Notebook

February 23, 2019

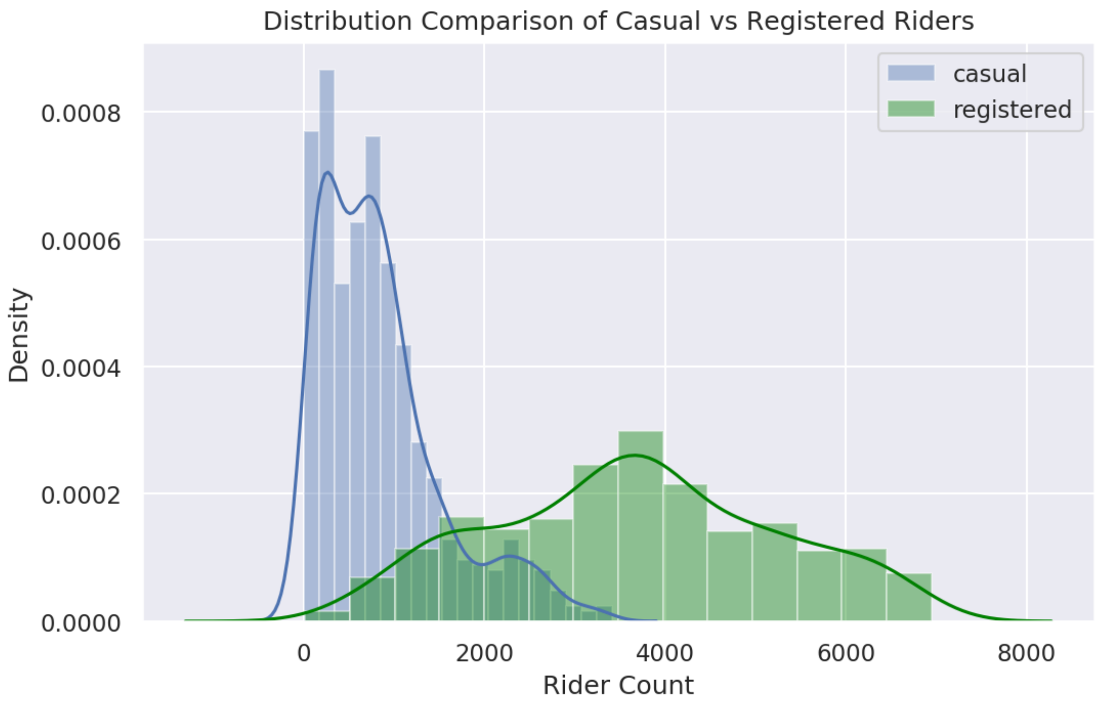
0.0.1 Question 2

Question 2a Use the `sns.distplot` 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 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', kde=True)
         plt.xlabel('Rider Count')
         plt.legend()
```

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



0.0.2 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.

0.0.3 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.

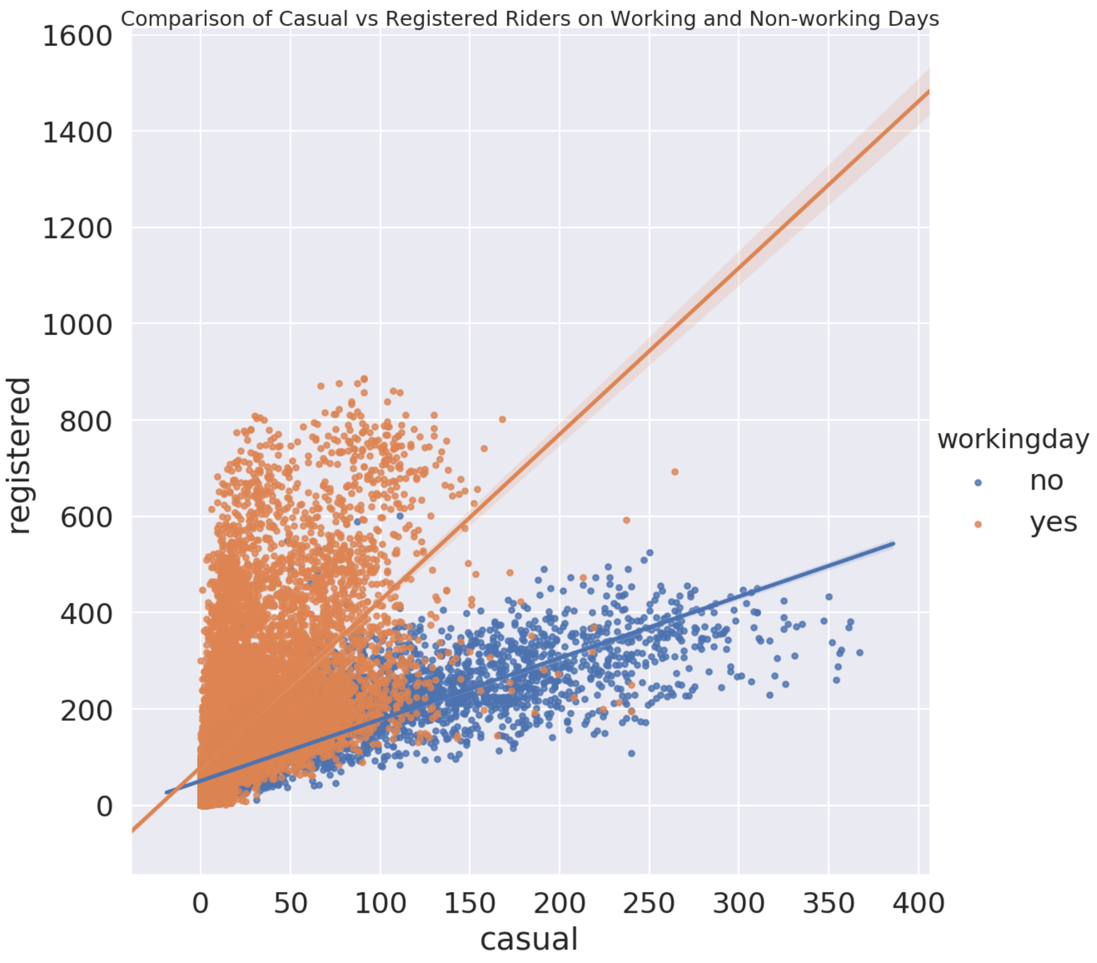
The `lmplot` function will also try to draw a linear regression line (just as you saw in Data 8). Color the points in the scatterplot according to whether or not the day is working day. There are