In [65]:
```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('hw2.ok')
```

```
=====================================================================
Assignment: hw2
OK, version v1.13.11
=====================================================================
```

Homework 2: Bike Sharing

Exploratory Data Analysis (EDA) and Visualization

Due Date: Tuesday 2/19, 6:00 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.

Rohan Divate

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	2
Question 2d	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
Total	34

In [66]:
```
# Run this cell to set up your notebook.  Make sure ds100_utils.py is in th
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

Variable	Description
registered	count of registered users
cnt	count of total rental bikes including casual and registered

Download the Data

In [67]:
```
# Run this cell to download the data.  No further action is needed

data_url = 'https://github.com/DS-100/sp19/raw/gh-pages/assets/datasets/hw2
file_name = 'data.zip'
data_dir = '.'

dest_path = ds100_utils.fetch_and_cache(data_url=data_url, data_dir=data_di
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: Sun Feb 17 16:47:34 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 [68]:
```
# Run this cell to look at the top of the file.  No further action is neede
for line in ds100_utils.head(data_dir/'bikeshare.txt'):
    print(line,end="")
```

```
instant,dteday,season,yr,mnth,hr,holiday,weekday,workingday,weathersit,te
mp,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 [69]:
```
# 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"), "lin
```

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

Loading the data

The following code loads the data into a Pandas DataFrame.

In [70]:
```
# Run this cell to load the data.  No further action is needed
bike = pd.read_csv(data_dir/'bikeshare.txt')
bike.head()
```

Out[70]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879

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 [71]:
```
bike.shape
```

Out[71]:
```
(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.

In [72]:
```
# Modify holiday weekday, workingday, and weathersit here
bike['holiday'].replace({0: "no", 1: "yes"}, inplace=True)
bike['workingday'].replace({0: "no", 1: "yes"}, inplace=True)
bike['weathersit'].replace({1: "Clear", 2: "Mist", 3: "Light", 4: "Heavy"},
bike['weekday'].replace({0: "Sun", 1: "Mon", 2: "Tue", 3: "Wed", 4: "Thu",
bike.head()
```

Out[72]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.2879
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.2727
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.2727
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.2879
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.2879

In [73]:
```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 6
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 1b (Holidays)

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

In [74]:
```
num_holidays = bike[bike['holiday']=="yes"].count()['holiday']
num_holidays
```

Out[74]:
```
500
```

In [75]:
```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

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.

In [77]:
```
daily_counts = bike.groupby('dteday').agg({'casual' : "sum", "registered" :
daily_counts.head(10)
```

Out[77]:

dteday	casual	registered	workingday
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
2011-01-06	88	1518	yes
2011-01-07	148	1362	yes
2011-01-08	68	891	no
2011-01-09	54	768	no
2011-01-10	41	1280	yes

