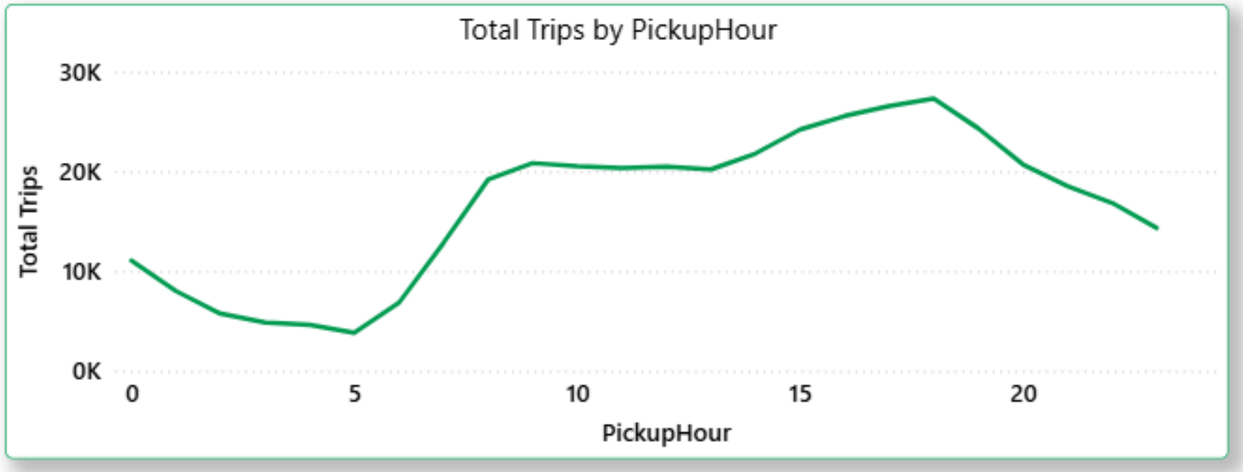
Trip DayName: All

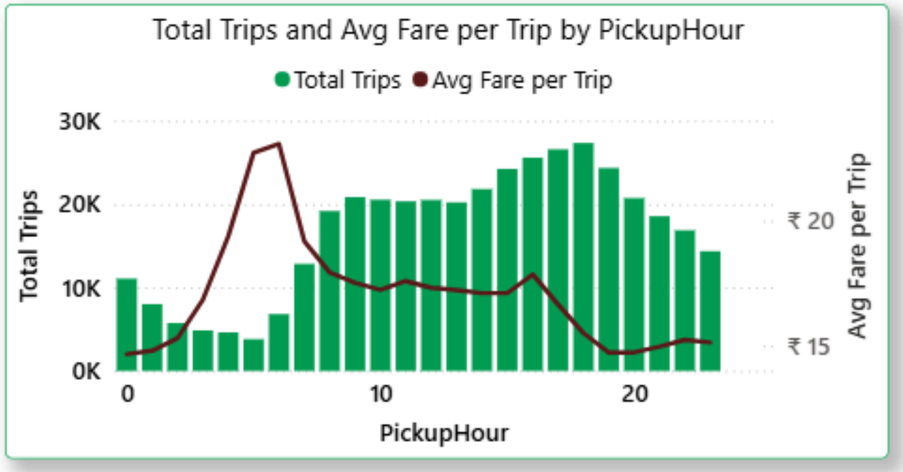
Borough: All

Friday
Peak Trip Day

6 PM
Peak Hour

43.92%
Weekend Trips %





Trip DayName	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	Total
Monday	1097	771	433	395	498	531	1086	2103	3096	3138	3033	2776	2800	2749	2892	3199	3557	3616	3665	3144	2528	2059	1684	1392	52242
Tuesday	907	584	368	316	395	512	1126	2194	3279	3326	3175	3048	3014	2895	3120	3485	3576	3935	3896	3290	2796	2338	2022	1578	55175
Wednesday	1025	652	420	364	475	486	1108	2197	3190	3466	3232	3120	3152	2874	3098	3538	3584	3948	4079	3742	3104	2555	2306	1791	57506
Thursday	1253	740	467	366	401	490	1091	2266	3364	3387	3114	3088	3130	2967	3331	3650	3837	4106	4242	3670	3110	2737	2527	2061	59395
Friday	1418	974	623	468	522	512	1143	2246	3313	3468	3150	3061	2859	2823	3185	3815	4148	4369	4676	4197	3551	3497	3261	2962	64241
Saturday	2602	2047	1637	1382	1080	652	663	1041	1665	2233	2592	2774	3006	3019	3296	3539	3798	3647	3774	3626	3315	3231	3109	3006	60734
Sunday	2772	2252	1801	1575	1256	630	605	795	1287	1840	2250	2488	2556	2897	2900	3012	3090	2974	3015	2681	2334	2148	1965	1573	50696
Total	11074	8020	5749	4866	4627	3813	6822	12842	19194	20858	20546	20355	20517	20224	21822	24238	25590	26595	27347	24350	20738	18565	16874	14363	399989

uber Trips Analytics | Geographic Pattern

Year: All

Zone: All

trip_type: All

Trip DayName: All

East Harlem South → East Harlem North

Top Route

Manhattan

Top Pickup Borough

Queens

Top Dropoff Borough

East Harlem North

Top Pickup Zone

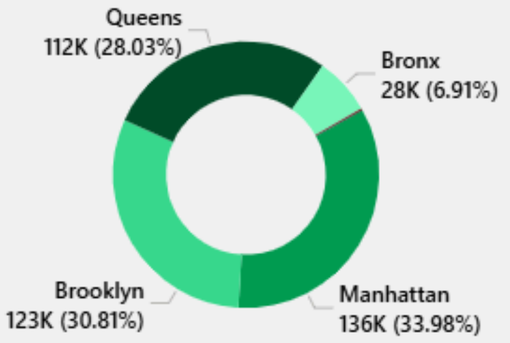
Astoria

Top Dropoff Zone

Top Routes

Route	Total Trips
Astoria → Astoria	16404
Central Harlem → Central Harlem	22424
Central Harlem North → Central Harlem North	14375
East Harlem North → East Harlem North	28588
East Harlem South → East Harlem South	24521
Elmhurst → Elmhurst	16104
Forest Hills → Forest Hills	11071
Fort Greene → Fort Greene	11484
Morningside Heights → Morningside Heights	14688
Washington Heights South → Washington Heights South	10111
Total	169770

