# Automatidata project

**Go Beyond the Numbers: Translate Data into Insights**

I am the newest data professional in a fictional data consulting firm: Automatidata. The team is still early into the project, having only just completed an initial plan of action and some early Python coding work.

Luana Rodriquez, the senior data analyst at Automatidata, is pleased with the work I have already completed and requests my assistance with some EDA and data visualization work for the New York City Taxi and Limousine Commission project (New York City TLC) to get a general understanding of what taxi ridership looks like. The management team is asking for a Python notebook showing data structuring and cleaning, as well as any matplotlib/seaborn visualizations plotted to help understand the data. At the very least, include a box plot of the ride durations and some time series plots, like a breakdown by quarter or month.

# Project: Exploratory data analysis

In this activity, I will examine data provided and prepare it for analysis. I will also design a professional data visualization that tells a story, and will help data-driven decisions for business needs.

**The purpose** of this project is to conduct exploratory data analysis on a provided data set. My mission is to continue the investigation I began in C2 and perform further EDA on this data with the aim of learning more about the variables.

**The goal** is to clean data set and create a visualization.

*This activity has 4 parts:*

**Part 1:** Imports, links, and loading

**Part 2:** Data Exploration

- Data cleaning

**Part 3:** Building visualizations

**Part 4:** Evaluate and share results

# Visualize a story in Python

# PACE stages

- Plan

- Analyze
- Construct
- Execute

Throughout these project notebooks, you'll see references to the problem-solving framework PACE. The following notebook components are labeled with the respective PACE stage: Plan, Analyze, Construct, and Execute.

# PACE: Plan

In this stage, we will consider the following questions where applicable to complete the code response:

**Exemplar response:**

1. Identify any outliers:

- What methods are best for identifying outliers?
- Use numpy functions to investigate the `mean()` and `median()` of the data and understand range of data values
- Use a boxplot to visualize the distribution of the data
- Use histograms to visualize the distribution of the data
- How do we make the decision to keep or exclude outliers from any future models?
- There are three main options for dealing with outliers: keeping them as they are, deleting them, or reassigning them. Whether we keep outliers as they are, delete them, or reassign values is a decision that we make taking into account the nature of the outlying data and the assumptions of the model we are building. To help us make the decision, we can start with these general guidelines:
  - Delete them: If we are sure the outliers are mistakes, typos, or errors and the dataset will be used for modeling or machine learning, then we are more likely to decide to delete outliers. If the three choices, we will use this one the least.
  - Reassign them: If the dataset is small and/or the data will be used for modeling or machine learning, we are more likely to choose a path of deriving new values to replace the outlier values.
  - Leave them: For a dataset that we plan to do EDA/analysis on and nothing else, or for a dataset we are preparing for a model that is resistant to outliers, it is most likely that we are going to leave them in.

## Task 1. Imports, links, and loading

For EDA of the data, import the data and packages that would be most helpful, such as pandas, numpy and matplotlib.

