drivers_LTV

January 25, 2025

[357]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import missingno as msno
```

[358]:

```
#read raw data
drivers = pd.read_csv('../data/driver_ids.csv')

rides = pd.read_csv('../data/ride_ids.csv')

ride_timestamp1 = pd.read_csv('../data/ride_timestamp_part_1.csv')
ride_timestamp2 = pd.read_csv('../data/ride_timestamp_part_2.csv')
ride_timestamp3 = pd.read_csv('../data/ride_timestamp_part_3.csv')

#merge dataset
ride_ts = pd.concat([ride_timestamp1, ride_timestamp2, ride_timestamp3],␣
 ↪ignore_index= True)
```

[359]:

```
#drivers pool
print('Total drivers: ', drivers['driver_id'].nunique(),'\n')
#print(drivers.head())
```

```
Total drivers:  937
```

[360]:

```
#rides summary
print('Total rides: ',rides['ride_id'].nunique(),'\n')
print(rides.describe())
#print(rides.head())
```

```
Total rides:  193502

       ride_distance  ride_duration  ride_prime_time
count  193502.000000  193502.000000    193502.000000
mean     6955.218266     858.966099        17.305893
std      8929.444606     571.375818        30.825800
min        -2.000000       2.000000         0.000000
25%      2459.000000     491.000000         0.000000
50%      4015.000000     727.000000         0.000000
```

```
75%      7193.000000    1069.000000       25.000000
max    724679.000000   28204.000000      500.000000
```

[361]:
```
#set timestamp to datetime
ride_ts['timestamp'] = pd.to_datetime(ride_ts['timestamp'])

#total rides timestamp
print('Rides with timestamp', ride_ts['ride_id'].nunique(),'\n')

#Last rides
print('Last ride\n',ride_ts[ride_ts['timestamp']== ride_ts['timestamp'].
 ↪max()],'\n')

#First rides
print('First ride\n',ride_ts[ride_ts['timestamp']== ride_ts['timestamp'].
 ↪min()],'\n')

#different events
print('The recorded events are: ',ride_ts['event'].unique())
```

```
Rides with timestamp 194081

Last ride
                                  ride_id        event           timestamp
507986  86519fac0a61d5deb4d2a4782ae59475  accepted_at 2023-10-31 23:59:56

First ride
                                  ride_id         event           timestamp
595845  9d3349ce979e58ca08ff4a1819ef5e6c  requested_at 2023-10-01 00:00:03

The recorded events are:  ['requested_at' 'accepted_at' 'arrived_at'
'picked_up_at' 'dropped_off_at']
```

[362]:
```
#Pricing model is given by the following formula:
#(base fare + cost per mile * ride_distance + cost per minute * ride_duration)␣
 ↪* (1+ (ride_prime_time/100)) + service_fee

base_fare = 2
cost_per_mile= 1.15
cost_per_min= 0.22
service_fee= 1.75
min_fare = 5.00
max_fare = 400.00

#distance is in meters and duration in seconds so they have to be converted
#prime time is a percentage so to use it as multiplier it should be divided by␣
 ↪100
#estimating every ride's price
```

```
rides['price'] = (base_fare + cost_per_mile*rides['ride_distance']*.000621 +
↪cost_per_min*rides['ride_duration']/60)*(1 + rides['ride_prime_time']/100) +
↪service_fee

#the prices are capped by the min & max fare
rides['price'] =  rides['price'].clip(lower= min_fare, upper= max_fare)


print(rides[['ride_id','price']].head())
```

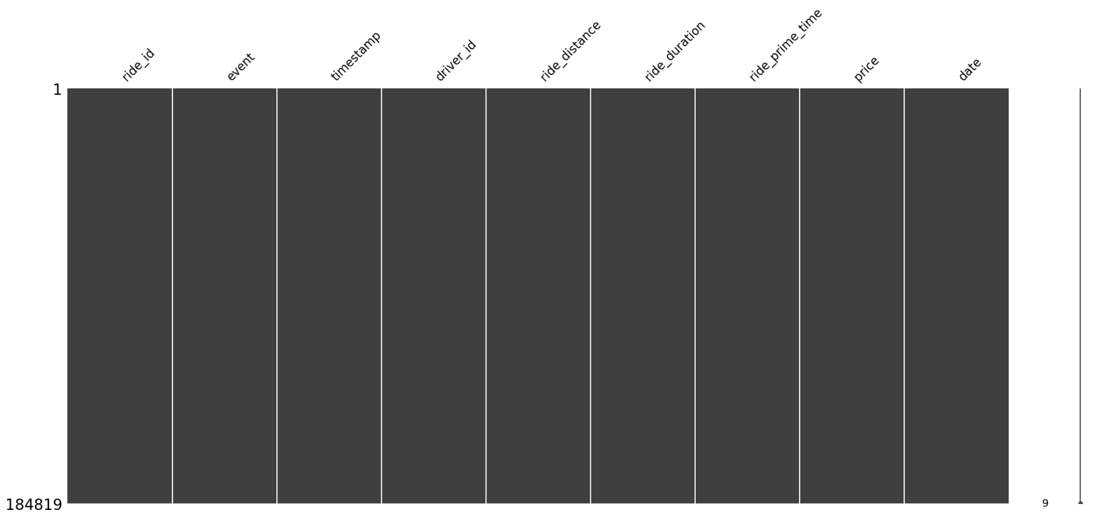
```
                            ride_id      price
0  006d61cf7446e682f7bc50b0f8a5bea5   8.488488
1  01b522c5c3a756fbdb12e95e87507eda   9.117306
2  029227c4c2971ce69ff2274dc798ef43   8.191174
3  034e861343a63ac3c18a9ceb1ce0ac69  77.826485
4  034f2e614a2f9fc7f1c2f77647d1b981  17.662788
```

[363]:
```
#Merging ride info into single df
#the ride is completed once the customer was dropped off
ride_completed = ride_ts[ride_ts['event'] == 'dropped_off_at' ]

