Lyft in Toledo, OH Case Study

Vivek Saravanan

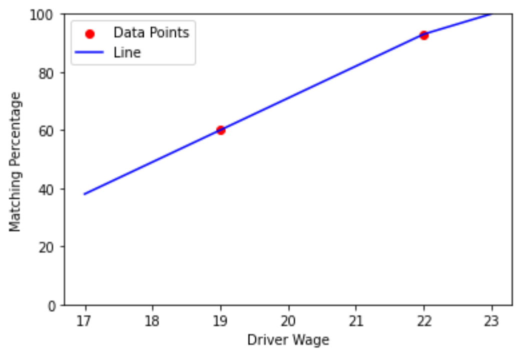
In [157]:
```
# Importing useful packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import linregress
from scipy.optimize import minimize
import pandas as pd
```

Part 1: Exploration and Discovery

Roadmap: In this section, my objective is to utilize the information presented in the case study to gain a deeper understanding of the situation, derive relevant metrics, and determine the optimal driver wage.

The code below creates a plot that illustrates the projection of the driver wage and matching percentage. It assumes a linear relationship between the two variables.

In [319]:
```
# Define the coordinates of the two points
x = np.array([19, 22])
y = np.array([60, 93])

# Calculate the slope and intercept using linregress
slope, intercept, _, _, _ = linregress(x, y)

# Create an array of x-values for the line
x_line = np.linspace(17, 23, 7)

# Calculate the corresponding y-values for the line
y_line = slope * x_line + intercept

# Set a maximum y-value of 100
y_line = np.minimum(y_line, 100)

# Create a DataFrame with x and y values
data = pd.DataFrame({'Driver Wage': x_line, 'Matching Percentage': y_line})

# Plot the points and the line
plt.scatter(x, y, color='red', label='Data Points')
plt.plot(x_line, y_line, color='blue', label='Line')
plt.xlabel('Driver Wage')
plt.ylabel('Matching Percentage')
plt.ylim(0, 100)  # Set the y-axis limits
plt.legend()
plt.show()
```



In [320]:
```
data.transpose()
```

Out[320]:

	0	1	2	3	4	5	6
Driver Wage	17.0	18.0	19.0	20.0	21.0	22.0	23.0
Matching Percentage	38.0	49.0	60.0	71.0	82.0	93.0	100.0

It's notable that after $23 the matching percentage between drivers and riders is above 100\% - the cutoff is therefore included in the plot.

Adding onto the table, I included some other helpful numbers and metrics that correspond with each value of the driver wage.

- Churn Rate: Calculated using the match rate and the 2 different churn scenarios: 10% for customers that found a match, and 33% for customers without a match
- Lyft Cut: How much Lyft makes on each ride
- ARPU: An acronym for Average Revenue Per User that is pretty self explanatory, calculated using the Lyft Cut and Matching Percentage
- Customer Lifetime Value: How much the customer will bring in with revenue over their span (Note: This is on a monthly basis and does not take into account an entire year)

In [321]:
```
# Churn Calculation is conducted by taken by multiplying the matched rides percentage by 10%
## and the unmatched rides percentage by 33%

data['Churn Rate'] = 100 * ((data['Matching Percentage']/100 * .1) + ((1 - (data['Matching Percentage']/100)) * .33))
```

In [322]:
```
# Lyft's cut is calculated by subtracting how much the rider pays which is fixed by 25 by the amount that is paid
## to the driver (Driver Wage)

data['Lyft Cut'] = 25 - data['Driver Wage']
```

In [323]:
```
# The Average Revenue Per User is calculated by multiplying how much Lyft makes per rider (Lyft Cut), by the
## percentage of rides that are completed (Matching Percentage)

data['ARPU'] = data['Lyft Cut'] * data['Matching Percentage']/100
```

In [324]:
```
# The Customer Lifetime Value is calculated using the formula APRU/Churn Rate

data['Customer Lifetime Value'] = round(data['ARPU']/(data['Churn Rate']/100), 2)
```

In [325]:
```
data
```

Out[325]:

	Driver Wage	Matching Percentage	Churn Rate	Lyft Cut	ARPU	Customer Lifetime Value
0	17.0	38.0	24.26	8.0	3.04	12.53
1	18.0	49.0	21.73	7.0	3.43	15.78
2	19.0	60.0	19.20	6.0	3.60	18.75
3	20.0	71.0	16.67	5.0	3.55	21.30
4	21.0	82.0	14.14	4.0	3.28	23.20
5	22.0	93.0	11.61	3.0	2.79	24.03
6	23.0	100.0	10.00	2.0	2.00	20.00

The above table compiles all the information I have collected so far! From this we can see that a **driver wage of $22** has the highest Customer Lifetime Value at **$24.03**.

I could conclude my analysis here, however the metric that I have used as the deciding factor is only based on monthly churn. Our goal is to maximize net revenue over a 12 month span.

This led me to the next step of calculating the Customer Lifetime Value (CLV) over a year. The function below accomplishes this task.

In [326]:
```
# This function calculates the Annual Customer Lifetime Value by factoring in monthly churn for 12 months
# We start with an arbitrary number of riders: 100 and calculate the total revenue generated each month for a year

def annual_clv_calc(df):
    annual_clv_col = []
    riders = 100
    total_rev = 0
    for _ in range(12):
        total_rev += (riders * df['Matching Percentage']/100) * df['Lyft Cut']
        riders = riders * (1 - df['Churn Rate']/100)
    clv = round(total_rev/100, 2)
    df['Annual CLV'] = clv
    return df
```

In [327]:
```
annual_clv_calc(data)
```

Out[327]:

	Driver Wage	Matching Percentage	Churn Rate	Lyft Cut	ARPU	Customer Lifetime Value	Annual CLV
0	17.0	38.0	24.26	8.0	3.04	12.53	12.08
1	18.0	49.0	21.73	7.0	3.43	15.78	14.95
2	19.0	60.0	19.20	6.0	3.60	18.75	17.30
3	20.0	71.0	16.67	5.0	3.55	21.30	18.91
4	21.0	82.0	14.14	4.0	3.28	23.20	19.47
5	22.0	93.0	11.61	3.0	2.79	24.03	18.57
6	23.0	100.0	10.00	2.0	2.00	20.00	14.35

The table above includes an additional column 'Annual CLV' that shows the customer lifetime value for a full year! From this table, I can see that a **driver wage of $21** has the highest Customer Lifetime Value at **$19.47**.

From both of these metrics, it seems that paying Lyft drivers more, results in a higher customer lifetime value which would correlate with more revenue for Lyft.

So far, I have used discrete numbers from $17-$23. However, I would have to consider if there was a non-integer price point that could yield a higher annual CLV. In the code below, I repeat what I have done before, but this time I take in 10,000 unique price points between $17-$23.

In [331]:
```
# Repetition of the process over 10,000 data points between driver wage values between 17-23 to find a continuous
## driver wage value that maximizes the annual CLV

x_line = np.linspace(17, 23, 10000)
y_line = slope * x_line + intercept
y_line = np.minimum(y_line, 100)
data = pd.DataFrame({'Driver Wage': x_line, 'Matching Percentage': y_line})
data['Churn Rate'] = 100 * ((data['Matching Percentage']/100 * .1) + ((1 - (data['Matching Percentage']/100)) * .33))
data['Lyft Cut'] = 25 - data['Driver Wage']
data['ARPU'] = data['Lyft Cut'] * data['Matching Percentage']/100
data['Customer Lifetime Value'] = round(data['ARPU']/(data['Churn Rate']/100), 2)
annual_clv_calc(data)
```

Out[331]:

	Driver Wage	Matching Percentage	Churn Rate	Lyft Cut	ARPU	Customer Lifetime Value	Annual CLV
0	17.0000	38.000000	24.260000	8.0000	3.040000	12.53	12.08
1	17.0006	38.006601	24.258482	7.9994	3.040300	12.53	12.09
2	17.0012	38.013201	24.256964	7.9988	3.040600	12.53	12.09
3	17.0018	38.019802	24.255446	7.9982	3.040900	12.54	12.09
4	17.0024	38.026403	24.253927	7.9976	3.041199	12.54	12.09
...	...	...	...	...	...	...	...
9995	22.9976	100.000000	10.000000	2.0024	2.002400	20.02	14.37
9996	22.9982	100.000000	10.000000	2.0018	2.001800	20.02	14.36
9997	22.9988	100.000000	10.000000	2.0012	2.001200	20.01	14.36
9998	22.9994	100.000000	10.000000	2.0006	2.000600	20.01	14.36
9999	23.0000	100.000000	10.000000	2.0000	2.000000	20.00	14.35

10000 rows × 7 columns

In [332]:
```
filtered_df = data.where(data['Annual CLV'] == data['Annual CLV'].max()).dropna()
```

In [335]:
```
filtered_df['Annual CLV'].mean(), 2), round(filtered_df['Lyft Cut'].mean(), 2), round(filtered_df['Matching Percentage'
```

Out[335]:
```
(20.94, 19.48, 4.06, 81.33)
```

From this expanded dataset, I filtered to find which driver wage corresponded to the maximum annual CLV. A driver wage of **$20.94** has the highest Customer Lifetime Value at **$19.48**.

From our previous experiment, the difference in wage is 6 cents and the difference in CLV is a gain of 1 cent.

Part 2: Simulation and Validation

Roadmap: In this section, my objective is to simulate a year of Lyft rides at different data points, and compare them to see which results in the highest revenue.

This simulation is designed and based on a few assumptions.

1. Initial Driver and Rider Numbers: Drivers complete 100 rides a month, and riders on average request 1 ride a month. This occurs when 60% of rider requests are met with matches. Using this information, I came up with a starting number of 16667 riders and 100 drivers, since 100 riders/ driver would be 10,000 rides, roughly 60% of the number of riders.
2. Rider and Driver Monthly Growth: Each month, the number of riders grows by 15%-25% and the number of drivers grows by 5%-10% before accounting for churn. These were arbitrarily chosen growth values.
3. Randomness: In the real world, the numbers aren't exact, and to emulate this the number of ride requests, the matches, and the percentages of rider and driver growth all have some randomness introduced to them.

The simulation will be run for 3 price points, and I will compare net revenue and total number of drivers. They are as follows:

- Driver Wage of $19 (Lyft Cut $6)
- Driver Wage of $21 (Lyft Cut $4)
- Driver Wage of $20.94 (Lyft Cut $4.06)

In [352]:
```
def simulation(num_months, match_percentage, lyft_cost, lyft_cut):
    metrics = {
    'Month': [],
    'Riders': [],
    'Drivers': [],
    'Rides Requested': [],
    'Rides Completed': [],
    'Churned Riders': [],
    'Churned Drivers': [],
    'New Riders': [],
    'New Drivers': [],
    'Rider Growth %' : [],
    'Driver Growth %': [],
    'Total Revenue': [],
    'Driver Wages': [],
    'Net Revenue': []
    }

    riders = 16667
    drivers = 100
    lyft_wage = lyft_cost - lyft_cut

    for month in range(num_months):
        metrics['Month'].append(month + 1)
        metrics['Riders'].append(riders)
        metrics['Drivers'].append(drivers)

        ride_requests = np.random.normal(loc=1, scale=0.1) * riders  # Average ride requests per rider per month (norma
        matches = np.random.binomial(int(ride_requests), match_percentage)  # Probability of finding a match

        churned_drivers = int(drivers * 0.05)  # Churn rate for drivers
        churned_riders_match = int(matches * 0.1)  # Churn rate for riders who found a match
        churned_riders_no_match = int((ride_requests - matches) * 0.33)  # Churn rate for riders who didn't find a matc
        churned_riders = churned_riders_match + churned_riders_no_match
        churn_rate = churned_riders/riders

        rider_growth = np.random.uniform(0.15, 0.25)
        driver_growth = np.random.uniform(0.05, 0.10)
        new_riders = int(riders * rider_growth)
        new_drivers = int(drivers * driver_growth)

        riders = riders + new_riders - churned_riders
        drivers = drivers + new_drivers - churned_drivers

        total_revenue = matches * lyft_cost
        driver_wage = matches * lyft_wage

        net_revenue = total_revenue - driver_wage

        metrics['Rides Requested'].append(int(ride_requests))
        metrics['Rides Completed'].append(matches)
        metrics['Churned Riders'].append(churned_riders)
        metrics['Churned Drivers'].append(churned_drivers)
        metrics['New Riders'].append(new_riders)
        metrics['New Drivers'].append(new_drivers)
        metrics['Rider Growth %'].append(round(rider_growth*100, 2))
        metrics['Driver Growth %'].append(round(driver_growth*100, 2))
        metrics['Total Revenue'].append(total_revenue)
        metrics['Driver Wages'].append(driver_wage)
        metrics['Net Revenue'].append(round(net_revenue, 2))

    df = pd.DataFrame(metrics)
    return df
```