Then, import the dataset.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt
import seaborn as sns
```

In [2]:
```
df=pd.read_csv('data/2017_Yellow_Taxi_Trip_Data.csv')
```

# PACE: Analyze

Consider these questions in your PACE Strategy Document to reflect on the Analyze stage.

## Task 2a. Data exploration and cleaning

Decide which columns are applicable

Given our scenario, which data columns are most applicable? Which data columns can we eliminate, knowing they won't solve our problem scenario?

Consider functions that help us understand and structure the data.

- `head()`
- `describe()`
- `info()`
- `groupby()`
- `sortby()`

Consider these questions as we work:

What do we do about missing data (if any)?

Are there data outliers?

What do the distributions of the variables tell us about the question we're asking or the problem we're trying to solve?

Find these answers later in the notebook.

Start by discovering, using head and size.

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

Out[3]:

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

In [4]:
```
df.size
```

Out[4]:
```
408582
```

Use describe...

In [5]:
```
df.describe()
```

Out[5]:

| | Unnamed: 0 | VendorID | passenger_count | trip_distance | RatecodeID | PULocationID |
|---|---|---|---|---|---|---|
| **count** | 2.269900e+04 | 22699.000000 | 22699.000000 | 22699.000000 | 22699.000000 | 22699.000000 |
| **mean** | 5.675849e+07 | 1.556236 | 1.642319 | 2.913313 | 1.043394 | 162.412353 |
| **std** | 3.274493e+07 | 0.496838 | 1.285231 | 3.653171 | 0.708391 | 66.633373 |
| **min** | 1.212700e+04 | 1.000000 | 0.000000 | 0.000000 | 1.000000 | 1.000000 |
| **25%** | 2.852056e+07 | 1.000000 | 1.000000 | 0.990000 | 1.000000 | 114.000000 |
| **50%** | 5.673150e+07 | 2.000000 | 1.000000 | 1.610000 | 1.000000 | 162.000000 |
| **75%** | 8.537452e+07 | 2.000000 | 2.000000 | 3.060000 | 1.000000 | 233.000000 |
| **max** | 1.134863e+08 | 2.000000 | 6.000000 | 33.960000 | 99.000000 | 265.000000 |

