Analysis completed by Stephen Stark

Jumpman23 - Exploratory Data Analysis

Jumpman23 is an on-demand delivery platform connecting “Jumpmen” and customers purchasing a variety of goods. Jumpman23 will send Jumpmen to merchants to purchase and pickup any items requested by the customer. Whenever possible, Jumpman23 will order the requested items ahead to save the Jumpmen time. Each time a Jumpman23 delivery is completed, a record is saved to the Jumpman23 database that contains information about that delivery. Jumpman23 is growing fast and has just launched in its newest market -- New York City.

Objective

The objective of this notebook is to answer the questions:

- How are things going in New York?
- Are there data integrity issues?
    - If so, where are they and how do they impact the analysis?

I will use the dataset provided by Postmates.

Summary

Understanding the Delivery Process

A customer orders something from a business such as a restaurant or grocery store. Jumpman23 connects that customer with a Jumpman who will then travel to go pickup the item. The Jumpman picks up the item to bring back to the customer. The Jumpman travels by foot, bike, car, scooter, or other method to deliver the item to the customer. Once the item is delivered, a record is created with the following journey attribute information.

Understanding Delivery Attributes

- **Job_ID:** a unique identifier of a delivery
- **Customer_id:** a unique identifier for the Jumpman23 customer
- **Jumpman_id:** a unique identifier for the Jumpman who completed the delivery
- **vehicle_type:** The method of transport the Jumpman used to complete the delivery
- **pickup_place:** The name of the Pickup location
- **place_category:** A categorization of the Pickup location
- **Item_name:** the name of the item requested
- **Item_quantity:** how many of that item was requested
- **Item_category_name:** categorization provided by the merchant, think “appetizers”, “soups” etc
- **How_long_it_took_to_order:** how long it took to place the order [interval]
- **pickup_lat:** the coordinates of the pickup location
- **pickup_lon:** the coordinates of the pickup location
- **dropoff_lat:** the coordinations of the dropoff location
- **dropoff_lon:** the coordinations of the dropoff location
- **when_the_delivery_started:** localized timestamp representing when the delivery began
- **when_the_Jumpman_arrived_at_pickup:** localized timestamp representing when the Jumpman arrived at the pickup location
- **when_the_Jumpman_left_pickup:** localized timestamp representing when the Jumpman left the pickup location
- **when_the_Jumpman_arrived_at_dropoff :** localized timestamp representing when the Jumpman reached the customer

Import Neccesary Dependencies

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import pyplot
import numpy as np
import seaborn as sns
from collections import Counter
import haversine as hs
import os as os
from datetime import datetime, timedelta
import re
import missingno as msno
import geopy
from uszipcode import SearchEngine
import time

%matplotlib inline
```

In [2]:
```
#ignore warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
os.listdir()
```

Out[3]:
```
['.DS_Store',
 'Stephen_Stark_Analysis.pptx',
 'README.md',
 'img',
 'tableau_exports',
 '~$Stephen_Stark_Analysis.pptx',
 '.ipynb_checkpoints',
 'Stephen_Stark_Jumpman23Analysis.ipynb',
 '.git']
```

In [4]:
```
df = pd.read_csv('../Jumpman23/analyze_me.csv')
```

Understand the data

In order to answer the question of 'how are things going in NYC', we need to first spot check the data. We know there are potential data integrity issues. Lets look column by column to determine what sort of analysis we can do.

In [5]:
```
print(df.shape)
print(df.info())
```

```
(5983, 18)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5983 entries, 0 to 5982
Data columns (total 18 columns):
delivery_id                             5983 non-null int64
customer_id                             5983 non-null int64
jumpman_id                              5983 non-null int64
vehicle_type                            5983 non-null object
pickup_place                            5983 non-null object
place_category                          5100 non-null object
item_name                               4753 non-null object
item_quantity                           4753 non-null float64
item_category_name                      4753 non-null object
how_long_it_took_to_order               3038 non-null object
pickup_lat                              5983 non-null float64
pickup_lon                              5983 non-null float64
dropoff_lat                             5983 non-null float64
dropoff_lon                             5983 non-null float64
when_the_delivery_started               5983 non-null object
when_the_Jumpman_arrived_at_pickup      5433 non-null object
when_the_Jumpman_left_pickup            5433 non-null object
when_the_Jumpman_arrived_at_dropoff     5983 non-null object
dtypes: float64(5), int64(3), object(10)
memory usage: 841.5+ KB
None
```

In [6]:
```
df.iloc[2]
```

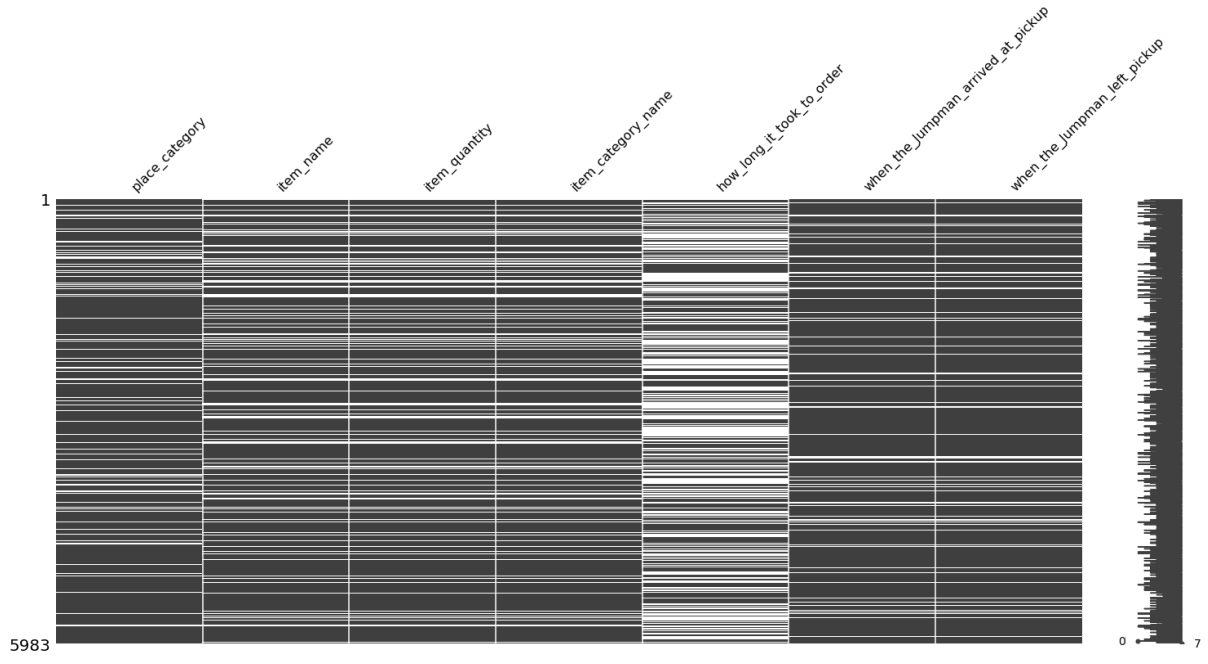
Out[6]:
```
delivery_id                                    1476547
customer_id                                      83095
jumpman_id                                      132725
vehicle_type                                   bicycle
pickup_place                                Bareburger
place_category                                  Burger
item_name                                   Bare Sodas
item_quantity                                        1
item_category_name                              Drinks
how_long_it_took_to_order                00:06:44.541717
pickup_lat                                      40.7285
pickup_lon                                     -73.9984
dropoff_lat                                     40.7286
dropoff_lon                                    -73.9951
when_the_delivery_started           2014-10-28 21:39:52.654394
when_the_Jumpman_arrived_at_pickup  2014-10-28 21:37:18.793405
when_the_Jumpman_left_pickup         2014-10-28 21:59:09.98481
when_the_Jumpman_arrived_at_dropoff 2014-10-28 22:04:40.634962
Name: 2, dtype: object
```

In [7]:
```
#Count of N/A or Null values in each column
df.isna().sum()
```

Out[7]:
```
delivery_id                            0
customer_id                            0
jumpman_id                             0
vehicle_type                           0
pickup_place                           0
place_category                       883
item_name                           1230
item_quantity                       1230
item_category_name                  1230
how_long_it_took_to_order           2945
pickup_lat                             0
pickup_lon                             0
dropoff_lat                            0
dropoff_lon                            0
when_the_delivery_started              0
when_the_Jumpman_arrived_at_pickup   550
when_the_Jumpman_left_pickup         550
when_the_Jumpman_arrived_at_dropoff    0
dtype: int64
```

In [8]:
```
#visualization of missing data
missing_data_cols = ['place_category',
                     'item_name',
                     'item_quantity',
                     'item_category_name',
                     'how_long_it_took_to_order',
                     'when_the_Jumpman_arrived_at_pickup',
                     'when_the_Jumpman_left_pickup']

msno.matrix(df[missing_data_cols])
plt.show()
```



'delivery_id','customer_id','jumpman_id'

In [9]:
```
print('Count of records:', len(df))
print('---')
print('Count of unique elements in each id column:')

id_cols = ['delivery_id','customer_id','jumpman_id']
for i in id_cols:
    print(i,':', df[i].nunique())
```

```
Count of records: 5983
---
Count of unique elements in each id column:
delivery_id : 5214
customer_id : 3192
jumpman_id : 578
```

It is important to note the total number of records 5983 is greater than the unique elements for each id. I would expect customer_id and jumpman_id to be used mutliple times. However, I want to look into why the same delivery_id has been used multiple times.

In [10]:
```
df[df['delivery_id'].duplicated()].head(3)
```

Out[10]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_name
82	1314550	348787	119813	bicycle	Otto Enoteca Pizzeria	Italian	Prosciutto Arugula
207	1332526	48677	152676	bicycle	Shake Shack	Burger	Smoke Shack
244	1319971	94027	119255	walker	Trader Joe's	Grocery Store	Organic Autumn Whea

Based on the query below, it looks like multiple items from the same order are broken out on different records. I would expect all the other attributes to be the same for all cases. One way I could handle this is to merge the items into a list for each record. For the purposes of this analysis, I'll drop the duplicate records as I don't see a meaningful reason for including them in the analysis.

In [11]:
```
#sample duplicate row
df[df['delivery_id']==1272701]
```

Out[11]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_nan
1008	1272701	81085	112646	bicycle	Mighty Quinn's BBQ	BBQ	Brisk
5080	1272701	81085	112646	bicycle	Mighty Quinn's BBQ	BBQ	Housema Iced T

In [12]:
```
#duplicate row dataset
df[df['delivery_id'].duplicated()].sort_values(by='delivery_id')
```

Out[12]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_nan
5080	1272701	81085	112646	bicycle	Mighty Quinn's BBQ	BBQ	Housema Iced T
2299	1274248	208020	60149	car	Murray's Falafel	Middle Eastern	Morocc Cigars p
2986	1274248	208020	60149	car	Murray's Falafel	Middle Eastern	Watermel
5386	1274328	255435	23359	bicycle	Lure Fishbar	Seafood	Ki Salm
4578	1274372	82041	133293	bicycle	Parm	Italian	Chick Par
...	...	...	...	...	...	...	
3614	1490188	166368	174143	motorcycle	Prosperity Dumpling	Chinese	Chives a Pc Dumplin in So
4119	1490188	166368	174143	motorcycle	Prosperity Dumpling	Chinese	Vegetab and Pc Dumplin in So
4983	1490744	52256	38597	bicycle	Han Dynasty	Chinese	Dan D Nooc
4074	1490744	52256	38597	bicycle	Han Dynasty	Chinese	Bok Ch with Bla Mushroor
1988	1491424	391367	172130	walker	Veselka	Russian	Small Pla of Pierog

769 rows × 18 columns

In [13]:
```
#drop duplicate rows based on delivery_id column
df = df.drop_duplicates(subset=['delivery_id'])
```

'vehicle_type','pickup_place','place_category','item_name','item_category_name'

The integrity of the following columns can be assessed by simply looking at the counts of the unique values in each set, as well as the size of the set itself.

In [14]:
```
select_cols = ['vehicle_type',
               'pickup_place',
               'place_category',
               'item_name',
               'item_category_name']

for col in select_cols:
    print(pd.DataFrame(df[col].value_counts()))
```

	vehicle_type
bicycle	3740
car	1050
walker	234
van	69
scooter	64
truck	38
motorcycle	19

	pickup_place
Shake Shack	266
Momofuku Milk Bar	162
The Meatball Shop	153
sweetgreen	138
Blue Ribbon Fried Chicken	115
...	...
Little Poland Restaurant	1
The Stanton Social	1
Adrienne's Pizza Bar	1
Sugar and Plumm	1
Minca	1

[898 rows x 1 columns]

	place_category
Italian	437
Burger	395
American	357
Japanese	335
Dessert	277
Chinese	265
Sushi	203
Salad	192
Mexican	165
Grocery Store	130
Bakery	126
BBQ	114
Pizza	94
Juice Bar	91
Indian	80
Fast Food	80
Donut	71
Seafood	69
Drug Store	68
Mediterranean	61
Vegetarian	60
Coffee	60
Deli	56
Middle Eastern	56
Ice Cream	54
Gluten-Free	53
Breakfast	45
Shop	41
South American	33
Steak	31
Thai	31
French	24
Southern	23
Electronics Store	19

```
Promo                          19
Vegan                          19
Korean                         18
Convenience Store              17
Food Truck                     17
Spanish                        12
Asian                          11
Eastern European               10
Department Store                9
Russian                         8
Vietnamese                      6
Caribbean                       6
Office Supplies Store           5
Specialty Store                 5
German                          4
Kids & Baby                     3
Beauty Supply                   2
Clothing                        1
African                         1
Restaurant                      1
Art Store                       1
Book Store                      1
Pet Supplies Store              1
                item_name
Fries                  68
Cheese Fries           29
Chicken                28
Shackburger            27
Shack Burger           26
...                   ...
Tofu Soba Salad         1
Saag Paneer             1
Pad Sew Yew             1
Iron Man MOB            1
Pirozhok                1

[2013 rows x 1 columns]
                        item_category_name
Sides                                  158
Burgers                                133
Appetizers                             122
Sandwiches                             104
Fries                                   94
...                                    ...
Lunch Soups & Salads                     1
Bastilla                                 1
Raw Snacks                               1
MOBs                                     1
Appetizers & Side Orders                 1

[719 rows x 1 columns]
```

'item_quantity'

The integrity of the following columns can be assessed by looking at several measures of the statistical distribution of each set.

In [15]:
```
df['item_quantity'].describe()
```

Out[15]:
```
count    3984.000000
mean        1.245231
std         0.781632
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max        16.000000
Name: item_quantity, dtype: float64
```

Pickup & Dropoff Locations

Assessment of pickup and dropoff locations

In [16]:
```
import ipyplot

images_list = ['img/Pickup.png','img/Dropoff.png',
               'img/Pickup_zoom.png','img/Dropoff_zoom.png']

ipyplot.plot_images(images_list, img_width=500)
```

0

img/Pickup.png

1

img/Dropoff.png

2

img/Pickup_zoom.png

3

img/Dropoff_zoom.png

The pickup and dropoff locations all appear to be valid.

'when_the_delivery_started','when_the_Jumpman_arrived_at_pickup','when_the_Ju

Check the min and max values for the relevant timestamp columns. It the data is for the month of October.

In [17]:
```
date_cols = ['when_the_delivery_started',
             'when_the_Jumpman_arrived_at_pickup',
             'when_the_Jumpman_left_pickup',
             'when_the_Jumpman_arrived_at_dropoff']

time_cols = ['how_long_it_took_to_order']
for i in date_cols:
    df[i] = pd.to_datetime(df[i])

for i in time_cols:
    df[i] = pd.to_timedelta(df[i])

for col in date_cols:
    print(col+":",
          df[col].min(),
          ",",
          df[col].max())

for col in time_cols:
    print(col+":",
          df[col].min(),
          ",",
          df[col].max())
```

```
when_the_delivery_started: 2014-10-01 00:07:58.632482 , 2014-10-30 23:0
8:43.481900
when_the_Jumpman_arrived_at_pickup: 2014-10-01 00:39:31.086322 , 2014-1
0-30 23:10:31.062088
when_the_Jumpman_left_pickup: 2014-10-01 00:59:57.522402 , 2014-10-30 2
3:23:51.143279
when_the_Jumpman_arrived_at_dropoff: 2014-10-01 00:30:21.109149 , 2014-
10-30 23:29:44.866438
how_long_it_took_to_order: 0 days 00:01:22.997519 , 0 days 01:13:13.266
118
```

In [18]:
```
df['how_long_it_took_to_order'] = df['how_long_it_took_to_order'].astype
('timedelta64[m]')
df['how_long_it_took_to_order'].describe()
```

Out[18]:
```
count    2579.000000
mean        7.202404
std         5.710237
min         1.000000
25%         4.000000
50%         6.000000
75%         9.000000
max        73.000000
Name: how_long_it_took_to_order, dtype: float64
```

Missing Data

Now that we've gone through all of the columns, we'll need to come up with a strategy for how to handle the remaining missing information.

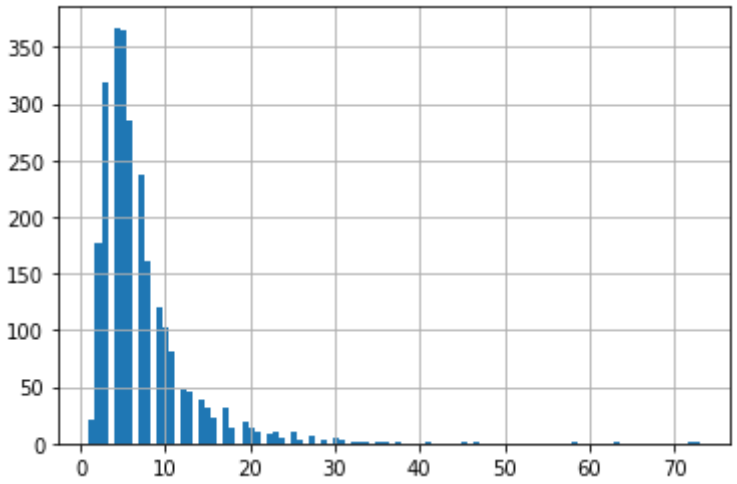
In [19]:
```
df.isnull().sum()
```

Out[19]:
```
delivery_id                          0
customer_id                          0
jumpman_id                           0
vehicle_type                         0
pickup_place                         0
place_category                     841
item_name                         1230
item_quantity                     1230
item_category_name                1230
how_long_it_took_to_order         2635
pickup_lat                           0
pickup_lon                           0
dropoff_lat                          0
dropoff_lon                          0
when_the_delivery_started            0
when_the_Jumpman_arrived_at_pickup   495
when_the_Jumpman_left_pickup         495
when_the_Jumpman_arrived_at_dropoff    0
dtype: int64
```

I plan to make an assumption of a value to fill for the missing 'how_long_it_took_to_order' values. First, let's look at a distribution

In [20]:
```
df['how_long_it_took_to_order'].hist(bins=100)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faedf056b90>
```



In [21]:
```
df['place_category'].unique()
```

Out[21]:
```
array(['American', 'Pizza', 'Burger', 'Juice Bar', 'Japanese', 'India
n',
       'Sushi', 'Bakery', nan, 'Mexican', 'BBQ', 'Dessert', 'Vegan',
       'Fast Food', 'Korean', 'Drug Store', 'Italian', 'Grocery Store',
       'Breakfast', 'Coffee', 'Salad', 'Middle Eastern', 'Mediterranea
n',
       'Seafood', 'Russian', 'Southern', 'Vegetarian', 'Deli',
       'South American', 'French', 'Chinese', 'Asian', 'Donut',
       'Gluten-Free', 'Office Supplies Store', 'Convenience Store',
       'Shop', 'Food Truck', 'Clothing', 'German', 'Promo',
       'Electronics Store', 'Steak', 'Ice Cream', 'Thai', 'Spanish',
       'Caribbean', 'Vietnamese', 'Book Store', 'Specialty Store',
       'Eastern European', 'Kids & Baby', 'Department Store',
       'Beauty Supply', 'African', 'Restaurant', 'Art Store',
       'Pet Supplies Store'], dtype=object)
```

In [22]:
```
#check how long it took to order by restaurant category
df.groupby(by = 'place_category')['how_long_it_took_to_order'].describe
().sort_values(by='count', ascending=False)
```

Out[22]:

place_category	count	mean	std	min	25%	50%	75%	max
Italian	382.0	6.772251	4.663812	1.0	4.00	6.0	8.00	29.0
Japanese	315.0	7.336508	5.826124	2.0	4.00	6.0	9.00	45.0
Chinese	235.0	6.314894	5.815062	1.0	4.00	5.0	7.00	72.0
American	194.0	8.025773	5.549995	1.0	5.00	6.0	10.00	31.0
Sushi	189.0	8.645503	4.854943	2.0	5.00	7.0	11.00	27.0
Salad	125.0	7.488000	7.413606	1.0	4.00	6.0	9.00	73.0
Burger	90.0	6.277778	3.932086	1.0	4.00	5.0	7.00	21.0
Indian	67.0	5.865672	4.667410	1.0	3.00	5.0	7.00	29.0
Pizza	64.0	7.781250	8.373179	1.0	4.00	5.5	9.00	63.0
Mexican	62.0	7.435484	5.838316	2.0	3.00	6.0	9.75	27.0
Seafood	56.0	6.946429	6.839510	2.0	4.00	5.0	7.00	47.0
Middle Eastern	52.0	7.576923	4.679308	2.0	4.00	6.0	9.25	26.0
Vegetarian	51.0	9.450980	8.936025	1.0	4.00	6.0	10.00	37.0
Deli	48.0	7.375000	8.078406	2.0	4.00	6.0	8.00	58.0
Mediterranean	48.0	5.750000	3.386456	1.0	3.00	5.0	7.00	16.0
BBQ	32.0	10.843750	7.278379	2.0	5.75	9.0	12.25	32.0
Steak	30.0	7.000000	3.600766	3.0	4.25	7.0	8.00	19.0
South American	28.0	7.071429	5.304984	2.0	3.75	5.0	8.50	24.0
Breakfast	28.0	6.607143	6.051258	2.0	4.00	5.0	7.00	33.0
Thai	28.0	6.642857	4.975602	3.0	4.00	5.0	6.50	23.0
Coffee	27.0	4.888889	2.207214	2.0	3.00	5.0	6.00	11.0
Vegan	15.0	6.466667	4.033196	2.0	3.50	6.0	8.50	15.0
Korean	13.0	7.000000	5.744563	3.0	4.00	6.0	7.00	25.0
Spanish	12.0	7.333333	4.599078	2.0	3.75	6.5	9.25	16.0
Food Truck	11.0	6.727273	4.027180	3.0	3.50	5.0	9.50	15.0
Russian	8.0	5.250000	1.752549	3.0	3.75	5.5	7.00	7.0
Bakery	8.0	9.000000	6.676184	4.0	4.75	5.5	11.50	20.0
Eastern European	7.0	10.714286	6.210590	3.0	6.00	11.0	14.50	20.0
Vietnamese	6.0	5.833333	3.060501	2.0	4.25	5.5	6.75	11.0
Caribbean	5.0	4.200000	3.563706	1.0	2.00	3.0	5.00	10.0
Southern	5.0	8.000000	7.582875	2.0	4.00	5.0	8.00	21.0
German	4.0	7.500000	4.932883	2.0	5.75	7.0	8.75	14.0
French	3.0	8.333333	1.527525	7.0	7.50	8.0	9.00	10.0

place_category	count	mean	std	min	25%	50%	75%	max
Asian	2.0	16.500000	19.091883	3.0	9.75	16.5	23.25	30.0
African	1.0	6.000000	NaN	6.0	6.00	6.0	6.00	6.0
Ice Cream	1.0	31.000000	NaN	31.0	31.00	31.0	31.00	31.0
Juice Bar	1.0	3.000000	NaN	3.0	3.00	3.0	3.00	3.0
Dessert	1.0	9.000000	NaN	9.0	9.00	9.0	9.00	9.0
Art Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Beauty Supply	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Book Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Clothing	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Convenience Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Specialty Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Department Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Donut	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Kids & Baby	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Drug Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Electronics Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Fast Food	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Restaurant	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Promo	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Gluten-Free	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Pet Supplies Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Office Supplies Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Grocery Store	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Shop	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [23]:
```
avg_time_to_order = df['how_long_it_took_to_order'].mean()
avg_time_to_order
```

Out[23]: 7.2024040325707634

It appears there isn't meaningful deviation between the most popular restaurants. Few restaurants with smaller counts have greater deviation. I feel that it is safe to use an average value of ~7.2 minutes to fill the remaining N/As.

In [24]:
```
df['place_category'].fillna('Not Disclosed', inplace=True)
df['item_name'].fillna('Not Disclosed', inplace=True)
df['item_quantity'].fillna('Not Disclosed', inplace=True)
df['item_category_name'].fillna('Not Disclosed', inplace=True)
df['how_long_it_took_to_order'].fillna(avg_time_to_order, inplace=True)
```

In [25]:
```
df.isnull().sum()/len(df)
```

Out[25]:
```
delivery_id                             0.000000
customer_id                             0.000000
jumpman_id                              0.000000
vehicle_type                            0.000000
pickup_place                            0.000000
place_category                          0.000000
item_name                               0.000000
item_quantity                           0.000000
item_category_name                      0.000000
how_long_it_took_to_order               0.000000
pickup_lat                              0.000000
pickup_lon                              0.000000
dropoff_lat                             0.000000
dropoff_lon                             0.000000
when_the_delivery_started               0.000000
when_the_Jumpman_arrived_at_pickup      0.094937
when_the_Jumpman_left_pickup            0.094937
when_the_Jumpman_arrived_at_dropoff     0.000000
dtype: float64
```

In [26]:
```
#Less than 10% of the Jumpman Arrived at Pickup & left Pickup are empty, therefore, I'm comfortable dropping the entries with missing data
df = df.dropna(subset=['when_the_Jumpman_arrived_at_pickup','when_the_Jumpman_left_pickup'])
```

In [27]:
```
df.isnull().sum()
```

Out[27]:
```
delivery_id                             0
customer_id                             0
jumpman_id                              0
vehicle_type                            0
pickup_place                            0
place_category                          0
item_name                               0
item_quantity                           0
item_category_name                      0
how_long_it_took_to_order               0
pickup_lat                              0
pickup_lon                              0
dropoff_lat                             0
dropoff_lon                             0
when_the_delivery_started               0
when_the_Jumpman_arrived_at_pickup      0
when_the_Jumpman_left_pickup            0
when_the_Jumpman_arrived_at_dropoff     0
dtype: int64
```

Feature Engineering

Calculate the distance between the pickup and dropoff locations using Haversine distance. The Haversine distance is the angular distance between two points on the surface of a sphere. It is important to note this is distance is "as the crow flies", not distance such as driving directions on a map.

In [28]:
```
def haversine_distance_miles(lat1, lon1, lat2, lon2):
    r = 3959
    phi1 = np.radians(lat1)
    phi2 = np.radians(lat2)
    delta_phi = np.radians(lat2 - lat1)
    delta_lambda = np.radians(lon2 - lon1)
    a = np.sin(delta_phi / 2)**2 + np.cos(phi1) * np.cos(phi2) *   np.sin
(delta_lambda / 2)**2
    res = r * (2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a)))
    return np.round(res, 2)

df['haversine_distance_mi'] = haversine_distance_miles(df['pickup_lat'],
                                                       df['pickup_lon'],
                                                       df['dropoff_lat'],
                                                       df['dropoff_lon'])
```

In [29]:
```
df.iloc[1]
```

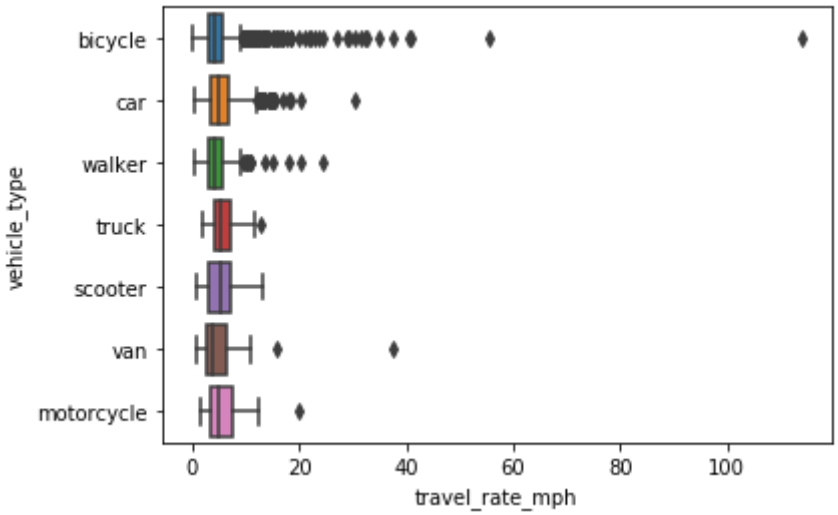
Out[29]:
```
delivery_id                                1476547
customer_id                                  83095
jumpman_id                                  132725
vehicle_type                               bicycle
pickup_place                            Bareburger
place_category                              Burger
item_name                               Bare Sodas
item_quantity                                    1
item_category_name                          Drinks
how_long_it_took_to_order                        6
pickup_lat                                 40.7285
pickup_lon                                -73.9984
dropoff_lat                                40.7286
dropoff_lon                               -73.9951
when_the_delivery_started          2014-10-28 21:39:52.654394
when_the_Jumpman_arrived_at_pickup 2014-10-28 21:37:18.793405
when_the_Jumpman_left_pickup       2014-10-28 21:59:09.984810
when_the_Jumpman_arrived_at_dropoff 2014-10-28 22:04:40.634962
haversine_distance_mi                                    0.17
Name: 2, dtype: object
```

In [30]:
```
#calculate travel time in seconds, minutes, and hours
travel_time_seconds = (df['when_the_Jumpman_arrived_at_dropoff'] - df['w
hen_the_Jumpman_left_pickup']).astype('timedelta64[s]')
df['travel_time_seconds'] = travel_time_seconds
df['travel_time_minutes'] = travel_time_seconds / 60
df['travel_time_hours'] = travel_time_seconds / 60 / 60

#calculate travel rate in mph
travel_rate_mph = df['haversine_distance_mi'] / df['travel_time_hours']
df['travel_rate_mph'] = travel_rate_mph
```

In [31]:
```
sns.boxplot(x="travel_rate_mph", y="vehicle_type", data=df, width=.8)
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7faede579310>



In [32]:
```
df[df['travel_rate_mph'] >50]
```

Out[32]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_nam
1633	1311664	115231	104533	bicycle	Izakaya Ten	Japanese	Bu Kimche
4072	1381438	68942	30743	bicycle	Petco	Pet Supplies Store	N Disclose

2 rows × 23 columns

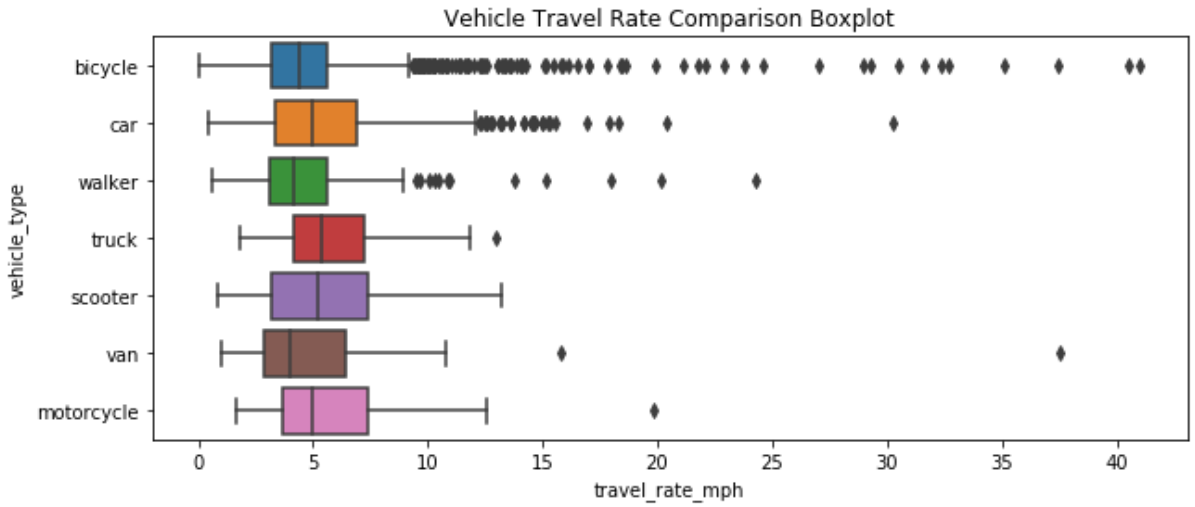
Exploratory Data Analysis (EDA)

In [33]:
```
#drop the record where the bicyclist traveled >100mph as that appears ot
be an error
#Not concerned with high rates (walker, for instance, of 15+ mph) as the
y can take the subway or public transit bus, etc.

i = df[df['travel_rate_mph'] >55].index
df.drop(i, inplace=True)
```

In [34]:
```
import matplotlib.pyplot as plt
plt.subplots(figsize=(10,4))
sns.boxplot(x="travel_rate_mph", y="vehicle_type", data=df, width=.8)
plt.xticks(np.arange(0, 45, 5))
plt.title('Vehicle Travel Rate Comparison Boxplot')
```

Out[34]: Text(0.5, 1.0, 'Vehicle Travel Rate Comparison Boxplot')



In [35]:
```
#difference between leaving pickup and arriving at pickup and how long i
t took to order
diff1 = df['when_the_Jumpman_left_pickup'] - df['when_the_Jumpman_arrive
d_at_pickup'] - df['how_long_it_took_to_order'].astype('timedelta64[m]')

#simply the amount of time the jumpman waited at the restaurant
diff2 = df['when_the_Jumpman_left_pickup'] - df['when_the_Jumpman_arrive
d_at_pickup']

df['Jumpmen_wait_time'] = diff2.astype('timedelta64[m]')
```

In [36]:
```
#seems odd that 370 have a negative value

print(len(diff1))
print(diff1[diff1.astype('timedelta64[m]')<0])
```

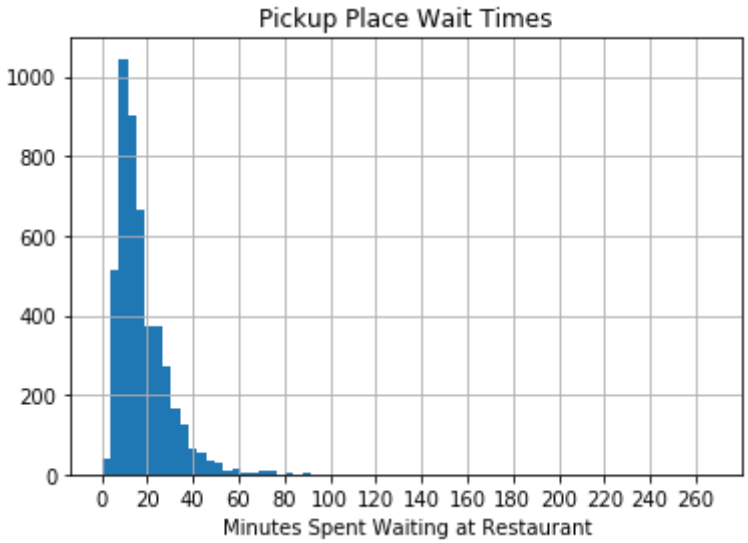
```
4717
1       -1 days +23:57:20.970322
8       -1 days +23:58:49.768211
10      -1 days +23:57:35.599710
45      -1 days +23:54:20.805456
64      -1 days +23:54:18.790127
                   ...
5924    -1 days +23:58:37.487807
5925    -1 days +23:53:50.027812
5960    -1 days +23:59:19.889030
5961    -1 days +23:58:50.123971
5974    -1 days +23:57:25.519233
Length: 371, dtype: timedelta64[ns]
```

In [37]:
```
diff2.astype('timedelta64[m]').describe()
```

Out[37]:
```
count    4717.000000
mean       17.725885
std        11.717126
min         0.000000
25%        10.000000
50%        15.000000
75%        22.000000
max       267.000000
dtype: float64
```

In [38]:
```
diff2.astype('timedelta64[m]').hist(bins=70)
plt.xticks(np.arange(0,270, 20))
plt.title('Pickup Place Wait Times')
plt.xlabel('Minutes Spent Waiting at Restaurant')
```

Out[38]:
```
Text(0.5, 0, 'Minutes Spent Waiting at Restaurant')
```



In [39]:
```
print('minutes: ',diff2.astype('timedelta64[m]').sum())
print('hours: ',diff2.astype('timedelta64[m]').sum()/60)
```

```
minutes:  83613.0
hours:  1393.55
```

In [40]:
```
print('Jumpman wait time <10 mins: ', len(df[df['Jumpmen_wait_time']<10
]))
print('Jumpman wait time >10 mins: ', len(df[df['Jumpmen_wait_time']>10
]))
```

```
Jumpman wait time <10 mins:  1056
Jumpman wait time >10 mins:  3390
```

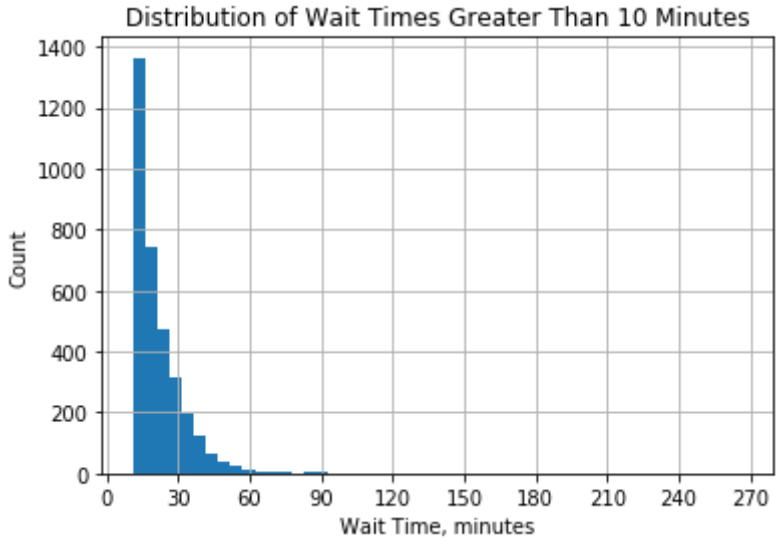
In [41]:
```
round(df[df['Jumpmen_wait_time']>10]['Jumpmen_wait_time'].describe(),1)
```

Out[41]:
```
count    3390.0
mean       21.7
std        11.6
min        11.0
25%        14.0
50%        18.0
75%        26.0
max       267.0
Name: Jumpmen_wait_time, dtype: float64
```

In [42]:
```
df[df['Jumpmen_wait_time']>10]['Jumpmen_wait_time'].hist(bins=50)

plt.xlabel('Wait Time, minutes')
plt.ylabel('Count')
plt.xticks((np.arange(0,300,30)))
plt.title('Distribution of Wait Times Greater Than 10 Minutes')
```

Out[42]:
```
Text(0.5, 1.0, 'Distribution of Wait Times Greater Than 10 Minutes')
```



In [43]:
```
gtr_than_15 = df[df['Jumpmen_wait_time']>15]['Jumpmen_wait_time']
gtr_than_30 = df[df['Jumpmen_wait_time']>30]['Jumpmen_wait_time']
```

In [44]:
```
(gtr_than_30 - 15)
```

Out[44]:
```
7        30.0
12       18.0
35       23.0
62       68.0
65       28.0
         ...
5938     21.0
5945     23.0
5955     23.0
5958     20.0
5966     76.0
Name: Jumpmen_wait_time, Length: 536, dtype: float64
```

In [45]:
```
sum(gtr_than_30 - 15)
```

Out[45]:
```
14240.0
```

In [46]:
```
(gtr_than_30 - 15).mean()
```

Out[46]:
```
26.567164179104477
```

In [47]:
```
diff_15 = gtr_than_15 - 15
```

In [48]:
```
diff_15.sum()
```

Out[48]:
```
25132.0
```

In [49]:
```
print((gtr_than_15).sum())
print((gtr_than_15*.95).sum())
```

```
58342.0
55424.899999999994
```

Count of how many unique items are on the menu

In [50]:
```
len(df['item_name'].unique())
```

Out[50]:
```
1897
```

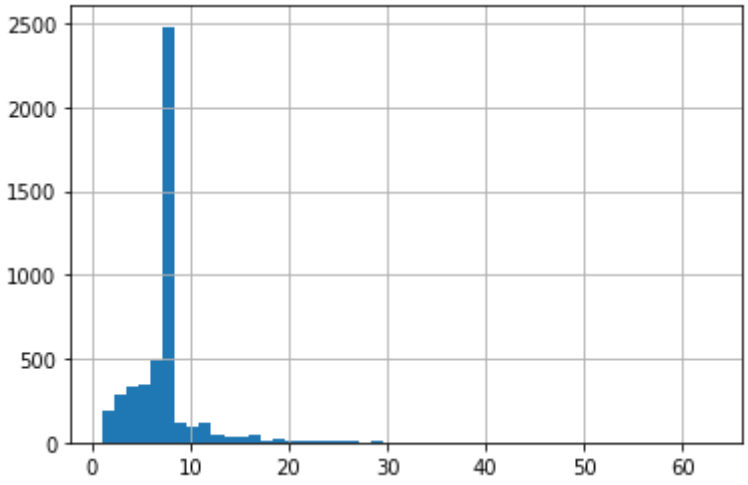
In [ ]:
```

```

Check the distribution for how long it took to order

In [51]:
```
df['how_long_it_took_to_order'].hist(bins=50)
```

Out[51]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faede66d410>
```



In [52]:
```
df['place_category'].value_counts()
```

Out[52]:

Not Disclosed	729
Italian	410
Burger	368
American	335
Japanese	300
Dessert	269
Chinese	246
Sushi	182
Salad	176
Mexican	143
Bakery	123
Grocery Store	112
BBQ	104
Pizza	88
Juice Bar	81
Indian	76
Donut	67
Seafood	63
Fast Food	58
Mediterranean	57
Vegetarian	55
Middle Eastern	52
Deli	50
Coffee	49
Gluten-Free	49
Ice Cream	48
Drug Store	48
Breakfast	42
Shop	39
South American	33
Thai	30
Steak	28
Southern	23
French	19
Electronics Store	17
Vegan	17
Food Truck	17
Korean	13
Spanish	12
Asian	11
Eastern European	10
Promo	9
Russian	8
Convenience Store	8
Department Store	8
Vietnamese	6
Caribbean	6
Specialty Store	5
Office Supplies Store	4
German	4
Kids & Baby	3
Beauty Supply	2
Art Store	1
Book Store	1
African	1
Restaurant	1

```
Clothing                    1
Name: place_category, dtype: int64
```

In [ ]:

In [ ]:

In [ ]:

In [53]:
```
#what timeframe of data?

print(df.when_the_Jumpman_arrived_at_dropoff.min())
print(df.when_the_Jumpman_arrived_at_dropoff.max())
```

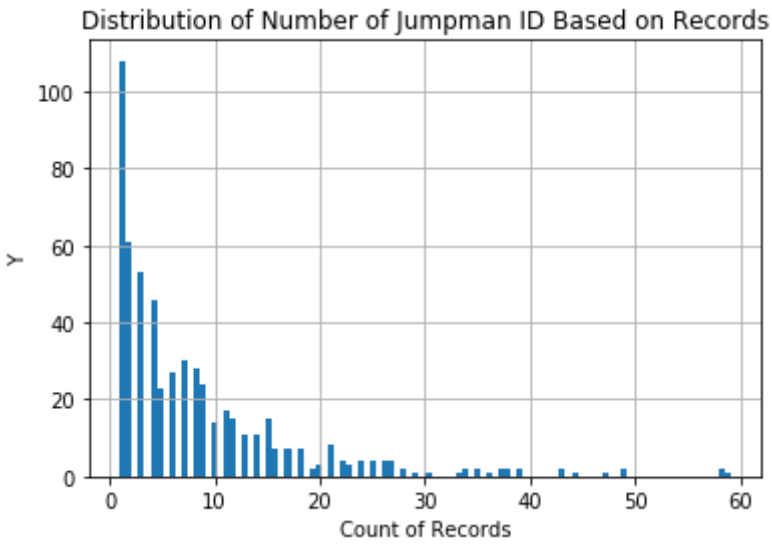
```
2014-10-01 01:04:14.355157
2014-10-30 23:29:44.866438
```

In [54]:
```
#what is the most popular method of delivery?
df['vehicle_type'].value_counts()
```

Out[54]:
```
bicycle       3371
car            967
walker         209
van             60
scooter         58
truck           34
motorcycle      18
Name: vehicle_type, dtype: int64
```

In [55]:
```
#distribution of jumpman delivery people
df.jumpman_id.value_counts().hist(bins=100)
plt.title('Distribution of Number of Jumpman ID Based on Records')
plt.xlabel('Count of Records')
plt.ylabel('Y')
```

Out[55]:
```
Text(0, 0.5, 'Y')
```



In [56]:
```
df.jumpman_id.value_counts().sort_values(ascending=False)
```

Out[56]:
```
99219     59
142394    58
104533    58
30743     49
3296      49
          ..
177125     1
64628      1
94614      1
137979     1
159647     1
Name: jumpman_id, Length: 565, dtype: int64
```

In [57]:
```
#top 3 jumpmen
df.jumpman_id.value_counts().sort_values(ascending=False)[:3]
```

Out[57]:
```
99219     59
142394    58
104533    58
Name: jumpman_id, dtype: int64
```

In [58]:
```
#"One Delivery" jumpmen
```

In [59]:
```
j_vc = df.jumpman_id.value_counts()

#custom dataframe which has the 5 most active Jumpmen, and a random samp
le of the 10 least active
ab = pd.DataFrame({'Jumpman ID':j_vc[j_vc==1].sample(n=10, random_state
= 23).keys(),
                   'Count':j_vc[j_vc==1].sample(n=10, random_state = 23).valu
es})

bc = pd.DataFrame({'Jumpman ID':df.jumpman_id.value_counts().sort_values
(ascending=False)[:5].keys(),
                   'Count':df.jumpman_id.value_counts().sort_values(ascending
=False)[:5].values})


pd.concat([ab,bc]).sort_values(by='Count', ascending=False)
```

Out[59]:

	Jumpman ID	Count
0	99219	59
1	142394	58
2	104533	58
3	30743	49
4	3296	49
0	46336	1
1	128471	1
2	88874	1
3	74914	1
4	170880	1
5	170602	1
6	54356	1
7	167003	1
8	166482	1
9	30365	1

In [60]:
```
c_vc = df.customer_id.value_counts()

#custom dataframe which has the 5 most active Jumpmen, and a random samp
le of the 10 least active
ab = pd.DataFrame({'Customer ID':c_vc[c_vc==1].sample(n=10, random_state
= 23).keys(),
                   'Count':c_vc[c_vc==1].sample(n=10, random_state = 23).valu
es})

bc = pd.DataFrame({'Customer ID':df.customer_id.value_counts().sort_valu
es(ascending=False)[:5].keys(),
                   'Count':df.customer_id.value_counts().sort_values(ascendin
g=False)[:5].values})

pd.concat([ab,bc]).sort_values(by='Count', ascending=False)
```

Out[60]:

	Customer ID	Count
0	369272	23
1	52832	17
2	47440	12
3	125123	12
4	91817	11
0	5056	1
1	157160	1
2	379236	1
3	194778	1
4	153994	1
5	114410	1
6	337814	1
7	121300	1
8	385168	1
9	351372	1

In [61]:
```
top_3 = ['99219','104533','142394']
df[df.jumpman_id.isin(top_3)]
```

Out[61]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_nar
1	1377056	64452	104533	bicycle	Prince Street Pizza	Pizza	Neapolit Rice Ba
21	1356218	128224	99219	bicycle	Osteria Morini	Italian	Gargan
24	1488027	396432	99219	bicycle	Juice Press	Juice Bar	Rehab Sh
32	1467996	301380	142394	bicycle	ilili Restaurant	Middle Eastern	Homm
53	1407558	41130	99219	bicycle	Village Yogurt	Not Disclosed	N Disclos
...	...	...	...	...	...	...	
5453	1412950	374826	104533	bicycle	Waverly Diner	American	Linzer T
5473	1272439	354016	99219	bicycle	Waverly Diner	American	Cappucci
5544	1489657	390459	104533	bicycle	Shake Shack	Burger	Bottl Wa
5548	1341499	67370	104533	bicycle	Duane Reade	Drug Store	N Disclos
5802	1399659	373485	99219	bicycle	sweetgreen	Salad	Guacam Gree

175 rows × 24 columns

In [ ]:
```
```

Customer Loyalty

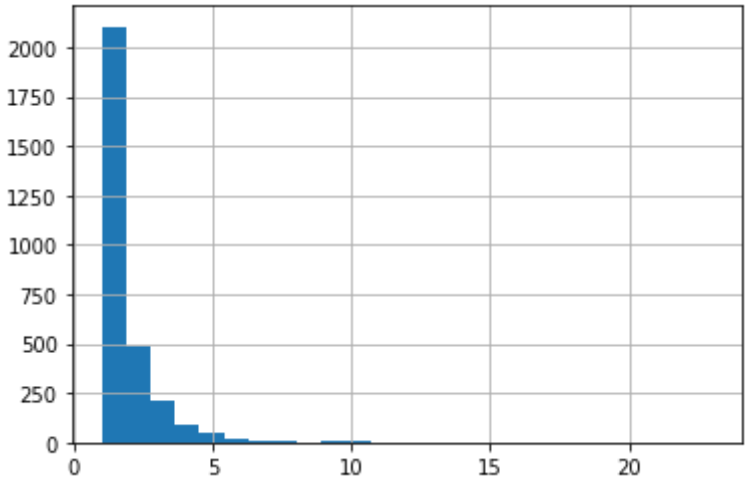
In [62]:
```
customer_vc = df['customer_id'].value_counts()

print(len(customer_vc[customer_vc==1]))
print(len(customer_vc[customer_vc>1]))
print(customer_vc.mean())
```

```
2104
880
1.5807640750670242
```

In [63]:
```
customer_vc.hist(bins=25)
```

Out[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faedf579f10>
```



In [64]:
```
customer_vc.max()
```

Out[64]:
```
23
```

In [65]:
```
customer_vc
```

Out[65]:
```
369272    23
52832     17
47440     12
125123    12
91817     11
          ..
35631      1
119596     1
58142      1
181018     1
376836     1
Name: customer_id, Length: 2984, dtype: int64
```

In [66]:
```
customer_vc.to_csv('../Additional_Datasets/customer_loyalty.csv')
```

In [67]:
```
df_vc = df['customer_id'].value_counts()

print('order once:', len(df_vc[df_vc==1]))
print('order twice:', len(df_vc[df_vc==2]))
print('order three times:', len(df_vc[df_vc>3]))
```

```
order once: 2104
order twice: 482
order three times: 190
```

In [68]:
```
df['when_the_delivery_started'].dt.strftime('%m/%d/%Y')
```

Out[68]:
```
1        10/16/2014
2        10/28/2014
3        10/30/2014
4        10/10/2014
5        10/22/2014
            ...
5976     10/05/2014
5977     10/05/2014
5979     10/12/2014
5980     10/01/2014
5981     10/27/2014
Name: when_the_delivery_started, Length: 4717, dtype: object
```

What is the most popular hour for a delivery to start?

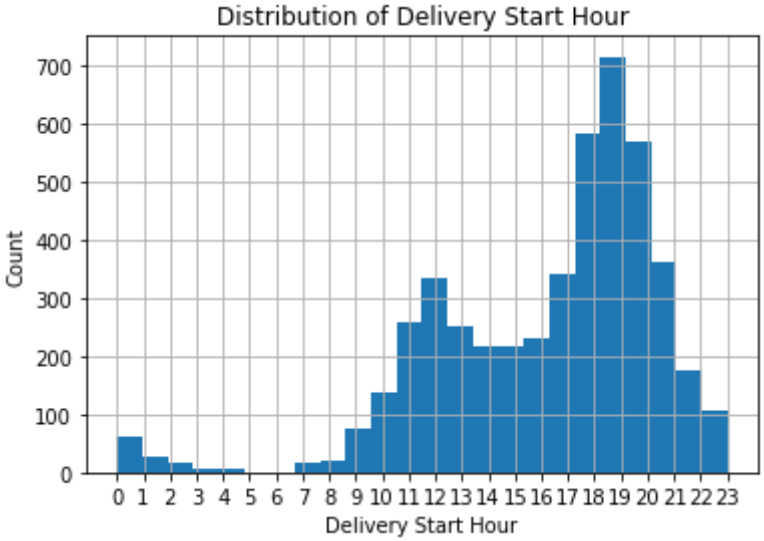
In [69]:
```
delivery_start_hour = []
my_lst = df.index.to_list()

for i in my_lst:
    delivery_start_hour.append(df['when_the_delivery_started'].loc[i].ho
ur)

df['delivery_start_hour'] = delivery_start_hour
```

In [70]:
```
df['delivery_start_hour'].hist(bins = 24, align='mid')
plt.title('Distribution of Delivery Start Hour')
plt.xlabel('Delivery Start Hour')
plt.xticks(np.arange(0, 24, 1))
plt.ylabel('Count')
```

Out[70]:
```
Text(0, 0.5, 'Count')
```



In [71]:
```
df['delivery_start_hour'].value_counts()
```

Out[71]:
```
19    714
18    583
20    568
21    363
17    340
12    335
11    259
13    250
16    229
14    216
15    215
22    175
10    136
23    107
9      75
0      60
1      28
8      21
7      15
2      15
3       7
4       5
6       1
Name: delivery_start_hour, dtype: int64
```

In [72]:
```
df['delivery_start_hour'].value_counts()[:3].sum()
```

Out[72]:
```
1865
```

In [73]:
```
df['delivery_start_hour'].value_counts().sum()
```

Out[73]:
```
4717
```

In [74]:
```
1865/4717
```

Out[74]:
```
0.39537841848632604
```

In [ ]:
```

```

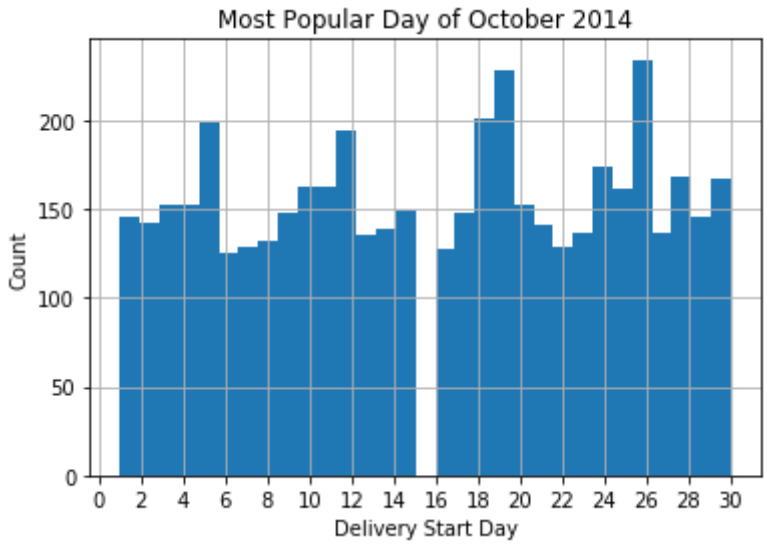
In [75]:
```
delivery_start_day = []
my_lst = df.index.to_list()

for i in my_lst:
    delivery_start_day.append(df['when_the_delivery_started'].loc[i].day
)

df['delivery_start_day'] = delivery_start_day
```

In [76]:
```
df['delivery_start_day'].hist(bins = 31)
plt.title('Most Popular Day of October 2014')
plt.xlabel('Delivery Start Day')
plt.xticks(np.arange(0, 31, 2))
plt.ylabel('Count')
```

Out[76]: Text(0, 0.5, 'Count')



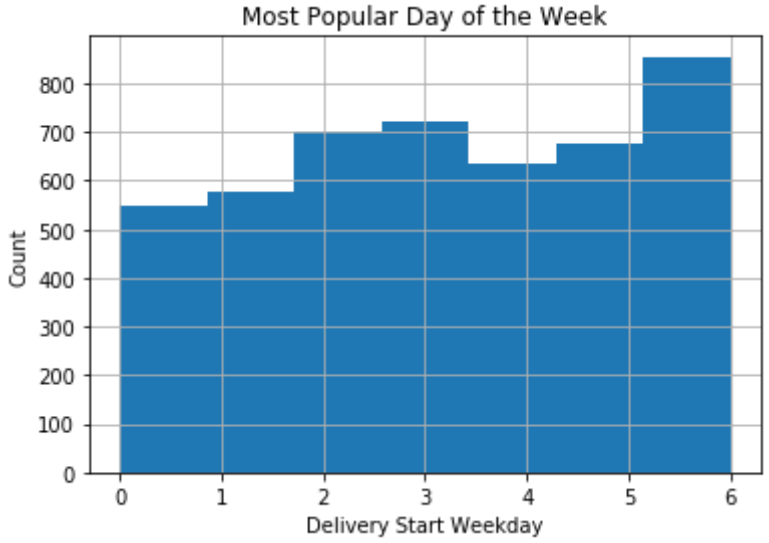
In [77]:
```
delivery_start_weekday = []
my_lst = df.index.to_list()

for i in my_lst:
    delivery_start_weekday.append(df['when_the_delivery_started'].loc[i]
.weekday())

df['delivery_start_weekday'] = delivery_start_weekday
```

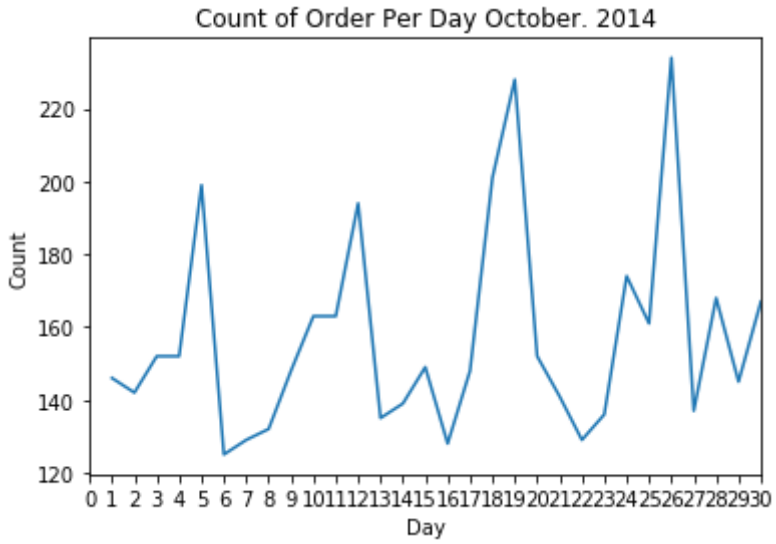
In [78]:
```
df['delivery_start_weekday'].hist(bins = 7, align='mid')
plt.title('Most Popular Day of the Week')
plt.xlabel('Delivery Start Weekday')
plt.xticks(np.arange(0, 7, 1))
plt.ylabel('Count')
```

Out[78]: Text(0, 0.5, 'Count')



In [79]:
```
df['delivery_start_day'].value_counts().sort_index().plot()
plt.title('Count of Order Per Day October. 2014')
plt.xlabel('Day')
plt.xticks(np.arange(0, 31, 1))
plt.ylabel('Count')
```

Out[79]: Text(0, 0.5, 'Count')



In [80]:
```
print(df['delivery_start_day'].value_counts().sort_index()[3:10].sum())
print(df['delivery_start_day'].value_counts().sort_index()[11:17].sum())
print(df['delivery_start_day'].value_counts().sort_index()[18:24].sum())
print(df['delivery_start_day'].value_counts().sort_index()[25:31].sum())
```

```
1048
893
960
851
```

In [81]:
```
df['delivery_start_day'].value_counts().sort_index()
```

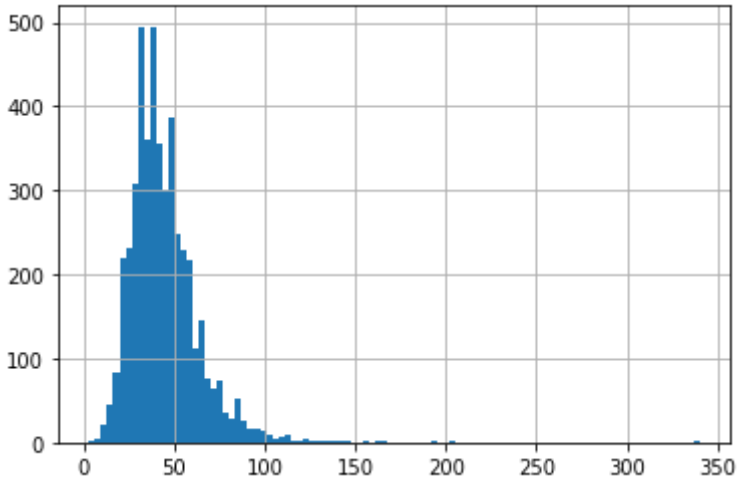
Out[81]:
```
1     146
2     142
3     152
4     152
5     199
6     125
7     129
8     132
9     148
10    163
11    163
12    194
13    135
14    139
15    149
16    128
17    148
18    201
19    228
20    152
21    141
22    129
23    136
24    174
25    161
26    234
27    137
28    168
29    145
30    167
Name: delivery_start_day, dtype: int64
```

Total Delivery Time

Total Time to Delivery calculated as the time the Jumpman arrived at dropoff less the time the delivery started. I would like to see a distribution of the data to see its characteristics

In [82]:
```
delivery_time = df['when_the_Jumpman_arrived_at_dropoff'] - df['when_the
_delivery_started']
delivery_time.astype('timedelta64[m]').hist(bins=100)
```

Out[82]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7faee0029f50>
```



In [83]:
```
round(delivery_time.astype('timedelta64[m]').describe(),1)
```

Out[83]:
```
count    4717.0
mean       44.4
std        18.9
min         3.0
25%        32.0
50%        41.0
75%        53.0
max       340.0
dtype: float64
```

In [84]:
```
df['customer_id'].value_counts().describe()
```

Out[84]:
```
count    2984.000000
mean        1.580764
std         1.294219
min         1.000000
25%         1.000000
50%         1.000000
75%         2.000000
max        23.000000
Name: customer_id, dtype: float64
```

In [ ]:

I want to calculate the number of orders per week.

In [85]:
```
#Count of order per week
week_1_count = df['delivery_start_day'].value_counts().sort_index()[:7].
sum()
week_2_count = df['delivery_start_day'].value_counts().sort_index()[7:14
].sum()
week_3_count = df['delivery_start_day'].value_counts().sort_index()[14:2
1].sum()
week_4_count = df['delivery_start_day'].value_counts().sort_index()[21:2
8].sum()
week_5_count = df['delivery_start_day'].value_counts().sort_index()[28:3
1].sum()

print(week_1_count)
print(week_2_count)
print(week_3_count)
print(week_4_count)
print(week_5_count)
```

```
1045
1074
1147
1139
312
```

Most popular dropoff zip codes

In [86]:
```
df.iloc[0]
```

Out[86]:
```
delivery_id                                                1377056
customer_id                                                  64452
jumpman_id                                                  104533
vehicle_type                                               bicycle
pickup_place                                  Prince Street Pizza
place_category                                              Pizza
item_name                                  Neapolitan Rice Balls
item_quantity                                                  3
item_category_name                                     Munchables
how_long_it_took_to_order                                      25
pickup_lat                                                40.7231
pickup_lon                                               -73.9946
dropoff_lat                                               40.7197
dropoff_lon                                              -73.9919
when_the_delivery_started              2014-10-16 21:58:58.654910
when_the_Jumpman_arrived_at_pickup     2014-10-16 22:26:02.120931
when_the_Jumpman_left_pickup           2014-10-16 22:48:23.091253
when_the_Jumpman_arrived_at_dropoff    2014-10-16 22:59:22.948873
haversine_distance_mi                                        0.27
travel_time_seconds                                           659
travel_time_minutes                                       10.9833
travel_time_hours                                        0.183056
travel_rate_mph                                           1.47496
Jumpmen_wait_time                                              22
delivery_start_hour                                            21
delivery_start_day                                             16
delivery_start_weekday                                          3
Name: 1, dtype: object
```

Who are the jumpmen?

In [87]:
```
df['jumpman_id'].value_counts()
```

Out[87]:
```
99219     59
142394    58
104533    58
30743     49
3296      49
          ..
122573     1
155385     1
14610      1
125160     1
159647     1
Name: jumpman_id, Length: 565, dtype: int64
```

In [88]:
```
value_counts = df['jumpman_id'].value_counts()
value_counts[value_counts==1]
```

Out[88]:
```
179183    1
170959    1
153533    1
167758    1
161713    1
         ..
122573    1
155385    1
14610     1
125160    1
159647    1
Name: jumpman_id, Length: 108, dtype: int64
```

In [89]:
```
value_counts[value_counts>5]
```

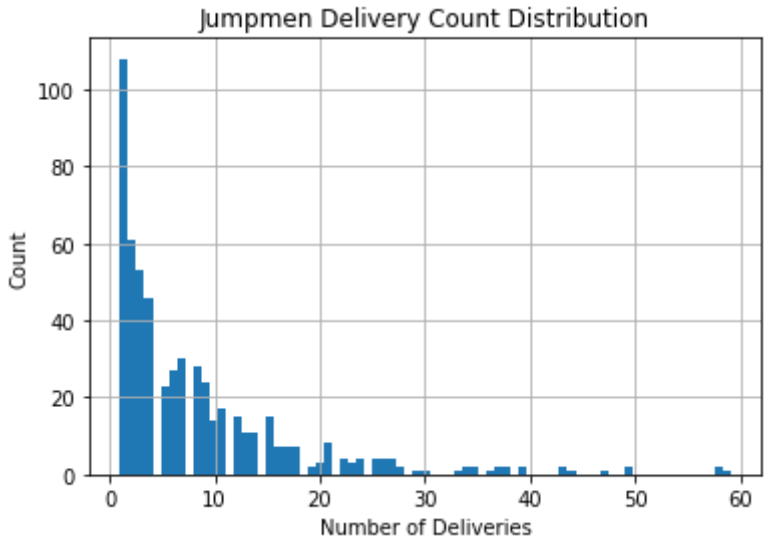
Out[89]:
```
99219     59
142394    58
104533    58
30743     49
3296      49
          ..
143807     6
110192     6
20124      6
156008     6
175555     6
Name: jumpman_id, Length: 274, dtype: int64
```

In [90]:
```
df['jumpman_id'].value_counts().sum()
```

Out[90]:
```
4717
```

In [91]:
```
df['jumpman_id'].value_counts().hist(bins=75)
plt.title('Jumpmen Delivery Count Distribution')
plt.xlabel('Number of Deliveries')
plt.ylabel('Count')
```

Out[91]:
```
Text(0, 0.5, 'Count')
```



In [92]:
```
df['jumpman_id'].value_counts().mean()
```

Out[92]:
```
8.34867256637168
```

In [93]:
```
df['jumpman_id'].value_counts().max()
```

Out[93]:
```
59
```

In [94]:
```
df['jumpman_id'].nunique()
```

Out[94]:
```
565
```

In [95]:
```
unique_jumpmen = df['jumpman_id'].unique()

df[df.jumpman_id.isin(unique_jumpmen)]['vehicle_type'].value_counts()
```

Out[95]:
```
bicycle       3371
car            967
walker         209
van             60
scooter         58
truck           34
motorcycle      18
Name: vehicle_type, dtype: int64
```

Zip Code

I want to add several demographic statistics to the dataset. To do this, I will need to use the uszipcode package. I can find the correct zipcode using latitude and longitude coordinates. I will then be able to add estimated population and median household income population.

In [96]:
```
from uszipcode import SearchEngine
```

In [97]:
```
search = SearchEngine(simple_zipcode=True) # set simple_zipcode=False to use rich info database
```

I want to add the pickup and dropoff location zipcodes to the dataset.

In [98]:
```
f = lambda x, y : search.by_coordinates(lat=x, lng=y, returns=1)[0].to_dict()['zipcode']
df['pickup_zipcode'] = df[['pickup_lat','pickup_lon']].apply(lambda x : f(*x), axis=1)
df['dropoff_zipcode'] = df[['dropoff_lat','dropoff_lon']].apply(lambda x : f(*x), axis=1)
```

In [99]:
```
f = lambda x, y : search.by_coordinates(lat=x, lng=y, returns=1)[0].to_dict()['population']
df['pickup_population'] = df[['pickup_lat','pickup_lon']].apply(lambda x : f(*x), axis=1)
df['dropoff_population'] = df[['dropoff_lat','dropoff_lon']].apply(lambda x : f(*x), axis=1)
```

In [100]:
```
f = lambda x, y : search.by_coordinates(lat=x, lng=y, returns=1)[0].to_dict()['median_household_income']
df['pickup_med_hh_income'] = df[['pickup_lat','pickup_lon']].apply(lambda x : f(*x), axis=1)
df['dropoff_med_hh_income'] = df[['dropoff_lat','dropoff_lon']].apply(lambda x : f(*x), axis=1)
```

In [101]:
```
df.sort_values(by='dropoff_med_hh_income', ascending =False)
```

Out[101]:

	delivery_id	customer_id	jumpman_id	vehicle_type	pickup_place	place_category	item_nam
5197	1470245	70001	39733	bicycle	Han Dynasty	Chinese	Vegetab Sprir Rol
1046	1431400	354948	101359	bicycle	Westville Hudson	American	Gree Sala
1303	1410254	99288	102353	car	Dig Inn Seasonal Market	American	N Disclose
4974	1392532	59217	120531	car	Serafina Meatpacking	Italian	N Disclose
4958	1318566	47440	138560	bicycle	Blue Ribbon Fried Chicken	American	Tend
...	...	...	...	...	...	...	
5873	1393083	58675	140096	bicycle	Shake Shack	Burger	Frie
5910	1314103	87747	153766	bicycle	Blue Ribbon Fried Chicken	American	Drumstic
5913	1368358	95049	61900	bicycle	Nobu Next Door	Sushi	Roc Shrim Tempu
5945	1348292	123479	155879	car	Boqueria	Spanish	Age Mahc (Menorc
5980	1274438	355090	153113	bicycle	Shake Shack	Burger	Frie

4717 rows × 33 columns

I want to determine what percent of each zipcode we have captured. This market penetration level, along with median household income, will give us some ideas for places to target. I'll start by collecting the 10 largest zipcodes by population. I will then collect the 10 zipcodes with teh greatest median household income.

In [102]:
```
df[['dropoff_zipcode','dropoff_population','dropoff_med_hh_income']].gro
upby('dropoff_zipcode').mean().sort_values(by='dropoff_population', asce
nding=False)[:10]
```

Out[102]:

dropoff_zipcode	dropoff_population	dropoff_med_hh_income
11226	101572	40734.0
10025	94600	68516.0
11211	90117	46848.0
11206	81677	28559.0
10002	81410	33218.0
11221	78895	39178.0
10029	76003	31888.0
11215	63488	95654.0
10009	61347	59929.0
10023	60998	103534.0

In [ ]:
```

```

In [103]:
```
#manually cycle through the list of zips
lst_a = df[['dropoff_zipcode','dropoff_population','dropoff_med_hh_incom
e']].groupby('dropoff_zipcode').mean().sort_values(by='dropoff_populatio
n', ascending=False)[:10].index

for i in range(10):
    print(i,
          lst_a[i],
          df[df['dropoff_zipcode']==lst_a[i]]['dropoff_population'].uniq
ue(),
          len(df[df['dropoff_zipcode']==lst_a[i]]['customer_id'].unique
()))
```