In [78]:
```
ok.grade("q1c");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

2: Exploring the Distribution of Riders

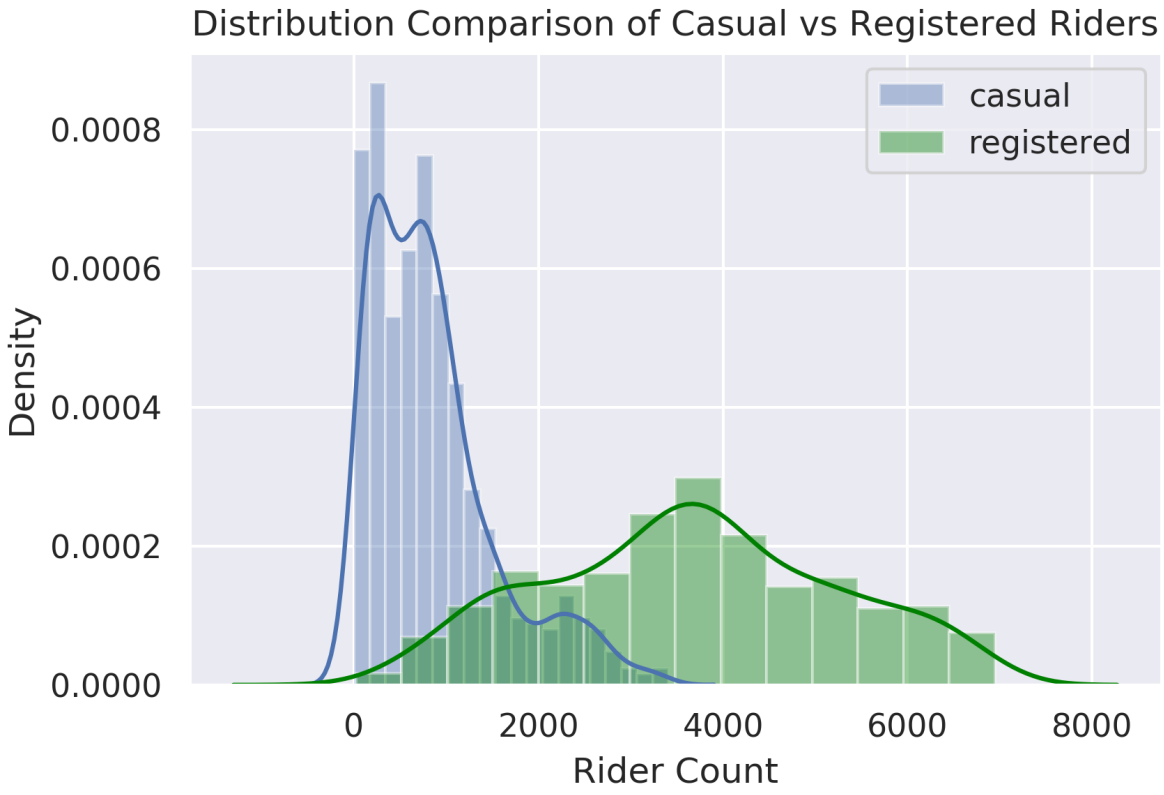
Let's begin by comparing the distribution of the daily counts of casual and registered riders.

Question 2

Question 2a

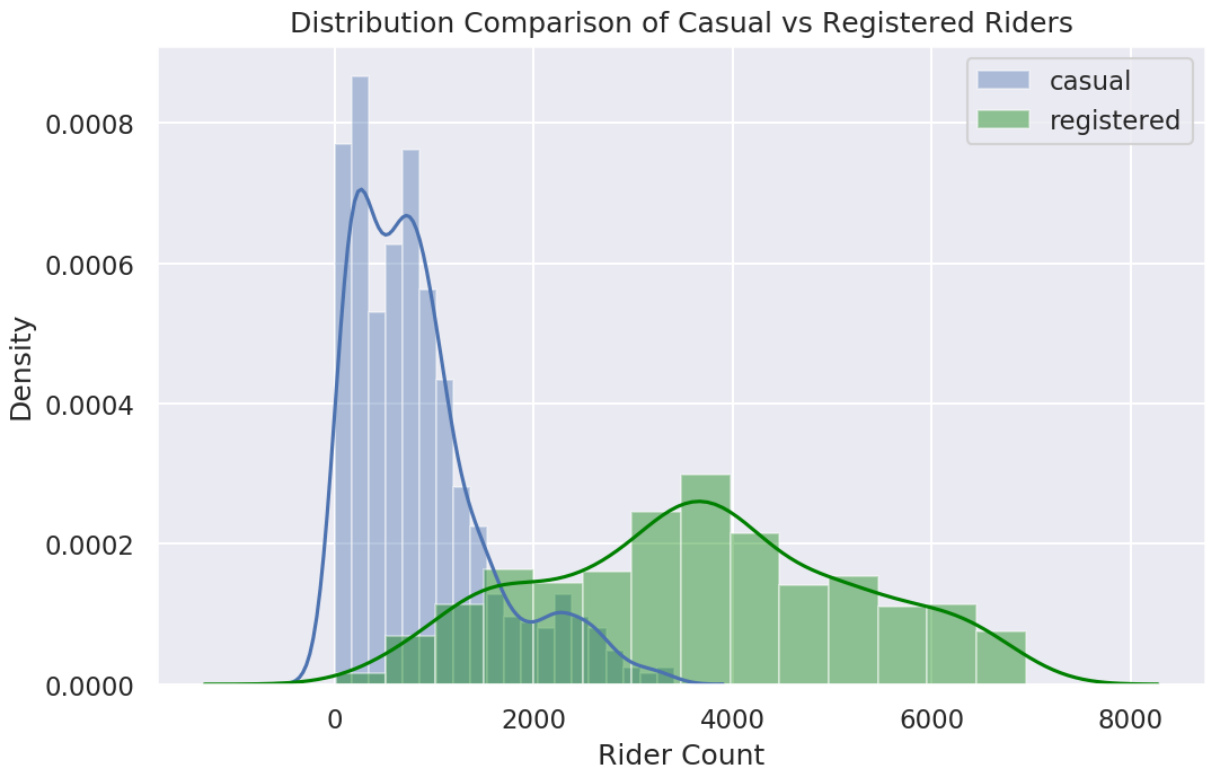
Use the `sns.distplot` (https://seaborn.pydata.org/generated/seaborn.distplot.html) function to create a plot that overlays the distribution of the daily counts of `casual` and `registered` users. The temporal granularity of the records should be daily counts, which you should have after completing question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



In [79]:
```
plt.figure(figsize=(8,5))
plt.title('Distribution Comparison of Casual vs Registered Riders')
plt.ylabel('Density')
sns.distplot(daily_counts['casual'], label='casual', kde=True)
sns.distplot(daily_counts['registered'], label='registered', color='green',
plt.xlabel('Rider Count')
plt.legend()
```

Out[79]: <matplotlib.legend.Legend at 0x7f889d24d9b0>



Question 2b

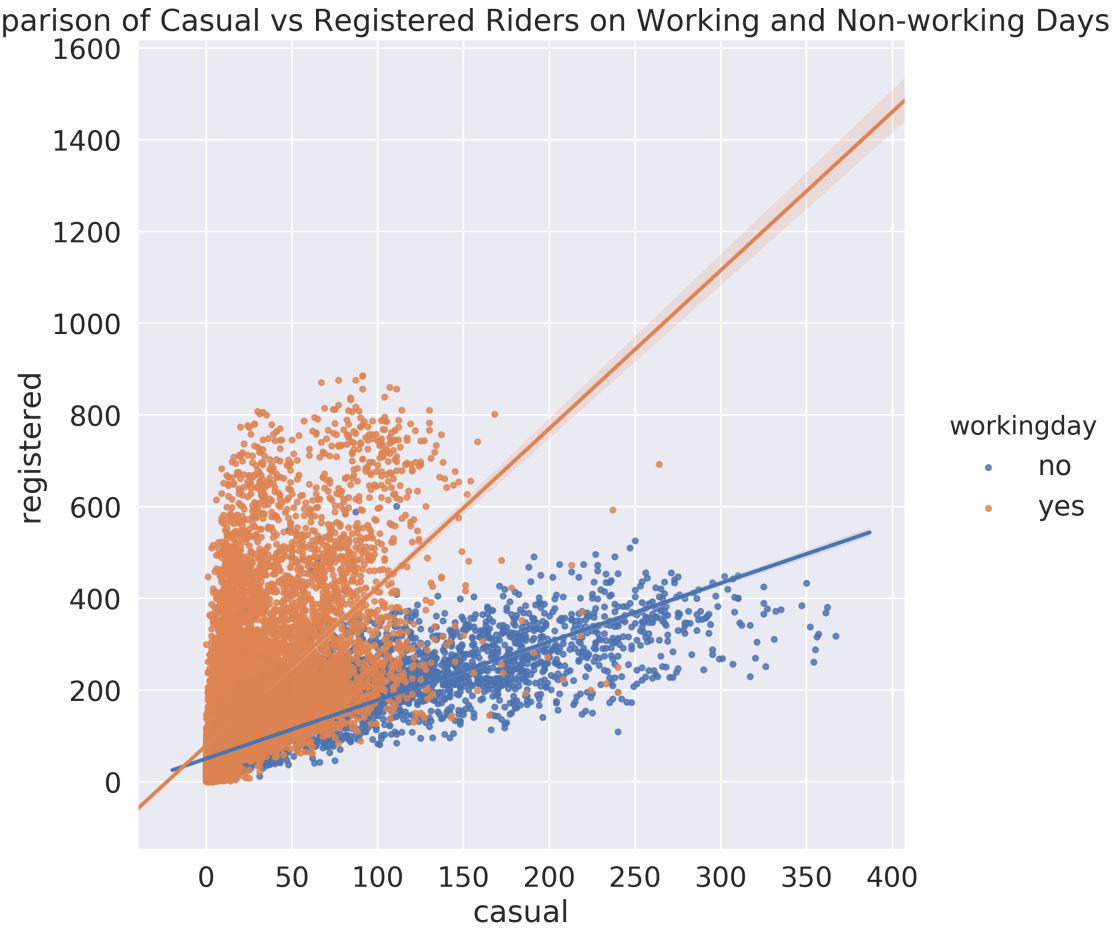
In the cell below, descibe the differences you notice between the density curves for casual and registered riders. Consider concepts such as modes, symmetry, skewness, tails, gaps and outliers. Include a comment on the spread of the distributions.

The plot for casual riders appears to be bimodal and skewed right. Since this plot is a lot taller and narrower than the plot for registered riders, it shows that there tends to be less variation in the low number of casual riders, and that these low rider values tend to occur with much higher frequency. The plot for registered riders appears to be a lot more symmetric, and roughly resembles a normal distribution with the mean/median at around 4000 riders. Since this plot is a lot shorter and wider than the plot for casual riders, it shows that there tends to be more variation in the high number of casual riders, and that these high rider values tend to occur with much lower frequency.
Based on the two plots, we can see that, in general, there appear to be a lot more registered riders than casual riders, which makes sense because if people are registered, it is probably because they use it more often. There do appear to be a few outliers toward the tail end of the casual riders plot, as evidenced by their low density. There don't appear to be any major gaps in either of the graphs.

Question 2c

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`.

In [83]:
```
# Make the font size a bit bigger
sns.set(font_scale=1.5)
sns.lmplot(x='casual', y='registered', height=8, hue="workingday", fit_reg=
plt.suptitle("Comparison of Casual vs Registered Riders on Working and Non-
```

Out[83]:
```
Text(0.5, 0.98, 'Comparison of Casual vs Registered Riders on Working and
Non-working Days')
```

Question 2d

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?

It seems as though registered riders tend to ride more on working days (i.e. weekdays), whereas on non-working days (i.e. weekends), the number of casual and registered riders are fairly similar since the linear regression line looks like the identity line.

Overplotting makes it difficult to determine whether or not there are blue dots underneath the orange dots toward the bottom left hand side of the graph.

3: Visualization

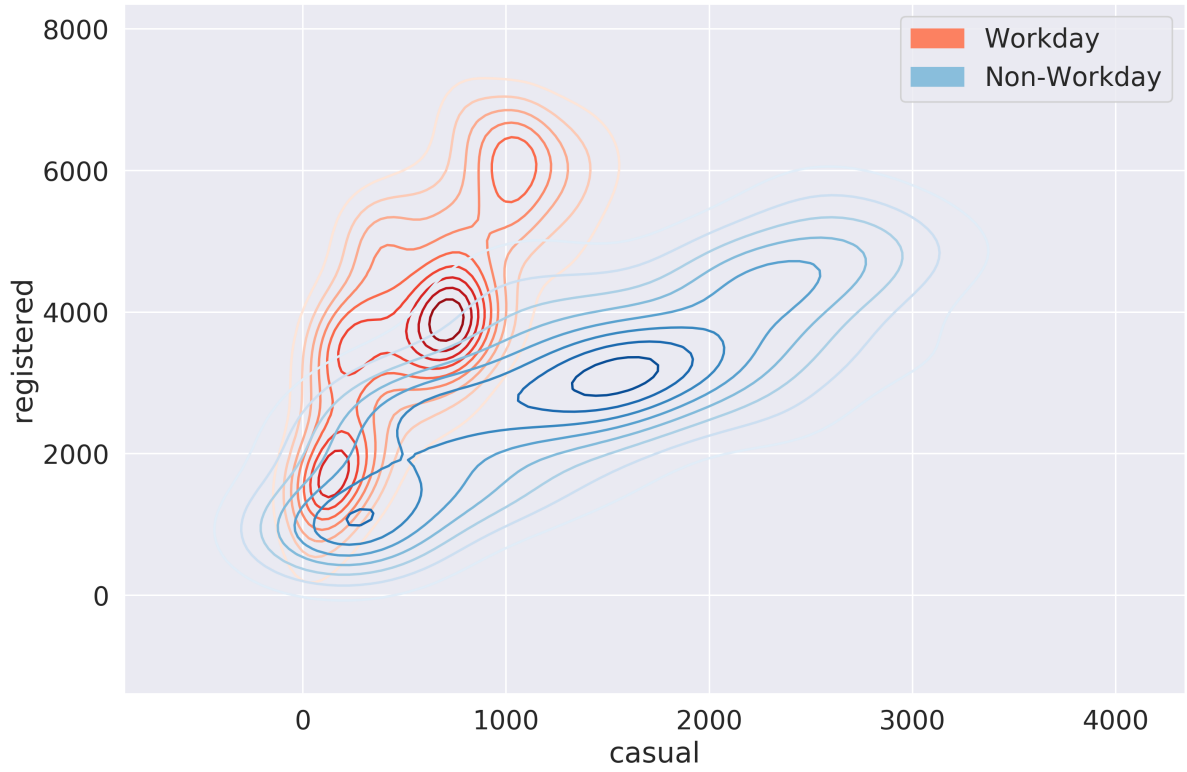
Question 3

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)

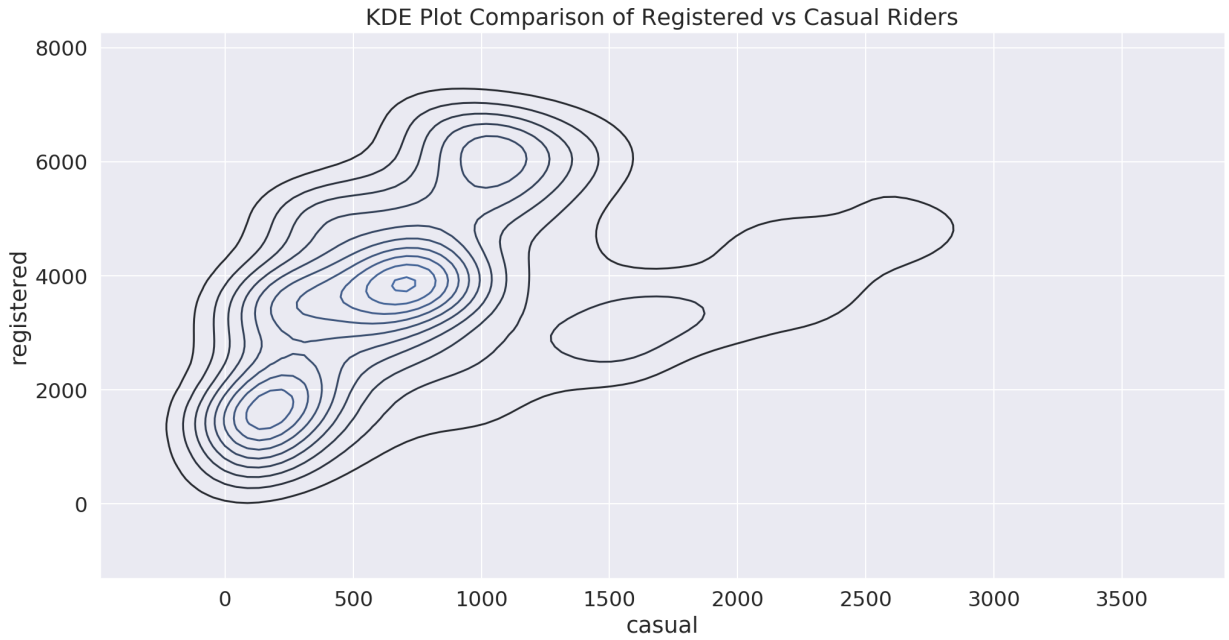
The result we wish to achieve should be a plot that looks like this:



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 [84]:
```
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

