Lab 5: Bike Sharing

Exploratory Data Analysis (EDA) and Visualization

Introduction

Bike sharing systems are a 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

In [ ]:
```
# Run this cell to set up your notebook.
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

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

import warnings
warnings.filterwarnings("ignore")

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

Loading the data

The following code loads the data into a Pandas DataFrame.

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

Out[ ]:

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

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

Out[ ]:
```
(17379, 17)
```

0: Examining the Data

Before we start working with the data, let's examine its granularity.

Question 0

Question 0A

What is the granularity of the data (i.e. what does each row represent)?

Each row represents hourly data for a specific day.

Question 0B

For this assignment, we'll be using this data to study bike usage in Washington D.C. Based on the granularity and the variables present in the data, what might some limitations of using this data be? What are two additional data categories/variables that you can collect to address some of these limitations?

Limitations include the difficulty of analyzing broad trends from highly granular data and the lack of geographic context. We can collect additional data on pickup/drop-off locations and user demographics

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, in order of the previous sentence.
3. `workingday`: Convert to `yes` and `no`.
4. `weathersit`: You should replace each value with one of `Clear`, `Mist`, `Light`, or `Heavy`. Assume `1` corresponds to `Clear`, `2` corresponds to `Mist`, and so on in order of the previous sentence.

Note: If you mutate any of the tables above, then they will not be in the format of their original `.csv` file. As a debugging tip, if you want to revert changes, run the cell that reloads the csv.

Hint: One approach is to use the replace method of the pandas DataFrame class. Take a look at the link by clicking on the word `replace` in the previous sentence. We have already included `replace` in the cell below so you can focus on creating the "nested-dictonaries" described in the documentation.

In [ ]:
```
# Modify holiday weekday, workingday, and weathersit here
# gpt is used to learn hot to decode
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()
```

Out[ ]:

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

Question 1b (Holidays)

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

In [ ]:
```
num_holidays = (bike['holiday'] == 'yes').sum()
num_holidays
```

Out[ ]:
```
500
```

Question 1c (Computing Daily Total Counts)

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 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 [ ]:
```
# gpt is used to learn to use .agg()

daily_counts = bike.groupby('dteday').agg({
    'casual': 'sum',
```

```
    'registered': 'sum',
    'workingday': 'first'
})
daily_counts.head()
```

Out[ ]:

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

2: Exploring the Distribution of Riders

Let's begin by comparing the distribution of the daily counts of casual and registered riders. Questions 2-7 require using many visualization methods so for your convenience, we have summarized a few useful ones below.

Matplotlib and Seaborn Table of Common Functions

`x` and `y` are sequences of values (i.e. arrays, lists, or Series).

Function	Description
`plt.plot(x, y)`	Creates a line plot of `x` against `y`
`plt.title(name)`	Adds a title `name` to the current plot
`plt.xlabel(name)`	Adds a label `name` to the x-axis
`plt.ylabel(name)`	Adds a label `name` to the y-axis
`plt.scatter(x, y)`	Creates a scatter plot of `x` against `y`
`plt.hist(x, bins=None)`	Creates a histogram of `x`; `bins` can be an integer or a sequence
`plt.bar(x, height)`	Creates a bar plot of categories `x` and corresponding heights `height`
`sns.histplot(data, x, y, hue, kde)`	Creates a distribution plot; `data` is a DataFrame; `x`, `y` are column names in `data` that specify positions on the x and y axes; `hue` is a column name in `data` that adds subcategories to the plot based on `hue`; `kde` is a boolean that determines whether to overlay a KDE curve
`sns.lineplot(data, x, y, hue)`	Creates a line plot

Function	Description
`sns.scatterplot(data, x, y, hue, size)`	Creates a scatter plot; `size` is a vector that contains the size of point for each subcategory based on `hue`
`sns.kdeplot(x, y)`	Creates a kernel density estimate plot; `x`, `y` are series of data that indicate positions on the `x` and `y` axis
`sns.jointplot(x, y, data, kind)`	Creates a joint plot of 2 variables with KDE plot in the middle and a distribution plot for each variable on the sides; `kind` determines the visualization type for the distribution plot, can be `scatter`, `kde` or `hist`

Note: This list of functions and parameters is **not** exhaustive. You may need to reference and explore more documentation to answer the following questions, but we will help you through that process.

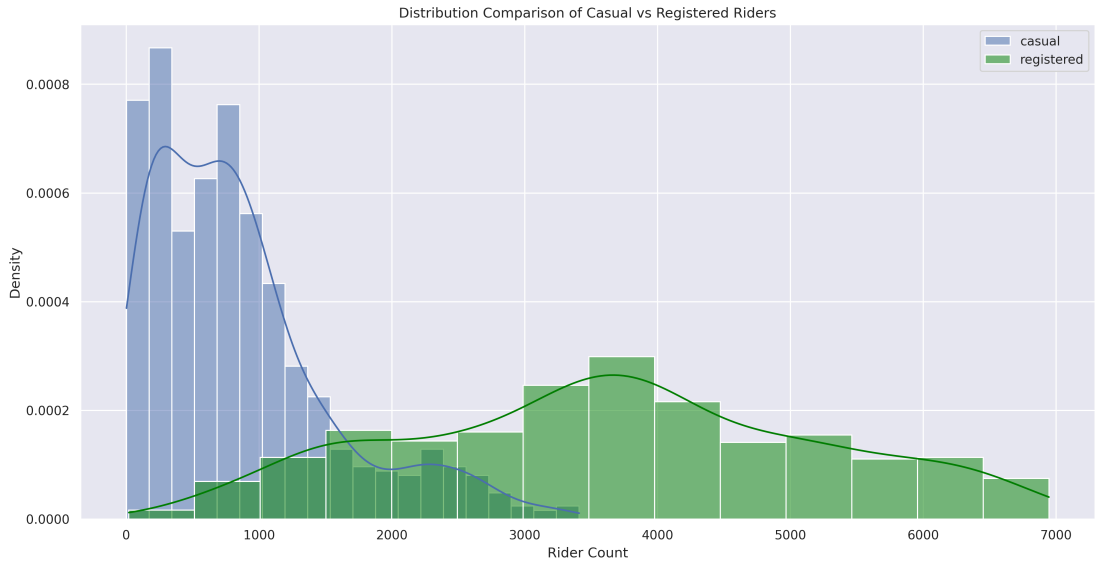
Question 2

Question 2a

Use the `sns.histplot` function to create a plot that overlays the distribution of the daily counts of bike users, using blue to represent `casual` riders, and green to represent `registered` riders. The temporal granularity of the records should be daily counts, which you should have after completing question 1c.

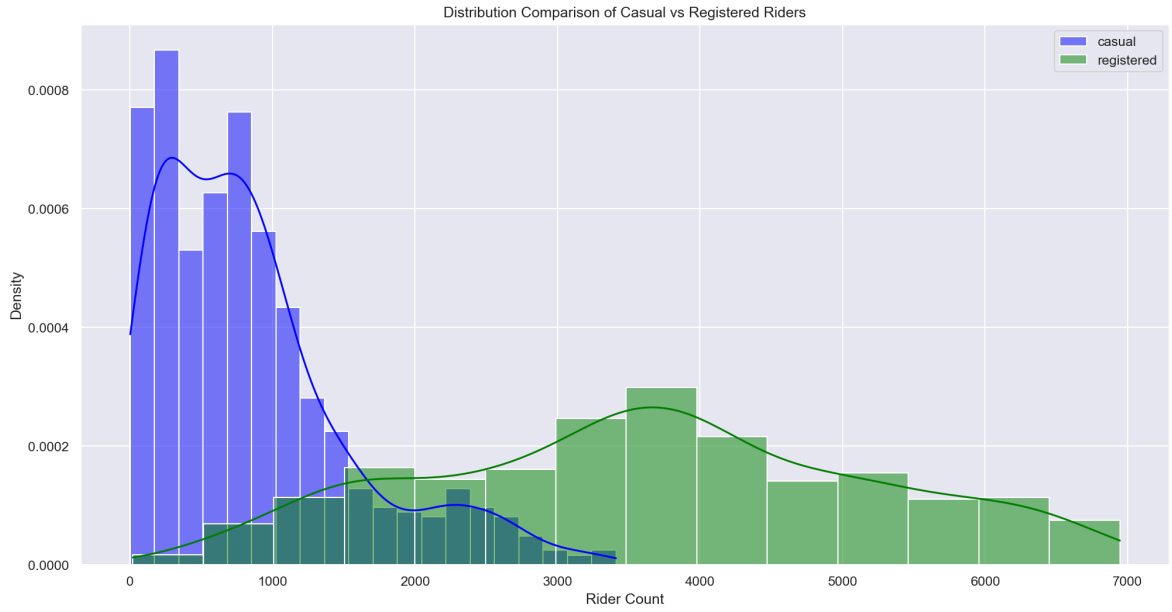
Hint: You will need to set the `stat` parameter appropriately to match the desired plot.

Include a legend, xlabel, ylabel, and title. Read the seaborn plotting tutorial 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 [ ]:
```
#GPT is used to learn how to plot the histogram with sns.histplot

