Notebook

February 16, 2019

0.0.1 Question 2

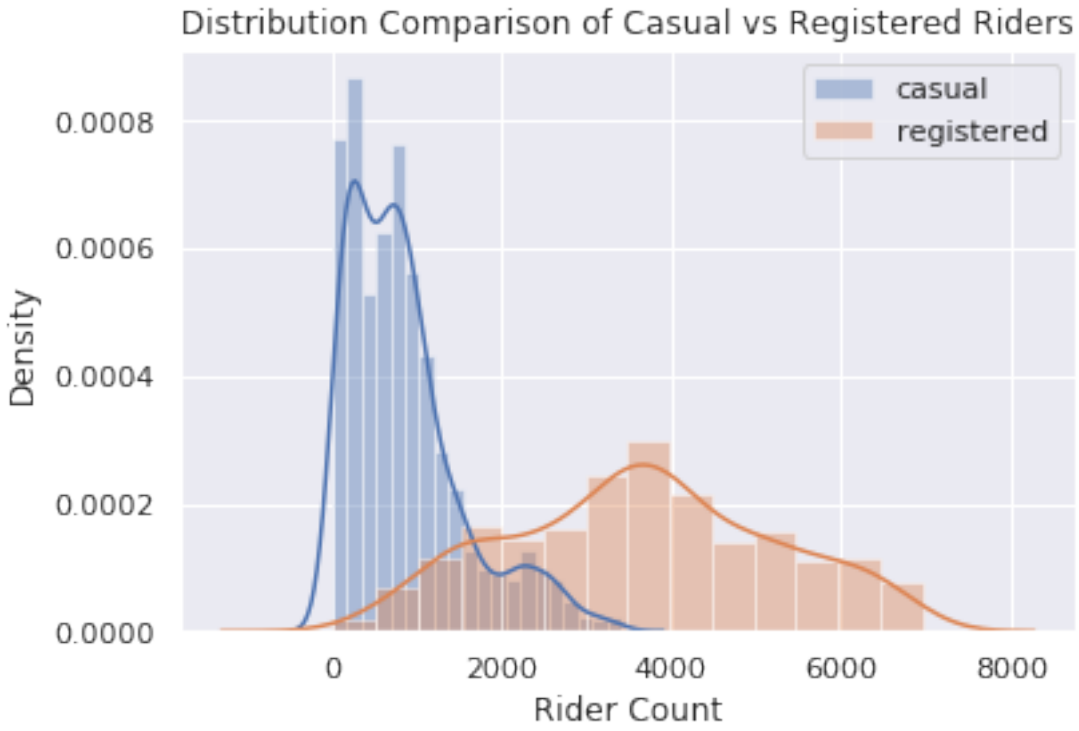
Question 2a Use the `sns.distplot` function to create a plot that overlays the distribution of the daily counts of `casual` and `registered` users. The temporal granularity of the records should be daily counts, which you should have after completing question 1c.

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 [14]: sns.distplot(daily_counts['casual'], kde = True, label = 'casual')
         sns.distplot(daily_counts['registered'], kde = True, label = 'registered')

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

```
/srv/conda/envs/data100/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a no
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



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.

There's a lot of variance on how much registered riders rent a bike day to day. This density is unimodal, centered, has a mode of 3,500 - 4,000, with few gaps and outliers. On the other hand, there's relatively small variance on how often casual riders rent a bike day to day. This density is bimodal with a right skew and a gap at around 350-500 with a tail on the right. This density indicates that relatively few casual riders rent on any given day with small variance.

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 [ ]: # Make the font size a bit bigger
        sns.set(font_scale=1.5)
        sns.lmplot(data = bike, x = 'casual', y = 'registered', hue = 'workingday', scatter_kws = {'s':

        plt.title('Comparison of Casual vs Registered Rider 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?

The proportion of casual riders to registered riders on non-working days is relatively higher compared to working days (i.e. the proportion of registered riders on working days is relatively higher). On weekdays, the number of casual and registered riders seems to have a relatively linear relationship.

Overplotting makes it difficult to visually determine the actual density at highly dense regions of the plot. In other words all sufficiently dense areas look the same on the plot.

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

# Set 'is_workingday' to a boolean array that is true for all working_days
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 riders on weekdays
# Hint: use loc and is_workingday to splice out the relevant rows and column (casual/registered
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', label = 'Workday')

# Repeat the same steps above but for rows corresponding to non-workingdays
casual_weekend = daily_counts.loc[~is_workingday, 'casual']
registered_weekend = 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_weekend, registered_weekend, cmap = 'Blues', label = 'Non-Workday')
plt.title('KDE Plot Comparison of Registered vs Casual Riders')

workday = mpatches.Patch(color = sns.color_palette('Reds')[2], label = 'Workday')
non_workday = mpatches.Patch(sns.color_palette('Blues')[2], label = 'Non-Workday')

plt.legend(handles = [workday, non_workday]);
```

Question 3b What additional details can you identify from this contour plot that were difficult to determine from the scatter plot?

Compared to the scatter plot, one can more accurately determine the density in certain areas where overplotting might make it difficult in a scatter plot.

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 [ ]: sns.jointplot(data = daily_counts, x = 'casual', y = 'registered', kind = 'kde').set_axis_labels
        plt.suptitle('KDE Contours of Casual vs Registered Rider Count')
        plt.subplots_adjust(top=0.9);
```

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 [ ]: sns.lineplot(data = bike.groupby('hr').agg('mean')[['casual', 'registered']], dashes = False)
        plt.xlabel('Hour of the Day')
        plt.ylabel('Average Count')
        plt.legend(loc='upper left');
```

Question 5b What can you observe from the plot? Hypothesize about the meaning of the peaks in the registered riders' distribution.

Casual users generally have low usage throughout the day, regardless of hours, with a slight uptick in usage after 10am.

Registered users have large spikes in usage around 6-7am and 5pm, which trends with times when people might be going to or leaving their place of work. There is a mild spike around 11am-12noon, around lunch break time.

For both groups, very little usage is seen from midnight to 5am.

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 $=$ Celsius $* \frac{9}{5} + 32$.

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

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

    plt.figure(figsize=(10,8))
    weekdays = ['Sat', 'Sun','Mon', 'Tue', 'Wed', 'Thu', 'Fri']
    for weekday in weekdays:
        xobs = ((9 / 5) * (bike[bike['weekday'] == weekday]['temp'] * 41)) + 32
        yobs = bike[bike['weekday'] == weekday]['prop_casual']
        ysmooth = lowess(yobs, xobs, return_sorted=False)
        sns.lineplot(x = xobs, y = ysmooth, label = weekday)
    plt.title('Temperature vs Casual Rider Proportion by Weekday')
    plt.xlabel('Temperature (Fahrenheit)')
    plt.ylabel('Casual Rider Proportion')
    plt.legend();
```

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?

As temperature increases, the proportion of casual riders (compared to registered riders) increases. This effect is particularly noticeable on the weekends.