Multiple Linear Regression on 2024 Chicago Taxi Data

Capstone Project for Western Governor's University Master of Science, Data Analytics

Submitted by Kimberly Hubacek Student ID: 001249836

A. Research Question

The research question for this capstone project is, “Are trip miles, trip duration, tolls, taxi fare, or type of payment statistically correlated with customers leaving a tip for their cab driver in the city of Chicago?

In 2015, the city of Chicago granted transportation network provider licenses to all drivers operating under the rideshare companies Uber and Lyft. Ever since, Chicago cab companies have faced harsh competition from rideshare services. Not only have their customer shares declined, but many of their drivers are quitting due to driver dissatisfaction and low pay (Knowles, 2017). This data analysis seeks to reveal what factors are correlated with cutomers tipping their drivers. To improve driver satisfaction and keep their drivers from quitting, Chicago taxi companies may choose to provide their drivers with extra bonuses for taking on customer trips that statistically tip less.

A multiple linear regression model will be used in this analysis to see if trip miles, trip duration, tolls, taxi fare, or type of payment statistically correlated with customers leaving a tip for their cab driver.

For the purposes of this analysis, I have established the following null and alternative hypotheses:

Null Hypothesis: Customers tipping their taxi drivers in the city of Chicago is not statistically correlated with trip miles, trip duration, tolls, or taxi fare.

Alternative Hypothesis: Customers tipping their taxi drivers in the city of Chicago is statistically correlated with trip miles, trip duration, tolls, or taxi fare.

B. Data Collection

The data for this project was collected from the city of Chicago’s official website, cityofchicago.org. The city of Chicago has a data portal with numerous official government datasets available for public use. When viewing the Chicago Taxi Trip 2024 dataset’s page, users can click on the “Export" button in the upper right hand corner. This will allow them to download either a standard CSV file or a CSV file created specifically for Microsoft Excel. The dataset’s webpage also has a bar graph comparing the total fares for the months of January and February as well as a preview of the dataset. The Chicago Taxi Trip 2024 consists of taxi ride information for the months of January and February. Each row of data represents one taxi ride. The dataset contains 425,219 rows and 23 columns documenting the following information:

• Trip ID: A unique identifier for the trip.

• Taxi ID: A unique identifier for the taxi.

• Trip Start Timestamp: Date and time the trip started, rounded to the nearest 15 minutes.

• Trip End Timestamp: Date and time the trip ended, rounded to the nearest 15 minutes.

• Trip Seconds: Time of the trip in seconds.

• Trip Miles: Distance of the trip in miles.

• Pickup Census Tract: The Census Tract where the trip began. This column will be blank for locations outside Chicago.

• Dropoff Census Tract: The Census Tract where the trip ended. This column will be blank for locations outside Chicago.

• Pickup Community Area: The Community Area where the trip began. This column will be blank for locations outside Chicago.

• Dropoff Community Area: The Community Area where the trip ended. This column will be blank for locations outside Chicago.

• Fare: The fare for the trip.

• Tips: The amount customers tipped their driver for the trip.

• Tolls: The tolls for the trip.

• Extras: Extra charges for the trip.

• Trip Total: Total cost of the trip, the total of the previous columns.

• Payment Type: Type of payment for the trip.

• Company: The taxi company.

• Pickup Centroid Latitude: The latitude of the center of the pickup census tract or the community area. This column will be blank for locations outside Chicago.

• Pickup Centroid Longitude: The longitude of the center of the pickup census tract or the community area. This column will be blank for locations outside Chicago.

• Pickup Centroid Location: The location of the center of the pickup census tract or the community area. This column will be blank for locations outside Chicago.

• Dropoff Centroid Latitude: The latitude of the center of the dropoff census tract or the community area. This column will be blank for locations outside Chicago.

• Dropoff Centroid Longitude: The longitude of the center of the dropoff census tract or the community area. This column will be blank for locations outside Chicago.

• Dropoff Centroid Location: The location of the center of the dropoff census tract or the community area. This column will be blank for locations outside Chicago.

The primary advantage of this gathering methodology is how simple it is. The data is readily available on the city of Chicago's website and can be downloaded as a CSV file. The CSV file is easily loaded into a Jupyter notebook and read by Python file using Panda's read.csv() function.

The data is also from the city of Chicago's official government webpage, meaning it is authoritative. All taxi companies operating within the city of Chicago must follow local laws and ordinances. Reporting trip information allows taxi companies to receive operating licenses and special contracts within the city. As a result, the information contained in the dataset should be considered authoritative. The city of Chicago also regularly updates their data, with the Chicago taxi dataset receiving monthly updates containing the previous month's data. This allows users to create up to date analyses and observe how the data evolves from month to month, making long term projects possible. The primary disadvantage of this data gathering methodology is I am bound by what the city of Chicago provides. While the data contains the names of taxi companies operating within the city of Chicago, it does not list contact information for these companies. I attempted to contact two of the companies listed in the dataset for more information on what costs were associated with the "Extras" column and neither company responded to me. The data also contains information that has been somewhat altered to protect customer privacy, and there is no way for me to discern what information is completely accurate and what information has been altered.

The rubric asked for me to disclose challenges with collecting the data. I did not have any challenges locating or collecting the data, but I did face significant challenges with preparing the data for analysis. Specifically, there were many null values in the location columns due to taxi trips that started or ended outside of the city of Chicago. The "Trip Seconds" column data seemed to be very inaccurate, and I had a difficult time deciding how to handle outliers since over half of the trip seconds recorded did not make sense in comparison to the other data contained in the row for that taxi ride. This was a surprisingly difficult problem to solve, as multiple approaches to fix the outliers left me with data that still seemed inaccurate. I overcame this challenge by trying multiple outlier detection and imputation methods until I discovered flooring and capping the data provided results that seemed the most accurate within the context of the other data.

C. Tools and Data Extraction

I have chosen to submit my report as a PDF copy of my Jupyter Notebook environment. All my data extraction and preparation code are available to view herein. In addition to the narration in the report, I have commented on my code to provide further information and clarity on my handling of the data and the thought process behind my decisions. The combination of the written report and the annotated code describe the data preparation process in full detail.

Tools Used in Analysis

• Jupyter Notebook version 6.3.0 (Anaconda environment)

• Python programming language version 3.8.8

• The following Python libraries:

o pandas

o NumPy

o MatPlotLib

o Seaborn

o Statsmodel

Jupyter Notebook was used as the programming environment for this analysis. Jupyter Notebook is a fantastic environment for data science projects. It allows users to run code in a nonlinear order, allowing users to compare different lines of code easily and quickly. This can be done by running one version of code in one cell to view the outcome, restarting the kernel, and running an alternative line of code in another cell to compare the outcome with the original code. I used this technique to run multiple versions of code to detect outliers and impute different values for the Trip Seconds column during data preparation. Once I found a code solution to solve this problem, it was easy for me to use Jupyter Notebook's "cut" function to quickly delete the lines of code I was no longer using. One disadvantage to using Jupyter Notebook is the lack of user customization compared to an Integrated Development Environment I(DE) like PyCharm. IDE's allow users to easily change the color of the notebook or the text to run in ADA compliant color schemes or night mode to reduce eye strain. There are no officially supported night modes or alternative text colors for Jupyter notebook.

Anaconda was used to launch Jupyter Notebook and manage Python files and libraries on my laptop. Anaconda as an environment manager is useful because it contains numerous data science software programs and programming environments within one easy to navigate graphic user interface (GUI). This allows users to easily install or uninstall different software libraries. A disadvantage to Anaconda is it often lags or freezes, leading me to have to restart my computer to relaunch a program. Another disadvantage is although Anaconda has a user-friendly GUI, I often had to open PowerShell and work with the command line to downgrade or install specific Python packages required throughout the master's in data analytics degree program. Python was the programming language used in this analysis. Python has simple syntax and intuitive layout allows users to easily read and edit code. Python's ease of use has made it one of the most popular programming languages in the world. Since Python is utilized by more people than R, users have a higher likelihood of finding outside help if they need to troubleshoot their code. Python has numerous modules, tools, and libraries that can handle tasks across multiple platforms and disciplines. Many of the libraries support machine learning and data science tasks. One disadvantage to the Python programming language is it's high memory consumption. This is especially apparent when dealing with large datasets. My laptop is an older model and I often had to restart my computer due to my browser consuming large amounts of RAM to run Jupyter Notebook.

The pandas Python library allowed me to "read" the CSV file and load it into the dataframe. Other dataframe manipulations such as adding or dropping columns were performed using pandas. Pandas contains functions that make manipulating the dataframe simple and intuitive. One disadvantage of the pandas library is code lines can become long and cumbersome when performing complex data manipulations. There were many times throughout my time as a WGU student where I was searching for over an hour to figure out how to execute code that required multiple dataframe manipulations at one time.

While NumPy wasn't used directly in a line of code, it was vital to the analysis. NumPy's mathematical operations are utilized by other packages and libraries in Python, including pandas and Matplotlib. For this analysis, NumPy was used to calculate interquartile ranges for certain columns during data cleaning and was used by Matplotlib and Seaborn during the creation of visualizations. One disadvantage associated with NumPy is it is not flexible, requiring all variables to be the same datatype to run certain calculations. I ran into issues when trying to perform calculations when some data were float64 and other data were int. I had to come up with different approach to run my calculation.

All visualizations were handled with Matplotlib and Seaborn. Matplotlib is built off NumPy and allows users to easily create basic to highly detailed graphs quickly and without using excessive computing resources. Seaborn is built off Matplotlib and acts as an extension to it. Seaborn offers more customization options and statistical functions compared to Matplotlib, but Matplotlib allows users to create interactive and animated graphics. Both libraries make creating detailed visualizations in Python simple. One disadvantage to Matplotlib is it can be complex to use and can be difficult for new users to learn. One disadvantage to Seaborn is it is dependent on Matplotlib, meaning users need to have both libraries installed to use Seaborn (GeeksforGeeks, 2022).

Statsmodels was used to create the multiple linear regression model and perform feature elimination. Statsmodels is a Python module used for running statistical tests and data explorations. The Statsmodels OLS Report provided statistical measurements of the data used to evaluate the multiple linear regression model. Statsmodels variance inflation factor was sued to address multicollinearity issues and eliminate variables that were correlated. One advantage to Statsmodels is allows users to customize algorithms for OLS unlike scikitlearn. One disadvantage to Statsmodels is it does not perform regularization on the data to prevent overfitting like scikitlearn OLS libraries does (Sutton, 2022).

After loading all the Python libraries and packages into Jupyter notebook, I loaded the dataset by using panda's "readcsv" function. I renamed the dataset "taxi" while loading it into Jupyter notebook. I performed exploratory data analysis by viewing the first and last five rows of the dataset using pandas "head" and "tail" functions. I then used pandas "df.info" function to call information about the dataset, including column names and the number of non-null values in each column. Panda's "df.shape" was used to return the total number of rows and columns. The taxi dataset contained 425,219 rows and 23 columns. Finally, I used pandas "df.describe" function to return the summary statistics of the dataset.

In [1]:

```
# Programing Environment: Jupyter Notebook Ver. 6.3, Python ver. 3.8.8
```

In [2]:

```
# Loading libraries and packages

# loading and manipulating dataframe
import pandas as pd
# mathmetical calculations
import numpy as np
# visualizations
import seaborn as sns
import matplotlib.pyplot as plt
# regression model and OLS report
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

```python
from statsmodels.tools.tools import add_constant
# Varaince Inflation Factor to address multicolinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [3]:

```python
# Loading dataset and renaming it 'taxi'
taxi = pd.read_csv("TaxiTrips22024.csv")
```

In [4]:

```python
# Setting the display to preview the maximum number of columns instead of just a few
pd.set_option("display.max_columns", None)
```

In [5]:

```python
taxi.head(5)
```

Out[5]:

	Trip ID	Taxi ID	Trip Start Timestamp	Trip End Timestamp	Trip Seconds	Trip Miles	C
0	0000184e7cd53cee95af32eba49c44e4d20adcd8	f538e6b729d1aaad4230e9dcd9dc2fd9a168826ddadbd6...	1/19/2024 17:00	1/19/2024 18:00	4051.0	17.12	1.
1	000072ee076c9038868e239ca54185eb43959db0	e51e2c30caec952b40b8329a68b498e18ce8a1f40fa75c...	1/28/2024 14:30	1/28/2024 15:00	1749.0	12.70	
2	000074019d598c2b1d6e77fbae79e40b0461a2fc	aeb280ef3be3e27e081eb6e76027615b0d40925b84d3eb...	1/5/2024 9:00	1/5/2024 9:00	517.0	3.39	
3	00007572c5f92e2ff067e6f838a5ad74e83665d3	7d21c2ca227db8f27dda96612bfe5520ab408fa9a462c8...	1/22/2024 8:45	1/22/2024 9:30	2050.0	15.06	
4	00007c3e7546e2c7d15168586943a9c22c3856cf	8ef1056519939d511d24008e394f83e925d2539d668a00...	1/18/2024 19:15	1/18/2024 19:30	1004.0	1.18	1.

In [6]:

```python
taxi.tail(5)
```

Out[6]:

	Trip ID	Taxi ID	Trip Start Timestamp	Trip End Timestamp	Trip Seconds	Trip Miles
425214	ffff03be50d2e1d53c75b272a98878bba9a7e43e	87867da8f769326d50dd5facc1ea7d28eceefb785d5b24...	1/3/2024 6:30	1/3/2024 7:15	2168.0	13.78
425215	ffff15eb6b0994515eab5db3341f529722262c42	ae7a61c41decb6f41d165aba54911ea50c4fbf9f418142...	1/24/2024 10:00	1/24/2024 10:00	360.0	0.00
425216	ffff751bd7c1fd095c4e0496677d8dd7bb289d0a	5adbc97abe353b4ad223abfb31d0311bd30800e15f43a2...	1/23/2024 19:00	1/23/2024 19:15	922.0	3.01
425217	ffffb68c307fdcbbe336270164f12b4c5f7f3878	91f06db58053143c1ed5453714469f839b2b1ed1cdd93f...	1/29/2024 19:15	1/29/2024 19:45	1324.0	14.31
425218	ffffe1fa6eab28ec0c585f0f5705d0d926a349f0	71299f519d81a7d3bd0fd6bb0d0f8ebd553ea3a851d071...	1/12/2024 15:00	1/12/2024 15:00	492.0	1.76

In [7]:

```python
# Info about dataset including column names and non-null values values in each column
taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425219 entries, 0 to 425218
Data columns (total 23 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Trip ID                 425219 non-null  object
 1   Taxi ID                 425219 non-null  object
 2   Trip Start Timestamp    425219 non-null  object
 3   Trip End Timestamp      425219 non-null  object
 4   Trip Seconds            425137 non-null  float64
 5   Trip Miles              425219 non-null  float64
 6   Pickup Census Tract     158338 non-null  float64
 7   Dropoff Census Tract    151451 non-null  float64
 8   Pickup Community Area   414116 non-null  float64
 9   Dropoff Community Area  382796 non-null  float64
 10  Fare                    424205 non-null  float64
```

```
 11  Tips                        424205 non-null  float64
 12  Tolls                       424205 non-null  float64
 13  Extras                      424205 non-null  float64
 14  Trip Total                  424205 non-null  float64
 15  Payment Type                425219 non-null  object
 16  Company                     425219 non-null  object
 17  Pickup Centroid Latitude    414232 non-null  float64
 18  Pickup Centroid Longitude   414232 non-null  float64
 19  Pickup Centroid Location    414232 non-null  object
 20  Dropoff Centroid Latitude   385242 non-null  float64
 21  Dropoff Centroid Longitude  385242 non-null  float64
 22  Dropoff Centroid  Location  385242 non-null  object
dtypes: float64(15), object(8)
memory usage: 74.6+ MB
```

In [8]:

```
# Data frame shape
print(taxi.shape)
```

```
(425219, 23)
```

In [9]:

```
# Data Statistics
taxi.describe().round(2)
```

Out[9]:

	Trip Seconds	Trip Miles	Pickup Census Tract	Dropoff Census Tract	Pickup Community Area	Dropoff Community Area	Fare	Tips	Tolls	Extras	Trip Total	P Cer Lat
count	425137.00	425219.00	1.583380e+05	1.514510e+05	414116.00	382796.00	424205.00	424205.00	424205.00	424205.00	424205.00	4142
mean	1151.36	6.58	1.703152e+10	1.703141e+10	36.96	25.75	22.18	2.78	0.07	2.22	27.42	
std	1639.51	7.91	3.732052e+05	3.391789e+05	26.64	20.59	20.12	4.18	13.15	18.31	37.81	
min	0.00	0.00	1.703101e+10	1.703101e+10	1.00	1.00	0.00	0.00	0.00	0.00	0.00	
25%	455.00	0.90	1.703116e+10	1.703108e+10	8.00	8.00	8.00	0.00	0.00	0.00	10.00	
50%	903.00	3.03	1.703132e+10	1.703132e+10	32.00	28.00	16.00	0.00	0.00	0.00	19.06	
75%	1587.00	11.74	1.703198e+10	1.703184e+10	73.00	32.00	33.75	4.00	0.00	4.00	41.50	
max	86135.00	1626.85	1.703198e+10	1.703198e+10	77.00	77.00	3668.50	200.00	4444.44	5051.10	8912.13	

C. Data Preparation

With the data loaded into Jupyter Notebook and explored, it was time to start cleaning the data. I I checked the dataset for duplicate values using a combination of Pandas' duplicated and sum functions. This returned a table of the columns represented by a number and a bool "True/False" statement which indicated if a column had duplicates. All the columns returned a "False" value, indicating there are no duplicate values in the data set.

In [10]:

```
# Detecting Duplicates
taxi.duplicated().sum
```

Out[10]:

```
<bound method NDFrame._add_numeric_operations.<locals>.sum of 0         False
1         False
2         False
3         False
4         False
          ...
425214    False
425215    False
425216    False
425217    False
425218    False
Length: 425219, dtype: bool>
```

I detected all null values in the data set using a combination of Pandas' isnull and sum functions. This returned the name of all columns along with the total number of null values found in each column. The function showed significant null values in the geolocation columns. Pickup Census Track and Dropoff Census track in particular had over 50% of the columns as null values. Given that was intentional by the creators of the dataset, any attempt at imputing values for these nulls would result in incoorect data. In addition, the location data would not be utilized in the multiple regression model as they did not pertain to my research question. Given these facts, I decided the best course of action would be to drop the columns containing location data, including Pickup Census Tract, Pickup

Centroid Latitude, Pickup Centroid Longitude, Pickup Centroid Location, Drop off Census Tract, Dropoff Centroid Latitude, Dropoff Centroid Longitude, Dropoff Centroid Location. One disadvantage of this decision is it reduced the dimensionality of the data. One advantage of this decision is it maintained the integrity of the data since I was not imputing false values. After dropping the location columns, the number of columns in the dataset was reduced from 23 to 13.