many points in the scatter plot so make them small to help reduce overplotting. Also make sure to set `fit_reg=True`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

- 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=True, data=bike, sca
         plt.suptitle("Comparison of Casual vs Registered Riders on Working and Non-working Days", font
```

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



0.0.4 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?

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.

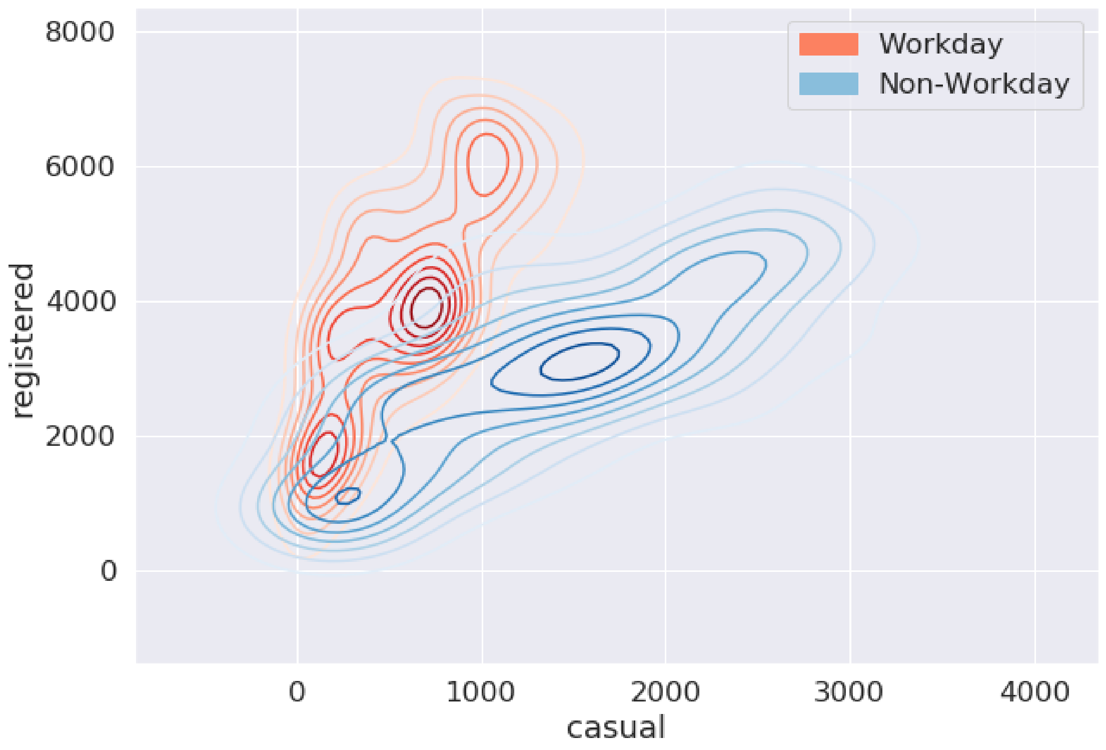
Overplotting makes it difficult to determine whether or not there are blue dots underneath the orange dots toward the bottom left hand side of the graph.