#joining both df to have date and price for each completed ride
full_rides= pd.merge(ride_completed, rides,how= 'inner' ,on= 'ride_id')
full_rides['date'] = full_rides['timestamp'].dt.date

#making sure there are no duplicates
print(full_rides['ride_id'].nunique())
print(full_rides.shape)
```

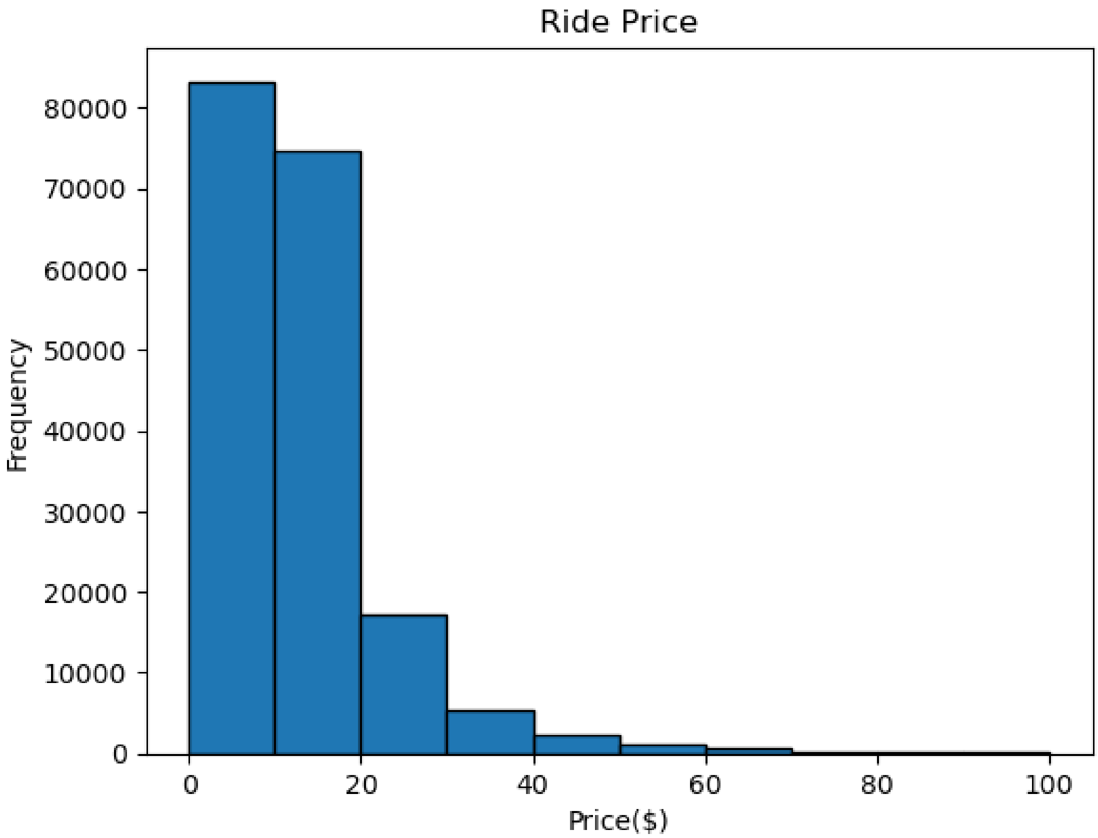
```
184819
(184819, 9)
```

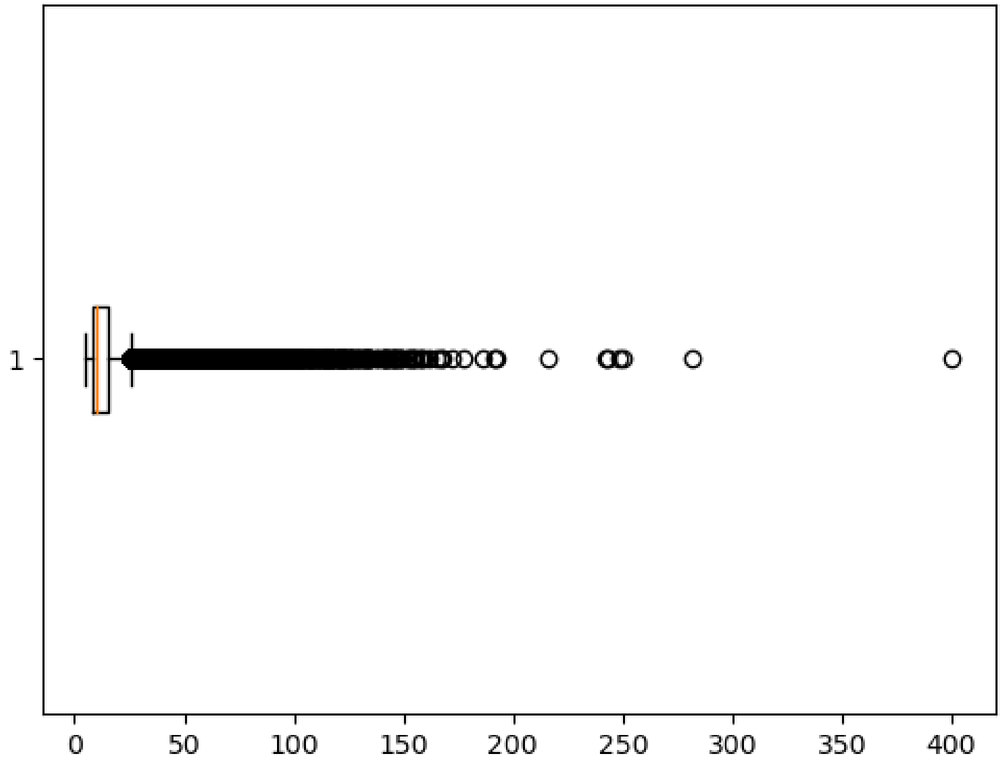
[364]:
```
#making sure there is no missing data in ride info
msno.matrix(full_rides)
```

[364]: <Axes: >



[365]:
```
#Distribution on individual ride prices
plt.hist(full_rides['price'],bins = 10, range=(0, 100),edgecolor= 'black')
plt.title('Ride Price')
plt.xlabel('Price($)')
plt.ylabel('Frequency')
plt.show()