In [364]:
```
df1 = simulation(12, 0.6, 25, 6)
df2 = simulation(12, 0.82, 25, 4)
df3 = simulation(12, 0.8133, 25, 4.06)
```

In [366]:
```
df1
```

Out[366]:

	Month	Riders	Drivers	Rides Requested	Rides Completed	Churned Riders	Churned Drivers	New Riders	New Drivers	Rider Growth %	Driver Growth %	Total Revenue	Driver Wages	Net Revenue
0	1	16667	100	16612	9999	3181	5	2553	7	15.32	7.58	249975	189981	59994
1	2	16039	102	16927	10081	3267	5	3545	8	22.11	8.68	252025	191539	60486
2	3	16317	105	16443	9890	3151	5	2749	7	16.85	7.16	247250	187910	59340
3	4	15915	107	16963	10208	3249	5	3616	9	22.73	8.76	255200	193952	61248
4	5	16282	111	15024	8989	2889	5	3878	10	23.82	9.02	224725	170791	53934
5	6	17271	116	18552	11125	3563	5	2904	9	16.82	8.47	278125	211375	66750
6	7	16612	120	20409	12260	3915	6	3483	8	20.97	7.15	306500	232940	73560
7	8	16180	122	15663	9257	3039	6	2780	8	17.18	7.12	231425	175883	55542
8	9	15921	124	14875	8959	2847	6	2487	10	15.62	8.40	223975	170221	53754
9	10	15561	128	15056	9010	2896	6	2519	6	16.19	5.05	225250	171190	54060
10	11	15184	128	16590	9969	3181	6	2512	9	16.54	7.35	249225	189411	59814
11	12	14515	131	13323	7986	2559	6	3029	9	20.87	7.25	199650	151734	47916

In [368]:
```
df2
```

Out[368]:

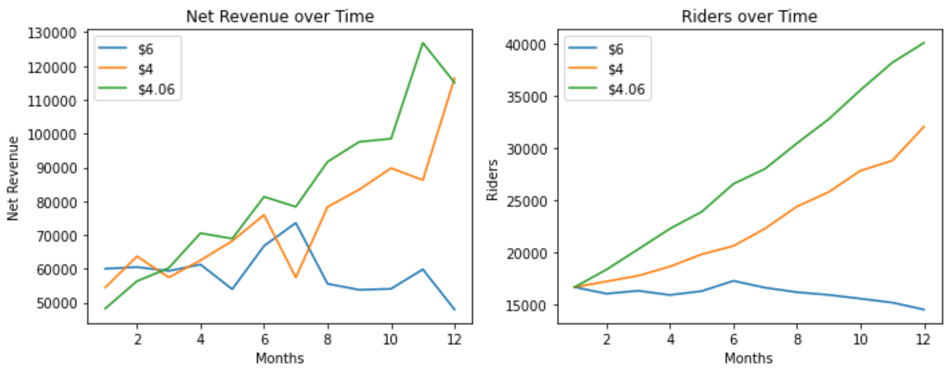
	Month	Riders	Drivers	Rides Requested	Rides Completed	Churned Riders	Churned Drivers	New Riders	New Drivers	Rider Growth %	Driver Growth %	Total Revenue	Driver Wages	Net Revenue
0	1	16667	100	16630	13606	2358	5	2899	9	17.40	9.55	340150	285726	54424
1	2	17208	104	19454	15919	2757	5	3304	8	19.20	8.44	397975	334299	63676
2	3	17755	107	17468	14360	2461	5	3345	10	18.84	9.84	359000	301560	57440
3	4	18639	112	19095	15623	2707	5	3883	7	20.84	6.95	390575	328083	62492
4	5	19815	114	20798	17058	2939	5	3735	10	18.85	8.84	426450	358218	68232
5	6	20611	119	23120	18981	3264	5	4944	6	23.99	5.83	474525	398601	75924
6	7	22291	120	17377	14351	2433	6	4540	7	20.37	6.64	358775	301371	57404
7	8	24398	121	23969	19572	3408	6	4799	7	19.67	6.19	489300	411012	78288
8	9	25789	122	25442	20859	3597	6	5643	9	21.88	7.55	521475	438039	83436
9	10	27835	125	27449	22443	3896	6	4862	10	17.47	8.27	561075	471303	89772
10	11	28801	129	26266	21557	3709	6	6969	11	24.20	8.78	538925	452697	86228
11	12	32061	134	35207	29097	4925	6	6045	12	18.86	9.32	727425	611037	116388

In [367]:
```
df3
```

Out[367]:

	Month	Riders	Drivers	Rides Requested	Rides Completed	Churned Riders	Churned Drivers	New Riders	New Drivers	Rider Growth %	Driver Growth %	Total Revenue	Driver Wages	Net Revenue
0	1	16667	100	14515	11874	2058	5	3732	8	22.39	8.64	296850	248641.56	48208.44
1	2	18341	103	16964	13873	2407	5	4362	6	23.79	5.90	346825	290500.62	56324.38
2	3	20296	104	18192	14830	2592	5	4561	9	22.48	9.32	370750	310540.20	60209.80
3	4	22265	108	21312	17371	3037	5	4665	7	20.96	7.04	434275	363748.74	70526.26
4	5	23893	110	20975	16971	3018	5	5693	10	23.83	9.48	424275	355372.74	68902.26
5	6	26568	115	24470	20025	3469	5	4925	8	18.54	7.55	500625	419323.50	81301.50
6	7	28024	118	23722	19294	3390	5	5830	11	20.81	9.90	482350	404016.36	78333.64
7	8	30464	124	27821	22575	3988	6	6302	11	20.69	9.06	564375	472720.50	91654.50
8	9	32778	129	29458	24032	4193	6	6989	7	21.32	6.04	600800	503230.08	97569.92
9	10	35574	130	29816	24255	4260	6	6888	12	19.36	9.79	606375	507899.70	98475.30
10	11	38202	136	38555	31247	5535	6	7438	12	19.47	8.84	781175	654312.18	126862.82
11	12	40105	142	34821	28341	4972	7	7493	9	18.68	6.77	708525	593460.54	115064.46

In [365]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
axes[0].plot(df1['Month'], df1['Net Revenue'], label='$6')
axes[0].plot(df2['Month'], df2['Net Revenue'], label='$4')
axes[0].plot(df3['Month'], df3['Net Revenue'], label='$4.06')


# Adding labels and title
axes[0].set_xlabel('Months')
axes[0].set_ylabel('Net Revenue')
axes[0].set_title('Net Revenue over Time')

# Adding a legend
axes[0].legend()

axes[1].plot(df1['Month'], df1['Riders'], label='$6')
axes[1].plot(df2['Month'], df2['Riders'], label='$4')
axes[1].plot(df3['Month'], df3['Riders'], label='$4.06')


# Adding labels and title
axes[1].set_xlabel('Months')
axes[1].set_ylabel('Riders')
axes[1].set_title('Riders over Time')

# Adding a legend
axes[1].legend()

plt.tight_layout()
```



In [358]:
```
print("Net Revenue @ $6:", round(sum(df1['Net Revenue']),2))
print("Net Revenue @ $4:", round(sum(df2['Net Revenue']),2))
print("Net Revenue @ $4.06:", round(sum(df3['Net Revenue']),2))
```

```
Net Revenue @ $6: 780444
Net Revenue @ $4: 840160
Net Revenue @ $4.06: 903719.46
```

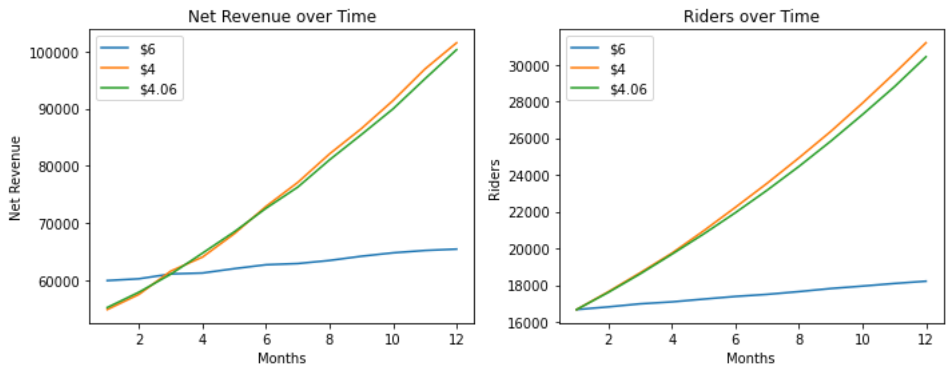
Results: After running my simulation, I found that the price of **$4.06** resulted in the highest net revenue over 12 months! It also correlated with the highest number of riders. This validates our idea from before that a driver Wage of $20.94 (Lyft Cut $4.06) would perform the best.

However, one simulation would not be enough to validate this. Therefore, I created a function that would run the simulation 1,000 times and return the averages. The plots are shown below.

In [359]:
```
def run_simulation(num_simulations, num_months, match_percentage, lyft_cost, lyft_cut):
    simulation_results = []

    for i in range(num_simulations):
        simulation_df = simulation(num_months, match_percentage, lyft_cost, lyft_cut)
        simulation_results.append(simulation_df)

    # Concatenate all the simulation results
    combined_results = pd.concat(simulation_results)

    # Calculate the averages for each value in each month
    averages = combined_results.groupby('Month').mean()

    return averages
```

In [370]:
```
df1 = run_simulation(1000, 12, 0.6, 25, 6)
df2 = run_simulation(1000, 12, 0.82, 25, 4)
df3 = run_simulation(1000, 12, 0.8133, 25, 4.06)
```

In [371]:
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))
axes[0].plot(df1.index, df1['Net Revenue'], label='$6')
axes[0].plot(df2.index, df2['Net Revenue'], label='$4')
axes[0].plot(df3.index, df3['Net Revenue'], label='$4.06')


# Adding labels and title
axes[0].set_xlabel('Months')
axes[0].set_ylabel('Net Revenue')
axes[0].set_title('Net Revenue over Time')

# Adding a legend
axes[0].legend()

axes[1].plot(df1.index, df1['Riders'], label='$6')
axes[1].plot(df2.index, df2['Riders'], label='$4')
axes[1].plot(df3.index, df3['Riders'], label='$4.06')


# Adding labels and title
axes[1].set_xlabel('Months')
axes[1].set_ylabel('Riders')
axes[1].set_title('Riders over Time')

# Adding a legend
axes[1].legend()

plt.tight_layout()
```



In [362]:
```
print("Net Revenue @ $6:", round(sum(df1['Net Revenue']),2))
print("Net Revenue @ $4:", round(sum(df2['Net Revenue']),2))
print("Net Revenue @ $4.06:", round(sum(df3['Net Revenue']),2))
```

```
Net Revenue @ $6: 754137.89
Net Revenue @ $4: 913485.17
Net Revenue @ $4.06: 913071.38
```

The lines in the plots above are much smoother since they are averaged across 1,000 trials. From these plots, the difference in net revenue between the driver wage at $21 and \20.94 is a lot closer with $21 being favored.

Conclusion: In the end, I can conclude that a higher driver wage would be the better choice for maximizing net revenue. I would go with a final driver wage of $21 since that yielded the highest in the end.