Total Trips by Borough



Borough-to-Borough Matrix

Pickup Borough	Bronx	Brooklyn	EWR	Manhattan	Queens	Staten Island	Unknown	Total
Bronx	27648							27648
Brooklyn		123231						123231
EWR			9					9
Manhattan				135912				135912
Queens					112134			112134
Staten Island						212		212
Unknown							843	843
Total	27648	123231	9	135912	112134	212	843	399989

uber Trips Analytics | Passenger & Distance Insights

1.21
Avg Passengers per Trip

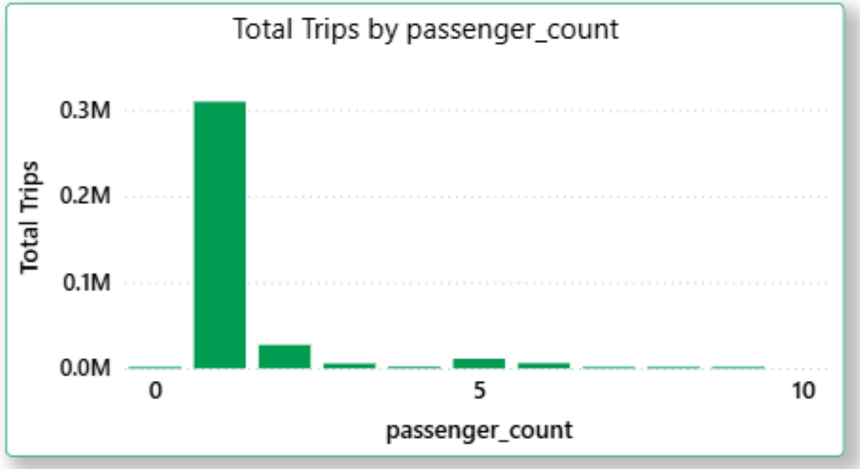
6.20 KM
Avg Trip Distance

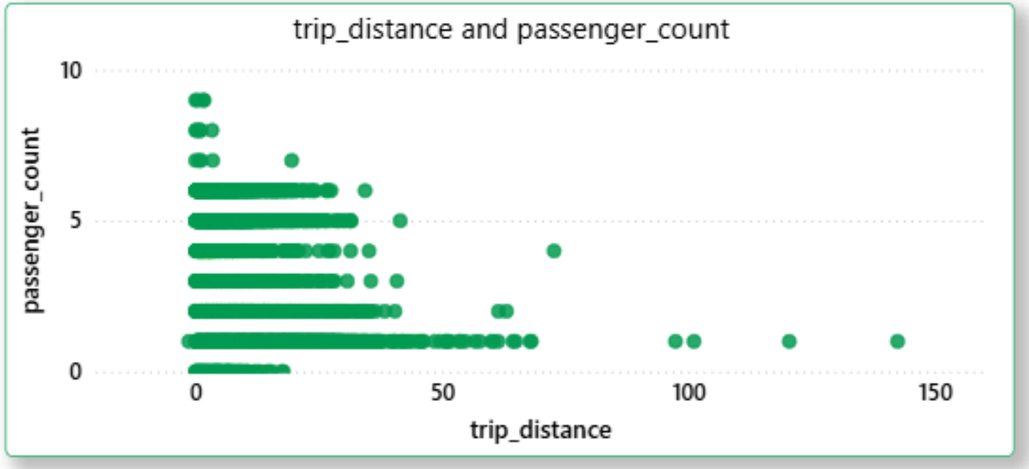
passenger_count
All

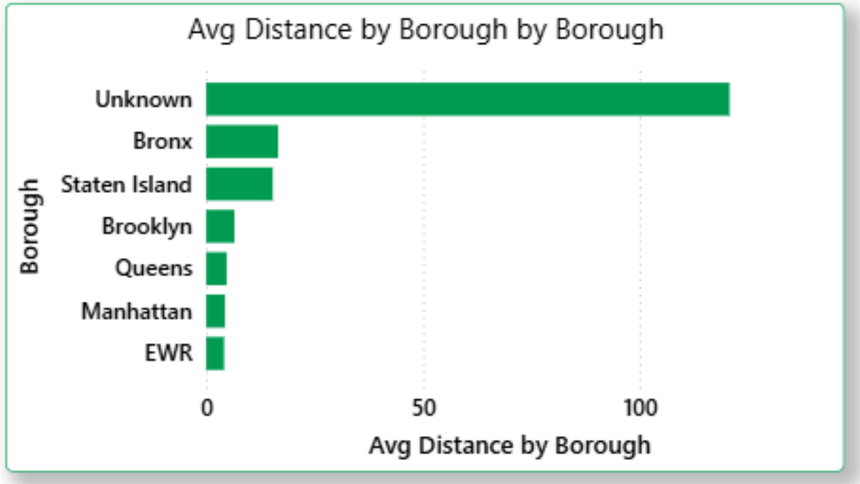
MonthName
All

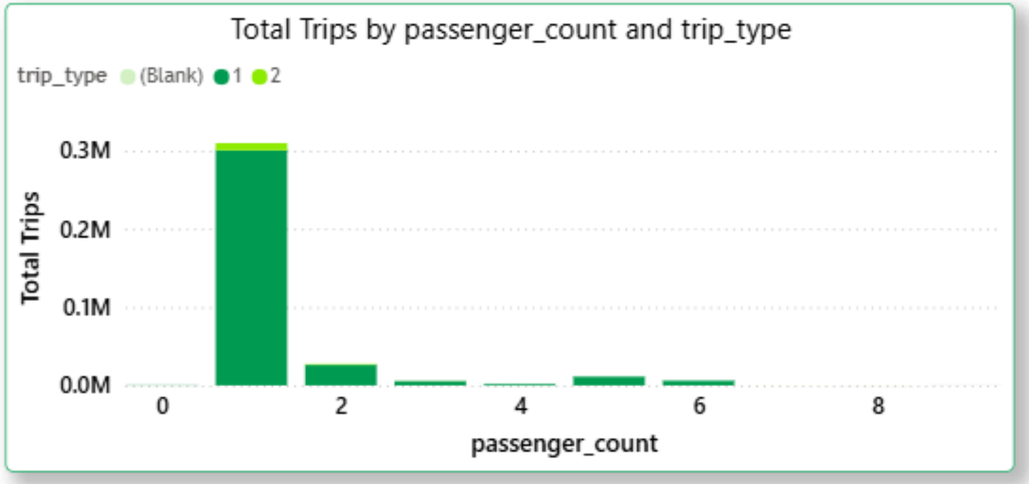
Year
All

Borough
All









uber Trips Analytics | Payment & Tips Insights

Borough: All

Year: All

Payment Type Name: All

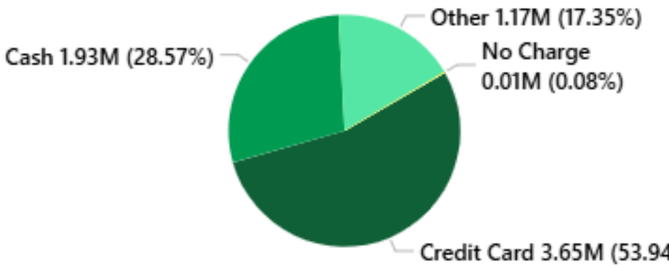
Zone: All

₹ 1.11	40.17%	₹ 442K	₹ 17.11
Avg Tip per Trip	% Cash Trips	Total Tips	Top Route Avg Tip

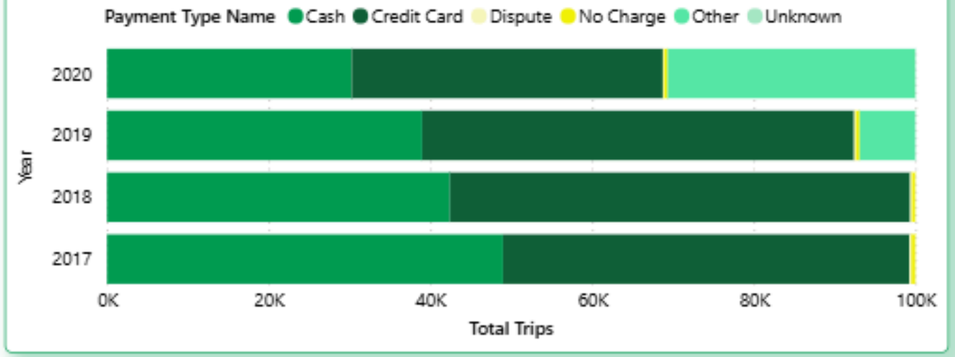
Top high tip Routes

Route	Total Tips
Central Harlem → Central Harlem	₹ 20,426
East Harlem North → East Harlem North	₹ 32,551
East Harlem South → East Harlem South	₹ 27,501
Morningside Heights → Morningside Heights	₹ 21,542