plt.boxplot(full_rides['price'], vert=False)
plt.show()
#As depicted most rides are below $20
```


Ride Price
Frequency
80000
70000
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
Price($)



[366]:

```
#Aggregating data per driver per date
#calculating revenue per driver
revenue_driver = full_rides.groupby(['driver_id','date']).
 ↪agg(daily_revenue=('price','sum'), daily_rides=('ride_id','count')).
 ↪reset_index()
revenue_driver['avg_revenue_per_ride'] = round(revenue_driver['daily_revenue']/
 ↪revenue_driver['daily_rides'],2)

#print(revenue_driver.shape)
#drivers who completed trips during the analyzed period
print('Drivers who completed at least one trip:',revenue_driver['driver_id'].
 ↪nunique())

revenue_driver.head()
```

```
Drivers who completed at least one trip: 844
```

[366]:

	driver_id	date	daily_revenue	daily_rides	\
0	002be0ffdc997bd5c50703158b7c2491	2023-10-01	200.707383	19	
1	002be0ffdc997bd5c50703158b7c2491	2023-10-02	51.831718	6	
2	002be0ffdc997bd5c50703158b7c2491	2023-10-03	101.562257	8	
3	002be0ffdc997bd5c50703158b7c2491	2023-10-04	39.190640	4	

