Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Part 3: NYC Accidents Data

In the real world, data isn't always nicely bundled in one file; data can be sourced from many places with many formats. Now we will use NYC accident data to try to improve our set of features.

In this part of the project, you'll do some EDA over the combined data set. We'll do a lot of the coding work for you, but there will be a few coding subtasks for you to complete on your own, as well as many results to interpret.

Note

If your kernel dies unexpectedly, make sure you have shutdown all other notebooks. Each notebook uses valuable memory which we will need for this part of the project.

Imports

Let us start by loading the Python libraries and custom tools we will use in this part.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile
import os
from pathlib import Path

sns.set(style="whitegrid", palette="muted")

plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 12

%matplotlib inline
```

Downloading the Data

We will use the `fetch_and_cache` utility to download the dataset.

In [3]:
```
# Download and cache urls and get the file objects.
from utils import fetch_and_cache
data_url = 'https://github.com/DS-100/fa18/raw/gh-pages/assets/datasets/collisions.zip'
file_name = 'collisions.zip'
dest_path = fetch_and_cache(data_url=data_url, file=file_name)

print(f'Located at {dest_path}')
```

```
Using version already downloaded: Sat Dec  1 01:41:10 2018
MD5 hash of file: a445b925d24f319cb60bd3ace6e4172b
Located at data/collisions.zip
```

We will store the taxi data locally before loading it.

In [4]:
```
collisions_zip = zipfile.ZipFile(dest_path, 'r')

#Extract zip files
collisions_dir = Path('data/collisions')
collisions_zip.extractall(collisions_dir)
```

Loading and Formatting Data

The following code loads the collisions data into a Pandas DataFrame.

In [5]:
```
# Run this cell to load the collisions data.
skiprows = None
collisions = pd.read_csv(collisions_dir/'collisions_2016.csv', index_col='UNIQUE KEY',
                         parse_dates={'DATETIME':["DATE","TIME"]}, skiprows=skiprows)
collisions['TIME'] = pd.to_datetime(collisions['DATETIME']).dt.hour
collisions['DATE'] = pd.to_datetime(collisions['DATETIME']).dt.date
collisions = collisions.dropna(subset=['LATITUDE', 'LONGITUDE'])
collisions = collisions[collisions['LATITUDE'] <= 40.85]
collisions = collisions[collisions['LATITUDE'] >= 40.63]
collisions = collisions[collisions['LONGITUDE'] <= -73.65]
collisions = collisions[collisions['LONGITUDE'] >= -74.03]
collisions.info()
```

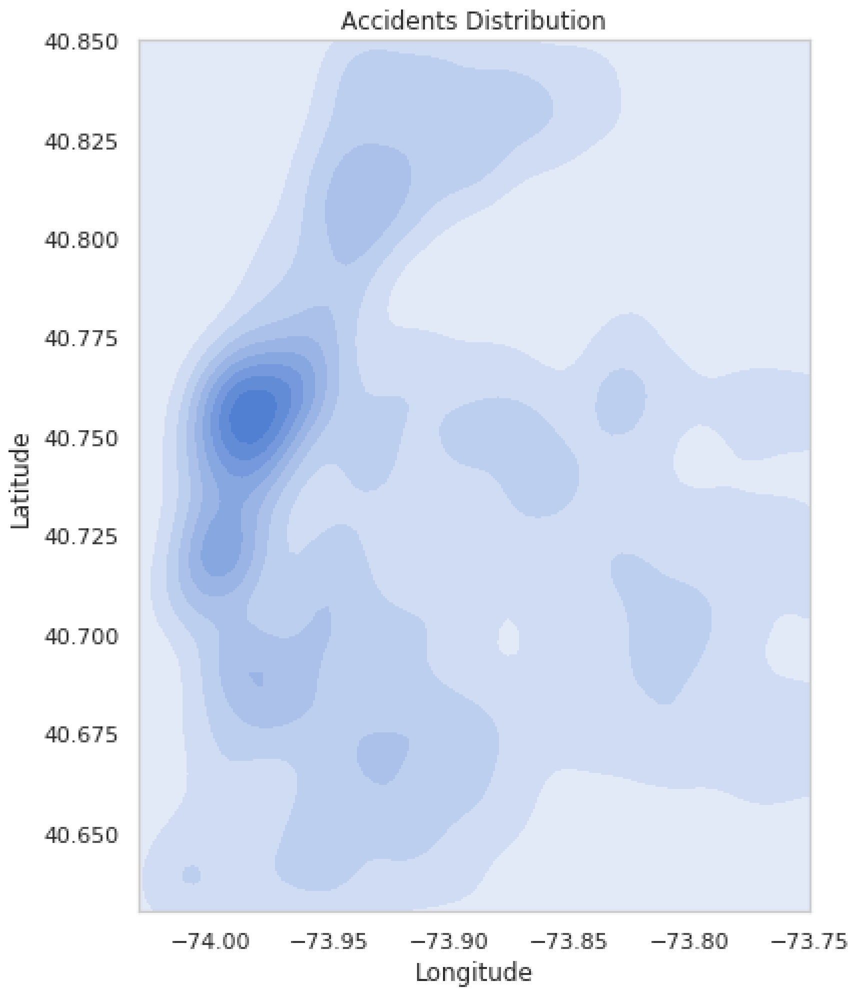
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 116691 entries, 3589202 to 3363795
Data columns (total 30 columns):
DATETIME                         116691 non-null datetime64[ns]
Unnamed: 0                       116691 non-null int64
BOROUGH                          100532 non-null object
ZIP CODE                         100513 non-null float64
LATITUDE                         116691 non-null float64
LONGITUDE                        116691 non-null float64
LOCATION                         116691 non-null object
ON STREET NAME                   95914 non-null object
CROSS STREET NAME                95757 non-null object
OFF STREET NAME                  61545 non-null object
NUMBER OF PERSONS INJURED        116691 non-null int64
NUMBER OF PERSONS KILLED         116691 non-null int64
NUMBER OF PEDESTRIANS INJURED    116691 non-null int64
NUMBER OF PEDESTRIANS KILLED     116691 non-null int64
NUMBER OF CYCLIST INJURED        116691 non-null int64
NUMBER OF CYCLIST KILLED         116691 non-null int64
```

1: EDA of Accidents

Let's start by plotting the latitude and longitude where accidents occur. This may give us some insight on taxi ride durations. We sample N times (given) from the collisions dataset and create a 2D KDE plot of the longitude and latitude. We make sure to set the x and y limits according to the boundaries of New York, given below.

Here is a [map of Manhattan (https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488)](https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488) for your convenience.

In [6]:
```
# Plot lat/lon of accidents, will take a few seconds
N = 20000
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)

sample = collisions.sample(N)
plt.figure(figsize=(6,8))
sns.kdeplot(sample["LONGITUDE"], sample["LATITUDE"], shade=True)
plt.xlim(city_long_border)
plt.ylim(city_lat_border)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Accidents Distribution")
plt.show();
```



Question 1a

What can you say about the location density of NYC collisions based on the plot above?

Hint: Here is a [page (https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488)](https://www.google.com/maps/place/Manhattan,+New+York,+NY/@40.7590402,-74.0394431,12z/data=!3m1!4b1!4m5!3m4!1s0x89c2588f046ee661:0xa0b3281fcecc08c!8m2!3d40.7830603!4d-73.9712488) that may be useful, and [another page (https://www.6sqft.com/what-nycs-population-looks-like-day-vs-night/)](https://www.6sqft.com/what-nycs-population-looks-like-day-vs-night/) that may be useful.

In [7]:
```
q1a_answer = r"""
Collisions happen most frequently in Midtown of Manhattan. And the closer the locations are to Midtown, the larger
the frequency of colissions there is likely to be.

Brooklyn and Queens have a much smaller frequency of collisions than Manhattan.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1a_answer)
```

```
Collisions happen most frequently in Midtown of Manhattan. And the closer the locations are to Midtown, the larger
the frequency of colissions there is likely to be.

Brooklyn and Queens have a much smaller frequency of collisions than Manhattan.
```

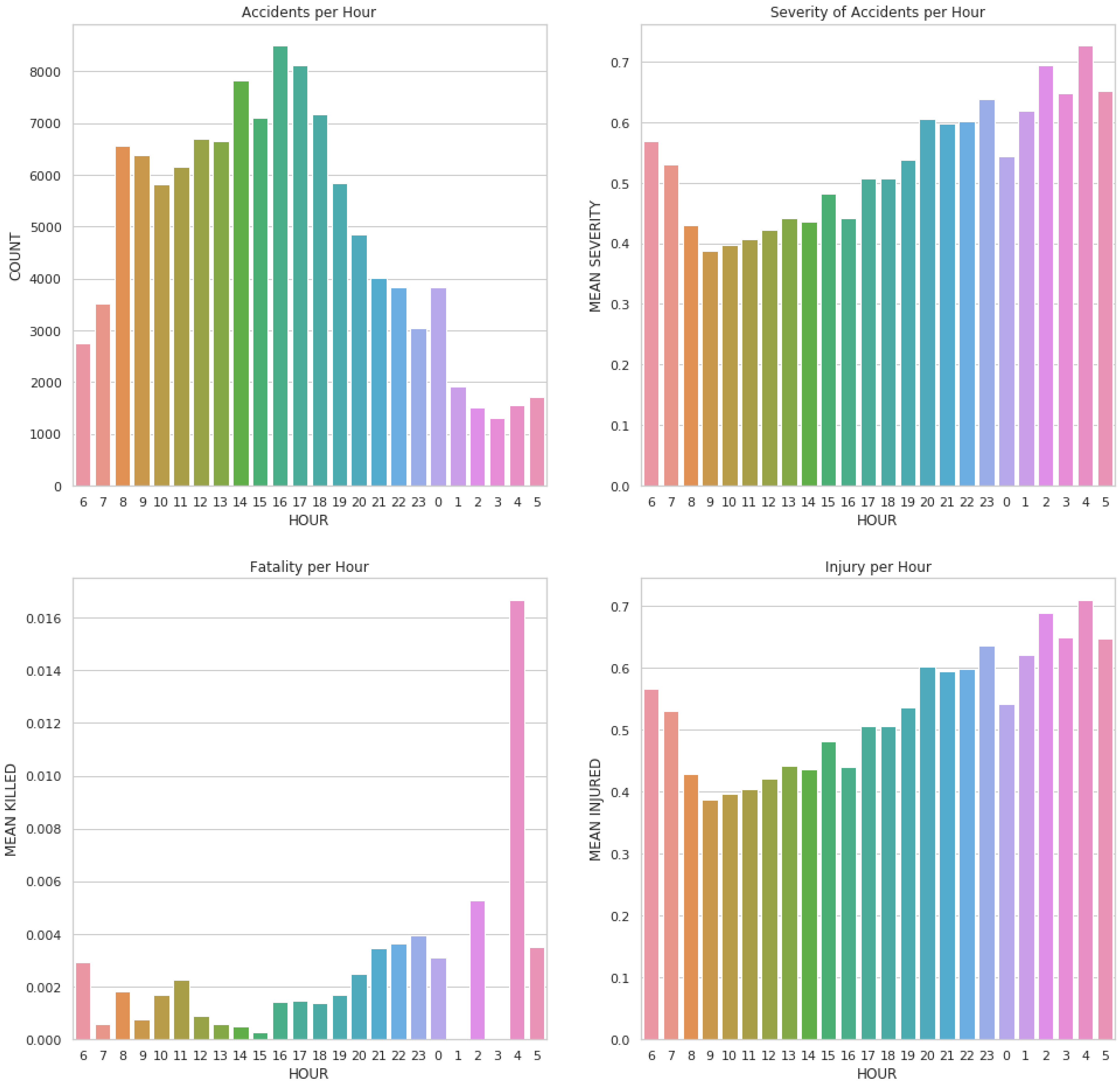
We see that an entry in accidents contains information on number of people injured/killed. Instead of using each of these columns separately, let's combine them into one column called `'SEVERITY'`. Let's also make columns `FATALITY` and `INJURY`, each aggregating the fatalities and injuries respectively.

In [8]:
```
collisions['SEVERITY'] = collisions.filter(regex=r'NUMBER OF *').sum(axis=1)
collisions['FATALITY'] = collisions.filter(regex=r'KILLED').sum(axis=1)
collisions['INJURY'] = collisions.filter(regex=r'INJURED').sum(axis=1)
```

Now let's group by time and compare two aggregations: count vs mean. Below we plot the number of collisions and the mean severity of collisions by the hour, i.e. the `TIME` column. We visualize them side by side and set the start of our day to be 6 a.m.

Let's also take a look at the mean number of casualties per hour and the mean number of injuries per hour, plotted below.

In [9]:
```python
fig, axes = plt.subplots(2, 2, figsize=(16,16))
order = np.roll(np.arange(24), -6)
ax1 = axes[0,0]
ax2 = axes[0,1]
ax3 = axes[1,0]
ax4 = axes[1,1]

collisions_count = collisions.groupby('TIME').count()
collisions_count = collisions_count.reset_index()
sns.barplot(x='TIME', y='SEVERITY', data=collisions_count, order=order, ax=ax1)
ax1.set_title("Accidents per Hour")
ax1.set_xlabel("HOUR")
ax1.set_ylabel('COUNT')


collisions_mean = collisions.groupby('TIME').mean()
collisions_mean = collisions_mean.reset_index()
sns.barplot(x='TIME', y='SEVERITY', data=collisions_mean, order=order, ax=ax2)
ax2.set_title("Severity of Accidents per Hour")
ax2.set_xlabel("HOUR")
ax2.set_ylabel('MEAN SEVERITY')

fatality_count = collisions.groupby('TIME').mean()
fatality_count = fatality_count.reset_index()
sns.barplot(x='TIME', y='FATALITY', data=fatality_count, order=order, ax=ax3)
ax3.set_title("Fatality per Hour")
ax3.set_xlabel("HOUR")
ax3.set_ylabel('MEAN KILLED')

injury_count = collisions.groupby('TIME').mean()
injury_count = injury_count.reset_index()
sns.barplot(x='TIME', y='INJURY', data=injury_count, order=order, ax=ax4)
ax4.set_title("Injury per Hour")
ax4.set_xlabel("HOUR")
ax4.set_ylabel('MEAN INJURED')

plt.show();
```



Question 1b

Based on the visualizations above, what can you say about each? Make a comparison between the accidents per hour vs the mean severity per hour. What about the number of fatalities per hour vs the number of injuries per hour? Why do we chose to have our hours start at 6 as opposed to 0?

In [10]:

```
q1b_answer = r"""
1.
(1).In the plot of accidents per hour, we can see that accidents happen more frequently in daytime than nighttime,
and accidents happen most frequently in rush hours, especially rush hours off work.
(2)In the plot of severity of accidents per hour, the accidents happen in the nighttime more severe those in the
daytime.
(3) In the plot of Fatality per hour, the mean killed numbers in nighttime are larger than that in daytime. And
meam killed number are extremy high at 4am
(4) In the plot of Injury per hour, the mean killed numbers in nighttime are larger than that in daytime.

2.When the number of the accidents per hour is large, the mean severity is small, and vice versa.

3.When the number of fatalities per hour is large, the number of injuries is also large, and vice versa.

4.

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1b_answer)
```

```
1.
(1).In the plot of accidents per hour, we can see that accidents happen more frequently in daytime than nighttime,
and accidents happen most frequently in rush hours, especially rush hours off work.
(2)In the plot of severity of accidents per hour, the accidents happen in the nighttime more severe those in the
daytime.
(3) In the plot of Fatality per hour, the mean killed numbers in nighttime are larger than that in daytime. And
meam killed number are extremy high at 4am
(4) In the plot of Injury per hour, the mean killed numbers in nighttime are larger than that in daytime.

2.When the number of the accidents per hour is large, the mean severity is small, and vice versa.

3.When the number of fatalities per hour is large, the number of injuries is also large, and vice versa.

4.
```

Let's also check the relationship between location and severity. We provide code to visualize a heat map of collisions, where the x and y coordinate are the location of the collision and the heat color is the severity of the collision. Again, we sample N points to speed up visualization.

In [11]:

```
N = 10000
sample = collisions.sample(N)

# Round / bin the latitude and longitudes
sample['lat_bin'] = np.round(sample['LATITUDE'], 3)
sample['lng_bin'] = np.round(sample['LONGITUDE'], 3)

# Average severity for regions
gby_cols = ['lat_bin', 'lng_bin']

coord_stats = (sample.groupby(gby_cols)
               .agg({'SEVERITY': 'mean'})
               .reset_index())

# Visualize the average severity per region
city_long_border = (-74.03, -73.75)
city_lat_border = (40.63, 40.85)
fig, ax = plt.subplots(ncols=1, nrows=1, figsize=(14, 10))

scatter_trips = ax.scatter(sample['LONGITUDE'].values,
                           sample['LATITUDE'].values,
                           color='grey', s=1, alpha=0.5)

scatter_cmap = ax.scatter(coord_stats['lng_bin'].values,
                          coord_stats['lat_bin'].values,
                          c=coord_stats['SEVERITY'].values,
                          cmap='viridis', s=10, alpha=0.9)

cbar = fig.colorbar(scatter_cmap)
cbar.set_label("Manhattan average severity")
ax.set_xlim(city_long_border)
ax.set_ylim(city_lat_border)
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
plt.title('Heatmap of Manhattan average severity')
plt.axis('off');
```

Question 1c

Do you think the location of the accident has a significant impact on the severity based on the visualization above? Additionally, identify something that could be improved in the plot above and describe how we could improve it.

In [12]:
```
q1c_answer = r"""
I don't think the location of the accident has a significant impact on the severity.

We could drop some outliers and decrese the range of severity, so that we can make the color easier to
distinguish.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1c_answer)
```

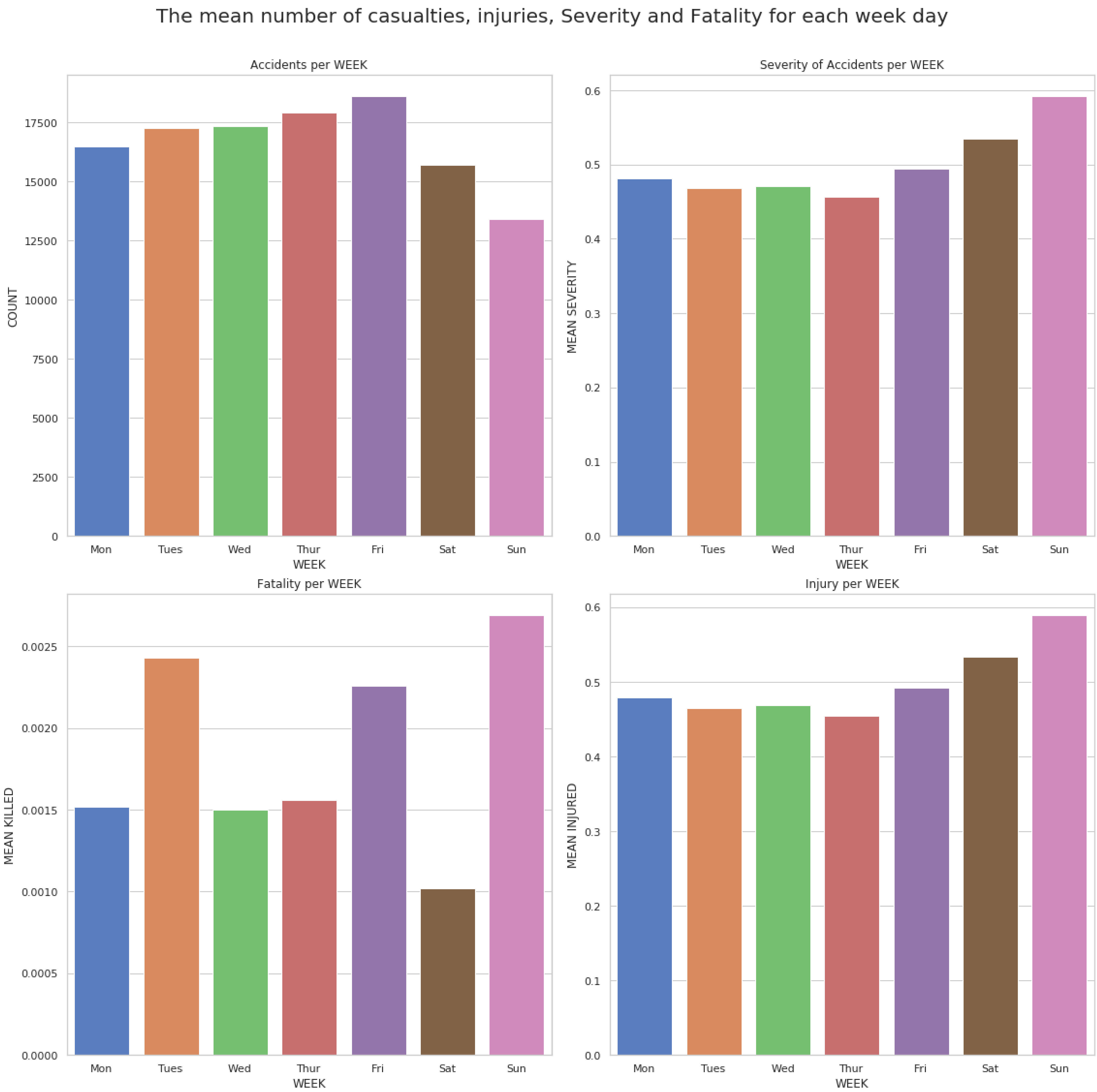
```
I don't think the location of the accident has a significant impact on the severity.

We could drop some outliers and decrese the range of severity, so that we can make the color easier to
distinguish.
```

Question 1d

Create a plot to visualize one or more features of the `collisions` table.

In [13]:

```python
#the mean number of casualties, injuries, Severity and Fatality for each week day

fig, axes = plt.subplots(2, 2, figsize=(16,16))
weekday = ['Mon', 'Tues', 'Wed', 'Thur','Fri','Sat', 'Sun']
ax1 = axes[0,0]
ax2 = axes[0,1]
ax3 = axes[1,0]
ax4 = axes[1,1]

collisions['WEEK'] = collisions['DATETIME'].dt.weekday

plt.title('the mean number of casualties, injuries, Severity and Fatality for each week day')
collisions_count = collisions.groupby('WEEK').count()
collisions_count = collisions_count.reset_index()
sns.barplot(x='WEEK', y='SEVERITY', data=collisions_count,  ax=ax1)
plt.sca(ax1)
plt.xticks(range(7), weekday)
ax1.set_title("Accidents per WEEK")
ax1.set_xlabel("WEEK")
ax1.set_ylabel('COUNT')


collisions_mean = collisions.groupby('WEEK').mean()
collisions_mean = collisions_mean.reset_index()
sns.barplot(x='WEEK', y='SEVERITY', data=collisions_mean, ax=ax2)
plt.sca(ax2)
plt.xticks(range(7), weekday)
ax2.set_title("Severity of Accidents per WEEK")
ax2.set_xlabel("WEEK")
ax2.set_ylabel('MEAN SEVERITY')

fatality_count = collisions.groupby('WEEK').mean()
fatality_count = fatality_count.reset_index()
sns.barplot(x='WEEK', y='FATALITY', data=fatality_count, ax=ax3)
plt.sca(ax3)
plt.xticks(range(7), weekday)
ax3.set_title("Fatality per WEEK")
ax3.set_xlabel("WEEK")
ax3.set_ylabel('MEAN KILLED')

injury_count = collisions.groupby('WEEK').mean()
injury_count = injury_count.reset_index()
sns.barplot(x='WEEK', y='INJURY', data=injury_count, ax=ax4)
plt.sca(ax4)
plt.xticks(range(7), weekday)
ax4.set_title("Injury per WEEK")
ax4.set_xlabel("WEEK")
ax4.set_ylabel('MEAN INJURED')

fig.tight_layout()
plt.suptitle("The mean number of casualties, injuries, Severity and Fatality for each week day",size = 20)
fig.subplots_adjust(top = 0.92)
plt.show();
collisions = collisions.drop(columns=['WEEK'])
```



Question 1e

Answer the following questions regarding your plot in 1d.