Washington Heights South → Washington Heights South	₹ 18,020
Total	**₹ 1,20,041**

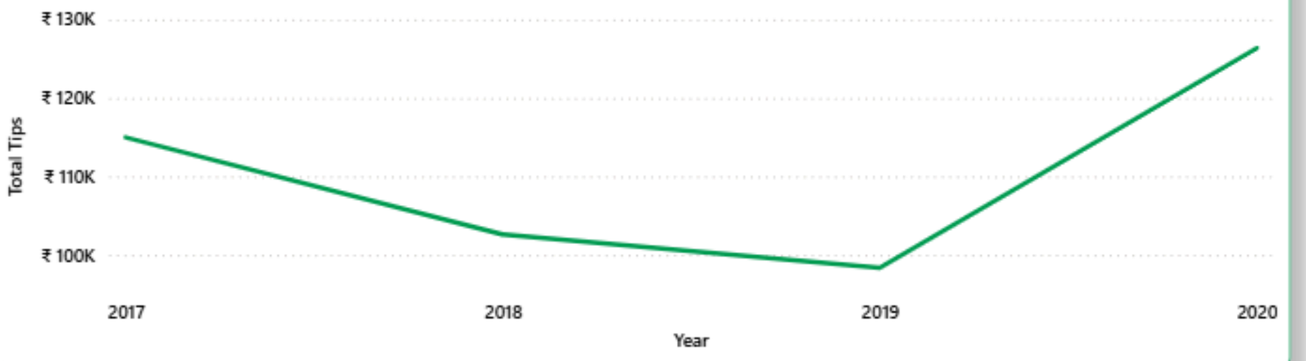
Sum of total_amount by Payment Type Name



Total Trips by Year and Payment Type Name



Total Tips by Year



SUMMARY

The data reveals a robust and expanding service characterized by predictable demand patterns and distinct customer behaviors. The key insights are as follows:

- Strong Financial Performance: The business demonstrates significant growth, having generated $6.69 million in revenue from 400,000 trips. This is underscored by a 75% year-over-year growth rate and a healthy Average Revenue Per Trip of $16.72.

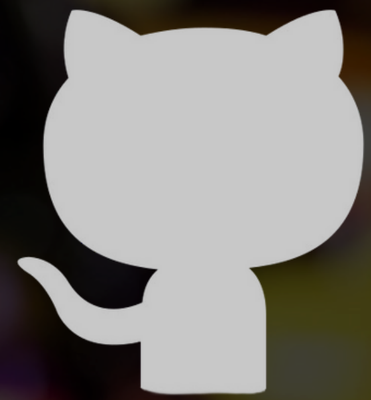
- Predictable Demand Peaks: Operational data identifies clear peak periods, with the highest demand occurring on Fridays at 6:00 PM. Furthermore, weekends account for 44% of all trip volume, indicating a primary driver of demand is social and leisure travel.

- Dominance of Local Travel: The most frequent trips are short-distance, intra-zone journeys (e.g., within East Harlem). Geographically, Manhattan is the primary demand hub for pickups, while Queens is the leading destination, highlighting key inter-borough traffic flow.