```
4  002be0ffdc997bd5c50703158b7c2491  2023-10-05     236.059445           24

   avg_revenue_per_ride
0                 10.56
1                  8.64
2                 12.70
3                  9.80
4                  9.84
```

[367]:

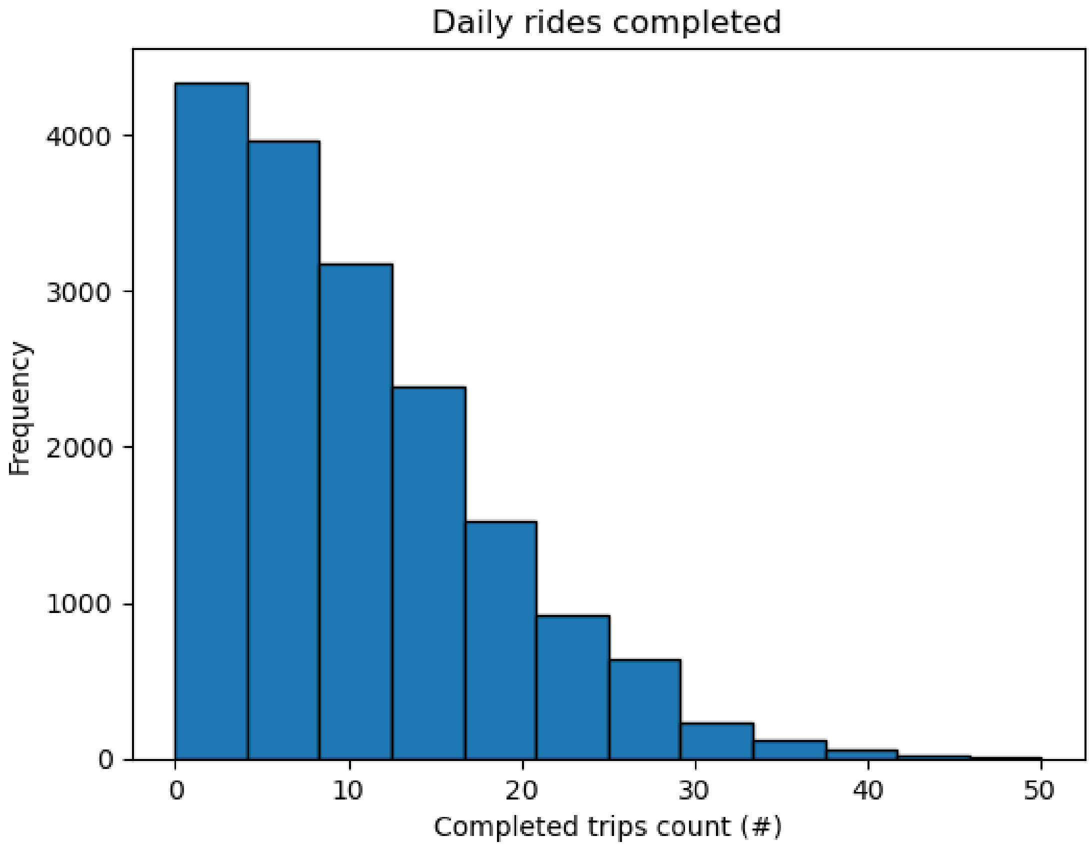
```
#Descriptive statistics of each worked day
revenue_driver.describe()
```

[367]:

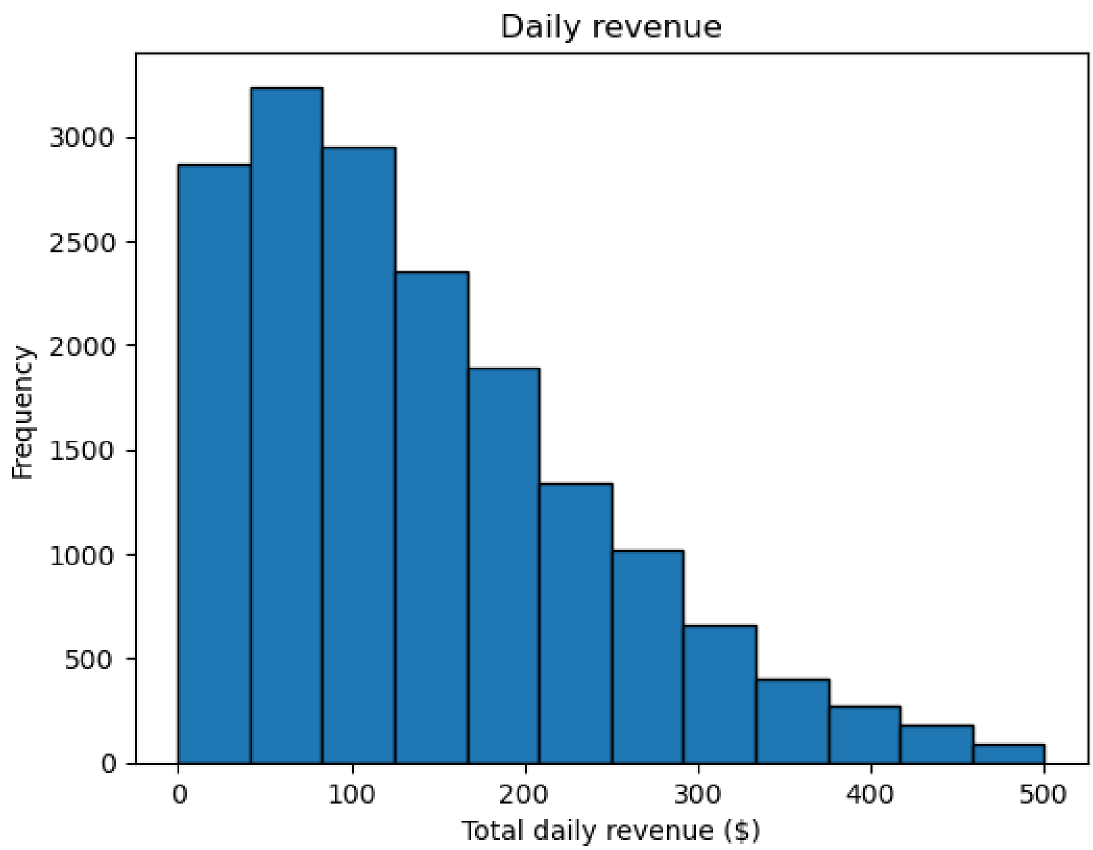
	daily_revenue	daily_rides	avg_revenue_per_ride
count	17395.000000	17395.000000	17395.000000
mean	143.789206	10.624835	14.047468
std	107.620684	7.795852	6.375898
min	5.000000	1.000000	5.000000
25%	59.943399	5.000000	10.930000
50%	119.349317	9.000000	12.860000
75%	202.231302	15.000000	15.480000
max	827.000306	58.000000	249.900000

[368]:

```
#Distribution of how many rides are completed by drivers
plt.hist(revenue_driver['daily_rides'], bins= 12, range=(0,50)␣
↪,edgecolor='black')
plt.title('Daily rides completed')
plt.xlabel('Completed trips count (#)')
plt.ylabel('Frequency')
plt.show()
```



[369]:
```
#Distribution of the daily revenue drivers earn
plt.hist(revenue_driver['daily_revenue'],bins=12, range=(0,500),␣
 ↪edgecolor='black')
plt.title('Daily revenue')
plt.xlabel('Total daily revenue ($)')
plt.ylabel('Frequency')
plt.show()
```



[370]:
```python
#Aggregate ride data for each individual driver
#Average Daily Revenue per Driver = Total Revenue per Driver / Total Number of␣
↪Working Days
drivers_ridesummary = revenue_driver.groupby('driver_id').agg(avg_daily_rev=␣
↪('daily_revenue','mean'),␣
↪avg_daily_rides=('daily_rides','mean'),working_days = ('date','count'))
drivers_ridesummary.head()
```

[370]:

driver_id	avg_daily_rev	avg_daily_rides	working_days
002be0ffdc997bd5c50703158b7c2491	118.668555	9.233333	30
007f0389f9c7b03ef97098422f902e62	29.221336	2.818182	11
011e5c5dfc5c2c92501b8b24d47509bc	53.588801	3.777778	9
0152a2f305e71d26cc964f8d4411add9	93.221038	6.821429	28
01674381af7edd264113d4e6ed55ecda	185.504995	12.931034	29

[371]:
```python
#When was the driver last active? This datapoint is requried to estimate a␣
↪driver's lifespan
#We assume last active was the last customer dropped off
#getting last day activity of each driver
```

```
alive= full_rides.groupby('driver_id').agg(last_active=('timestamp','max'))
alive.head()
```

[371]:
```
                                         last_active
driver_id
002be0ffdc997bd5c50703158b7c2491 2023-10-31 22:47:03
007f0389f9c7b03ef97098422f902e62 2023-10-29 22:48:33
011e5c5dfc5c2c92501b8b24d47509bc 2023-10-26 20:18:29
0152a2f305e71d26cc964f8d4411add9 2023-10-31 14:44:17
01674381af7edd264113d4e6ed55ecda 2023-10-31 13:17:59
```

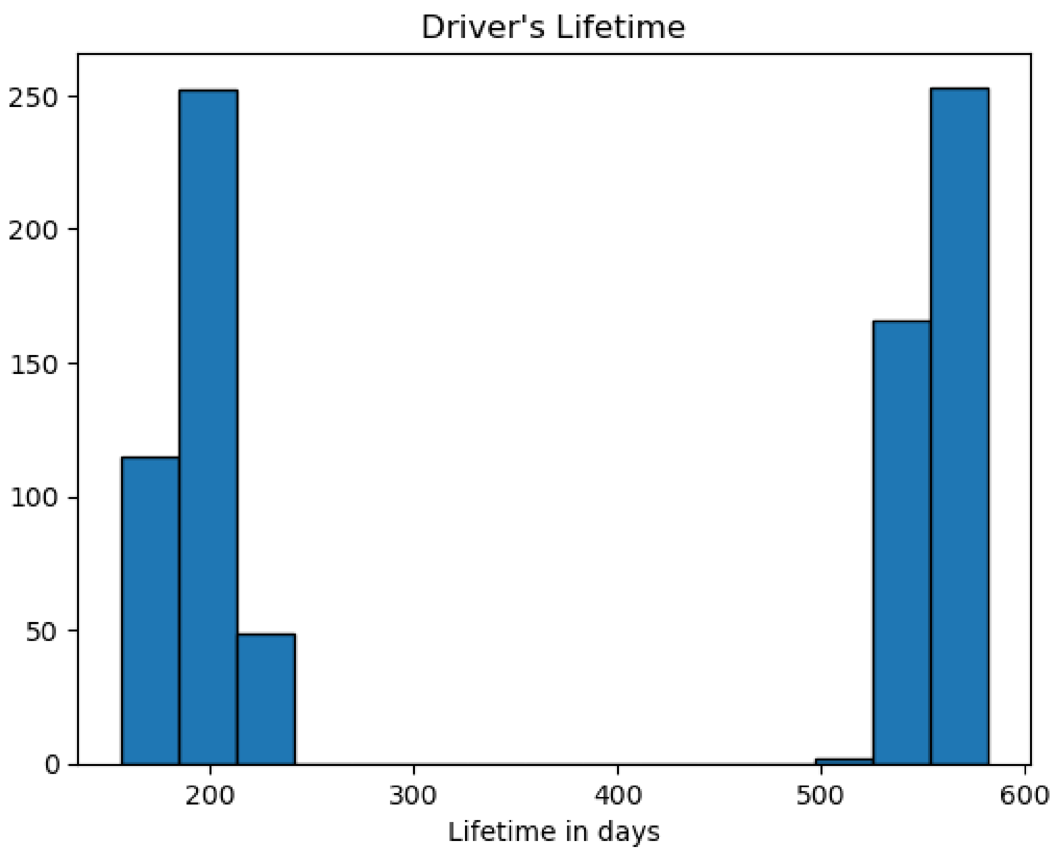
[372]:
```
#Calculating driver's lifespan
drivers_lt= pd.merge(drivers,alive, how= 'inner',on ='driver_id')
drivers_lt['driver_onboard_date'] = pd.
 ↪to_datetime(drivers_lt['driver_onboard_date'])
drivers_lt['last_active']= pd.to_datetime(drivers_lt['last_active'])
drivers_lt['lifespan']= drivers_lt['last_active'] -␣
 ↪drivers_lt['driver_onboard_date']
drivers_lt['lifespan'] = np.ceil(drivers_lt['lifespan'].dt.days)