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 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 [18]: import matplotlib.patches as mpatches  # see the tutorial for how we use mpatches to generate

         # 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 riders on weekdays
         # Hint: use loc and is_workingday to splice out the relevant rows and column (casual/registere
         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 weekday rides
         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 weekday rides
         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()
```



8000
6000
4000
2000
0
registered
0
1000
2000
3000
4000
casual
Workday
Non-Workday

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.

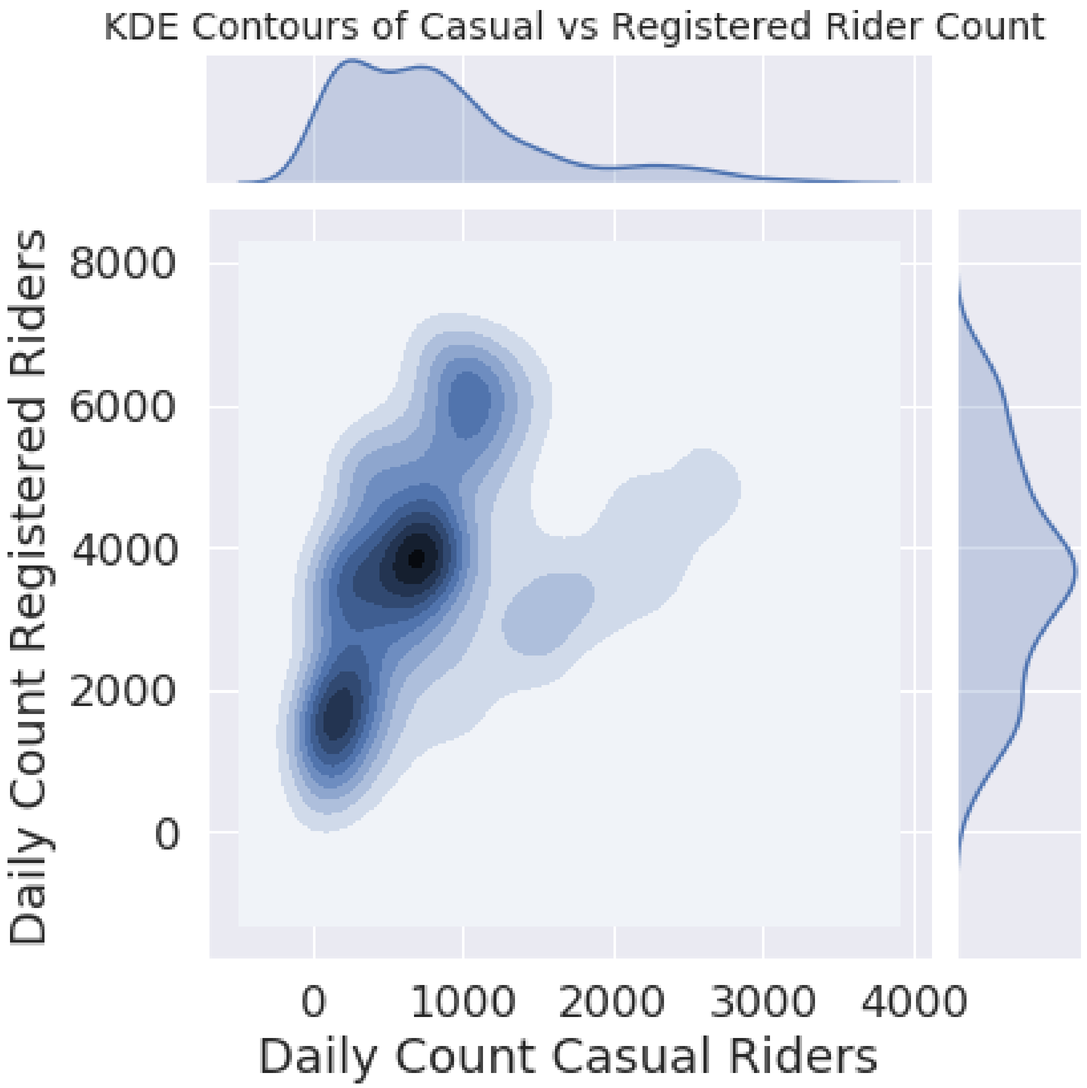
0.1 4: Joint Plot

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. * `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 [19]: g = (sns.jointplot(x="casual", y="registered", data=daily_counts, kind="kde").set_axis_labels(
         plt.suptitle('KDE Contours of Casual vs Registered Rider Count', fontsize=14)
```

```
Out[19]: Text(0.5, 0.98, 'KDE Contours of Casual vs Registered Rider Count')
```