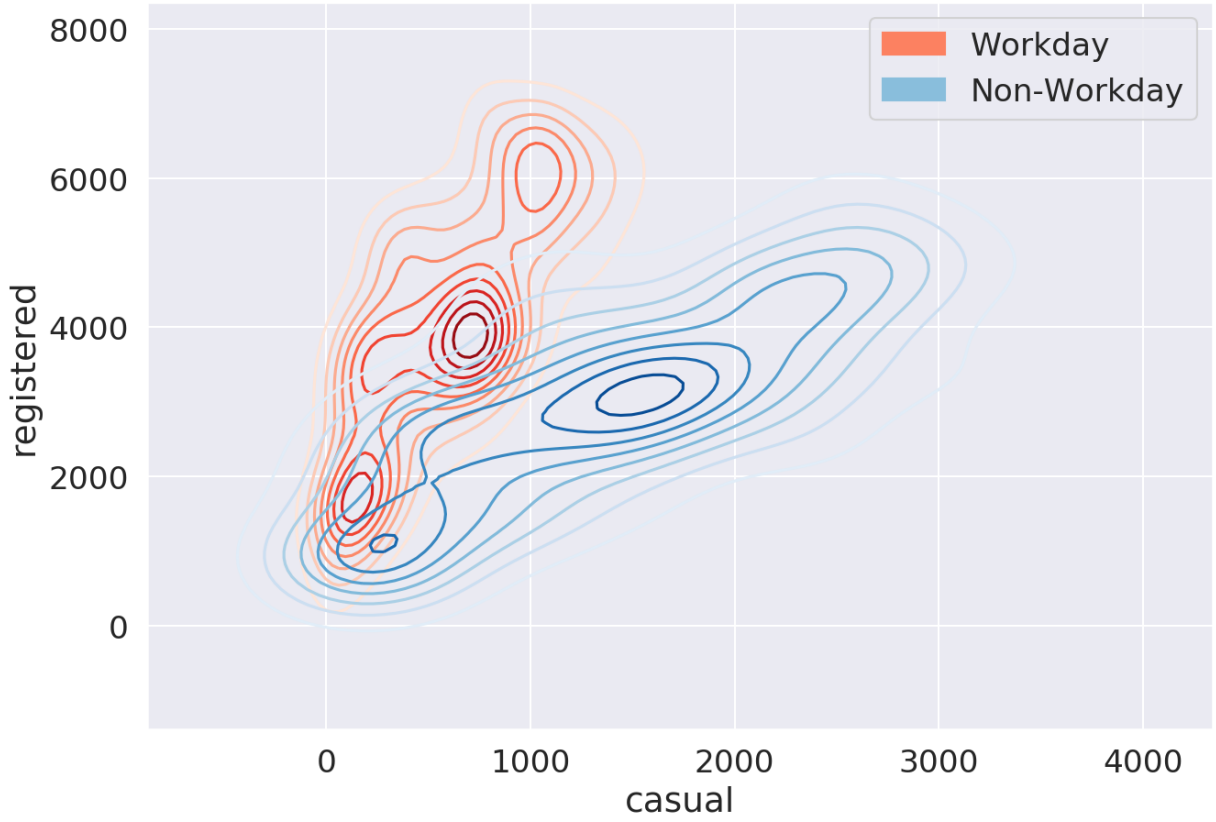
walkthrough below, feel free to use whatever method you wish however if you do not want to follow the walkthrough.

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`.

In [85]:
```
import matplotlib.patches as mpatches  # see the tutorial for how we use mp

# Set 'is_workingday' to a boolean array that is true for all working_days
plt.figure(figsize=(10, 7))
is_workingday = (daily_counts['workingday'] == 'yes').values

# Bivariate KDEs require two data inputs.
# In this case, we will need the daily counts for casual and registered rid
# Hint: use loc and is_workingday to splice out the relevant rows and colum
casual_weekday = daily_counts.loc[is_workingday, 'casual']
registered_weekday = daily_counts.loc[is_workingday, 'registered']

# Use sns.kdeplot on the two variables above to plot the bivariate KDE for
sns.kdeplot(casual_weekday, registered_weekday, cmap="Reds")

# Repeat the same steps above but for rows corresponding to non-workingdays
is_not_workingday = (daily_counts['workingday'] == 'no').values
casual_weekend = daily_counts.loc[is_not_workingday, 'casual']
registered_weekend = daily_counts.loc[is_not_workingday, 'registered']

# Use sns.kdeplot on the two variables above to plot the bivariate KDE for
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.show()
```



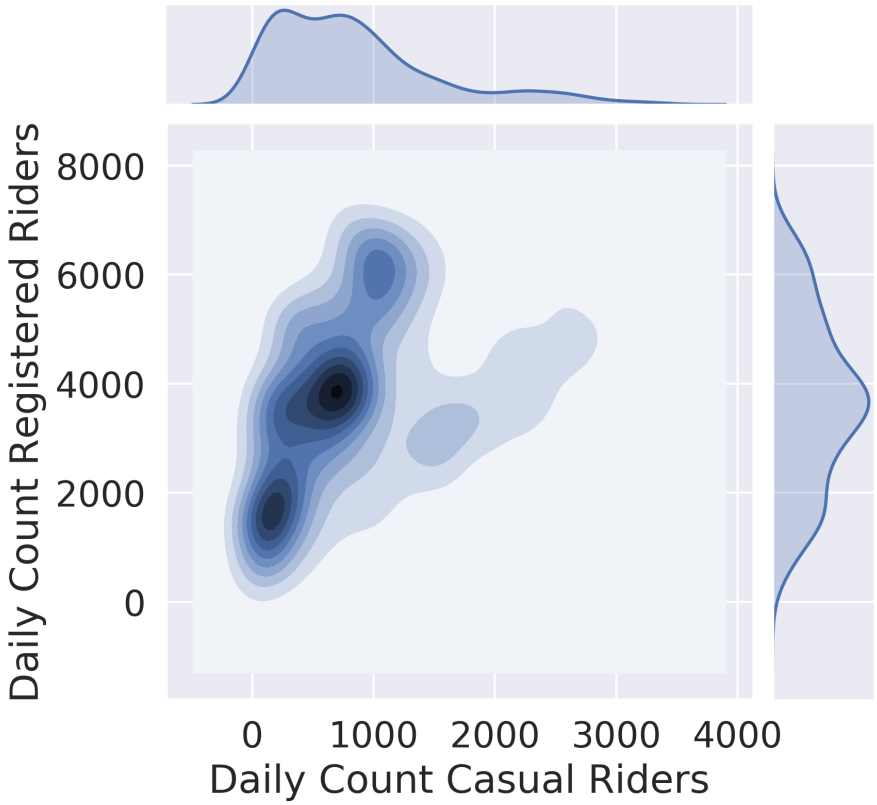
Question 3b

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

The contour plot makes it a lot easier to see where the areas of higher density (i.e. the inner circles) are for both registered and casual riders. This was a lot harder to determine from the scatter plot because of overfitting.

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).

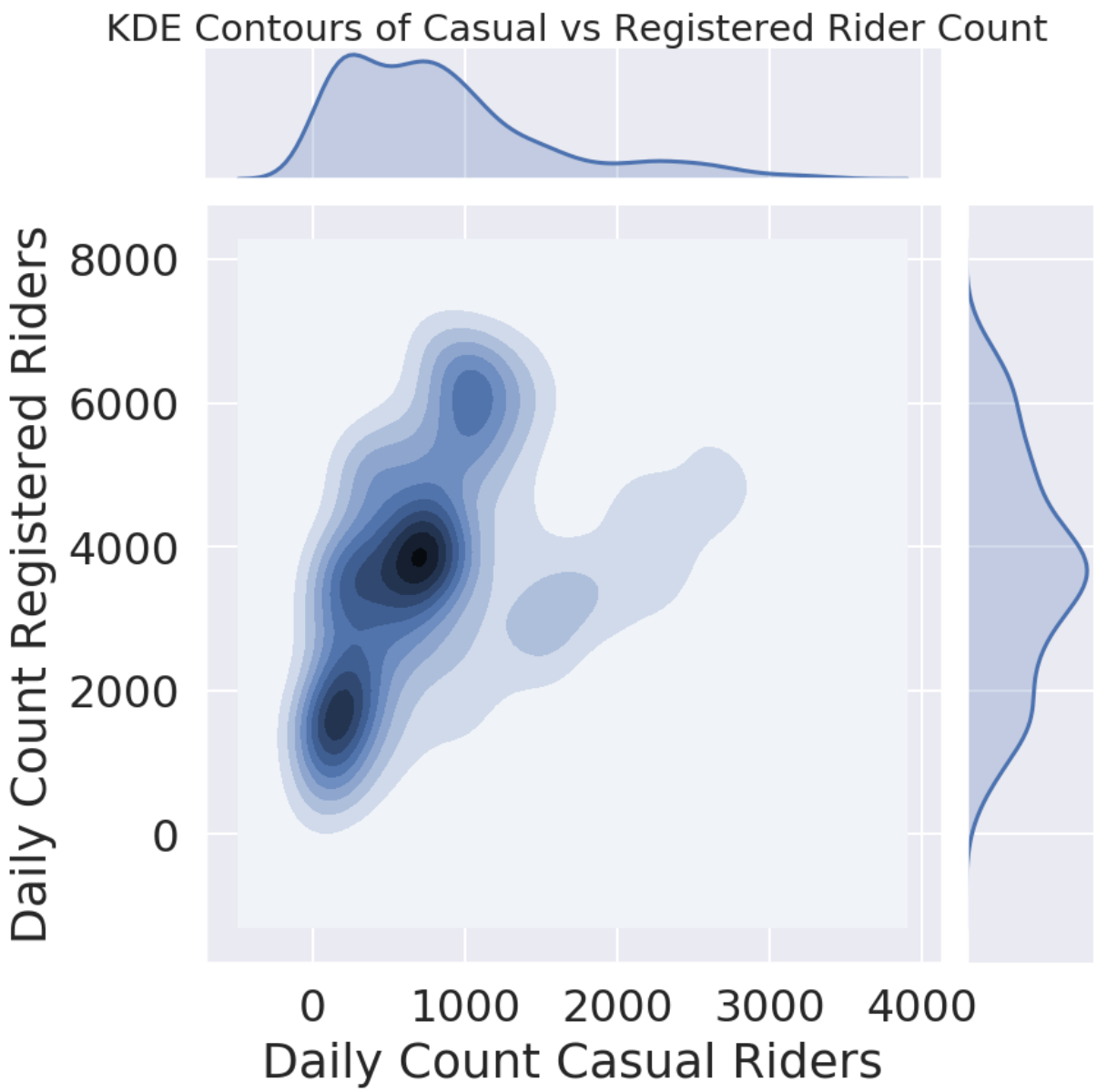


Hints:

- 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

In [86]:
```
g = (sns.jointplot(x="casual", y="registered", data=daily_counts, kind="kde
plt.suptitle('KDE Contours of Casual vs Registered Rider Count', fontsize=1
```

Out[86]:
```
Text(0.5, 0.98, 'KDE Contours of Casual vs Registered Rider Count')
```



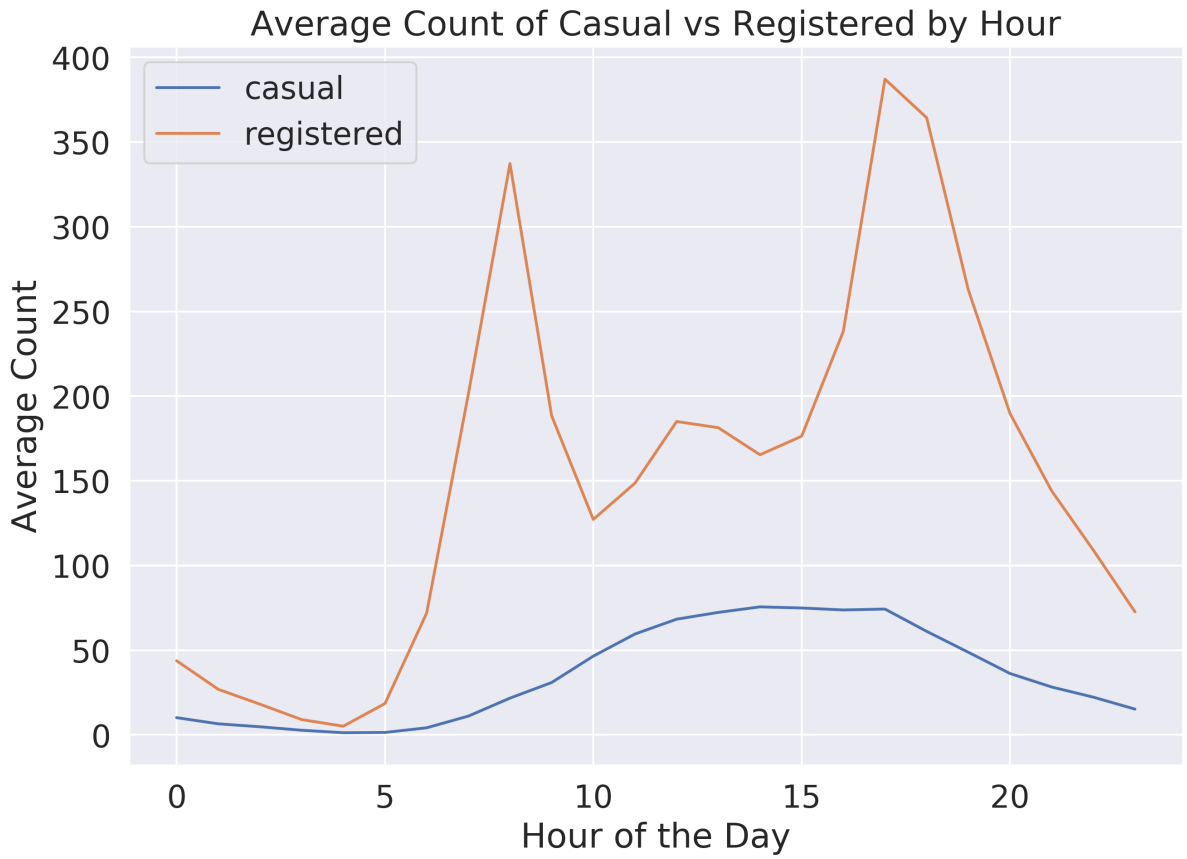
5: Understanding Daily Patterns

Question 5

Question 5a

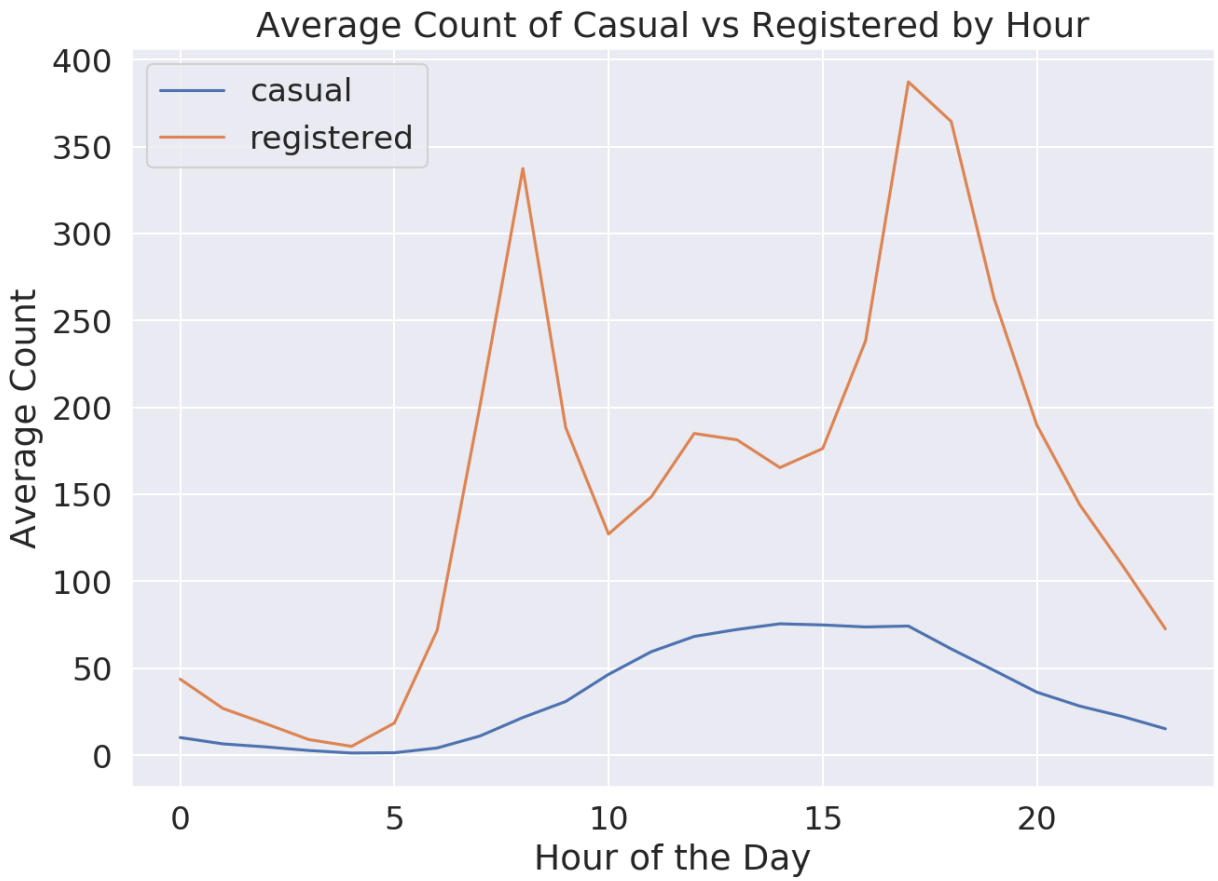
Let's examine the behavior of riders by plotting the average number of riders for each hour of the day over the **entire dataset**, stratified by rider type.

Your plot should look like the following:



In [87]:
```
plt.figure(figsize=(10, 7))
df = bike.copy()
df['avgCasual'] = bike.groupby(['hr']).mean()['casual']
df['avgRegistered'] = bike.groupby(['hr']).mean()['registered']
df_filter = df[['hr', 'avgCasual', 'avgRegistered']]
#df_filter.dropna()
casual_plot = sns.lineplot(x='hr', y='avgCasual', data=df_filter, label="ca
registered_plot = sns.lineplot(x='hr', y='avgRegistered', data=df_filter, l
casual_plot.set(xlabel='Hour of the Day', ylabel='Average Count')
casual_plot.set_title("Average Count of Casual vs Registered by Hour")
```

Out[87]:
```
Text(0.5, 1.0, 'Average Count of Casual vs Registered by Hour')
```



Question 5b

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

From the plot, we can see that casual riders tend to ride more frequently during the middle of the day between 10am and 4pm. Registered riders, however, tend to ride more frequently during peak commuting hours, and not during the middle of the day, since this is usually the time when they have to work. For the most part, there don't appear to be as many riders during the tail ends of the day (i.e. really early in the morning or really late at night) for either casual riders or registered riders. The peaks in the registered riders' distribution is probably indicative of the rush hour working day times. The first peak occurs at around 8-9am, which is when employees commute to work, hence the increase in average count of registered riders. The second peak occurs at around 6-7pm, which is when employees commute from work to home.

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.

In [88]:
```
bike['prop_casual'] = bike['casual'] / bike['cnt']
bike.head()
```

Out[88]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	2011-01-01	1	0	1	0	no	Sat	no	Clear	0.24	0.2879
1	2	2011-01-01	1	0	1	1	no	Sat	no	Clear	0.22	0.2727
2	3	2011-01-01	1	0	1	2	no	Sat	no	Clear	0.22	0.2727
3	4	2011-01-01	1	0	1	3	no	Sat	no	Clear	0.24	0.2879
4	5	2011-01-01	1	0	1	4	no	Sat	no	Clear	0.24	0.2879

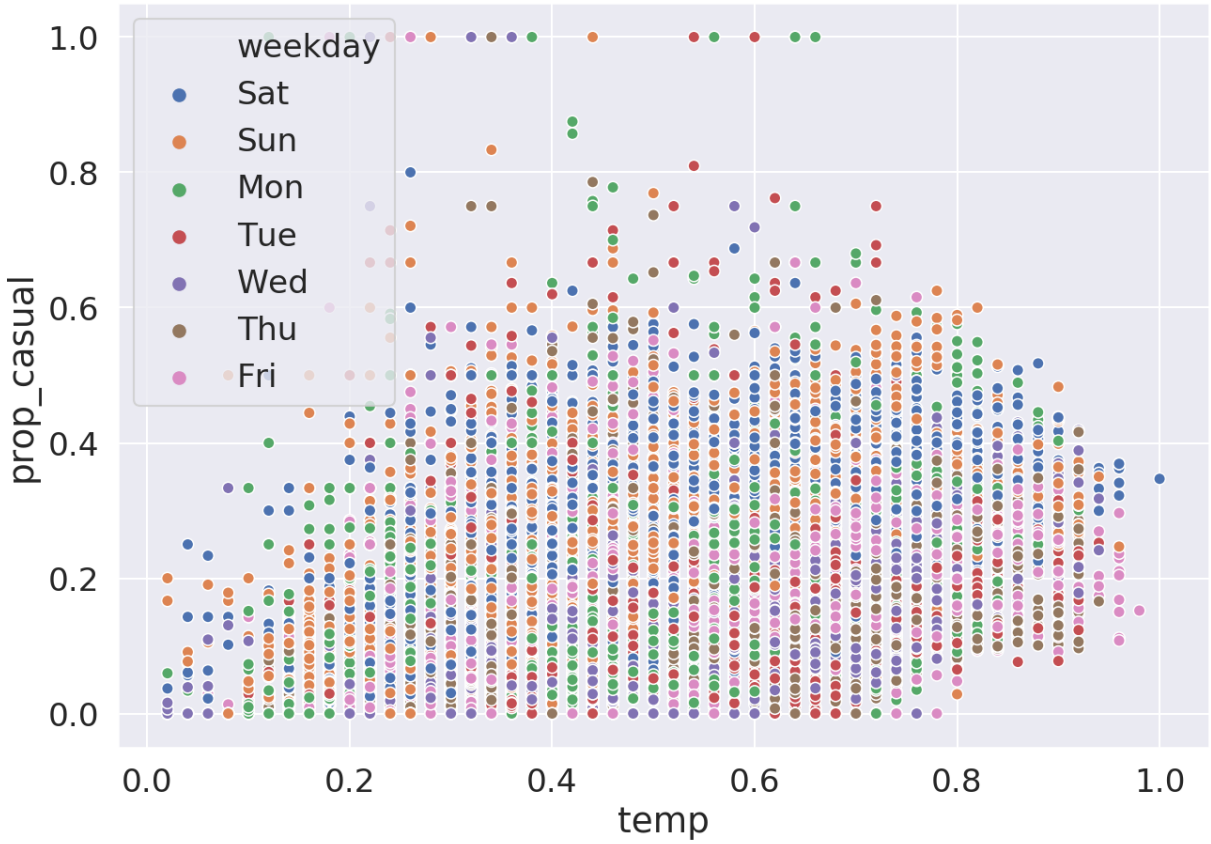
In [89]:
```
ok.grade("q6a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 6b

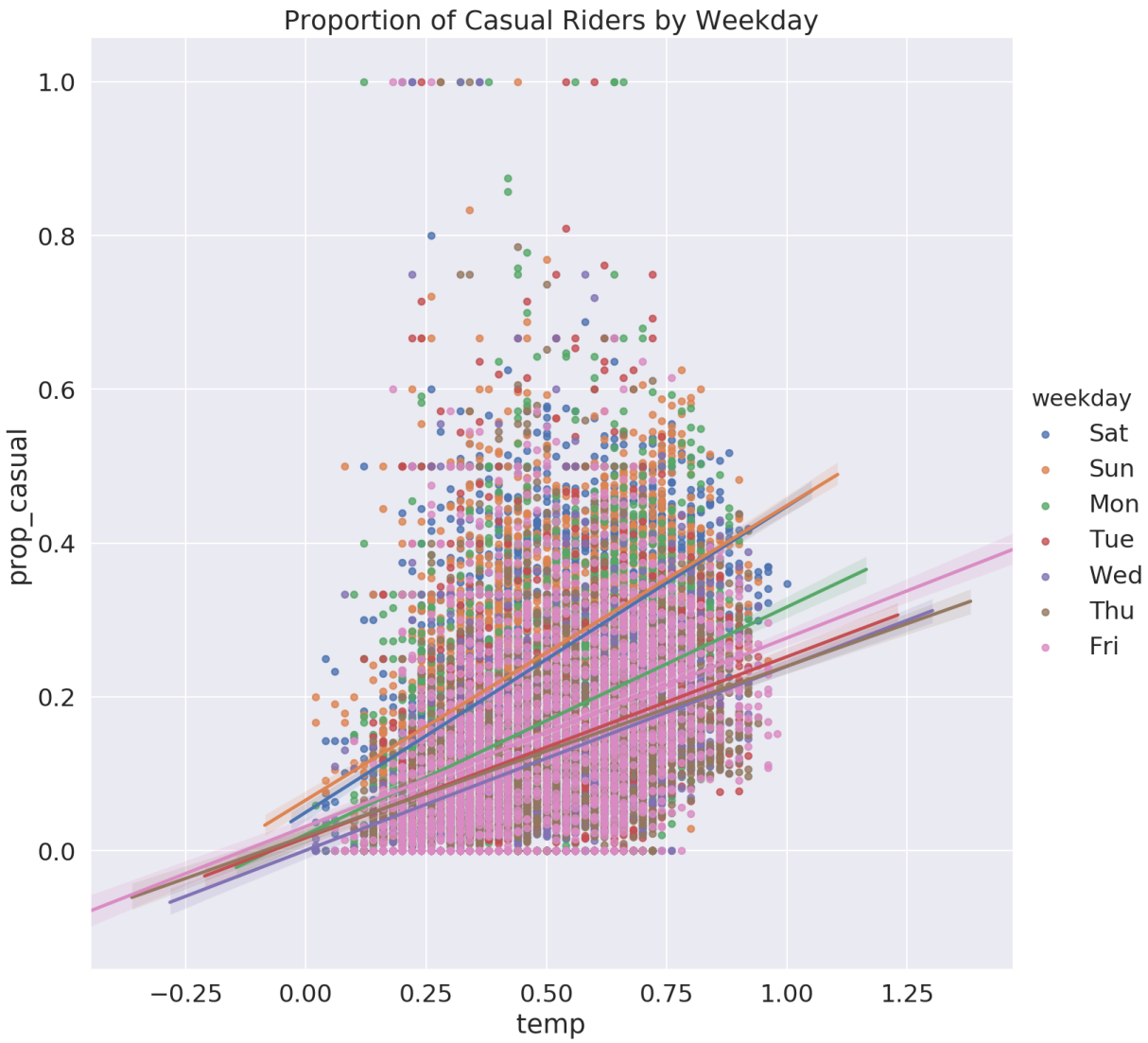
In order to examine the relationship between proportion of casual riders and temperature, we can create a scatterplot using `sns.scatterplot`. We can even use color/hue to encode the information about day of week. Run the cell below, and you'll see we end up with a big mess that is impossible to interpret.

In [90]:
```
plt.figure(figsize=(10, 7))
sns.scatterplot(data=bike, x="temp", y="prop_casual", hue="weekday");
```



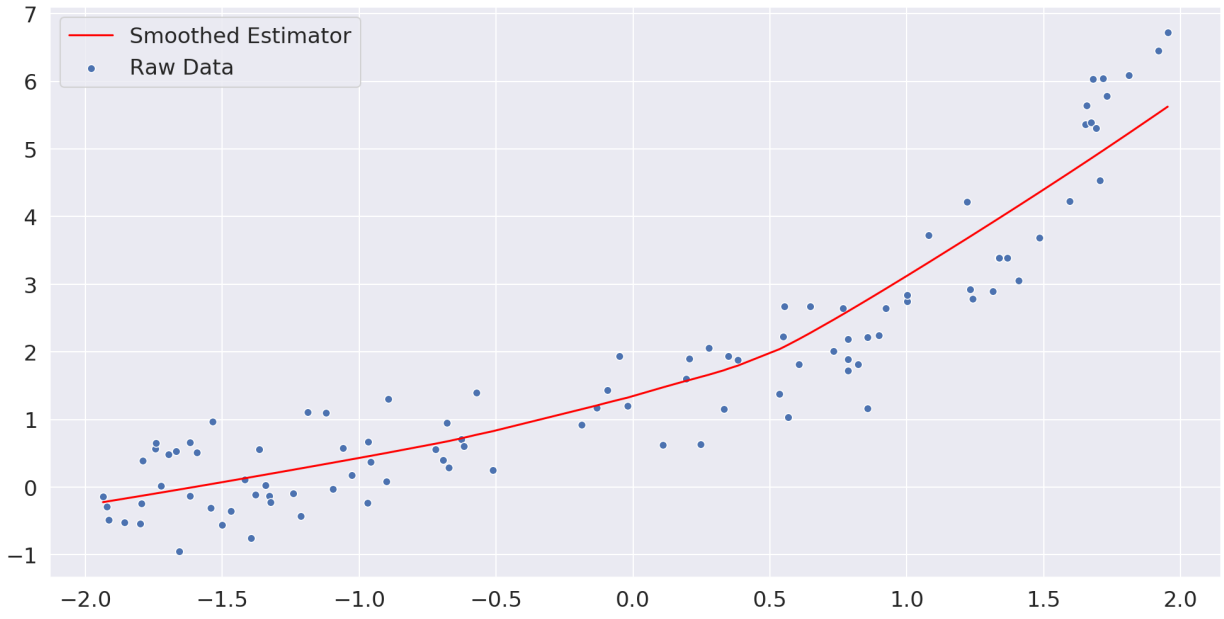
We could attempt linear regression using `sns.lmplot` as shown below, which hint at some relationships between temperature and proportional casual, but the plot is still fairly unconvincing.

In [91]:
```
sns.lmplot(data=bike, x="temp", y="prop_casual", hue="weekday", scatter_kws
plt.title("Proportion of Casual Riders by Weekday");
```

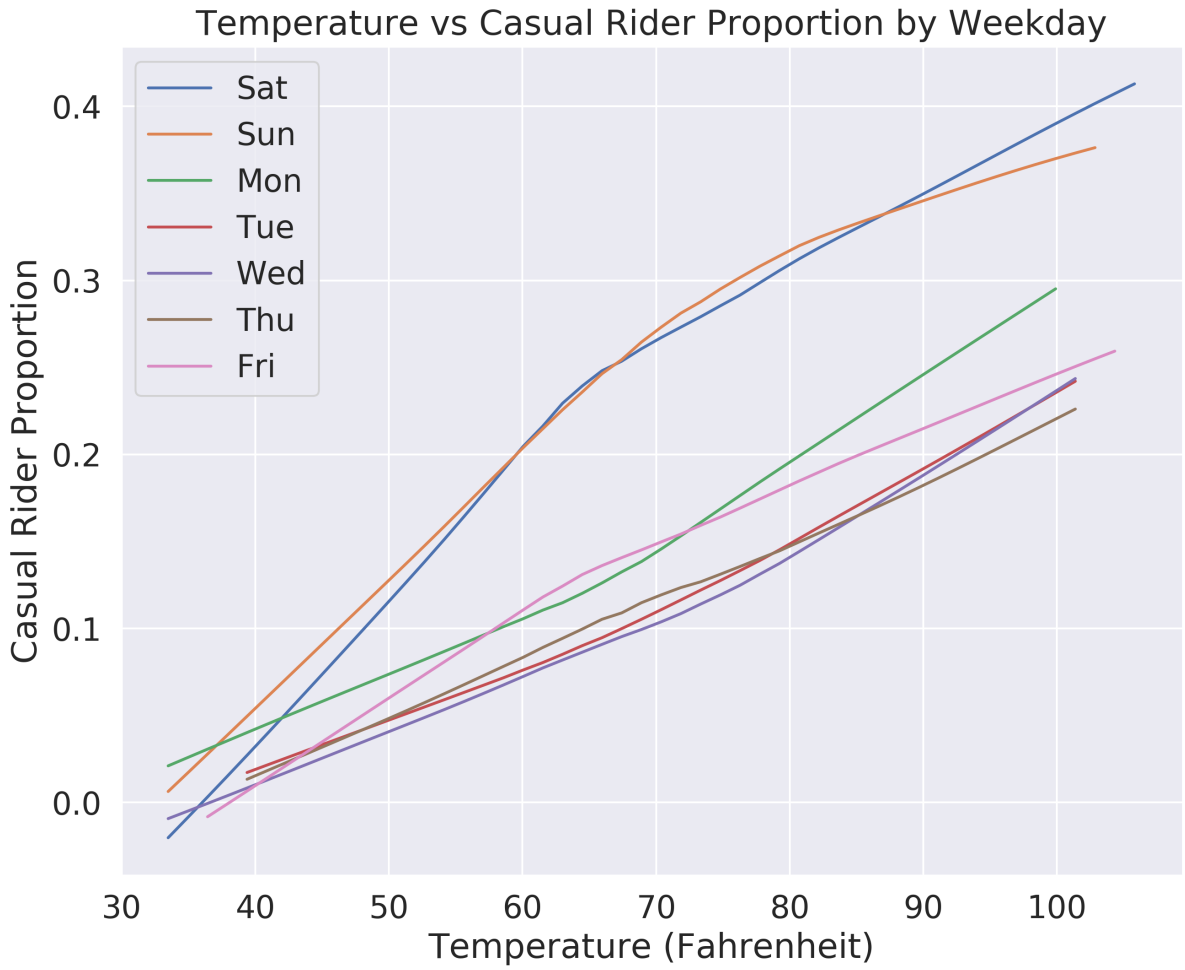


A better approach is to use local smoothing. The basic idea is that for each x value, we compute some sort of representative y value that captures the data close to that x value. One technique for local smoothing is "Locally Weighted Scatterplot Smoothing" or LOWESS. An example is below. The red curve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

In [92]:
```
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.ht 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.

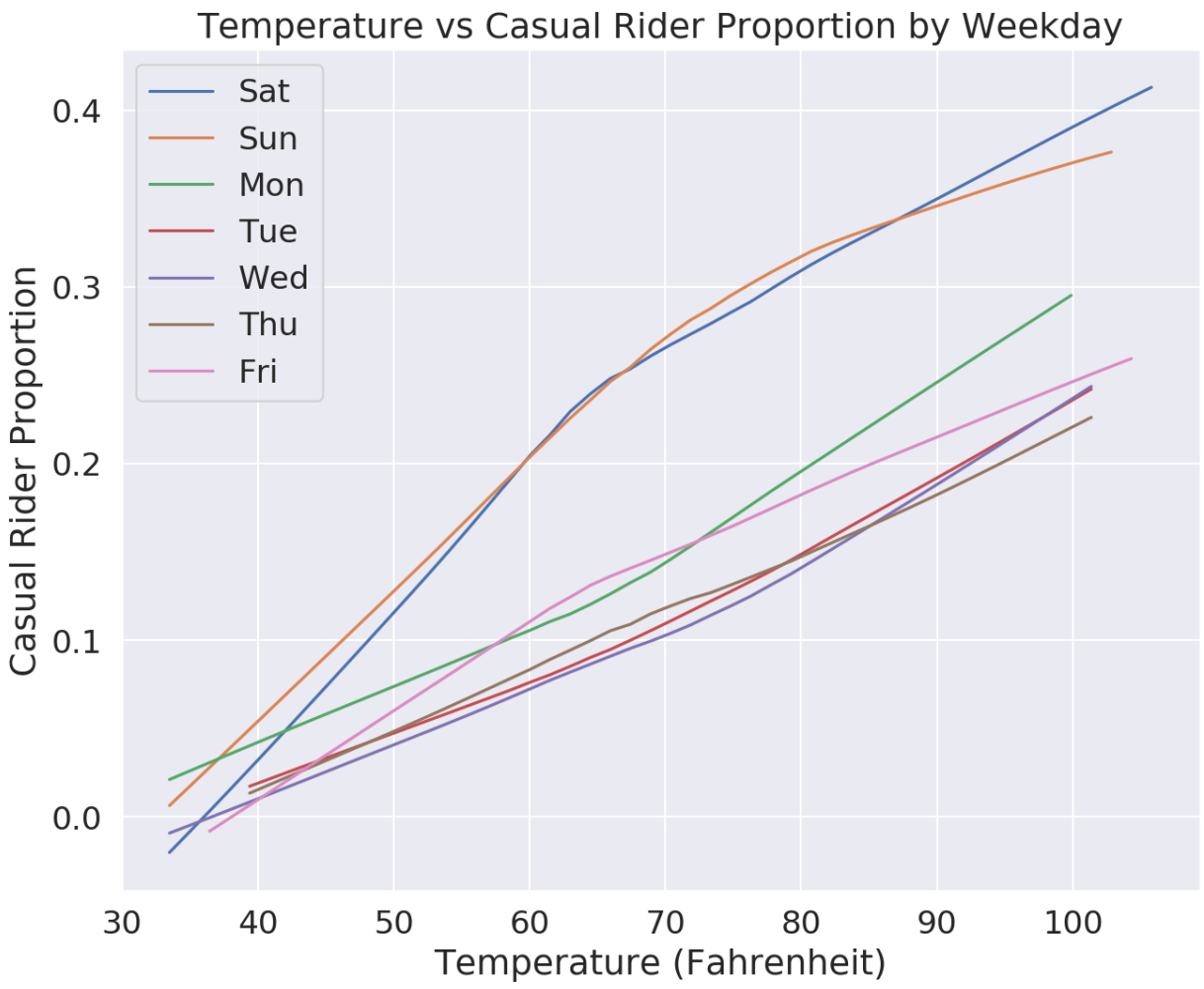
Hints:

- 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!

In [94]:
```
from statsmodels.nonparametric.smoothers_lowess import lowess

plt.figure(figsize=(10,8))
day_plot = None
for day in bike['weekday'].unique():
    tempdf = bike[bike['weekday'].str.contains(day)]
    yobs = tempdf['prop_casual'].values
    xobs = ((tempdf['temp'].values) * 41) * (9/5) + 32
    ysmooth = lowess(yobs, xobs, return_sorted=False)
    day_plot = sns.lineplot(xobs, ysmooth, label=day)

day_plot.set(xlabel='Temperature (Fahrenheit)', ylabel='Casual Rider Propor
day_plot.set_title("Temperature vs Casual Rider Proportion by Weekday")
```

Out[94]:
```
Text(0.5, 1.0, 'Temperature vs Casual Rider Proportion by Weekday')
```



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?

For the most part, the casual rider proportion (i.e. prop_casual) tends to increase on the weekends (i.e. Saturday and Sunday) and also tends to increase with increasing temperature. On the weekdays, the temperature has to be a lot warmer, compared to the weekend, in order for the casual rider proportion to increase. For instance, it only takes 60ºF on the weekends for the casual rider proportion to hit 0.2, whereas on Monday, it needs to hit 80ºF in order to reach the same casual rider proportion of 0.2.
It's interesting that the plots don't decrease when the temperature crosses 90º Fahrenheit. I would think that once the temperature crosses 90º F, people would want to stay indoors and not ride bikes outside. It's also interesting to see that, for the most part, the plots on the weekdays appear to be relatively similar.

In [95]:
```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('hw2.ipynb', 'hw2.pdf')
ok.submit()
```

```
Generating PDF...
Saved hw2.pdf

Saving notebook...

ERROR  | auth.py:91 | {'error': 'invalid_grant'}

Could not save your notebook. Make sure your notebook is saved before sen
ding it to OK!
Performing authentication
Please enter your bCourses email.
bCourses email: kushal.singh@berkeley.edu

Copy the following URL and open it in a web browser. To copy,
highlight the URL, right-click, and select "Copy".

https://okpy.org/client/login/ (https://okpy.org/client/login/)

After logging in, copy the code from the web page, paste it below,
and press Enter. To paste, right-click and select "Paste".

Paste your code here: xRuGfrJ8CtvkRMewNPP8t3uraSYOUs
Successfully logged in as kushal.singh@berkeley.edu
Submit... 100% complete
Submission successful for user: kushal.singh@berkeley.edu
URL: https://okpy.org/cal/data100/sp19/hw2/submissions/YWvqop (https://ok
py.org/cal/data100/sp19/hw2/submissions/YWvqop)
```

In [ ]: