Homework 2: Bike Sharing

Exploratory Data Analysis (EDA) and Visualization

Due Date: Friday 7/5, 11:59 PM

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** below.

Collaborators: *list collaborators here*

Introduction

Bike sharing systems are new generation of traditional bike rentals where the process of signing up, renting and returning is automated. Through these systems, users are able to easily rent a bike from one location and return them to another. We will be analyzing bike sharing data from Washington D.C.

In this assignment, you will perform tasks to clean, visualize, and explore the bike sharing data. You will also investigate open-ended questions. These open-ended questions ask you to think critically about how the plots you have created provide insight into the data.

After completing this assignment, you should be comfortable with:

- reading plaintext delimited data into `pandas`
- wrangling data for analysis
- using EDA to learn about your data
- making informative plots

Grading

Grading is broken down into autograded answers and free response.

For autograded answers, the results of your code are compared to provided and/or hidden tests.

For free response, readers will evaluate how well you answered the question and/or fulfilled the requirements of the question.

For plots, your plots should be *similar* to the given examples. We will tolerate small variations such as color differences or slight variations in scale. However it is in your best interest to make the plots as similar as possible, as similarity is subject to the readers.

Note that for ALL plotting questions from here on out, we will expect appropriate titles, axis labels, legends, etc. The following question serves as a good guideline on what is "enough": If I directly downloaded the plot and viewed it, would I be able to tell what was being visualized without knowing the question?

Score breakdown

Question	Points
Question 1a	2
Question 1b	1
Question 1c	2
Question 2a	2
Question 2b	2
Question 2c	1
Question 2d	1
Question 2e	2
Question 2f	2
Question 3a	5
Question 3b	3
Question 4	2
Question 5a	2
Question 5b	2
Question 6a	1
Question 6b	4
Question 6c	2
Question 6d	2
Total	38

In [1]:

```
# Run this cell to set up your notebook.  Make sure ds100_utils.py is in this as
signment's folder
import seaborn as sns
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import zipfile
from pathlib import Path
import ds100_utils

# Default plot configurations
%matplotlib inline
plt.rcParams['figure.figsize'] = (16,8)
plt.rcParams['figure.dpi'] = 150
sns.set()

from IPython.display import display, Latex, Markdown
```

Loading Bike Sharing Data

The data we are exploring is collected from a bike sharing system in Washington D.C.

The variables in this data frame are defined as:

Variable	**Description**
instant	record index
dteday	date
season	1. spring 2. summer 3. fall 4. winter
yr	year (0: 2011, 1:2012)
mnth	month (1 to 12)
hr	hour (0 to 23)
holiday	whether day is holiday or not
weekday	day of the week
workingday	if day is neither weekend nor holiday
weathersit	1. clear or partly cloudy 2. mist and clouds 3. light snow or rain 4. heavy rain or snow
temp	normalized temperature in Celsius (divided by 41)
atemp	normalized "feels-like" temperature in Celsius (divided by 50)
hum	normalized percent humidity (divided by 100)
windspeed	normalized wind speed (divided by 67)
casual	count of casual users
registered	count of registered users
cnt	count of total rental bikes including casual and registered

Download the Data

In [2]:

```
# Run this cell to download the data.  No further action is needed

data_url = 'https://github.com/DS-100/su19/raw/gh-pages/assets/datasets/hw2-bike
share.zip'
file_name = 'data.zip'
data_dir = '.'

dest_path = ds100_utils.fetch_and_cache(data_url=data_url, data_dir=data_dir, fi
le=file_name)
print('Saved at {}'.format(dest_path))

zipped_data = zipfile.ZipFile(dest_path, 'r')

data_dir = Path('data')
zipped_data.extractall(data_dir)


print("Extracted Files:")
for f in data_dir.glob("*"):
    print("\t",f)
```

```
Using version already downloaded: Fri Jun 21 11:20:36 2019
MD5 hash of file: 2bcd2ca89278a8230f4e9461455c0811
Saved at data.zip
Extracted Files:
        data/bikeshare.txt
```

Examining the file contents

Can you identify the file format? (No answer required.)

In [3]:

```
# Run this cell to look at the top of the file.  No further action is needed
for line in ds100_utils.head(data_dir/'bikeshare.txt'):
    print(line,end="")
```

```
instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathers
it,temp,atemp,hum,windspeed,casual,registered,cnt
1,2011-01-01,1,0,1,0,0,6,0,1,0.24,0.2879,0.81,0,3,13,16
2,2011-01-01,1,0,1,1,0,6,0,1,0.22,0.2727,0.8,0,8,32,40
3,2011-01-01,1,0,1,2,0,6,0,1,0.22,0.2727,0.8,0,5,27,32
4,2011-01-01,1,0,1,3,0,6,0,1,0.24,0.2879,0.75,0,3,10,13
```

Size

Is the file big? How many records do we expect to find? (No answers required.)

In [4]:

```
# Run this cell to view some metadata.  No further action is needed
print("Size:", (data_dir/"bikeshare.txt").stat().st_size, "bytes")
print("Line Count:", ds100_utils.line_count(data_dir/"bikeshare.txt"), "lines")
```

```
Size: 1156736 bytes
Line Count: 17380 lines
```

Loading the data

The following code loads the data into a Pandas DataFrame.

In [5]:

```
# Run this cell to load the data.  No further action is needed
bike = pd.read_csv(data_dir/'bikeshare.txt')
bike.head()
```

Out[5]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	ter
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.2
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.2
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.2
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.2
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.2

Below, we show the shape of the file. You should see that the size of the DataFrame matches the number of lines in the file, minus the header row.

In [6]:

```
bike.shape
```

Out[6]:

```
(17379, 17)
```

1: Data Preparation

A few of the variables that are numeric/integer actually encode categorical data. These include `holiday`, `weekday`, `workingday`, and `weathersit`. In the following problem, we will convert these four variables to strings specifying the categories. In particular, use 3-letter labels (`Sun`, `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, and `Sat`) for `weekday`. You may simply use `yes`/`no` for `holiday` and `workingday`.

In this exercise we will *mutate* the data frame, **overwriting the corresponding variables in the data frame.** However, our notebook will effectively document this in-place data transformation for future readers. Make sure to leave the underlying datafile `bikeshare.txt` unmodified.

Question 1

Question 1a (Decoding `weekday`, `workingday`, and `weathersit`)

Decode the `holiday`, `weekday`, `workingday`, and `weathersit` fields:

1. `holiday`: Convert to `yes` and `no`. Hint: There are fewer holidays...
2. `weekday`: It turns out that Monday is the day with the most holidays. Mutate the `'weekday'` column to use the 3-letter label (`'Sun'`, `'Mon'`, `'Tue'`, `'Wed'`, `'Thu'`, `'Fri'`, and `'Sat'` ...) instead of its current numerical values. Assume `0` corresponds to `Sun`, `1` to `Mon` and so on.
3. `workingday`: Convert to `yes` and `no`.
4. `weathersit`: You should replace each value with one of `Clear`, `Mist`, `Light`, or `Heavy`.

Note: If you want to revert changes, run the cell that reloads the csv.

Hint: One approach is to use the replace (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html) method of the pandas DataFrame class. We haven't discussed how to do this so you'll need to look at the documentation. The most concise way is with the approach described in the documentation as "nested-dictonaries", though there are many possible solutions.

```
BEGIN QUESTION
name: q1a
points: 2
```

In [7]:

```
# Modify holiday weekday, workingday, and weathersit here
# BEGIN SOLUTION
factor_dict = {
    'holiday': {
        0:'no',
        1:'yes'
    },
    'weekday': {
        0: 'Sun',
        1: 'Mon',
        2: 'Tue',
        3: 'Wed',
        4: 'Thu',
        5: 'Fri',
        6: 'Sat'
    },
    'workingday': {
        0: 'no',
        1: 'yes'
    },
    'weathersit': {
        1: 'Clear',
        2: 'Mist',
        3: 'Light',
        4: 'Heavy'
    }
}

bike.replace(factor_dict, inplace=True)
bike#.head()
# END SOLUTION
```

Out[7]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersi
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear

5	6	2011-01-01	1	0	1	5	no	Sat	no	Mist
6	7	2011-01-01	1	0	1	6	no	Sat	no	Clear
7	8	2011-01-01	1	0	1	7	no	Sat	no	Clear
8	9	2011-01-01	1	0	1	8	no	Sat	no	Clear
9	10	2011-01-01	1	0	1	9	no	Sat	no	Clear
10	11	2011-01-01	1	0	1	10	no	Sat	no	Clear
11	12	2011-01-01	1	0	1	11	no	Sat	no	Clear
12	13	2011-01-01	1	0	1	12	no	Sat	no	Clear
13	14	2011-01-01	1	0	1	13	no	Sat	no	Mist
14	15	2011-01-01	1	0	1	14	no	Sat	no	Mist
15	16	2011-01-01	1	0	1	15	no	Sat	no	Mist
16	17	2011-01-01	1	0	1	16	no	Sat	no	Mist
17	18	2011-01-01	1	0	1	17	no	Sat	no	Mist
18	19	2011-01-01	1	0	1	18	no	Sat	no	Light
19	20	2011-01-01	1	0	1	19	no	Sat	no	Light
20	21	2011-01-01	1	0	1	20	no	Sat	no	Mist
21	22	2011-01-01	1	0	1	21	no	Sat	no	Mist
22	23	2011-01-01	1	0	1	22	no	Sat	no	Mist
		2011-								

23	24	01-01	1	0	1	23	no	Sat	no	Mist
24	25	2011-01-02	1	0	1	0	no	Sun	no	Mist
25	26	2011-01-02	1	0	1	1	no	Sun	no	Mist
26	27	2011-01-02	1	0	1	2	no	Sun	no	Mist
27	28	2011-01-02	1	0	1	3	no	Sun	no	Mist
28	29	2011-01-02	1	0	1	4	no	Sun	no	Mist
29	30	2011-01-02	1	0	1	6	no	Sun	no	Light
...	...	...	...	...	...	...	...	...	...	...
17349	17350	2012-12-30	1	1	12	18	no	Sun	no	Mist
17350	17351	2012-12-30	1	1	12	19	no	Sun	no	Clear
17351	17352	2012-12-30	1	1	12	20	no	Sun	no	Clear
17352	17353	2012-12-30	1	1	12	21	no	Sun	no	Clear
17353	17354	2012-12-30	1	1	12	22	no	Sun	no	Clear
17354	17355	2012-12-30	1	1	12	23	no	Sun	no	Clear
17355	17356	2012-12-31	1	1	12	0	no	Mon	yes	Clear
17356	17357	2012-12-31	1	1	12	1	no	Mon	yes	Clear
17357	17358	2012-12-31	1	1	12	2	no	Mon	yes	Clear
17358	17359	2012-12-31	1	1	12	3	no	Mon	yes	Clear
17359	17360	2012-12-31	1	1	12	4	no	Mon	yes	Clear

17360	17361	2012-12-31	1	1	12	5	no	Mon	yes	Clear
17361	17362	2012-12-31	1	1	12	6	no	Mon	yes	Clear
17362	17363	2012-12-31	1	1	12	7	no	Mon	yes	Clear
17363	17364	2012-12-31	1	1	12	8	no	Mon	yes	Clear
17364	17365	2012-12-31	1	1	12	9	no	Mon	yes	Mist
17365	17366	2012-12-31	1	1	12	10	no	Mon	yes	Mist
17366	17367	2012-12-31	1	1	12	11	no	Mon	yes	Mist
17367	17368	2012-12-31	1	1	12	12	no	Mon	yes	Mist
17368	17369	2012-12-31	1	1	12	13	no	Mon	yes	Mist
17369	17370	2012-12-31	1	1	12	14	no	Mon	yes	Mist
17370	17371	2012-12-31	1	1	12	15	no	Mon	yes	Mist
17371	17372	2012-12-31	1	1	12	16	no	Mon	yes	Mist
17372	17373	2012-12-31	1	1	12	17	no	Mon	yes	Mist
17373	17374	2012-12-31	1	1	12	18	no	Mon	yes	Mist
17374	17375	2012-12-31	1	1	12	19	no	Mon	yes	Mist
17375	17376	2012-12-31	1	1	12	20	no	Mon	yes	Mist
17376	17377	2012-12-31	1	1	12	21	no	Mon	yes	Clear
17377	17378	2012-12-31	1	1	12	22	no	Mon	yes	Clear
17378	17379	2012-	1	1	12	23	no	Mon	yes	Clear

		12-31								

17379 rows × 17 columns

In [8]:

```
# TEST
isinstance(bike, pd.DataFrame)
```

Out[8]:

```
True
```

In [9]:

```
# TEST
bike['holiday'].dtype == np.dtype('O')
```

Out[9]:

```
True
```

In [10]:

```
# TEST
list(bike['holiday'].iloc[370:375]) == ['no', 'no', 'yes', 'yes', 'yes']
```

Out[10]:

```
True
```

In [11]:

```
# TEST
bike['weekday'].dtype == np.dtype('O')
```

Out[11]:

```
True
```

In [12]:

```
# TEST
bike['workingday'].dtype == np.dtype('O')
```

Out[12]:

```
True
```

In [13]:

```
# TEST
bike['weathersit'].dtype == np.dtype('O')
```

Out[13]:

```
True
```

In [14]:

```
# HIDDEN TEST
bike.shape == (17379, 17) or bike.shape == (17379, 18)
```

Out[14]:

```
True
```

In [15]:

```
# HIDDEN TEST
print(list(bike['weekday'].iloc[::2000]))
```

```
['Sat', 'Tue', 'Mon', 'Mon', 'Mon', 'Sun', 'Sun', 'Sat', 'Sun']
```

In [16]:

```
# HIDDEN TEST
print(list(bike['workingday'].iloc[::2000]))
```

```
['no', 'yes', 'yes', 'yes', 'yes', 'no', 'no', 'no', 'no']
```

In [17]:

```
# HIDDEN TEST
print(list(bike['weathersit'].iloc[::2000]))
```

```
['Clear', 'Clear', 'Clear', 'Clear', 'Clear', 'Clear', 'Clear', 'Cle
ar', 'Clear']
```

Question 1b (Holidays)

How many entries in the data correspond to holidays? Set the variable `num_holidays` to this value.

```
BEGIN QUESTION
name: q1b
points: 1
```

In [18]:

```
num_holidays = bike['holiday'].value_counts()['yes'] # SOLUTION
```

In [19]:

```
# TEST
1 <= num_holidays <= 10000
```

Out[19]:

```
True
```

In [20]:

```
# HIDDEN TEST
num_holidays == 500
```

Out[20]:

```
True
```

Question 1c (Computing Daily Total Counts)

The granularity of this data is at the hourly level. However, for some of the analysis we will also want to compute daily statistics. In particular, in the next few questions we will be analyzing the daily number of registered and unregistered users.

Construct a data frame named `daily_counts` indexed by `dteday` with the following columns:

- `casual`: total number of casual riders for each day
- `registered`: total number of registered riders for each day
- `workingday`: whether that day is a working day or not (`yes` or `no`)

Hint: `groupby` and `agg`. For the `agg` method, please check the documentation (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.core.groupby.DataFrameGroupBy.agg.html) for examples on applying different aggregations per column. If you use the capability to do different aggregations by column, you can do this task with a single call to `groupby` and `agg`. For the `workingday` column we can take any of the values since we are grouping by the day, thus the value will be the same within each group. Take a look at the `'first'` or `'last'` aggregation functions.

```
BEGIN QUESTION
name: q1c
points: 2
```

In [21]:

```
daily_counts = ...
# BEGIN SOLUTION NO PROMPT
daily_counts = (
    bike
    .groupby(['dteday'])
    .agg(
        {
        "casual": sum,
        "registered": sum,
        "workingday": 'last'
        }
    )
)
daily_counts.head()
# END SOLUTION
```

Out[21]:

	casual	registered	workingday
dteday			
2011-01-01	331	654	no
2011-01-02	131	670	no
2011-01-03	120	1229	yes
2011-01-04	108	1454	yes
2011-01-05	82	1518	yes

In [22]:

```
# TEST
np.round(daily_counts['casual'].mean())
```

Out[22]:

```
848.0
```

In [23]:

```
# TEST
np.round(daily_counts['casual'].var())
```

Out[23]:

```
471450.0
```

In [24]:

```
# HIDDEN TEST
np.round(daily_counts['registered'].mean())
```

Out[24]:

```
3656.0
```

In [25]:

```
# HIDDEN TEST
np.round(daily_counts['registered'].var())
```

Out[25]:

```
2434400.0
```

In [26]:

```
# HIDDEN TEST
sorted(list(daily_counts['workingday'].value_counts()))
```

Out[26]:

```
[231, 500]
```

2: Exploring the Distribution of Riders

Let's begin by comparing the distribution of the daily counts of casual and registered riders.

Question 2

Question 2a

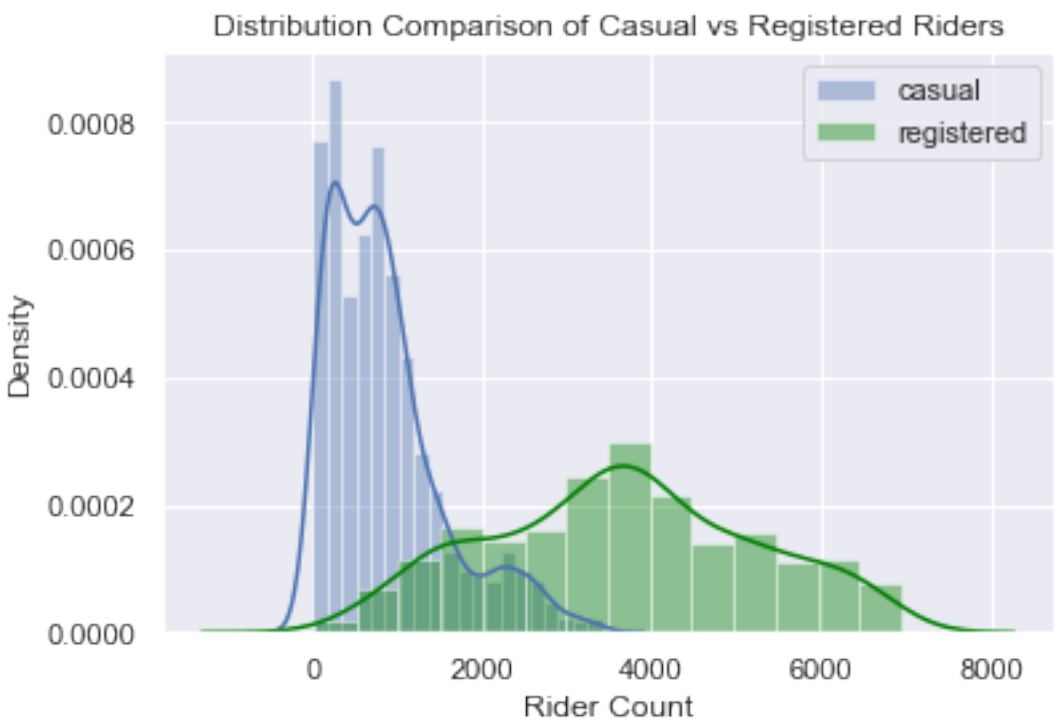
Use the `sns.distplot` (https://seaborn.pydata.org/generated/seaborn.distplot.html) function to create a plot that overlays the distribution of the daily counts of `casual` and `registered` users. The temporal granularity of the records should be daily counts, which you should have after completing question 1c.

Include a legend, xlabel, ylabel, and title. Read the seaborn plotting tutorial (https://seaborn.pydata.org/tutorial/distributions.html) if you're not sure how to add these. After creating the plot, look at it and make sure you understand what the plot is actually telling us, e.g on a given day, the most likely number of registered riders we expect is ~4000, but it could be anywhere from nearly 0 to 7000.

```
BEGIN QUESTION
name: q2a
points: 2
manual: true
```

In [27]:

```
# BEGIN SOLUTION
sns.distplot(daily_counts['casual'], label='casual')
sns.distplot(daily_counts['registered'],  label='registered', color='green')
plt.legend()
plt.title("Distribution Comparison of Casual vs Registered Riders")
plt.xlabel("Rider Count")
plt.ylabel("Density");
# plt.savefig("images/casual_v_registered.png", bbox_inches='tight', dpi=300);
# END SOLUTION
```



Question 2b

In the cell below, descibe the differences you notice between the density curves for casual and registered riders. Consider concepts such as modes, symmetry, skewness, tails, gaps and outliers. Include a comment on the spread of the distributions.

```
BEGIN QUESTION
name: q2b
points: 2
manual: true
```

SOLUTION: The casual riders distribution has a sharp peak at 1000 that may be bimodal. This distribution is skewed right and has a long right tail with a small set of daily counts around 2500. The distribution of registered riders has a more symmetric distribution centered around almost 4000 daily riders. This distribution does not have heavy skew. Its spread is much wider than the casual riders.

Ethical Considerations

City planners, transportation agencies, and policy makers have started to collaborate with bike sharing companies in order to reduce congestion and transportation costs. Recently city planners and policy makers have also been trying to make transportation more equitable.

Equity in transportation includes: finding ways to make transportation more accessible to people in all neighborhoods within a given region, making the costs of transportation affordable to people across all income levels, and assessing how inclusive transportation systems are over time. Data about city residents may shed light on how to better assess transportation cost and equity impacts on transportation users. Bearing this in mind, answer the following two questions on the nature of the data, their possible shortcomings, and ethical considerations associated with how we as data scientists use, manipulate, and share this data.

Question 2c

In addition to the type of rider (casual vs. registered) and the overall count of each, what other kinds of demographic data would be useful (e.g. identity, neighborhood, monetary expenses, etc.)?

What is an example of a privacy or consent issue that could occur when accessing the demographic data you brought up in the previous question?

```
BEGIN QUESTION
name: q2c
points: 1
manual: true
```

SOLUTION: This question is somewhat broad and open-ended, credit will be awarded leniently. Possible data that could improve the level of demographic analysis would be information on registered rider costs for bike sharing along with their expenses on other forms of transportation, zip codes of bank accounts and credit cards used to pay for the service (or home address), household income, race, ethnicity, gender, age, etc. from riders

Question 2d

What is an example of a privacy or consent issue that could occur when accessing the demographic data you brought up in the previous question?

```
BEGIN QUESTION
name: q2d
points: 1
manual: true
```

SOLUTION: A comprehensive answer should inculde factors and considerations such as, consequences of using different techniques to gather this data (e.g. who opts in to surveys or sharing their data; what geographical locations to sample / at what time of day), how companies share data with each other or other organizations, preserving user anonymity, how lack of anonymity could allow companies to piece together more information about people they were not intended to have, getting access to the demographic data (e.g. having credit card companies, organizations that have this data etc. to allow the data to be shared)

Question 2e

The density plots do not show us how the counts for registered and casual riders vary together. Use `sns.lmplot` (https://seaborn.pydata.org/generated/seaborn.lmplot.html) to make a scatter plot to investigate the relationship between casual and registered counts. This time, let's use the `bike` DataFrame to plot hourly counts instead of daily counts.

The `lmplot` function will also try to draw a linear regression line (just as you saw in Data 8). Color the points in the scatterplot according to whether or not the day is working day. There are many points in the scatter plot so make them small to help reduce overplotting. Also make sure to set `fit_reg=True` to generate the linear regression line. You can set the `height` parameter if you want to adjust the size of the `lmplot`. Make sure to include a title.

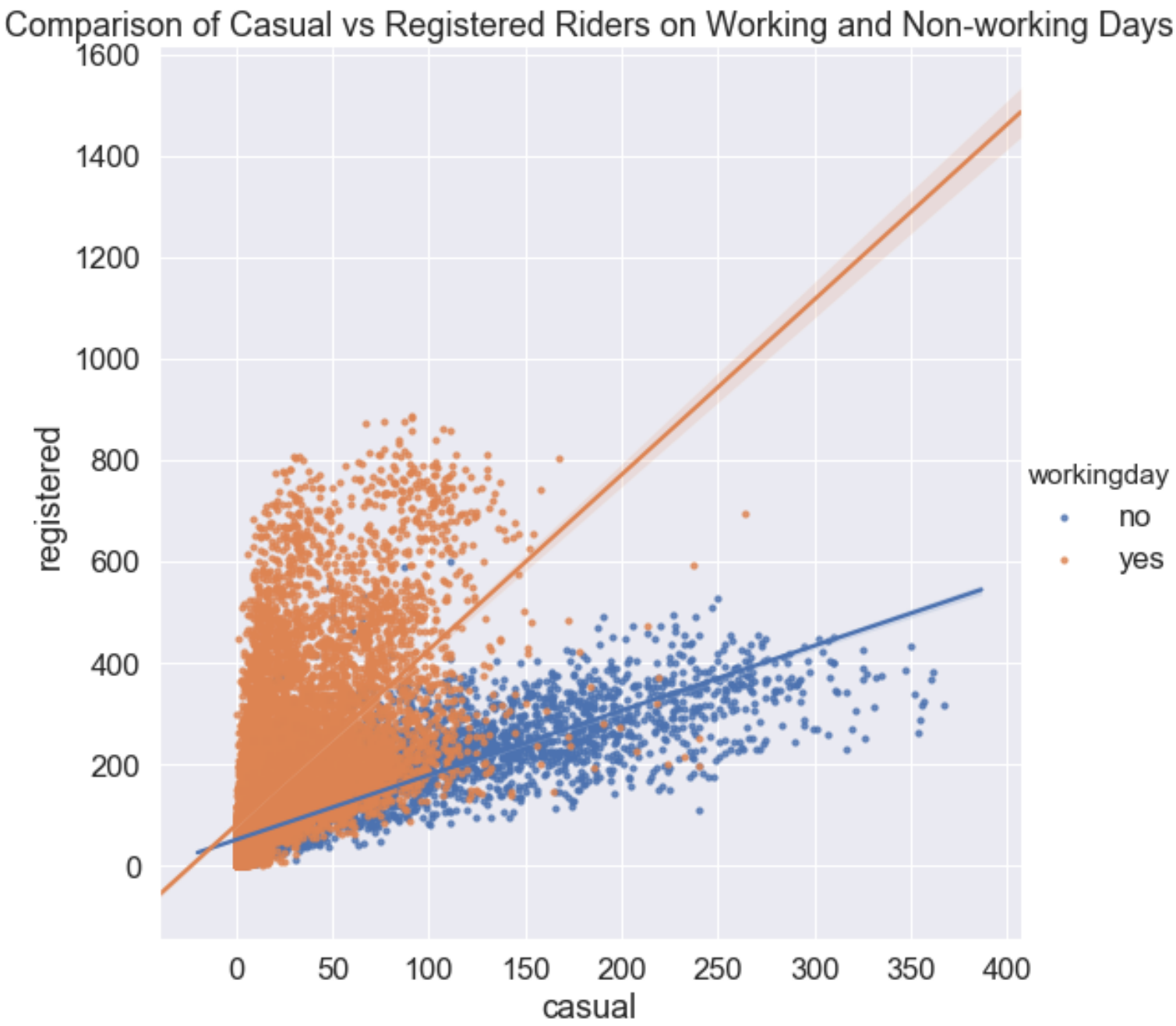
Hints:

- Checkout this helpful tutorial on `lmplot` (https://seaborn.pydata.org/tutorial/regression.html).
- You will need to set `x`, `y`, and `hue` and the `scatter_kws`.

```
BEGIN QUESTION
name: q2e
points: 2
manual: true
```

In [28]:

```
# Make the font size a bit bigger
sns.set(font_scale=1.5)
# BEGIN SOLUTION
sns.lmplot(x="casual", y="registered", hue="workingday",
           data=bike, fit_reg=True, height=8, scatter_kws={"s": 10})
plt.title("Comparison of Casual vs Registered Riders on Working and Non-working
Days");
# plt.savefig("images/casual_registered_working_nonworking.png", bbox_inches='ti
ght', dpi=300);
# END SOLUTION
```



Question 2f

What does this scatterplot seem to reveal about the relationship (if any) between casual and registered riders and whether or not the day is on the weekend? What effect does overplotting (http://www.textbook.ds100.org/ch/06/viz_principles_2.html) have on your ability to describe this relationship?

```
BEGIN QUESTION
name: q2f
points: 2
manual: true
```

SOLUTION: There appears to be a linear relationship between the counts for registered and casual riders, and this relationship depends on whether the day is a work day or a weekend day. Due to overplotting, it's not possible to see the shape of the blue dataset (workingday=no) because it is occluded, and it is also difficult to determine the degree to which values are concentrated around the regression lines near 0.

3: Visualization

Question 3

Question 3a Bivariate Kernel Density Plot

To address overplotting, let's try visualizing the data with another technique, the bivariate kernel density estimate.

You will want to read up on the documentation for `sns.kdeplot` which can be found at https://seaborn.pydata.org/generated/seaborn.kdeplot.html (https://seaborn.pydata.org/generated/seaborn.kdeplot.html)

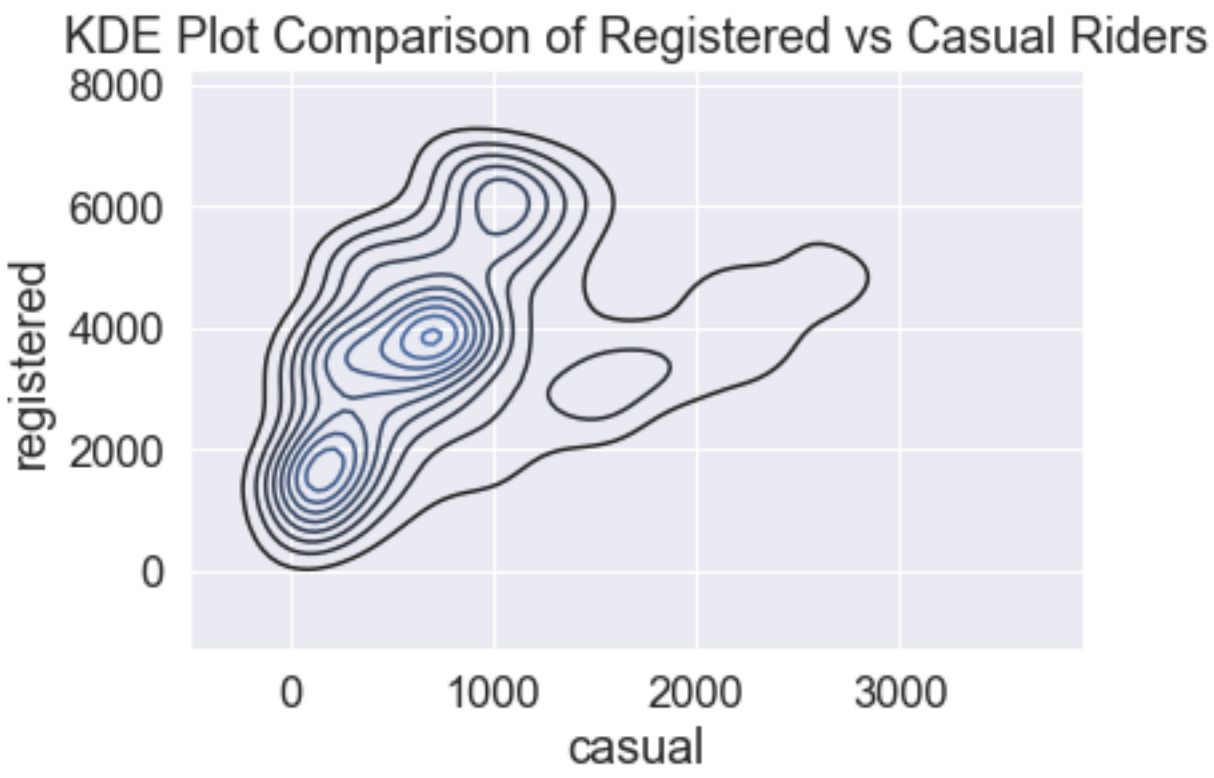
The result we wish to achieve should be a plot that looks like this:

You can think of this plot as an overhead countour or topographical map, where the "high" regions are those with more data points, and "low" regions are those with fewer data points.

A basic kde plot of all the data is quite easy to generate. However, this plot includes both weekend and weekday data, which isn't what we want (see example figure above).

In [29]:

```
sns.kdeplot(daily_counts['casual'], daily_counts['registered'])
plt.title('KDE Plot Comparison of Registered vs Casual Riders');
```



Generating the plot with weekend and weekday separated can be complicated so we will provide a walkthrough below, feel free to use whatever method you wish however if you do not want to follow the walkthrough.

Hints:

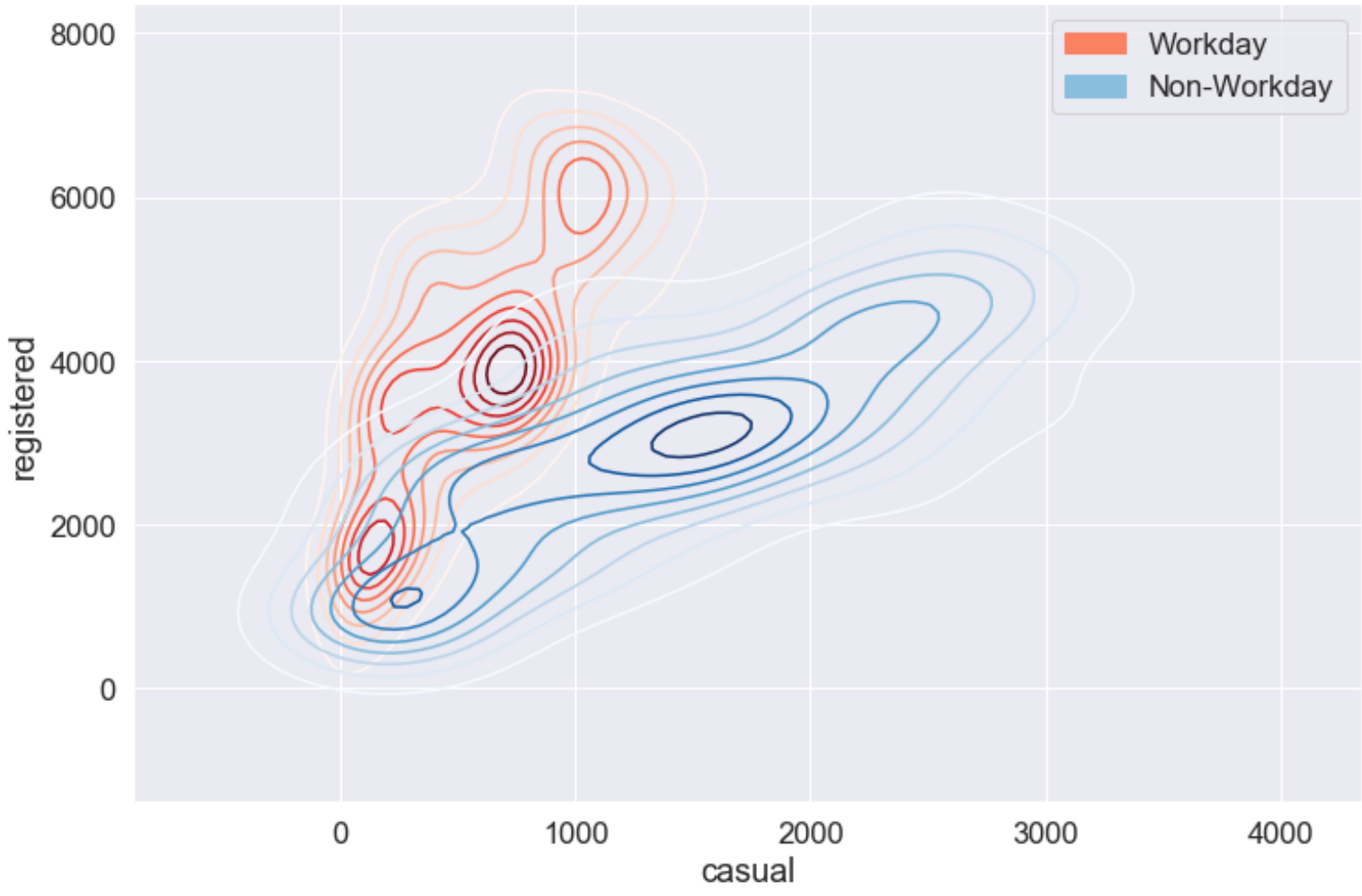
- You can use `loc` with a boolean array and column names at the same time
- You will need to call kdeplot twice.
- Check out this tutorial (http://financeandpython.com/SeabornDataVisualization/8/3.html) to see an example of how to set colors for each dataset and how to create a legend. The legend part uses some weird matplotlib syntax that we haven't learned! You'll probably find creating the legend annoying, but it's a good exercise to learn how to use examples to get the look you want.
- You will want to set the `cmap` parameter of `kdeplot` to `"Reds"` and `"Blues"` (or whatever two contrasting colors you'd like).

After you get your plot working, experiment by setting `shade=True` in `kdeplot` to see the difference between the shaded and unshaded version. Please submit your work with `shade=False`.

```
BEGIN QUESTION
name: q3a
points: 5
manual: true
```

In [30]:

```
import matplotlib.patches as mpatches  # see the tutorial for how we use mpatche
s to generate this figure!

# Set 'is_workingday' to a boolean array that is true for all working_days
is_workingday = ...

# Bivariate KDEs require two data inputs.
# In this case, we will need the daily counts for casual and registered riders o
n weekdays
# Hint: use loc and is_workingday to splice out the relevant rows and column (ca
sual/registered).
casual_weekday = ...
registered_weekday = ...

# Use sns.kdeplot on the two variables above to plot the bivariate KDE for weekd
ay rides
...

# Repeat the same steps above but for rows corresponding to non-workingdays
casual_weekend = ...
registered_weekend = ...

# Use sns.kdeplot on the two variables above to plot the bivariate KDE for weekd
ay rides
# BEGIN SOLUTION
plt.figure(figsize=(12,8))
is_workingday = daily_counts['workingday'] == 'yes'
casual_weekday = daily_counts.loc[is_workingday, 'casual']
registered_weekday = daily_counts.loc[is_workingday, 'registered']

sns.kdeplot(casual_weekday, registered_weekday, cmap="Reds")

not_workingday = ~is_workingday
casual_weekend = daily_counts.loc[not_workingday, 'casual']
registered_weekend = daily_counts.loc[not_workingday, 'registered']

sns.kdeplot(casual_weekend, registered_weekend, cmap="Blues")
r = sns.color_palette("Reds")[2]
b = sns.color_palette("Blues")[2]

red_patch = mpatches.Patch(color=r, label='Workday')
blue_patch = mpatches.Patch(color=b, label='Non-Workday')

plt.legend(handles=[red_patch,blue_patch]);
# plt.savefig("images/bivariate_kde_of_daily_rider_types.png", bbox_inches='tigh
t', dpi=300);
# END SOLUTION
```



Question 3b

What additional details can you identify from this contour plot that were difficult to determine from the scatter plot?

```
BEGIN QUESTION
name: q3b
points: 3
manual: true
```

SOLUTION: The association between registered and casual riders appears linear for both categories of days, but with a much higher slope for workdays. We can see from the contour plot that the variability is higher on non-wordays, and the non-workday joint distribution is bimodal. The workday joint distribution appears to be trimodal.

4: Joint Plot

As an alternative approach to visualizing the data, construct the following set of three plots where the main plot shows the contours of the kernel density estimate of daily counts for registered and casual riders plotted together, and the two "margin" plots (at the top and right of the figure) provide the univariate kernel density estimate of each of these variables. Note that this plot makes it harder see the linear relationships between casual and registered for the two different conditions (weekday vs. weekend).

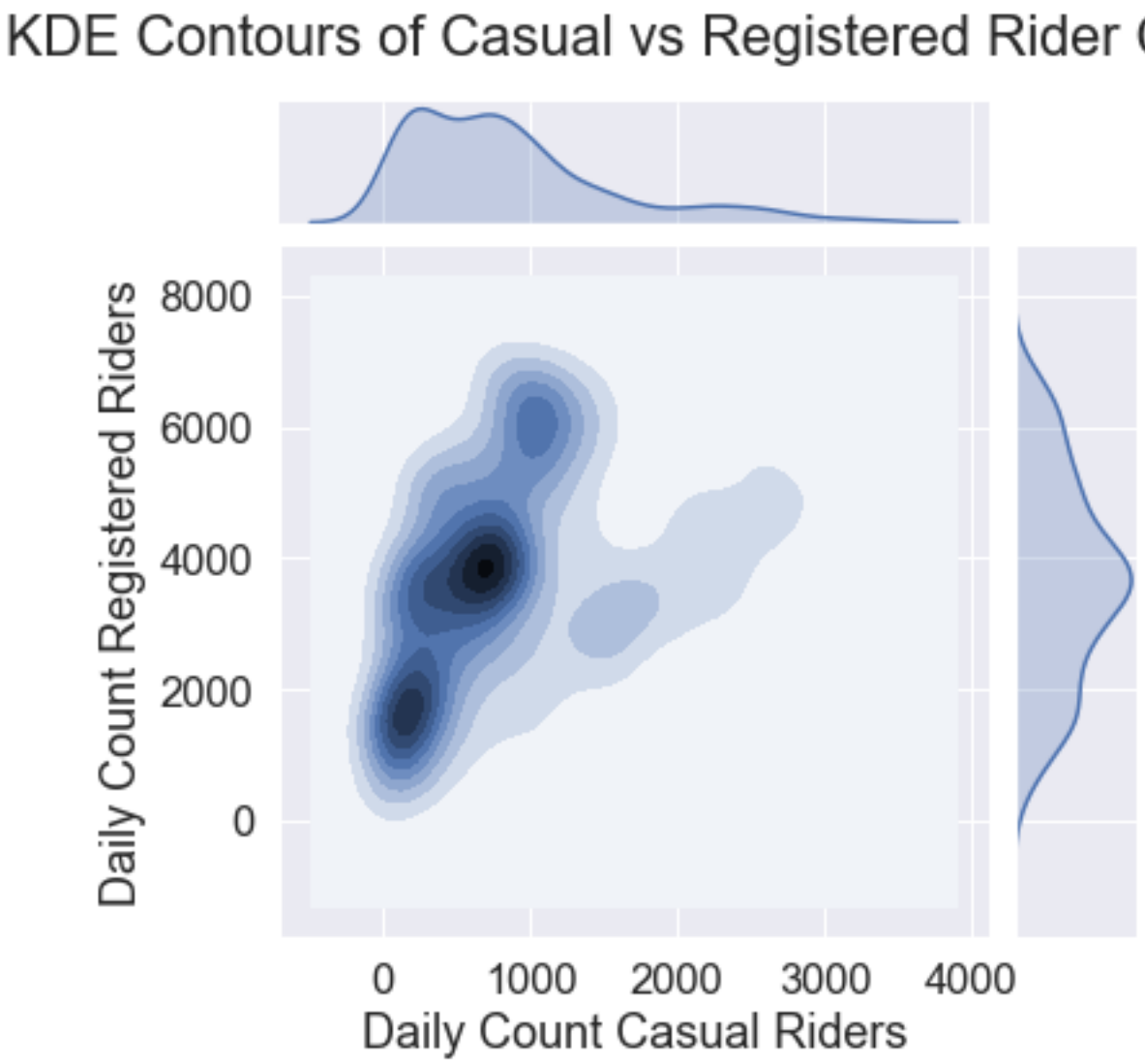
Hints:

- The seaborn plotting tutorial (https://seaborn.pydata.org/tutorial/distributions.html) has examples that may be helpful.
- Take a look at `sns.jointplot` and its `kind` parameter.
- `set_axis_labels` can be used to rename axes on the contour plot.
- `plt.suptitle` from lab 1 can be handy for setting the title where you want.
- `plt.subplots_adjust(top=0.9)` can help if your title overlaps with your plot

```
BEGIN QUESTION
name: q4
points: 2
manual: true
```

In [31]:

```
# BEGIN SOLUTION
g = sns.jointplot(x="casual", y="registered", data=daily_counts, kind="kde");
g.set_axis_labels("Daily Count Casual Riders", "Daily Count Registered Riders")
plt.suptitle("KDE Contours of Casual vs Registered Rider Count")
plt.subplots_adjust(top=0.9);
# plt.savefig("images/joint_distribution_of_daily_rider_types.png", bbox_inches=
'tight', dpi=300);
# END SOLUTION
```



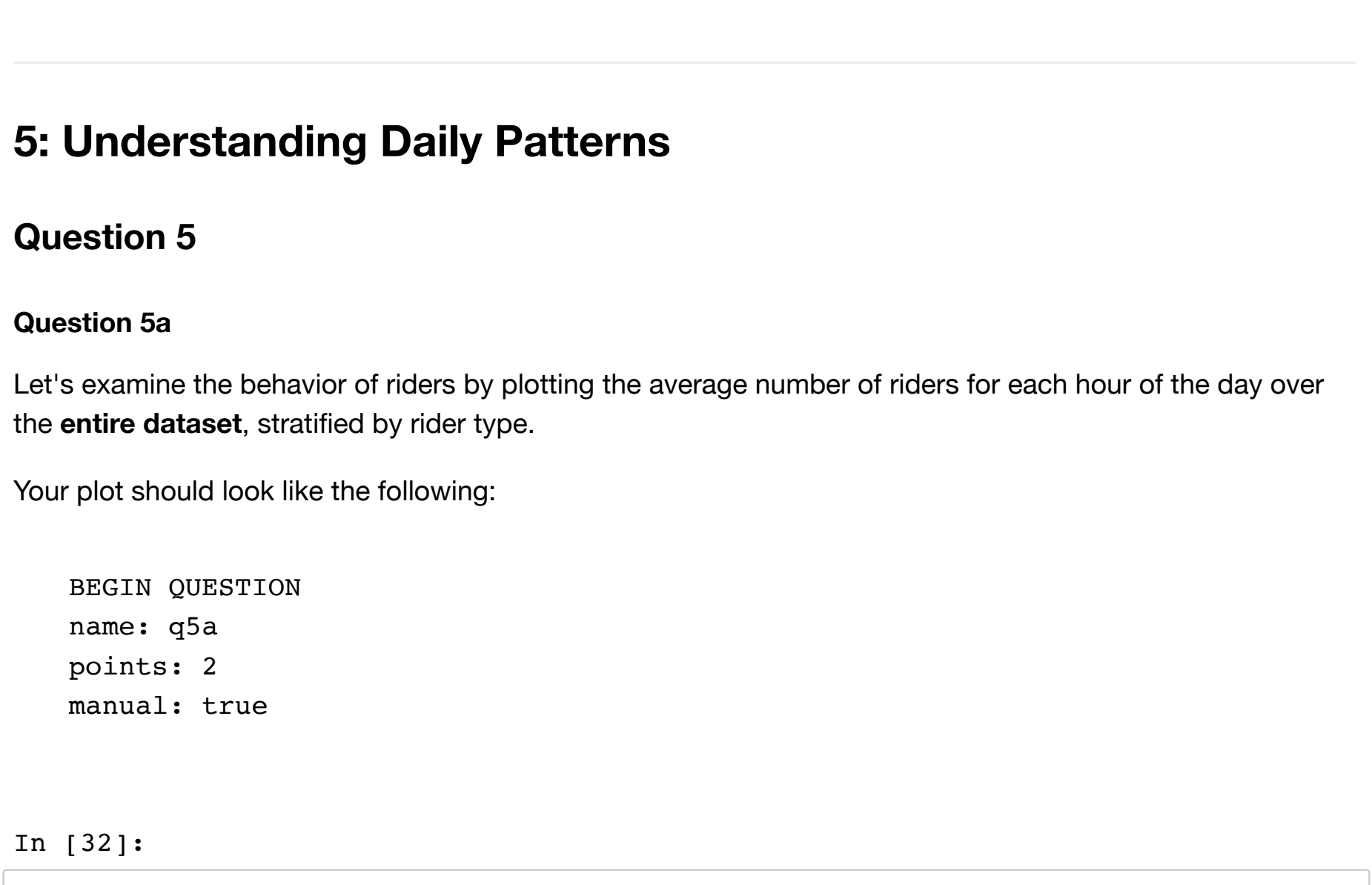
5: Understanding Daily Patterns

Question 5

Question 5a

Let's examine the behavior of riders by plotting the average number of riders for each hour of the day over the **entire dataset**, stratified by rider type.

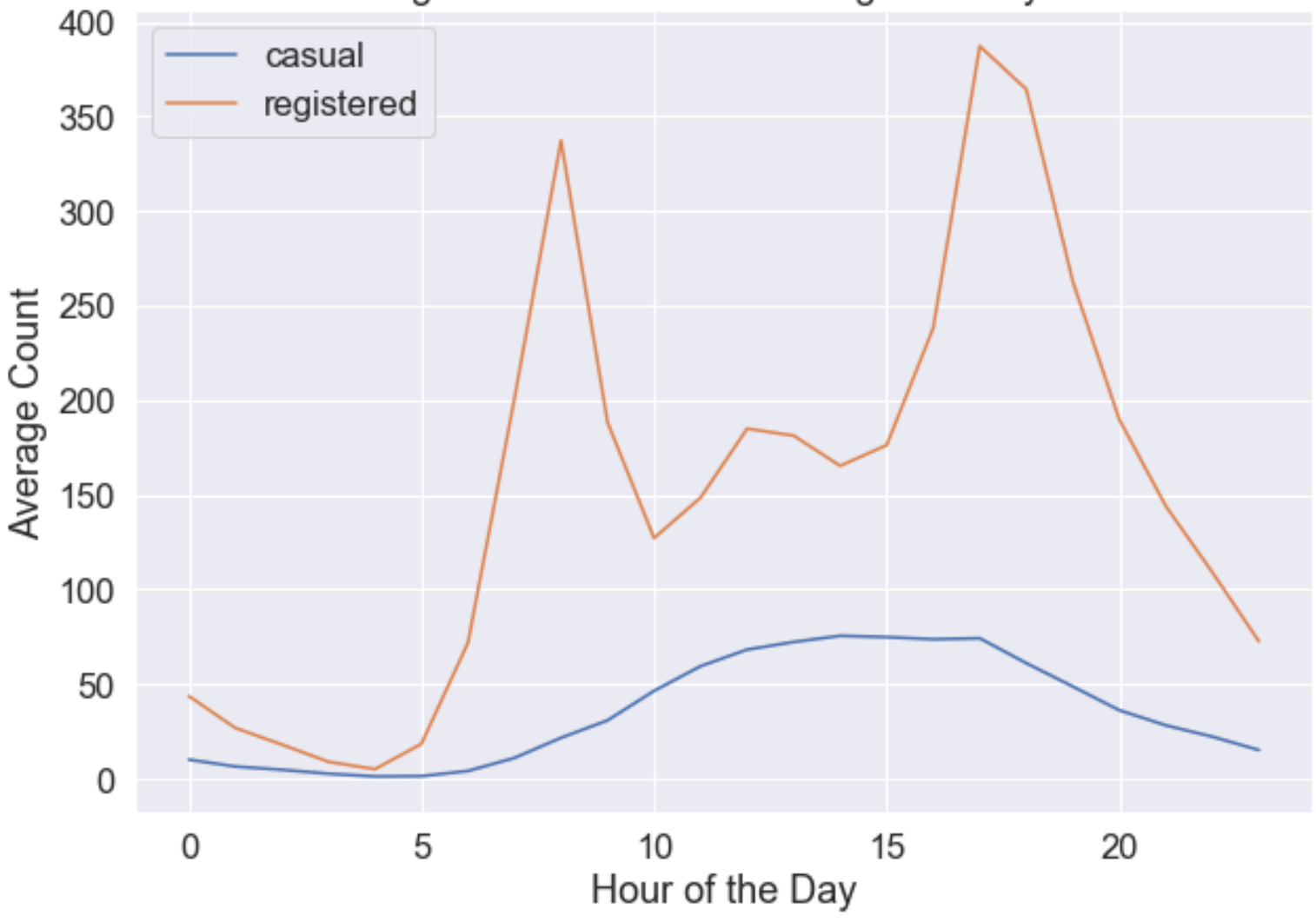
Your plot should look like the following:



```
BEGIN QUESTION
name: q5a
points: 2
manual: true
```

In [32]:

```
# BEGIN SOLUTION
plt.figure(figsize=(10, 7))
hourly_means = bike.groupby('hr').mean()
sns.lineplot(x = hourly_means.index, y = hourly_means['casual'], label = 'casual
')
sns.lineplot(x = hourly_means.index, y = hourly_means['registered'], label = 're
gistered')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Count')
plt.title('Average Count of Casual vs. Registered by Hour');
# END SOLUTION
```



Question 5b

What can you observe from the plot? Hypothesize about the meaning of the peaks in the registered riders' distribution.

```
BEGIN QUESTION
name: q5b
points: 2
manual: true
```

SOLUTION: In the above plot we see strong evidence of daily patterns in both datasets. The casual riders appear to ride throughout the day with peak hours in the mid-afternoon. Alternatively, while the registered riders also ride more during the day than at night there are very strong spikes during the morning and evening commute hours with a small bump during lunch.

6: Exploring Ride Sharing and Weather

Now let's examine how the weather is affecting rider's behavior. First let's look at how the proportion of casual riders changes as weather changes.

Question 6

Question 6a

Create a new column `prop_casual` in the `bike` DataFrame representing the proportion of casual riders out of all riders.

```
BEGIN QUESTION
name: q6a
points: 1
```

In [33]:

```
# BEGIN SOLUTION
bike['prop_casual'] = bike['casual'] / (bike['casual'] + bike['registered'])
# END SOLUTION
```

In [34]:

```
# TEST
int(bike["prop_casual"].sum())
```

Out[34]:

```
2991
```

In [ ]:

```
# HIDDEN TEST
np.round(bike["prop_casual"].mean(), 2)
```

Out[ ]:

```
0.17000000000000001
```

Question 6b

In order to examine the relationship between proportion of casual riders and temperature, we can create a scatterplot using `sns.scatterplot`. We can even use color/hue to encode the information about day of week. Run the cell below, and you'll see we end up with a big mess that is impossible to interpret.

In [ ]:

```
plt.figure(figsize=(10, 7))
sns.scatterplot(data=bike, x="temp", y="prop_casual", hue="weekday");
```

We could attempt linear regression using `sns.lmplot` as shown below, which hint at some relationships between temperature and proportional casual, but the plot is still fairly unconvincing.

In [ ]:

```
sns.lmplot(data=bike, x="temp", y="prop_casual", hue="weekday", scatter_kws={"s"
: 20}, height=10)
plt.title("Proportion of Casual Riders by Weekday");
```

A better approach is to use local smoothing. The basic idea is that for each x value, we compute some sort of representative y value that captures the data close to that x value. One technique for local smoothing is "Locally Weighted Scatterplot Smoothing" or LOWESS. An example is below. The red curve shown is a smoothed version of the scatterplot.

In [ ]:

```
from statsmodels.nonparametric.smoothers_lowess import lowess
# Make noisy data
xobs = np.sort(np.random.rand(100)*4.0 - 2)
yobs = np.exp(xobs) + np.random.randn(100) / 2.0
sns.scatterplot(xobs, yobs, label="Raw Data")

# Predict 'smoothed' valued for observations
ysmooth = lowess(yobs, xobs, return_sorted=False)
sns.lineplot(xobs, ysmooth, label="Smoothed Estimator", color='red')
plt.legend();
```

In our case with the bike ridership data, we want 7 curves, one for each day of the week. The x-axis will be the temperature and the y-axis will be a smoothed version of the proportion of casual riders.

You should use `statsmodels.nonparametric.smoothers_lowess.lowess` (http://www.statsmodels.org/dev/generated/statsmodels.nonparametric.smoothers_lowess.lowess.html) just like the example above. Unlike the example above, plot ONLY the lowess curve. Do not plot the actual data, which would result in overplotting. For this problem, the simplest way is to use a loop.

Hints:

- Start by just plotting only one day of the week to make sure you can do that first.
- The `lowess` function expects y coordinate first, then x coordinate.
- Look at the top of this homework notebook for a description of the temperature field to know how to convert to Fahrenheit. By default, the temperature field ranges from 0.0 to 1.0. In case you need it, $\text{Fahrenheit} = \text{Celsius} * \frac{9}{5} + 32$.

Note: If you prefer plotting temperatures in Celsius, that's fine as well!

```
BEGIN QUESTION
name: q6b
points: 4
manual: true
```

In [ ]:

```
from statsmodels.nonparametric.smoothers_lowess import lowess

plt.figure(figsize=(10,8))
# BEGIN SOLUTION
for day in bike['weekday'].unique():
    this_day = bike[bike['weekday'] == day].copy()
    this_day['temp'] = this_day['temp'] * 41 * 9 / 5 + 32
    ysmooth = lowess(this_day['prop_casual'], this_day['temp'], return_sorted=Fa
lse)
    sns.lineplot(this_day['temp'], ysmooth, label=day)

plt.title("Temperature vs Casual Rider Proportion by Weekday")
plt.xlabel("Temperature (Fahrenheit)")
plt.ylabel("Casual Rider Proportion")
plt.legend();
# plt.savefig("images/curveplot_temp_prop_casual", bbox_inches='tight', dpi=300)
;
# END SOLUTION
```

Question 6c

What do you see from the curve plot? How is `prop_casual` changing as a function of temperature? Do you notice anything else interesting?

```
BEGIN QUESTION
name: q6c
points: 2
manual: true
```

SOLUTION: As temperature increases, the proportion of casual riders increases as well, and this trend appears to continue even into very hot weather. Weekends (Saturday, Sunday) have higher proportion of casual riders (which we saw before). There are four distinct typees of days: weekends, Mondays, Fridays, and mid-week days.

A map of areas with bike sharing systems and other forms of micro mobility as of 2018 is provided, above (Source: NACTO (https://nacto.org/shared-micromobility-2018/))

Question 6d

Based on the data you have explored (distribution of orders, daily patterns, weather, additional data/information you have seen), do you think bike sharing should be realistically scaled across major cities in the the US in order to alleviate congestion, provide geographic connectivity, reduce carbon emissions, and promote inclusion among communities? Why or why not? Provide a visualisation and justify how it supports your answer

```
BEGIN QUESTION
name: q6d
points: 2
manual: true
```

SOLUTION: The following solution is not intended to be comprehensive, but instead serves as a set of possible valid answers. Answers that claim bike sharing should be scaled should argue that bike sharing is good, for reasons such as environmental benefits, and practical/logisitcal benefits afforded to commuters and students. Relevant visualisations include a plot displaying average rider count for each hour of the day, or one displaying temperature vs casual rider proportion by weekday. In comparison, answers that claim bike sharing shouldn't be scaled could argue points related to feasibility, or privacy concerns associated with oversharing of important user data. Relevant visualisations could be KDE contours of casual vs registered rider count or the environmental/geographic landscape of the United States.