- Significant Cash Usage and Loyal Tippers: A considerable 40% of all transactions are cash-based, highlighting a vital payment preference. Analysis also identifies high-value customer segments on recurring local routes in areas like Harlem and Morningside Heights, who demonstrate strong loyalty through higher tipping.

Strategic Conclusion: The core business is on a strong growth trajectory. To capitalize on this momentum, strategic focus should be placed on ensuring operational reliability during predictable peak hours and optimizing service for high-frequency local routes, while maintaining support for diverse payment methods.

PART 3: MACHINE LEARNING – PREDICTIVE MODELING



PROJECT LINK

Uber Data Predictive Modeling (2017–2020 + T
Zones)

In [3]:
```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from prophet import Prophet
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score
import os

# Set folder path
folder = "trips"

# Read all yearly files automatically
files = [f for f in os.listdir(folder) if f.endswith('_trimmed.csv')]
df_list = [pd.read_csv(os.path.join(folder, f)) for f in files]
df = pd.concat(df_list, ignore_index=True)

# Read taxi zone file
zones = pd.read_csv(os.path.join(folder, 'taxi_zones.csv'))

print("✅ Data Loaded Successfully")
print("Total Records:", len(df))
df.head()
```

```
✅ Data Loaded Successfully
Total Records: 400000
```

Out[3]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PUL
0	2.0	2017-01-04 18:03:23.000	2017-01-04 18:10:41.000	N	1.0	
1	2.0	2017-02-21 14:36:40.000	2017-02-21 14:44:06.000	N	1.0	
2	2.0	2017-03-09 08:53:53.000	2017-03-09 08:59:02.000	N	1.0	
3	2.0	2017-12-05 20:15:50.000	2017-12-05 20:18:26.000	N	1.0	
4	2.0	2017-07-12 14:45:33.000	2017-07-12 14:50:52.000	N	1.0	

In [4]:
```
# Merge Pickup Boroughs
df = df.merge(zones[['LocationID', 'Borough']], how='left', left_on='PULocationI
df = df.rename(columns={'Borough': 'pickup_borough'}).drop('LocationID', axis=1)

# Merge Dropoff Boroughs
df = df.merge(zones[['LocationID', 'Borough']], how='left', left_on='DOLocationI
df = df.rename(columns={'Borough': 'dropoff_borough'}).drop('LocationID', axis=1

print("✅ Boroughs added successfully")
df[['PULocationID', 'pickup_borough', 'DOLocationID', 'dropoff_borough']].head()
```

```
✅ Boroughs added successfully
```

Out[4]:

	PULocationID	pickup_borough	DOLocationID	dropoff_borough
0	33	Brooklyn	52	Brooklyn
1	25	Brooklyn	97	Brooklyn
2	41	Manhattan	166	Manhattan
3	260	Queens	260	Queens
4	17	Brooklyn	17	Brooklyn

In [6]:
```
df['lpep_pickup_datetime'] = pd.to_datetime(df['lpep_pickup_datetime'], errors='

df['year'] = df['lpep_pickup_datetime'].dt.year
df['month'] = df['lpep_pickup_datetime'].dt.month
df['week'] = df['lpep_pickup_datetime'].dt.isocalendar().week
df['hour'] = df['lpep_pickup_datetime'].dt.hour
df['day_of_week'] = df['lpep_pickup_datetime'].dt.day_name()

# Drop null times
df = df.dropna(subset=['lpep_pickup_datetime'])
print("✅ Time features created")
```

```
✅ Time features created
```

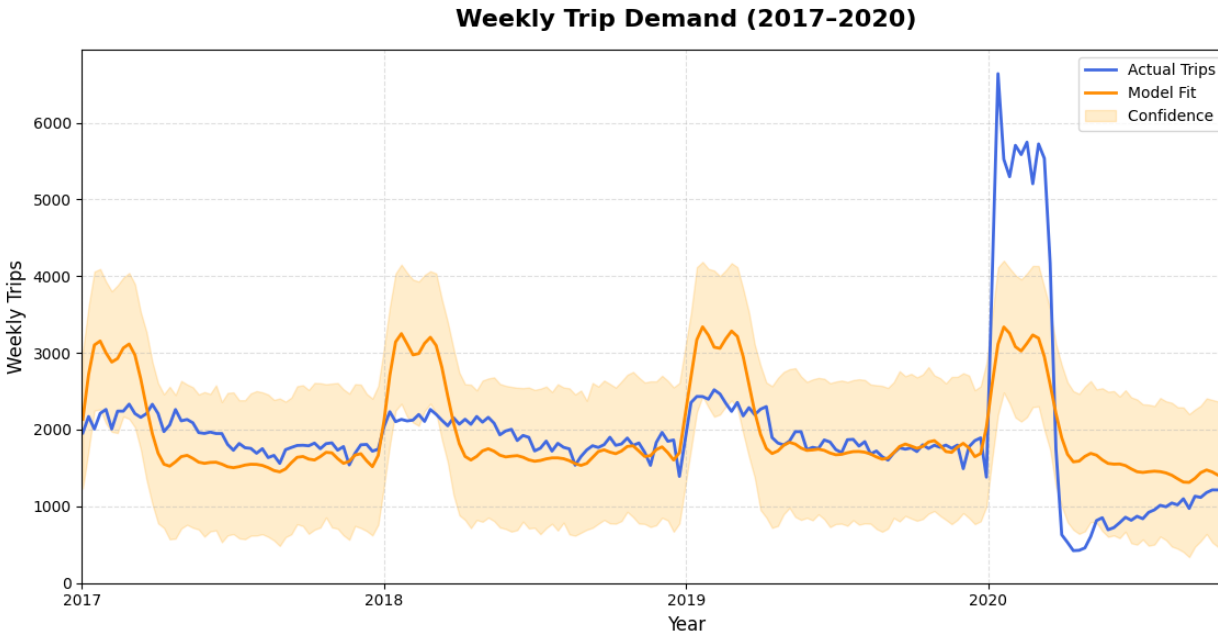
In [7]:
```
# Group by Year and ISO Week
weekly_trips = df.groupby(['year', 'week']).size().reset_index(name='Weekly_Trip

# Ensure year and week are integers
weekly_trips['year'] = weekly_trips['year'].astype(int)
weekly_trips['week'] = weekly_trips['week'].astype(int)

# Properly create 'week_start' date
weekly_trips['week_start'] = pd.to_datetime(
    weekly_trips['year'].astype(str) + '-' + weekly_trips['week'].astype(str) +
    format='%Y-%W-%w',
    errors='coerce'
)

# ✅ Filter only valid dates between 2017 and 2020
weekly_trips = weekly_trips[(weekly_trips['week_start'] >= '2017-01-01') & (week

# Prepare for Prophet
df_prophet = weekly_trips.rename(columns={'week_start': 'ds', 'Weekly_Trips': 'y
df_prophet = df_prophet.sort_values('ds')

print(df_prophet.head())
print(df_prophet.tail())
```

```
   year  week     y         ds
3  2017     1  1956 2017-01-02
4  2017     2  2170 2017-01-09
5  2017     3  2006 2017-01-16
6  2017     4  2211 2017-01-23
7  2017     5  2262 2017-01-30
     year  week     y         ds
206  2020    48  1057 2020-11-30
207  2020    49  1193 2020-12-07
208  2020    50  1249 2020-12-14
209  2020    51  1053 2020-12-21
210  2020    52   980 2020-12-28
```

Weekly Trip Demand Forecasting

In [11]:
```python
# --- Import libraries ---
from prophet import Prophet
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import pandas as pd

# --- Model training ---
model = Prophet()
model.fit(df_prophet)

# --- Forecast generation ---
future = model.make_future_dataframe(periods=12, freq='W')
forecast = model.predict(future)

# --- Filter only 2017–2020 range for display ---
forecast_filtered = forecast[
    (forecast['ds'] >= '2017-01-01') & (forecast['ds'] <= '2020-12-31')
]

# --- Plot ---
plt.figure(figsize=(12,6))

# Actual trips (blue)
plt.plot(df_prophet['ds'], df_prophet['y'], color='royalblue', label='Actual Tri

# Model fit (orange)
plt.plot(forecast_filtered['ds'], forecast_filtered['yhat'], color='darkorange',

# Confidence interval
plt.fill_between(
    forecast_filtered['ds'],
    forecast_filtered['yhat_lower'],
    forecast_filtered['yhat_upper'],
    color='orange',
    alpha=0.2,
    label='Confidence Interval'
)

# --- Formatting ---
plt.title("Weekly Trip Demand (2017–2020)", fontsize=16, fontweight='bold', pad=
plt.xlabel("Year", fontsize=12)
plt.ylabel("Weekly Trips", fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.4)

# Show year ticks
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

# ✅ Fix x-axis limit using datetime objects
plt.xlim(pd.Timestamp('2017-01-01'), pd.Timestamp('2020-12-31'))

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

```
11:14:55 - cmdstanpy - INFO - Chain [1] start processing
11:14:56 - cmdstanpy - INFO - Chain [1] done processing
```



Insight: The forecasting model successfully identifies recurring weekly demand cycles between 2017–2020. A moderate upward trend with seasonal variations is obse and future projections suggest steady trip volumes with minimal volatility.

High Tip Prediction Model

In [4]:
```
df.columns.tolist()
```

Out[4]:
```
['VendorID',
 'lpep_pickup_datetime',
 'lpep_dropoff_datetime',
 'store_and_fwd_flag',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'passenger_count',
 'trip_distance',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'payment_type',
 'trip_type',
 'congestion_surcharge']
```

In [5]:
```
# Convert pickup datetime if not already
df['lpep_pickup_datetime'] = pd.to_datetime(df['lpep_pickup_datetime'])

# Extract hour and day of week
df['hour'] = df['lpep_pickup_datetime'].dt.hour
df['day_of_week'] = df['lpep_pickup_datetime'].dt.day_name()

# Map boroughs using taxi_zones.csv
```

```
taxi_zones = pd.read_csv("trips/taxi_zones.csv")

# Merge pickup and dropoff boroughs
df = df.merge(taxi_zones[['LocationID', 'Borough']], left_on='PULocationID', rig
df = df.rename(columns={'Borough': 'pickup_borough'})
df = df.drop(columns=['LocationID'])

df = df.merge(taxi_zones[['LocationID', 'Borough']], left_on='DOLocationID', rig
df = df.rename(columns={'Borough': 'dropoff_borough'})
df = df.drop(columns=['LocationID'])
```

In [4]:

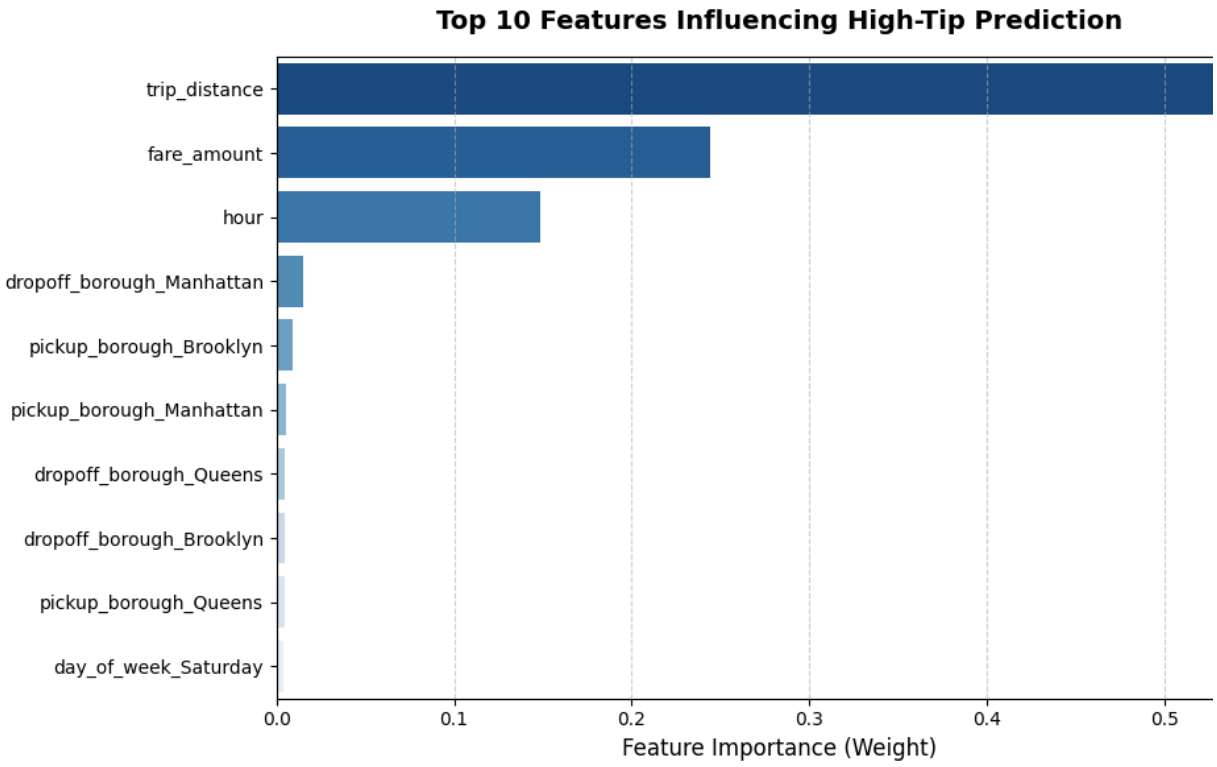
```
# Create binary column for High Tip
df['High_Tip'] = (df['tip_amount'] > df['tip_amount'].mean()).astype(int)

# Feature selection
features = ['trip_distance', 'fare_amount', 'pickup_borough', 'dropoff_borough',

df_encoded = pd.get_dummies(df[features + ['High_Tip']], drop_first=True)

X = df_encoded.drop('High_Tip', axis=1)
y = df_encoded['High_Tip']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Train model
rf = RandomForestClassifier(n_estimators=150, random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("🎯 Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
🎯 Accuracy: 0.6306125

Classification Report:
               precision    recall  f1-score   support

           0       0.70      0.76      0.73     51727
           1       0.47      0.39      0.43     28273

    accuracy                           0.63     80000
   macro avg       0.58      0.58      0.58     80000
weighted avg       0.62      0.63      0.62     80000
```

In [7]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

# Sort features by importance
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
features_list = X.columns

feat_imp_df = pd.DataFrame({
    'Feature': features_list[indices],
    'Importance': importances[indices]
})
```

```
plt.figure(figsize=(10,6))
sns.barplot(
    data=feat_imp_df.head(10),
    x='Importance',
    y='Feature',
    hue='Feature',
    dodge=False,
    legend=False,
    palette='Blues_r'
)

plt.title("Top 10 Features Influencing High-Tip Prediction", fontsize=14, weight
plt.xlabel("Feature Importance (Weight)", fontsize=12)
plt.ylabel("")
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Route-Based Revenue Forecast

In [8]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# --- Step 1: Prepare dataset ---
# Keep only relevant years
df = df[(df['lpep_pickup_datetime'].dt.year >= 2017) & (df['lpep_pickup_datetime

# Create route name
df['route'] = df['pickup_borough'] + " → " + df['dropoff_borough']

# Create month column
df['month'] = df['lpep_pickup_datetime'].dt.to_period('M').dt.to_timestamp()

# Aggregate revenue per route per month
```

```
monthly_route = df.groupby(['route', 'month']).agg(
    total_revenue=('total_amount', 'sum'),
    total_trips=('total_amount', 'count'),
    avg_fare=('fare_amount', 'mean'),
    avg_distance=('trip_distance', 'mean')
).reset_index()

print(monthly_route.head())
```

```
          route      month  total_revenue  total_trips   avg_fare  \
0  Bronx → Bronx 2017-01-01        2715.92          241   9.841909
1  Bronx → Bronx 2017-02-01        2874.64          276   9.215580
2  Bronx → Bronx 2017-03-01        4264.67          364  10.381319
3  Bronx → Bronx 2017-04-01        3091.79          270  10.195556
4  Bronx → Bronx 2017-05-01        2664.08          237   9.905105

   avg_distance
0      2.044689
1      1.823007
2      2.147418
3      2.057407
4      1.981983
```

In [9]:
```
# Extract time features
monthly_route['year'] = monthly_route['month'].dt.year
monthly_route['month_num'] = monthly_route['month'].dt.month
monthly_route['is_year_start'] = monthly_route['month'].dt.is_year_start.astype(

# Sort for lag features
monthly_route = monthly_route.sort_values(['route', 'month'])

# Lag (previous month revenue) — helps model learn trends
monthly_route['prev_month_revenue'] = monthly_route.groupby('route')['total_reve
monthly_route['revenue_growth'] = (
    monthly_route['total_revenue'] - monthly_route['prev_month_revenue']
) / monthly_route['prev_month_revenue']

# Drop NaN from first months
monthly_route.dropna(inplace=True)
```