And info.

In [6]:
```
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
```

**Exemplar note:** There is no missing data according to the results from the `info()` function.

## Task 2b. Assess whether dimensions and measures are correct

In Python, consider the data types of the columns. *Consider:* Do they make sense?

### Task 2c. Select visualization type(s)

Selected data visualization types that will help us understand and explain the data.

Now that we know which data columns we'll use, it is time to decide which data visualization makes the most sense for EDA of the TLC dataset. What type of data visualization(s) would be most helpful?

- Line graph
- Bar chart
- Box plot
- Histogram
- Heat map
- Scatter plot
- A geographic map

**Exemplar response:**

As we'll see below, a bar chart, box plot and scatter plot will be most helpful in the understanding of this data.

*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.*

# PACE: Construct

Consider these questions in your PACE Strategy Document to reflect on the Construct stage.

### Task 3. Data visualization

We've assessed the data, and decided on which data variables are most applicable. It's time to plot the visualization(s)!

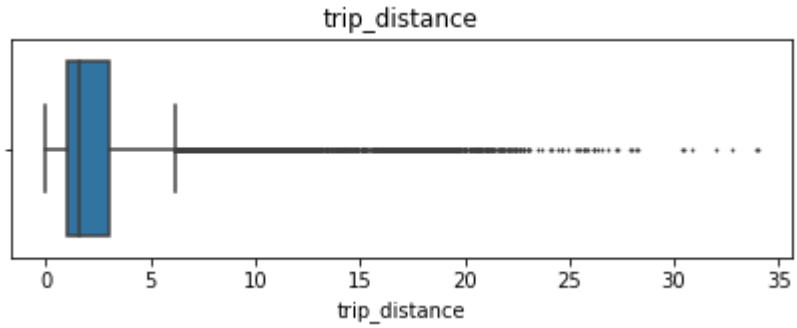
# Boxplots

Perform a check for outliers on relevant columns such as trip distance and trip duration. Remember, some of the best ways to identify the presence of outliers in data are box plots and histograms.

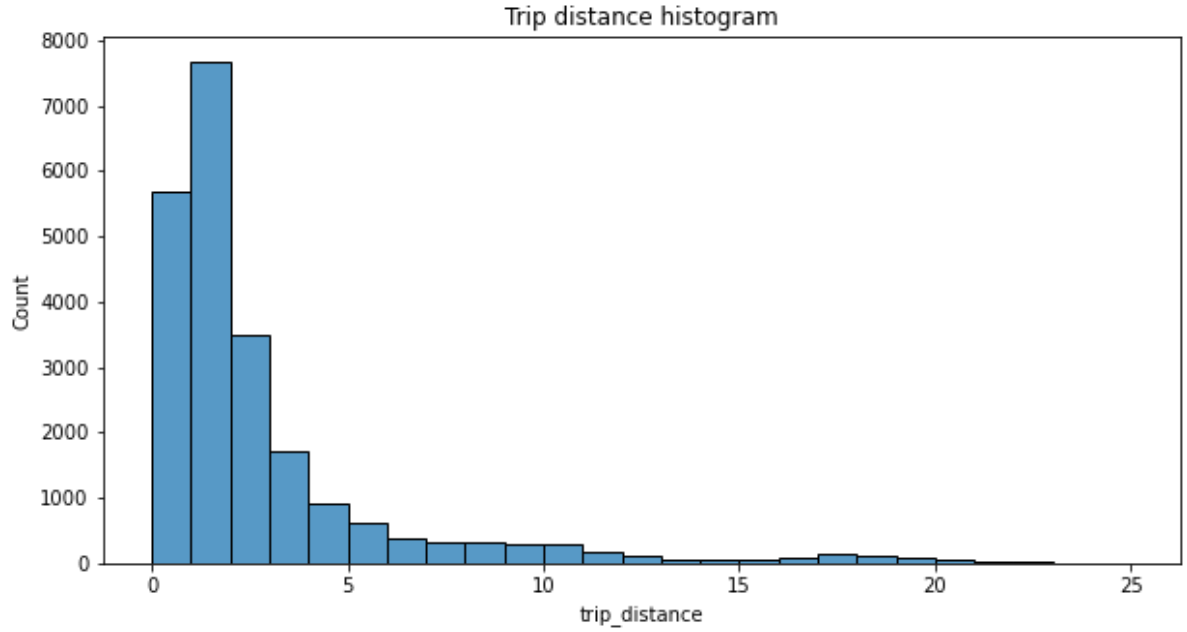
In [7]:
```
# Convert data columns to datetime
df['tpep_pickup_datetime']=pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime']=pd.to_datetime(df['tpep_dropoff_datetime'])
```

**trip_distance**

In [8]:
```
# 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);
```



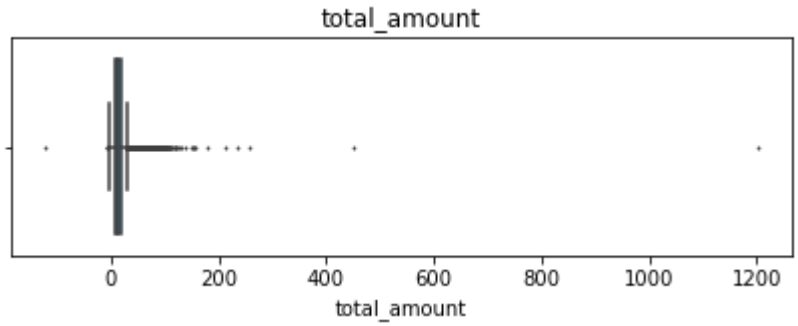

In [9]:
```
# Create histogram of trip_distance
plt.figure(figsize=(10,5))
sns.histplot(df['trip_distance'], bins=range(0,26,1))
plt.title('Trip distance histogram');
```



