Visualizations With Python (TLC Project)

December 5, 2025

[1]:
```
#The purpose of this project is to conduct exploratory data analysis on a␣
↪provided data set.
#The goal is to clean data set and create a visualization.


#Part 1: Imports, links, and loading
#Part 2: Data Exploration and Data cleaning
#Part 3: Building visualizations
#Part 4: Evaluate and share results
```

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt
import seaborn as sns
```

[3]:
```
df=pd.read_csv('data/2017_Yellow_Taxi_Trip_Data.csv')
```

[4]:
```
df.head(10)
```

[4]:
```
   Unnamed: 0  VendorID    tpep_pickup_datetime   tpep_dropoff_datetime  \
0    24870114         2   03/25/2017 8:55:43 AM   03/25/2017 9:09:47 AM
1    35634249         1   04/11/2017 2:53:28 PM   04/11/2017 3:19:58 PM
2   106203690         1   12/15/2017 7:26:56 AM   12/15/2017 7:34:08 AM
3    38942136         2   05/07/2017 1:17:59 PM   05/07/2017 1:48:14 PM
4    30841670         2  04/15/2017 11:32:20 PM  04/15/2017 11:49:03 PM
5    23345809         2   03/25/2017 8:34:11 PM   03/25/2017 8:42:11 PM
6    37660487         2   05/03/2017 7:04:09 PM   05/03/2017 8:03:47 PM
7    69059411         2   08/15/2017 5:41:06 PM   08/15/2017 6:03:05 PM
8     8433159         2   02/04/2017 4:17:07 PM   02/04/2017 4:29:14 PM
9    95294817         1   11/10/2017 3:20:29 PM   11/10/2017 3:40:55 PM

   passenger_count  trip_distance  RatecodeID store_and_fwd_flag  \
0                6           3.34           1                  N
1                1           1.80           1                  N
2                1           1.00           1                  N
3                1           3.70           1                  N
```

```
4                1            4.37             1                 N
5                6            2.30             1                 N
6                1           12.83             1                 N
7                1            2.98             1                 N
8                1            1.20             1                 N
9                1            1.60             1                 N

   PULocationID  DOLocationID  payment_type  fare_amount  extra  mta_tax  \
0           100           231             1         13.0    0.0      0.5
1           186            43             1         16.0    0.0      0.5
2           262           236             1          6.5    0.0      0.5
3           188            97             1         20.5    0.0      0.5
4             4           112             2         16.5    0.5      0.5
5           161           236             1          9.0    0.5      0.5
6            79           241             1         47.5    1.0      0.5
7           237           114             1         16.0    1.0      0.5
8           234           249             2          9.0    0.0      0.5
9           239           237             1         13.0    0.0      0.5

   tip_amount  tolls_amount  improvement_surcharge  total_amount
0        2.76           0.0                    0.3         16.56
1        4.00           0.0                    0.3         20.80
2        1.45           0.0                    0.3          8.75
3        6.39           0.0                    0.3         27.69
4        0.00           0.0                    0.3         17.80
5        2.06           0.0                    0.3         12.36
6        9.86           0.0                    0.3         59.16
7        1.78           0.0                    0.3         19.58
8        0.00           0.0                    0.3          9.80
9        2.75           0.0                    0.3         16.55
```

[5]:
```
df.size
```

[5]: 408582

[6]:
```
df.describe()
```

[6]:
```
         Unnamed: 0      VendorID  passenger_count  trip_distance  \
count  2.269900e+04  22699.000000     22699.000000   22699.000000
mean   5.675849e+07      1.556236         1.642319       2.913313
std    3.274493e+07      0.496838         1.285231       3.653171
min    1.212700e+04      1.000000         0.000000       0.000000
25%    2.852056e+07      1.000000         1.000000       0.990000
50%    5.673150e+07      2.000000         1.000000       1.610000
75%    8.537452e+07      2.000000         2.000000       3.060000
max    1.134863e+08      2.000000         6.000000      33.960000
```

```
       RatecodeID  PULocationID  DOLocationID  payment_type   fare_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000  22699.000000
mean       1.043394    162.412353    161.527997      1.336887     13.026629
std        0.708391     66.633373     70.139691      0.496211     13.243791
min        1.000000      1.000000      1.000000      1.000000   -120.000000
25%        1.000000    114.000000    112.000000      1.000000      6.500000
50%        1.000000    162.000000    162.000000      1.000000      9.500000
75%        1.000000    233.000000    233.000000      2.000000     14.500000
max       99.000000    265.000000    265.000000      4.000000    999.990000

              extra       mta_tax    tip_amount  tolls_amount  \
count  22699.000000  22699.000000  22699.000000  22699.000000
mean       0.333275      0.497445      1.835781      0.312542
std        0.463097      0.039465      2.800626      1.399212
min       -1.000000     -0.500000      0.000000      0.000000
25%        0.000000      0.500000      0.000000      0.000000
50%        0.000000      0.500000      1.350000      0.000000
75%        0.500000      0.500000      2.450000      0.000000
max        4.500000      0.500000    200.000000     19.100000

       improvement_surcharge  total_amount
count           22699.000000  22699.000000
mean                0.299551     16.310502
std                 0.015673     16.097295
min                -0.300000   -120.300000
25%                 0.300000      8.750000
50%                 0.300000     11.800000
75%                 0.300000     17.800000
max                 0.300000   1200.290000
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 22699 entries, 0 to 22698
Data columns (total 18 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Unnamed: 0             22699 non-null  int64
 1   VendorID               22699 non-null  int64
 2   tpep_pickup_datetime   22699 non-null  object
 3   tpep_dropoff_datetime  22699 non-null  object
 4   passenger_count        22699 non-null  int64
 5   trip_distance          22699 non-null  float64
 6   RatecodeID             22699 non-null  int64
 7   store_and_fwd_flag     22699 non-null  object
 8   PULocationID           22699 non-null  int64
 9   DOLocationID           22699 non-null  int64
```

```
 10  payment_type           22699 non-null  int64
 11  fare_amount            22699 non-null  float64
 12  extra                  22699 non-null  float64
 13  mta_tax                22699 non-null  float64
 14  tip_amount             22699 non-null  float64
 15  tolls_amount           22699 non-null  float64
 16  improvement_surcharge  22699 non-null  float64
 17  total_amount           22699 non-null  float64
dtypes: float64(8), int64(7), object(3)
memory usage: 3.1+ MB
```

[8]:

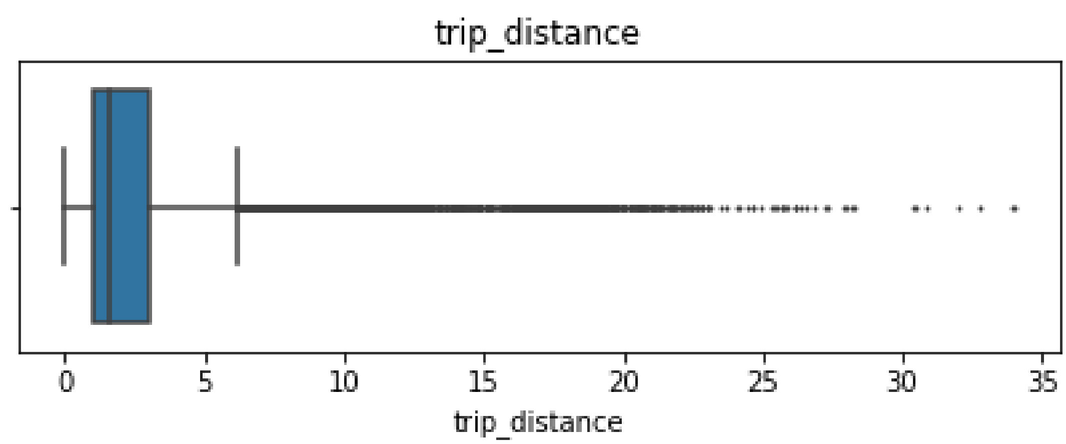
```
#A box plot will be helpful to determine outliers and where the bulk of the␣
↪data points reside in terms of trip_distance, duration, and total_amount

#A scatter plot will be helpful to visualize the trends and patters and␣
↪outliers of critical variables, such as trip_distance and total_amount

#A bar chart will help determine average number of trips per month, weekday,␣
↪weekend, etc.
```

[9]:

```
# Convert data columns to datetime
df['tpep_pickup_datetime']=pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime']=pd.to_datetime(df['tpep_dropoff_datetime'])
```

[10]:

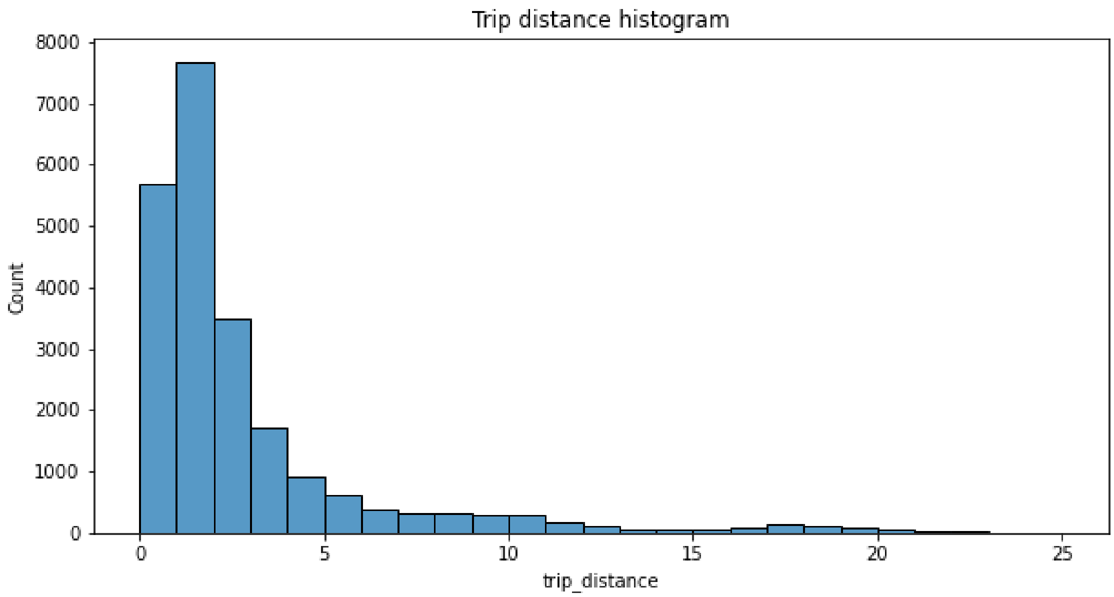
```
 # Create box plot of trip_distance
plt.figure(figsize=(7,2))
plt.title('trip_distance')
sns.boxplot(data=None, x=df['trip_distance'], fliersize=1);
```



[11]:

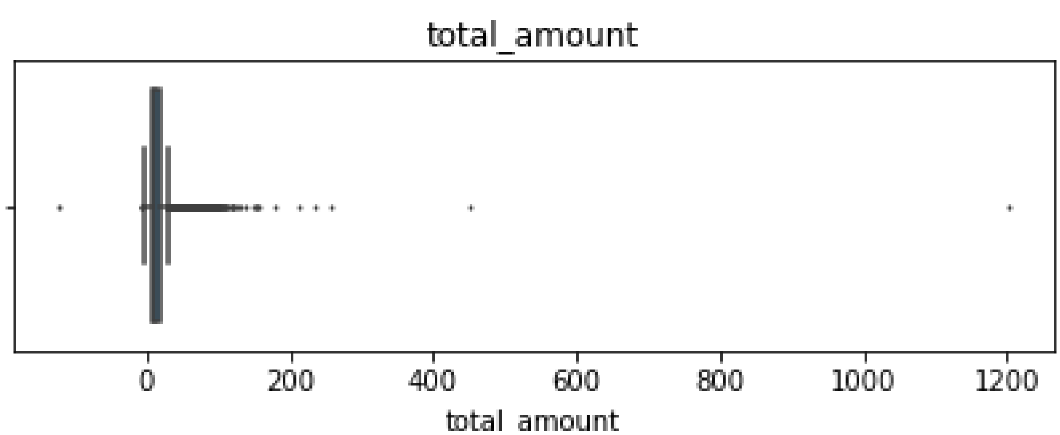
```
# Create histogram of trip_distance
plt.figure(figsize=(10,5))
```

```
sns.histplot(df['trip_distance'], bins=range(0,26,1))
plt.title('Trip distance histogram');
```

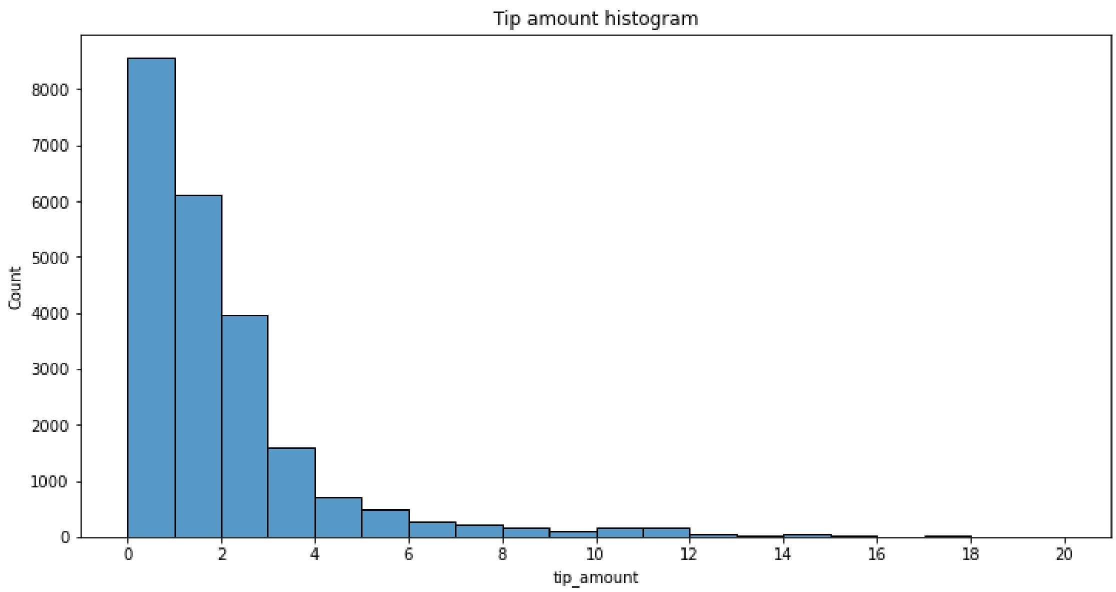


[12]:
```
#The majority of trips were journeys of less than two miles. The number of␣
↪trips falls away steeply as the distance traveled increases beyond two miles.
```

[13]:
```
# Create box plot of total_amount
plt.figure(figsize=(7,2))
plt.title('total_amount')
sns.boxplot(x=df['total_amount'], fliersize=1);
```

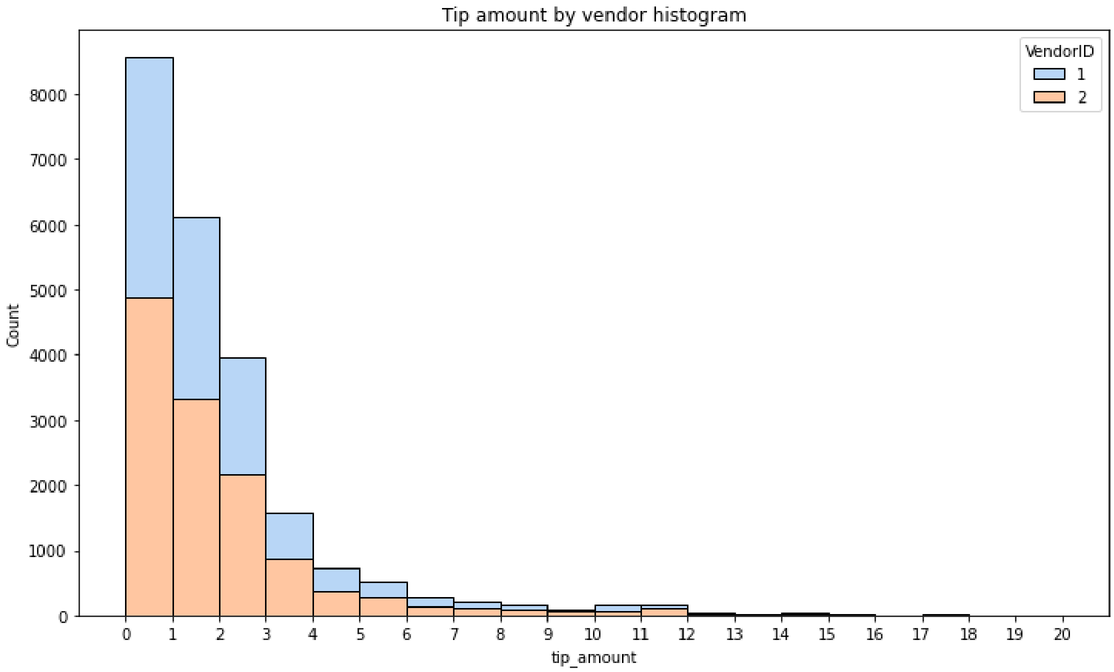


[14]:
```
# Create histogram of tip_amount
plt.figure(figsize=(12,6))
ax = sns.histplot(df['tip_amount'], bins=range(0,21,1))
ax.set_xticks(range(0,21,2))
ax.set_xticklabels(range(0,21,2))
plt.title('Tip amount histogram');
```



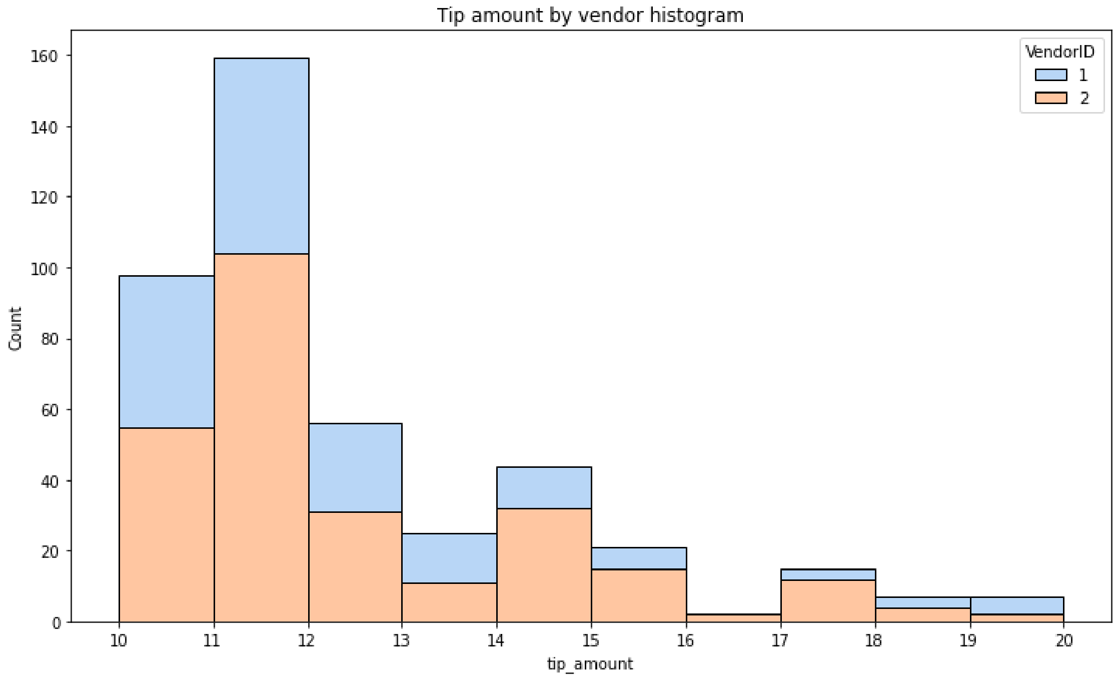
[15]:
```
#The distribution for tip amount is right-skewed, with nearly all the tips in␣
↪the $0-3 range.
```

[16]:
```
# Create histogram of tip_amount by vendor
plt.figure(figsize=(12,7))
ax = sns.histplot(data=df, x='tip_amount', bins=range(0,21,1),
                  hue='VendorID',
                  multiple='stack',
                  palette='pastel')
ax.set_xticks(range(0,21,1))
ax.set_xticklabels(range(0,21,1))
plt.title('Tip amount by vendor histogram');
```



[17]:
```
#Separating the tip amount by vendor reveals that there are no noticeable␣
↪aberrations in the distribution of tips between the two vendors in the␣
↪dataset. Vendor two has a slightly higher share of the rides, and this␣
↪proportion is approximately maintained for all tip amounts.
```

[18]:
```
# Create histogram of tip_amount by vendor for tips > $10
tips_over_ten = df[df['tip_amount'] > 10]
plt.figure(figsize=(12,7))
ax = sns.histplot(data=tips_over_ten, x='tip_amount', bins=range(10,21,1),
                  hue='VendorID',
                  multiple='stack',
                  palette='pastel')
ax.set_xticks(range(10,21,1))
ax.set_xticklabels(range(10,21,1))
plt.title('Tip amount by vendor histogram');
```



[19]:
```
#The proportions are maintained even at these higher tip amounts, with the
 ↪exception being at highest extremity, but this is not noteworthy due to the
 ↪low sample size at these tip amounts.
```

[20]:
```
df['passenger_count'].value_counts()
```

[20]:
```
1    16117
2     3305
5     1143
3      953
6      693
4      455
0       33
Name: passenger_count, dtype: int64
```

[21]:
```
# Nearly two thirds of the rides were single occupancy, though there were still
 ↪nearly 700 rides with as many as six passengers. Also, there are 33 rides
 ↪with an occupancy count of zero, which doesn't make sense. These would
 ↪likely be dropped unless a reasonable explanation can be found for them.
```

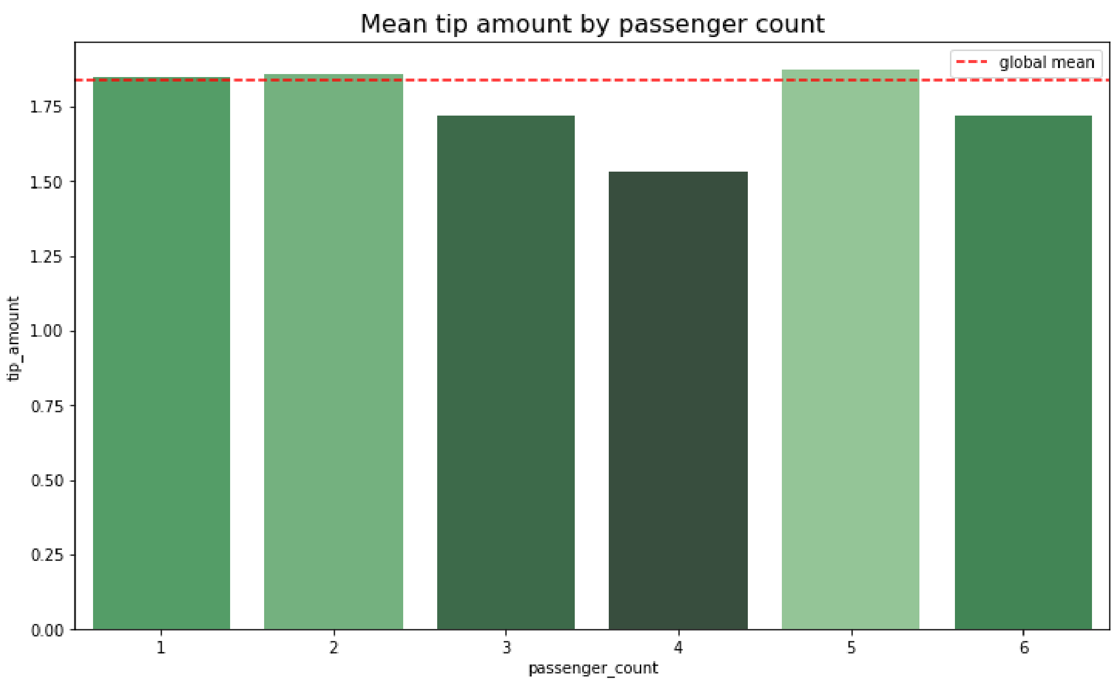
[22]:
```
# Calculate mean tips by passenger_count
mean_tips_by_passenger_count = df.groupby(['passenger_count']).
 ↪mean()[['tip_amount']]
mean_tips_by_passenger_count
```

[22]:

```
                  tip_amount
passenger_count
0                   2.135758
1                   1.848920
2                   1.856378
3                   1.716768
4                   1.530264
5                   1.873185
6                   1.720260
```

[23]:

```python
# Create bar plot for mean tips by passenger count
data = mean_tips_by_passenger_count.tail(-1)
pal = sns.color_palette("Greens_d", len(data))
rank = data['tip_amount'].argsort().argsort()
plt.figure(figsize=(12,7))
ax = sns.barplot(x=data.index,
            y=data['tip_amount'],
            palette=np.array(pal[::-1])[rank])
ax.axhline(df['tip_amount'].mean(), ls='--', color='red', label='global mean')
ax.legend()
plt.title('Mean tip amount by passenger count', fontsize=16);
```



[24]:

```
#Mean tip amount varies very little by passenger count. Although it does drop␣
↪noticeably for four-passenger rides, it's expected that there would be a␣
↪higher degree of fluctuation because rides with four passengers were the␣
↪least plentiful in the dataset (aside from rides with zero passengers).
```

[25]:
```
# Create a month column
df['month'] = df['tpep_pickup_datetime'].dt.month_name()
# Create a day column
df['day'] = df['tpep_pickup_datetime'].dt.day_name()
```

[26]:
```
# Get total number of rides for each month
monthly_rides = df['month'].value_counts()
monthly_rides
```

[26]:
```
March        2049
October      2027
April        2019
May          2013
January      1997
June         1964
December     1863
November     1843
February     1769
September    1734
August       1724
July         1697
Name: month, dtype: int64
```

[28]:
```
#The months are out of order.
#Reorder the results to put the months in calendar order.
```

[29]:
```
# Reorder the monthly ride list so months go in order
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
         'August', 'September', 'October', 'November', 'December']

monthly_rides = monthly_rides.reindex(index=month_order)
monthly_rides
```

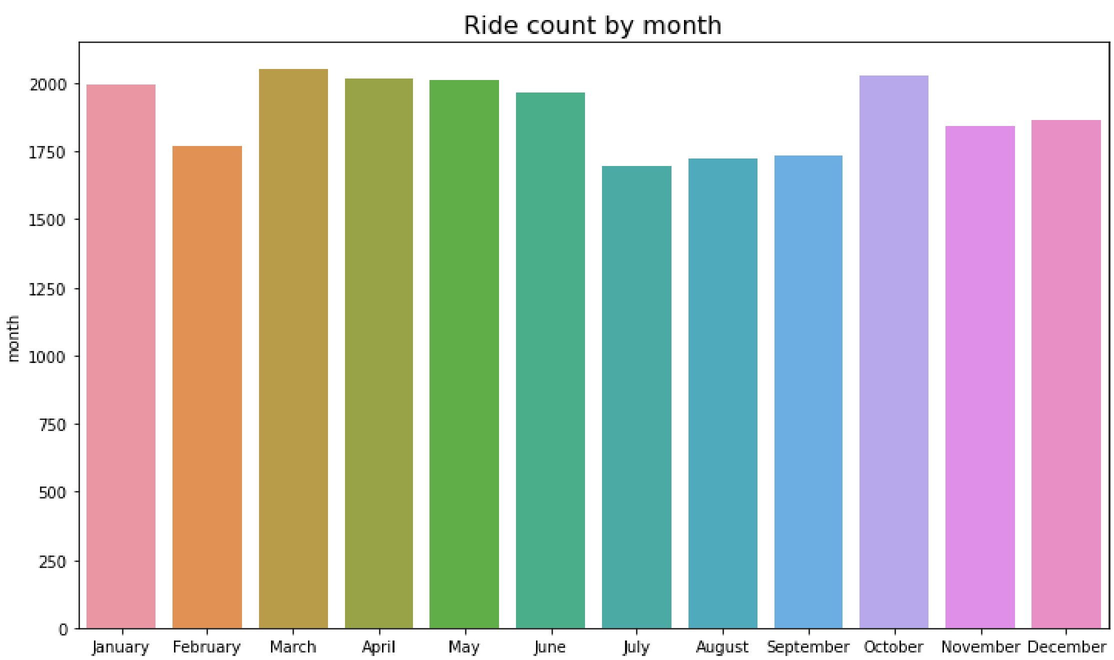
[29]:
```
January      1997
February     1769
March        2049
April        2019
May          2013
June         1964
July         1697
August       1724
September    1734
```

```
October      2027
November     1843
December     1863
Name: month, dtype: int64
```

[30]:
```
# Show the index
monthly_rides.index
```

[30]:
```
Index(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype='object')
```

[31]:
```
# Create a bar plot of total rides per month
plt.figure(figsize=(12,7))
ax = sns.barplot(x=monthly_rides.index, y=monthly_rides)
ax.set_xticklabels(month_order)
plt.title('Ride count by month', fontsize=16);
```



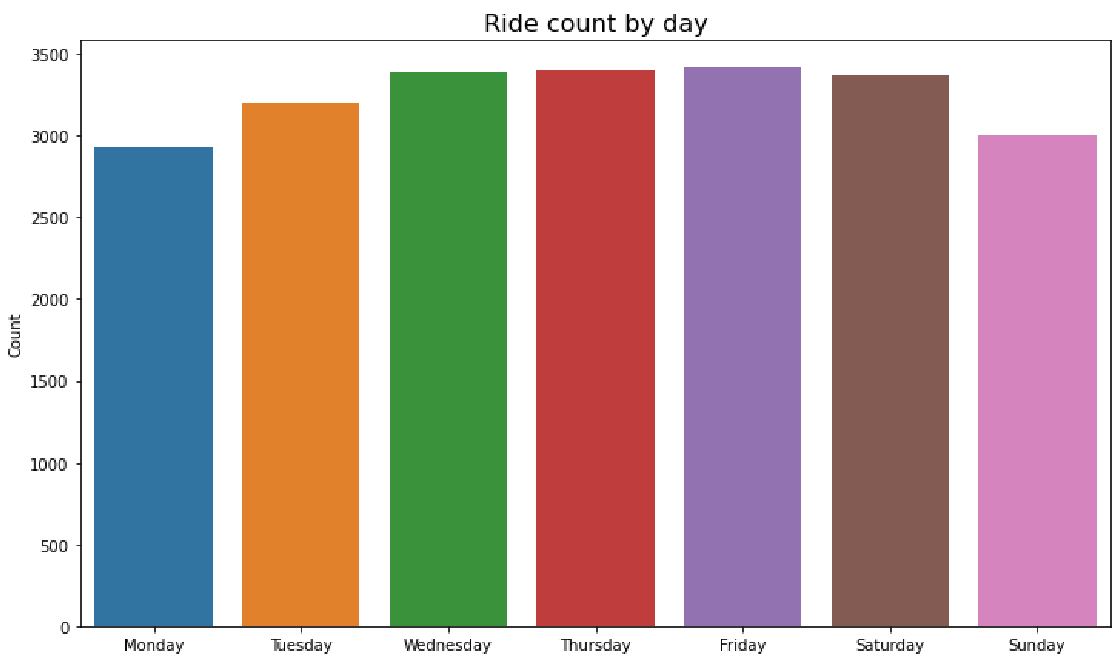
[32]:
```
#Monthly rides are fairly consistent, with notable dips in the summer months of␣
↪July, August, and September, and also in February.
```

[33]:
```
# Repeat the above process, this time for rides by day
daily_rides = df['day'].value_counts()
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
↪'Saturday', 'Sunday']
```

```
daily_rides = daily_rides.reindex(index=day_order)
daily_rides
```

[33]:
```
Monday       2931
Tuesday      3198
Wednesday    3390
Thursday     3402
Friday       3413
Saturday     3367
Sunday       2998
Name: day, dtype: int64
```

[34]:
```
# Create bar plot for ride count by day
plt.figure(figsize=(12,7))
ax = sns.barplot(x=daily_rides.index, y=daily_rides)
ax.set_xticklabels(day_order)
ax.set_ylabel('Count')
plt.title('Ride count by day', fontsize=16);
```



[35]:
```
#Suprisingly, Wednesday through Saturday had the highest number of daily rides,␣
↪while Sunday and Monday had the least.
```

[36]:
```
# Repeat the process, this time for total revenue by day
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',␣
↪'Saturday', 'Sunday']
```

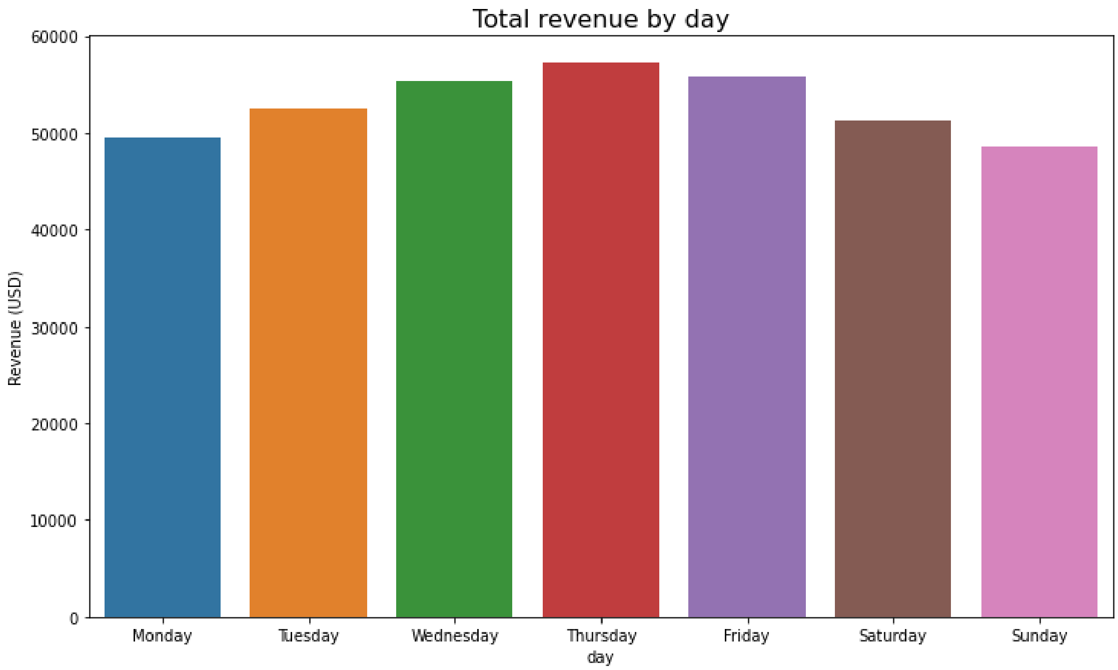
```
total_amount_day = df.groupby('day').sum()[['total_amount']]
total_amount_day = total_amount_day.reindex(index=day_order)
total_amount_day
```

[36]:

```
           total_amount
day
Monday         49574.37
Tuesday        52527.14
Wednesday      55310.47
Thursday       57181.91
Friday         55818.74
Saturday       51195.40
Sunday         48624.06
```

[37]:

```
# Create bar plot of total revenue by day
plt.figure(figsize=(12,7))
ax = sns.barplot(x=total_amount_day.index, y=total_amount_day['total_amount'])
ax.set_xticklabels(day_order)
ax.set_ylabel('Revenue (USD)')
plt.title('Total revenue by day', fontsize=16);
```



[38]:

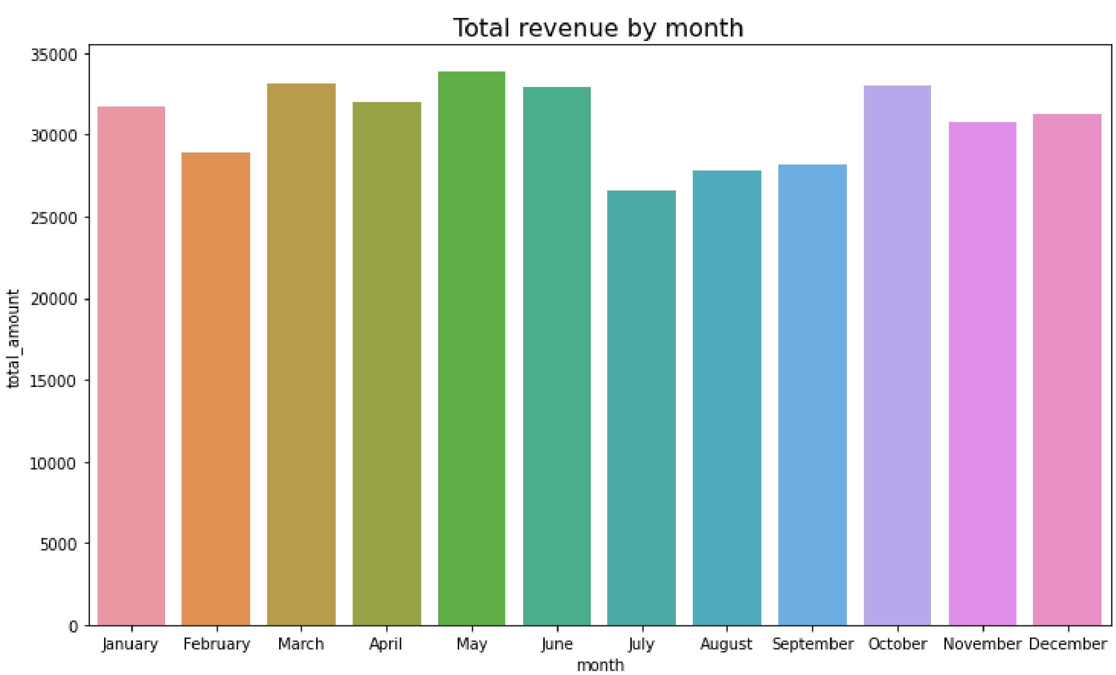
```
#Thursday had the highest gross revenue of all days, and Sunday and Monday had␣
↪the least. Interestingly, although Saturday had only 35 fewer rides than␣
↪Thursday, its gross revenue was ~$6,000 less than Thursday's-more than a 10%␣
↪drop.
```

[39]:
```
# Repeat the process, this time for total revenue by month
total_amount_month = df.groupby('month').sum()[['total_amount']]
total_amount_month = total_amount_month.reindex(index=month_order)
total_amount_month
```

[39]:

month	total_amount
January	31735.25
February	28937.89
March	33085.89
April	32012.54
May	33828.58
June	32920.52
July	26617.64
August	27759.56
September	28206.38
October	33065.83
November	30800.44
December	31261.57

[40]:
```
# Create a bar plot of total revenue by month
plt.figure(figsize=(12,7))
ax = sns.barplot(x=total_amount_month.index,␣
↪y=total_amount_month['total_amount'])
plt.title('Total revenue by month', fontsize=16);
```



[41]:
```
#Monthly revenue generally follows the pattern of monthly rides, with␣
↪noticeable dips in the summer months of July, August, and September, and␣
↪also one in February.
```

[42]:
```
# Get number of unique drop-off location IDs
df['DOLocationID'].nunique()
```

[42]: 216

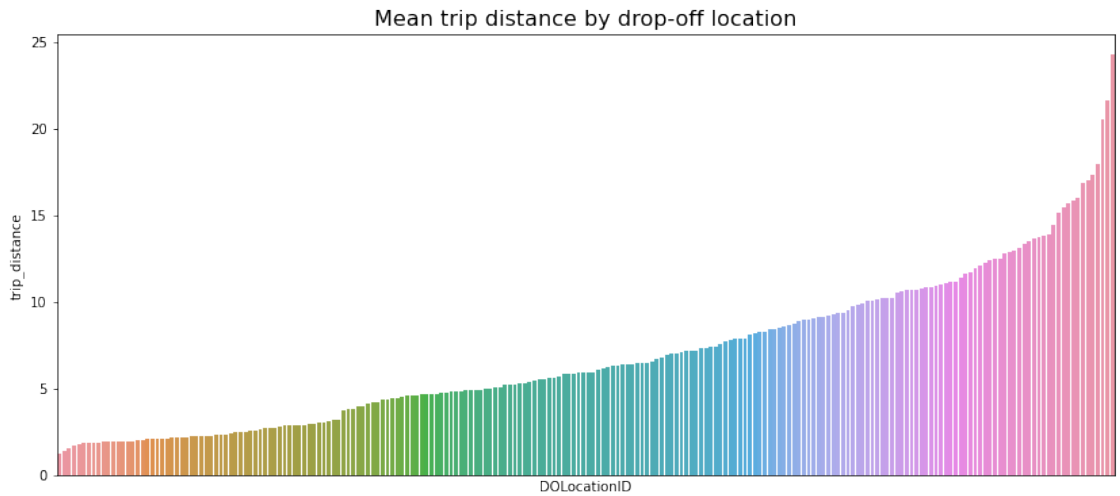
[43]:
```
# Calculate the mean trip distance for each drop-off location
distance_by_dropoff = df.groupby('DOLocationID').mean()[['trip_distance']]

# Sort the results in descending order by mean trip distance
distance_by_dropoff = distance_by_dropoff.sort_values(by='trip_distance')
distance_by_dropoff
```

[43]:

DOLocationID	trip_distance
207	1.200000
193	1.390556
237	1.555494
234	1.727806
137	1.818852
…	…
51	17.310000
11	17.945000
210	20.500000
29	21.650000
23	24.275000

[216 rows x 1 columns]

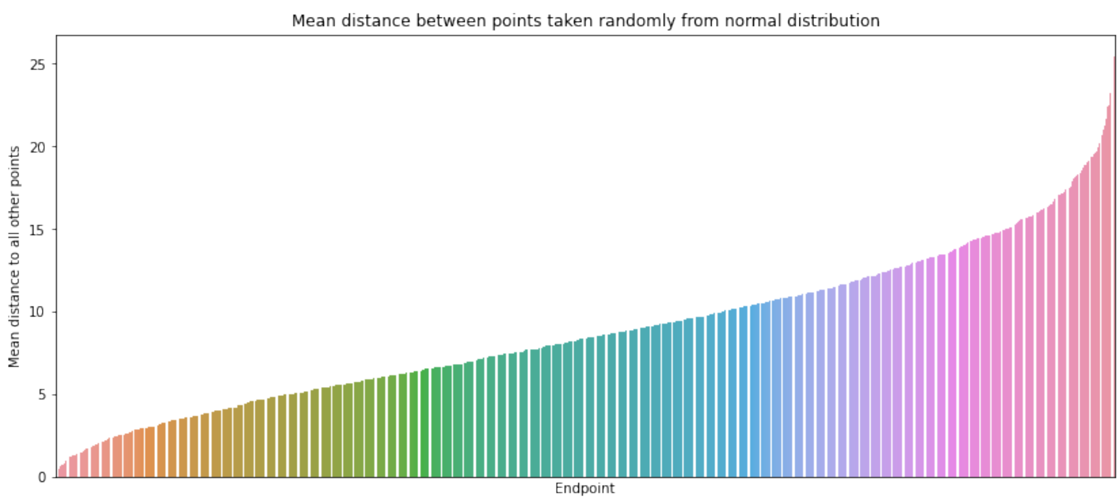
[44]:
```
# Create a bar plot of mean trip distances by drop-off location in ascending␣
↪order by distance
plt.figure(figsize=(14,6))
ax = sns.barplot(x=distance_by_dropoff.index,
                 y=distance_by_dropoff['trip_distance'],
                 order=distance_by_dropoff.index)
ax.set_xticklabels([])
ax.set_xticks([])
plt.title('Mean trip distance by drop-off location', fontsize=16);
```



[45]:
```
#This plot presents a characteristic curve related to the cumulative density
 ↪function of a normal distribution. In other words, it indicates that the
 ↪drop-off points are relatively evenly distributed over the terrain. This is
 ↪good to know, because geographic coordinates were not included in this
 ↪dataset, so there was no obvious way to test for the distibution of
 ↪locations.

#To confirm this conclusion, consider the following experiment:

#Create a sample of coordinates from a normal distribution-in this case 1,500
 ↪pairs of points from a normal distribution with a mean of 10 and a standard
 ↪deviation of 5
#Calculate the distance between each pair of coordinates
#Group the coordinates by endpoint and calculate the mean distance between that
 ↪endpoint and all other points it was paired with
#Plot the mean distance for each unique endpoint
```

[46]:
```
# 1. Generate random points on a 2D plane from a normal distribution
test = np.round(np.random.normal(10, 5, (3000, 2)), 1)
midway = int(len(test)/2)  # Calculate midpoint of the array of coordinates
start = test[:midway]      # Isolate first half of array ("pick-up locations")
end = test[midway:]        # Isolate second half of array ("drop-off locations")

# 2. Calculate Euclidean distances between points in first half and second half
 ↪of array
distances = (start - end)**2
distances = distances.sum(axis=-1)
distances = np.sqrt(distances)

# 3. Group the coordinates by "drop-off location", compute mean distance
```

```
test_df = pd.DataFrame({'start': [tuple(x) for x in start.tolist()],
                   'end': [tuple(x) for x in end.tolist()],
                   'distance': distances})
data = test_df[['end', 'distance']].groupby('end').mean()
data = data.sort_values(by='distance')

# 4. Plot the mean distance between each endpoint ("drop-off location") and all␣
 ↪points it connected to
plt.figure(figsize=(14,6))
ax = sns.barplot(x=data.index,
                 y=data['distance'],
                 order=data.index)
ax.set_xticklabels([])
ax.set_xticks([])
ax.set_xlabel('Endpoint')
ax.set_ylabel('Mean distance to all other points')
ax.set_title('Mean distance between points taken randomly from normal␣
 ↪distribution');
```



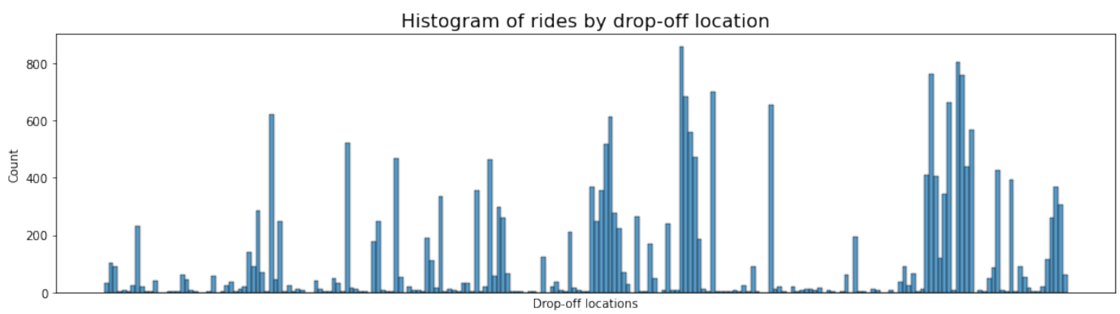
[47]:
```
# The curve described by this graph is nearly identical to that of the mean␣
 ↪distance traveled by each taxi ride to each drop-off location. This reveals␣
 ↪that the drop-off locations in the taxi dataset are evenly distributed␣
 ↪geographically. Note, however, that this does not mean that there was an␣
 ↪even distrubtion of rides to each drop-off point.
```

[48]:
```
# Check if all drop-off locations are consecutively numbered
df['DOLocationID'].max() - len(set(df['DOLocationID']))
```

[48]: 49

[49]:
```
#There are 49 numbers that do not represent a drop-off location.

#To eliminate the spaces in the historgram that these missing numbers would
↪create, sort the unique drop-off location values, then convert them to
↪strings. This will make the histplot function display all bars directly next
↪to each other.
```

[50]:
```
plt.figure(figsize=(16,4))
# DOLocationID column is numeric, so sort in ascending order
sorted_dropoffs = df['DOLocationID'].sort_values()
# Convert to string
sorted_dropoffs = sorted_dropoffs.astype('str')
# Plot
sns.histplot(sorted_dropoffs, bins=range(0, df['DOLocationID'].max()+1, 1))
plt.xticks([])
plt.xlabel('Drop-off locations')
plt.title('Histogram of rides by drop-off location', fontsize=16);
```



[51]:
```
#Notice that out of the 200+ drop-off locations, a disproportionate number of
↪locations receive the majority of the traffic, while all the rest get
↪relatively few trips. It's likely that these high-traffic locations are near
↪popular tourist attractions
```

[ ]:
```

```