0.2 5: Understanding Daily Patterns

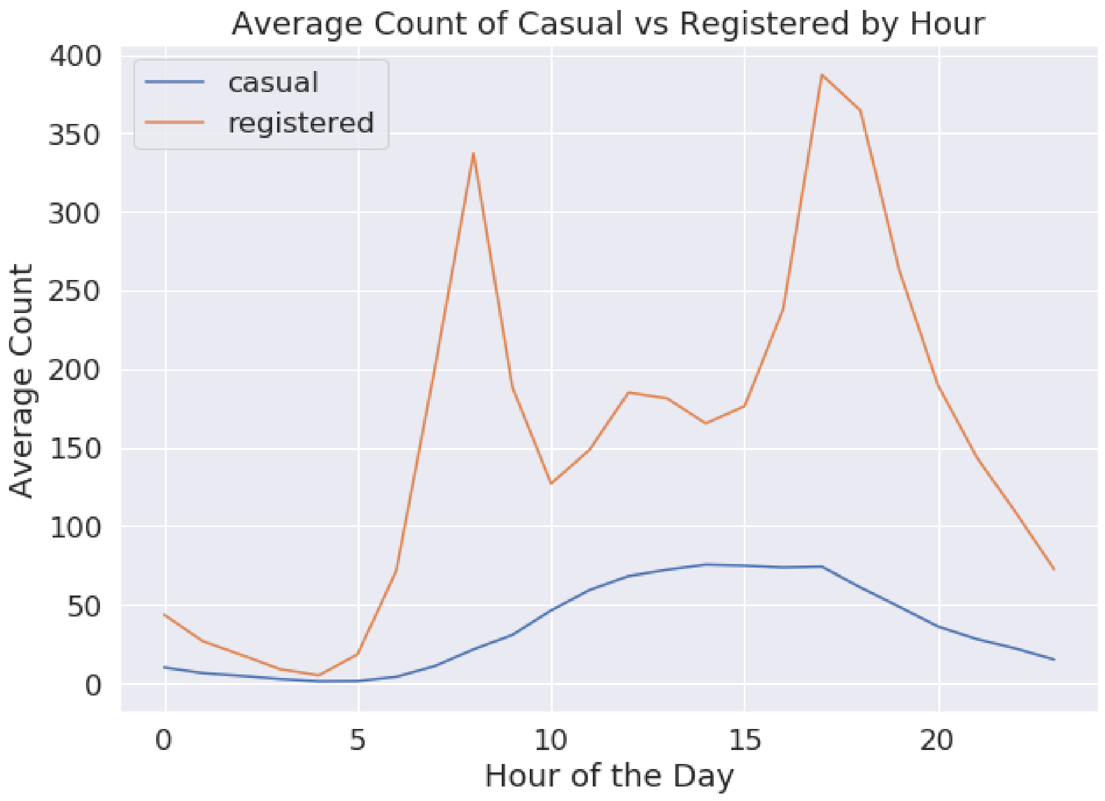
0.2.1 Question 5

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 [20]: 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="casual")
         registered_plot = sns.lineplot(x='hr', y='avgRegistered', data=df_filter, label="registered")
         casual_plot.set(xlabel='Hour of the Day', ylabel='Average Count')
         casual_plot.set_title("Average Count of Casual vs Registered by Hour")
         #plt.show()
```

```
Out[20]: 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.