Null Values

In [11]:
```
# Detecting Null Values
taxi.isnull().sum()
```

Out[11]:
```
Trip ID                          0
Taxi ID                          0
Trip Start Timestamp             0
Trip End Timestamp               0
Trip Seconds                    82
Trip Miles                       0
Pickup Census Tract         266881
Dropoff Census Tract        273768
Pickup Community Area        11103
Dropoff Community Area       42423
Fare                          1014
Tips                          1014
Tolls                         1014
Extras                        1014
Trip Total                    1014
Payment Type                     0
Company                          0
Pickup Centroid Latitude     10987
Pickup Centroid Longitude    10987
Pickup Centroid Location     10987
Dropoff Centroid Latitude    39977
Dropoff Centroid Longitude   39977
Dropoff Centroid  Location   39977
dtype: int64
```

In [12]:
```
# dropping columns with GPS coordinates that do not relate to the research question
# dropping Pickup Census Tract, Pickup Centroid Latitude, Pickup Centroid Longitude, Pickup Centroid Location
# dropping Drop off Census Tract, Dropoff Centroid Latitude, Dropoff Centroid Longitude, Dropoff Centroid Locatic
taxi = taxi.drop({'Pickup Census Tract',
                  'Pickup Centroid Latitude',
                  'Pickup Centroid Longitude',
                  'Pickup Centroid Location',
                  'Pickup Community Area',
                  'Dropoff Community Area',
                  'Dropoff Census Tract',
                  'Dropoff Centroid Latitude',
                  'Dropoff Centroid Longitude',
                  'Dropoff Centroid  Location'}, axis='columns')
```

In [13]:
```
# Viewing reduced dataframe
taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 425219 entries, 0 to 425218
Data columns (total 13 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Trip ID               425219 non-null  object 
 1   Taxi ID               425219 non-null  object 
 2   Trip Start Timestamp  425219 non-null  object 
 3   Trip End Timestamp    425219 non-null  object 
 4   Trip Seconds          425137 non-null  float64
 5   Trip Miles            425219 non-null  float64
 6   Fare                  424205 non-null  float64
 7   Tips                  424205 non-null  float64
 8   Tolls                 424205 non-null  float64
 9   Extras                424205 non-null  float64
 10  Trip Total            424205 non-null  float64
 11  Payment Type          425219 non-null  object 
 12  Company               425219 non-null  object 
dtypes: float64(7), object(6)
memory usage: 42.2+ MB
```

In [14]:
```
# Detecting Null Values in reduced data
```

```
taxi.isnull().sum()
```

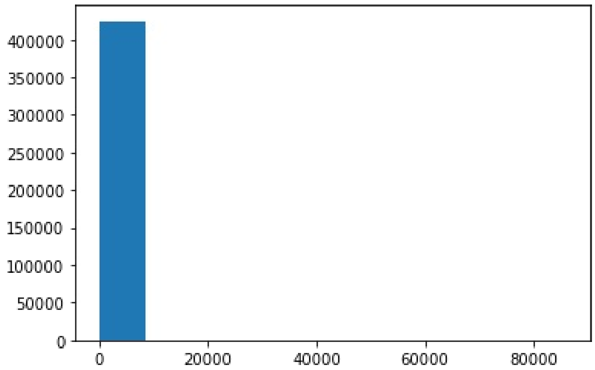
Out[14]:
```
Trip ID                     0
Taxi ID                     0
Trip Start Timestamp        0
Trip End Timestamp          0
Trip Seconds               82
Trip Miles                  0
Fare                     1014
Tips                     1014
Tolls                    1014
Extras                   1014
Trip Total               1014
Payment Type                0
Company                     0
dtype: int64
```

After deleting the columns with location data, there were still nulls present in the Trip Seconds, Fare, Tips, Tolls, Extras and Trip Total columns. I created histograms using Matplotlib's "hist" function to view the distribution of data in the columns. I used this information to determine if I used the mean or median to impute missing values. A normal bell curve distribution would indicate it would be best to impute the mean, while a skewed distribution would indicate it would be best to impute the median. All columns had a skewed distribution. I calculated the median of each column using numpy's ".median" function and imputed the median for the missing values in the Trip Seconds, Fare, Tips, Tolls, Extras and Trip Total columns using pandas "fillna" function. One advantage of imputing the median is it will not change the distribution of the data like imputing the mean. Imputing the mean would coerce the data into more of a bell curve distribution. One disadvantage of imputing the median is it might not be accurate to the data surrounding the missing value.

In [15]:
```
# visualizing the columns with nulls through histograms to view the distribution.
# This will help me decide between mean and median fill
```

In [16]:
```
# Create Histogram for 'Trip Seconds' Column
plt.hist(taxi['Trip Seconds'])
```

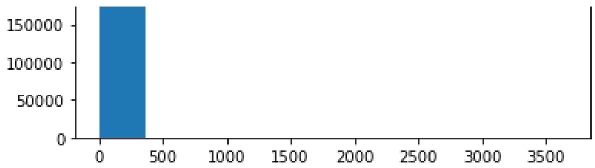
Out[16]:
```
(array([4.24532e+05, 2.73000e+02, 5.50000e+01, 3.00000e+01, 5.40000e+01,
        5.50000e+01, 5.10000e+01, 5.30000e+01, 2.50000e+01, 9.00000e+00]),
 array([    0. ,  8613.5, 17227. , 25840.5, 34454. , 43067.5, 51681. ,
        60294.5, 68908. , 77521.5, 86135. ]),
 <BarContainer object of 10 artists>)
```



In [17]:
```
# distribution of Trip Seconds is skewed right
```

In [18]:
```
# Create Histogram for 'Fare' Column
plt.hist(taxi['Fare'])
```

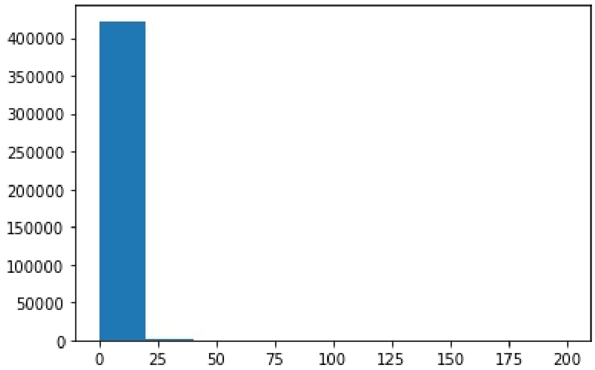
Out[18]:
```
(array([4.24147e+05, 4.50000e+01, 5.00000e+00, 5.00000e+00, 1.00000e+00,
        0.00000e+00, 1.00000e+00, 0.00000e+00, 0.00000e+00, 1.00000e+00]),
 array([   0.  ,  366.85,  733.7 , 1100.55, 1467.4 , 1834.25, 2201.1 ,
        2567.95, 2934.8 , 3301.65, 3668.5 ]),
 <BarContainer object of 10 artists>)
```



In [19]:
```
# distibution of Fare is skewed right
```

In [20]:
```
# Create Histogram for 'Tips' Column
plt.hist(taxi['Tips'])
```

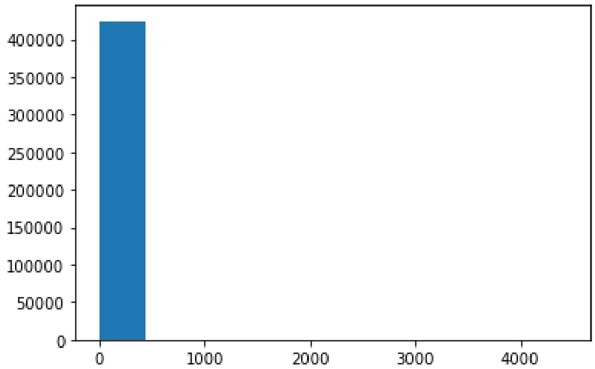
Out[20]:
```
(array([4.22218e+05, 1.80800e+03, 1.27000e+02, 2.20000e+01, 1.80000e+01,
        9.00000e+00, 0.00000e+00, 2.00000e+00, 0.00000e+00, 1.00000e+00]),
 array([  0.,  20.,  40.,  60.,  80., 100., 120., 140., 160., 180., 200.]),
 <BarContainer object of 10 artists>)
```



In [21]:
```
# distribution of Tips is skewed right
```

In [22]:
```
# Create Histogram for 'Tolls' Column
plt.hist(taxi['Tolls'])
```

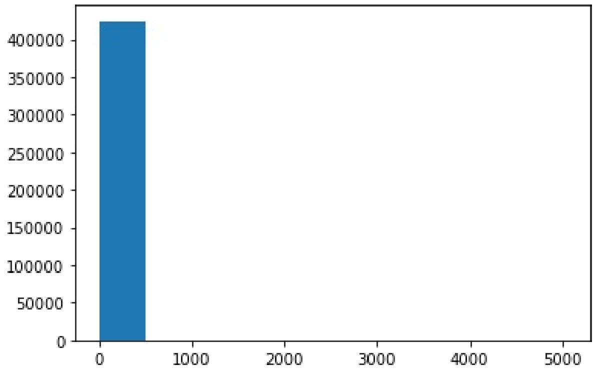
Out[22]:
```
(array([4.242e+05, 0.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 1.000e+00, 0.000e+00, 3.000e+00]),
 array([   0.   ,  444.444,  888.888, 1333.332, 1777.776, 2222.22 ,
        2666.664, 3111.108, 3555.552, 3999.996, 4444.44 ]),
 <BarContainer object of 10 artists>)
```



In [23]:
```
# distribution of Tolls is skewed right
```

In [24]:
```
# Create Histogram for 'Extras' Column
plt.hist(taxi['Extras'])
```

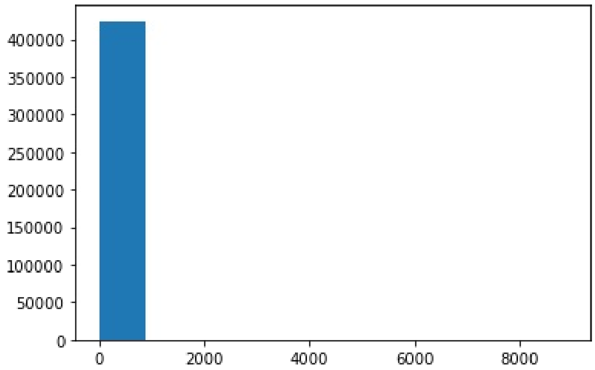
Out[24]:
```
(array([4.24197e+05, 0.00000e+00, 1.00000e+00, 1.00000e+00, 0.00000e+00,
        0.00000e+00, 1.00000e+00, 0.00000e+00, 3.00000e+00, 2.00000e+00]),
 array([   0.  ,  505.11, 1010.22, 1515.33, 2020.44, 2525.55, 3030.66,
        3535.77, 4040.88, 4545.99, 5051.1 ]),
 <BarContainer object of 10 artists>)
```



In [25]:
```
# distribution of Extras is skewed right
```

In [26]:
```
# Create Histogram for 'Trip Total' Column
plt.hist(taxi['Trip Total'])
```

Out[26]:
```
(array([4.24187e+05, 9.00000e+00, 2.00000e+00, 0.00000e+00, 1.00000e+00,
        2.00000e+00, 0.00000e+00, 1.00000e+00, 0.00000e+00, 3.00000e+00]),
 array([   0.   ,  891.213, 1782.426, 2673.639, 3564.852, 4456.065,
        5347.278, 6238.491, 7129.704, 8020.917, 8912.13 ]),
 <BarContainer object of 10 artists>)
```



In [27]:
```
# distribution of Trip Total is skewed right
```

In [28]:
```
# Since data is skewed, median will be used to impute missing values for nulls
```

In [29]:
```
# Fill nulls with median for 'Trip Seconds' Column
taxi['Trip Seconds'].fillna(taxi['Trip Seconds'].median(), inplace=True)
```

In [30]:
```
# Fill nulls with median for 'Fare' Column
taxi['Fare'].fillna(taxi['Fare'].median(), inplace=True)
```

In [31]:
```
# Fill nulls with median for 'Tips' Column
taxi['Tips'].fillna(taxi['Tips'].median(), inplace=True)
```

In [32]:
```
# Fill nulls with median for 'Tolls' Column
taxi['Tolls'].fillna(taxi['Tolls'].median(), inplace=True)
```

In [33]:
```
# Fill nulls with median for 'Extras' Column
taxi['Extras'].fillna(taxi['Extras'].median(), inplace=True)
```

In [34]:
```
# Fill nulls with median for 'Trip Total' Column
taxi['Trip Total'].fillna(taxi['Trip Total'].median(), inplace=True)
```

In [35]:
```
# Rechecking for null values after imputation
taxi.isnull().sum()
```

Out[35]:
```
Trip ID                 0
Taxi ID                 0
Trip Start Timestamp    0
Trip End Timestamp      0
Trip Seconds            0
Trip Miles              0
Fare                    0
Tips                    0
Tolls                   0
Extras                  0
Trip Total              0
Payment Type            0
Company                 0
dtype: int64
```

Addressing Outliers

After addressing null values, I created box plots of the Trip Seconds, Trip Miles, Fare, Tips, Tolls, Extras, and Trip Total columns using Seaborn's "boxplot" function. After creating a boxplot I called the rows with the highest value for each column to view the data of the entire row using a combination of panda's groupby().max() functions. Each column contained outliers.

Trip Seconds

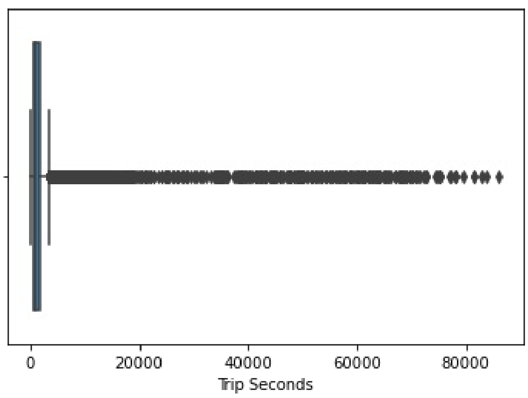
Trip Seconds was a particularly problematic column. Trip seconds depicted the number of seconds the taxi trip lasted. The values ranged from 0 seconds to 86,135 seconds, or 23 hours, 55 minutes, and 35 seconds. Obviously, a taxi ride wouldn't last almost a full 24 hours and the largest values in the column were likely due to data entry errors or equipment malfunctions. The miles travelled and fares did not add up with the high trip seconds values. For example, the taxi trip lasting 86,135 seconds or nearly 24 hours had 2.42 trip miles recorded with a fare of $38.75. Rows with low values in the Trip Seconds column had similar issues, with a taxi trips recorded as lasting from a range of 0 seconds to 4 seconds had Trip miles ranging from 9.75 miles to 17.96 miles. The fares for these trips varied wildly. At this point it was clear to me that the data in the Trip Seconds column was inaccurate and unreliable in context with the other data values.

I decided to floor and cap in the Trip Seconds column to address outliers using NumPy mathematical functions observed in the code below. Flooring and capping drops rows containing outliers with values below a user set lower quantile of data and above a user set higher quantile of the data values within a column. I set my lower quantile at the 10th percentile and my upper quantile at the 90th percentile. An advantage of choosing these percentiles is outliers with data values far removed from the mean will be removed while minimizing the number of rows dropped from the dataset. A disadvantage of choosing 10th percentile instead of 25th percentile for my lower bound and 90th percentile instead of 75th percentile for my upper bound is outliers may still be present in the data after flooring and capping. Flooring and capping are best for data with extreme outliers far removed from the mean since this method can introduce bias if the outliers are not extreme (Magaga, 2021). The Trip Seconds data contains extreme outliers and is suitable for flooring and capping. Flooring and capping the outliers in the Trip Seconds column dropped a total of 1,472 rows from the dataset, which is less than 5% of the total number of rows in the dataset. The remaining number of rows in the dataset after flooring and capping Trip Seconds was 423,747.

Rechecking Trip Seconds for outliers using Seaborn boxplots and calling the rows with the highest values revealed outliers were still present in the Trip Seconds column, but they made sense within the context of the data in the row. For example, the highest value in Trip Seconds was 5198 seconds, or 1 hour and 26 minutes with 24.23 recorded trip miles and a fare of $69.75. These are all reasonable values and seem legitimate, especially if the taxi trip was taken during rush hour. Since the remaining outliers appear legitimate and make sense, they will be retained.