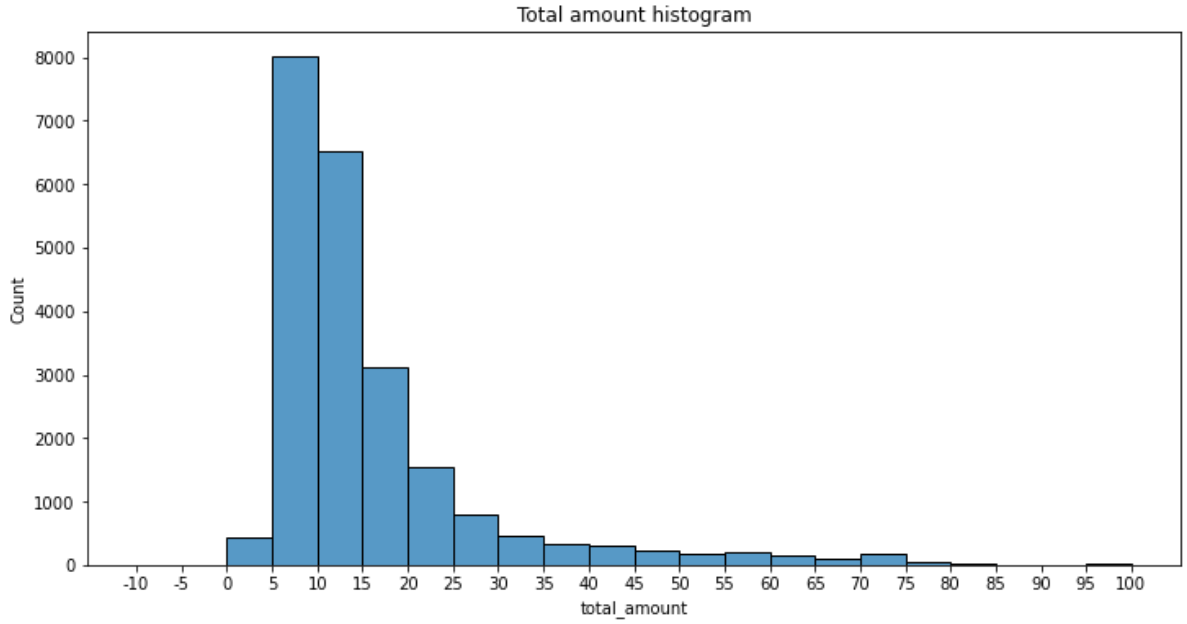

**Exemplar note:** The majority of trips were journeys of less than two miles. The number of trips falls away steeply as the distance traveled increases beyond two miles.

**total_amount**

In [10]:
```
# Create box plot of total_amount
plt.figure(figsize=(7,2))
plt.title('total_amount')
sns.boxplot(x=df['total_amount'], fliersize=1);
```



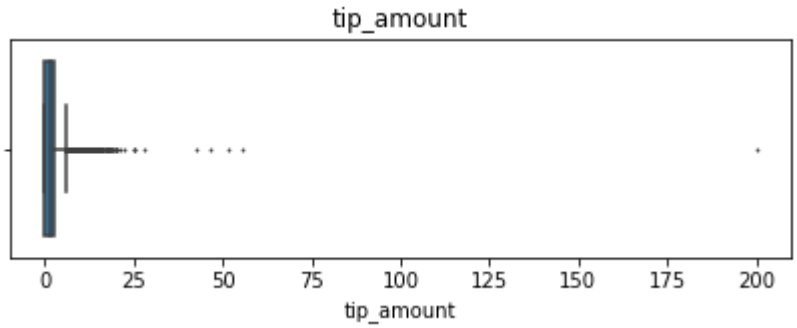

In [11]:
```
# Create histogram of total_amount
plt.figure(figsize=(12,6))
ax = sns.histplot(df['total_amount'], bins=range(-10,101,5))
ax.set_xticks(range(-10,101,5))
ax.set_xticklabels(range(-10,101,5))
plt.title('Total amount histogram');
```



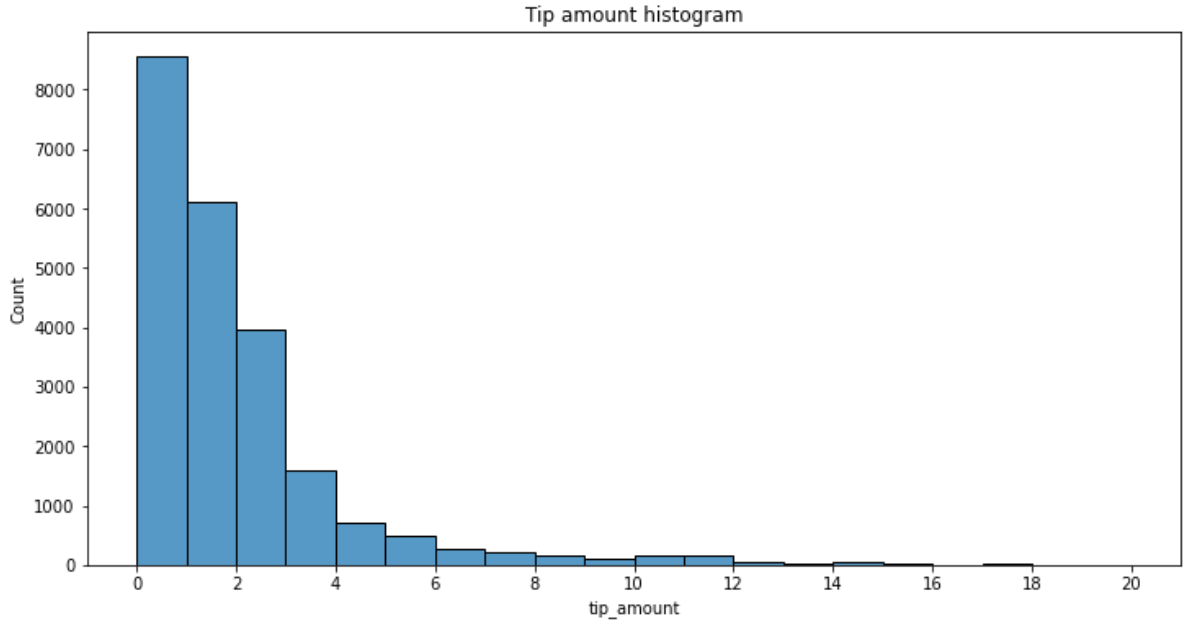

**Exemplar note:** The total cost of each trip also has a distribution that skews right, with most costs falling in the $5-15 range.

**tip_amount**

In [12]:
```
# Create box plot of tip_amount
plt.figure(figsize=(7,2))
plt.title('tip_amount')
sns.boxplot(x=df['tip_amount'], fliersize=1);
```



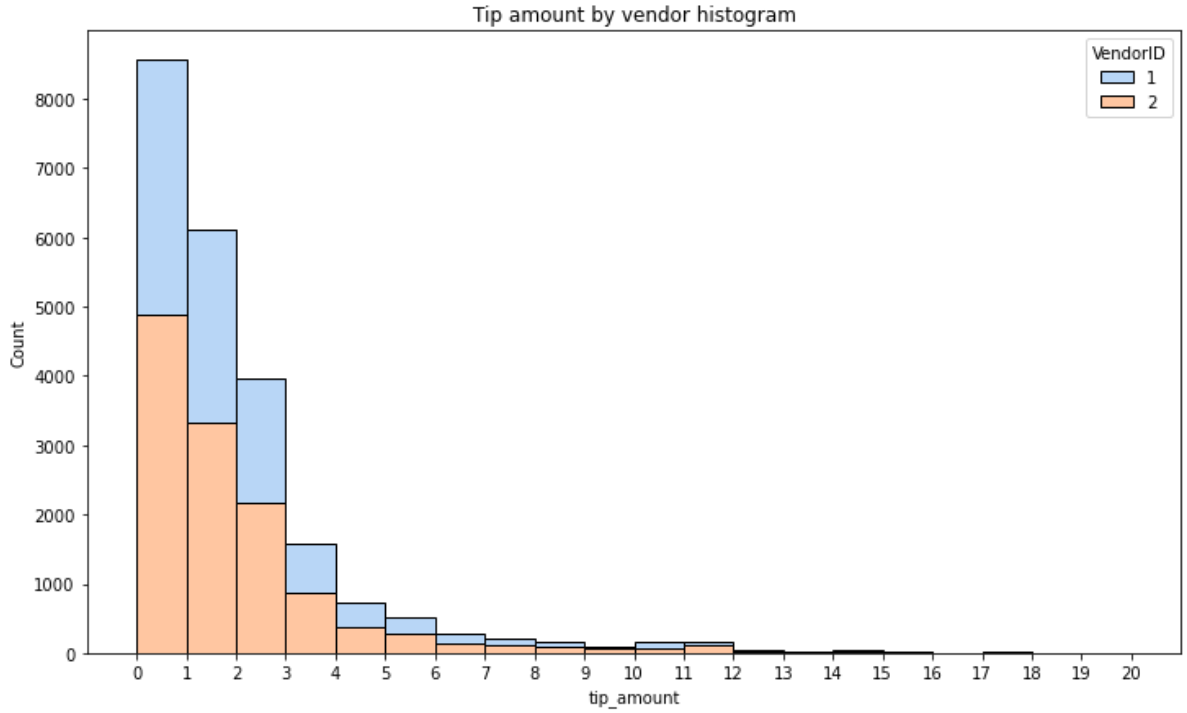

In [13]:
```
# Create histogram of tip_amount
plt.figure(figsize=(12,6))
ax = sns.histplot(df['tip_amount'], bins=range(0,21,1))
ax.set_xticks(range(0,21,2))
ax.set_xticklabels(range(0,21,2))
plt.title('Tip amount histogram');
```




**Exemplar note:** The distribution for tip amount is right-skewed, with nearly all the tips in the $0-3 range.

**tip_amount by vendor**

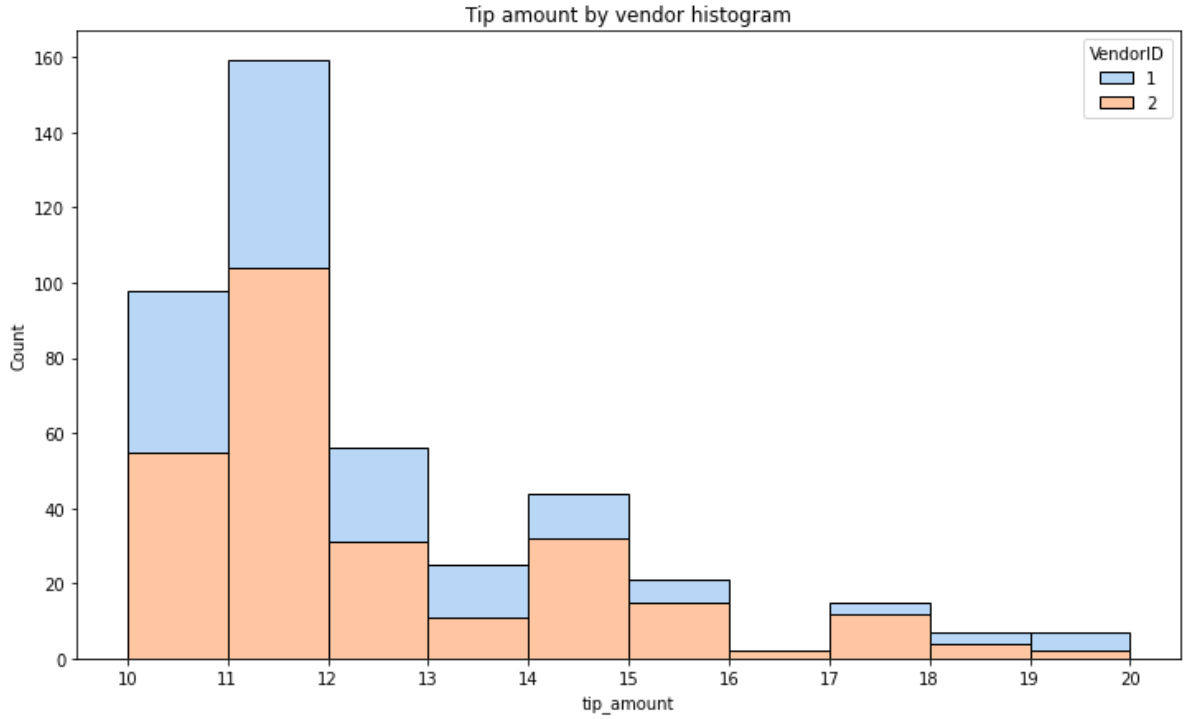
In [14]:
```
# Create histogram of tip_amount by vendor
plt.figure(figsize=(12,7))
ax = sns.histplot(data=df, x='tip_amount', bins=range(0,21,1),
                  hue='VendorID',
                  multiple='stack',
                  palette='pastel')
ax.set_xticks(range(0,21,1))
ax.set_xticklabels(range(0,21,1))
plt.title('Tip amount by vendor histogram');
```




**Exemplar note:** 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.

Next, zoom in on the upper end of the range of tips to check whether vendor one gets noticeably more of the most generous tips.