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.

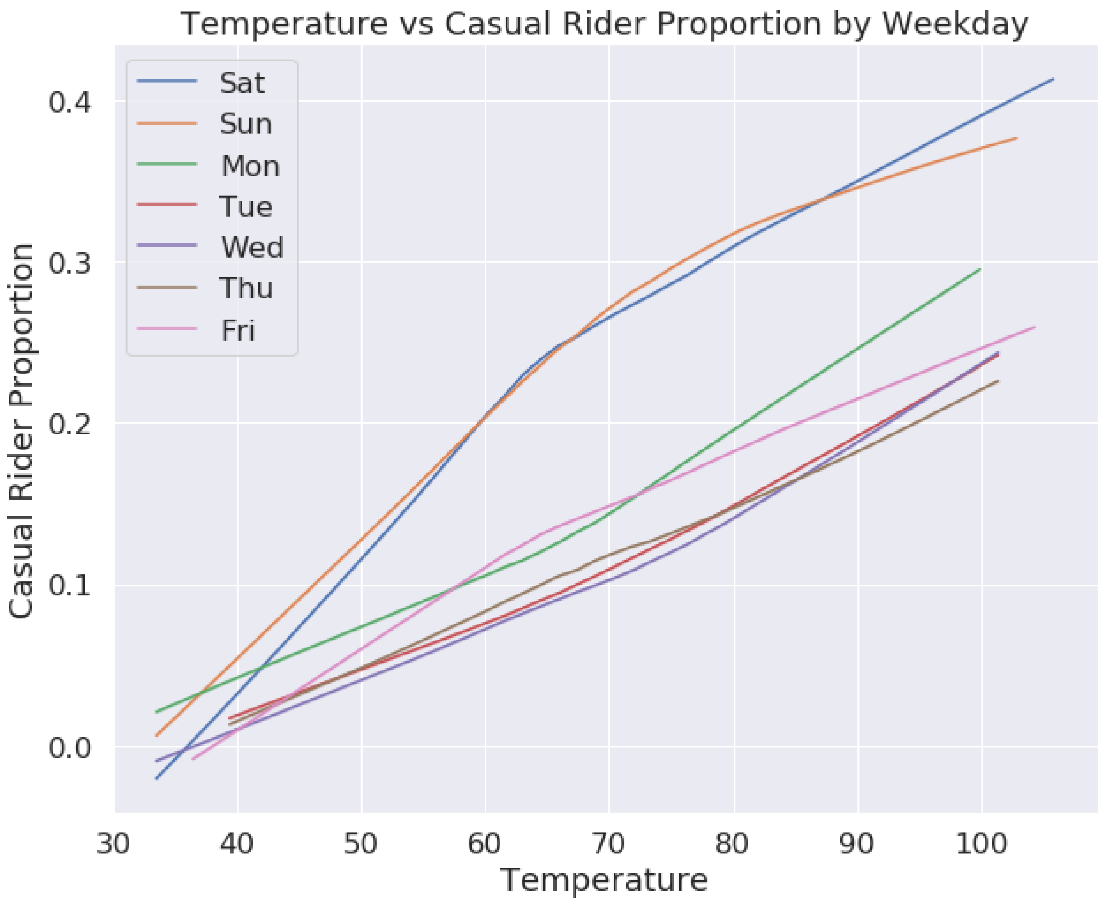
Hints: * Start by just plotting only one day of the week to make sure you can do that first.

- 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, Fahrenheit $= \text{Celsius} * \frac{9}{5} + 32$.

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

```
In [41]: 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 Proportion')
         day_plot.set_title("Temperature vs Casual Rider Proportion by Weekday")
```

```
Out[41]: Text(0.5, 1.0, 'Temperature vs Casual Rider Proportion by Weekday')
```


Temperature vs Casual Rider Proportion by Weekday
Sat
Sun
Mon
Tue
Wed
Thu
Fri
Casual Rider Proportion
0.0
0.1
0.2
0.3
0.4
Temperature
30
40
50
60
70
80
90
100

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 60žF on the weekends for the casual rider proportion to hit 0.2, whereas on Monday, it needs to hit 80žF 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 90ž Fahrenheit. I would think that once the temperature crosses 90ž 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.