In [36]:
```python
# Boxplot to detect outliers in Trip Seconds column
sns.boxplot(x='Trip Seconds', data=taxi)
```

Out[36]: <AxesSubplot:xlabel='Trip Seconds'>



In [37]:

```
# Calling rows with highest Trip Seconds to view data
taxi.groupby('Trip Seconds').max(10)
```

Out[37]:

Trip Seconds	Trip Miles	Fare	Tips	Tolls	Extras	Trip Total
0.0	664.90	1251.25	80.00	8.0	267.00	1251.25
1.0	11.89	90.00	22.62	0.0	5.00	113.12
2.0	17.96	416.00	33.82	0.0	31.25	416.50
3.0	9.75	300.00	33.10	0.0	18.50	300.00
4.0	12.83	400.00	80.50	0.0	5.50	483.00
...	...	...	...	...	...	...
81540.0	8.36	15.25	0.00	0.0	0.00	15.25
81552.0	3.00	12.50	0.00	0.0	0.00	12.50
83040.0	53.77	153.25	0.00	0.0	0.00	153.25
83900.0	5.95	83.75	0.00	0.0	0.00	83.75
86135.0	2.42	38.75	0.00	0.0	0.00	38.75

6076 rows × 6 columns

In [38]:

```
# Calling rows with highest Trip Seconds
out = taxi['Trip Seconds'].sort_values(ascending=False)
print(out)
```

```
54126     86135.0
412365    83900.0
331954    83040.0
39579     81552.0
388258    81540.0
           ...
7624          0.0
381634        0.0
300071        0.0
7633          0.0
72582         0.0
Name: Trip Seconds, Length: 425219, dtype: float64
```

In [39]:

```
# The Trip Seconds Column contain many outliers that do not make sense
# There are many data points above 10,000 seconds which would be a 2 hr 40min taxi ride
# Flooring and Capping due to data skew
```

In [40]:

```
# Data frame shape
print(taxi.shape)
```

```
(425219, 13)
```

In [41]:

```
# Flooring and Capping Trip Seconds
Q1 = taxi['Trip Seconds'].quantile(0.10)
Q3 = taxi['Trip Seconds'].quantile(0.90)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
index=taxi['Trip Seconds'][(taxi['Trip Seconds']>upper_whisker)|(taxi['Trip Seconds']<lower_whisker)].index
taxi.drop(index, inplace = True)
```

In [42]:

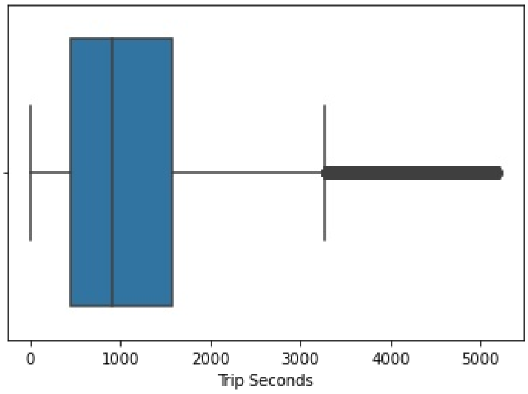
```
# Data frame shape after flooring and capping Trip Seconds
print(taxi.shape)
```

```
(423747, 13)
```

In [43]:
```
# Flooring and capping dropped 1,472 rows from dataset, which is less than 1% of the total data
# limit for dropping rows is 5% of total data, so this is okay
```

In [44]:
```
# Rechecking for outliers with box plot
sns.boxplot(x='Trip Seconds', data=taxi)
```

Out[44]: <AxesSubplot:xlabel='Trip Seconds'>



In [45]:
```
# Calling rows with highest Trip Seconds to view data after flooring and capping
taxi.groupby('Trip Seconds').max(10)
```

Out[45]:

Trip Seconds	Trip Miles	Fare	Tips	Tolls	Extras	Trip Total
0.0	664.90	1251.25	80.00	8.0	267.00	1251.25
1.0	11.89	90.00	22.62	0.0	5.00	113.12
2.0	17.96	416.00	33.82	0.0	31.25	416.50
3.0	9.75	300.00	33.10	0.0	18.50	300.00
4.0	12.83	400.00	80.50	0.0	5.50	483.00
...	...	...	...	...	...	...
5193.0	24.81	72.25	10.60	0.0	5.50	72.25
5198.0	52.83	124.50	0.00	0.0	0.00	124.50
5199.0	11.70	39.25	8.75	0.0	4.00	52.50
5200.0	17.60	60.00	0.00	0.0	0.00	60.00
5205.0	22.33	57.00	0.00	1.9	4.00	62.90

4889 rows × 6 columns

In [46]:
```
# While outliers still exist after flooring and capping, they make sense within the context of the data and will
```

Trip Miles

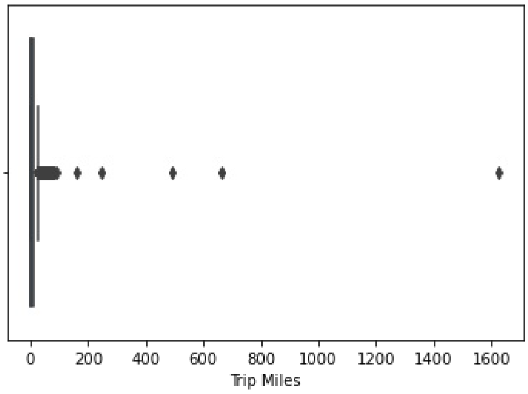
Trip Miles also contained outliers. Calling the rows with the highest values in Trip Miles revealed the rows with the highest Trip Miles values conflicted with the other data in the row. For example, a row with 664.9 reported trip miles had 0 Trip Seconds recorded and a fare of $41.25. Some rows with low Trip Miles recorded had high Trip Seconds and Fares, but this makes sense as these trip could have been to O'Hare airport with the passenger was waiting in the cab to pick someone up after their flight with the meter running. Looking at the data revealed most of the extreme outliers contained recorded Trip Miles above 158. While 158 miles seems like a far distance to travel in a cab, it could have been someone commuting from Wisconsin or Indiana to downtown Chicago for work, or taking a taxi to these location from an airport in Chicago. I decided to drop rows with recoded Trip Miles above 158 miles by using panda's "drop" function and setting the parameters to drop rows containing values outside the range of 0 to 158. This removed outliers that were caused by data entry error or equipment malfunction while maintaining the integrity of the remaining data. A disadvantage to this approach is it reduced the size of the dataset. Dropping rows with recorded trip miles above 158 miles decreased the dataset by 5 rows. The newly reduced taxi dataset contained 423742 rows.

After dropping rows containing extreme outliers from Trip Miles. I rechecked for outliers by creating another boxplot and calling the rows with the highest values. The Trip Miles ranged from 0 to 92.9. While the boxplot revealed outliers were present, all outliers made sense within the context of the remaining data in that row and appeared legitimate. They were retained.

In [47]:
```
# Boxplot to detect outliers in Trip Miles column
```

```
# Boxplot to detect outliers in Trip Miles column
sns.boxplot(x='Trip Miles', data=taxi)
```

Out[47]: <AxesSubplot:xlabel='Trip Miles'>



In [48]:

```
# Calling rows with highest Trip Miles
out = taxi['Trip Miles'].sort_values(ascending=False)
print(out)
```

```
183089    1626.85
70422      664.90
217993     493.03
181832     244.20
282716     159.00
            ...
123909       0.00
365157       0.00
198491       0.00
198492       0.00
321971       0.00
Name: Trip Miles, Length: 423747, dtype: float64
```

In [49]:

```
# Calling rows with highest Trip Miles to view data
taxi.groupby('Trip Miles').max(10)
```

Out[49]:

Trip Miles	Trip Seconds	Fare	Tips	Tolls	Extras	Trip Total
0.00	5160.0	1525.00	100.00	4444.44	5051.1	8896.63
0.01	4943.0	110.00	25.65	2.50	67.0	111.15
0.02	4313.0	90.00	25.00	4.00	57.0	113.12
0.03	4153.0	120.00	24.10	0.01	99.5	144.60
0.04	2204.0	90.00	18.10	0.00	87.0	108.60
...	...	...	...	...	...	...
159.00	1380.0	39.50	8.80	0.00	4.0	52.30
244.20	60.0	8.85	0.00	0.00	0.0	8.85
493.03	1788.0	32.00	7.30	0.00	4.0	43.80
664.90	0.0	41.25	0.00	0.00	4.0	45.25
1626.85	56.0	3668.50	0.00	0.00	0.0	3668.50

4091 rows × 6 columns

In [50]:

```
# Data frame shape
print(taxi.shape)
```

```
(423747, 13)
```

In [51]:

```
# dropping rows in Trip Miles with extreme outliers
taxi = taxi.drop(taxi[(taxi['Trip Miles'] < 0) | (taxi['Trip Miles'] > 158)].index)
```

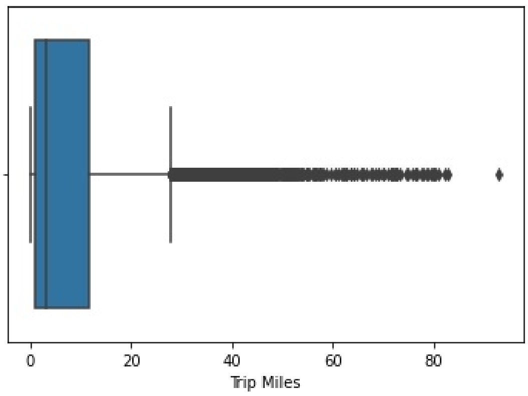
In [52]:
```
# Data frame shape after dropping rows in Trip Miles with extreme outliers
print(taxi.shape)
```

```
(423742, 13)
```

In [53]:
```
# 5 rows were dropped from the dataset
```

In [54]:
```
# Boxplot to recheck outliers in Trip Miles
sns.boxplot(x='Trip Miles', data=taxi)
```

Out[54]: <AxesSubplot:xlabel='Trip Miles'>



In [55]:
```
# Calling rows with highest Trip Miles to view data
taxi.groupby('Trip Miles').max(10)
```

Out[55]:

Trip Miles	Trip Seconds	Fare	Tips	Tolls	Extras	Trip Total
0.00	5160.0	1525.00	100.00	4444.44	5051.10	8896.63
0.01	4943.0	110.00	25.65	2.50	67.00	111.15
0.02	4313.0	90.00	25.00	4.00	57.00	113.12
0.03	4153.0	120.00	24.10	0.01	99.50	144.60
0.04	2204.0	90.00	18.10	0.00	87.00	108.60
...	...	...	...	...	...	...
80.29	903.0	1367.25	0.00	0.00	2.00	1369.25
80.96	4457.0	185.75	10.00	0.00	102.75	299.50
82.00	0.0	189.50	0.00	0.00	99.50	289.00
82.62	5005.0	190.25	0.00	0.00	99.50	290.75
92.90	4680.0	212.25	0.00	0.00	0.00	212.25

4086 rows × 6 columns

In [56]:
```
# After dropping extreme outliers, all remaining data makes sense in context
# Example: Trip Miles, Trip Seconds, and Fare are all plausible numbers
# All other "outliers" will be retained
```

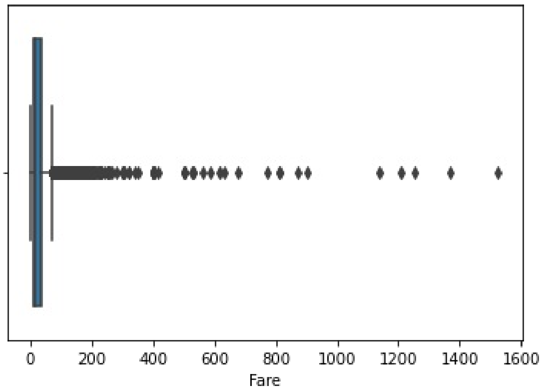
Fare

The outliers in the Fare column do not appear to be legitimate. The data in Trip Seconds and Trip Miles did not align with the high Fare values in rows with outliers. Rows with abnormally low Fare values also did not contribute to the story the rest of the data the row was telling. For example, a row with a recorded fare of a penny had 42.84 recorded trip miles and 5,160 Trip Seconds. On the high end, many rows with recorded trip Fares above $1,000 had zero values for Trip Seconds, and Trip Miles. Flooring and capping were used to address outliers in Fare. Like Trip Seconds, I set my lower quantile at the 10th percentile of the data and my upper quantile at the 90th percentile of the data. Flooring and capping dropped 696 rows from the dataset which is less than 1% of the total data. The newly reduced dataset contained 423,046 rows.

Rechecking for outliers in Fare using boxplots indicated outliers were still present, but printing the rows with the highest values in Fare revealed they were legitimate. All remaining outliers were retained.

In [57]:
```
# Boxplot to detect outliers in Fare column
sns.boxplot(x='Fare', data=taxi)
```

Out[57]: <AxesSubplot:xlabel='Fare'>



In [58]:
```
# Calling rows with highest Fare
out = taxi['Fare'].sort_values(ascending=False)
print(out)
```

```
167310    1525.00
405197    1367.25
206358    1251.25
136249    1206.00
83329     1139.25
           ...
348401       0.00
212756       0.00
251856       0.00
178645       0.00
176867       0.00
Name: Fare, Length: 423742, dtype: float64
```

In [59]:
```
# Calling rows with highest Fare to view data
taxi.groupby('Fare').max(10)
```

Out[59]:

Fare	Trip Seconds	Trip Miles	Tips	Tolls	Extras	Trip Total
0.00	5160.0	34.37	12.10	0.0	60.0	72.10
0.01	4196.0	42.84	0.00	0.0	0.0	0.01
0.02	4612.0	11.52	0.00	0.0	0.0	0.02
0.05	5190.0	45.78	0.11	0.0	0.0	0.66
0.09	60.0	0.00	0.00	0.0	0.0	0.09
...	...	...	...	...	...	...
1139.25	0.0	0.00	0.00	0.0	0.0	1139.25
1206.00	903.0	26.19	0.00	0.0	0.0	1206.00
1251.25	0.0	0.00	0.00	0.0	0.0	1251.25
1367.25	903.0	80.29	0.00	0.0	2.0	1369.25
1525.00	2946.0	0.00	0.00	0.0	0.0	1525.00

4504 rows × 6 columns

In [60]:
```
# Outliers in Fare are not legimate.
# Trip Second and trip miles do not account for high fares
```

In [61]:
```
# Data frame shape
print(taxi.shape)
```

(423742, 13)

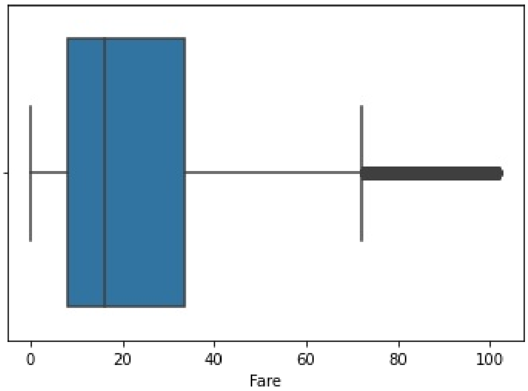
In [62]:
```
# Flooring and Capping Fare to address outliers
Q1 = taxi['Fare'].quantile(0.10)
Q3 = taxi['Fare'].quantile(0.90)
IQR = Q3 - Q1
whisker_width = 1.5
lower_whisker = Q1 - (whisker_width*IQR)
upper_whisker = Q3 + (whisker_width*IQR)
index=taxi['Fare'][(taxi['Fare']>upper_whisker)|(taxi['Fare']<lower_whisker)].index
taxi.drop(index, inplace = True)
```

In [63]:
```
# Data frame shape after dropping rows in Fare with extreme outliers
print(taxi.shape)
```

(423046, 13)

In [64]:
```
# Rechecking for outliers with box plot
sns.boxplot(x='Fare', data=taxi)
```

Out[64]: <AxesSubplot:xlabel='Fare'>



In [65]:
```
# Calling rows with highest Fare to view data
taxi.groupby('Fare').max(10)
```

Out[65]:

Fare	Trip Seconds	Trip Miles	Tips	Tolls	Extras	Trip Total
0.00	5160.0	34.37	12.10	0.0	60.0	72.10
0.01	4196.0	42.84	0.00	0.0	0.0	0.01
0.02	4612.0	11.52	0.00	0.0	0.0	0.02
0.05	5190.0	45.78	0.11	0.0	0.0	0.66
0.09	60.0	0.00	0.00	0.0	0.0	0.09
...	...	...	...	...	...	...
101.00	3460.0	42.46	0.00	0.0	9.5	111.00
101.25	4076.0	43.34	55.00	5.0	62.5	197.70
101.50	4500.0	43.80	22.65	6.0	58.0	183.00
101.75	3000.0	43.43	32.03	1.9	65.0	192.18
102.00	5177.0	43.70	20.60	5.0	61.0	189.00

4225 rows × 6 columns

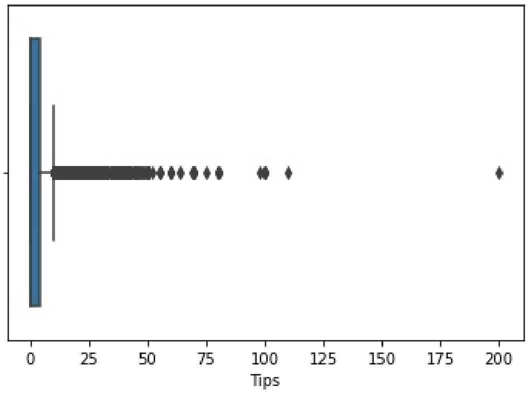
In [66]:
```
# All of the highest Fare rows appear legitimate and will be retained
```

Tips

There was one outlier in Tips. Tipping is subjective and all values but one were in a range of \$0 - \$110, with the largest tip being 200 dollars. The 200 dollar tip might have been legitimate or a data entry error. I decided to delete the row with the 200 dollar tip by dropping all values outside a range of $0-$110 using panda's "drop" function and setting my range. Multiple linear regression is sensitive to outliers. Eliminating an outlier in our target variable will increase the accuracy of our model, which is a huge advantage in this analysis. The disadvantage of eliminating the 200 dollar tip from the dataset is it reduces the size of the data. Dropping the row with a 200 dollar tip reduced the dataframe to 423,045 rows.

In [67]:
```
# Boxplot to detect outliers in Tips column
sns.boxplot(x='Tips', data=taxi)
```

Out[67]: <AxesSubplot:xlabel='Tips'>



In [68]:
```
# Calling rows with highest Tips to view data
taxi.groupby('Tips').max(10)
```

Out[68]:

Tips	Trip Seconds	Trip Miles	Fare	Tolls	Extras	Trip Total
0.00	5205.0	67.80	102.00	4444.44	5051.1	8912.13
0.01	3840.0	41.30	97.50	0.00	58.5	156.01
0.02	3455.0	30.30	85.50	2.00	45.0	130.52
0.03	2520.0	19.12	47.25	0.00	29.5	76.78
0.04	3812.0	19.88	51.75	0.00	17.0	57.29
...	...	...	...	...	...	...
80.00	1500.0	17.50	80.00	0.00	51.0	174.50
98.00	60.0	0.30	3.75	0.00	99.5	201.25
100.00	780.0	7.00	19.50	0.00	0.0	119.50
110.00	1800.0	17.30	43.50	0.00	4.0	157.50
200.00	1260.0	17.90	44.00	0.00	0.0	244.00

2078 rows × 6 columns

In [69]:
```
# $200 tip is an outlier. It will be dropped from dataframe
```

In [70]:
```
# Data frame shape
print(taxi.shape)
```

```
(423046, 13)
```

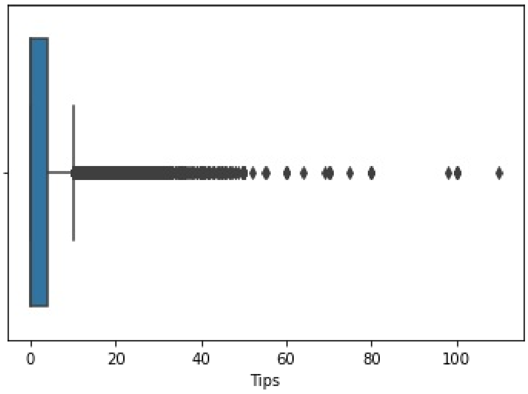
In [71]:
```
# dropping rows in Tips with extreme outliers
taxi = taxi.drop(taxi[(taxi['Tips'] < 0) | (taxi['Tips'] > 110)].index)
```

In [72]:
```
# Data frame shape after dropping outlier in Tips
print(taxi.shape)
```

```
(423045, 13)
```

In [73]:
```python
# Rechecking boxplot to detect outliers in Tips column
sns.boxplot(x='Tips', data=taxi)
```

Out[73]: <AxesSubplot:xlabel='Tips'>



In [74]:
```python
# Calling rows with highest Tips to view data
taxi.groupby('Tips').max(10)
```

Out[74]:

Tips	Trip Seconds	Trip Miles	Fare	Tolls	Extras	Trip Total
0.00	5205.0	67.80	102.00	4444.44	5051.1	8912.13
0.01	3840.0	41.30	97.50	0.00	58.5	156.01
0.02	3455.0	30.30	85.50	2.00	45.0	130.52
0.03	2520.0	19.12	47.25	0.00	29.5	76.78
0.04	3812.0	19.88	51.75	0.00	17.0	57.29
...	...	...	...	...	...	...
75.00	2638.0	31.38	75.25	0.00	10.5	161.25
80.00	1500.0	17.50	80.00	0.00	51.0	174.50
98.00	60.0	0.30	3.75	0.00	99.5	201.25
100.00	780.0	7.00	19.50	0.00	0.0	119.50
110.00	1800.0	17.30	43.50	0.00	4.0	157.50

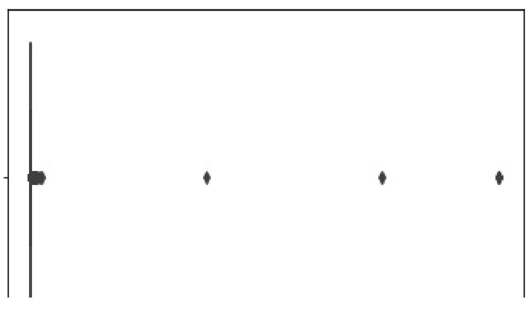
2077 rows × 6 columns

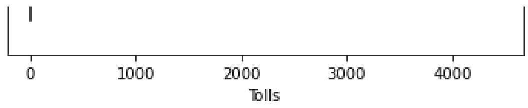
Tolls

The Tolls column had outliers that seemed like deliberate data entry errors such as “3333.3” and “4444.4” Other abnormally high toll amounts did not make sense in the context of the other data contained in the row. For example, a taxi trip with $93 in tolls lasted 1,423 seconds or 23 minutes with a distance of 15 miles. The boxplots indicate most of the normally distributed datapoints contain values ranging from 0 to 65. I decided to drop rows with Toll values above 75 using panda’s “drop” function. This dropped a total of 7 rows from the dataset, reducing the number of rows to 423,038. Calling the rows with the highest Toll values after dropping the 7 rows demonstrate the values made sense within the context of the data. All other outliers were retained.

In [75]:
```python
#Boxplot to detect outliers in Tolls column
sns.boxplot(x='Tolls', data=taxi)
```

Out[75]: <AxesSubplot:xlabel='Tolls'>





In [76]:

```
# Calling rows with highest Tolls to view data
taxi.groupby('Tolls').max(10)
```

Out[76]:

Tolls	Trip Seconds	Trip Miles	Fare	Tips	Extras	Trip Total
0.00	5200.0	67.80	102.00	110.00	4890.00	4900.00
0.01	3974.0	42.42	100.25	35.00	65.00	176.26
0.02	2654.0	36.24	85.25	5.00	47.50	132.77
0.03	2003.0	26.58	63.50	15.55	32.97	93.30
0.05	0.0	0.00	3.25	0.00	42.75	46.05
...	...	...	...	...	...	...
93.00	1423.0	15.05	37.75	0.00	0.00	130.75
99.00	1713.0	10.21	28.75	32.06	0.00	160.31
1666.66	660.0	1.30	8.50	0.00	0.00	1675.16
3333.33	1140.0	0.20	13.75	0.00	3333.33	6680.41
4444.44	1500.0	0.40	23.25	0.00	4444.44	8912.13

134 rows × 6 columns

In [77]:

```
# Max values in tolls do not appear to be legitimate
```

In [78]:

```
# Data frame shape
print(taxi.shape)
```

```
(423045, 13)
```

In [79]:

```
# dropping rows with extreme outliers in Tolls
taxi = taxi.drop(taxi[(taxi['Tolls'] < 0) | (taxi['Tolls'] > 90)].index)
```

In [80]:

```
#Data frame shape after dropping outliers in Tolls
print(taxi.shape)
```

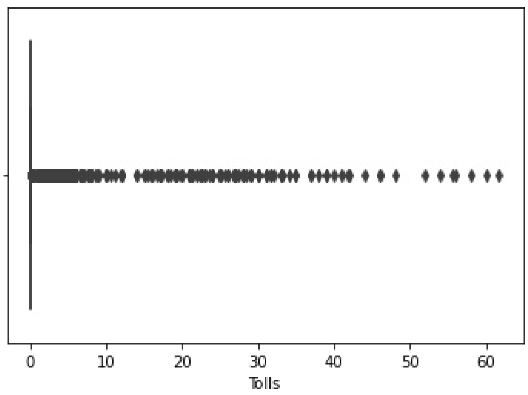
```
(423038, 13)
```

In [81]:

```
#Boxplot to detect outliers in Tolls column
sns.boxplot(x='Tolls', data=taxi)
```

Out[81]:

```
<AxesSubplot:xlabel='Tolls'>
```



In [82]:

```python
# Calling rows with highest Tolls to view data
taxi.groupby('Tolls').max(10)
```

Out[82]:

Tolls	Trip Seconds	Trip Miles	Fare	Tips	Extras	Trip Total
0.00	5200.0	67.80	102.00	110.00	4890.00	4900.00
0.01	3974.0	42.42	100.25	35.00	65.00	176.26
0.02	2654.0	36.24	85.25	5.00	47.50	132.77
0.03	2003.0	26.58	63.50	15.55	32.97	93.30
0.05	0.0	0.00	3.25	0.00	42.75	46.05
...	...	...	...	...	...	...
55.55	3360.0	39.60	94.50	15.00	7.55	172.60
56.00	116.0	0.55	4.50	9.15	0.00	70.15
58.00	4214.0	39.10	99.75	0.00	15.00	173.25
60.00	3300.0	0.00	61.50	0.00	48.00	169.50
61.75	18.0	0.00	3.25	0.00	0.00	65.50

129 rows × 6 columns

In [83]:

```python
# Remaining highest Tolls rolls appear legitimate and will be retained
```

Extras

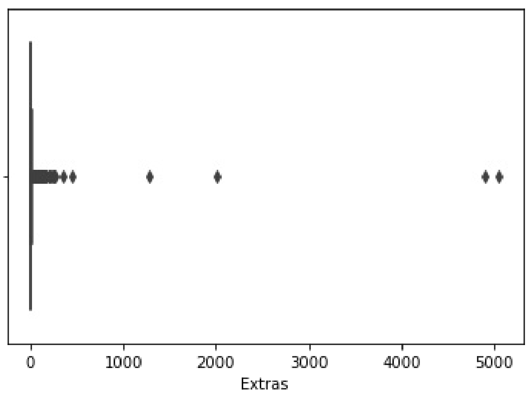
It was difficult to discern how to address in the Extras column because the data dictionary did not define what extras were. They could have been anything major such as parking tickets, damage to the vehicle, to something as minor as purchasing a bottle of water the taxi driver kept in a cooler for customers. Viewing the maximum and minimum values revealed most extras were in a range of $0 - $100. As such, I viewed Extras with values $1,000+ as outliers caused by data entry errors or equipment malfunction. I decided to drop rows with Extras values outside the range of 0 – 300 using pandas “drop” function and setting my range parameters. A total of 6 rows were dropped from the dataset. The newly reduced dataset contained 423,032 rows.

Creating a second boxplot and calling the rows with the highest values after dropping rows with extreme outliers revealed outliers were still present, but they appeared legitimate and made sense in context of the other data within the row. As such, all remaining outliers in Extras were retained.

In [84]:

```python
#Boxplot to detect outliers in Extras column
sns.boxplot(x='Extras', data=taxi)
```

Out[84]: <AxesSubplot:xlabel='Extras'>



In [85]:

```python
# Calling rows with highest Extras to view data
taxi.groupby('Extras').max(10)
```

Out[85]:

Extras	Trip Seconds	Trip Miles	Fare	Tips	Tolls	Trip Total
0.00	5200.0	67.80	102.00	100.00	61.75	160.50
0.01	1680.0	11.50	31.00	5.10	0.00	31.01
0.02	1780.0	16.83	42.00	5.00	0.00	47.52

0.03	1440.0	17.30	43.00	0.00	0.00	43.03
0.04	3331.0	16.26	41.25	8.36	0.00	50.15
...	...	...	...	...	...	...
448.40	1200.0	0.60	25.75	0.00	0.00	474.15
1278.98	2280.0	17.20	44.25	0.00	0.49	1323.72
2006.55	2160.0	1.00	45.25	0.00	0.00	2051.80
4890.00	180.0	0.10	10.00	0.00	0.00	4900.00
5051.10	60.0	0.00	3.50	0.00	0.50	5055.10

826 rows × 6 columns

In [86]:

```
# There is no context for what "extras" are. They could be gas, damage to vehicle, parking tickets, etc
# Since there is no context to know if the outliers are legitimate, I will drop extreme outliers outside 0-300
```

In [87]:

```
# Data frame shape
print(taxi.shape)
```

```
(423038, 13)
```

In [88]:

```
# dropping rows with extreme outliers in Extras
taxi = taxi.drop(taxi[(taxi['Extras'] < 0) | (taxi['Extras'] > 300)].index)
```

In [89]:

```
# Data frame shape after removing outliers in Extras
print(taxi.shape)
```

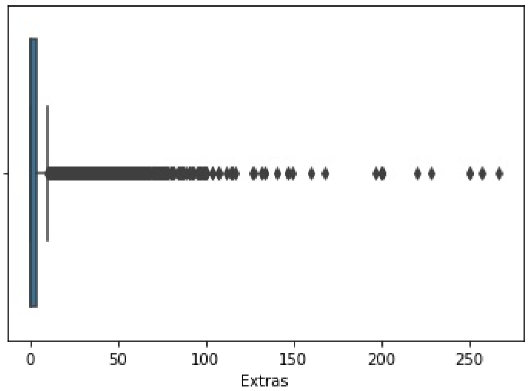
```
(423032, 13)
```

In [90]:

```
# Rechecking outliers in Extras column with boxplot
sns.boxplot(x='Extras', data=taxi)
```

Out[90]:

```
<AxesSubplot:xlabel='Extras'>
```



In [91]:

```
# Calling rows with highest Extras to view data
taxi.groupby('Extras').max(10)
```

Out[91]:

	Trip Seconds	Trip Miles	Fare	Tips	Tolls	Trip Total
Extras						
0.00	5200.0	67.80	102.00	100.00	61.75	160.50
0.01	1680.0	11.50	31.00	5.10	0.00	31.01
0.02	1780.0	16.83	42.00	5.00	0.00	47.52
0.03	1440.0	17.30	43.00	0.00	0.00	43.03
0.04	3331.0	16.26	41.25	8.36	0.00	50.15
...	...	...	...	...	...	...
220.00	50.0	0.00	3.50	30.00	0.00	254.00

228.00	1505.0	16.68	42.00	0.00	0.00	270.00
250.00	480.0	0.10	3.25	0.00	0.00	253.25
257.00	37.0	0.00	3.25	10.00	0.00	270.75
267.00	0.0	0.00	3.25	0.00	0.00	270.25

820 rows × 6 columns

In [92]:

```
# The Extras data appears skewed to the left
# While there is no context, the extras totals make sense in context with the other data in each row.
# All other outliers will be retained
```

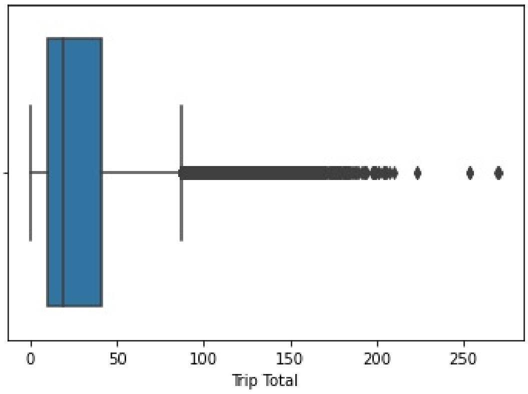
Trip Total

All outliers in Trip Total appeared legimate and were retained.

In [93]:

```
#Boxplot to detect outliers in Trip Total column
sns.boxplot(x='Trip Total', data=taxi)
```

Out[93]: <AxesSubplot:xlabel='Trip Total'>



In [94]:

```
# Calling rows with highest Trip Total values to view data
taxi.groupby('Trip Total').max(10)
```

Out[94]:

	Trip Seconds	Trip Miles	Fare	Tips	Tolls	Extras
Trip Total						
0.00	5160.0	34.37	0.00	0.0	0.0	0.0
0.01	4196.0	42.84	0.01	0.0	0.0	0.0
0.02	4612.0	11.52	0.02	0.0	0.0	0.0
0.05	5190.0	45.78	0.05	0.0	0.0	0.0
0.09	60.0	0.00	0.09	0.0	0.0	0.0
...	...	...	...	...	...	...
253.25	480.0	0.10	3.25	0.0	0.0	250.0
254.00	50.0	0.00	3.50	30.0	0.0	220.0
270.00	1505.0	16.68	42.00	0.0	0.0	228.0
270.25	0.0	0.00	3.25	0.0	0.0	267.0
270.75	37.0	0.00	3.25	10.0	0.0	257.0

7409 rows × 6 columns

In [95]:

```
# While there are outliers present, they make sense in context with the other data and will be retained
```

Renaming Columns to Remove Spaces

In order to keep my code concise and readable for future dataframe manipulations, I decided to remove spaces from

In order to keep my code concise and readable for future dataframe manipulations, I decided to remove spaces from the remaining column names. I did this by using pandas "rename(column_name)" function.

In [96]:
```
taxi.rename(columns = {'Trip ID':'TripID',
'Taxi ID':'TaxiID',
'Trip Start Timestamp':'TripStartTimestamp',
'Trip End Timestamp':'TripEndTimestamp',
'Trip Seconds':'TripSeconds',
'Trip Miles':'TripMiles',
'Trip Total':'TripTotal',
'Payment Type':'PaymentType'},
inplace=True)
```

C. Summary Statistics And Visualizations

It was time to explore the newly cleaned data by creating histograms of the distributions of a single column using matplotlib's "plot" function. Bivariate visualizations containing information from multiple columns were created using Seaborn The summary statistics of the data was returned using pandas "describe()" function and value counts for categorical variables were called using pandas "value_counts()" function. The ability to understand and see each column of applicable data provides a huge andvantage in understanding the results of the analysis. The primary disadvantages of this step are it can be time consuming and it increases the memory usage of Jupyter Notebook within the browser.

Trip Seconds

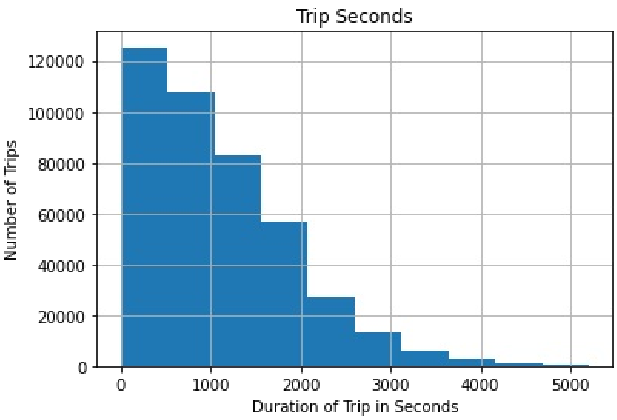
Trip Seconds has a mean ride duration of 1096 seconds or 18 minutes. The minimum ride was 0 seconds, probably for customers who entered the taxi, asked about the rates, and decided to use other forms of transportation. The longest taxi trip lasted 5205 seconds or one hour and 26 minutes. The data for Trip Seconds skews right in the histogram.

In [97]:
```
# Summary Statistics for Trip Seconds
taxi.TripSeconds.describe()
```

Out[97]:
```
count    423032.000000
mean       1096.408411
std         829.380795
min           0.000000
25%         453.000000
50%         900.000000
75%        1574.000000
max        5205.000000
Name: TripSeconds, dtype: float64
```

In [98]:
```
# histogram of Trip Seconds
taxi.hist('TripSeconds')
plt.xlabel('Duration of Trip in Seconds')
plt.ylabel('Number of Trips')
plt.title('Trip Seconds')
```

Out[98]:
```
Text(0.5, 1.0, 'Trip Seconds')
```



Trip Miles

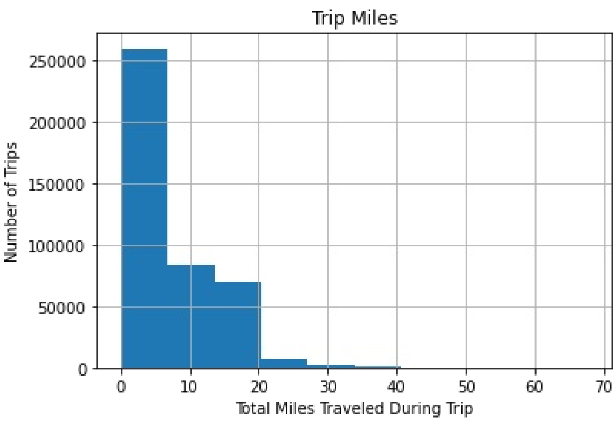
Trip Seconds has a mean travel distance of 6.49 miles. The minimum mileage was 0 miles, probably for customers who entered the taxi, asked about the rates, and decided to use other forms of transportation, or for customers who sat in the taxi with the meter running while waiting for someone. The longest distance travelled on a taxi trip was 67 miles. The data for Trip Miles skews right in the histogram.

In [99]:
```
# Summary Statistics for Trip Miles
taxi.TripMiles.describe()
```

Out[99]:
```
count    423032.000000
mean          6.486420
std           6.878504
min           0.000000
25%           0.900000
50%           3.000000
75%          11.700000
max          67.800000
Name: TripMiles, dtype: float64
```

In [100…
```
# histogram of Trip Miles
taxi.hist('TripMiles')
plt.xlabel('Total Miles Traveled During Trip')
plt.ylabel('Number of Trips')
plt.title('Trip Miles')
```

Out[100…
```
Text(0.5, 1.0, 'Trip Miles')
```



Fare

The mean fare for a taxi trip was 21.74. The minimum fare was 0, probably for customers who entered the taxi, asked about the rates, and decided to use other forms of transportation. The highest fare recorded for a taxi trip was 102. The data for Fare skews right in the histogram.

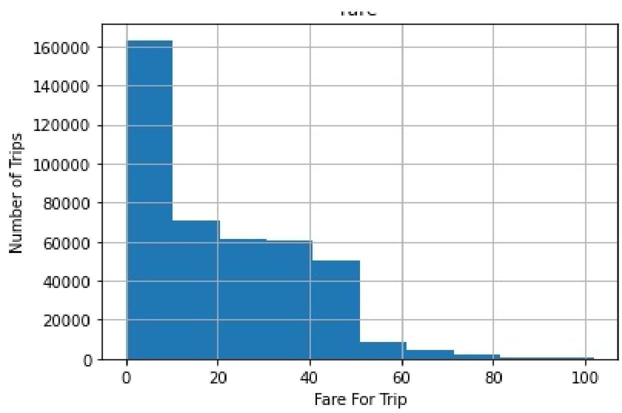
In [101…
```
# Summary Statistics for Fare
taxi.Fare.describe()
```

Out[101…
```
count    423032.000000
mean         21.744080
std          16.304927
min           0.000000
25%           8.000000
50%          15.980000
75%          33.680000
max         102.000000
Name: Fare, dtype: float64
```

In [102…
```
# histogram of Fare
taxi.hist('Fare')
plt.xlabel('Fare For Trip')
plt.ylabel('Number of Trips')
plt.title('Fare')
```

Out[102…
```
Text(0.5, 1.0, 'Fare')
```

Fare



Tips

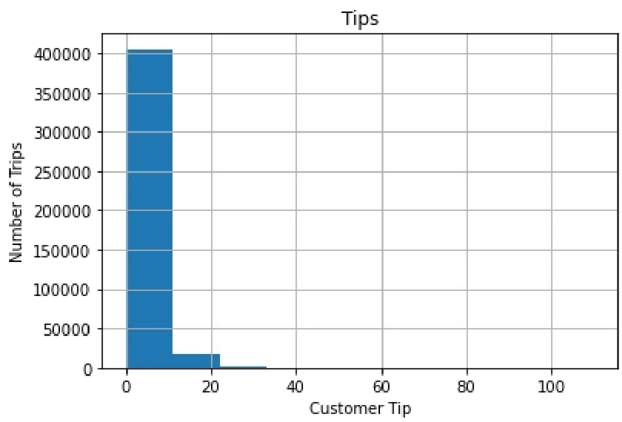
The mean tip amount for a taxi trip was 2.76. The minimum amount tipped was 0. The highest customer tip recorded for a taxi trip was 110. The data for Tips skews right in the histogram.

In [103…
```
# Summary Statistics for Tips
taxi.Tips.describe()
```

Out[103…
```
count    423032.000000
mean          2.762978
std           4.067669
min           0.000000
25%           0.000000
50%           0.000000
75%           4.000000
max         110.000000
Name: Tips, dtype: float64
```

In [104…
```
# histogram of Tips
taxi.hist('Tips')
plt.xlabel('Customer Tip')
plt.ylabel('Number of Trips')
plt.title('Tips')
```

Out[104…
```
Text(0.5, 1.0, 'Tips')
```



Tolls

The mean toll amount for a taxi trip was two cents. The minimum toll amount was 0. The highest toll amount recorded for a taxi trip was 61.75. The data for Tolls skews right in the histogram.

In [105…
```
# Summary Statistics for Tolls
taxi.Tolls.describe()
```

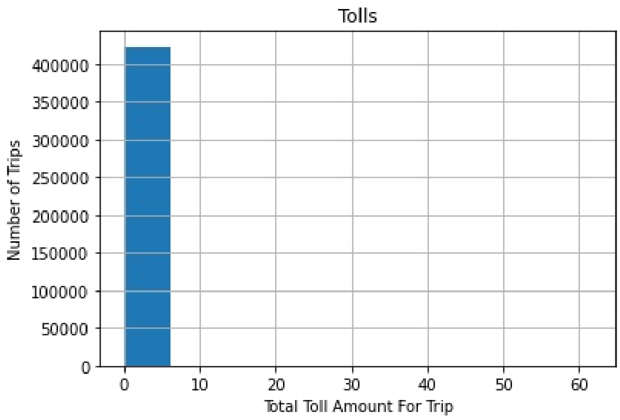
Out[105…
```
count    423032.000000
mean          0.022473
std           0.553061
```

```
min              0.000000
25%              0.000000
50%              0.000000
75%              0.000000
max             61.750000
Name: Tolls, dtype: float64
```

In [106…

```
# histogram of Tolls
taxi.hist('Tolls')
plt.xlabel('Total Toll Amount For Trip')
plt.ylabel('Number of Trips')
plt.title('Tolls')
```

Out[106… Text(0.5, 1.0, 'Tolls')



Extras

The mean amount charged for Extras was 2.09. The minimum amount charged for extras was 0. The highest amount charged for extras recorded for a taxi trip was 267. The data for Extras skews right in the histogram.

In [107…

```
# Summary Statistics for Extras
taxi.Extras.describe()
```

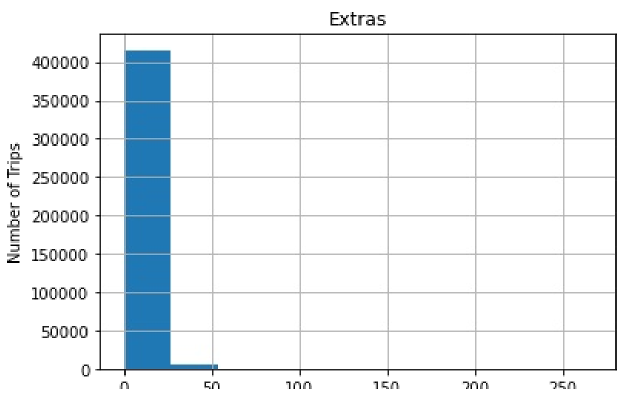
Out[107…

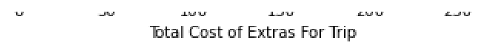
```
count    423032.000000
mean          2.099612
std           5.931692
min           0.000000
25%           0.000000
50%           0.000000
75%           4.000000
max         267.000000
Name: Extras, dtype: float64
```

In [108…

```
# histogram of Extras
taxi.hist('Extras')
plt.xlabel('Total Cost of Extras For Trip')
plt.ylabel('Number of Trips')
plt.title('Extras')
```

Out[108… Text(0.5, 1.0, 'Extras')





Trip Total

The mean total amount charged for a taxi ride was 26.81. The minimum total amount charged for a taxi ride was 0, probably for customers who entered the taxi, asked about the rates, and decided to use other forms of transportation. The highest total amount charged for a taxi trip was 270.75. The data for Trip Total skews right in the histogram.

In [109…

```
# Summary Statistics for Trip Total
taxi.TripTotal.describe()
```

Out[109…

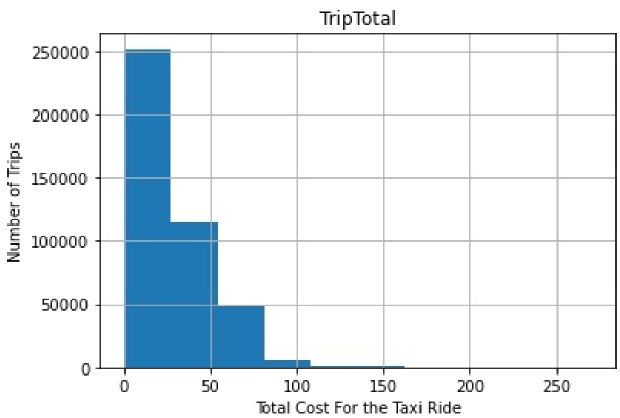
```
count    423032.000000
mean         26.809571
std          21.724755
min           0.000000
25%          10.000000
50%          19.000000
75%          41.000000
max         270.750000
Name: TripTotal, dtype: float64
```

In [110…

```
# histogram of Trip Total
taxi.hist('TripTotal')
plt.xlabel('Total Cost For the Taxi Ride')
plt.ylabel('Number of Trips')
plt.title('TripTotal')
```

Out[110…

```
Text(0.5, 1.0, 'TripTotal')
```



Payment Type

The most common form of payment type used by customers was a credit card. The second most common form of payment was cash. The least common form of payment was no payment due to disputed charges and the second least form of payment was no charge, likely due to customers entering the taxi, asking about the rates, and deciding to use other forms of transportation.

In [111…

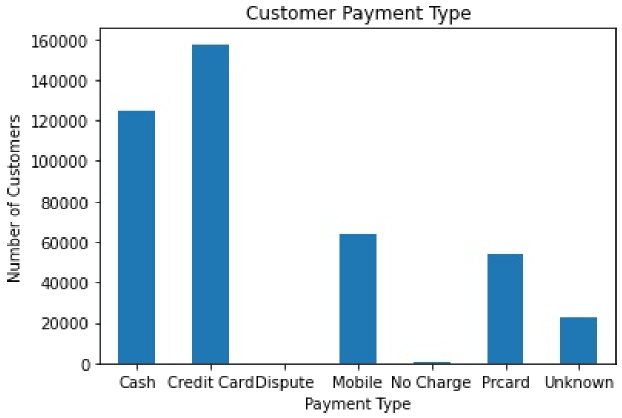
```
# Value counts for Payment Type
taxi.value_counts('PaymentType')
```

Out[111…

```
PaymentType
Credit Card    157700
Cash           124568
Mobile          64145
Prcard          53847
Unknown         22375
No Charge         267
Dispute           130
dtype: int64
```

In [112…

```
# Visualization of Payment Type
taxi.groupby('PaymentType').size().plot.bar(rot=0)
plt.xlabel('Payment Type')
plt.ylabel('Number of Customers')
plt.title('Customer Payment Type')
plt.show()
```



Company

The most popular taxi cab companies among customers are Flash Cab and Taxi Affiliation Services. The least popular cab companies are Reny Cab Co. and Checker Cab Dispatch.

In [113…

```
# Value counts for Comapny
taxi.value_counts('Company')
```

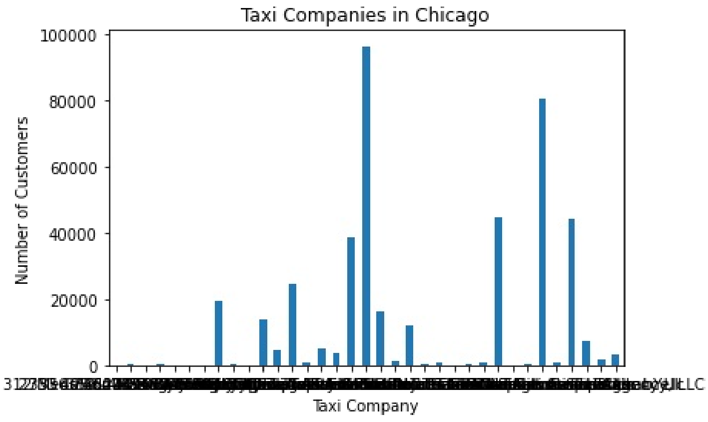
Out[113…

```
Company
Flash Cab                                  96514
Taxi Affiliation Services                  80435
Sun Taxi                                   44702
Taxicab Insurance Agency Llc               44122
City Service                               38411
Chicago Independents                       24789
5 Star Taxi                                19704
Globe Taxi                                 16047
Blue Ribbon Taxi Association               14036
Medallion Leasin                           12097
Taxicab Insurance Agency, LLC               7587
Choice Taxi Association                     4812
Chicago City Taxi Association               4429
Choice Taxi Association Inc                 3596
U Taxicab                                   3098
Top Cab                                     1742
Koam Taxi Association                       1174
Taxi Affiliation Services Llc - Yell         886
Patriot Taxi Dba Peace Taxi Associat         782
Chicago Taxicab                              736
Star North Taxi Management Llc               697
312 Medallion Management Corp                509
Setare Inc                                   466
3591 - 63480 Chuks Cab                       333
Metro Jet Taxi A.                            312
5167 - 71969 5167 Taxi Inc                   241
Tac - Yellow Cab Association                 183
Petani Cab Corp                               96
6574 - Babylon Express Inc.                   95
3556 - 36214 RC Andrews Cab                   95
2733 - 74600 Benny Jona                       78
4623 - 27290 Jay Kim                          75
4053 - 40193 Adwar H. Nikola                  68
Tac - Checker Cab Dispatch                    59
4787 - 56058 Reny Cab Co                      26
dtype: int64
```

In [114…

```
# Visualization of Company
taxi.groupby('Company').size().plot.bar(rot=0)
plt.xlabel('Taxi Company')
plt.ylabel('Number of Customers')
plt.title('Taxi Companies in Chicago')
```

```
plt.show()
```



Bivariate Visualizations with Tips, the independent variable

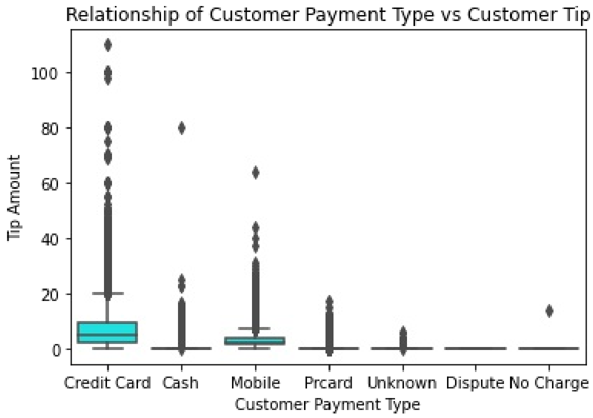
Payment Type VS Tips

In [115…

```
# Visualization with Tips and Payment Type

sns.boxplot(data=taxi, x="PaymentType", y='Tips', color="cyan")
plt.xlabel("Customer Payment Type")
plt.ylabel("Tip Amount")
plt.title("Relationship of Customer Payment Type vs Customer Tip")
```

Out[115… Text(0.5, 1.0, 'Relationship of Customer Payment Type vs Customer Tip')

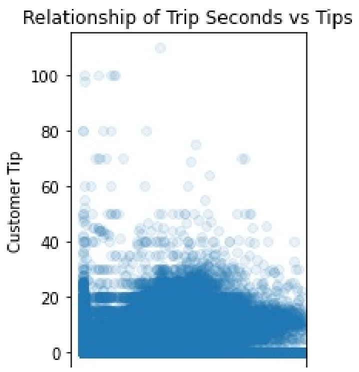


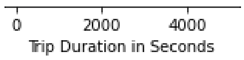
Customers who tip with credit cards tip the highest amount compared to customers who use other payment types

Trip Seconds VS Tips

In [116…

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Trip Seconds vs Tips")
sns.regplot(data=taxi, x="TripSeconds", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Trip Duration in Seconds")
plt.ylabel("Customer Tip");
```



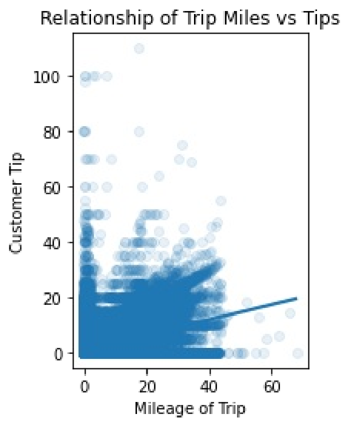


This visualization demonstrates customers who tip tend to tip under 30 regardless of trip duration. Customer who tip more than 30 are more likely to leave a larger tip on a shorter duration trip of 2,000 seconds or less.

Trip Miles vs Tips

In [117...

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Trip Miles vs Tips")
sns.regplot(data=taxi, x="TripMiles", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Mileage of Trip")
plt.ylabel("Customer Tip");
```

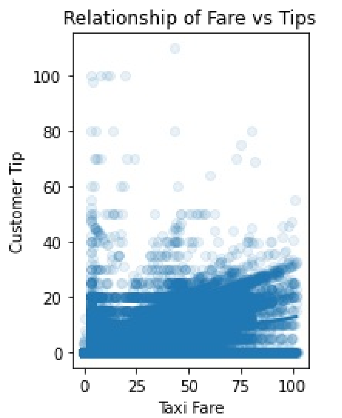


Customers tend to tip a range of $ 1 - $ 30 on trips under 45 miles, with customers tipping the highest amounts on trips under 20 miles. Customers tend to tip an average of $ 10 - $ 15 on trips longer than 40 miles.

Trip Fare Vs Tips

In [118...

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Fare vs Tips")
sns.regplot(data=taxi, x="Fare", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Taxi Fare")
plt.ylabel("Customer Tip");
```

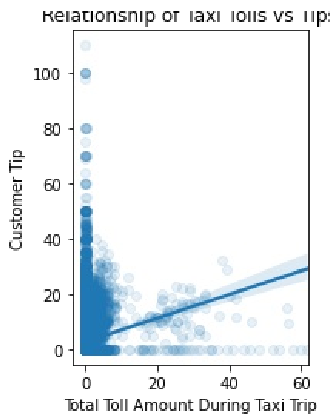


most customers who leave a tip tip between $ 1-$ 25 regardless of fare. Customers tend to tip a higher monetary amount on trips with fares $ 50 or less.

Tolls vs Tips

In [119...

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Taxi Tolls vs Tips")
sns.regplot(data=taxi, x="Tolls", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Total Toll Amount During Taxi Trip")
plt.ylabel("Customer Tip");
```

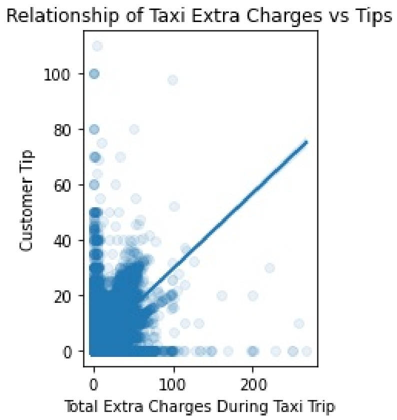


Customers are more likely to leave a tip if their taxi trip does not include tolls. As the toll amount increases, the total tip the customer leaves the driver also increases.

Extras vs Tips

In [120…

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Taxi Extra Charges vs Tips")
sns.regplot(data=taxi, x="Extras", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Total Extra Charges During Taxi Trip")
plt.ylabel("Customer Tip");
```

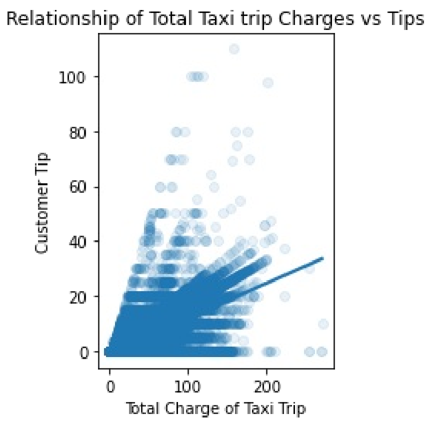


The majority of customers who leave a tip for their driver are likely to incur extra charges of $0-$100. The total tip a customer leaves increases linearly with the increase of extra charges.

Trip Total vs Tips

In [121…

```
plt.subplot(1, 2, 2)
plt.title("Relationship of Total Taxi trip Charges vs Tips")
sns.regplot(data=taxi, x="TripTotal", y="Tips", x_jitter=0.3, scatter_kws={'alpha' : 1/10})
plt.xlabel("Total Charge of Taxi Trip")
plt.ylabel("Customer Tip");
```



Most customers who tip their taxi driver tip between 1 -30. After a trip total reaches 150, customers who tip tend to

Most customers who tip their taxi driver tip between 1- 30. After a trip total reaches 150, customers who tip tend to tip 25 or more, with the tip amount typically increasing as the trip total increases.

Data Wrangling

Now that the data had been cleaned and visualized, it was time to wrangle the data to make it suitable for multiple linear regression. First, I dropped columns that were unnecessary. TripID, TaxiID, TripStartTimeStamp, TripEndTimeStamp, and Company were dropped from the taxi dataset. One advantage of dropping the unnecessary columns is the data contained within the columns will not skew the results of the multiple linear regression model. One disadvantage of dropping the columns is it reduces the dimensionality of the dataset and could cause multicollinearity issues in the remaining variables.

Multiple Linear regression requires all variables to be expressed as numerical values. The Payment Type Column was categorical. I used One Hot Encoding to convert the Payment Type column to numerical values. One-hot encoding creates a new column for each feature within the original categorical column and assigns either a one or a zero depending on if that feature was present in a specific row of data. One of the feature columns is dropped before running the multiple linear regression model to reduce multicollinearity. For Payment Type, the Cash feature column was dropped. The user is still able to determine if the dropped feature applies to the row if all other feature columns contain zeros. For example, if a customer paid for their taxi with a credit card, the credit card column would have a “1” while the remaining Payment Type columns would contain zeros for that row. After the columns had been created, they were joined to the dataset using pandas “pd.concat([df, ColumnName]) function. One advantage to one hot encoding is it automatically drops a column to avoid multicollinearity. One disadvantage is it increases the dimensionality of the data with sparse data since most of the values generated in the new columns will be zero.

Finally, a separate dataframe was created that included the individual Payment Type columns for the multiple linear regression model. The multiple linear regression model dataframe contained the following columns: Tips, TripSeconds, TripMiles, Fare, Tolls, Extras, TripTotal, PaymentCreditCard, PaymentDispute, PaymentMobile, PaymenyNoCharge, PaymentPrcard, and PaymentUnknown.

Dropping Columns

In [122…
```
# Dropping unnecessary columns that would increase dimensionality of data
taxi = taxi.drop(columns=['TripID',
                           'TaxiID',
                           'TripStartTimestamp',
                           'TripEndTimestamp',
                           'Company'])
```

In [123…
```
# Reduced dataframe
taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 423032 entries, 0 to 425218
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   TripSeconds  423032 non-null  float64
 1   TripMiles    423032 non-null  float64
 2   Fare         423032 non-null  float64
 3   Tips         423032 non-null  float64
 4   Tolls        423032 non-null  float64
 5   Extras       423032 non-null  float64
 6   TripTotal    423032 non-null  float64
 7   PaymentType  423032 non-null  object
dtypes: float64(7), object(1)
memory usage: 45.2+ MB
```

One-Hot Encoding Columns With Nominal Data and Renaming The Resulting Columns

In [124…
```
# Generating columns of dummy values for taxi's Payment Type column and renaming the new columns
payment_tempdf = pd.get_dummies(data=taxi["PaymentType"], drop_first=True)
```

In [125…
```
# Calling column names
payment_tempdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 423032 entries, 0 to 425218
Data columns (total 6 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   Credit Card  423032 non-null  uint8
```

```
 1   Dispute      423032 non-null  uint8
 2   Mobile       423032 non-null  uint8
 3   No Charge    423032 non-null  uint8
 4   Prcard       423032 non-null  uint8
 5   Unknown      423032 non-null  uint8
dtypes: uint8(6)
memory usage: 21.8 MB
```

In [126…

```
# renaming the dummy Payment columns for clarity and to remove any spaces
payment_tempdf = payment_tempdf.rename(columns={'Credit Card' : 'PaymentCreditCard',
                                                'Dispute' : 'PaymentDispute',
                                                'Mobile' : 'PaymentMobile',
                                                'No Charge' : 'PaymentNoCharge',
                                                'Prcard' : 'PaymenyPrcard',
                                                'Unknown': 'PaymentUnknown'})
payment_tempdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 423032 entries, 0 to 425218
Data columns (total 6 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   PaymentCreditCard  423032 non-null  uint8
 1   PaymentDispute     423032 non-null  uint8
 2   PaymentMobile      423032 non-null  uint8
 3   PaymentNoCharge    423032 non-null  uint8
 4   PaymenyPrcard      423032 non-null  uint8
 5   PaymentUnknown     423032 non-null  uint8
dtypes: uint8(6)
memory usage: 21.8 MB
```

Joining Dummy Dataframe Columns With taxi Dataset

In [127…

```
# Joining PaymentCreditCard
PaymentCreditCard = payment_tempdf["PaymentCreditCard"]
taxi = pd.concat([taxi,PaymentCreditCard], axis = 1)
```

In [128…

```
# Joining PaymentDispute
PaymentDispute = payment_tempdf["PaymentDispute"]
taxi = pd.concat([taxi,PaymentDispute], axis = 1)
```

In [129…

```
# Joining PaymentMobile
PaymentMobile = payment_tempdf["PaymentMobile"]
taxi = pd.concat([taxi,PaymentMobile], axis = 1)
```

In [130…

```
# Joining PaymentNoCharge
PaymentNoCharge = payment_tempdf["PaymentNoCharge"]
taxi = pd.concat([taxi,PaymentNoCharge], axis = 1)
```

In [131…

```
# Joining PaymenyPrcard
PaymenyPrcard = payment_tempdf["PaymenyPrcard"]
taxi = pd.concat([taxi,PaymenyPrcard], axis = 1)
```

In [132…

```
# Joining PaymentUnknown
PaymentUnknown = payment_tempdf["PaymentUnknown"]
taxi = pd.concat([taxi,PaymentUnknown], axis = 1)
```

In [133…

```
# Dropping Payment Type from dataframe
taxi = taxi.drop(columns=['PaymentType'])
```

In [134…

```
# Checking newly added columns
taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 423032 entries, 0 to 425218
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   TripSeconds        423032 non-null  float64
```

```
 1   TripMiles          423032 non-null  float64
 2   Fare               423032 non-null  float64
 3   Tips               423032 non-null  float64
 4   Tolls              423032 non-null  float64
 5   Extras             423032 non-null  float64
 6   TripTotal          423032 non-null  float64
 7   PaymentCreditCard  423032 non-null  uint8
 8   PaymentDispute     423032 non-null  uint8
 9   PaymentMobile      423032 non-null  uint8
 10  PaymentNoCharge    423032 non-null  uint8
 11  PaymenyPrcard      423032 non-null  uint8
 12  PaymentUnknown     423032 non-null  uint8
dtypes: float64(7), uint8(6)
memory usage: 44.4 MB
```

Creating Seperate Multiple Linear Regression Dataframe

In [135…

```
# Creating mlr_taxi dataframe with tips as independent variable
mlr_taxi = taxi[["Tips",
                  "TripSeconds",
                  "TripMiles",
                  "Fare",
                  "Tolls",
                  "Extras",
                  "TripTotal",
                  "PaymentCreditCard",
                  "PaymentDispute",
                  "PaymentMobile",
                  "PaymentNoCharge",
                  "PaymenyPrcard",
                  "PaymentUnknown"]]
```

In [136…

```
mlr_taxi.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 423032 entries, 0 to 425218
Data columns (total 13 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Tips               423032 non-null  float64
 1   TripSeconds        423032 non-null  float64
 2   TripMiles          423032 non-null  float64
 3   Fare               423032 non-null  float64
 4   Tolls              423032 non-null  float64
 5   Extras             423032 non-null  float64
 6   TripTotal          423032 non-null  float64
 7   PaymentCreditCard  423032 non-null  uint8
 8   PaymentDispute     423032 non-null  uint8
 9   PaymentMobile      423032 non-null  uint8
 10  PaymentNoCharge    423032 non-null  uint8
 11  PaymenyPrcard      423032 non-null  uint8
 12  PaymentUnknown     423032 non-null  uint8
dtypes: float64(7), uint8(6)
memory usage: 44.4 MB
```

In [137…

```
# saving mlr_taxi dataset to CSV
mlr_taxi.to_csv('D214mlr_taxi.csv', index = True)
```

D. Analysis Technique

Multiple linear regression was used to see if trip miles, trip duration, tolls, taxi fare, or type of payment was statistically correlated with customers leaving a tip for their cab driver. Multiple linear regression demonstrates if a correlation between a single target variable and multiple explanatory variables is present.

Comparing linear regression with multiple linear regression can help people understand multiple linear regression. If driving 40 miles takes you 1 hour, and driving 80 miles takes 2 hours, we can start to form a relationship that lets us predict that driving 120 miles will take us 3 hours. This is an example of linear regression. Linear regression involves a single explanatory variable (distance) and a single target variable (time taken). Unfortunately driving is rarely as simple as calculating the distance and time. Things such as construction, traffic, and weather also affect the time taken to travel a certain distance. Multiple linear regression can account for the effects of multiple explanatory variables on the target variable. Multiple linear regression demonstrates relationships by comparing the data points of a variable in a graph to the slope of a line.

I chose multiple linear regression for this analysis since there is one numerical target variable (Tips) and numerous numerical explanatory variables influencing the target variable. The explanatory variables influencing Tips are Trip Seconds, TripMiles, Fare, Tolls, Extras, TripTotal, and the Payment Type variables that were converted to numerical values.

The primary advantage of multiple linear regression is it's ability to determine if multiple explanatory variables are influencing the target variable. One disadvantage of multiple linear regression is it sensitive to data with wide ranges of values, which can cause heteroskedasticity, or unequal variance amongst the residuals of the models.

For a multiple linear regression model to be successful, these four assumptions must be followed:

• There must be a linear relationship between the target variable and the explanatory variables. If no relationship exists between x1 and y, then there's no relationship for the multiple regression to tease out for x1 and y when compared with x2, x3, and so on. If the make of a car has no relationship with the time taken to drive a particular distance, then there's no reason to include it in an analysis of factors that affect the time it takes to a drive a certain distance.

• There must be no multicollinearity (correlation) between the explanatory variables. Where x variables are closely related to each other, this confounds the multiple regression model's attempt to find a relationship between x and y, because the different x variables are essentially "talking" to each other throughout the process. If a construction zone is in a high traffic area, it would be difficult to determine if delays were caused by traffic or construction.

• All observations must be independent from each other. Each row in a data set represents an observation. If a driver takes 55 minutes to drive 80 miles, encountering a certain traffic density, at certain speeds, in specific weather, this cannot have an impact (must be independent) on the next observation of a driver going on another trip. If observations are related to eachother, it causes similar issues to multicolinearity, where different variables are "talking" to each other throughout the process of trying to find a relationship between x and y.

• The residuals have constant variance at every point in the linear model and have no discernable pattern when graphed or plotted. This is called homoscedasticity. Homoscedasticity refers to the tendency for a model's residuals to be relatively constant. If these residuals are not constant and instead vary significantly, then the residuals are actually heteroscedastic. If heteroscedasticity undermines the multiple regression model, making it unreliable (Zach, 2021).

The initial multiple linear regresion model is below. I set Tips as the Y-intercept and loading the explanatory variables listed above as the x-intercepts. I assigned a constant of 1 and ran statsmodel's OLS report. The OLS (Ordinary Least Sqares) process creates a single regression equation to predict correlation between the target variable and explanatory variable. The OLS report contains statistical results and residuals that can be used to evaluate the performance of the model and the explanatory variables it contains.

For the sake of this analysis, I focused on the adjusted R-squared value and P-values. Adjusted R Squared calculates the variance for Tips that is explained by the explanatory variables in the regression model. For example, if the model had an adjusted R-Squared value of 0.400, that would indicate that only 40% of the variance found in Tips was explained by the explanatory variables. The closer to 1.000 the score is, the better the score. Unlike r-squared, adjusted r-squared takes the total number of variables into consideration in the equation (Potters, 2023). P-values measure the statistical significance a single explanatory variable has on the target variable, Tips. P-value scores range from 0.00 to 1.0. A lower P-value indicates stronger statistical significance while a higher P-Value rating indicates little or no statistical significance. Generally, P-value scores of 0.05 or under are considered statistically significant.

Residual standard error will also be used to evaluate the initial model and the final model. It is not included in the OLS report, but it can be calculated using Statsmodels “results.resid.std(dataframe=dfshape)” function. The residual standard error measures the standard error of the resulting residuals under that model. The lower the score the better the model.

Initial Model

In [138…

```
# Set dependent variable
y = mlr_taxi.Tips
# Set independent variables
X = mlr_taxi = taxi[["TripSeconds",
                   "TripMiles",
                   "Fare",
                   "Tolls",
                   "Extras",
                   "TripTotal",
                   "PaymentCreditCard",
                   "PaymentDispute",
                   "PaymentMobile",
                   "PaymentNoCharge",
                   "PaymenyPrcard",
                   "PaymentUnknown"]].assign(const=1)
model = model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Tips   R-squared:                       0.994
Model:                            OLS   Adj. R-squared:                  0.994
```

```
Method:                 Least Squares   F-statistic:                 5.754e+06
Date:                Wed, 27 Mar 2024   Prob (F-statistic):               0.00
Time:                        14:59:20   Log-Likelihood:            -1.1482e+05
No. Observations:              423032   AIC:                         2.297e+05
Df Residuals:                  423019   BIC:                         2.298e+05
Df Model:                          12
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
TripSeconds        1.392e-05   9.38e-07     14.845      0.000    1.21e-05    1.58e-05
TripMiles            -0.0109      0.000    -78.601      0.000      -0.011      -0.011
Fare                 -0.9791      0.000  -4543.219      0.000      -0.980      -0.979
Tolls                -0.9837      0.001  -1080.732      0.000      -0.986      -0.982
Extras               -0.9795      0.000  -4248.783      0.000      -0.980      -0.979
TripTotal             0.9835      0.000   5078.917      0.000       0.983       0.984
PaymentCreditCard    -0.2419      0.002   -151.577      0.000      -0.245      -0.239
PaymentDispute        0.0337      0.028      1.208      0.227      -0.021       0.088
PaymentMobile        -0.0027      0.002     -1.590      0.112      -0.006       0.001
PaymentNoCharge       0.0639      0.019      3.284      0.001       0.026       0.102
PaymenyPrcard         0.0662      0.002     38.921      0.000       0.063       0.070
PaymentUnknown        0.0282      0.002     11.935      0.000       0.024       0.033
const                -0.0998      0.001    -88.898      0.000      -0.102      -0.098
==============================================================================
Omnibus:                   474342.605   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         41623165.824
Skew:                          -5.908   Prob(JB):                         0.00
Kurtosis:                      50.136   Cond. No.                     7.85e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 7.85e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [139…

```
# Calculating Residual Standard Error of initial model
results.resid.std(ddof=X.shape[1])
```

Out[139…

```
0.31742854221311806
```

The initial model has an adjusted R-squared score 0.994, meaning 99.4% of the varaince found in Tip was explained by the explanatory variables. WHile this is a very good score, some of the explanatory variables contain P-values above 0.05, indicating they are not statisitically significant. This could be artificially inflating the adjusted R-squared score. Additonally, the notes of initial OLS model indicate there are multicolinearity issues. The presence of multicolinear variables could also cause the model to artificially inflate the adjusted R-square score. The multicolinearity issues and non stastically significant explanatory variables needed to be addressed.

Variance Inflation Factor to Check for Multicolinearity

To address the multicollinearity warning in the OLS report notes, I ran a variance inflation factor test using Statsmodels variance inflation factor function. Variance inflation factor measures how correlated the explanatory variables are to each other. The variance inflation factor has a scale from one to ten, with one indicating little to no correlation and ten indicating that the variables are identical. All variables in a regression model should have a variance inflation factor of under 5.0.

Variation inflation factor is a wrapper method, meaning an analyst removes one variable at a time before rechecking the VIF scores of the remaining variables (Katari, 2021). One advantage to using varaince inflation factor is it can solve multicolinearity issues without creating a correlation matrix or a heatmap which would require me to create two more separate dataframes. One disadvantage to using variance inflation factor is it is time consuming since each correlated variable needs to be deleted one at a time.

I started by removing the variable with the highest variable inflation factor, which was TripTotal with a VIF of 172. I then reran Variance inflation factor and removed the variable with the highest VIF score, which was Fare with a VIF score of 10. Varaince inflation factor was run a third time and Trip Seconds had the highest VIF score at 5.02 and was deleted. Running the variance inflation factor for a final time revealed all explanatory variables had a VIF score of under 5.0. This solved the multicollinearity issues.

In [140…

```
# Multicolinearity warning. Runnung VIF to check for multicolinearity
# All columns with a VIF of over 5.0 will be deleted, one at a time, starting with the column with the highest VI

X = mlr_taxi = taxi[["TripSeconds",
                    "TripMiles",
                    "Fare",
                    "Tolls",
                    "Extras",
```

```
                    "TripTotal",
                    "PaymentCreditCard",
                    "PaymentDispute",
                    "PaymentMobile",
                    "PaymentNoCharge",
                    "PaymenyPrcard",
                    "PaymentUnknown"]]

vif_df = pd.DataFrame()
vif_df['variable'] = X.columns
vif_df['VIF'] = [variance_inflation_factor(X.values, i)
for i in range(X.shape[1])]
print(vif_df)
```

```
             variable         VIF
0         TripSeconds    6.639831
1           TripMiles    6.945663
2                Fare  128.192249
3               Tolls    1.062175
4              Extras    8.310520
5           TripTotal  174.380364
6   PaymentCreditCard    2.856870
7      PaymentDispute    1.000582
8       PaymentMobile    1.300265
9     PaymentNoCharge    1.000553
10      PaymenyPrcard    1.386322
11     PaymentUnknown    1.189475
```

In [141…

```
# Deleting Trip Total (VIF 174) and redoing VIF

X = mlr_taxi = taxi[["TripSeconds",
                    "TripMiles",
                    "Fare",
                    "Tolls",
                    "Extras",
                    "PaymentCreditCard",
                    "PaymentDispute",
                    "PaymentMobile",
                    "PaymentNoCharge",
                    "PaymenyPrcard",
                    "PaymentUnknown"]]

vif_df = pd.DataFrame()
vif_df['variable'] = X.columns
vif_df['VIF'] = [variance_inflation_factor(X.values, i)
for i in range(X.shape[1])]
print(vif_df)
```

```
             variable        VIF
0         TripSeconds   6.628825
1           TripMiles   6.811608
2                Fare  10.573917
3               Tolls   1.008877
4              Extras   1.391640
5   PaymentCreditCard   1.869212
6      PaymentDispute   1.000511
7       PaymentMobile   1.168959
8     PaymentNoCharge   1.000465
9       PaymenyPrcard   1.343428
10     PaymentUnknown   1.176540
```

In [142…

```
# Deleting Fare (VIF 10.57) and redoing VIF

X = mlr_taxi = taxi[["TripSeconds",
                    "TripMiles",
                    "Tolls",
                    "Extras",
                    "PaymentCreditCard",
                    "PaymentDispute",
                    "PaymentMobile",
                    "PaymentNoCharge",
                    "PaymenyPrcard",
                    "PaymentUnknown"]]

vif_df = pd.DataFrame()
vif_df['variable'] = X.columns
vif_df['VIF'] = [variance_inflation_factor(X.values, i)
for i in range(X.shape[1])]
print(vif_df)
```

```
            variable       VIF
```

```
0        TripSeconds  5.026288
1          TripMiles  4.393030
2              Tolls  1.008815
3             Extras  1.376443
4  PaymentCreditCard  1.746314
5     PaymentDispute  1.000375
6      PaymentMobile  1.146348
7    PaymentNoCharge  1.000337
8      PaymenyPrcard  1.335160
9     PaymentUnknown  1.141724
```

In [143…

```python
# Deleting TripSeconds (VIF 5.02) and redoing VIF

X = mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentDispute",
                   "PaymentMobile",
                   "PaymentNoCharge",
                   "PaymenyPrcard",
                   "PaymentUnknown"]]

vif_df = pd.DataFrame()
vif_df['variable'] = X.columns
vif_df['VIF'] = [variance_inflation_factor(X.values, i)
for i in range(X.shape[1])]
print(vif_df)
```

```
            variable       VIF
0          TripMiles  2.086551
1              Tolls  1.008587
2             Extras  1.375264
3  PaymentCreditCard  1.568882
4     PaymentDispute  1.000159
5      PaymentMobile  1.062972
6    PaymentNoCharge  1.000200
7      PaymenyPrcard  1.228375
8     PaymentUnknown  1.031462
```

In [144…

```python
# All columns have a VIF score under 5.0. Multicolinearity issues should be solved.
```

Backwards Stepwise Elimination

Backwards stepwise elimination was my feature selection method. It utilized P-values to determine if a variable has statistical significance in the model. P-value scores range from 0.00 to 1.0. A lower P-value indicates stronger statistical significance while a higher P-Value rating indicates little or no statistical significance. I chose to retain variables with P- Values of 0.05 or less.

Like Variance Inflation Factor, Backwards Stepwise elimination is a wrapper method, meaning each grouping of variables is evaluated as one set. It is crucial to eliminate only one variable at a time and rerun the model to perform backwards stepwise elimination a second time. Dropping a variable will turn the remaining variables into a new set (Katari, 2021). I eliminated the variable with the highest P-value and lowest statistical significance each round and continued until all my variables were statistically significant with P-values of 0.05 or less. The explanatory variables eliminated included PaymentDispute, PaymentUnknown, and PaymentNoCharge.

One advantage to using backwards stepwise elimination is I would not delete a variable from the model that was statistically significant. Backwards stepwise elimination also allowed me to witness the drastic impact removing one variable could have on the model. One disadvantage to using backwards stepwise elimination is it is time consuming since only one variable can be eliminated at a time.

In [145…

```python
# Creating mlr_taxi dataframe with tips as independent variable
mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentDispute",
                   "PaymentMobile",
                   "PaymentNoCharge",
                   "PaymenyPrcard",
                   "PaymentUnknown",
                   "Tips"]]
```

In [146…

```python
# Running the model again with all high VIF variables removed to observe the P-values. Aiming for P-values 0.05 o
```

```
# Set dependent variable
y = mlr_taxi.Tips
# Set independent variables
X = mlr_taxi = taxi[["TripMiles",
                    "Tolls",
                    "Extras",
                    "PaymentCreditCard",
                    "PaymentDispute",
                    "PaymentMobile",
                    "PaymentNoCharge",
                    "PaymenyPrcard",
                    "PaymentUnknown"]].assign(const=1)
model = model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Tips   R-squared:                       0.598
Model:                            OLS   Adj. R-squared:                  0.598
Method:                 Least Squares   F-statistic:                 6.979e+04
Date:                Wed, 27 Mar 2024   Prob (F-statistic):               0.00
Time:                        14:59:34   Log-Likelihood:            -1.0013e+06
No. Observations:              423032   AIC:                         2.003e+06
Df Residuals:                  423022   BIC:                         2.003e+06
Df Model:                           9
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
TripMiles             0.1931      0.001    302.013      0.000       0.192       0.194
Tolls                 0.0852      0.007     11.842      0.000       0.071       0.099
Extras                0.1035      0.001    139.936      0.000       0.102       0.105
PaymentCreditCard     4.9328      0.010    486.910      0.000       4.913       4.953
PaymentDispute       -0.0374      0.226     -0.165      0.869      -0.481       0.406
PaymentMobile         3.4103      0.013    271.606      0.000       3.386       3.435
PaymentNoCharge       0.1314      0.158      0.831      0.406      -0.178       0.441
PaymenyPrcard        -0.5375      0.014    -39.207      0.000      -0.564      -0.511
PaymentUnknown        0.0043      0.019      0.228      0.820      -0.033       0.041
const                -0.9964      0.008   -127.970      0.000      -1.012      -0.981
==============================================================================
Omnibus:                   343978.075   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         85831530.987
Skew:                           3.027   Prob(JB):                         0.00
Kurtosis:                      72.519   Cond. No.                         578.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [147…

```
# Calculating Residual Standard Error of non multicolinear model
results.resid.std(ddof=X.shape[1])
```

Out[147… 2.580546610501322

In [148…

```
# Creating mlr_taxi dataframe with tips as independent variable
mlr_taxi = taxi[["TripMiles",
                "Tolls",
                "Extras",
                "PaymentCreditCard",
                "PaymentDispute",
                "PaymentMobile",
                "PaymentNoCharge",
                "PaymenyPrcard",
                "PaymentUnknown",
                "Tips"]]
```

In [149…

```
# Dropping PaymentDispute (P-Value = 0.869) and running the model again

# Set dependent variable
y = mlr_taxi.Tips
# Set independent variables
X = mlr_taxi = taxi[["TripMiles",
                    "Tolls",
                    "Extras",
                    "PaymentCreditCard",
                    "PaymentMobile",
                    "PaymentNoCharge",
```

```
                   "PaymenyPrcard",
                   "PaymentUnknown"]].assign(const=1)
model = model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Tips   R-squared:                       0.598
Model:                            OLS   Adj. R-squared:                  0.598
Method:                 Least Squares   F-statistic:                 7.851e+04
Date:                Wed, 27 Mar 2024   Prob (F-statistic):               0.00
Time:                        14:59:34   Log-Likelihood:            -1.0013e+06
No. Observations:              423032   AIC:                         2.003e+06
Df Residuals:                  423023   BIC:                         2.003e+06
Df Model:                           8
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
TripMiles             0.1931      0.001    302.020      0.000       0.192       0.194
Tolls                 0.0852      0.007     11.842      0.000       0.071       0.099
Extras                0.1035      0.001    139.940      0.000       0.102       0.105
PaymentCreditCard     4.9328      0.010    487.051      0.000       4.913       4.953
PaymentMobile         3.4103      0.013    271.655      0.000       3.386       3.435
PaymentNoCharge       0.1315      0.158      0.832      0.406      -0.178       0.441
PaymenyPrcard        -0.5374      0.014    -39.209      0.000      -0.564      -0.511
PaymentUnknown        0.0043      0.019      0.230      0.818      -0.032       0.041
const                -0.9965      0.008   -128.038      0.000      -1.012      -0.981
==============================================================================
Omnibus:                   343978.175   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         85831587.366
Skew:                           3.027   Prob(JB):                         0.00
Kurtosis:                      72.519   Cond. No.                         403.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [150…

```
# Creating mlr_taxi dataframe with tips as independent variable
mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentDispute",
                   "PaymentMobile",
                   "PaymentNoCharge",
                   "PaymenyPrcard",
                   "PaymentUnknown",
                   "Tips"]]
```

In [151…

```
# Dropping PaymentUnknown  (P-Value = 0.818) and running the model again

# Set dependent variable
y = mlr_taxi.Tips
# Set independent variables
X = mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentMobile",
                   "PaymenyPrcard",
                   "PaymentNoCharge"]].assign(const=1)
model = model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Tips   R-squared:                       0.598
Model:                            OLS   Adj. R-squared:                  0.598
Method:                 Least Squares   F-statistic:                 8.972e+04
Date:                Wed, 27 Mar 2024   Prob (F-statistic):               0.00
Time:                        14:59:34   Log-Likelihood:            -1.0013e+06
No. Observations:              423032   AIC:                         2.003e+06
Df Residuals:                  423024   BIC:                         2.003e+06
Df Model:                           7
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
```

```
TripMiles              0.1931      0.001    302.276      0.000       0.192       0.194
Tolls                  0.0852      0.007     11.841      0.000       0.071       0.099
Extras                 0.1035      0.001    140.185      0.000       0.102       0.105
PaymentCreditCard      4.9321      0.010    507.090      0.000       4.913       4.951
PaymentMobile          3.4096      0.012    279.056      0.000       3.386       3.434
PaymenyPrcard         -0.5381      0.013    -40.250      0.000      -0.564      -0.512
PaymentNoCharge        0.1308      0.158      0.828      0.408      -0.179       0.441
const                 -0.9958      0.007   -137.114      0.000      -1.010      -0.982
==============================================================================
Omnibus:                   343978.240   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         85831751.605
Skew:                           3.027   Prob(JB):                         0.00
Kurtosis:                      72.519   Cond. No.                         403.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [152…

```
# Creating mlr_taxi dataframe with tips as independent variable
mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentDispute",
                   "PaymentMobile",
                   "PaymentNoCharge",
                   "PaymenyPrcard",
                   "PaymentUnknown",
                   "Tips"]]
```

In [153…

```
# Dropping PaymentNoCharge  (P-Value = 0.408) and running the model again

# Set dependent variable
y = mlr_taxi.Tips
# Set independent variables
X = mlr_taxi = taxi[["TripMiles",
                   "Tolls",
                   "Extras",
                   "PaymentCreditCard",
                   "PaymentMobile",
                   "PaymenyPrcard"]].assign(const=1)
model = model = sm.OLS(y, X)
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Tips   R-squared:                       0.598
Model:                            OLS   Adj. R-squared:                  0.598
Method:                 Least Squares   F-statistic:                 1.047e+05
Date:                Wed, 27 Mar 2024   Prob (F-statistic):               0.00
Time:                        14:59:34   Log-Likelihood:            -1.0013e+06
No. Observations:              423032   AIC:                         2.003e+06
Df Residuals:                  423025   BIC:                         2.003e+06
Df Model:                           6
Covariance Type:            nonrobust
=====================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------------
TripMiles              0.1931      0.001    302.275      0.000       0.192       0.194
Tolls                  0.0853      0.007     11.843      0.000       0.071       0.099
Extras                 0.1035      0.001    140.188      0.000       0.102       0.105
PaymentCreditCard      4.9319      0.010    507.282      0.000       4.913       4.951
PaymentMobile          3.4094      0.012    279.113      0.000       3.385       3.433
PaymenyPrcard         -0.5384      0.013    -40.275      0.000      -0.565      -0.512
const                 -0.9956      0.007   -137.192      0.000      -1.010      -0.981
==============================================================================
Omnibus:                   343979.062   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):         85831022.489
Skew:                           3.027   Prob(JB):                         0.00
Kurtosis:                      72.519   Cond. No.                         41.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

In [154…

```
# All P-values under 0.05. This is the final model.
```

E. Data Summary

In [155…

```
# Calculating Residual Standard Error of final model
results.resid.std(ddof=X.shape[1])
```

Out[155… 2.580539794112331

In [156…

```
#### Regression Equation
```

After addressing multicollinearity issues and removing non statistically significant explanatory variables the final model had six explanatory variables: TripMiles, Tolls, Extras, PaymentCreditCard, PaymentMobile, and PaymentPrcard. Each explanatory variable had a P-Value of 0.00, indicating it was statically significant to Tips. The adjusted R-squared of the final model was low at 0.598. This means the remaining explanatory variables only account for 60% of the variance found in Tips. This is not a good score, and indicates that there are other variables not included in the final model that are influencing Tips. The residual standard error of the final model was 2.58. A lower residual standard error score is better.

The initial model had a much higher Adjusted R-squared value of 0.994. This could mean that some of the explanatory variables removed to fix multicollinearity issues may have explained more variance in Tips. Since multicollinearity was present in the initial model, the adjusted R-squared score of that model may not be reliable. The residual standard error of 0.317 Comparing the residual standard error scores of the initial and final model revealed the initial model was a “better” model since it had a lower residual standard of error.

While the final model did not have multicollinearity issues, the residual plots revealed heteroskedasticity issues in TripMiles, Tolls, Extras, and PaymentCreditCard. Heteroskedasticity issues are apparent in the cone shape or reverse cone shape the data points form in the visual plots. It indicates that the data violated the homoscedasticity assumption. Heteroskedasticity can be caused by data with wide ranges of values. I observed this during data preparation, particularly while addressing outliers. I could not reduce the range of values significantly without compromising the integrity of the data and producing a model with inaccurate results. Outliers were also visible in the residual plots of TripMiles, Extras, PaymentMobile, and PaymentprCard.

Based on the bivariate visualizations created with Tips and the explanatory variables during data preparation, I suspect the final model was over reduced and some of the explanatory variables present in the initial model did influence Tips, but I cannot prove this statistically with the models produced in this analysis. The final model is not very accurate and violates the assumptions of multiple linear regression.

Model Comparison

In [157…

```
#Initial Model: adj. r squared: 0.994
#Reduced Model: adj. r squared: 0.594

#Initial Model Residual Standard Error: 0.317
#Final Model Residual Standard Error: 2.580

# Initial model was better due to higher adj. r squared and lower residual styandard error.
```

Multicolinearity Check - VIF

In [158…

```
# Running VIF on reduced models explanatory variables
X = mlr_taxi = taxi[["TripMiles",
                    "Tolls",
                    "Extras",
                    "PaymentCreditCard",
                    "PaymentMobile",
                    "PaymenyPrcard"]]
vif_df = pd.DataFrame()
vif_df['variable'] = X.columns
vif_df['VIF'] = [variance_inflation_factor(X.values, i)
for i in range(X.shape[1])]
print(vif_df)
```

```
            variable       VIF
0          TripMiles  2.022621
1              Tolls  1.008526
2             Extras  1.370567
3  PaymentCreditCard  1.556957
4      PaymentMobile  1.061081
5      PaymenyPrcard  1.221375
```

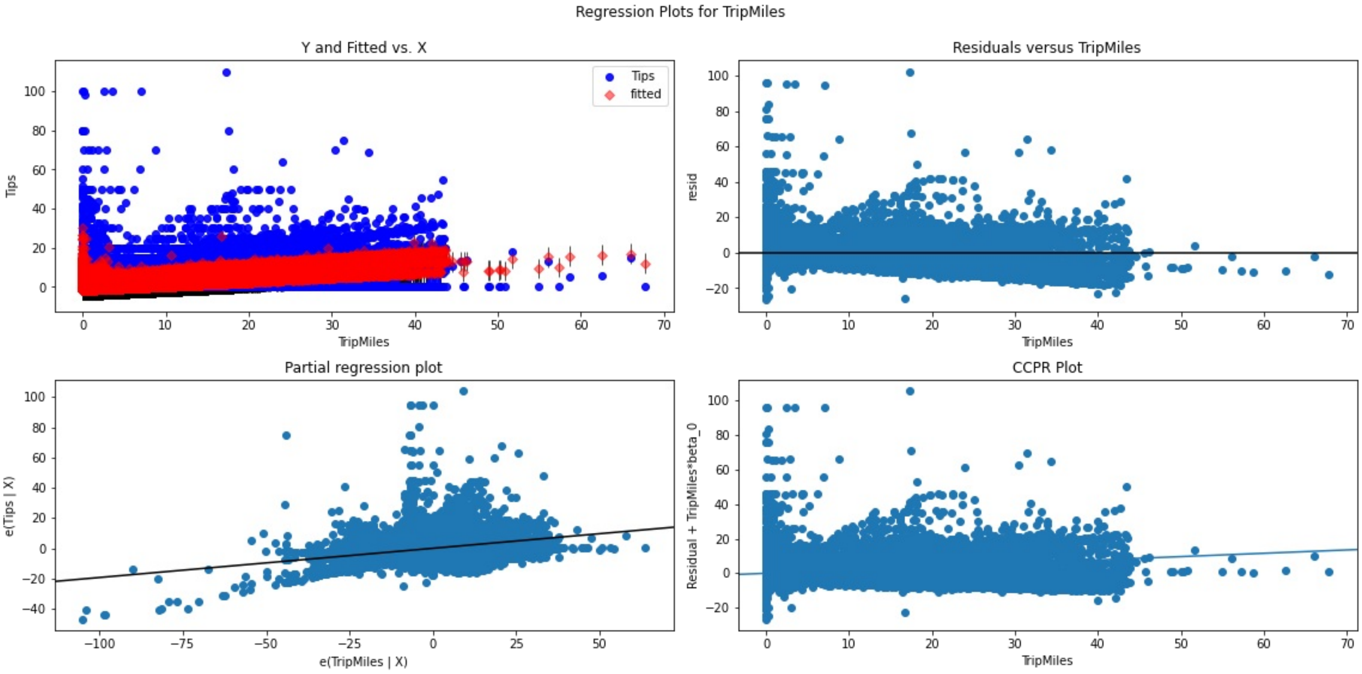
In [159…

```
# All VIF's are under 5.0, no multicolinearity detected
```

Residual Plots

In [160…

```python
# Trip Miles Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'TripMiles', fig=fig);
```

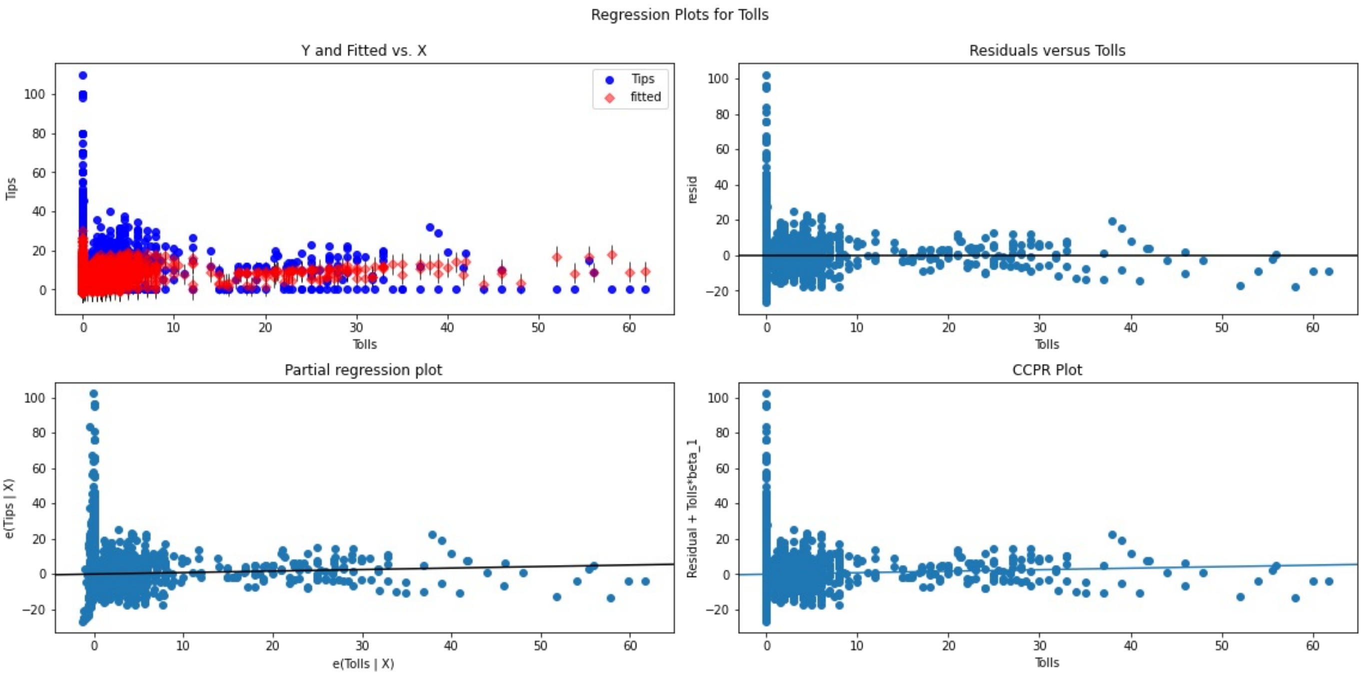


In [161…

```python
# Outliers are present, heteroskedasticity apparant with cone shape in fitted plot
```

In [162…

```python
# Tolls Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'Tolls', fig=fig);
```

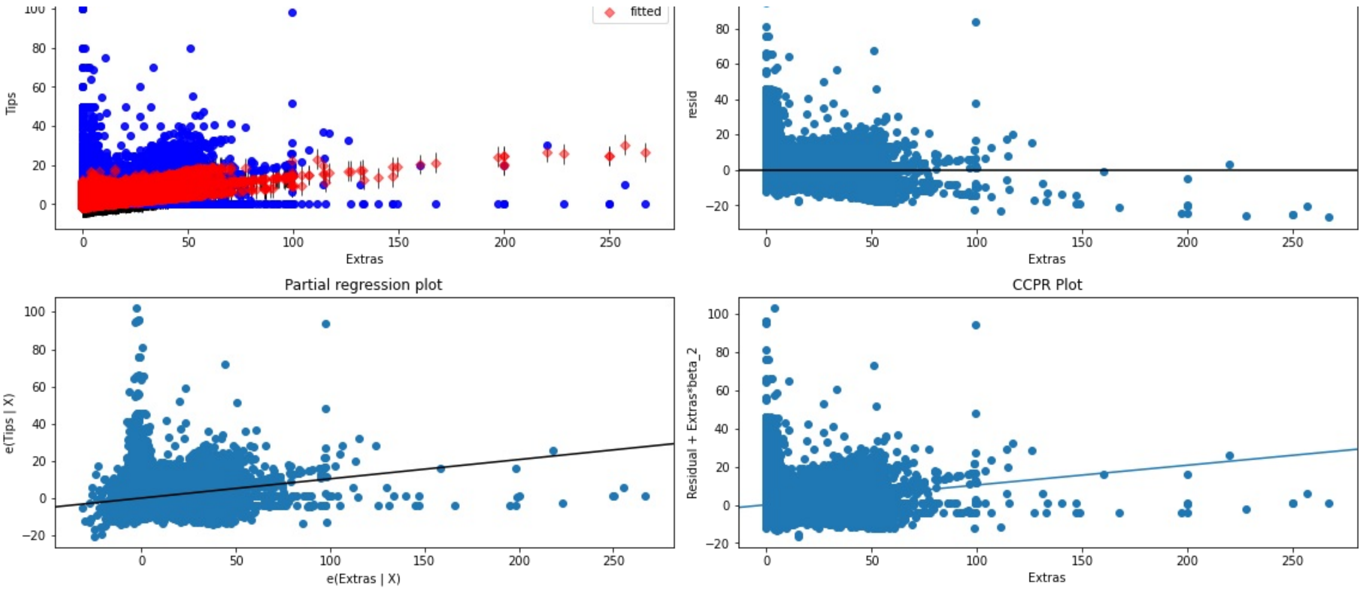


In [163…

```python
# heteroskedasticity issues with the data creating a reverse cone as values increase in fitted plot.
```

In [164…

```python
# Extras Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'Extras', fig=fig);
```

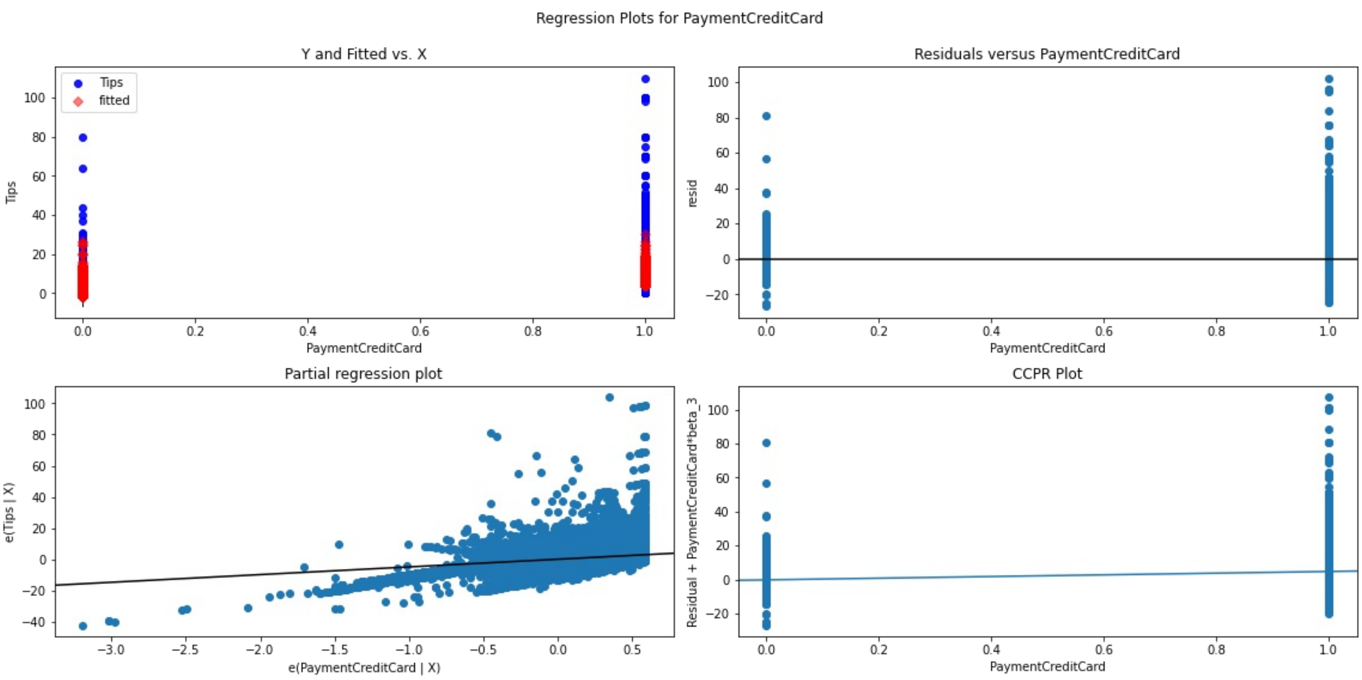


In [165]:

```
# outliers visible at Extras $100 point, heteroskedasticity cone shape funnels as Extras increase
```

In [166]:

```
# PaymentCreditCard Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'PaymentCreditCard', fig=fig);
```

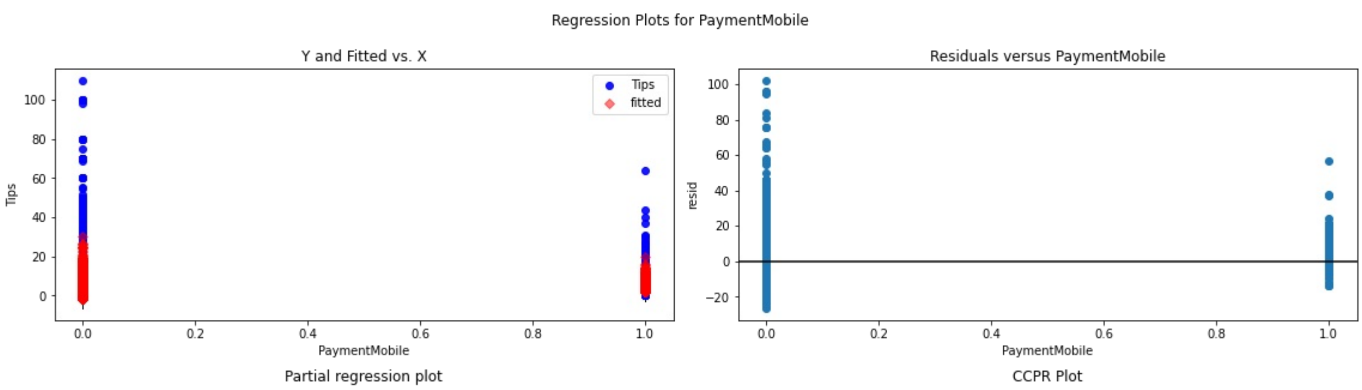


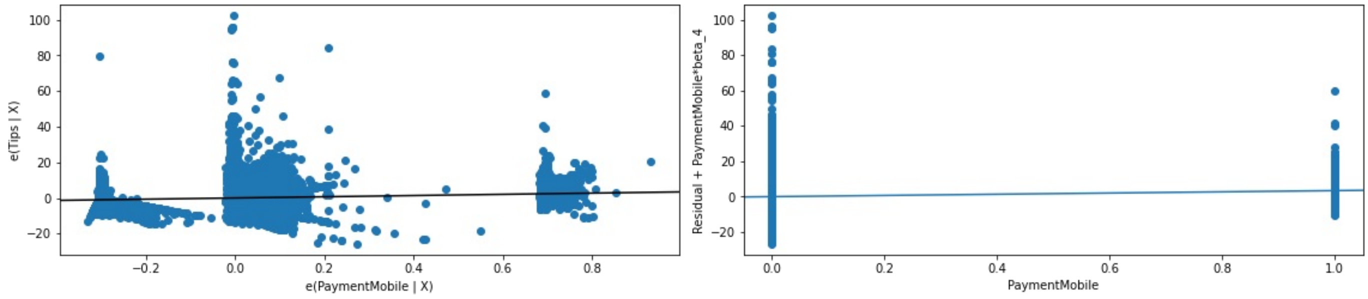
In [167]:

```
# cone shape forms in partial regression plot - heteroskedasticity issues
```

In [168]:

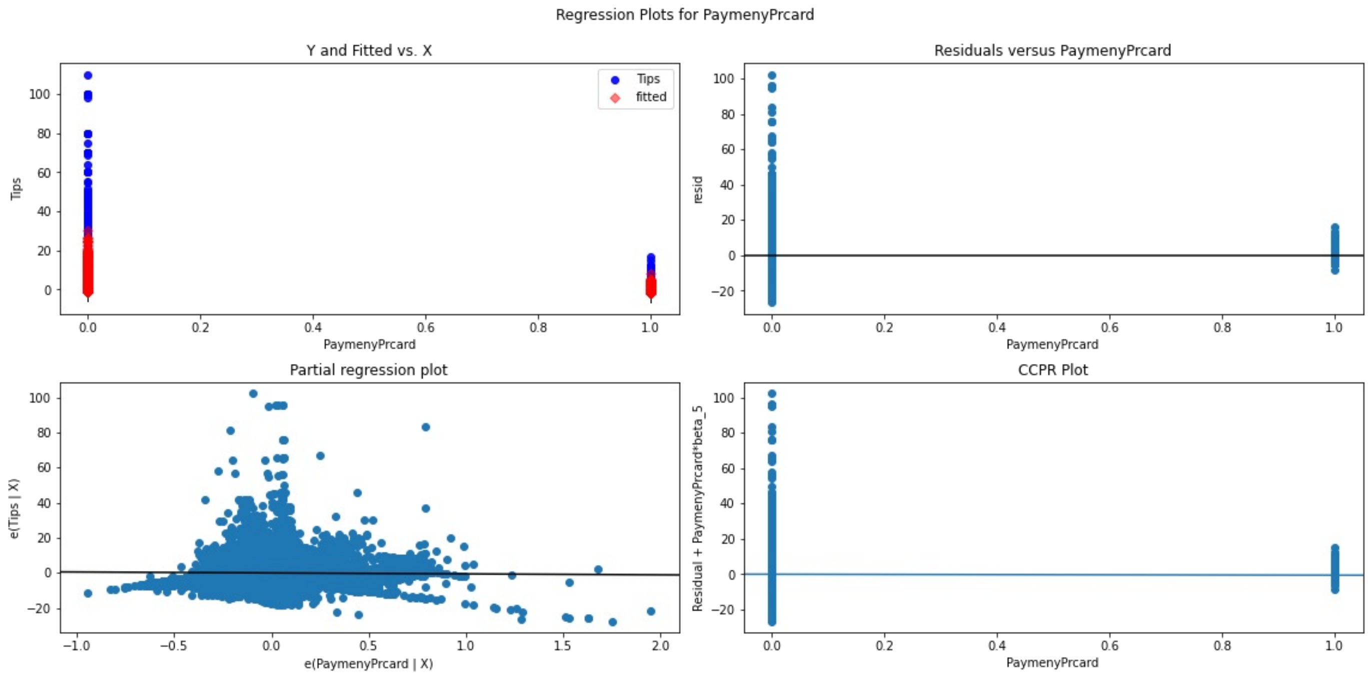
```
# PaymentMobile Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'PaymentMobile', fig=fig);
```





In [169…
```
# multiple data cluster points on partial regression plots, outliers apparant around 0 on X-axis
```

In [170…
```
# PaymenyPrcard Residuals
fig = plt.figure(figsize = [16,8])
sm.graphics.plot_regress_exog(results, 'PaymenyPrcard', fig=fig);
```



In [171…
```
# partial regression plot shows data point clustering and outliers at multiple points on the x-axis
```

In [172…
```
# Saving mlr dataframe to CSV
mlr_taxi_csv_data = mlr_taxi.to_csv('mlrchurn.csv', index = True)
```

E. Recommended Course of Action and Proposal of Future Studies

Since the multiple linear regression model had low accuracy as indicated by the adjusted r-score and violated the homoscedasticity assumption of multiple linear regression, I cannot recommend the city of Chicago or any taxi companies operating inside the city use the results of this model to make business decisions. Per my research question, the null hypothesis was proven. Multiple linear regression was not able to accurately establish correlations between Tips and trip miles, trip duration, tolls, taxi fare, or payment method in this analysis.

The summary statistics and visualization did indicate customers tipped more when paying by credit card. I would recommend taxi companies operating in the city of Chicago provide a small bonus of $0.10 every time a driver accepts a non-credit card payment to compensate for any lost tips. This may increase driver satisfaction and stop drivers from quitting to become independent contractors with Uber or Lyft.

One limitation of this analysis was the data had many outliers and a wide range of values. Multiple linear regression is sensitive to these things. The model might not have been able to handle the outliers that were legitimate data values. If the city of Chicago and taxi companies operating within it wish to confirm correlation between Tips and other explanatory variables in the future, I would recommend creating another multiple linear regression model after the data has been transformed using a log transformation. A log transformation would coerce the data into a normal bell curve distribution. Since multiple linear regression models handle normally distributed data well, the resulting may be more accurate. Log transforming the data may also address multicollinearity issues.

Another potential future study of this data would be a time series analysis. The city of Chicago updates this dataset every month on the 7th of the month. Creating time series analyses of the data month by month after it has been updated to include the previous month's taxi trips could provide Chicago taxi companies with information such certain times of year when customers ride taxis more often. A time series analysis could be run on the data as is to determine if there are certain times of day when customers request taxi rides most frequently.

F. Code Sources

Imputing median for outlier values:

https://stackoverflow.com/questions/55268364/how-to-replace-outliers-with-median-in-pandas-dataframe

Dropping extreme values https://www.codeease.net/programming/python/pandas-how-to-drop-rows-with-extreme-values-in-a-single-column

Addressing Multicolinearity https://www.statology.org/multicollinearity-in-python/

Residual Standard Error https://techhelpnotes.com/residual-standard-error-of-a-regression-in-python/

F. Academic Sources

GeeksforGeeks. (2022). Data Visualizations in Python using Matplotlib and Seaborn. GeeksforGeeks. https://www.geeksforgeeks.org/data-visualisation-in-python-using-matplotlib-and-seaborn/

Katari, K. (2021, December 16). Multiple Linear Regression model using Python: Machine Learning. Medium. https://towardsdatascience.com/multiple-linear-regression-model-using-python-machine-learning-d00c78f1172a

Knowles, J. (2017, June 2). Cab crash: Is Chicago's cab industry on the verge of collapse? ABC7 Chicago. https://abc7chicago.com/chicago-taxis-cabs-taxi-medallions-lyft/2061655/

Magaga, A. M. (2021, November 12). Identifying, cleaning, and replacing outliers: Titanic dataset. Medium. https://medium.com/analytics-vidhya/identifying-cleaning-and-replacing-outliers-titanic-dataset-20182a062893

Potters, C. (2023). R-Squared vs. Adjusted R-Squared: What's the Difference? Investopedia. https://www.investopedia.com/ask/answers/012615/whats-difference-between-rsquared-and-adjusted-rsquared.asp#:~:text=The%20most%20obvious%20difference%20between,and%20R%2Dsquared%20does%20not

Sutton, B. (2022). Scikit-learn vs. StatsModels: Which, why, and how? The Data Incubator. https://www.thedataincubator.com/blog/2022/12/01/scikit-learn-vs-statsmodels/

What is NumPy? — NumPy v1.25 Manual. (n.d.). https://numpy.org/doc/stable/user/whatisnumpy.html

Zach. (2021). The Five Assumptions of Multiple Linear Regression. Statology. https://www.statology.org/multiple-linear-regression-assumptions/