In [15]:
```
# 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');
```




**Exemplar note:** 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.

**Mean tips by passenger count**

Examine the unique values in the `passenger_count` column.

In [16]:
```
df['passenger_count'].value_counts()
```

Out[16]:
```
1    16117
2     3305
5     1143
3      953
6      693
4      455
0       33
Name: passenger_count, dtype: int64
```

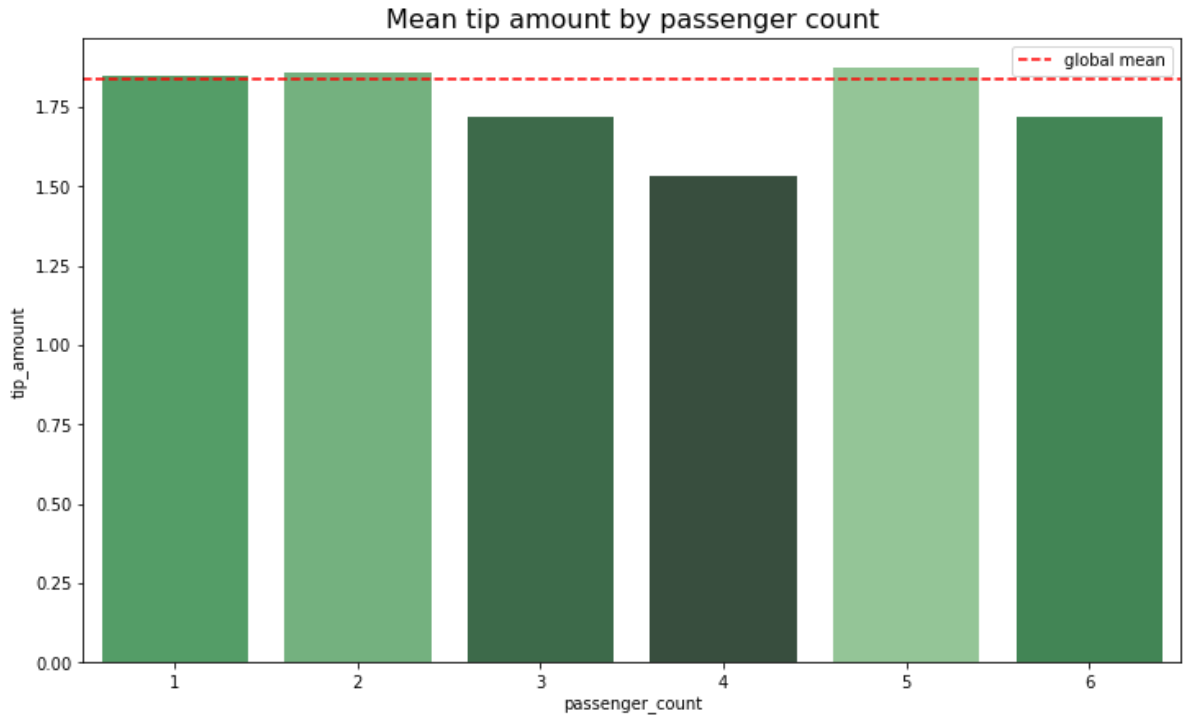
**Exemplar note:** Nearly two thirds of the rides were single occupancy, though there were still nearly 700 rides with as many as six passengers. Also, there are 33 rides with an occupancy count of zero, which doesn't make sense. These would likely be dropped unless a reasonable explanation can be found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

In [17]:
```
# Calculate mean tips by passenger_count
mean_tips_by_passenger_count = df.groupby(['passenger_count']).mean()[['tip_am
mean_tips_by_passenger_count
```

Out[17]:

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

In [18]:
```
# 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);
```




**Exemplar note:** 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).

**Create month and day columns**

In [19]:
```
# 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()
```

**Plot total ride count by month**

Begin by calculating total ride count by month.

In [20]:
```
# Get total number of rides for each month
monthly_rides = df['month'].value_counts()
monthly_rides
```

Out[20]:
```
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
```

**Exemplar note:** The months are out of order.

Reorder the results to put the months in calendar order.

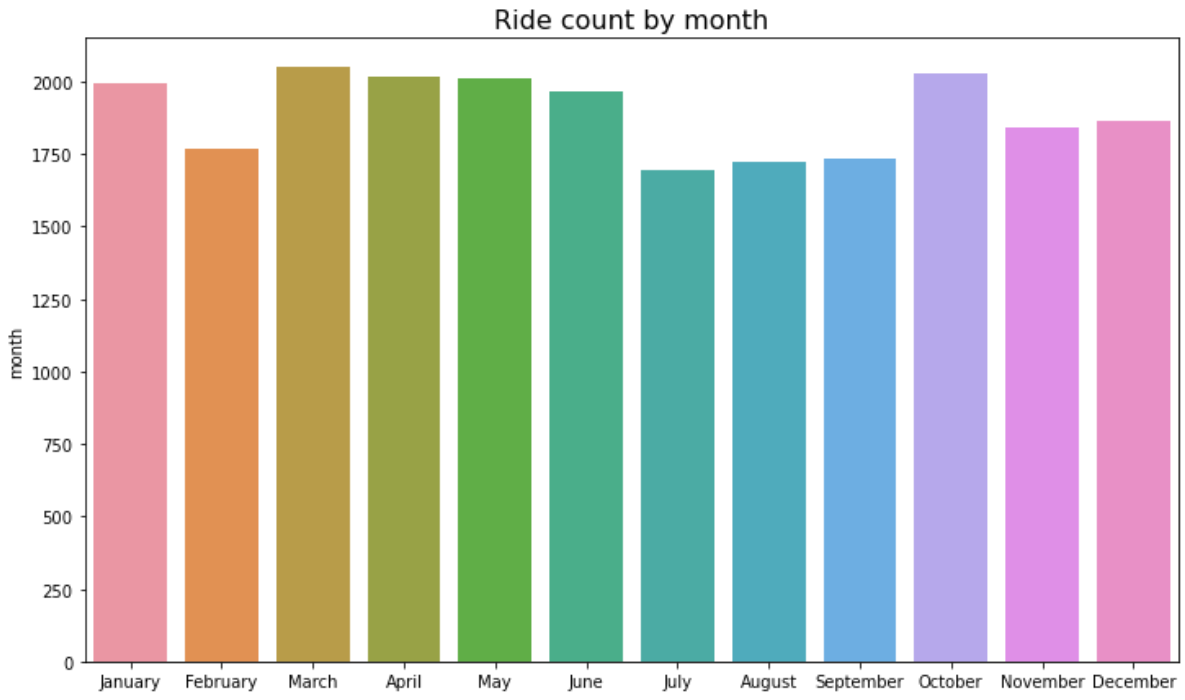
In [21]:
```
# Reorder the monthly ride list so months go in order
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July',
         'August', 'September', 'October', 'November', 'December']