```
0 11226 [101572] 3
1 10025 [94600] 26
2 11211 [90117] 31
3 11206 [81677] 2
4 10002 [81410] 135
5 11221 [78895] 2
6 10029 [76003] 39
7 11215 [63488] 13
8 10009 [61347] 121
9 10023 [60998] 73
```

Now do the same process for the 10 zipcodes with the greatest median household income

In [104]:
```
df[['dropoff_zipcode','dropoff_population','dropoff_med_hh_income']].groupby('dropoff_zipcode').mean().sort_values(by='dropoff_med_hh_income', ascending=False)[:10]
```

Out[104]:

dropoff_zipcode	dropoff_population	dropoff_med_hh_income
10282	4783	230952.0
10007	6988	216037.0
10069	5199	170630.0
10162	1685	168667.0
10280	7853	129574.0
11109	3523	125871.0
10005	7135	124670.0
10006	3011	119274.0
10065	32270	115519.0
10024	59283	109956.0

In [105]:
```
lst_b = df[['dropoff_zipcode','dropoff_population','dropoff_med_hh_income']].groupby('dropoff_zipcode').mean().sort_values(by='dropoff_med_hh_income', ascending=False)[:10].index

for i in range(10):
    print(i,
          lst_b[i],
          df[df['dropoff_zipcode']==lst_b[i]]['dropoff_med_hh_income'].unique(),
          df[df['dropoff_zipcode']==lst_b[i]]['dropoff_population'].unique(),
          len(df[df['dropoff_zipcode']==lst_b[i]]['customer_id'].unique()))
```

```
0 10282 [230952.] [4783] 27
1 10007 [216037.] [6988] 38
2 10069 [170630.] [5199] 45
3 10162 [168667.] [1685] 64
4 10280 [129574.] [7853] 18
5 11109 [125871.] [3523] 2
6 10005 [124670.] [7135] 18
7 10006 [119274.] [3011] 31
8 10065 [115519.] [32270] 99
9 10024 [109956.] [59283] 1
```

In [106]:
```
lst_b
```

Out[106]:
```
Index(['10282', '10007', '10069', '10162', '10280', '11109', '10005',
'10006',
       '10065', '10024'],
      dtype='object', name='dropoff_zipcode')
```

In [107]:
```
#only tells us how many orders went to each zip, does not reflect unique
ness in the customer id column
df.groupby('dropoff_zipcode').agg(['count'])['delivery_id'].sort_values(
by='count', ascending=False)
```

Out[107]:

dropoff_zipcode	count
10003	392
10012	388
10011	352
10001	276
10002	215
...	...
11106	1
11104	1
10039	1
11101	1
10024	1

69 rows × 1 columns

How many total miles traveled by Jumpmen

In [108]:
```
df['haversine_distance_mi'].sum()
```

Out[108]:
```
5390.469999999999
```

In [109]:
```
df['jumpman_id'].value_counts().describe()
```

Out[109]:
```
count    565.000000
mean       8.348673
std        9.359100
min        1.000000
25%        2.000000
50%        5.000000
75%       11.000000
max       59.000000
Name: jumpman_id, dtype: float64
```

In [ ]:

In [ ]:

In [110]:
```
#average median household income
(81671 + 33218 + 92540 + 104238 + 86594)/5
```

Out[110]: 79652.2

In [111]:
```
#average population
(21102 + 81410 + 56024 + 50984 + 24090)/5
```

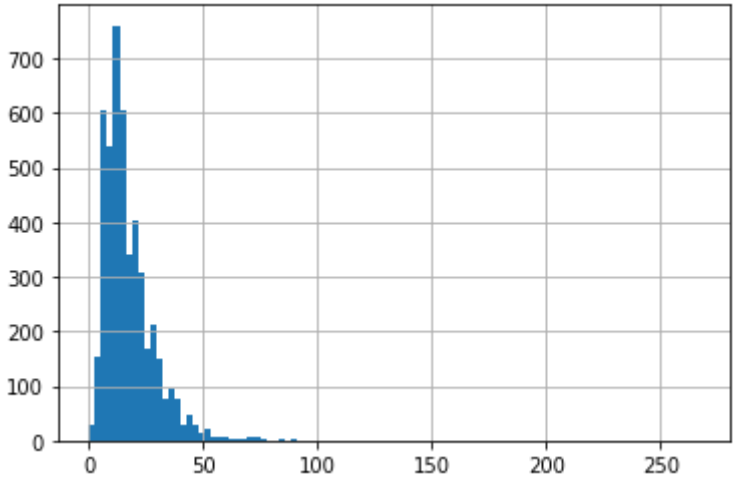
Out[111]: 46722.0

In [ ]:

In [ ]:

In [112]:
```
df['Jumpmen_wait_time'].hist(bins=100)
```

Out[112]: <matplotlib.axes._subplots.AxesSubplot at 0x7faedce7cad0>



In [113]:
```
df['Jumpmen_wait_time'].mean()
```

Out[113]: 17.725885096459614

In [114]:
```
round(df['Jumpmen_wait_time'].describe(),1)
```

Out[114]:
```
count    4717.0
mean       17.7
std        11.7
min         0.0
25%        10.0
50%        15.0
75%        22.0
max       267.0
Name: Jumpmen_wait_time, dtype: float64
```

In [ ]:

In [ ]:

Scratch/Archive

rate = df.iloc[0]['avg_vehicle_rate_kms'] distance = df.iloc[0]['haversine_distance_km'] time = distance/rate time = time.astype('timedelta64[s]') time

print(df.iloc[0]['when_the_Jumpman_arrived_at_dropoff']) print(df.iloc[0]
['when_the_Jumpman_arrived_at_dropoff'] - time)

df['time_pickup_to_dropoff'] = df['haversine_distance_km']/df['avg_vehicle_rate_kms']
df['time_pickup_to_dropoff'] = df['time_pickup_to_dropoff'].astype('timedelta64[s]')

for i in df[df['when_the_Jumpman_left_pickup'].isna()].index:

```
#rate = df.iloc[i]['avg_vehicle_rate_kms']
#distance = df.iloc[i]['haversine_distance_km']
#time = (distance/rate).astype('timedelta64[s]')
#df.iloc[i]['when_the_Jumpman_left_pickup'] = (df.iloc[i]['when_the_Jumpman_
arrived_at_dropoff'] - (df.iloc[0]['haversine_distance_km']/df.iloc[i]['avg_
vehicle_rate_kms']).astype('timedelta64[s]'))
print(df.iloc[i]['when_the_Jumpman_arrived_at_dropoff'] - df.iloc[i])
```

null_pickup = df[df['when_the_Jumpman_left_pickup'].isnull()].index.to_list()

for i in null_pickup: df.loc[i]['when_the_Jumpman_left_pickup'] = df.loc[i]
['when_the_Jumpman_arrived_at_dropoff'] - df.loc[i]['time_pickup_to_dropoff']

df.iloc[0]['how_long_it_took_to_order'].minutes

time_cols = ['how_long_it_took_to_order'] for i in time_cols: df[i] = pd.to_datetime(df[i])

def minutes(data_input): return data_input.minute*60.0 + data_input.second

minute = df.how_long_it_took_to_order.apply(minutes) df['min_to_order'] = round(minute/60,2)

df['how_long_it_took_to_order'] = pd.to_datetime(df.how_long_it_took_to_order, format = '%H:%M:%S.%f')

df['min_to_order'].hist(bins=50) df['min_to_order'].describe() sns.boxplot(y="min_to_order", data=df, orient='h') sns.violinplot(x="vehicle_type", y="min_to_order", data=df, split=False, inner="quart", linewidth=1.3)

def get_population (zip_code): '''Get demographic information by zipcode''' zipcode = search.by_zipcode(zip_code) some_zip = zipcode.to_dict() population = some_zip['population'] return population

def get_med_income (zip_code): '''Get demographic information by zipcode''' income = search.by_zipcode(zip_code) some_zip = zipcode.to_dict() income = some_zip['median_household_income'] return income

get_population(10022)

sample zipcode

search = SearchEngine(simple_zipcode=True) # set simple_zipcode=False to use rich info database zipcode = search.by_zipcode("10022") zip_10022 = zipcode.to_dict() zip_10022 zip_10022['median_household_income']

Define a funciton to calculate Haversine distance in km. 'r', or radius, corresponds to the spherical radius of the earth. Change this value depending on what units you are looking for.

def haversine_distance(lat1, lon1, lat2, lon2): r = 6371 phi1 = np.radians(lat1) phi2 = np.radians(lat2) delta_phi = np.radians(lat2 - lat1) delta_lambda = np.radians(lon2 - lon1) a = np.sin(delta_phi / 2)**2 + np.cos(phi1) *np.cos(phi2)* **np.sin(delta_lambda / 2)**2 res = r *(2* np.arctan2(np.sqrt(a), np.sqrt(1 - a))) return np.round(res, 2)

df['haversine_distance_km'] = haversine_distance(df['pickup_lat'], df['pickup_lon'], df['dropoff_lat'], df['dropoff_lon'])

separate dataframe

df_sub = df.copy() df_sub = df_sub.dropna(subset= ['when_the_Jumpman_arrived_at_dropoff','when_the_Jumpman_left_pickup'])

select_cols = ['delivery_id','vehicle_type','when_the_Jumpman_left_pickup', 'when_the_Jumpman_arrived_at_dropoff','haversine_distance_mi'] df_sub = df_sub[select_cols]

calculate travel time in seconds, minutes, and hours

travel_time_seconds = (df_sub['when_the_Jumpman_arrived_at_dropoff'] - df_sub['when_the_Jumpman_left_pickup']).astype('timedelta64[s]') df_sub['travel_time_seconds'] = travel_time_seconds df_sub['travel_time_minutes'] = travel_time_seconds / 60 df_sub['travel_time_hour'] = travel_time_seconds / 60 / 60

calculate travel rate in mph

travel_rate_mph = df_sub['haversine_distance_mi'] / df_sub['travel_time_hour'] df_sub['travel_rate_mph'] = travel_rate_mph

create avg rate dictionary that I can call later on

avg_rate_dic = {} unique_vehicles = df_sub['vehicle_type'].unique()

for i in unique_vehicles: avg_rate_dic[i] = df_sub[df_sub.vehicle_type==i]['travel_rate_mph'].mean()

add average vehicle rate to the dataframe

avg_vehicle_rate = [] for value in df['vehicle_type']: if value == 'van': avg_vehicle_rate.append(avg_rate_dic['van']) elif value == 'bicycle': avg_vehicle_rate.append(avg_rate_dic['bicycle']) elif value == 'car': avg_vehicle_rate.append(avg_rate_dic['car']) elif value == 'walker': avg_vehicle_rate.append(avg_rate_dic['walker']) elif value == 'truck': avg_vehicle_rate.append(avg_rate_dic['truck']) elif value == 'scooter': avg_vehicle_rate.append(avg_rate_dic['scooter']) elif value == 'motorcycle': avg_vehicle_rate.append(avg_rate_dic['motorcycle']) else: avg_vehicle_rate.append('N/A') df['avg_vehicle_rate_mph'] = avg_vehicle_rate

In [ ]:

In [ ]:

In [ ]: