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 [28]:
```
NAME = "Devin Hua"
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 [29]:
```
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 [30]:
```
# 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/col
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 20:05:22 2018
MD5 hash of file: a445b925d24f319cb60bd3ace6e4172b
Located at data/collisions.zip
```

We will store the taxi data locally before loading it.

In [31]:
```
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 [32]:
```
# Run this cell to load the collisions data.
skiprows = None
collisions = pd.read_csv(collisions_dir/'collisions_2016.csv', index_col='U
                         parse_dates={'DATETIME':["DATE","TIME"]}, 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()
```

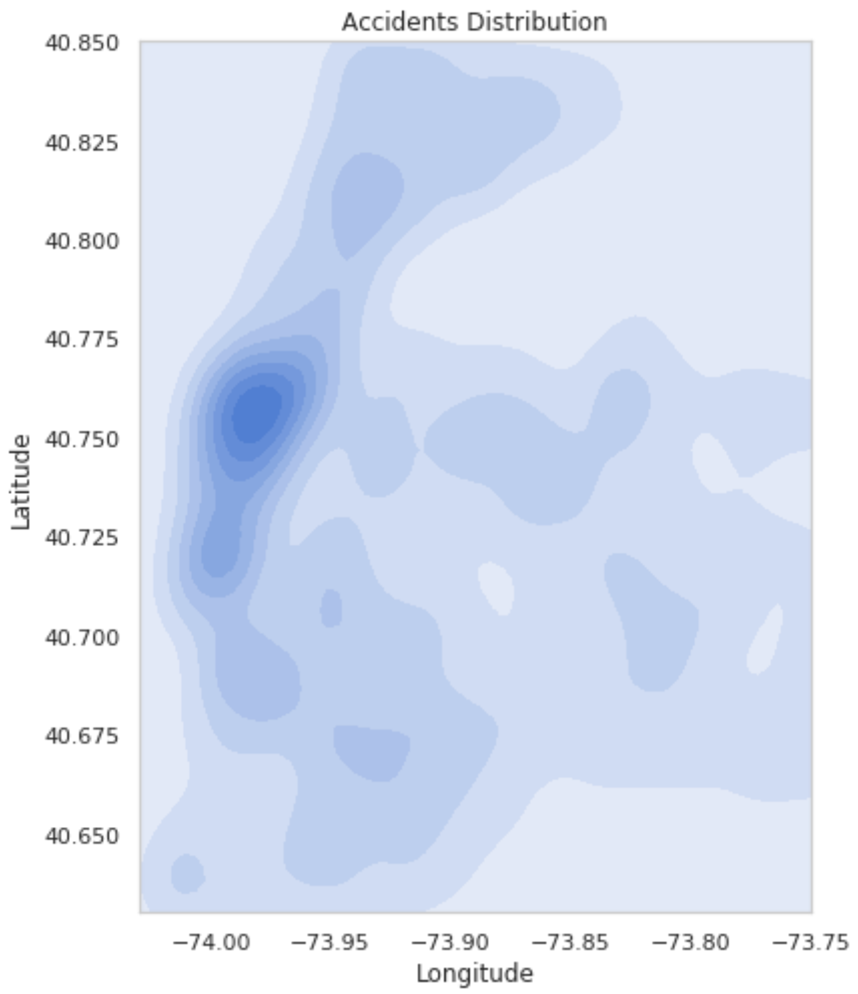
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 116691 entries, 3589202 to 3363795
Data columns (total 30 columns):
DATETIME                         116691 non-null datetime64[ns]
Unnamed: 0                       116691 non-null int64
BOROUGH                          100532 non-null object
ZIP 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
NUMBER OF MOTORIST INJURED       116691 non-null int64
NUMBER OF MOTORIST KILLED        116691 non-null int64
CONTRIBUTING FACTOR VEHICLE 1    115162 non-null object
CONTRIBUTING FACTOR VEHICLE 2    101016 non-null object
CONTRIBUTING FACTOR VEHICLE 3    7772 non-null object
CONTRIBUTING FACTOR VEHICLE 4    1829 non-null object
CONTRIBUTING FACTOR VEHICLE 5    434 non-null object
VEHICLE TYPE CODE 1              115181 non-null object
VEHICLE TYPE CODE 2              92815 non-null object
VEHICLE TYPE CODE 3              7260 non-null object
VEHICLE TYPE CODE 4              1692 non-null object
VEHICLE TYPE CODE 5              403 non-null object
TIME                             116691 non-null int64
DATE                             116691 non-null object
dtypes: datetime64[ns](1), float64(3), int64(10), object(16)
memory usage: 27.6+ MB
```

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/da 73.9712488) for your convenience.

In [33]:
```
# 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,-74.0394431,1 73.9712488) that may be useful, and another page (https://www.6sqft.com/what-nycs-population-looks-like-day-vs-night/) that may be useful.

In [34]:
```
q1a_answer = r"""

The area where the accidents occur the most seem to reside in the southern

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1a_answer)
```

```
The area where the accidents occur the most seem to reside in the souther
n half of Manhattan, which contains popular places such as the Madison Sq
uare Garden and the Empire State Building. This means that people are hea
vily attracted to such places and the number of taxi rides to those place
s increase. More taxi rides result in a higher chance for collision, caus
ing most crashes to be within that region.
```

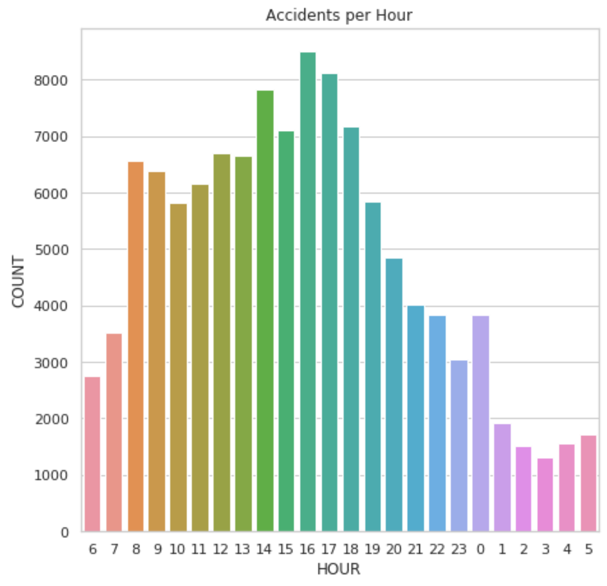
We see that an entry in accidents contains information on number of people injured/killed. Instead of using each of these columns separately, let's combine them into one column called `'SEVERITY'`. Let's also make columns `FATALITY` and `INJURY`, each aggregating the fatalities and injuries respectively.

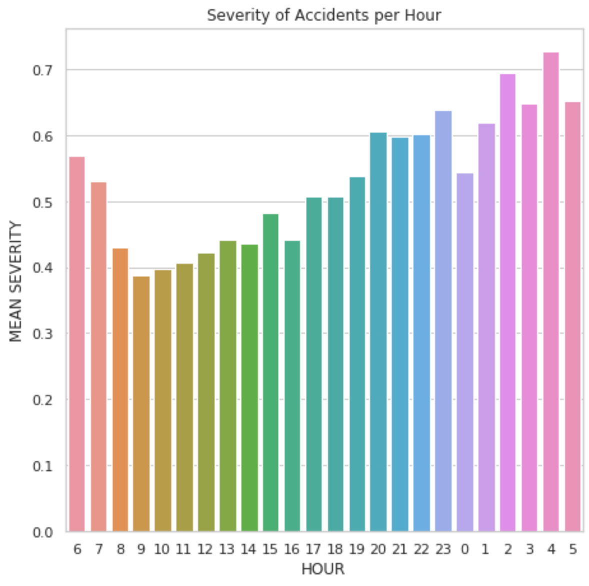
In [35]:
```
collisions['SEVERITY'] = collisions.filter(regex=r'NUMBER OF *').sum(axis=1
collisions['FATALITY'] = collisions.filter(regex=r'KILLED').sum(axis=1)
collisions['INJURY'] = collisions.filter(regex=r'INJURED').sum(axis=1)
```

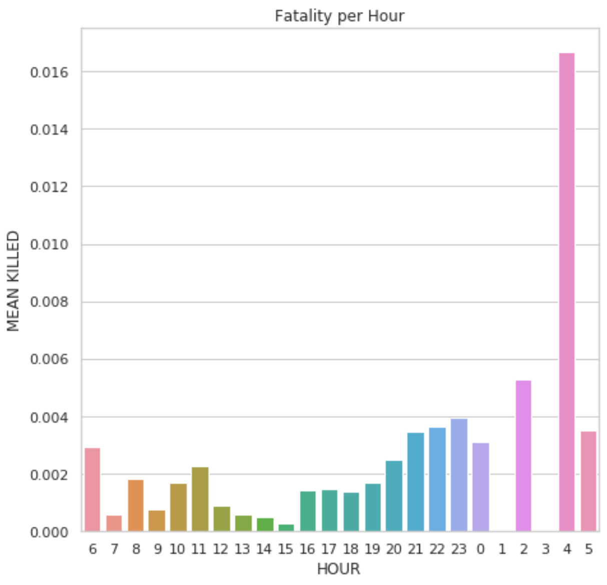
Now let's group by time and compare two aggregations: count vs mean. Below we plot the number of collisions and the mean severity of collisions by the hour, i.e. the `TIME` column. We visualize them side by side and set the start of our day to be 6 a.m.

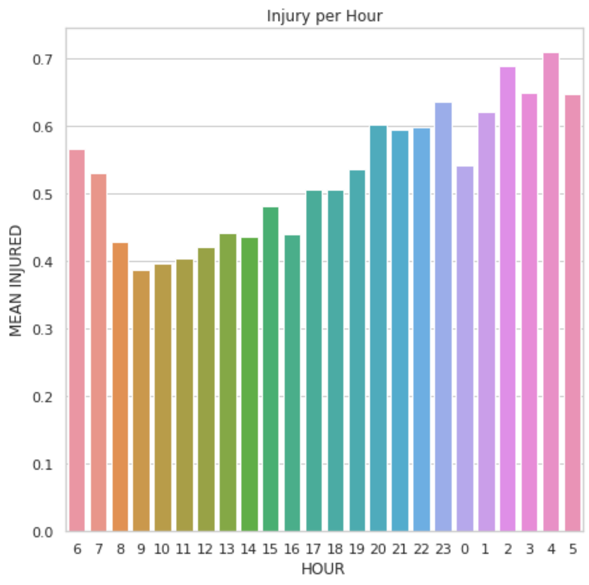
Let's also take a look at the mean number of casualties per hour and the mean number of injuries per hour, plotted below.

In [36]:
```
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.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=a
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=ax
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

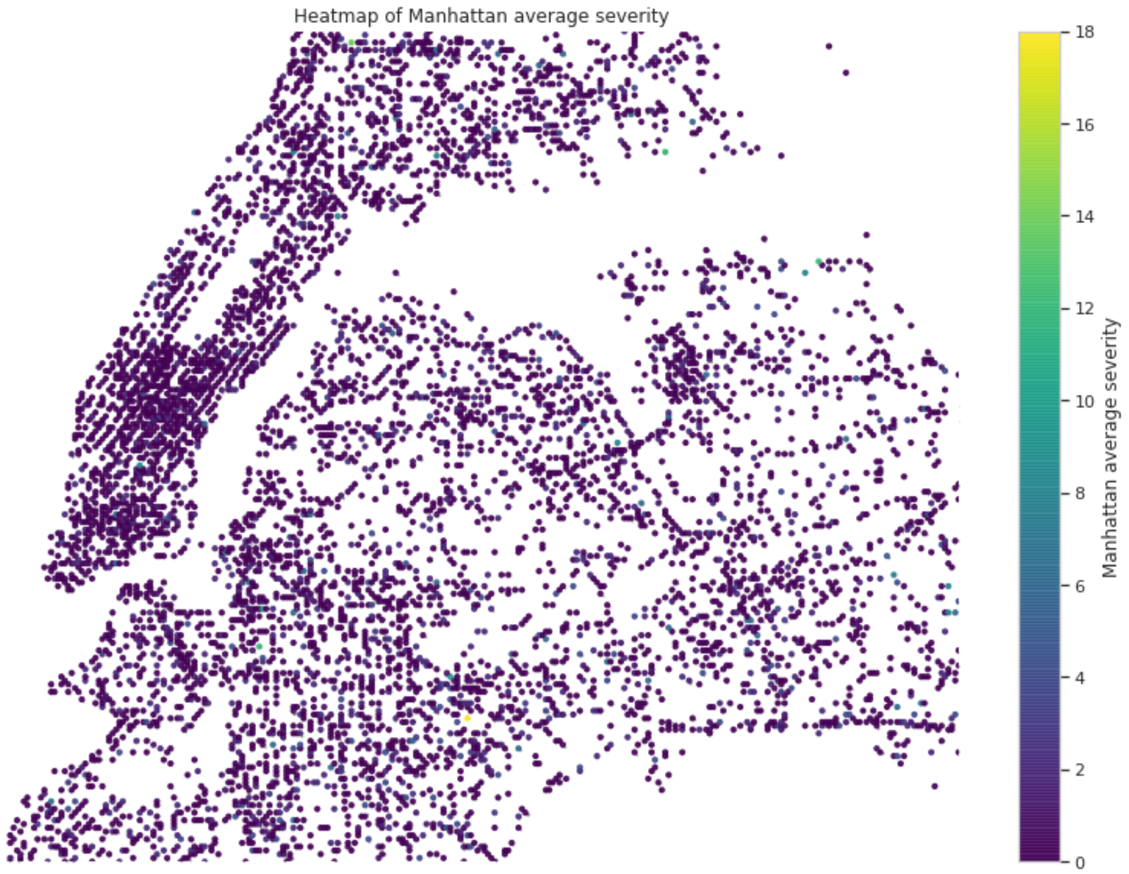
Based on the visualizations above, what can you say about each? Make a comparison between the accidents per hour vs the mean severity per hour. What about the number of fatalities per hour vs the number of injuries per hour? Why do we chose to have our hours start at 6 as opposed to 0?

In [37]:
```
q1b_answer = r"""

Although most accidents occur during the rush hour when everyone is heading

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1b_answer)
```

```
Although most accidents occur during the rush hour when everyone is headi
ng back from work, the most severe accidents occur early morning accordin
g to the graphs. The number of accidents peak from 5-6 PM while the sever
ity of the crashes peak from 1 am to around 5 am. Theee number of fatalit
ies have one giant peak around 4 in the morning, while the number of inju
ries steadily increases throughout the evening into the night and early m
orning.
```

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 [38]:
```
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');
```


Heatmap of Manhattan average severity
Manhattan average severity
18
16
14
12
10
8
6
4
2
0

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 [39]:
```
q1c_answer = r"""

Yes, I believe the location of the the accident affects the severity of the

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q1c_answer)
```

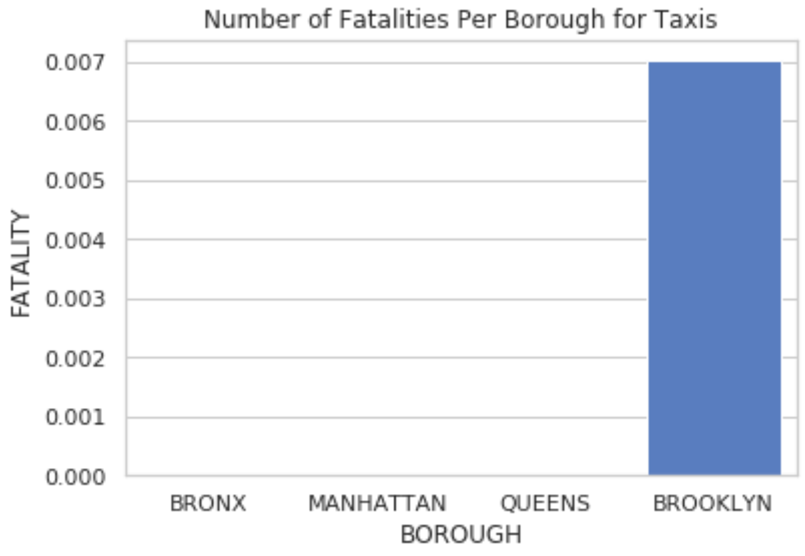
```
Yes, I believe the location of the the accident affects the severity of t
he crash. From the plot above, it seems like the more severe crashes are
in areas where the location density is relatively small, or away from the
crowded Center Manhattan. Something that might help and improve the plot
would be to have several plots that focus on different regions in Manhatt
an so that we can see more examples of severe crashes compared to just an
abundance of minor collisions.
```

Question 1d

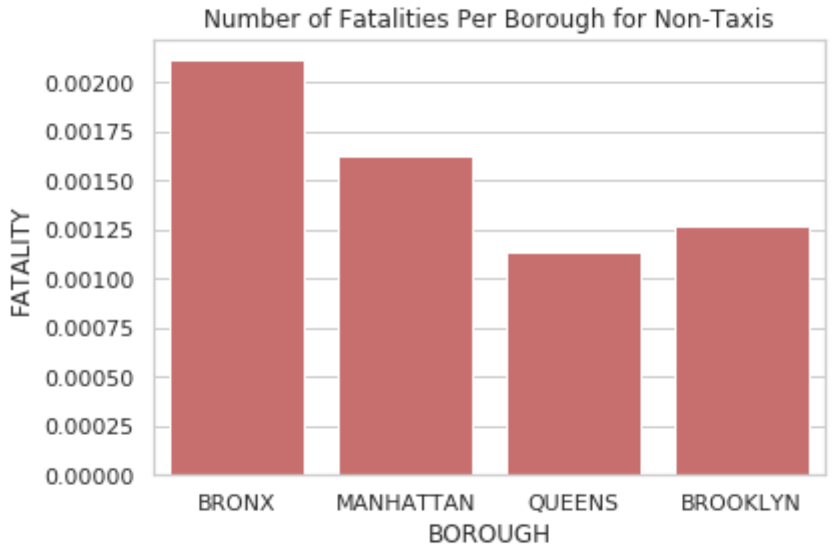
Create a plot to visualize one or more features of the `collisions` table.

In [40]:
```
taxi = collisions[collisions['VEHICLE TYPE CODE 1'] == 'TAXI']
not_taxi = collisions[collisions['VEHICLE TYPE CODE 1'] != 'TAXI']
borough = ['BRONX', 'MANHATTAN', 'QUEENS', 'BROOKLYN']
sns.barplot(taxi['BOROUGH'], taxi['FATALITY'], color='b', ci=None)
plt.title('Number of Fatalities Per Borough for Taxis')
```

Out[40]:
```
Text(0.5,1,'Number of Fatalities Per Borough for Taxis')
```



In [41]:
```
# YOUR CODE HERE
sns.barplot(not_taxi['BOROUGH'], not_taxi['FATALITY'], color='r', order=bor
plt.title('Number of Fatalities Per Borough for Non-Taxis');
#raise NotImplementedError()
```



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 [42]:
```
q1e_answer = r"""

1. I made two plots to compare the number of fatalities for each borough ca

2. I chose this feature to compare the level of intensity of taxi drivers t

3. I chose to use a barplot so that I can visualize the number of fatalitie
"""
# YOUR CODE HERE
#raise NotImplementedError()
print(q1e_answer)
```

```
1. I made two plots to compare the number of fatalities for each borough
caused by taxis and the number of fatalities for each borough caused by v
ehicles other than taxis.

2. I chose this feature to compare the level of intensity of taxi drivers
to other vehicle drivers in their respective boroughs. In general, it see
ms like taxi drivers have a much lower fatality rate possibly because the
re is fewer demand for taxis nowadays but perhaps also due to the fact th
at drivers in general take more precaution when driving someone else, mak
ing them drive more safely on the road. Moreover, specifically in Manhatt
an, the borough we constantly studied, the taxi drivers seem to be a lot
safer than other drivers in general as there were no reported fatalities
for taxi drivers, while there were several instances of fatalities for ot
her vehicles.

3. I chose to use a barplot so that I can visualize the number of fatalit
ies for each borough clearly as barplots provide a rough comparison betwe
en boroughs.
```

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 [43]:
```
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 [44]:
```
collisions_subset = collisions[['DATETIME', 'TIME', 'START', 'END', 'DATE',
# YOUR CODE HERE
#raise NotImplementedError()
collisions_subset.head(5)
```

Out[44]:

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

In [45]:
```
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 [46]:
```
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', '
train_df['date'] = train_df['tpep_pickup_datetime'].dt.date
```

In [47]:
```
collisions_subset = collisions_subset[collisions_subset['DATETIME'].dt.week
train_df = train_df[train_df['tpep_pickup_datetime'].dt.weekofyear == 3]
```

In [48]:
```
# merge the dataframe here
merged = train_df.merge(collisions_subset, how='left', left_on='date' ,righ

# YOUR CODE HERE
#raise NotImplementedError()

merged.head()
```

Out[48]:

	index	tpep_pickup_datetime	pickup_longitude	pickup_latitude	duration	date	DATETIME	TIMI
0	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 10:35:00	1
1	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 13:20:00	1
2	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 16:00:00	1
3	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 18:30:00	1
4	16709	2016-01-21 22:28:17	-73.997986	40.741215	736.0	2016-01-21	2016-01-21 00:05:00	(

In [49]:
```
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 [50]:
```
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 *
    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 [66]:
```
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 e
# (1) record's start_to_accident <= 5
# (2) pick up time is between start and end
is_accident_close = merged.loc[(merged['start_to_accident'] <= 5) & (merged

# YOUR CODE HERE
#raise NotImplementedError()

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

In [67]:
```
assert merged['accident_close'].sum() > 16000
```

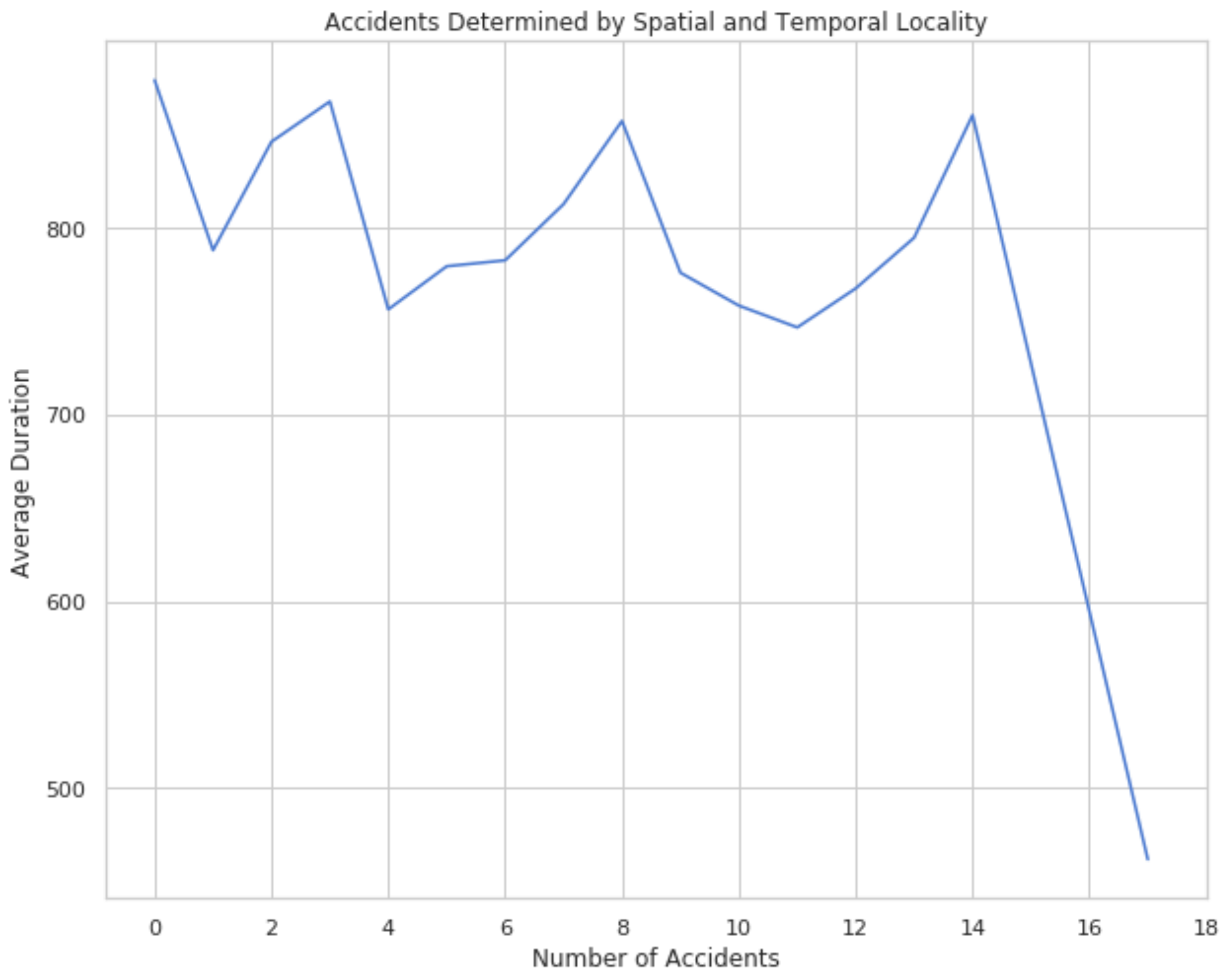
The last step is to aggregate the total number of proximal accidents. We want to count the total number of accidents that were close spatially and temporally and condition on that data.

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 [68]:
```
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
```

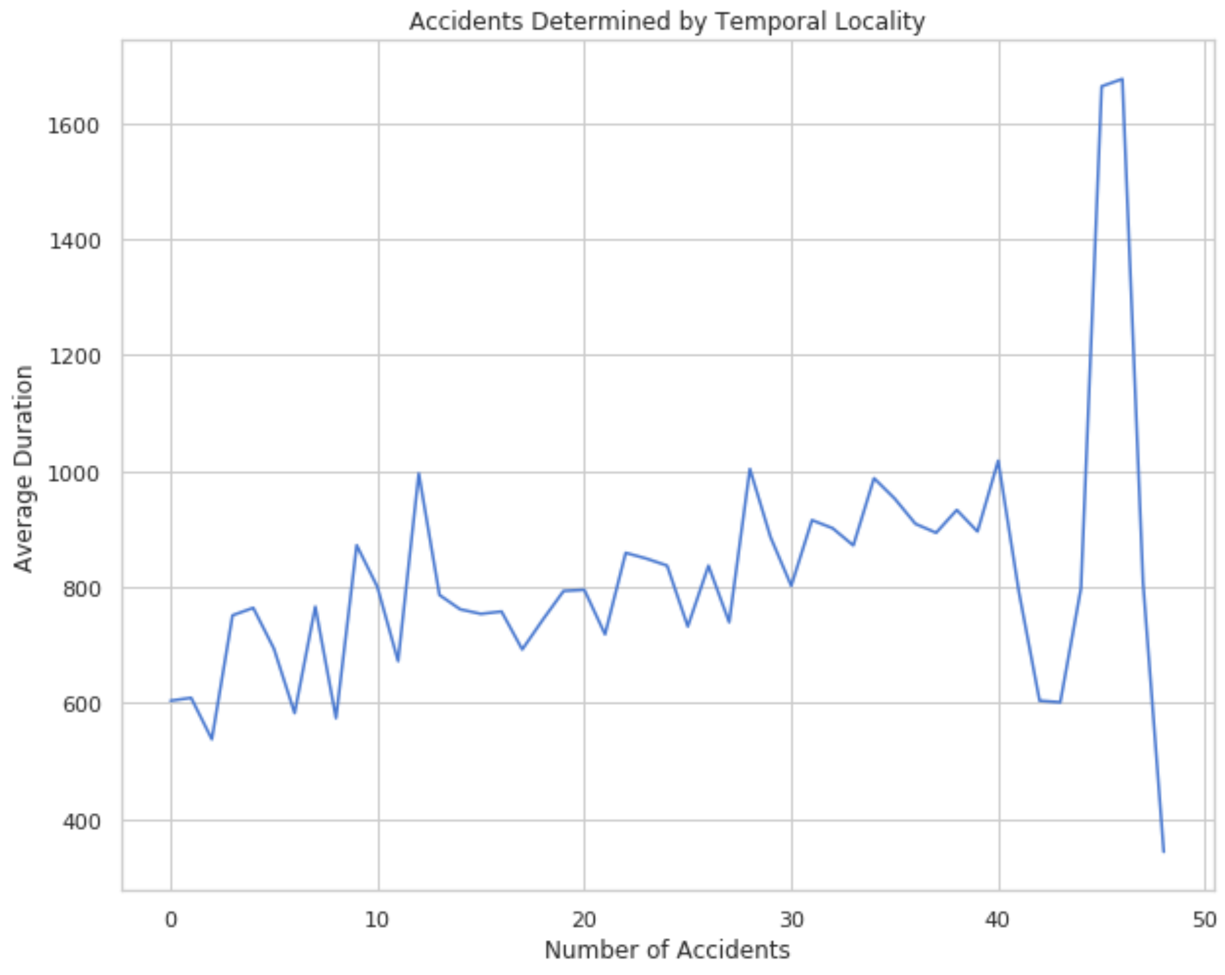
Next, for each value of `num_accidents` , we plot the average `duration` of rides with that number of accidents.

In [69]:
```
plt.figure(figsize=(10,8))
train_accidents.groupby('num_accidents')['duration'].mean().plot(xticks=np.
plt.title("Accidents Determined by Spatial and Temporal Locality")
plt.xlabel("Number of Accidents")
plt.ylabel("Average Duration")
plt.show();
```

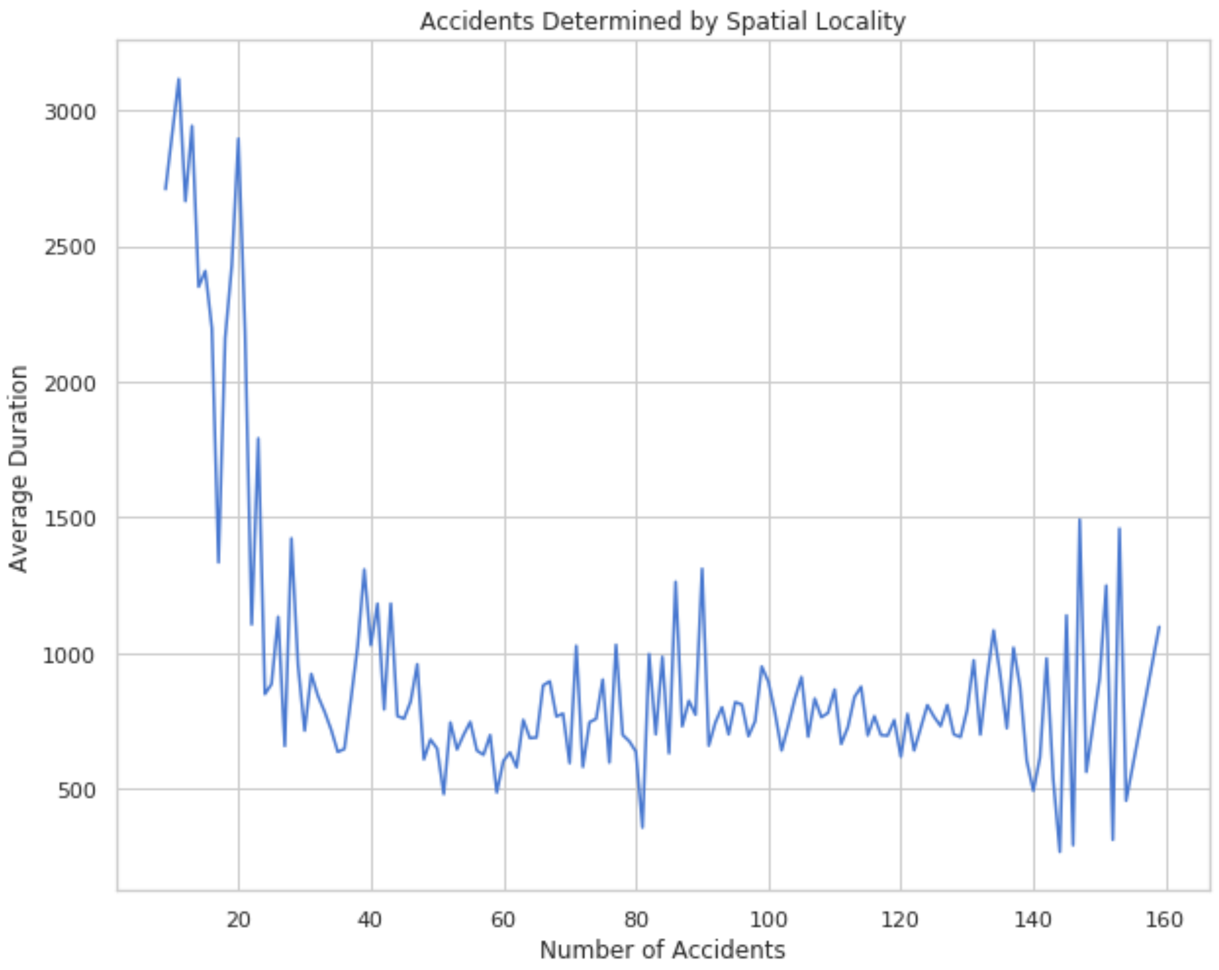


It seems that using both spatial and temporal proximity doesn't give us much insight on if collisions increase taxi ride durations. Let's try conditioning on spatial proximity and temporal proximity separately and see if there are more interesting results there.

In [71]:
```
# 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 [72]: # 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 [3]:
```
q2d_answer = r"""

The temporal and spatial proximity of accidents graph is very zigzaggity, a

"""

# YOUR CODE HERE
#raise NotImplementedError()

print(q2d_answer)
```

```
The temporal and spatial proximity of accidents graph is very zigzaggity,
alternating from values of 750 seconds to 900 seconds, up until about 14
accidents, where it takes a steep dive in average duration. This indicate
s that a majority of the accidents aren't that sever as they are easy to
handle and to take care of based on their average duration of 250 second
s. One way I would be able to test that hypothesis is to look at the temp
oral and spatial graphs separately and see if in general, a majority of a
ccidents were easily taken care of if in fact their respective durations
weren't that long. Separately, the temporal graph increases ever so sligh
t from 1 to about 40 accidents, which makes sense as one crash inevitably
affects the safety of other cars increasing their chances of crashing as
well within a certain time frame perhaps due to the possiblity of several
collisions at once or the fact that the respective road/path is very dang
erous. However, at 41 to 42 accidents, the plot takes a huge dive only to
increase just over double the duration at around 48 accidents, which is a
very interesting turn of events. This suggest that most of the accidents
have durations over 1600 seconds based on temporal locality. For the Spat
ial Locality Graph, the longest accidents are over 3000 seconds, emphasiz
ing the severity of the crashes and possibly hinting at several fatalitie
s. After 20 accidents, the graph normalizes to a duration about 1000 seco
nds for the rest of the graph. This means that for the most part, acciden
ts that happen within the same area are relatively minor and take only ab
out 15 minutes to handle. Not considering spatial proximity and only focu
sing on temporal locality would not affect the results too much. Given th
at there is a collision, the cars at fault would have have to crash simul
taenously. While it is true that crashes at different locations around th
e same time could affect the data, such cases are rare and ultimately wou
ld not affect the data too much.
```

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 [77]:
```
Path("data/part3").mkdir(parents=True, exist_ok=True)
data_file = Path("data/part3", "data_part3.hdf") # Path of hdf file
...
```

Out[77]:
```
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 [103]:
```
NAME = "Devin Hua"
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 [104]:
```
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, Grid

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 [105]:
```
# 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")
```

Testing

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

In [106]:
```
test_df = pd.read_csv("./proj2_test_data.csv")
test_df['tpep_pickup_datetime'] = pd.to_datetime(test_df['tpep_pickup_datet
test_df.head()
```

Out[106]:

	record_id	VendorID	tpep_pickup_datetime	passenger_count	trip_distance	pickup_longitude	pic
0	10000	1	2016-01-02 01:45:37	1	1.20	-73.982224	
1	19000	2	2016-01-02 03:05:16	1	10.90	-73.999977	
2	21000	1	2016-01-02 03:24:36	1	1.80	-73.986618	
3	23000	2	2016-01-02 03:47:38	1	5.95	-74.002922	
4	27000	1	2016-01-02 04:36:44	1	1.60	-73.986366	

In [107]:
```
test_df.describe()
```

Out[107]:

	record_id	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitud
count	1.377400e+04	13774.000000	13774.000000	13774.000000	13774.000000	13774.00000
mean	3.465950e+07	1.536082	1.663642	2.954688	-72.953619	40.18799
std	2.015133e+07	0.498714	1.311739	3.704427	8.628431	4.75318
min	1.000000e+04	1.000000	0.000000	0.000000	-77.039436	0.00000
25%	1.719975e+07	1.000000	1.000000	1.000000	-73.992058	40.73516
50%	3.457400e+07	2.000000	1.000000	1.700000	-73.981846	40.75243
75%	5.216875e+07	2.000000	2.000000	3.157500	-73.967119	40.76726
max	6.940400e+07	2.000000	6.000000	104.800000	0.000000	40.86821

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 [108]:
```
# 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 *
    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(
    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 + d
    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 [109]:
```
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 [110]:
```
# 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.p
y:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be remo
ved in a future version.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.p
y:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be remo
ved in a future version.
  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 [111]:
```
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 [112]:
```
X_train.iloc[0, :].dot(guided_model_1.coef_) + guided_model_1.intercept_
```

Out[112]:
```
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 [113]:
```
y_train_pred[0]
```

Out[113]:
```
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 [114]:
```
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 [115]:
```
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). Some of these may come in handy as part of your feature engineering!

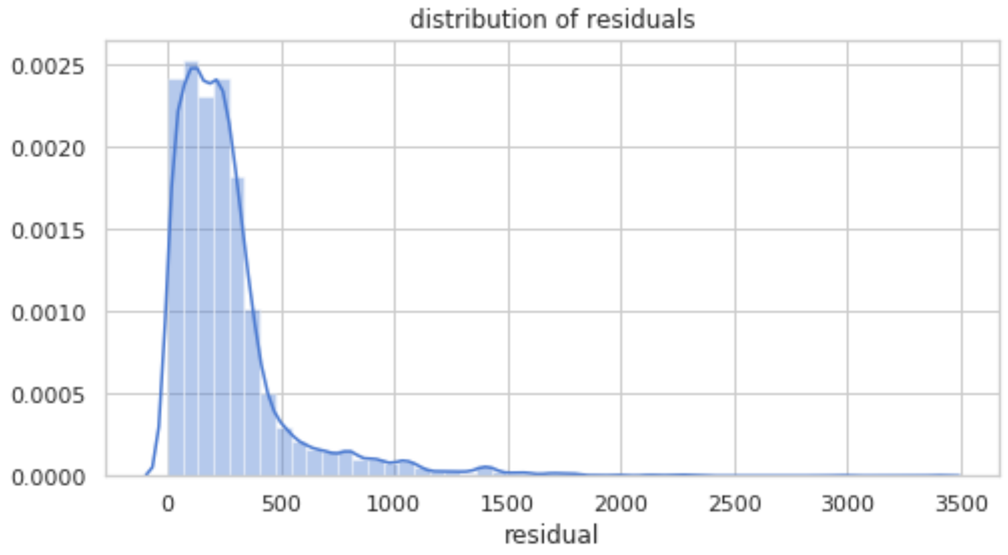
Visualizing Error

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.

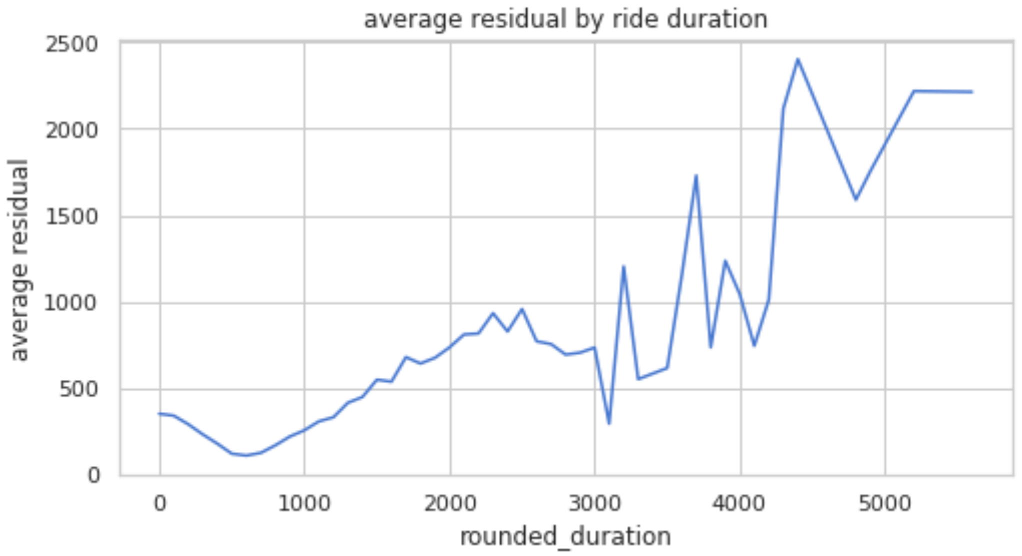
Now we will visualize the residual for the validation set. We will plot the following:

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

In [116]:
```
# 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 [117]:
```
# 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 [118]:
```
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', inc

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

In [119]:
```
X_test, _ = process_data_gm1(test_df, True)
```

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

In [120]:
```
assert list(X_train.columns) == list(X_test.columns), "Different columns or
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_2018-12-06T02:29:36.csv
You may now upload this CSV file to Kaggle for scoring.
```

In [121]:
```
# Check your submission
assert isinstance(submission_predictions, np.ndarray), "Submission not an a
assert all(submission_predictions >= 0), "Duration must be non-negative"
assert issubclass(submission_predictions.dtype.type, np.integer), "Seconds
```

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 [150]:

```
nse and finally I removed long trip distances that took a short time or vice
```

```
I first looked at the given train data set and realized that it only cont
ained taxi data only for the month of January. With that in mind, I looke
d to create a new DataFrame that included the months from January to June
so that the sample size would be much larger. However, this did not work
as planned so I decided to just create a new copy of the original datafra
me and clean the copied version. I then looked at the new copy I made and
then moved on to clean the values of the dataset first by removing lattit
ude and longitude values that both equal 0, because those infos are incor
rect. Next, I removed rides with no passengers because as taxis, if there
are no passengers, that means some files might contain data with a long d
uration but a total distance of zero. I also removed trip distances of 0
because that doesn't make sense and finally I removed long trip distances
that took a short time or vice versa because that's virtually impossible.
```

Question 1b

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

In [151]:
```
q1b_answer = r"""

I tried focusing on another specific month other than January but I found c

"""
print(q1b_answer)
# YOUR CODE HERE
#raise NotImplementedError()
```

```
I tried focusing on another specific month other than January but I found
out that those months were not too different from January's data so I dec
ided to just combine the months of January to June to get a larger sample
size. In the "Extras" section of the project, I tested two different mode
ls, one with January files only and the other with January-June files. In
the end I discovered that combining the months together only increased th
e number of outliers and ultimately had a larger MAE than just the Januar
y files so I decided to just use the original train data.
```

Question 1c

What was surprising in your search for good features?

In [152]:
```
q1c_answer = r"""

I noticed that some of the files had trip-distances of 0 which does not mak

"""
print(q1c_answer)
# YOUR CODE HERE
#raise NotImplementedError()
```

```
I noticed that some of the files had trip-distances of 0 which does not m
ake sense so I cleaned out the files and also I found out that there's a
flat rate of 52 to the airport despite the distance of the ride so that w
ill for sure skew the data. In the end I decided to just filter out fare_
amounts that equal to 52 and it helped the MAE.
```

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 [153]:
```
final_train = train_df.copy()
# passenger count cannot equal 0
final_train = final_train[final_train['passenger_count'] != 0]
# trip distance has to be greater than 0
final_train = final_train[final_train['trip_distance'] != 0]
# filter out missing latitude/longitude values
final_train = final_train[final_train['pickup_latitude'] != 0]
final_train = final_train[final_train['pickup_longitude'] != 0]
final_train = final_train[final_train['dropoff_latitude'] != 0]
final_train = final_train[final_train['dropoff_longitude'] != 0]
# filter out long duration that doesn't make sense
final_train = final_train[final_train["duration"] < 15000]
# fare amount has to be greater than 0, not equal to 52 (flat rate to the a
final_train = final_train[final_train["fare_amount"] != 52]
final_train = final_train[final_train["fare_amount"] < 75]
final_train = final_train[final_train["fare_amount"] > 0]
# duration has to be at least around 2 minutes but less than 2 hours
final_train = final_train[final_train["duration"] > 90]
final_train = final_train[final_train["duration"] < 7000]
# remove small trip distances that took a long time
final_train_df = final_train_df[(final_train_df['duration'] > 6000) & (fina
# remove longer trip distances that take little time
final_train_df = final_train_df[(final_train_df['duration'] < 250) & (final
```

In [154]:
```
: process_data_fm(data, test=False):
 # Put your final pipeline here
 X = (
 data

 # Transform data
 .pipe(add_time_columns)
 .pipe(add_distance_columns)
 .pipe(select_columns,
       "trip_distance",
       "fare_amount"

      )
 )
 if test:
     y = None
 else:
     y = data['duration']

 return X, y

om sklearn import linear_model

ıal_X_train, final_Y_train = process_data_fm(final_train)
ıal_X_val, final_Y_val = process_data_fm(val_df)
inal_model = lm.LinearRegression(fit_intercept=True)# Define your final mode
ıal_model = linear_model.Ridge(alpha=0.5)
ıal_model.fit(final_X_train, final_Y_train);

ıin_predict = final_model.predict(final_X_train)
._predict = final_model.predict(final_X_val)
'OUR CODE HERE
ıise NotImplementedError()
```

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

In [155]:
```
mae(train_predict, final_Y_train), mae(val_predict, final_Y_val)
```

Out[155]:
```
(59.947226448544555, 95.005505105641049)
```

In [156]:
```
# 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 g
```

```
Created a CSV file: submission_2018-12-06T03:11:21.csv
You may now upload this CSV file to Kaggle for scoring.
```

```
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.p
y:4388: FutureWarning: Attribute 'is_copy' is deprecated and will be remo
ved in a future version.
  object.__getattribute__(self, name)
/srv/conda/envs/data100/lib/python3.6/site-packages/pandas/core/generic.p
y:4389: FutureWarning: Attribute 'is_copy' is deprecated and will be remo
ved in a future version.
  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 [157]:
```
# NO TOUCH
```

In [158]:
```
# NOH
```

In [159]:
```
# STAHP
```

In [160]:
```
# NO MOLESTE
```

In [161]:
```
# VA-T'EN
```

In [162]:
```
# NEIN
```

In [163]:
```
# PLSNO
```

In [164]:
```
# THIS SPACE IS NOT YOURS
```

In [165]:
```
# TAWDEETAW
```

In [166]:
```
# MAU LEN
```

In [167]:
```
# ALMOST
```

In [168]:
```
# TO
```

In [169]:
```
# THE
```

In [170]:
```
# END
```

In [171]:
```
# Hmph
```

In [172]:
```
# Good riddance
```

In [174]:
```
generate_submission(test_df, submission_predictions, True)
```

```
Created a CSV file: submission_2018-12-06T03:12:13.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 = "Devin Hua"
COLLABORATORS = ""
```

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, Grid
from utils import timeit

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

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

%matplotlib inline
```

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

```
-rw-r--r-- 1 root root 2.1G Nov 27 07:32 /srv/db/taxi_2016_student_small.
sqlite
```

In [5]:
```
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(*) FR
```

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

In [6]:
```
q1d_query = f"""
        SELECT *
            FROM {TABLE_NAME}
            WHERE tpep_pickup_datetime
                BETWEEN '2016-01-01' AND '2016-12-31'
                AND record_id % 100 == 0
            ORDER BY tpep_pickup_datetime
            """

# YOUR CODE HERE
#raise NotImplementedError()
with timeit(): # This query should take less than 3 second
    q1d_df = pd.read_sql_query(q1d_query, sql_engine)
q1d_df.head()
```

```
4.30 s elapsed
```

Out[6]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	37300	1	2016-01-01 00:02:20	2016-01-01 00:11:58	2	1.20
1	37400	1	2016-01-01 00:03:04	2016-01-01 00:28:54	1	5.00
2	37500	2	2016-01-01 00:03:40	2016-01-01 00:12:47	6	2.54
3	37900	2	2016-01-01 00:05:38	2016-01-01 00:10:02	3	0.76
4	38500	1	2016-01-01 00:07:50	2016-01-01 00:23:42	1	2.40

In [7]:
```
with timeit(): # less than 3 seconds
    final_df = pd.read_sql_query(q1d_query, sql_engine)
final_df['tpep_pickup_datetime'] = pd.to_datetime(final_df['tpep_pickup_dat
final_df['tpep_dropoff_datetime'] = pd.to_datetime(final_df['tpep_dropoff_d
final_df.head()
```

```
4.26 s elapsed
```

Out[7]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	37300	1	2016-01-01 00:02:20	2016-01-01 00:11:58	2	1.20
1	37400	1	2016-01-01 00:03:04	2016-01-01 00:28:54	1	5.00
2	37500	2	2016-01-01 00:03:40	2016-01-01 00:12:47	6	2.54
3	37900	2	2016-01-01 00:05:38	2016-01-01 00:10:02	3	0.76
4	38500	1	2016-01-01 00:07:50	2016-01-01 00:23:42	1	2.40

In [64]:
```
cleaned_final_df = final_df.copy()
```

In [65]:
```
cleaned_final_df['duration'] = cleaned_final_df["tpep_dropoff_datetime"]- c
cleaned_final_df['duration'] = cleaned_final_df['duration'].dt.total_second
cleaned_final_df = cleaned_final_df[cleaned_final_df['duration'] < 12 * 360
```

In [66]:
```
cleaned_final_df = cleaned_final_df[cleaned_final_df['pickup_longitude'] <=
cleaned_final_df = cleaned_final_df[cleaned_final_df['pickup_longitude'] >=
cleaned_final_df = cleaned_final_df[cleaned_final_df['pickup_latitude'] <=
cleaned_final_df = cleaned_final_df[cleaned_final_df['pickup_latitude'] >=
cleaned_final_df = cleaned_final_df[cleaned_final_df['dropoff_longitude'] <
cleaned_final_df = cleaned_final_df[cleaned_final_df['dropoff_longitude'] >
cleaned_final_df = cleaned_final_df[cleaned_final_df['dropoff_latitude'] <=
cleaned_final_df = cleaned_final_df[cleaned_final_df['dropoff_latitude'] >=
cleaned_final_df = cleaned_final_df[cleaned_final_df['passenger_count'] > 0
```

In [67]:
```
from sklearn.model_selection import train_test_split
final_train_df, final_val_df = train_test_split(cleaned_final_df, test_size
```

In [68]:
```
Path("data/extra").mkdir(parents=True, exist_ok=True)
final_data_file = Path("data/extra", "final_cleaned_data.hdf") # Path of hd
final_train_df.to_hdf(final_data_file, "final_train") # Train data of hdf f
final_val_df.to_hdf(final_data_file, "final_val") # Val data of hdf file
```

In [69]:
```
# Run this cell to load the data.
final_data_file = Path("data/extra", "final_cleaned_data.hdf")
final_train_df = pd.read_hdf(final_data_file, "final_train")
```

In [70]:
```
test_df = pd.read_csv("./proj2_test_data.csv")
test_df['tpep_pickup_datetime'] = pd.to_datetime(test_df['tpep_pickup_datet
```

In [71]:
```
# passenger count cannot equal 0
final_train_df = final_train_df[final_train_df['passenger_count'] != 0]
# trip distance has to be greater than 0
final_train_df = final_train_df[final_train_df['trip_distance'] != 0]
# filter out missing latitude/longitude values
final_train_df = final_train_df[final_train_df['pickup_latitude'] != 0]
final_train_df = final_train_df[final_train_df['pickup_longitude'] != 0]
final_train_df = final_train_df[final_train_df['dropoff_latitude'] != 0]
final_train_df = final_train_df[final_train_df['dropoff_longitude'] != 0]
# filter out long duration that doesn't make sense
final_train_df = final_train_df[final_train_df["duration"] < 15000]
# fare amount has to be greater than 0, not equal to 52 (flat rate to the a
final_train_df = final_train_df[final_train_df["fare_amount"] != 52]
final_train_df = final_train_df[final_train_df["fare_amount"] < 75]
final_train_df = final_train_df[final_train_df["fare_amount"] > 0]
# duration has to be at least around 2 minutes but less than 2 hours
final_train_df = final_train_df[final_train_df["duration"] > 90]
final_train_df = final_train_df[final_train_df["duration"] < 7000]
final_train_df.head()
```

Out[71]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_dis
13903	6404500	1	2016-01-18 16:39:48	2016-01-18 16:49:43	1	
121829	56477700	2	2016-05-26 14:41:54	2016-05-26 14:45:55	1	
133961	60619000	2	2016-06-10 20:35:56	2016-06-10 20:43:07	1	
45834	21948700	2	2016-02-27 12:22:30	2016-02-27 12:34:54	5	
50933	23760800	1	2016-03-04 11:31:11	2016-03-04 11:52:53	1	

5 rows × 21 columns

In [72]:
```
final_train = train_df.copy()
# passenger count cannot equal 0
final_train = final_train[final_train['passenger_count'] != 0]
# trip distance has to be greater than 0
final_train = final_train[final_train['trip_distance'] != 0]
# filter out missing latitude/longitude values
final_train = final_train[final_train['pickup_latitude'] != 0]
final_train = final_train[final_train['pickup_longitude'] != 0]
final_train = final_train[final_train['dropoff_latitude'] != 0]
final_train = final_train[final_train['dropoff_longitude'] != 0]
# filter out long duration that doesn't make sense
final_train = final_train[final_train["duration"] < 15000]
# fare amount has to be greater than 0, not equal to 52 (flat rate to the a
final_train = final_train[final_train["fare_amount"] != 52]
final_train = final_train[final_train["fare_amount"] < 75]
final_train = final_train[final_train["fare_amount"] > 0]
# duration has to be at least around 2 minutes but less than 2 hours
final_train = final_train[final_train["duration"] > 90]
final_train = final_train[final_train["duration"] < 7000]
final_train.head()
```

Out[72]:

	record_id	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_dist
13242	5711100	1	2016-01-17 17:48:41	2016-01-17 17:55:53	1	
12723	4989400	1	2016-01-17 01:18:39	2016-01-17 01:21:15	1	
8508	2436400	2	2016-01-12 09:07:00	2016-01-12 09:41:17	1	
21304	10899100	2	2016-01-29 09:07:54	2016-01-29 09:18:25	1	
3817	1319400	1	2016-01-06 11:44:54	2016-01-06 11:49:55	1	

5 rows × 21 columns

In [73]:
```
# 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 *
    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(
    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 + d
    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 [74]:
```
def process_data_gm1(data, test=False):
    # Put your final pipeline here
    X = (
    data

    # Transform data
    .pipe(add_time_columns)
    .pipe(add_distance_columns)
    .pipe(select_columns,
          "trip_distance",
          "fare_amount"

         )
    )
    if test:
        y = None
    else:
        y = data['duration']

    return X, y
```

In [75]:
```python
from sklearn import linear_model
# Jan-June Train
X_train, y_train = process_data_gm1(final_train_df)
X_val, y_val = process_data_gm1(final_val_df)
guided_model_1 = lm.LinearRegression(fit_intercept=True)
guided_model_1.fit(X_train, y_train)

# Jan-June Predict
y_train_pred = guided_model_1.predict(X_train)
y_val_pred = guided_model_1.predict(X_val)

# Jan Train
final_X_train, final_Y_train = process_data_gm1(final_train)
final_X_val, final_Y_val = process_data_gm1(val_df)
final_model = lm.LinearRegression(fit_intercept=True)# Define your final mc
final_model.fit(final_X_train, final_Y_train);
# Jan Predict
train_predict = final_model.predict(final_X_train)
val_predict = final_model.predict(final_X_val)
```

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

In [76]:
```python
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 [77]:
```python
print("Validation Error: for January-June =", mae(y_val_pred, y_val))
print("Validation Error: for January", mae(val_predict, final_Y_val))
```

```
Validation Error: for January-June = 165.825984848
Validation Error: for January 95.0052854726
```

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.

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**