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

Out[21]:
```
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
```

In [22]:
```
# Show the index
monthly_rides.index
```

Out[22]:
```
Index(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype='object')
```

In [23]:
```
# 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);
```




**Exemplar note:** Monthly rides are fairly consistent, with notable dips in the summer months of July, August, and September, and also in February.

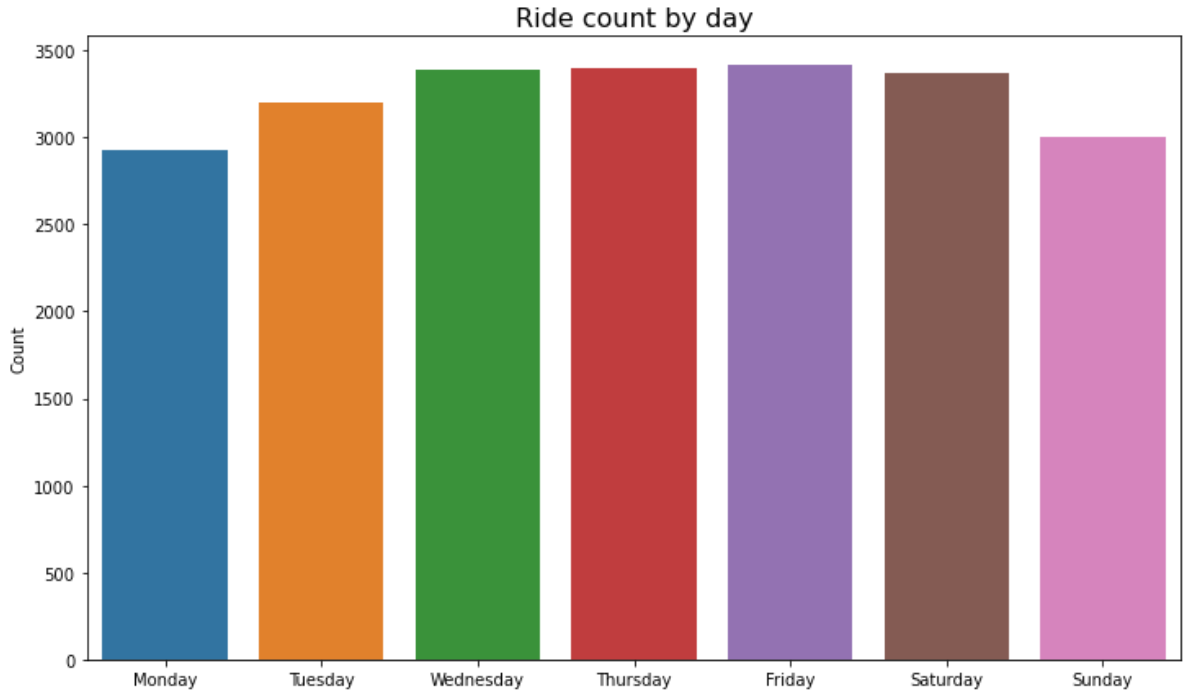
**Plot total ride count by day**

Repeat the above process, but now calculate the total rides by day of the week.

In [24]:
```
# Repeat the above process, this time for rides by day
daily_rides = df['day'].value_counts()
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday
daily_rides = daily_rides.reindex(index=day_order)
daily_rides
```

Out[24]:
```
Monday       2931
Tuesday      3198
Wednesday    3390
Thursday     3402
Friday       3413
Saturday     3367
Sunday       2998
Name: day, dtype: int64
```

In [25]:
```
# 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);
```




**Exemplar note:** Suprisingly, Wednesday through Saturday had the highest number of daily rides, while Sunday and Monday had the least.

**Plot total revenue by day of the week**

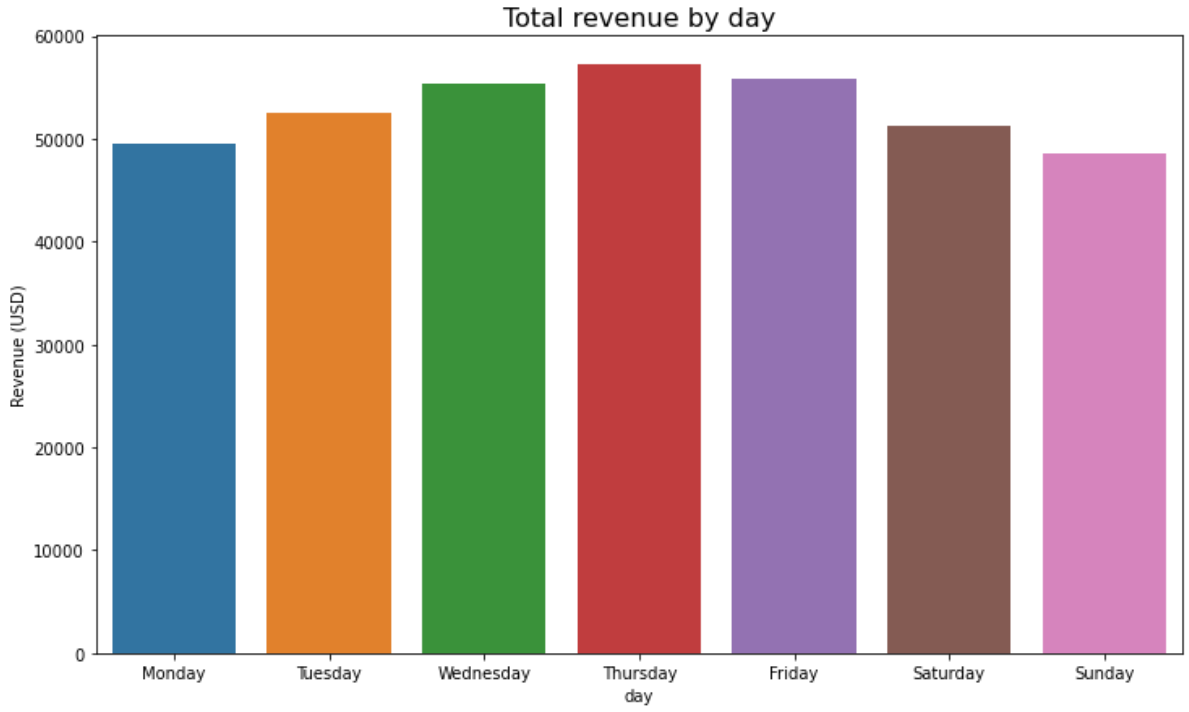
Repeat the above process, but now calculate the total revenue by day of the week.