1. What feature you're visualization
2. Why you chose this feature
3. Why you chose this visualization method

In [14]:
```
q1e_answer = r"""
(1)I visualized the mean number of accidents, injuries, Severity and Fatality for each week day
(2)I want to show if the number of accidents, injuries, Severity and fatality behave differently in Weekdays and weekends
(3)I chose barplot because there are only 7 days in a week, and I can list the data for each weekday. And it is easier to make compasion with the barplot.
"""
# YOUR CODE HERE
#raise NotImplementedError()
print(q1e_answer)
```

```
(1)I visualized the mean number of accidents, injuries, Severity and Fatality for each week day
(2)I want to show if the number of accidents, injuries, Severity and fatality behave differently in Weekdays and
weekends
(3)I chose barplot because there are only 7 days in a week, and I can list the data for each weekday. And it is
easier to make compasion with the barplot.
```

2: Combining External Datasets

It seems like accident timing and location may influence the duration of a taxi ride. Let's start to join our NYC Taxi data with our collisions data.

Let's assume that an accident will influence traffic in the surrounding area for around 1 hour. Below, we create two columns, `START` and `END` :

- `START` : contains the recorded time of the accident
- `END` : 1 hours after `START`

Note: We chose 1 hour somewhat arbitrarily, feel free to experiment with other time intervals outside this notebook.

In [15]:
```
collisions['START'] = collisions['DATETIME']
collisions['END'] = collisions['START'] + pd.Timedelta(hours=1)
```

Question 2a

Drop all of the columns besides the following: `DATETIME` , `TIME` , `START` , `END` , `DATE` , `LATITUDE` , `LONGITUDE` , `SEVERITY` . Feel free to experiment with other subsets outside of this notebook.

In [16]:
```
remaining_col = ['DATETIME', 'TIME', 'START', 'END', 'DATE', 'LATITUDE', 'LONGITUDE', 'SEVERITY']
collisions_subset = collisions.drop(columns= [col for col in collisions.columns if col not in remaining_col])
# YOUR CODE HERE
#raise NotImplementedError()
collisions_subset.head(5)
```

Out[16]:

	DATETIME	LATITUDE	LONGITUDE	TIME	DATE	SEVERITY	START	END
UNIQUE KEY								
3589202	2016-12-29 00:00:00	40.844107	-73.897997	0	2016-12-29	0	2016-12-29 00:00:00	2016-12-29 01:00:00
3587413	2016-12-26 14:30:00	40.692347	-73.881778	14	2016-12-26	0	2016-12-26 14:30:00	2016-12-26 15:30:00
3578151	2016-11-30 22:50:00	40.755480	-73.741730	22	2016-11-30	2	2016-11-30 22:50:00	2016-11-30 23:50:00
3567096	2016-11-23 20:11:00	40.771122	-73.869635	20	2016-11-23	0	2016-11-23 20:11:00	2016-11-23 21:11:00
3565211	2016-11-21 14:11:00	40.828918	-73.838403	14	2016-11-21	0	2016-11-21 14:11:00	2016-11-21 15:11:00

In [17]:
```
assert collisions_subset.shape == (116691, 8)
```

Question 2b

Now, let's merge our `collisions_subset` table with `train_df` . Start by merging with only the date. We will filter by a time window in a later question.

We should be performing a left join, where our `train_df` is the left table. This is because we want to preserve all of the taxi rides in our end result. It happens that an inner join will also work, since both tables contain data on each date.

Note that the resulting `merged` table will have multiple rows for every taxi ride row in the original `train_df` table. For example, `merged` will have 483 rows with `index` equal to 16709, because there were 483 accidents that occurred on the same date as ride #16709.

Because of memory limitation, we will select the third week of 2016 to analyze. Feel free to change to it week 1 or 2 to see if the observation is general.

In [18]:
```
data_file = Path("./", "cleaned_data.hdf")
train_df = pd.read_hdf(data_file, "train")
train_df = train_df.reset_index()
train_df = train_df[['index', 'tpep_pickup_datetime', 'pickup_longitude', 'pickup_latitude', 'duration']]
train_df['date'] = train_df['tpep_pickup_datetime'].dt.date
```

In [19]:
```
collisions_subset = collisions_subset[collisions_subset['DATETIME'].dt.weekofyear == 3]
train_df = train_df[train_df['tpep_pickup_datetime'].dt.weekofyear == 3]
```

In [20]:
```
# merge the dataframe here
merged = pd.merge(train_df,collisions_subset, how= 'left', left_on='date',right_on = 'DATE')

# YOUR CODE HERE
#raise NotImplementedError()

merged.head()
```

Out[20]:

	index	tpep_pickup_datetime	pickup_longitude	pickup_latitude	duration	date	DATETIME	LATITUDE	LONGITUDE	TIME	DATE	SEVERITY	START	END
0	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 10:35:00	40.701651	-73.991484	10	2016-01-21	0	2016-01-21 10:35:00	2016-01-21 11:35:00
1	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 13:20:00	40.704760	-74.014961	13	2016-01-21	0	2016-01-21 13:20:00	2016-01-21 14:20:00
2	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 16:00:00	40.732891	-73.920574	16	2016-01-21	4	2016-01-21 16:00:00	2016-01-21 17:00:00
3	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 18:30:00	40.714122	-73.831508	18	2016-01-21	0	2016-01-21 18:30:00	2016-01-21 19:30:00
4	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 00:05:00	40.700108	-73.953819	0	2016-01-21	0	2016-01-21 00:05:00	2016-01-21 01:05:00

In [21]:
```
assert merged.shape == (1528162, 14)
```

Question 2c

Now that our tables are merged, let's use temporal and spatial proximity to condition on the duration of the average length of a taxi ride. Let's operate under the following assumptions.

Accidents only influence the duration of a taxi ride if the following are satisfied:

1) The haversine distance between the the pickup location of the taxi ride and location of the recorded accident is within 5 (km). This is roughly 3.1 miles.

2) The start time of a taxi ride is within a 1 hour interval between the start and end of an accident.

Complete the code below to create an `'accident_close'` column in the `merged` table that indicates if an accident was close or not according to the assumptions above.

In [22]:
```
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Compute Manhattan distance
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b
```

In [23]:
```
start_to_accident = haversine(merged['pickup_latitude'].values,
                              merged['pickup_longitude'].values,
                              merged['LATITUDE'].values,
                              merged['LONGITUDE'].values)
merged['start_to_accident'] = start_to_accident

# initialze accident_close column to all 0 first
merged['accident_close'] = 0

# Boolean pd.Series to select the indices for which accident_close should equal 1:
# (1) record's start_to_accident <= 5
# (2) pick up time is between start and end
is_accident_close = merged[(merged['tpep_pickup_datetime'] >= merged['START']) &
                           (merged['tpep_pickup_datetime'] <= merged['END']) &
                           (merged['start_to_accident'] <= 5)].index

# YOUR CODE HERE
#raise NotImplementedError()

merged.loc[is_accident_close, 'accident_close'] = 1

```

In [24]:
```
assert merged['accident_close'].sum() > 16000
```

The last step is to aggregate the total number of proximal accidents. We want to count the total number of accidents that were close spatially and temporally and condition on that data.

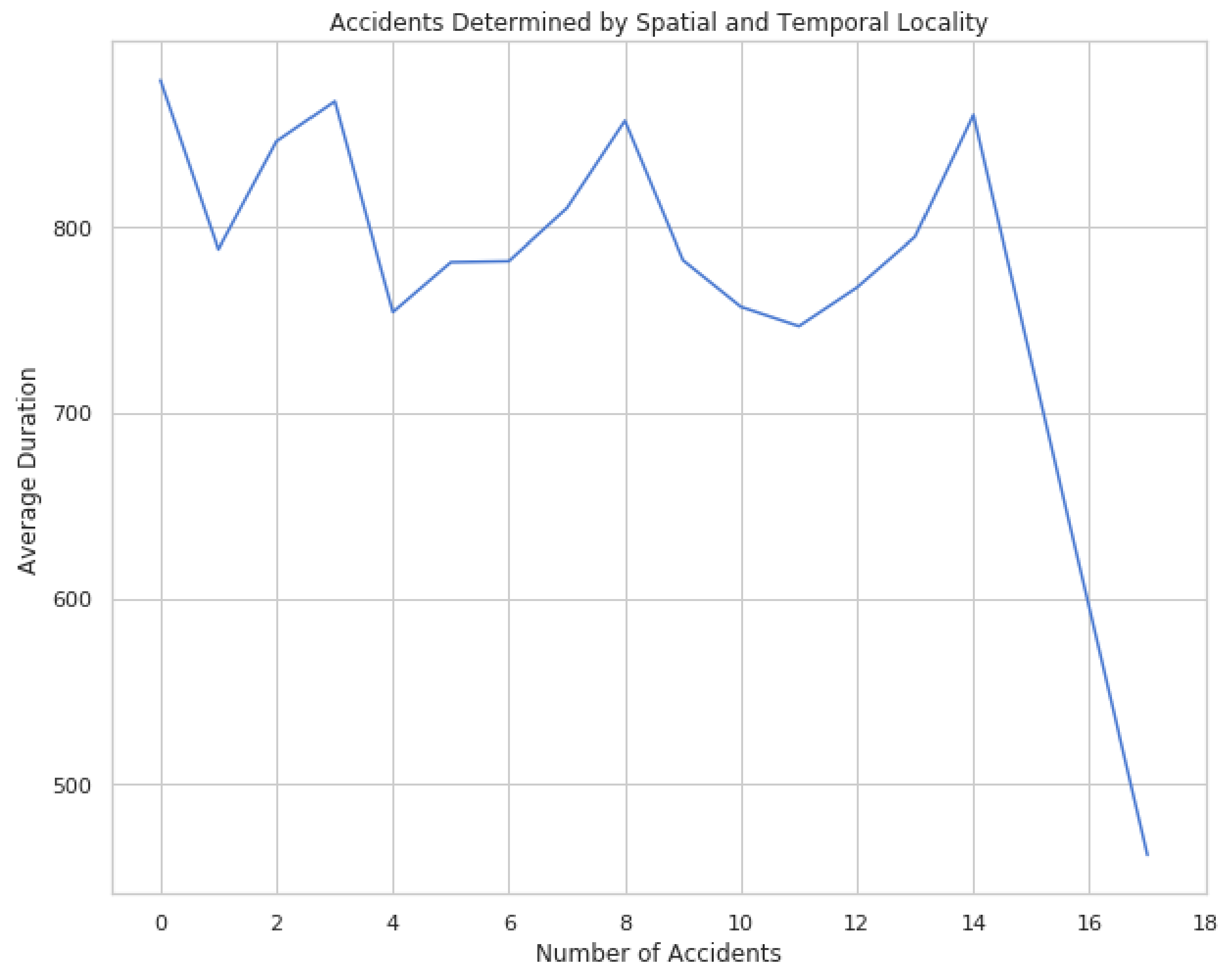
The code below create a new data frame called `train_accidents` , which is a copy of `train_df` , but with a new column that counts the number of accidents that were close (spatially and temporally) to the pickup location/time.

In [25]:
```
train_df = train_df.set_index('index')
num_accidents = merged.groupby(['index'])['accident_close'].sum().to_frame()
train_accidents = train_df.copy()
train_accidents['num_accidents'] = num_accidents
```

Next, for each value of `num_accidents` , we plot the average `duration` of rides with that number of accidents.

In [26]:

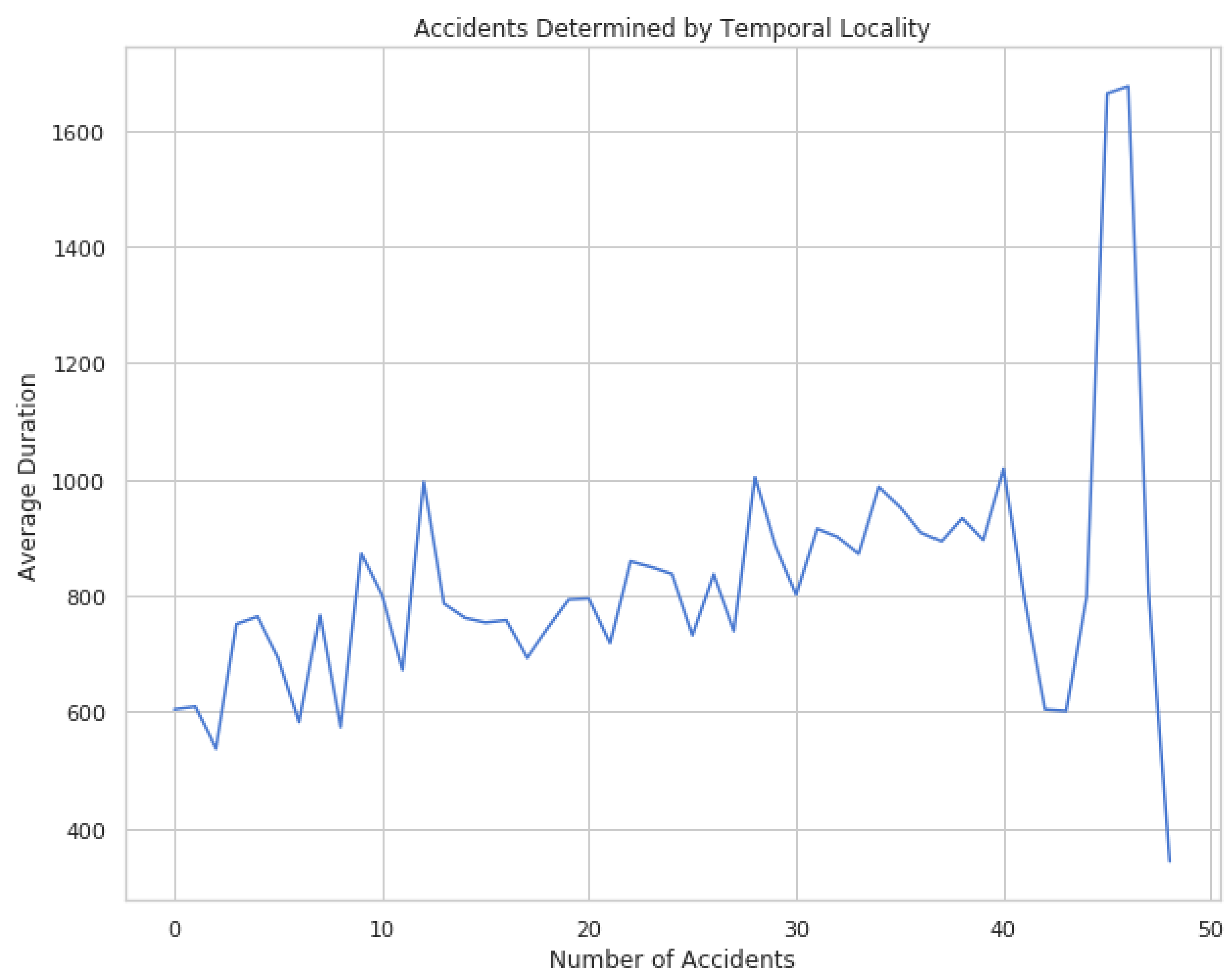
```python
plt.figure(figsize=(10,8))
train_accidents.groupby('num_accidents')['duration'].mean().plot(xticks=np.arange(0, 20, 2))
plt.title("Accidents Determined by Spatial and Temporal Locality")
plt.xlabel("Number of Accidents")
plt.ylabel("Average Duration")
plt.show();
```



It seems that using both spatial and temporal proximity doesn't give us much insight on if collisions increase taxi ride durations. Let's try conditioning on spatial proximity and temporal proximity separately and see if there are more interesting results there.

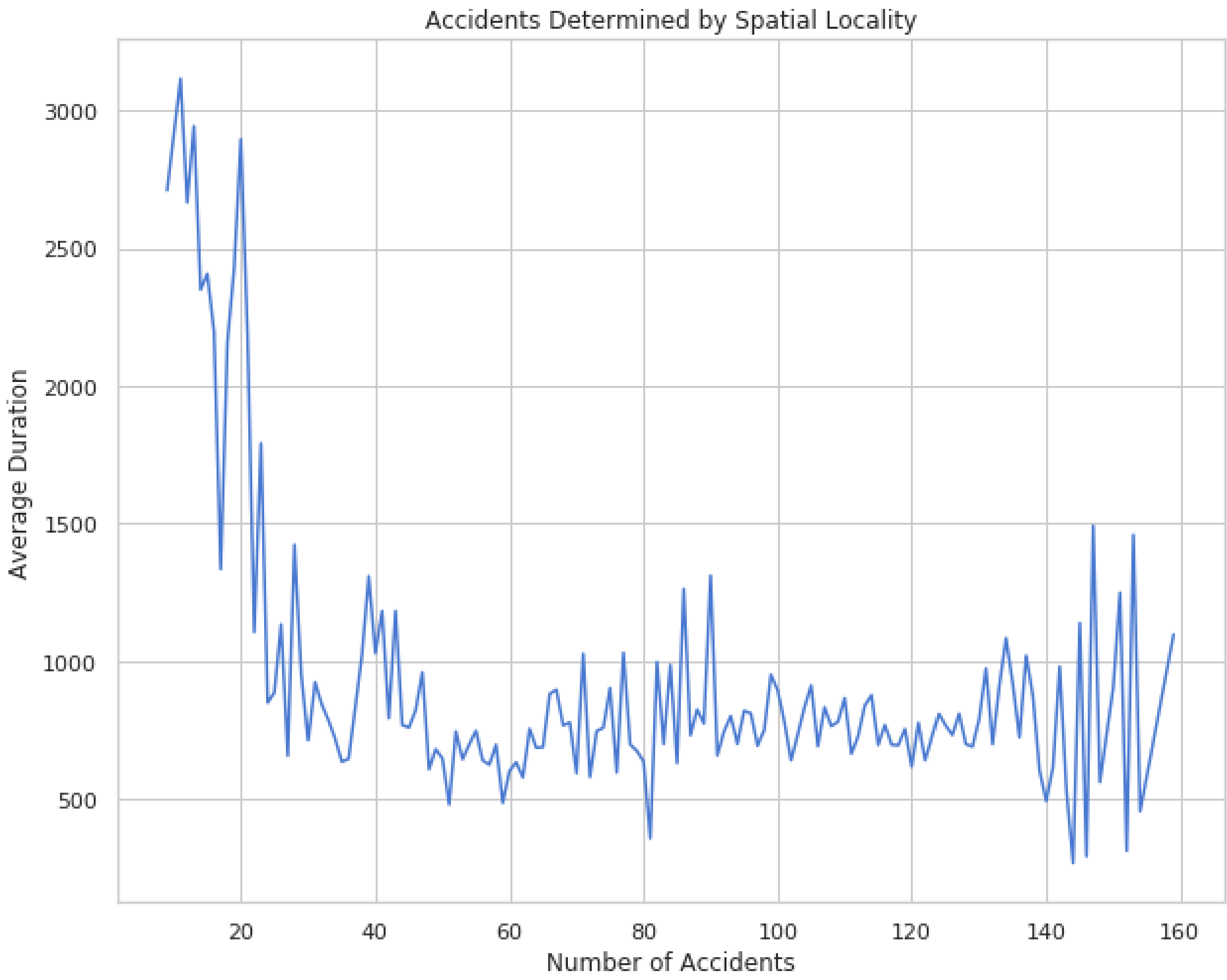
In [27]:

```python
# Temporal locality

# Condition on time
index = (((merged['tpep_pickup_datetime'] >= merged['START']) & \
         (merged['tpep_pickup_datetime'] <= merged['END'])))

# Count accidents
merged['accident_close'] = 0
merged.loc[index, 'accident_close'] = 1
num_accidents = merged.groupby(['index'])['accident_close'].sum().to_frame()
train_accidents_temporal = train_df.copy()
train_accidents_temporal['num_accidents'] = num_accidents

# Plot
plt.figure(figsize=(10,8))
train_accidents_temporal.groupby('num_accidents')['duration'].mean().plot()
plt.title("Accidents Determined by Temporal Locality")
plt.xlabel("Number of Accidents")
plt.ylabel("Average Duration")
plt.show();
```



In [28]:

```
# Spatial locality

# Condition on space
index = (merged['start_to_accident'] <= 5)

# Count accidents
merged['accident_close'] = 0
merged.loc[index, 'accident_close'] = 1
num_accidents = merged.groupby(['index'])['accident_close'].sum().to_frame()
train_accidents_spatial = train_df.copy()
train_accidents_spatial['num_accidents'] = num_accidents

# Plot
plt.figure(figsize=(10,8))
train_accidents_spatial.groupby('num_accidents')['duration'].mean().plot()
plt.title("Accidents Determined by Spatial Locality")
plt.xlabel("Number of Accidents")
plt.ylabel("Average Duration")
plt.show();
```



Question 2d

By conditioning on temporal and spatial proximity separately, we reveal different trends in average ride duration as a function of number of accidents nearby.

What can you say about the temporal and spatial proximity of accidents to taxi rides and the effect on ride duration? Think of a new hypothesis regarding accidents and taxi ride durations and explain how you would test it.

Additionally, comment on some of the assumptions being made when we condition on temporal and spatial proximity separately. What are the implications of only considering one and not the other?

In [29]:

```
merged
```

Out[29]:

	index	tpep_pickup_datetime	pickup_longitude	pickup_latitude	duration	date	DATETIME	LATITUDE	LONGITUDE	TIME	DATE	SEVERITY	START	END	start_to_accident	accident_close
0	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 10:35:00	40.701651	-73.991484	10	2016-01-21	0	2016-01-21 10:35:00	2016-01-21 11:35:00	4.433256	1
1	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 13:20:00	40.704760	-74.014961	13	2016-01-21	0	2016-01-21 13:20:00	2016-01-21 14:20:00	4.298554	1
2	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 16:00:00	40.732891	-73.920574	16	2016-01-21	4	2016-01-21 16:00:00	2016-01-21 17:00:00	6.587580	0
3	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 18:30:00	40.714122	-73.831508	18	2016-01-21	0	2016-01-21 18:30:00	2016-01-21 19:30:00	14.348166	0
4	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 00:05:00	40.700108	-73.953819	0	2016-01-21	0	2016-01-21 00:05:00	2016-01-21 01:05:00	5.894669	0
5	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 00:05:00	40.663972	-73.997766	0	2016-01-21	0	2016-01-21 00:05:00	2016-01-21 01:05:00	8.589078	0
6	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 00:05:00	40.663972	-73.997766	0	2016-01-21	0	2016-01-21 00:05:00	2016-01-21 01:05:00	8.589078	0

In [30]:
```python
q2d_answer = r"""
1.
(1)
For the accidents determined by temporal and spatial proximity, when number of such accidents increases, the
average duration decreases. And this conclusion doesn't comply with our experience, because accidents wiil result
in traffic jam, which will enlarge the mean durration for the rides that are temporal and spatial proximity.
(2)
My hypothsis: The mean severity of accidents determined by temporal and spatial proximity will affect the
duration. We can use the similiar visualization above to show the relationship between the mean severity of
accidents and the duration.

2.
(1)
For the accidents only determined by temporal proximity, the average duration increase as the number of such
accidents increases.
For the accidents only determined by spatial proximity, the average duration decreases as the number of such
accidents increases, and there is a obvious drop for the average duration when the number is around 30.
(2)
We should only consider temporal proximity, because the location will affect the avarage duration. In some
areas like Manhattan, the duration tends to be small. So spatial proximity will interference our test.
"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q2d_answer)
```


```
1.
(1)
For the accidents determined by temporal and spatial proximity, when number of such accidents increases, the
average duration decreases. And this conclusion doesn't comply with our experience, because accidents wiil result
in traffic jam, which will enlarge the mean durration for the rides that are temporal and spatial proximity.
(2)
My hypothsis: The mean severity of accidents determined by temporal and spatial proximity will affect the
duration. We can use the similiar visualization above to show the relationship between the mean severity of
accidents and the duration.

2.
(1)
For the accidents only determined by temporal proximity, the average duration increase as the number of such
accidents increases.
For the accidents only determined by spatial proximity, the average duration decreases as the number of such
accidents increases, and there is a obvious drop for the average duration when the number is around 30.
(2)
We should only consider temporal proximity, because the location will affect the avarage duration. In some
areas like Manhattan, the duration tends to be small. So spatial proximity will interference our test.
```


Part 3 Exports

We are not requiring you to export anything from this notebook, but you may find it useful to do so. There is a space below for you to export anything you wish.

In [31]:
```python
Path("data/part3").mkdir(parents=True, exist_ok=True)
data_file = Path("data/part3", "data_part3.hdf") # Path of hdf file
...
```

Out[31]: Ellipsis

Part 3 Conclusions

We merged the NYC Accidents dataset with our NYC Taxi dataset, conditioning on temporal and spatial locality. We explored potential features by visualizing the relationship between number of accidents and the average duration of a ride.

Please proceed to part 4 where we will be engineering more features and building our models using a processing pipeline.

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**
2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**

Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→ Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Part 4: Feature Engineering and Model Fitting

In this final part of the project, you will finally build a regression model that attempts to predict the duration of a taxi ride from all other available information.

You will build this model using a processing pipeline and submit your results to Kaggle. We will first walk you through a generic example using the data we saved from Part 1. Please carefully follow these steps as you will need to repeat this for your final model. After, we give you free reign and let you decide how you want to define your final model.

In [2]:
```
import os
import pandas as pd
import numpy as np
import sklearn.linear_model as lm
import matplotlib.pyplot as plt
import seaborn as sns
from pathlib import Path
from sqlalchemy import create_engine
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV

sns.set(style="whitegrid", palette="muted")

plt.rcParams['figure.figsize'] = (12, 9)
plt.rcParams['font.size'] = 12

%matplotlib inline
```

Training and Validation

The following code loads the training and validation data from part 1 into a Pandas DataFrame.

In [3]:
```
# Run this cell to load the data.
data_file = Path("./", "cleaned_data.hdf")
train_df = pd.read_hdf(data_file, "train")
val_df = pd.read_hdf(data_file, "val")
```

In [4]:
```
train_df.head()
```

Out[4]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	...	dropoff_latitude	payment_type	fare_amount	ex
13242	5711100	1	2016-01-17 17:48:41	2016-01-17 17:55:53	1	1.00	-74.006470	40.738766	1	N	...	40.735664	1	6.5	
12723	4989400	1	2016-01-17 01:18:39	2016-01-17 01:21:15	1	0.40	-73.989365	40.763000	1	N	...	40.766121	2	4.0	
8508	2436400	2	2016-01-12 09:07:00	2016-01-12 09:41:17	1	11.40	-73.984108	40.774509	1	N	...	40.770458	1	37.0	
21304	10899100	2	2016-01-29 09:07:54	2016-01-29 09:18:25	1	1.42	-74.002907	40.760262	1	N	...	40.742764	1	8.5	
3817	1319400	1	2016-01-06 11:44:54	2016-01-06 11:49:55	1	0.80	-73.969742	40.760273	1	N	...	40.751129	2	5.0	

5 rows × 21 columns

Testing

Here we load our testing data on which we will evaluate your model.

In [5]:
```
test_df = pd.read_csv("./proj2_test_data.csv")
test_df['tpep_pickup_datetime'] = pd.to_datetime(test_df['tpep_pickup_datetime'])
test_df.head()
```

Out[5]:

	record_id	VendorID	tpep_pickup_datetime	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	store_and_fwd_flag	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax
0	10000	1	2016-01-02 01:45:37	1	1.20	-73.982224	40.768620	1	N	-73.983765	40.779598	1	6.0	0.5	0.5
1	19000	2	2016-01-02 03:05:16	1	10.90	-73.999977	40.738121	1	N	-73.888657	40.824364	1	31.5	0.5	0.5
2	21000	1	2016-01-02 03:24:36	1	1.80	-73.986618	40.747379	1	N	-73.978508	40.729622	1	8.5	0.5	0.5
3	23000	2	2016-01-02 03:47:38	1	5.95	-74.002922	40.744572	1	N	-73.942413	40.786419	1	20.5	0.5	0.5
4	27000	1	2016-01-02 04:36:44	1	1.60	-73.986366	40.759464	1	N	-73.963081	40.760353	2	8.0	0.5	0.5

In [6]:
```
test_df.describe()
```

Out[6]:

	record_id	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	RatecodeID	dropoff_longitude	dropoff_latitude	payment_type	fare_amount	extra	mta_tax	tip_amount
count	1.377400e+04	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000	13774.000000
mean	3.465950e+07	1.536082	1.663642	2.954688	-72.953619	40.187999	1.043778	-73.055577	40.245056	1.340061	12.836930	0.333091	0.497985	1.805420
std	2.015133e+07	0.498714	1.311739	3.704427	8.628431	4.753186	0.877637	8.191366	4.512564	0.490019	10.707619	0.429590	0.036632	2.416784
min	1.000000e+04	1.000000	0.000000	0.000000	-77.039436	0.000000	1.000000	-77.039436	0.000000	1.000000	-93.300000	-0.500000	-0.500000	0.000000
25%	1.719975e+07	1.000000	1.000000	1.000000	-73.992058	40.735166	1.000000	-73.991318	40.734002	1.000000	6.500000	0.000000	0.500000	0.000000
50%	3.457400e+07	2.000000	1.000000	1.700000	-73.981846	40.752432	1.000000	-73.979897	40.753263	1.000000	9.500000	0.000000	0.500000	1.350000
75%	5.216875e+07	2.000000	2.000000	3.157500	-73.967119	40.767264	1.000000	-73.962749	40.768455	2.000000	14.500000	0.500000	0.500000	2.360000
max	6.940400e+07	2.000000	6.000000	104.800000	0.000000	40.868210	99.000000	0.000000	41.540859	4.000000	156.040000	4.500000	1.740000	40.000000

Modeling

We've finally gotten to a point where we can specify a simple model. Remember that we will be fitting our model on the training set we created in part 1. We will use our validation set to evaluate how well our model might perform on future data.

Reusable Pipeline

Throughout this assignment, you should notice that your data flows through a single processing pipeline several times. From a software engineering perspective, this should be sufficient motivation to abstract parts of our code into reusable functions/methods. We will now encapsulate our entire pipeline into a single function `process_data_gm`. gm is shorthand for "guided model".

In [7]:

```
# Copied from part 2
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

# Copied from part 2
def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Compute Manhattan distance
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b

# Copied from part 2
def bearing(lat1, lng1, lat2, lng2):
    """
    Compute the bearing, or angle, from (lat1, lng1) to (lat2, lng2).
    A bearing of 0 refers to a NORTH orientation.
    """
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

# Copied from part 2
def add_time_columns(df):
    """
    Add temporal features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'month'] = df['tpep_pickup_datetime'].dt.month
    df.loc[:, 'week_of_year'] = df['tpep_pickup_datetime'].dt.weekofyear
    df.loc[:, 'day_of_month'] = df['tpep_pickup_datetime'].dt.day
    df.loc[:, 'day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek
    df.loc[:, 'hour'] = df['tpep_pickup_datetime'].dt.hour
    df.loc[:, 'week_hour'] = df['tpep_pickup_datetime'].dt.weekday * 24 + df['hour']
    return df

# Copied from part 2
def add_distance_columns(df):
    """
    Add distance features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'manhattan'] = manhattan_distance(lat1=df['pickup_latitude'],
                                                lng1=df['pickup_longitude'],
                                                lat2=df['dropoff_latitude'],
                                                lng2=df['dropoff_longitude'])

    df.loc[:, 'bearing'] = bearing(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    df.loc[:, 'haversine'] = haversine(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    return df

def select_columns(data, *columns):
    return data.loc[:, columns]
```

In [8]:

```
def process_data_gm1(data, test=False):
    X = (
        data

        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)

        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y
```

We will use our pipeline defined above to pre-process our training and test data in exactly the same way. Our functions make this relatively easy to do!

In [9]:

```
# Train
X_train, y_train = process_data_gm1(train_df)
X_val, y_val = process_data_gm1(val_df)
guided_model_1 = lm.LinearRegression(fit_intercept=True)
guided_model_1.fit(X_train, y_train)

# Predict
y_train_pred = guided_model_1.predict(X_train)
y_val_pred = guided_model_1.predict(X_val)
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

Here, `y_val` are the correct durations for each ride, and `y_val_pred` are the predicted durations based on the 7 features above (`vendorID` , `passenger_count` , `pickup_longitude` , `pickup_latitude` , `dropoff_longitude` , `dropoff_latitude` , `manhattan`).

In [10]:

```
assert 600 <= np.median(y_train_pred) <= 700
assert 600 <= np.median(y_val_pred) <= 700
```

The resulting model really is a linear model just like we saw in class, i.e. the predictions are simply generated by the product $\Phi\theta$. For example, the line of code below generates a prediction for x_1 by computing $\phi_1^T\theta$. Here `guided_model_1.coef_` is θ and `X_train.iloc[0, :]` is ϕ_1.

Note that unlike in class, here the dummy intercept term is not included in Φ.

In [11]:
```
X_train.iloc[0, :].dot(guided_model_1.coef_) + guided_model_1.intercept_
```

Out[11]: 558.751330511368

We see that this prediction is exactly the same (except for possible floating point error) as generated by the `predict` function, which simply computes the product $\Phi\theta$, yielding predictions for every input.

In [12]:
```
y_train_pred[0]
```

Out[12]: 558.75133051135344

In this assignment, we will use Mean Absolute Error (MAE), a.k.a. mean L1 loss, to measure the quality of our models. As a reminder, this quantity is defined as:

$$MAE = \frac{1}{n}\sum_i |y_i - \hat{y}_i|$$

Why may we want to use the MAE as a metric, as opposed to Mean Squared Error (MSE)? Using our domain knowledge that most rides are short in duration (median is roughly 600 seconds), we know that MSE is susceptible to outliers. Given that some of the outliers in our dataset are quite extreme, it is probably better to optimize for the majority of rides rather than for the outliers. You may want to remove some of these outliers later on.

In [13]:
```
def mae(actual, predicted):
    """
    Calculates MAE from actual and predicted values
    Input:
      actual (1D array-like): vector of actual values
      predicted (1D array-like): vector of predicted/fitted values
    Output:
      a float, the MAE
    """

    mae = np.mean(np.abs(actual - predicted))
    return mae
```

In [14]:
```
assert 200 <= mae(y_val_pred, y_val) <= 300
print("Validation Error: ", mae(y_val_pred, y_val))
```

```
Validation Error:  266.136130855
```

Side note: scikit-learn also has tools to compute mean absolute error (`sklearn.metrics.mean_absolute_error`). In fact, most metrics that we have discussed in this class can be found as part of the [`sklearn.metrics` module (https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics)](https://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics). Some of these may come in handy as part of your feature engineering!

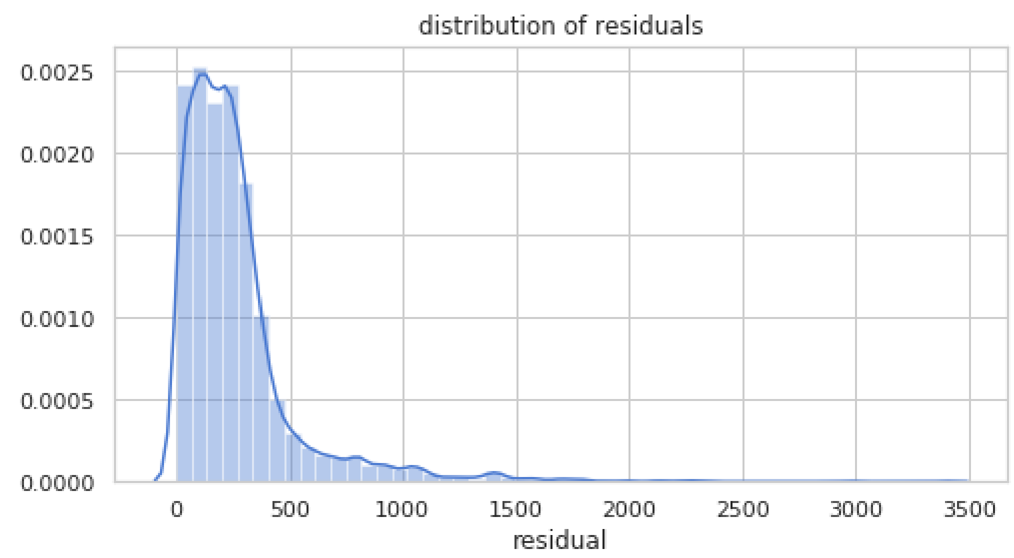
Visualizing Error

You should be getting between 200 and 300 MAE, which means your model was off by roughly 3-5 minutes on trips of average length 12 minutes. This is fairly decent performance given that our basic model uses only using the pickup/dropoff latitude and manhattan distance of the trip. 3-5 minutes may seem like a lot for a trip of 12 minutes, but keep in mind that this is the *average* error. This metric is susceptible to extreme outliers, which exist in our dataset.

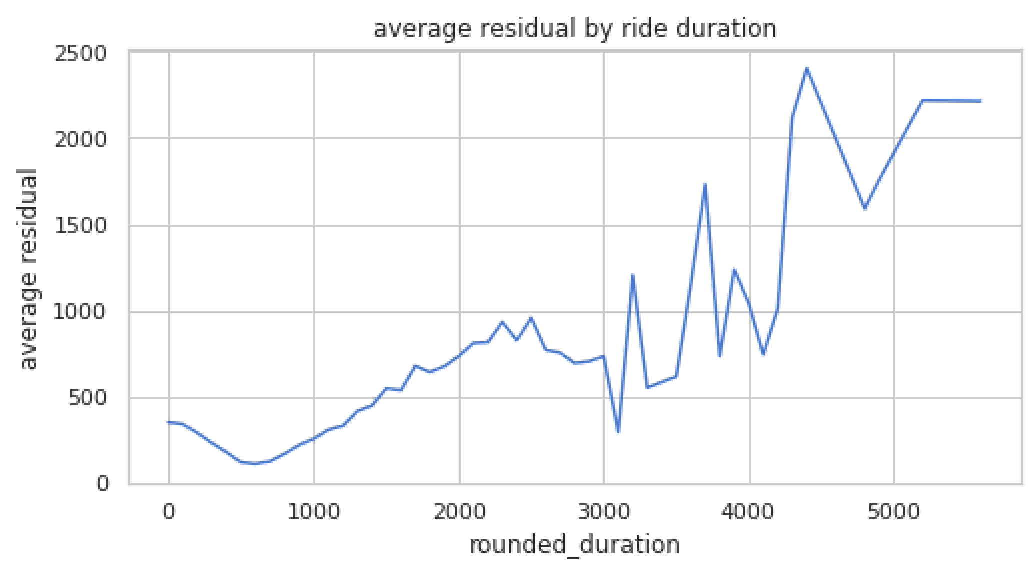
Now we will visualize the residual for the validation set. We will plot the following:

1. Distribution of residuals
2. Average residual grouping by ride duration

In [15]:
```
# Distribution of residuals
plt.figure(figsize=(8,4))
sns.distplot(np.abs(y_val - y_val_pred))
plt.xlabel('residual')
plt.title('distribution of residuals');
```



In [16]:
```
# Average residual grouping by ride duration
val_residual = X_val.copy()
val_residual['duration'] = y_val
val_residual['rounded_duration'] = np.around(y_val, -2)
val_residual['residual'] = np.abs(y_val - y_val_pred)
tmp = val_residual.groupby('rounded_duration').mean()
plt.figure(figsize=(8,4))
tmp['residual'].plot()
plt.ylabel('average residual')
plt.title('average residual by ride duration');
```



In the first visualization, we see that most of the residuals are centered around 250 seconds ~ 4 minutes. There is a minor right tail, suggesting that we are still unable to accurately fit some outliers in our data. The second visualization also suggests this, as we see the average residual increasing as a somewhat linear function of duration. But given that our average ride duration is roughly 600-700 seconds, it seems that we are indeed optimizing for the average ride because the residuals are smallest around 600-700.

Keep this in mind when creating your final model! Visualizing the error is a powerful tool and may help diagnose shortcomings of your model. Let's go ahead and submit to kaggle, although your error on the test set may be higher than 300.

Submission to Kaggle

The following code will write your predictions on the test dataset to a CSV, which you can submit to Kaggle. You may need to modify it to suit your needs, but we recommend you make a copy and preserve the original function.

Remember that if you've performed transformations or featurization on the training data, you must also perform the same transformations on the test data in order to make predictions. For example, if you've created features for the columns `pickup_datetime` or `pickup_latitude` on the training data, you must also extract the same features in order to use scikit-learn's `.predict(...)` method.

In [17]:
```
from datetime import datetime
def generate_submission(test, predictions, force=False):
    if force:
        if not os.path.isdir("submissions"):
            os.mkdir("submissions")
        submission_df = pd.DataFrame({
            "id": test_df.index.values,
            "duration": predictions,
        },
            columns=['id', 'duration'])

        timestamp = datetime.isoformat(datetime.now()).split(".")[0]

        submission_df.to_csv(f'submissions/submission_{timestamp}.csv', index=False)

        print(f'Created a CSV file: submission_{timestamp}.csv')
        print('You may now upload this CSV file to Kaggle for scoring.')
```

In [18]:
```
X_test, _ = process_data_gm1(test_df, True)
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

In [19]:
```
assert list(X_train.columns) == list(X_test.columns), "Different columns or different column ordering"
submission_predictions = (guided_model_1
                          .fit(X_train, y_train)
                          .predict(X_test))
submission_predictions = submission_predictions.astype(int)
submission_predictions[submission_predictions < 0] = 0
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-04T22:28:34.csv
You may now upload this CSV file to Kaggle for scoring.
```

In [20]:
```
# Check your submission
assert isinstance(submission_predictions, np.ndarray), "Submission not an array"
assert all(submission_predictions >= 0), "Duration must be non-negative"
assert issubclass(submission_predictions.dtype.type, np.integer), "Seconds must be integers"
```

Your Turn!

Now it's your turn! Draw upon everything you have learned this semester to find the best features to help your model accurately predict the duration of a taxi ride.

You may use whatever method you prefer in order to create features. You may use features that we created and features that you discovered yourself from any of the 2 datasets. However, we want to make it fair to students who are seeing these techniques for the first time. As such, you are only allowed regression models and their regularized forms. This means no random forest, k-nearest-neighbors, neural nets, etc.

Here are some ideas to improve your model:

- **Data selection**: January 2016 was an odd month for taxi rides due to the blizzard. Would it help to select training data differently?
- **Data cleaning**: Try cleaning your data in different ways. In particular, consider how to handle outliers.
- **Better features**: Explore the 2 datasets and find what features are most helpful. Utilize external datasets to improve your accuracy.
- **Regularization**: Try different forms of regularization to avoid fitting to the training set. Recall that `Ridge` and `Lasso` are the names of the classes in `sklearn.linear_model` that combine `LinearRegression` with regularization techniques.
- **Model selection**: You can adjust parameters of your model (e.g., the regularization parameter) to achieve higher accuracy. [GridSearchCV (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)](http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) may be helpful.
- **Validation**: Recall that you should use cross-validation to do feature and model selection properly! Otherwise, you will likely overfit to your training data.

There's many things you could try that could help your model. We have only suggested a few. Be creative and innovative! Please use `proj2_extras.ipynb` for all of your extraneous work. Note that you will be submitting `proj2_extras.ipynb` and we will be grading it. Please properly comment and format this notebook!

Once you are satisfied with your results, answer the questions in the Deliverables section. You may want to read this section in advance so you have an idea of what we're looking for.

Deliverables

Feature/Model Selection Process

Let's first look at selection of better features. In this following cell, describe the process of choosing good features to improve your model. You should use at least 3-4 sentences each to address the follow questions. Backup your responses with graphs supporting your claim (you can save figures and load them, no need to add the plotting code here). Use these questions to concisely summarize all of your extra work!

Question 1a

How did you find better features for your model?

In [21]:
```
q1a_answer = r"""
I testet more columns in the given dataset by considering each coloumn as a feature and visualizing the
relationship between such feature and duration.

If the values of the feature are discrete, I use barplot. And if the duration distincts from each value, I add
this feature to my feature matirx.

If the values of the feature are continuous, I use scatter plot(we can also find outliers with scatter plot).
And if the duration have a linear relationship with such feature, I add this feature to my feature matirx.

If the feature helps improve the accuray, it's a good feature.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 1b

What did you try that worked / didn't work?

In [22]:
```
q1b_answer = r"""

I found that 'tip_amount', 'haversine', 'trip_distance', 'ifdaytime', 'ifweekday', 'total_amount' and
'fare_amount' worked.

However, record_id', 'VendorID', passenger_count','RatecodeID','store_and_fwd_flag','payment_type',
'improvement_surcharge', 'total_amount', 'month','week_of_year' and 'day_of_month' didn't work.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 1c

What was surprising in your search for good features?

In [23]:
```
q1c_answer = r"""
I found that 'total_amount' is extremely helpful, it is proportional to the duration.
And I found replacing the outliers with average value can make the feature more helpful, like pickup_latitude,
pickup_longtitute have many values as 0.

"""
# YOUR CODE HERE
#raise NotImplementedError()
```

Question 2

Just as in the guided model above, you should encapsulate as much of your workflow into functions as possible. Define `process_data_fm` and `final model` in the cell below. In order to calculate your final model's MAE, we will run the code in the cell after that.

Note: You *MUST* name the model you wish to be evaluated on `final_model`. This is what we will be using to generate your predictions. We will take the state of `final_model` right after executing the cell below and run the following code:

```
# Load in test_df, solutions
X_test, _ = process_data_fm(test_df, True)
submission_predictions = final_model.predict(X_test)
# Generate score for autograding
```

We encourage you to conduct all of your exploratory work in `proj2_extras.ipynb`, which will be graded for 10 points.

In [24]:
```
data_file_fm = Path("./", "cleaned_data_2016.hdf")
train_df_fm = pd.read_hdf(data_file_fm, "train")
val_df_fm = pd.read_hdf(data_file_fm, "val")
```

In [25]:
```
def add_ifdaytime(data):
    data['ifdaytime'] = (data['hour'] >= 8) & (data['hour'] <= 18)
    return data

def add_ifweekday(data):
    data['ifweekday'] = data['day_of_week'] > 4
    return data

def drop_outlier(data, col, _filter):
    return data.loc[data[col][lambda x: _filter(x)].index]

def replace_outlier(data, col, _filter):
    mean = data[col][lambda x : _filter(x)].mean()
    data[col] = data[col].apply(lambda x : x if _filter(x) else mean)
    return data



```

In [26]:
```
def process_data_fm(data, test=False):
    # Put your final pipeline here

    # data cleaning
    if(test):
        clean_data = replace_outlier
    else:
        clean_data = drop_outlier
        data =  clean_data(data,'duration', lambda x : (x < 8000) & x > 0)

    filter_latitude = lambda x : (x >= 40.63) & (x <= 40.85)

    data = clean_data(data,'pickup_latitude', filter_latitude )
    data = clean_data(data,'dropoff_latitude', filter_latitude )

    filter_longitude  = lambda x : (x >= -74.03) & (x <= -73.75)

    data = clean_data(data,'pickup_longitude', filter_longitude)
    data = clean_data(data,'dropoff_longitude', filter_longitude )

    data = clean_data(data,'total_amount', lambda x: (x>0) & (x <= 90))
    data = clean_data(data,'fare_amount', lambda x: (x>0) & (x <= 80))
    data = clean_data(data,'tip_amount', lambda x :(x>0) & (x <= 20) )

    data = clean_data(data, 'trip_distance', lambda x : x < 50)

    X = (
        data
        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)
        .pipe(add_ifdaytime)
        .pipe(add_ifweekday)
        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
              'tip_amount',
              'haversine',
              'trip_distance',
               'ifdaytime',
              'ifweekday',
              'total_amount',
              'fare_amount',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y

# YOUR CODE HERE
#raise NotImplementedError()
```

In [27]:
```
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
final_model = lm.Ridge(alpha = 3, fit_intercept=True)

# Define your final model here, feel free to try other forms of regression
final_model.fit(X_train_fm, y_train_fm)

y_train_pred_fm = final_model.predict(X_train_fm)
y_val_pred_fm = final_model.predict(X_val_fm)

print(mae(y_train_pred_fm,y_train_fm))
print(mae(y_val_pred_fm,y_val_fm))
```

```
122.552163418
122.027044081
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

In [28]:
```
# Feel free to change this cell
X_test, _ = process_data_fm(test_df, True)
final_predictions = final_model.predict(X_test)
final_predictions = final_predictions.astype(int)
generate_submission(test_df, final_predictions, True) # Change to true to generate prediction
```

```
Created a CSV file: submission_2018-12-04T22:28:35.csv
You may now upload this CSV file to Kaggle for scoring.
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)
```

Question 3

The following hidden cells will test your model on the test set. Please do not delete any of them if you want credit!

In [29]:
```
# NO TOUCH
```

In [30]:
```
# NOH
```

In [31]:
```
# STAHP
```

In [32]:
```
# NO MOLESTE
```

In [33]:
```
# VA-T'EN
```

In [34]:
```
# NEIN
```

In [35]:
```
# PLSNO
```

In [36]:
```
# THIS SPACE IS NOT YOURS
```

In [37]:
```
# TAWDEETAW
```

In [38]:
```
# MAU LEN
```

In [39]:
```
# ALMOST
```

In [40]:
```
# TO
```

In [41]:
```
# THE
```

In [42]:
```
# END
```

In [43]:
```
# Hmph
```

In [44]:
```
# Good riddance
```

In [45]:
```
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-04T22:28:35.csv
You may now upload this CSV file to Kaggle for scoring.
```

This should be the format of your CSV file.
Unix-users can verify it running `!head submission_{datetime}.csv` in a jupyter notebook cell.

```
id,duration
id3004672,965.3950873305439
id3505355,1375.0665915134596
id1217141,963.2285454171943
id2150126,1134.7680929570924
id1598245,878.5495792656438
id0668992,831.6700312449248
id1765014,993.1692116960185
id0898117,1091.1171629594755
id3905224,887.9037911118357
```

Kaggle link: https://www.kaggle.com/t/f8b3c6acc3a045cab152060a5bc79670 (https://www.kaggle.com/t/f8b3c6acc3a045cab152060a5bc79670)

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**

2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**

Before you turn in the homework, make sure everything runs as expected. To do so, select **Kernel→Restart & Run All** in the toolbar above. Remember to submit both on **DataHub** and **Gradescope**.

Please fill in your name and include a list of your collaborators below.

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Project 2: NYC Taxi Rides

Extras

Put all of your extra work in here. Feel free to save figures to use when completing Part 4.

In [2]:
```
import os
import pandas as pd
import numpy as np
from pathlib import Path
from sqlalchemy import create_engine
from utils import timeit
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
!ls -lh /srv/db/taxi_2016_student_small.sqlite
```

```
-rw-r--r-- 1 root root 2.1G Nov  7 04:44 /srv/db/taxi_2016_student_small.sqlite
```

In [4]:
```
DB_URI = "sqlite:////srv/db/taxi_2016_student_small.sqlite"
TABLE_NAME = "taxi"

sql_engine = create_engine(DB_URI)
with timeit():
    print(f"Table {TABLE_NAME} has {sql_engine.execute(f'SELECT COUNT(*) FROM {TABLE_NAME}').first()[0]} rows!")
```

```
Table taxi has 15000000 rows!
1.15 s elapsed
```

Data Selection:

we chose data from April instead of January, and then we export the data

In [5]:
```
query = f"""
            SELECT *
            FROM (
            SELECT *
            FROM (
            SELECT *
            FROM (
            SELECT *
            FROM taxi
            WHERE tpep_pickup_datetime
                BETWEEN '2016-04-01' AND '2016-04-30'
                AND record_id % 100 == 0
            ORDER BY tpep_pickup_datetime
            )
            WHERE (julianday(tpep_dropoff_datetime) - julianday(tpep_pickup_datetime)) < 0.5
            )
            )
            WHERE passenger_count > 0
            """

with timeit(): # this query should take less than a second
    cleaned_df = pd.read_sql(query, sql_engine)
cleaned_df['tpep_pickup_datetime'] = pd.to_datetime(cleaned_df['tpep_pickup_datetime'])
cleaned_df['tpep_dropoff_datetime'] = pd.to_datetime(cleaned_df['tpep_dropoff_datetime'])
cleaned_df['duration'] = cleaned_df["tpep_dropoff_datetime"]-cleaned_df["tpep_pickup_datetime"]
cleaned_df['duration'] = cleaned_df['duration'].dt.total_seconds()
```

```
2.62 s elapsed
```

In [6]:
```
from sklearn.model_selection import train_test_split
train_df, val_df = train_test_split(cleaned_df, test_size=0.2, random_state=42)
data_file = Path("./", "cleaned_data_2016.hdf") # Path of hdf file
train_df.to_hdf(data_file, "train") # Train data of hdf file
val_df.to_hdf(data_file, "val") # Val data of hdf file
```

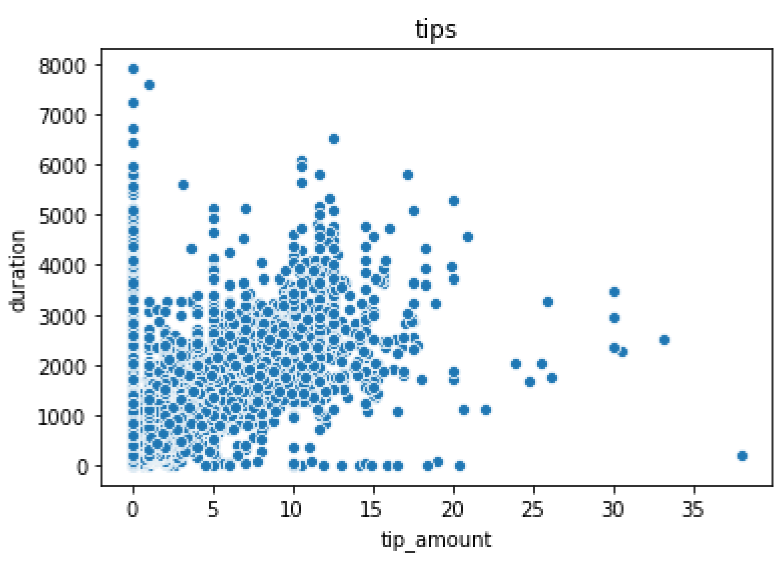
Better features

We test the the following features: 'tip_amount','trip_distance','ifdaytime','ifweekday','total_amount','fare_amount.

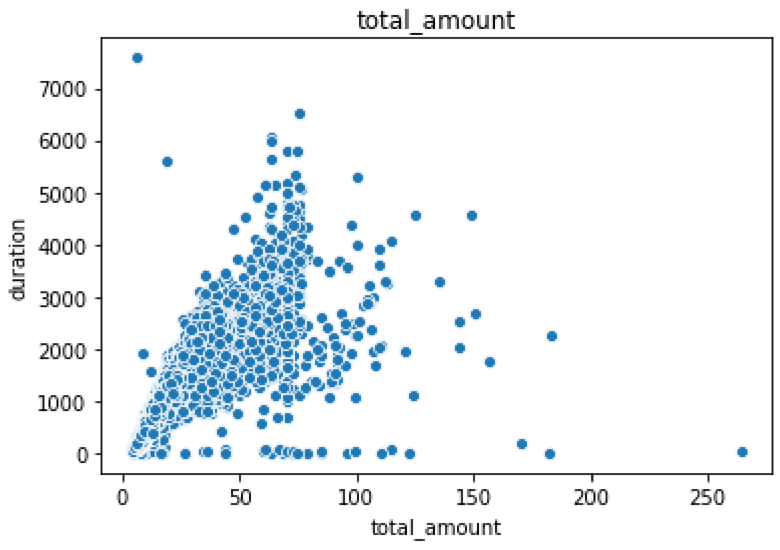
And we found those feature can help us predict duration

In [7]:
```
cleaned_df = cleaned_df[cleaned_df['duration'] < 10000]
```

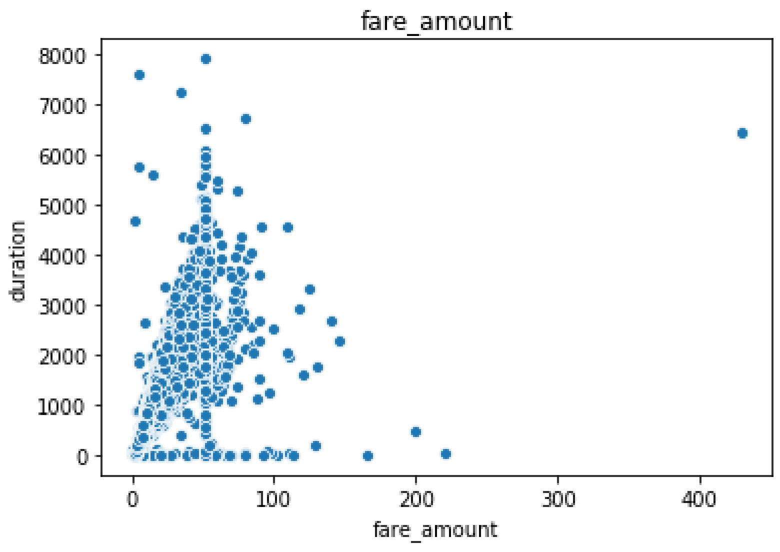
In [8]:
```
cleaned_df_feature = cleaned_df[cleaned_df['tip_amount'] < 40]
sns.scatterplot('tip_amount','duration', data = cleaned_df_feature)
plt.title('tips')
plt.show()
```



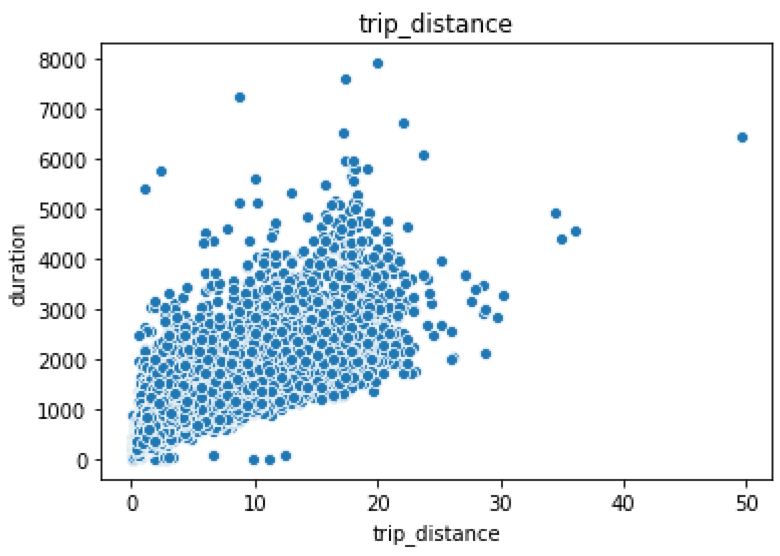
In [9]:
```
cleaned_df_feature = cleaned_df[cleaned_df['tip_amount'] > 0]
sns.scatterplot('total_amount','duration', data = cleaned_df_feature)
plt.title('total_amount')
plt.show()
```



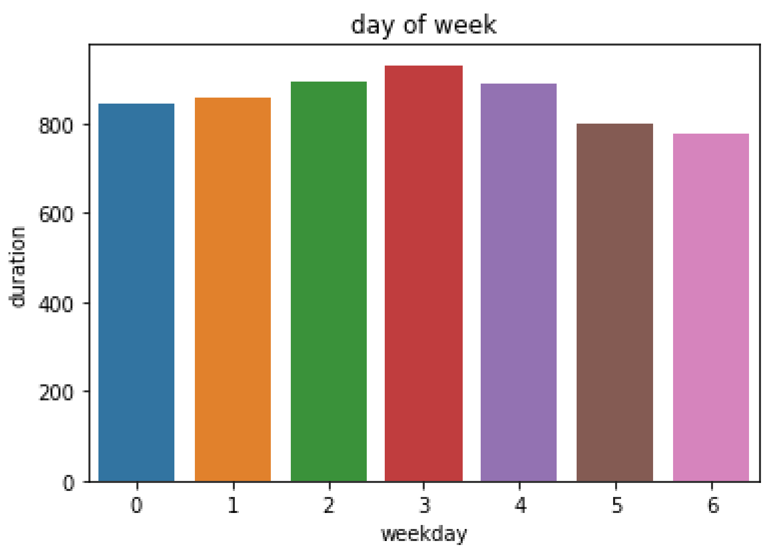
In [10]:
```
cleaned_df_feature = cleaned_df[cleaned_df['fare_amount'] > 0]
sns.scatterplot('fare_amount','duration', data = cleaned_df_feature)
plt.title('fare_amount')
plt.show()
```



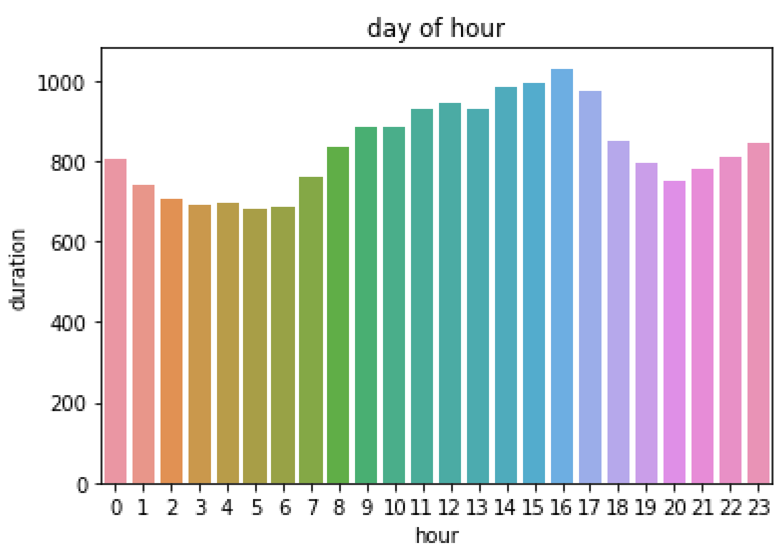
In [11]:
```
cleaned_df_feature = cleaned_df[cleaned_df['trip_distance'] > 0]
sns.scatterplot('trip_distance','duration', data = cleaned_df_feature)
plt.title('trip_distance')
plt.show()
```



In [12]:
```
cleaned_df['weekday'] =  cleaned_df['tpep_pickup_datetime'].dt.dayofweek
groupByweekday = cleaned_df.groupby('weekday')['duration'].mean()
sns.barplot(x =groupByweekday.index, y = groupByweekday)
plt.title('day of week')
plt.show()
```



In [13]:
```
cleaned_df['hour'] =  cleaned_df['tpep_pickup_datetime'].dt.hour
groupByhour = cleaned_df.groupby('hour')['duration'].mean()
sns.barplot(x =groupByhour.index, y = groupByhour)
plt.title('day of hour')
plt.show()
```



Feature Selection and Data Cleaning

In [14]:
```
data_file_fm = Path("./", "cleaned_data_2016.hdf")
train_df_fm = pd.read_hdf(data_file_fm, "train")
val_df_fm = pd.read_hdf(data_file_fm, "val")
```

In [15]:

```python
# Copied from part 2
def haversine(lat1, lng1, lat2, lng2):
    """
    Compute haversine distance
    """
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    average_earth_radius = 6371
    lat = lat2 - lat1
    lng = lng2 - lng1
    d = np.sin(lat * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng * 0.5) ** 2
    h = 2 * average_earth_radius * np.arcsin(np.sqrt(d))
    return h

# Copied from part 2
def manhattan_distance(lat1, lng1, lat2, lng2):
    """
    Compute Manhattan distance
    """
    a = haversine(lat1, lng1, lat1, lng2)
    b = haversine(lat1, lng1, lat2, lng1)
    return a + b

# Copied from part 2
def bearing(lat1, lng1, lat2, lng2):
    """
    Compute the bearing, or angle, from (lat1, lng1) to (lat2, lng2).
    A bearing of 0 refers to a NORTH orientation.
    """
    lng_delta_rad = np.radians(lng2 - lng1)
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    return np.degrees(np.arctan2(y, x))

# Copied from part 2
def add_time_columns(df):
    """
    Add temporal features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'month'] = df['tpep_pickup_datetime'].dt.month
    df.loc[:, 'week_of_year'] = df['tpep_pickup_datetime'].dt.weekofyear
    df.loc[:, 'day_of_month'] = df['tpep_pickup_datetime'].dt.day
    df.loc[:, 'day_of_week'] = df['tpep_pickup_datetime'].dt.dayofweek
    df.loc[:, 'hour'] = df['tpep_pickup_datetime'].dt.hour
    df.loc[:, 'week_hour'] = df['tpep_pickup_datetime'].dt.weekday * 24 + df['hour']
    return df

# Copied from part 2
def add_distance_columns(df):
    """
    Add distance features to df
    """
    df.is_copy = False # propogate write to original dataframe
    df.loc[:, 'manhattan'] = manhattan_distance(lat1=df['pickup_latitude'],
                                                lng1=df['pickup_longitude'],
                                                lat2=df['dropoff_latitude'],
                                                lng2=df['dropoff_longitude'])

    df.loc[:, 'bearing'] = bearing(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    df.loc[:, 'haversine'] = haversine(lat1=df['pickup_latitude'],
                                   lng1=df['pickup_longitude'],
                                   lat2=df['dropoff_latitude'],
                                   lng2=df['dropoff_longitude'])
    return df

def select_columns(data, *columns):
    return data.loc[:, columns]

def mae(actual, predicted):
    """
    Calculates MAE from actual and predicted values
    Input:
      actual (1D array-like): vector of actual values
      predicted (1D array-like): vector of predicted/fitted values
    Output:
      a float, the MAE
    """

    mae = np.mean(np.abs(actual - predicted))
    return mae
```

In [16]:

```python
def add_ifdaytime(data):
    data['ifdaytime'] = (data['hour'] >= 8) & (data['hour'] <= 18)
    return data

def add_ifweekday(data):
    data['ifweekday'] = data['day_of_week'] > 4
    return data

def drop_outlier(data, col, _filter):
    return data.loc[data[col][lambda x: _filter(x)].index]

def replace_outlier(data, col, _filter):
    mean = data[col][lambda x : _filter(x)].mean()
    data[col] = data[col].apply(lambda x : x if _filter(x) else mean)
    return data

```

In [17]:
```
def process_data_fm(data, test=False):
    # Put your final pipeline here

    # data cleaning
    if(test):
        clean_data = replace_outlier
    else:
        clean_data = drop_outlier
        data =  clean_data(data,'duration', lambda x : (x < 8000) & x > 0)

    filter_latitude = lambda x : (x >= 40.63) & (x <= 40.85)

    data = clean_data(data,'pickup_latitude', filter_latitude )
    data = clean_data(data,'dropoff_latitude', filter_latitude )

    filter_longitude  = lambda x : (x >= -74.03) & (x <= -73.75)

    data = clean_data(data,'pickup_longitude', filter_longitude)
    data = clean_data(data,'dropoff_longitude', filter_longitude )

    data = clean_data(data,'total_amount', lambda x: (x>0) & (x <= 90))
    data = clean_data(data,'fare_amount', lambda x: (x>0) & (x <= 80))
    data = clean_data(data,'tip_amount', lambda x :(x>0) & (x <= 20) )

    data = clean_data(data, 'trip_distance', lambda x : x < 50)

    X = (
        data
        # Transform data
        .pipe(add_time_columns)
        .pipe(add_distance_columns)
        .pipe(add_ifdaytime)
        .pipe(add_ifweekday)
        .pipe(select_columns,
              'pickup_longitude',
              'pickup_latitude',
              'dropoff_longitude',
              'dropoff_latitude',
              'manhattan',
              'tip_amount',
              'haversine',
              'trip_distance',
               'ifdaytime',
              'ifweekday',
              'total_amount',
              'fare_amount',
             )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y

# YOUR CODE HERE
#raise NotImplementedError()
```

Parameter Section for Ridge and Lasso and Model Selection

In [18]:
```
import sklearn.linear_model as lm
```

In [19]:
```
# Parameter Section for Ridge
_lambdas = [0.01,0.1,0.2,0.5,1,2,3,5,10]
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
for _lambda in _lambdas:
    final_model = lm.Ridge(alpha = _lambda, fit_intercept=True)

    # Define your final model here, feel free to try other forms of regression
    final_model.fit(X_train_fm, y_train_fm)

    y_val_pred_fm = final_model.predict(X_val_fm)

    print("for lambda " + str(_lambda) + ", validation accuracy: " + str(mae(y_val_pred_fm,y_val_fm)))
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future ver
sion.
  return object.__setattr__(self, name, value)

for lambda 0.01, validation accuracy: 122.773109937
for lambda 0.1, validation accuracy: 122.715747715
for lambda 0.2, validation accuracy: 122.655868278
for lambda 0.5, validation accuracy: 122.502909548
for lambda 1, validation accuracy: 122.325242267
for lambda 2, validation accuracy: 122.109814835
for lambda 3, validation accuracy: 122.027044081
for lambda 5, validation accuracy: 122.060878093
for lambda 10, validation accuracy: 122.361729584
```

In [20]:

```
# Parameter Section for Ridge
_lambdas = [0.01,0.1,0.2,0.5,1,2,3,5,10]
X_train_fm, y_train_fm = process_data_fm(train_df_fm)
X_val_fm, y_val_fm = process_data_fm(val_df_fm)

#final_model = lm.LinearRegression(fit_intercept=True)
for _lambda in _lambdas:
    final_model = lm.Lasso(alpha = _lambda, fit_intercept=True)

    # Define your final model here, feel free to try other forms of regression
    final_model.fit(X_train_fm, y_train_fm)

    y_val_pred_fm = final_model.predict(X_val_fm)

    print("for lambda " + str(_lambda) + ", validation accuracy: " + str(mae(y_val_pred_fm,y_val_fm)))
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future version.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.py:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be removed in a future version.
  return object.__setattr__(self, name, value)
/srv/conda/envs/data100/lib/python3.6/site-packages/sklearn/linear_model/coordinate_descent.py:491: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations. Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

```
for lambda 0.01, validation accuracy: 122.720904993
for lambda 0.1, validation accuracy: 122.359093687
for lambda 0.2, validation accuracy: 122.2259669
for lambda 0.5, validation accuracy: 122.37614613
for lambda 1, validation accuracy: 124.429678243
for lambda 2, validation accuracy: 124.532659775
for lambda 3, validation accuracy: 124.788272559
for lambda 5, validation accuracy: 125.625109785
for lambda 10, validation accuracy: 128.680014266
```

So we find using Ridge regression with lambda = 3 is the best

Submission

You're almost done!

Before submitting this assignment, ensure that you have:

1. Restarted the Kernel (in the menubar, select Kernel→Restart & Run All)
2. Validated the notebook by clicking the "Validate" button.

Then,

1. **Submit** the assignment via the Assignments tab in **Datahub**
2. **Upload and tag** the manually reviewed portions of the assignment on **Gradescope**