Uber Trips Analysis

Objective:

Ridesharing has undeniably been a blessing for a traffic-congested city like Dhaka. However, I found myself pondering over my personal experiences with it. Was it a luxury that I indulged in occasionally, or did I rely on it for all of my transportation needs? To gain insights into my ridesharing history, I decided to request my data from Uber. They promptly provided me with a comma-separated values (.csv) file containing all of my ride data associated with my email ID.
Now I have the data; my plan is to clean the data first in DBeaver with Postgresql codes. After that, I want to visualize key metrics with pandas (Python).

Data Cleaning with PostgreSQL using DBeaver:

First, creating a new table named ubrtrips. Since the file Uber gave me live in my machine, so, I will copy it from that file. While copything the data, I will stick to the data type following the best practices. Here's the code to copy data from a stored .csv file. All the column names should be exact matches.

```
drop table if exists ubrtrips;


create table ubrrips(
City varchar(20), "Product Type" varchar(20), "Trip or Order Status" varchar(20),
"Request Time" varchar(256), "Begin Trip Time" varchar(256), "Begin Trip Lat"
numeric, "Begin Trip Lng" numeric,
"Begin Trip Address" varchar(512), "Dropoff Time" varchar(256), "Dropoff Lat"
numeric,
"Dropoff Lng" numeric, "Dropoff Address" varchar(512), "Distance (miles)" numeric,
"Fare Amount" numeric, "Fare Currency" varchar(30)
);


copy ubrtrips
from 'C:\files\Uber Trips\trips_data.csv'
with (format csv, header);
```

Now that I have obtained my data and stored it in a SQL database, I need to address the formatting of certain data types. Specifically, the timestamp values are currently expressed as "13:34:45 +00 UTC." Since the "+00" already indicates the UTC timezone, there is no need to explicitly label it as "UTC." Furthermore, this format is not compatible with Postgresql's standards. Therefore, I will remove the "UTC" label and convert the timestamps to proper timestamp with time zone values in accordance with Postgresql's format.

```
alter table ubrtrips
add column "Time Request" timestamptz,
add column "Time Trip Begin" timestamptz,
add column "Time Dropoff" timestamptz;
update ubrtrips set "Time Request" = substring("Request Time", 1, 26)::timestamptz;
update ubrtrips set "Time Trip Begin" = substring("Begin Trip Time", 1,
26)::timestamptz;
update ubrtrips set "Time Dropoff" = substring("Dropoff Time", 1, 26)::timestamptz;
alter table uber_rides
drop column "Request Time",
drop column "Begin Trip Time",
drop column "Dropoff Time";
```

Now that the timestamp formatting is resolved, let's shift our attention to other columns. It appears that the naming of different products is inconsistent in the dataset. For instance, the same service is referred to as 'UberMoto,' 'Moto,' and 'Uber Moto.' Similarly, there are variations such as 'uberX,' 'Emergency Car,' and 'Cars Emergency' that represent the same product. To address this inconsistency, I have taken the following steps to correct the names accordingly.

```
alter table ubrtrips add column product_fix text;
update ubrtrips
set product_fix = "Product Type";
```

```
update ubrtrips
set product_fix = 'Uber Moto'
where "Product Type" like '%Moto';

update ubrtrips
set product_fix = 'UberX'
where "Product Type" like '%Emergency%';

update ubrtrips
set product_fix = 'UberX'
where "Product Type" like 'u%';

update ubrtrips
set product_fix = 'Uber Pool'
where "Product Type" like 'Pool%';

update ubrtrips
set product_fix = 'Uber Premier'
where "Product Type" like '%Premier%';

update ubrtrips
set "Product Type" = product_fix;

alter table ubrtrips
drop column product_fix;
```

Very Well! Let’s now add the colum for duration, in hours. Here’s the code for that-

```
alter table ubrtrips add column duration_hour numeric ;
update ubrtrips
set duration_hour =
date_part('hour', age("Time Dropoff" , "Time Trip Begin") ) +
date_part('minutes', age("Time Dropoff" , "Time Trip Begin") )/60+
date_part('seconds', age("Time Dropoff" , "Time Trip Begin") )/3600;
```

Answering Business Questions:

Now that our data is prepared and operations can begin, we can proceed to answer the question: What is the total expense for each year, categorized by Product Type, in the city of Dhaka? To provide a clear visualization of this information, I have created a pivot table. The pivot table will summarize the data and present the total expenses for each year corresponding to the respective Product Types.

```
select "Product Type",
coalesce (sum("Fare Amount") filter (where date_part('year', "Time Dropoff") = 2019),
0) as "2019 Fare",
coalesce (sum("Fare Amount") filter (where date_part('year', "Time Dropoff") = 2020),
0) as "2020 Fare",
coalesce (sum("Fare Amount") filter (where date_part('year', "Time Dropoff") = 2021),
0) as "2021 Fare",
coalesce (sum("Fare Amount") filter (where date_part('year', "Time Dropoff") = 2022),
0) as "2022 Fare",
coalesce (sum("Fare Amount") filter (where date_part('year', "Time Dropoff") = 2023),
0) as "2023 Fare"
from ubrtrips
where city= 'Dhaka' and
"Trip or Order Status"= 'COMPLETED'
group by 1
order by 1;
```

The above code returned following values. It's nice how it replaces null values with 0.

Product Type	2019 Fare	2020 Fare	2021 Fare	2022 Fare	2023 Fare
CNG	0	0	0	0	730.19
Connect	0	0	0	193.17	0
Uber Moto	1,885.27	1,020.98	207.11	0	0
Uber Pool	1,767.54	1,340	0	0	0
Uber Premier	2,325.97	722.9	0	1,936.34	0
UberX	14,387.39	5,881.42	19,702.53	18,580.73	4,784.47
UberXL	615.08	3,830.84	0	0	0
XL intercity	0	0	0	3,421.76	0

With the conclusion of my exploration using Postgresql, it is evident that UberX is my preferred choice for rides. Moving forward, I will utilize pandas to visually represent various aspects and insights derived from my Uber ride data. Pandas provides powerful capabilities for data manipulation and visualization, enabling me to effectively showcase different aspects of my Uber rides.