In [26]:
```
# Repeat the process, this time for total revenue by day
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday
total_amount_day = df.groupby('day').sum()[['total_amount']]
total_amount_day = total_amount_day.reindex(index=day_order)
total_amount_day
```

Out[26]:

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

In [27]:
```
# 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);
```




**Exemplar note:** 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.

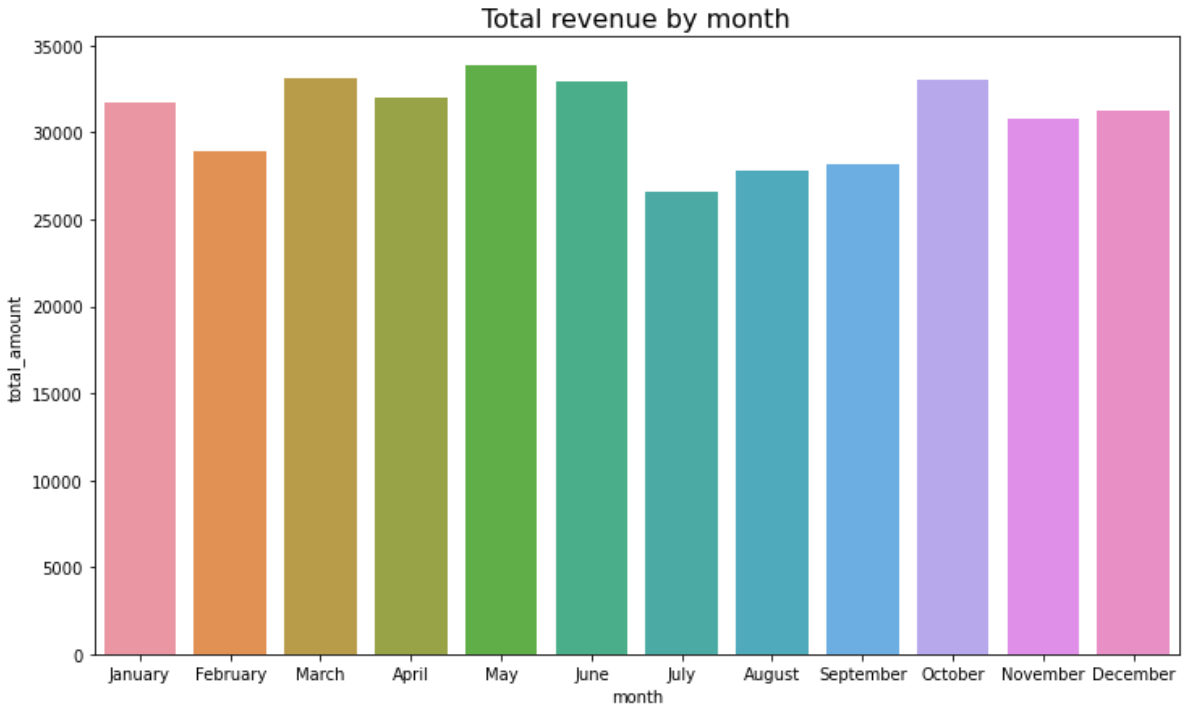
## Plot total revenue by month

In [28]:
```
# 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
```

Out[28]:

| 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 |

In [29]:
```
# 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_amoun
plt.title('Total revenue by month', fontsize=16);
```



**Exemplar note:** 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.

**Plot mean trip distance by drop-off location**

In [30]:
```
# Get number of unique drop-off location IDs
df['DOLocationID'].nunique()
```