drivers_lt.head()
```

[372]:
```
                          driver_id driver_onboard_date         last_active  \
0  002be0ffdc997bd5c50703158b7c2491          2023-03-29 2023-10-31 22:47:03
1  007f0389f9c7b03ef97098422f902e62          2022-03-29 2023-10-29 22:48:33
2  011e5c5dfc5c2c92501b8b24d47509bc          2022-04-05 2023-10-26 20:18:29
3  0152a2f305e71d26cc964f8d4411add9          2023-04-23 2023-10-31 14:44:17
4  01674381af7edd264113d4e6ed55ecda          2023-04-29 2023-10-31 13:17:59

   lifespan
0     216.0
1     579.0
2     569.0
3     191.0
4     185.0
```

[373]:
```
drivers_lt['lifespan'].describe()
```

[373]:
```
count    837.000000
mean     376.583035
std      183.133313
min      156.000000
25%      194.000000
50%      526.000000
75%      559.000000
max      582.000000
Name: lifespan, dtype: float64
```

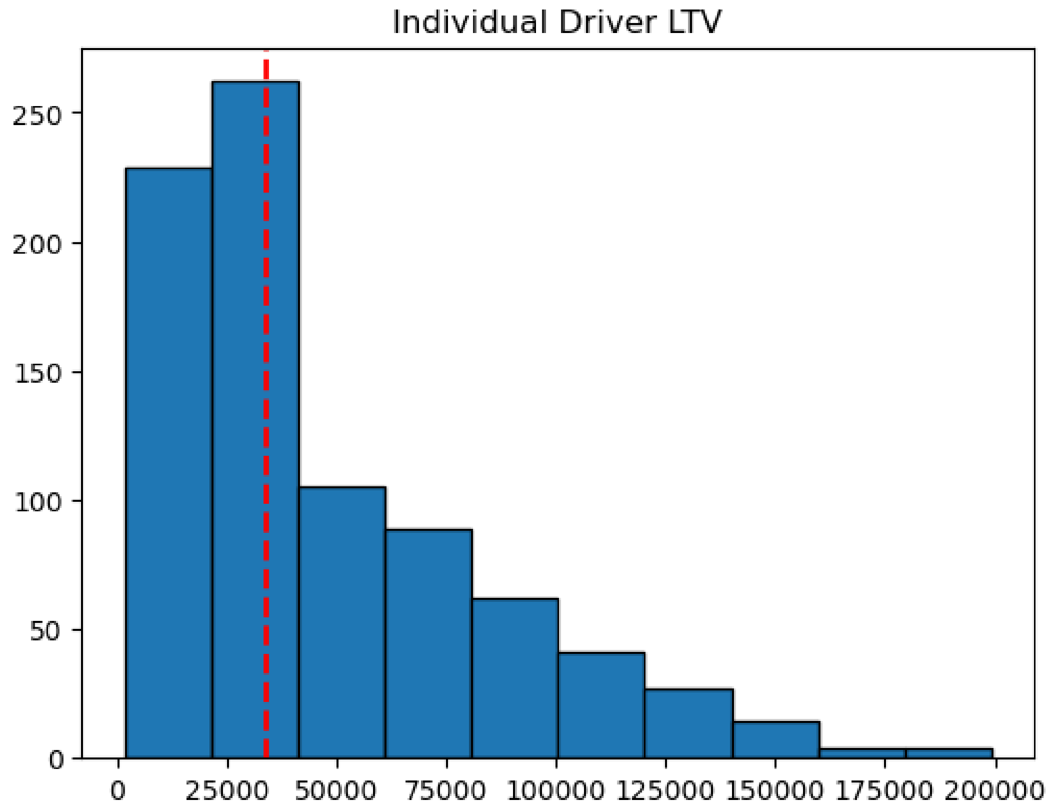
[374]:
```
#Distribution of drivers lifetime in days
#As seen in the chart drivers are clustered into two groups one around ~200␣
↪days and other around ~550 days
plt.hist(drivers_lt['lifespan'],bins=15, edgecolor='black')
plt.xlabel('Lifetime in days')
plt.title("Driver's Lifetime")
plt.show()
```



[375]:
```
#Now that we have every driver's lifespan we can get out first LTV estimate
# This calculation uses averages for all drivers during working days
#We do not have data on Cost of Acqusitiion so for practical matters we will␣
↪assume its $0

# LTV=(Average Revenue per Driver per working day × Average Lifespan) - Cost of␣
↪Acquisition
LTV = drivers_lt['lifespan'].mean() * drivers_ridesummary['avg_daily_rev'].
↪mean()
print('The average LTV for all drivers is: $', round(LTV,2))
```

```python
# However previous calculation was not accounting for variation in revenue for␣
↪each driver
#So the next step is calculating an individualized LTV

#Individual driver LTVi = (Avg Revenue per working day i  x Lifespan i) - Cost␣
↪of Acquisition i
driver_ltv = pd.merge(drivers_lt, drivers_ridesummary, how='inner' ,on=␣
↪'driver_id')
driver_ltv['LTV']= driver_ltv['lifespan'] * driver_ltv['avg_daily_rev']

driver_ltv.head()

#As we can see the LTV distribution is skewed to the right

print('The mean of individualized LTV is: $',round(driver_ltv['LTV'].mean(),2))
plt.hist(driver_ltv['LTV'], bins=10, edgecolor='black')
plt.axvline(driver_ltv['LTV'].median(), color='red', linestyle='--',␣
↪linewidth=2)
plt.title('Individual Driver LTV')
plt.show()
driver_ltv['LTV'].describe()
```

```
The average LTV for all drivers is: $ 46371.62
The mean of individualized LTV is: $ 47081.52
```



[375]:
```
count        837.000000
mean       47081.524944
std        37155.228386
min         1695.347735
25%        20123.795942
50%        33936.058838
75%        67282.636444
max       199245.045661
Name: LTV, dtype: float64
```

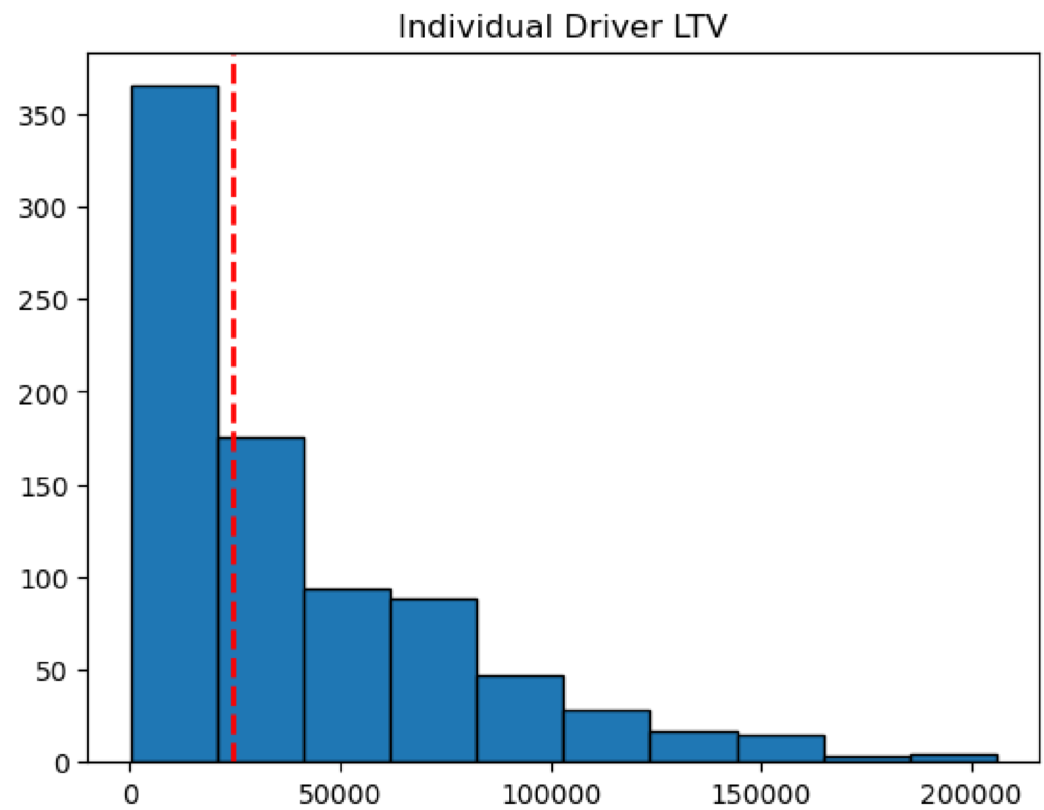
[376]:
```
#A major flaw in this calculation is assuming drivers are working every day of␣
↪their lifespan
#Ideally the probability of working every day should be fitted using a␣
↪distribution based on driver's historical data, however we do not have␣
↪access to it
#A way to model this probability is based on working intensity of each driver
#Individual driver LTVi = (Avg Revenue per working day i  x Lifespan i x␣
↪P(WI)i) - Cost of Acquisition i

#For estimating working intensity we divide P(WI) = #working days/ #days driver␣
↪was alive that month
```

```
start= pd.to_datetime('2023-10-01')
driver_ltv['days_alive']= driver_ltv['last_active']- start
driver_ltv['days_alive'] = driver_ltv['days_alive'].dt.days
driver_ltv['work_int'] = driver_ltv['working_days']/driver_ltv['days_alive']
driver_ltv['LTV_activity'] = driver_ltv['LTV'] * driver_ltv['work_int']
driver_ltv.head()
```

[376]:
```
                          driver_id driver_onboard_date         last_active  \
0  002be0ffdc997bd5c50703158b7c2491          2023-03-29 2023-10-31 22:47:03
1  007f0389f9c7b03ef97098422f902e62          2022-03-29 2023-10-29 22:48:33
2  011e5c5dfc5c2c92501b8b24d47509bc          2022-04-05 2023-10-26 20:18:29
3  0152a2f305e71d26cc964f8d4411add9          2023-04-23 2023-10-31 14:44:17
4  01674381af7edd264113d4e6ed55ecda          2023-04-29 2023-10-31 13:17:59

   lifespan  avg_daily_rev  avg_daily_rides  working_days           LTV  \
0     216.0     118.668555         9.233333            30  25632.407867
1     579.0      29.221336         2.818182            11  16919.153648
2     569.0      53.588801         3.777778             9  30492.027580
3     191.0      93.221038         6.821429            28  17805.218307
4     185.0     185.504995        12.931034            29  34318.424143

   days_alive  work_int  LTV_activity
0          30  1.000000  25632.407867
1          28  0.392857   6646.810362
2          25  0.360000  10977.129929
3          30  0.933333  16618.203753
4          30  0.966667  33174.476672
```

[377]:
```
#LTV is adjusted based on working intensity by assuming the driver worked with␣
 ↪similar intensity during his lifespan
#As shown in the working intensity adjusted LTV many of the drivers' LTV␣
 ↪decreases by shifting to the left of the graph
print('The mean of individual LTV is: $', round(driver_ltv['LTV_activity'].
 ↪mean(),2))
plt.hist(driver_ltv['LTV_activity'], bins=10, edgecolor='black')
plt.axvline(driver_ltv['LTV_activity'].median(), color='red', linestyle='--',␣
 ↪linewidth=2)
plt.title('Individual Driver LTV')
plt.show()
driver_ltv[['LTV_activity','work_int']].describe()
```

```
The mean of individual LTV is: $ 39212.79
```



[377]:

	LTV_activity	work_int
count	837.000000	837.000000
mean	39212.787717	0.722561
std	38738.075757	0.286763
min	282.557956	0.066667
25%	9116.274078	0.461538
50%	24473.568549	0.833333
75%	59718.430677	0.966667
max	205886.547183	1.250000

[378]:

```
#Now lets compare the individualized LTV calculations to see how much the␣
↪adjustment impacts the panormana
#The adjusted LTV is lower than the previously estimated after accounting for␣
↪the driver's working intensity
print('The adjusted LTV is lower by:',round((driver_ltv['LTV_activity'].mean()/␣
↪driver_ltv['LTV'].mean() *100)-100,2),'%')
```

The adjusted LTV is lower by: -16.71 %

[388]:

```
!jupyter nbconvert --to pdf drivers_LTV.ipynb --output-dir .\output
```

[NbConvertApp] Converting notebook drivers_LTV.ipynb to pdf

```
[NbConvertApp] Support files will be in drivers_LTV_files\
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Making directory .\drivers_LTV_files
[NbConvertApp] Writing 62505 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 196813 bytes to drivers_LTV.pdf
```

[ ]:

```
```