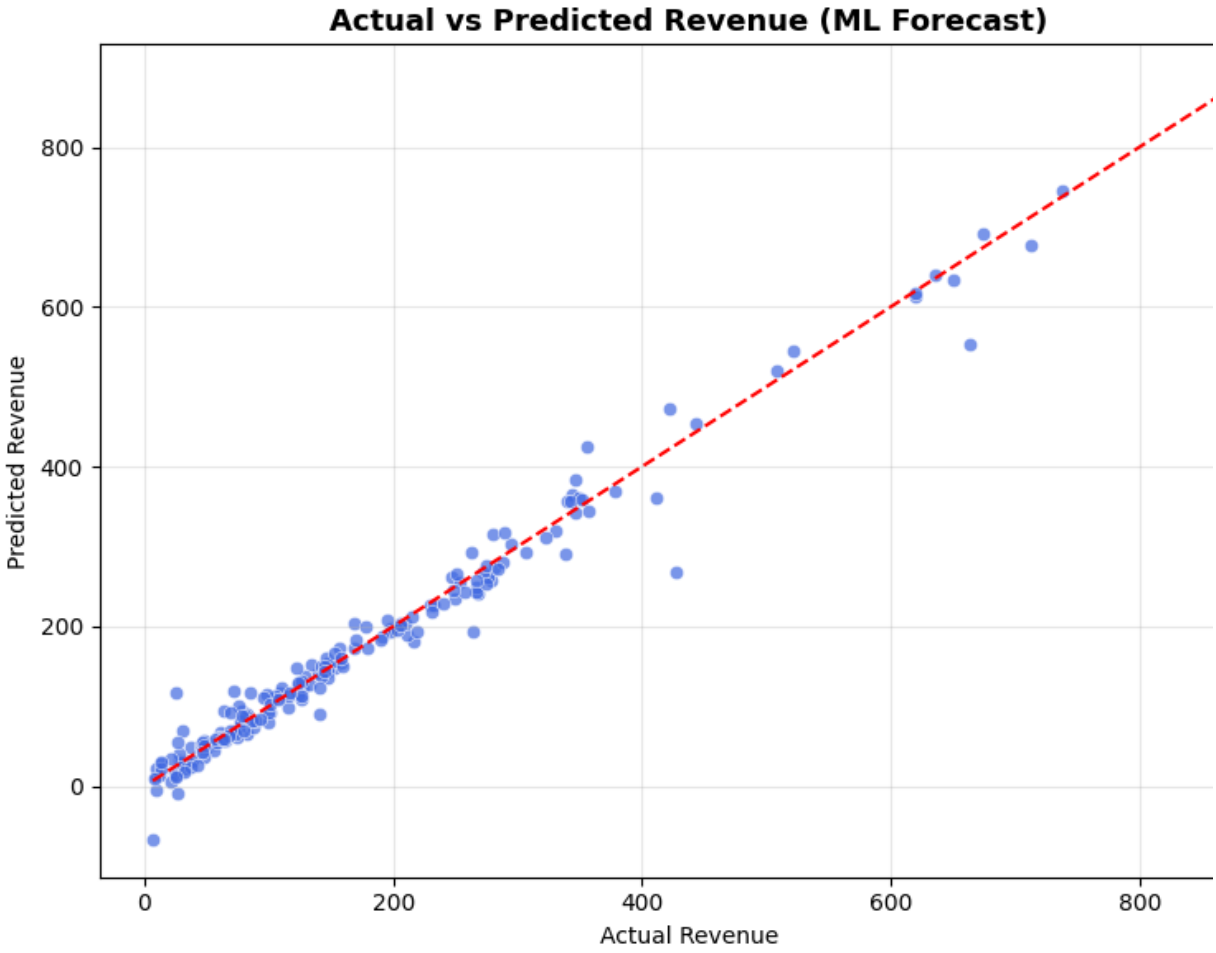
In [17]:
```
#  ROUTE-BASED REVENUE FORECASTING


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score


# 1 FEATURE SELECTION & ENCODING


# Features used for model training
features = [
    'total_trips', 'avg_fare', 'avg_distance',
    'year', 'month_num', 'is_year_start',
    'prev_month_revenue', 'revenue_growth'
```

```
]

# ✅ Ensure monthly_route exists
if 'monthly_route' not in locals():
    raise ValueError("monthly_route DataFrame not found. Please create it before

# ✅ Select only Top 20 routes to avoid MemoryError from one-hot encoding
top_routes = monthly_route['route'].value_counts().head(20).index
monthly_route_small = monthly_route[monthly_route['route'].isin(top_routes)].cop

# ✅ One-hot encode routes safely
monthly_route_encoded = pd.get_dummies(monthly_route_small, columns=['route'], d

# ✅ Build X and y
X = monthly_route_encoded[features + [col for col in monthly_route_encoded.colum
y = monthly_route_encoded['total_revenue']


# 2️⃣ CLEAN FEATURE MATRIX SAFELY

X = X.copy()
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.fillna(0, inplace=True)
y = y.replace([np.inf, -np.inf], np.nan).fillna(0)


# 3️⃣ TRAIN-TEST SPLIT & MODEL TRAINING

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, shuffle=True
)

lr = LinearRegression()
lr.fit(X_train, y_train)

# Predictions
y_pred = lr.predict(X_test)


# 4️⃣ EVALUATION

mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("📊 Model Performance:")
print(f"   • Mean Absolute Error (MAE): {mae:.2f}")
print(f"   • R² Score: {r2:.2f}")


# 5️⃣ VISUALIZATION: ACTUAL vs PREDICTED

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, color='royalblue', alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
plt.title("Actual vs Predicted Revenue (ML Forecast)", fontsize=13, weight='bold
plt.xlabel("Actual Revenue")
plt.ylabel("Predicted Revenue")
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

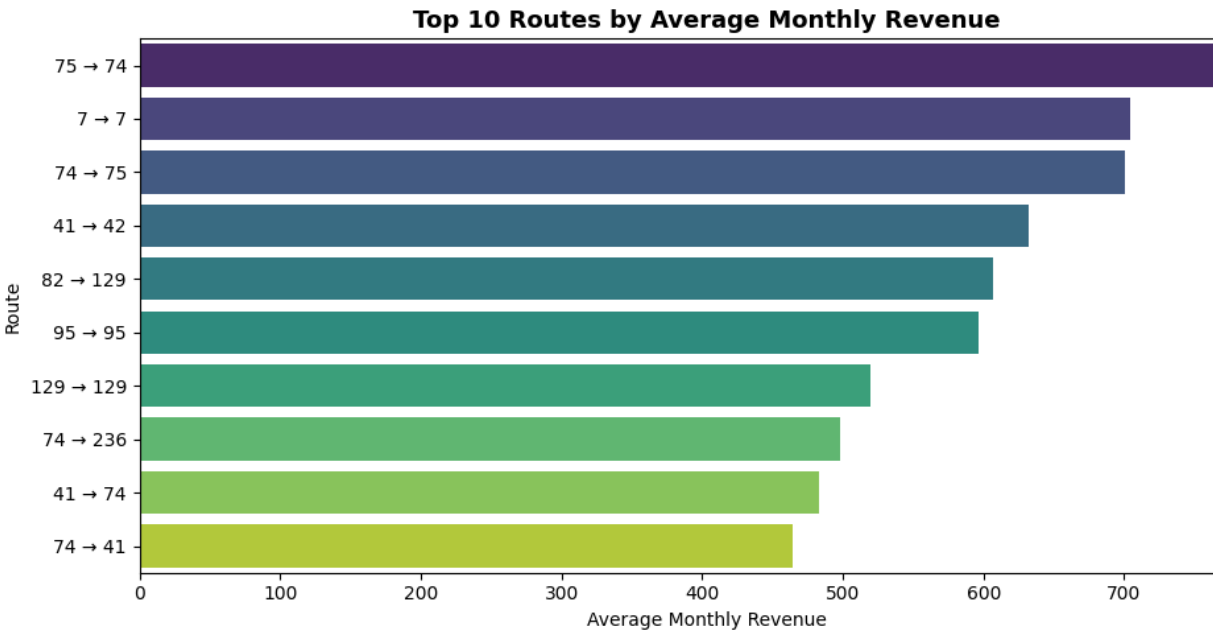
```
# 6 TOP ROUTES BY AVERAGE REVENUE

# Calculate top 10 routes by average monthly revenue
top_revenue_routes = (
    monthly_route.groupby('route')['total_revenue']
    .mean()
    .sort_values(ascending=False)
    .head(10)
)

# Convert to DataFrame for plotting
top_routes_df = top_revenue_routes.reset_index()
top_routes_df.columns = ['Route', 'Avg Monthly Revenue']

plt.figure(figsize=(10, 5))
sns.barplot(
    data=top_routes_df,
    x='Avg Monthly Revenue',
    y='Route',
    palette='viridis'
)
plt.title("Top 10 Routes by Average Monthly Revenue", fontsize=13, weight='bold'
plt.xlabel("Average Monthly Revenue")
plt.ylabel("Route")
plt.tight_layout()
plt.show()

print("\n🚀 Top 10 Routes by Average Monthly Revenue:\n")
print(top_routes_df.to_string(index=False))
```

📊 Model Performance:

- Mean Absolute Error (MAE): 14.31
- R^2 Score: 0.98




```
C:\Users\anshi\AppData\Local\Temp\ipykernel_16140\16477034.py:99: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.
Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(
```



🚀 Top 10 Routes by Average Monthly Revenue:

Route	Avg Monthly Revenue
75 → 74	768.600833
7 → 7	704.010208
74 → 75	700.487500
41 → 42	632.104792
82 → 129	606.688542
95 → 95	596.680208
129 → 129	519.283830
74 → 236	498.486667
41 → 74	483.401250
74 → 41	464.172708

SUMMARY

1. Weekly Trip Demand Forecasting

- Model Used: Facebook Prophet
- Goal: Predict weekly trip counts using past demand and seasonal trends.
- Process: Aggregated weekly trips (2017–2020) and modeled with Prophet.
- Result:

->Clear weekly seasonality observed.

->Forecast shows a stable upward trend in future demand.

- Insight: Helps in resource planning and peak week management.

2. **High-Tip Prediction Model**

- Model Used: Random Forest Classifier
- Goal: Predict whether a trip will have a tip above average.
- Key Features: Trip distance, fare amount, pickup/drop-off boroughs, time of day.
- Result:

->Accuracy: 63%

->Precision: 70% (for low-tip class), 47% (for high-tip class).

->Important Predictors: Fare amount, distance, pickup time.

- Insight: Identifies scenarios likely to yield higher tips — useful for driver performance optimization.

3. **Route-Based Revenue Forecasting**

- Model Used: Linear Regression (ML-based Forecasting)
- Goal: Predict monthly total revenue per route using historical trends.
- Features Used:

->Total trips, avg fare, avg distance

->Year, month indicators

->Previous month revenue & revenue growth

- Result:
- R^2 Score: 0.98
- Top Routes (Based on Zone IDs):
  - 75 → 74
  - 74 → 75
  - 41 → 42

- Insight:

->These zone numbers correspond to geographic locations defined in the Taxi Zone Lookup table provided by Uber/NYC TLC.

->For example:

Zone ID	Zone Name	Borough
75	East Village	Manhattan
74	Alphabet City	Manhattan
41	Midtown Center	Manhattan
42	Midtown East	Manhattan

- Thus, routes like 75 → 74 represent high-traffic intra-Manhattan connections, driving major revenue.

SUGGESTIONS

1. Optimize Fleet & Drivers
   - Deploy more drivers on Fridays & weekends (44% of weekly demand).
   - Focus on Manhattan, Queens, Brooklyn (92% of total trips).

2. Boost Revenue
   - Promote top routes like East Village → Alphabet City.
   - Use dynamic pricing during peak hours (Fri 6 PM).

3. Enhance Customer Experience
   - Keep cash payments (40% of transactions).
   - Reward drivers on high-tip routes to improve retention.

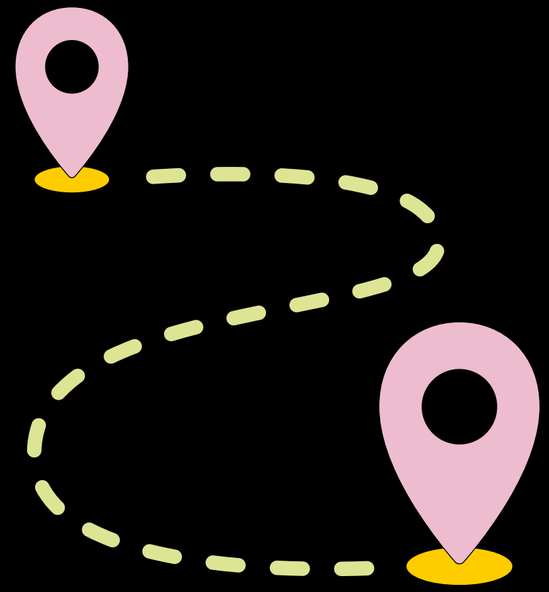
4. Data-Driven Planning
   - Use weekly demand forecasts for better fleet planning.
   - Monitor trip distance & borough revenue for trends.

5. Targeted Marketing
   - Run local campaigns in high-frequency zones (Harlem, Astoria).
   - Consider ride-pooling in high-demand areas.

Uber

THANK YOU

FOR YOUR ATTENTION !



Presented By- Anshika Tejwani