Out[30]: 216

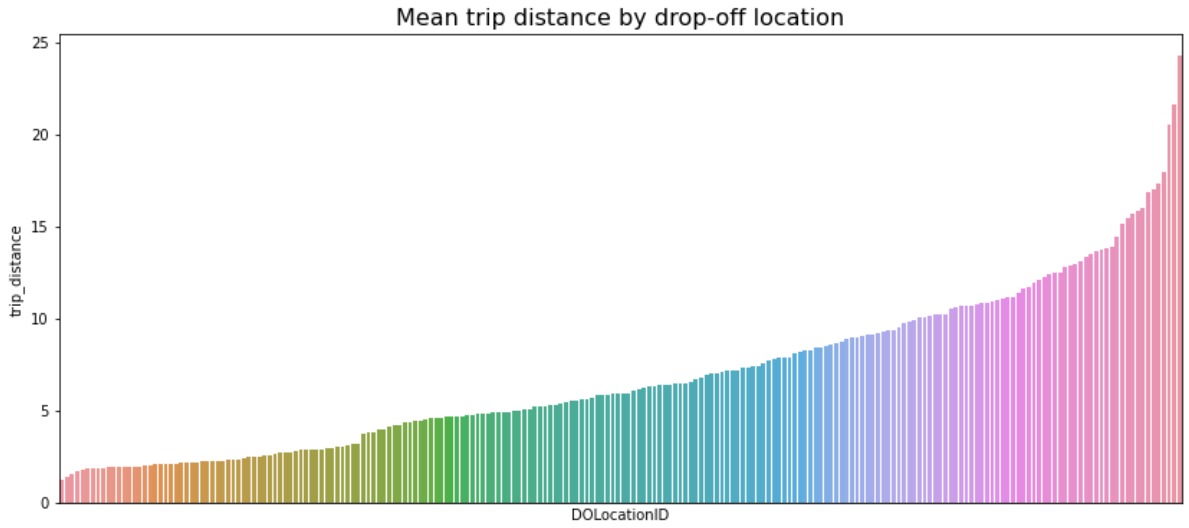
In [31]:
```
# Calculate the mean trip distance for each drop-off location
distance_by_dropoff = df.groupby('DOLocationID').mean()[['trip_distance']]

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

Out[31]:

| 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 × 1 columns

In [32]:
```
# Create a bar plot of mean trip distances by drop-off location in ascending or
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);
```

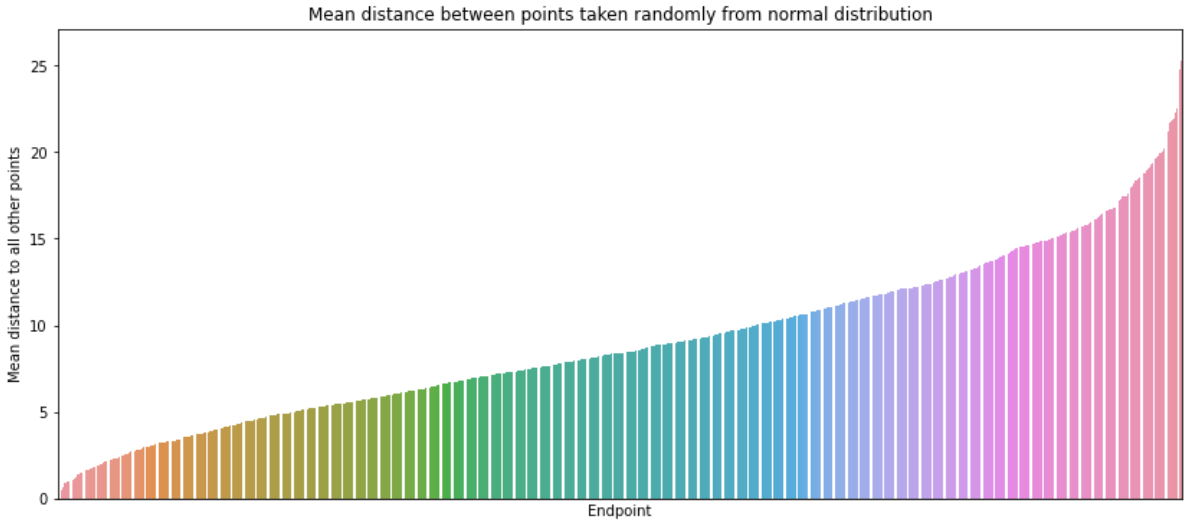



**Exemplar note:** This plot presents a characteristic curve related to the cumulative density function of a normal distribution. In other words, it indicates that the drop-off points are relatively evenly distributed over the terrain. This is good to know, because geographic coordinates were not included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

To confirm this conclusion, consider the following experiment:

1. 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
2. Calculate the distance between each pair of coordinates
3. Group the coordinates by endpoint and calculate the mean distance between that endpoint and all other points it was paired with
4. Plot the mean distance for each unique endpoint

In [33]:
```
# 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
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 al
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 distribut
```




**Exemplar note:** 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. Examine this next.

**Histogram of rides by drop-off location**

First, check whether the drop-off locations IDs are consecutively numbered. For instance, does it go 1, 2, 3, 4..., or are some numbers missing (e.g., 1, 3, 4...). If numbers aren't all consecutive, the histogram will look like some locations have very few or no rides when in reality there's no bar because there's no location.

There are many ways to do this.

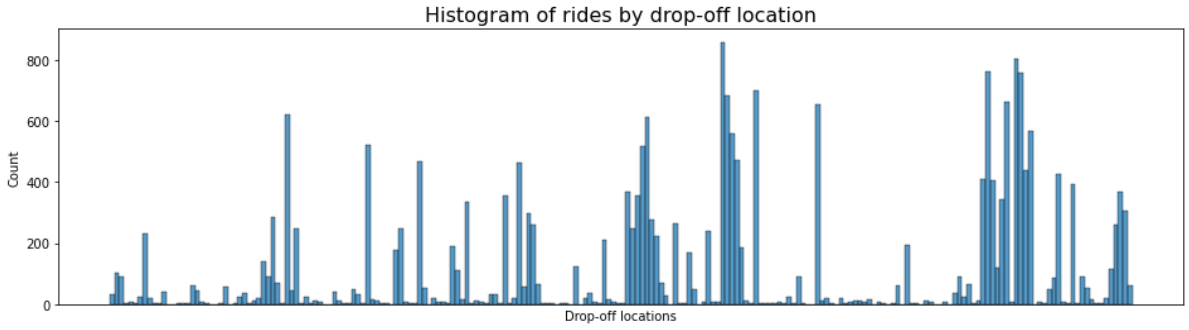
In [34]:
```
# Check if all drop-off locations are consecutively numbered
df['DOLocationID'].max() - len(set(df['DOLocationID']))
```

Out[34]: 49

**Exemplar note:** 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.

In [35]:
```
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);
```




**Exemplar note:** 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 like the Empire State Building or Times Square, airports, and train and bus terminals. However, it would be helpful to know the location that each ID corresponds with. Unfortunately, this is not in the data.

# PACE: Execute

Consider the PACE Strategy Document to reflect on the Execute stage.

## Task 4a. Results and evaluation

Having built visualizations in Python, what have we learned about the dataset? What other questions have the visualizations uncovered that we should pursue?

***Pro tip:*** Put yourself in the client's perspective. What would they want to know?

Use the following code fields to pursue any additional EDA based on the visualizations we've already plotted. Also use the space to make sure the visualizations are clean, easily understandable, and accessible.

***Ask yourself:*** Did we consider color, contrast, emphasis, and labeling?

I have learned .... the highest distribution of trip distances are below 5 miles, but there are outliers all the way out to 35 miles. There are no missing values.

My other questions are .... There are several trips that have a trip distance of "0.0." What might those trips be? Will they impact our model?

My client would likely want to know ... that the data includes dropoff and pickup times. We can use that information to derive a trip duration for each line of data. This would likely be something that will help the client with their model.

In [36]:
```
df['trip_duration'] = (df['tpep_dropoff_datetime']-df['tpep_pickup_datetime'])
```

In [37]:
```
df.head(10)
```

Out[37]:

| LocationID | DOLocationID | ... | fare_amount | extra | mta_tax | tip_amount | tolls_amount | improvement |
|---|---|---|---|---|---|---|---|---|
| 100 | 231 | ... | 13.0 | 0.0 | 0.5 | 2.76 | 0.0 | |
| 186 | 43 | ... | 16.0 | 0.0 | 0.5 | 4.00 | 0.0 | |
| 262 | 236 | ... | 6.5 | 0.0 | 0.5 | 1.45 | 0.0 | |
| 188 | 97 | ... | 20.5 | 0.0 | 0.5 | 6.39 | 0.0 | |
| 4 | 112 | ... | 16.5 | 0.5 | 0.5 | 0.00 | 0.0 | |
| 161 | 236 | ... | 9.0 | 0.5 | 0.5 | 2.06 | 0.0 | |
| 79 | 241 | ... | 47.5 | 1.0 | 0.5 | 9.86 | 0.0 | |
| 237 | 114 | ... | 16.0 | 1.0 | 0.5 | 1.78 | 0.0 | |
| 234 | 249 | ... | 9.0 | 0.0 | 0.5 | 0.00 | 0.0 | |
| 239 | 237 | ... | 13.0 | 0.0 | 0.5 | 2.75 | 0.0 | |

## Task 4b. Conclusion

You have visualized the data we need to share with the director now. Remember, the goal of a data visualization is for an audience member to glean the information on the chart in mere seconds.

*Questions to ask ourself for reflection:* Why is it important to conduct Exploratory Data Analysis? Why are the data visualizations provided in this notebook useful?

**Exemplar response:**

EDA is important because ...

- *EDA helps a data professional to get to know the data, understand its outliers, clean its missing values, and prepare it for future modeling.*

Visualizations helped me understand ..

- *That this dataset has some outliers that we will need to make decisions on prior to designing a model.*