sns.histplot(daily_counts['casual'], color='blue', label='casual'
             , kde=True, stat='density')
sns.histplot(daily_counts['registered'], color='green'
             , label='registered', kde=True, stat='density')

plt.xlabel("Rider Count")
plt.ylabel("Density")
plt.title("Distribution Comparison of Casual vs Registered Riders")
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x17fe33cd0>
```



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 registered riders generally have higher and more consistent daily counts compared to casual riders, who are fewer and more sporadic. This distribution difference shows that registered users are likely more regular in their usage, while casual users vary more significantly day by day.

Question 2c

The density plots do not show us how the counts for registered and casual riders vary together. Use `sns.lmplot` 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. Color the points in the scatterplot according to whether or not the day is a working day (your colors do

not have to match ours exactly, but they should be different based on whether the day is a 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`.

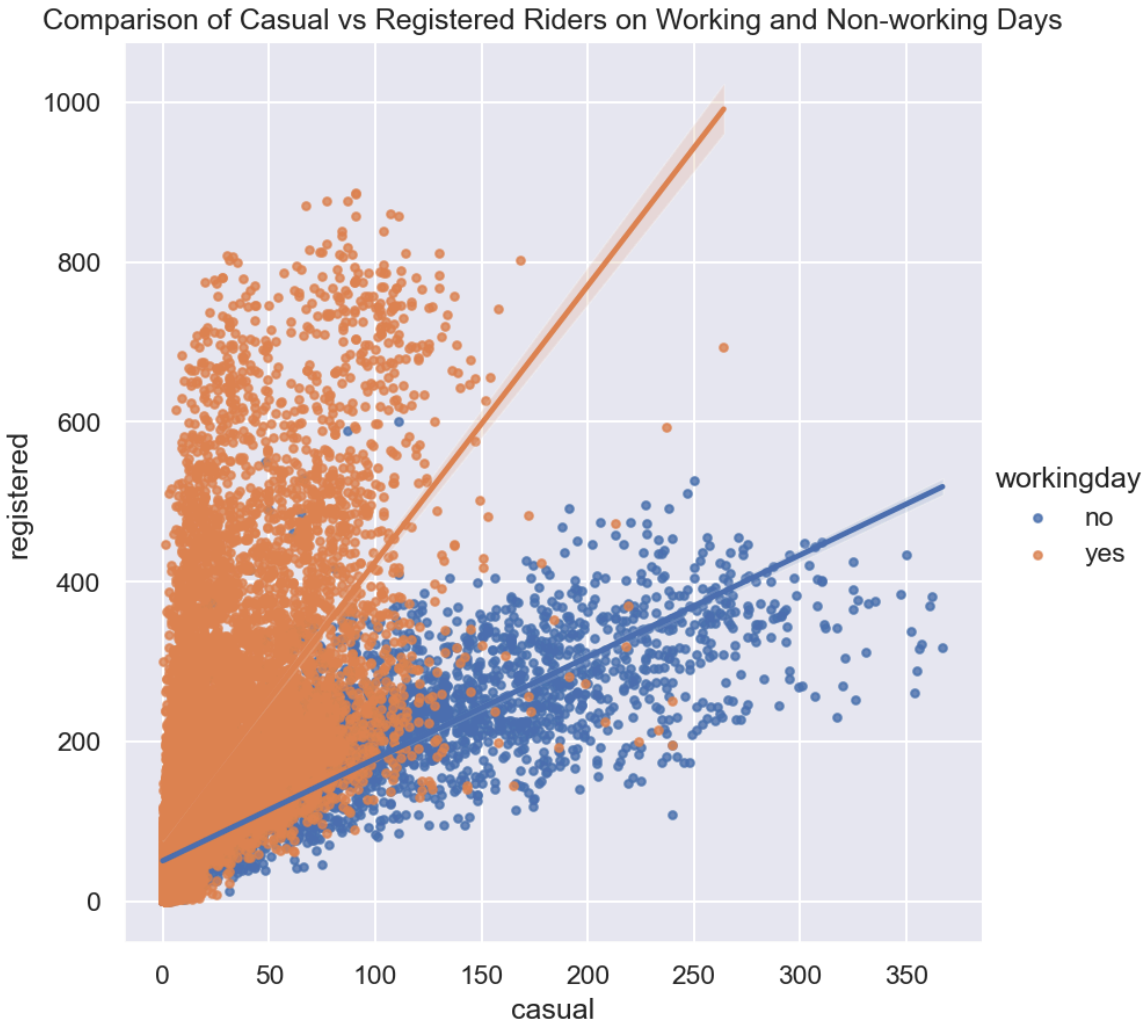
Hints:

- Checkout this helpful tutorial on `lmplot`.
- You will need to set `x`, `y`, and `hue` and the `scatter_kws` in the `sns.lmplot` call.
- You will need to call `plt.title` to add a title for the graph.

In [ ]:
```
# Make the font size a bit bigger
sns.set(font_scale=1)

sns.lmplot(
    data=bike,
    x='casual',
    y='registered',
    hue='workingday',
    fit_reg=True,
```

```
    height=6,
    scatter_kws={'s': 10}
)

# Adding title
plt.title("Comparison of Casual vs Registered Riders on Working and Non-w
```

Out[ ]: Text(0.5, 1.0, '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 have on your ability to describe this relationship?

The plot shows that bike usage patterns differ significantly between working and non-working days, with registered riders dominating working days and casual riders being more prevalent on non-working days. Overplotting limits the clarity of specific data points but still allows us to identify these broader trends.

3: Visualization

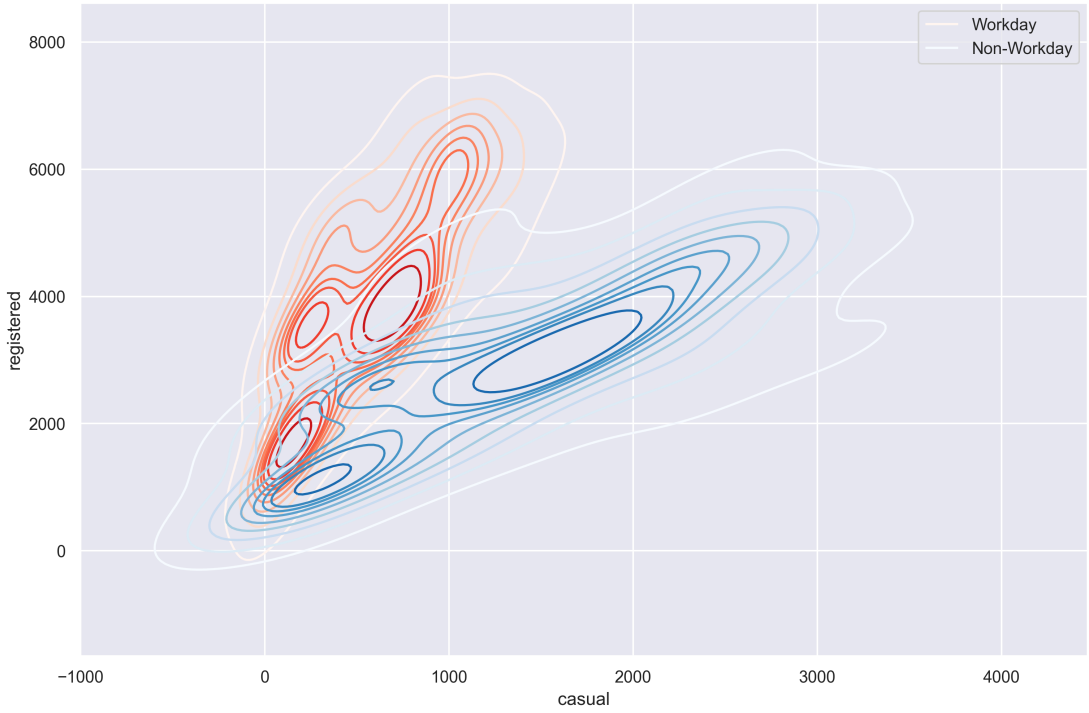
Question 3

Question 3a Bivariate Kernel Density Plot

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

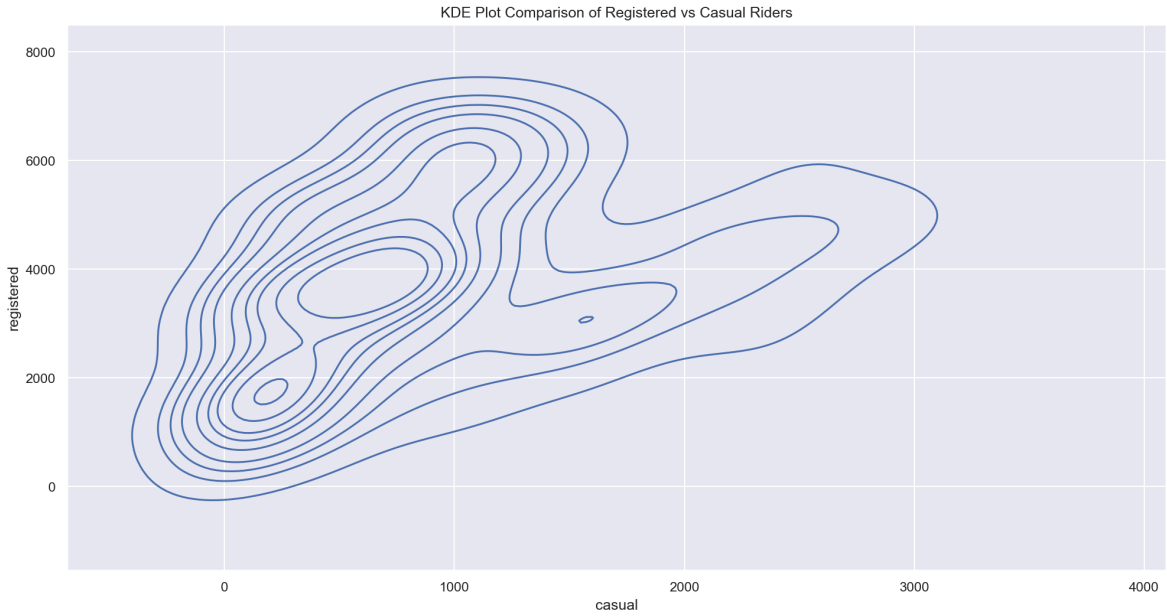
You will want to read up on the documentation for `sns.kdeplot`, which can be found here.

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



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 [ ]:
```
# GPT is used to learn more about kdeplot()

sns.kdeplot(x=daily_counts['casual'], y=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 if you do not want to follow the walkthrough.

Hints:

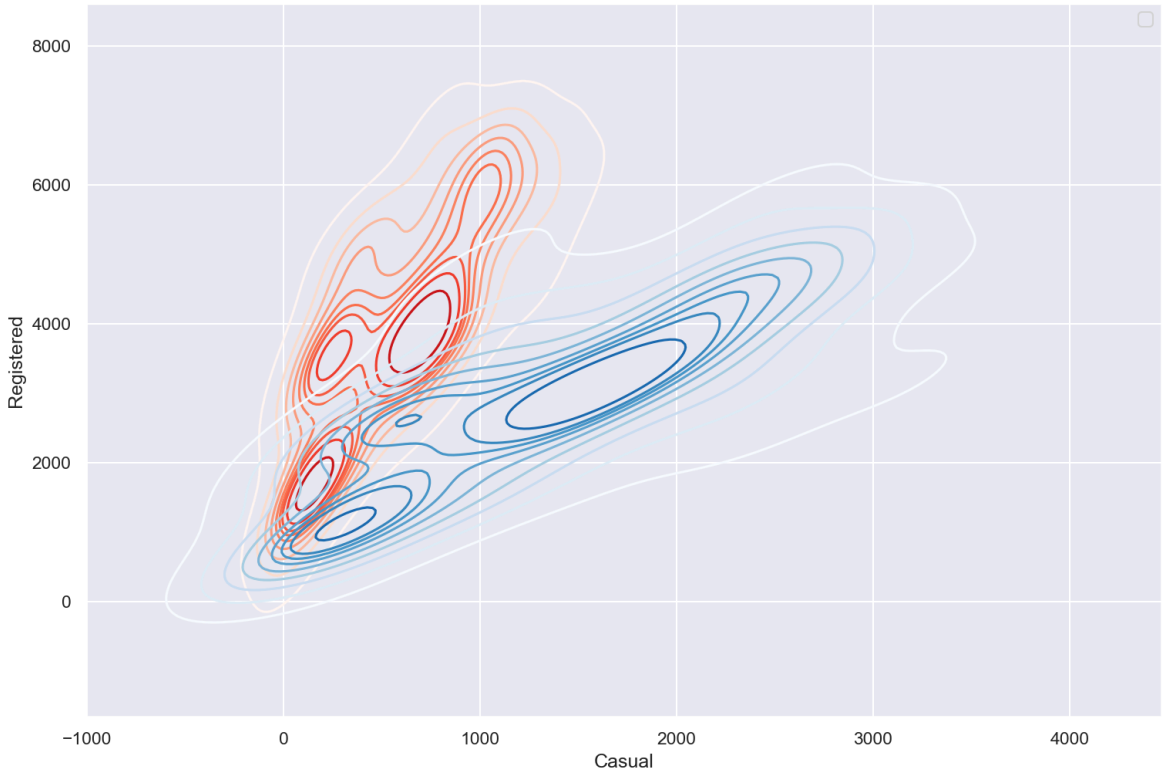
- You can use `loc` with a boolean array and column names at the same time
- You will need to call kdeplot twice, each time drawing different data from the `daily_counts` table.
- Check out this guide to see an example of how to create a legend. In particular, look at how the example in the guide makes use of the `label` argument in the call to `plt.plot()` and what the `plt.legend()` call does. This is 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), and also set the `label` parameter to address which type of day you want to plot. You are required for this question to use two sets of contrasting colors for your plots.

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 [ ]:
```
# Set the figure size for the plot
plt.figure(figsize=(12,8))

# Set 'is_workingday' to a boolean array that is true for all working_day
is_workingday = daily_counts['workingday'] == 'yes'

# Bivariate KDEs require two data inputs.
# In this case, we will need the daily counts for casual and registered r
# Hint: consider using the .loc method here.
casual_workday = daily_counts.loc[is_workingday, 'casual']
registered_workday = daily_counts.loc[is_workingday, 'registered']

# Use sns.kdeplot on the two variables above to plot the bivariate KDE fo
```

```
sns.kdeplot(
    x=casual_workday,
    y=registered_workday,
    cmap="Reds",
    shade=False,
    label="Working Days"
)

not_workingday = daily_counts['workingday'] == 'no'
# Repeat the same steps above but for rows corresponding to non-workingda
# Hint: Again, consider using the .loc method here.
casual_non_workday = daily_counts.loc[not_workingday, 'casual']
registered_non_workday = daily_counts.loc[not_workingday, 'registered']

# Use sns.kdeplot on the two variables above to plot the bivariate KDE fo
sns.kdeplot(
    x=casual_non_workday,
    y=registered_non_workday,
    cmap="Blues",
    shade=False,
    label="Non-working Days"
)

plt.xlabel("Casual")
plt.ylabel("Registered")

plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x139b6a970>



Question 3bi

In your own words, describe what the lines and the color shades of the lines signify about the data.

GPT is used to clarify my analysis

The lines in the contour plot represent levels of density for the registered and casual riders. Each line (or contour) encloses an area where there is a similar density of data points, with lines closer to the center indicating higher densities. The color shades intensify towards the center, showing areas with higher concentrations of data. In this plot, the red contours (working days) are concentrated at higher registered rider counts with lower casual rider counts, while the blue contours (non-working days) have a wider distribution across both casual and registered rider counts, indicating different patterns of bike usage depending on the day type.

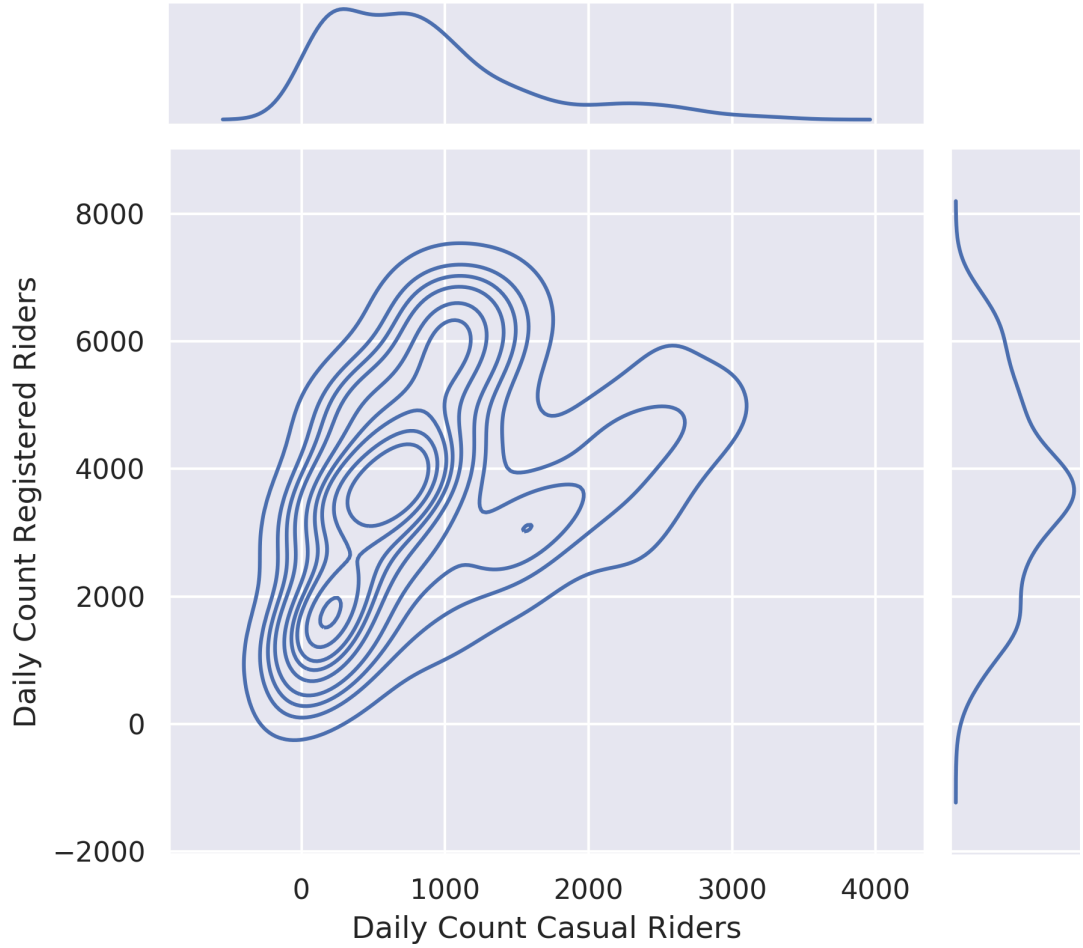
Question 3bii

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

The contour plot provides a clearer picture of the distribution shape and density for both working and non-working days, which was hard to see in the scatter plot due to overplotting. The contours show that on working days, registered riders dominate, with a high density around 4000 registered riders and fewer casual riders. In contrast, on non-working days, the density is spread more evenly across casual and registered riders, indicating that both types of users ride more frequently on non-working days. The contour plot also highlights where the distributions overlap and diverge, providing insights into the different usage patterns more effectively than the scattered points alone.

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:

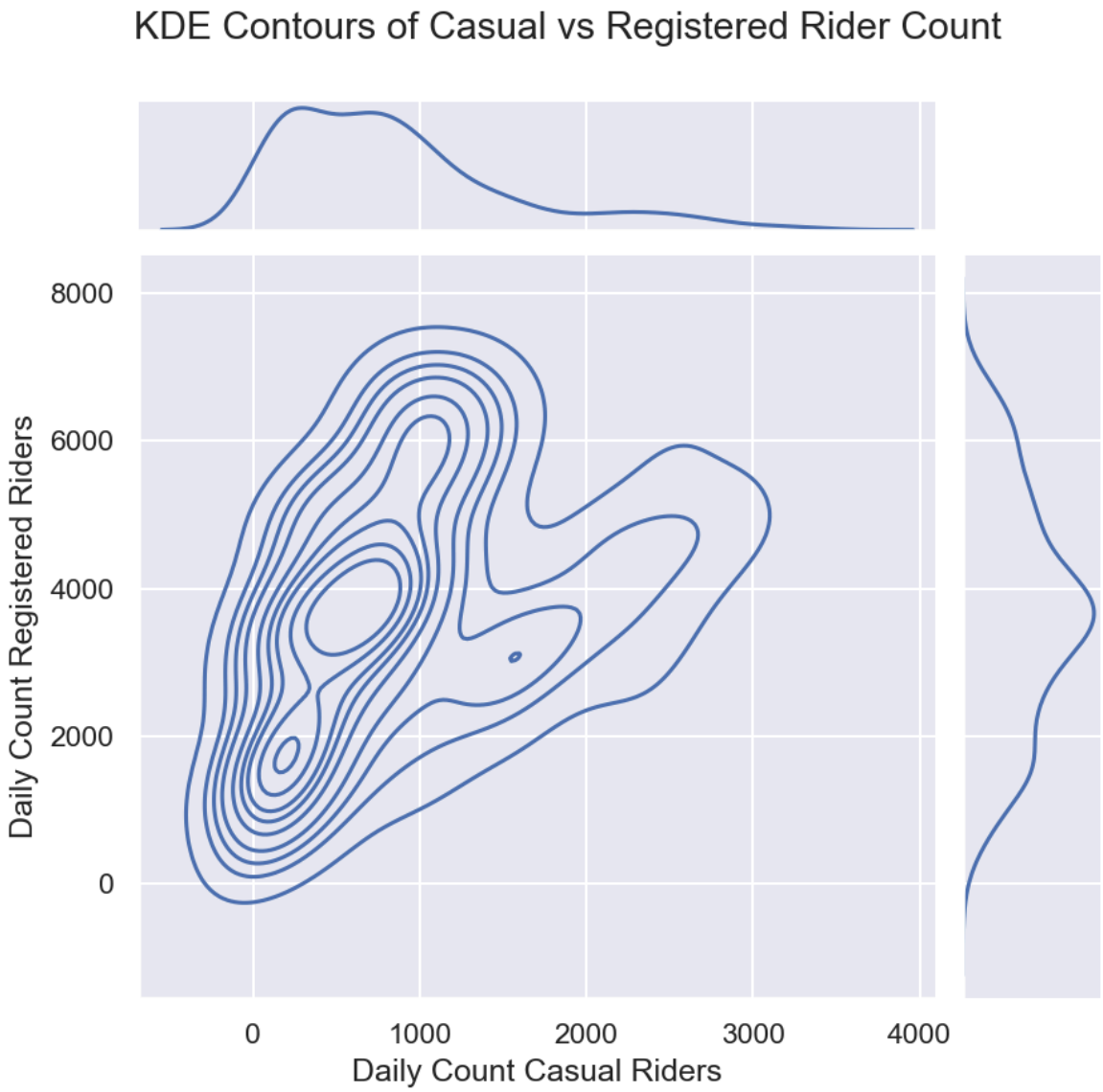
- The seaborn plotting tutorial has examples that may be helpful.
- Take a look at `sns.jointplot` and its `kind` parameter.
- `set_axis_labels` can be used to rename axes on the contour plot.

Note:

- At the end of the cell, we called `plt.suptitle` to set a custom location for the title.
- We also called `plt.subplots_adjust(top=0.9)` in case your title overlaps with your plot.

In [ ]:
```
#GPT is used to learn more about sns.jointplot()

joint_plot = sns.jointplot(
    data=daily_counts,
    x='casual',
    y='registered',
    kind='kde',
    fill=False
)

# Set axis labels and title
joint_plot.set_axis_labels("Daily Count Casual Riders"
```

```
                              , "Daily Count Registered Riders"
                              , fontsize=12)
plt.suptitle("KDE Contours of Casual vs Registered Rider Count")
plt.subplots_adjust(top=0.9);
```



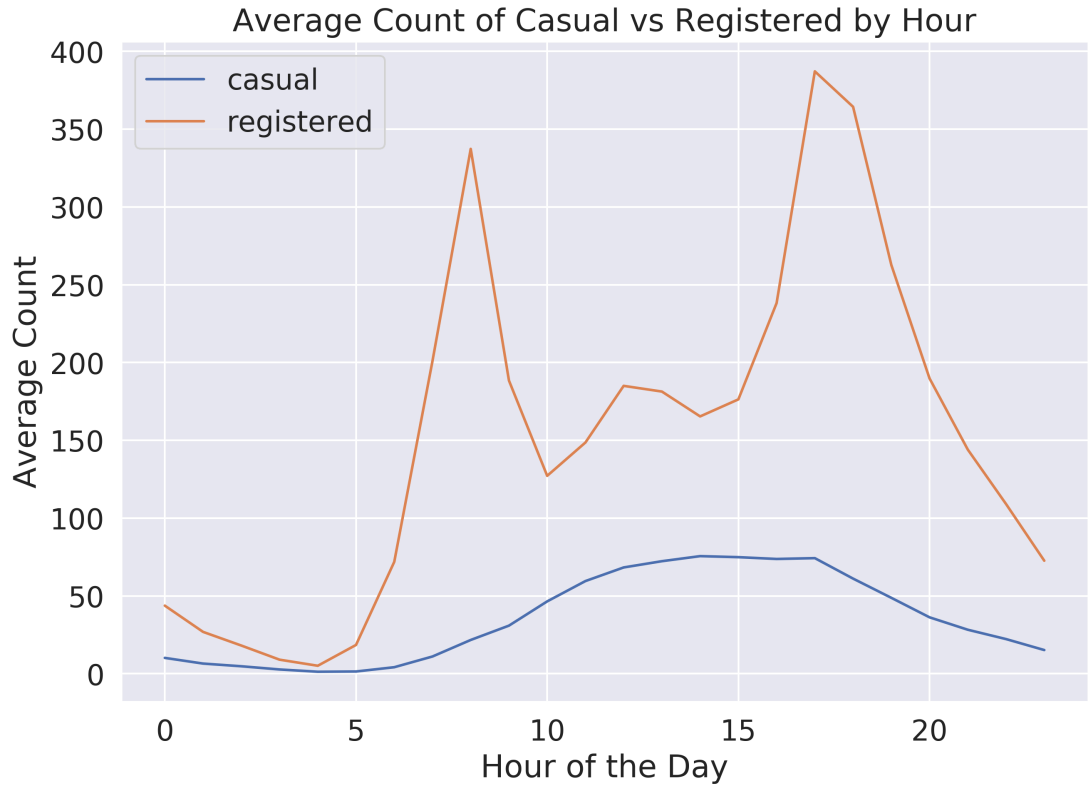
5: Understanding Daily Patterns

Question 5

Question 5a

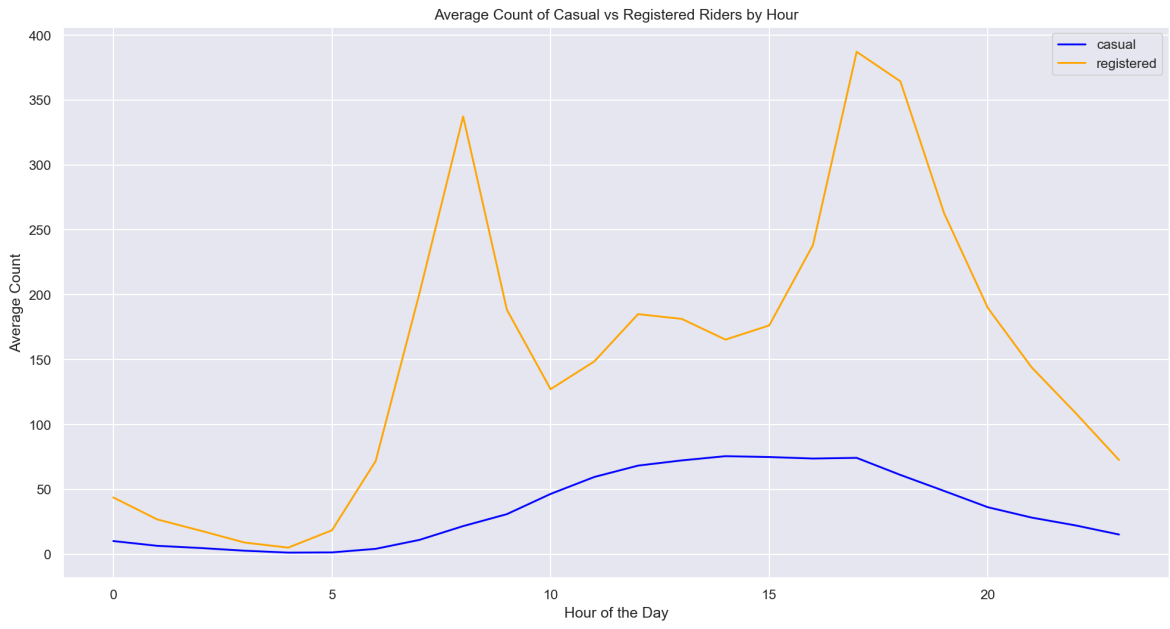
Let's examine the behavior of riders by plotting the average number of riders for each hour of the day over the **entire dataset**, stratified by rider type.

Your plot should look like the plot below. While we don't expect your plot's colors to match ours exactly, your plot should have different colored lines for different kinds of riders.



In [ ]:
```
hourly_avg = bike.groupby('hr')[['casual', 'registered']].mean()

# Plot the average counts
plt.plot(hourly_avg.index, hourly_avg['casual']
         , label='casual', color='blue')
plt.plot(hourly_avg.index, hourly_avg['registered'], label='registered',

plt.xlabel("Hour of the Day")
plt.ylabel("Average Count")
plt.title("Average Count of Casual vs Registered Riders by Hour")
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x1383a5d30>



Question 5b

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

The registered bike peak around 8 AM and 5 PM in the registered riders' distribution likely represent commute times. The 8 AM peak corresponds with the morning rush hour, while the 5 PM peak aligns with the end of the workday, when people are likely returning 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 for each record.

In [ ]:
```
bike['prop_casual'] = bike['casual'] / (bike['casual'] + bike['registered

bike
```

Out[ ]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	wea
0	1	2011-01-01	1	0	1	0	no	Sat	no	
1	2	2011-01-01	1	0	1	1	no	Sat	no	
2	3	2011-01-01	1	0	1	2	no	Sat	no	
3	4	2011-01-01	1	0	1	3	no	Sat	no	
4	5	2011-01-01	1	0	1	4	no	Sat	no	
...	...	...	...	...	...	...	...	...	...	
17374	17375	2012-12-31	1	1	12	19	no	Mon	yes	
17375	17376	2012-12-31	1	1	12	20	no	Mon	yes	
17376	17377	2012-12-31	1	1	12	21	no	Mon	yes	
17377	17378	2012-12-31	1	1	12	22	no	Mon	yes	
17378	17379	2012-12-31	1	1	12	23	no	Mon	yes	

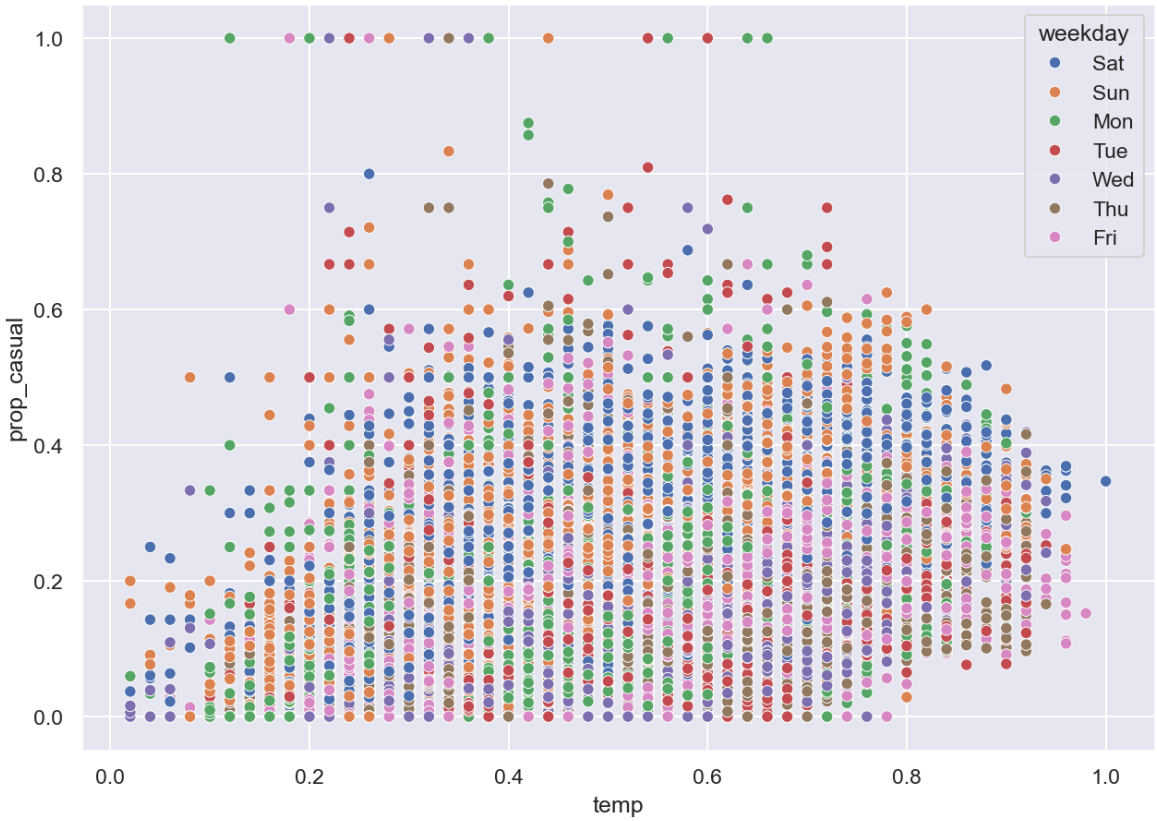
17379 rows × 18 columns

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.

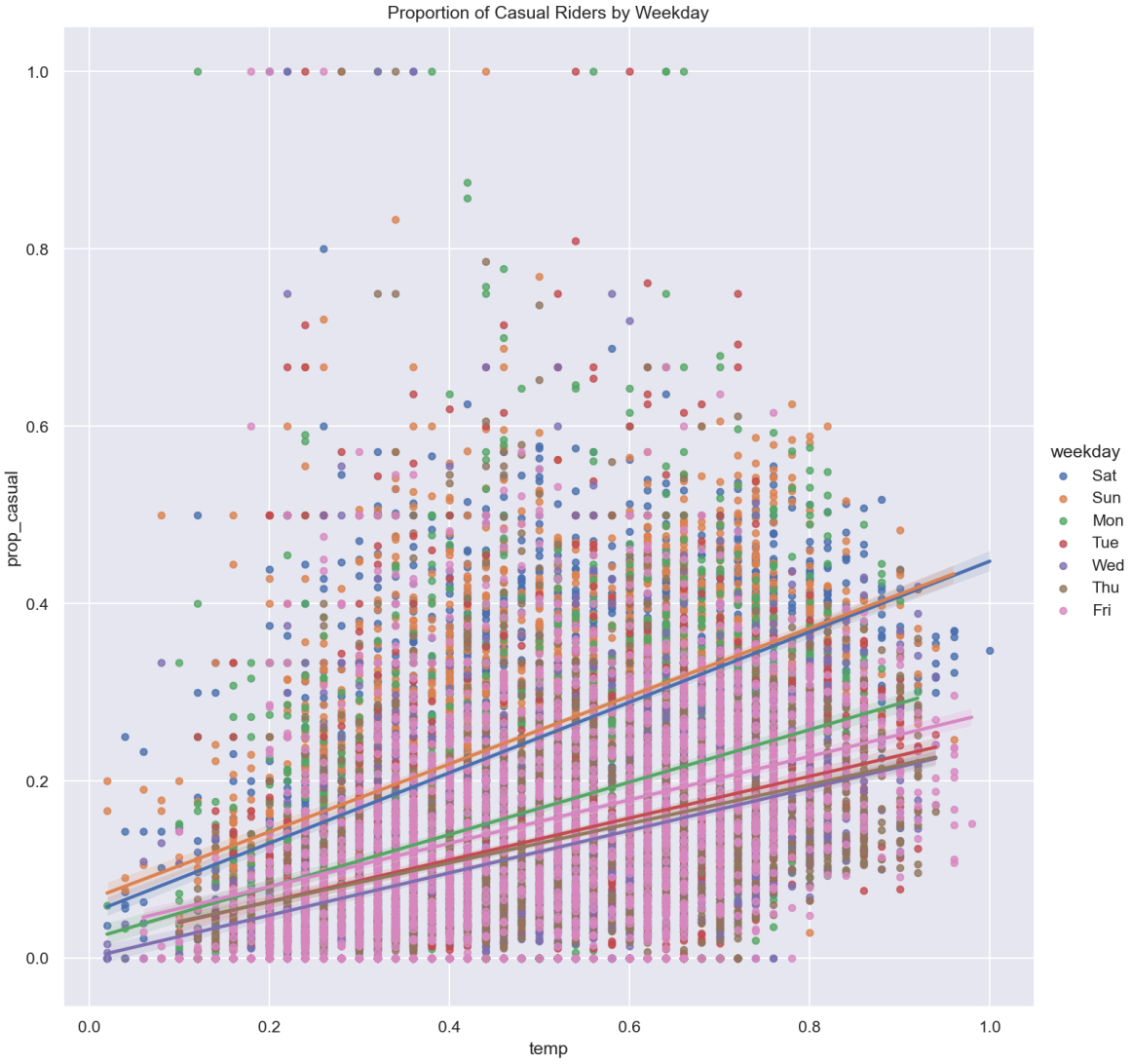
Hint: You will need to set the `data`, `x`, `y`, and `hue` in the `sns.scatterplot` call.

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



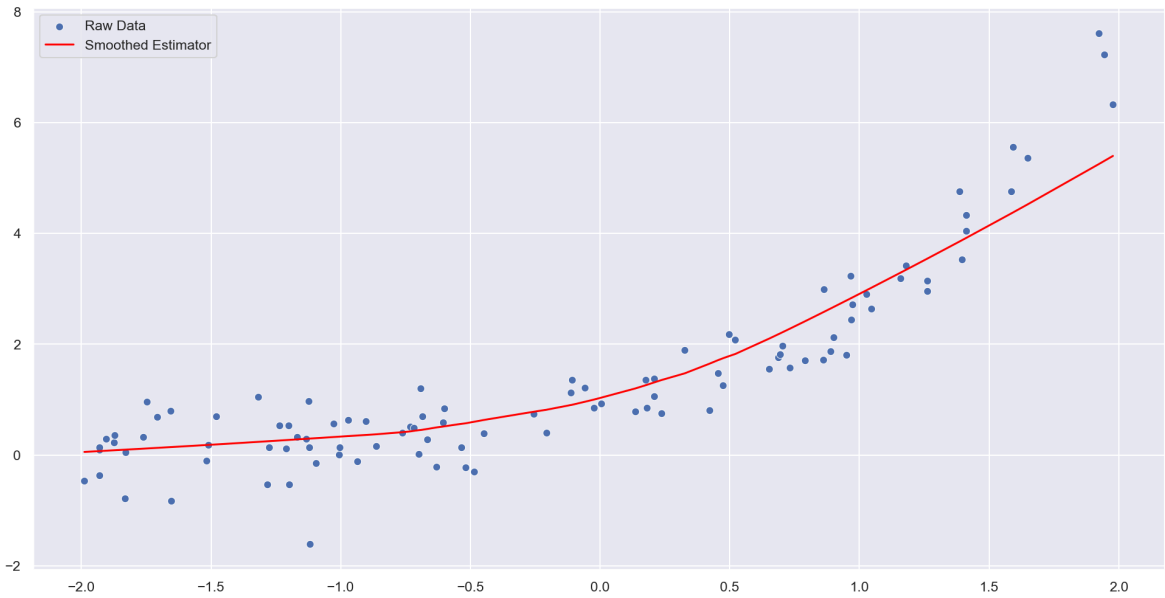
We could attempt linear regression using `sns.lmplot` as shown below, which hint at some relationships between temperature and proportional casual, but the plot is still fairly unconvincing.

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");
```

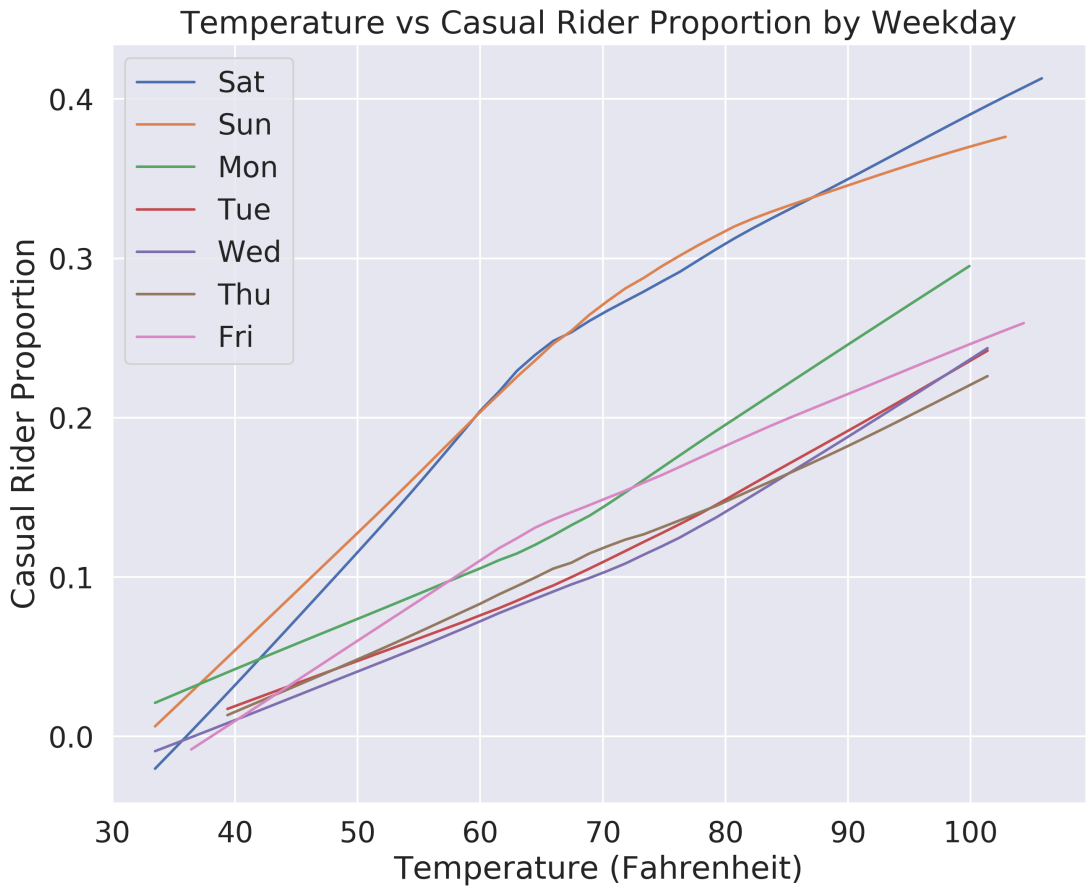


A better approach is to use local smoothing. The basic idea is that for each x value, we compute some sort of representative y value that captures the data close to that x value. One technique for local smoothing is "Locally Weighted Scatterplot Smoothing" or LOWESS. An example is below. The red curve 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(x=xobs, y=yobs, label="Raw Data")

# Predict 'smoothed' valued for observations
ysmooth = lowess(yobs, xobs, return_sorted=False)
sns.lineplot(x=xobs, y=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` 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.

You do not need to match the colors on our sample plot as long as the colors in your plot make it easy to distinguish which day they represent.

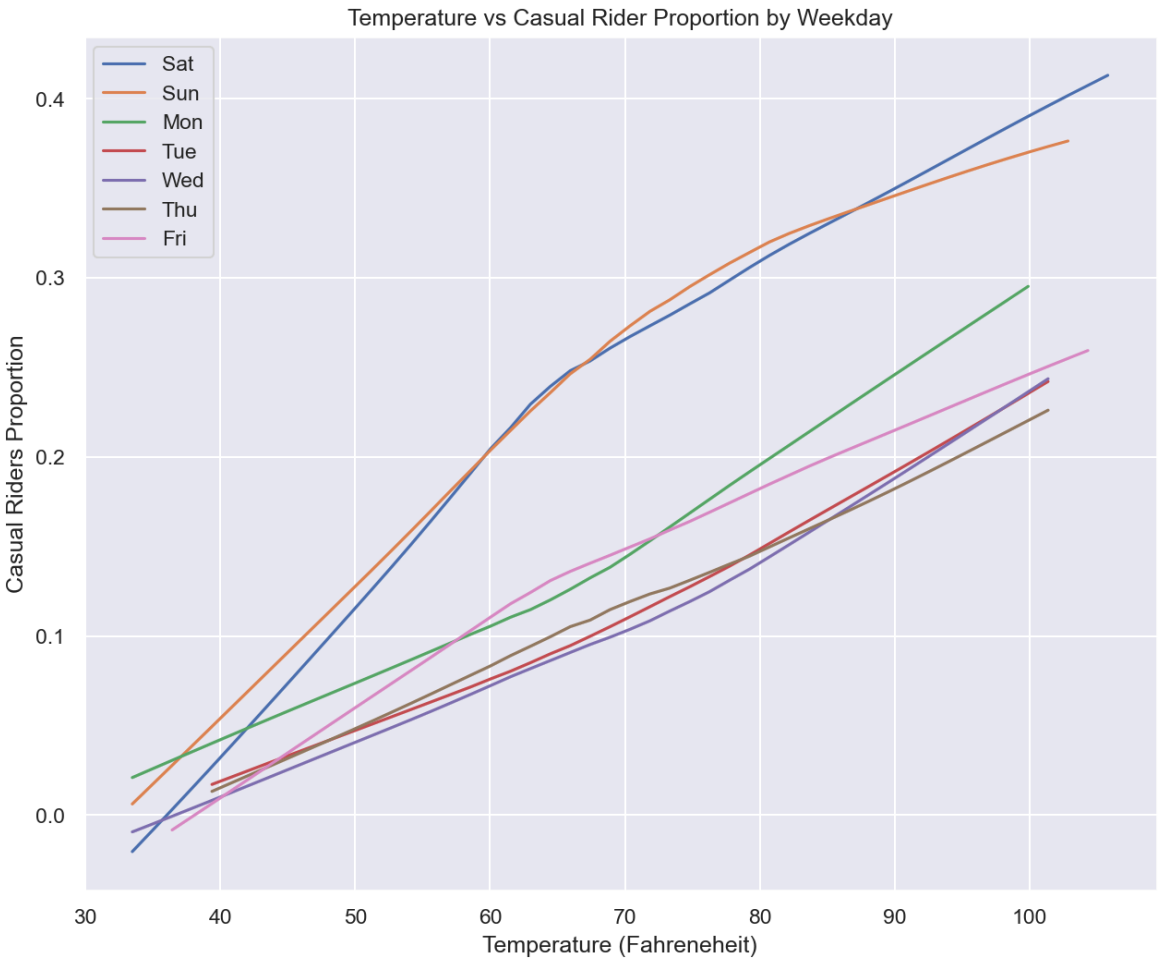
Hints:

- 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. You should also set the `return_sorted` field to `False`.
- 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 [ ]:
```
#GPT is used to learn how to use lowess()

from statsmodels.nonparametric.smoothers_lowess import lowess

plt.figure(figsize=(10,8))

# temperature = (bike[bike['weekday'] == 'Saturday'].temp.to_numpy() * 41
# prop_of_casual = bike[bike['weekday'] == 'Saturday'].prop_casual.to_num


# r = lowess(endog=prop_of_casual, exog=temperature, return_sorted=False)
# sns.lineplot(x=temperature, y=r, label="Smoothed Estimator", color='red

for day in bike['weekday'].unique():
    plotted = lowess(exog=(bike[bike['weekday'] == day]
                .temp.to_numpy() * 41 * 9/5) + 32
                , endog=bike[bike['weekday'] == day]
                .prop_casual.to_numpy(), return_sorted=False)
    sns.lineplot(y=plotted, x=(bike[bike['weekday'] == day]
                    .temp.to_numpy() * 41 * 9/5) + 32, label=day[0:3])

plt.xlabel('Temperature (Fahreneheit)')
plt.ylabel('Casual Riders Proportion')
plt.title('Temperature vs Casual Rider Proportion by Weekday')
```

Out[ ]:
```
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?

The plot shows that casual rider proportion increases with temperature, with weekends showing a consistently higher proportion of casual riders than weekdays, especially as temperatures rise. This observation supports the idea that casual ridership is influenced by both the temperature and the day of the week

7: Expanding our Analysis

Question 7

Question 7A

Imagine you are working for a Bike Sharing Company that collaborates with city planners, transportation agencies, and policy makers in order to implement bike sharing in a city. These stakeholders would like to reduce congestion and lower transportation costs. They also want to ensure the bike sharing program is

implemented equitably. In this sense, equity is a social value that is informing the deployment and assessment of your bike sharing technology.

Equity in transportation includes: improving the ability of people of different socio-economic classes, genders, races, and neighborhoods to access and afford the transportation services, and assessing how inclusive transportation systems are over time.

Do you think the `bike` data as it is can help you assess equity? If so, please explain. If not, how would you change the dataset? You may discuss how you would change the granularity, what other kinds of variables you'd introduce to it, or anything else that might help you answer this question.

GPT is used to find improvement

The current dataset is are not suitable for assessing equity because it lacks of demographic and socioeconomic data. We can collect anonymized information about users, including age, gender, and race/ethnicity. Besides, we can also include data on the socio-economic status of users, such as income level or employment status.

In [ ]:
```
# Use this cell for scratch work. If you need to add more cells for scrat
```

Question 7B

Bike sharing is growing in popularity and new cities and regions are making efforts to implement bike sharing systems that complement their other transportation offerings. The goals of these efforts are to have bike sharing serve as an alternate form of transportation in order to alleviate congestion, provide geographic connectivity, reduce carbon emissions, and promote inclusion among communities.

Bike sharing systems have spread to many cities across the country. The company you work for asks you to determine the feasibility of expanding bike sharing to additional cities of the U.S.

Based on your plots in this assignment, what would you recommend and why? Please list at least two reasons why, and mention which plot(s) you drew you analysis from.

Note: There isn't a set right or wrong answer for this question, feel free to come up with your own conclusions based on evidence from your plots!

GPT is used to check opinions

Based on the above plots, I recommend expanding bike sharing to more U.S. cities because. From the High Usage by Registered Riders' plot, the plot shows usage patterns by registered riders indicates consistent high demand, especially during

peak commuting hours, suggesting that bike sharing effectively supports daily transportation needs and could reduce congestion. The temperature vs. casual rider proportion plot shows increased usage as temperatures rise, suggesting that bike sharing could be popular in cities with mild to warm climates, making it a feasible addition to other cities with similar conditions.

In [ ]:
```
# Use this cell for scratch work. If you need to add more cells for scrat
```

Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. **Please save before exporting!**