S-109A Introduction to Data Science

Homework 3 - Forecasting Bike Sharing Usage

Harvard University
Summer 2018
Instructors: Pavlos Protopapas, Kevin Rader

INSTRUCTIONS

- To submit your assignment follow the instructions given in canvas.
- Restart the kernel and run the whole notebook again before you submit.
- If you submit individually and you have worked with someone, please include the name of your [one] partner below.

Names of people you have worked with goes here:

In [1]:

```
from IPython.core.display import HTML
def css_styling(): styles = open("cs109.css", "r").read(); return HTML(styles)
css_styling()
```

Out[1]:

Main Theme: Multiple Linear Regression, Subset Selection, Polynomial Regression

Overview

You are hired by the administrators of the Capital Bikeshare program (https://www.capitalbikeshare.com) program in Washington D.C., to **help them predict the hourly demand for rental bikes** and **give them suggestions on how to increase their revenue**. You will prepare a small report for them.

The hourly demand information would be useful in planning the number of bikes that need to be available in the system on any given hour of the day, and also in monitoring traffic in the city. It costs the program money if bike stations are full and bikes cannot be returned, or empty and there are no bikes available. You will use multiple linear regression and polynomial regression and will explore techniques for subset selection. The goal is to build a regression model that can predict the total number of bike rentals in a given hour of the day, based on attributes about the hour and the day.

An example of a suggestion to increase revenue might be to offer discounts during certain times of the day either during holidays or non-holidays. Your suggestions will depend on your observations of the seasonality of ridership.

The data for this problem were collected from the Capital Bikeshare program over the course of two years (2011 and 2012).

Use only the libraries below:

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

import statsmodels.api as sm
from statsmodels.api import OLS

from sklearn import preprocessing
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from pandas.plotting import scatter_matrix

import seaborn as sns


%matplotlib inline
```

Data Exploration & Preprocessing, Multiple Linear Regression, Subset Selection

Overview

The initial data set is provided in the file `data/BSS_hour_raw.csv`. You will add some features that will help us with the analysis and then separate it into training and test sets. Each row in this file contains 12 attributes and each entry represents one hour of a 24-hour day with its weather, etc, and the number of rental rides for that day divided in categories according to if they were made by registered or casual riders. Those attributes are the following:

- `dteday` (date in the format YYYY-MM-DD, e.g. 2011-01-01)
- `season` (1 = winter, 2 = spring, 3 = summer, 4 = fall)
- `hour` (0 for 12 midnight, 1 for 1:00am, 23 for 11:00pm)
- `weekday` (0 through 6, with 0 denoting Sunday)
- `holiday` (1 = the day is a holiday, 0 = otherwise)
- `weather`
  - 1: Clear, Few clouds, Partly cloudy, Partly cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm
  - 4: Heavy Rain + Thunderstorm + Mist, Snow + Fog
- `temp` (temperature in Celsius)
- `atemp` (apparent temperature, or relative outdoor temperature, in Celsius)
- `hum` (relative humidity)
- `windspeed` (wind speed)
- `casual` (number of rides that day made by casual riders, not registered in the system)
- `registered` (number of rides that day made by registered riders)

General Hints

- Use pandas .describe() to see statistics for the dataset.
- When performing manipulations on column data it is useful and often more efficient to write a function and apply this function to the column as a whole without the need for iterating through the elements.
- A scatterplot matrix or correlation matrix are both good ways to see dependencies between multiple variables.
- For Question 2, a very useful pandas method is .groupby(). Make sure you aggregate the rest of the columns in a meaningful way. Print the dataframe to make sure all variables/columns are there!

Resources

[http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html
(http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html)](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html)

Question 1: Explore how Bike Ridership varies with Hour of the Day

Learn your Domain and Perform a bit of Feature Engineering

1.1 Load the dataset from the csv file `data/BSS_hour_raw.csv` into a pandas dataframe that you name `bikes_df`. Do any of the variables' ranges or averages seem suspect? Do the data types make sense?

1.2 Notice that the variable in column `dteday` is a pandas `object`, which is **not** useful when you want to extract the elements of the date such as the year, month, and day. Convert `dteday` into a `datetime` object to prepare it for later analysis.

1.3 Create three new columns in the dataframe:

- `year` with 0 for 2011 and 1 for 2012.
- `month` with 1 through 12, with 1 denoting Jan.
- `counts` with the total number of bike rentals for that day (this is the response variable for later).

1.4 Use visualization to inspect and comment on how **casual** rentals and **registered** rentals vary with the `hour`.

1.5 Use the variable `holiday` to show how **holidays** affect the relationship in question 1.4. What do you observe?

1.6 Use visualization to show how **weather** affects **casual** and **registered** rentals. What do you observe?

Answers

1.1 Load the dataset from the csv file ...

In [3]:

```
bikes_df = pd.read_csv("data/BSS_hour_raw.csv", index_col=0)
```

None of the variable ranges really seem suspect. The only one that seems a little confusing to me is hum. It seems weird that humidity can be so high during the winter time. For example, one in Jan says .81, which I'm guessing is 81%? Data types make sense to me.

Also, there is one issue with the temperature. It seems to only go between 0 and 1, but only in celcuis, so it must've been normalized?

1.2 Notice that the variable in column ....

In [4]:

```
pd.to_datetime(bikes_df.index)
```

Out[4]:

```
DatetimeIndex(['2011-01-01', '2011-01-01', '2011-01-01', '2011-01-01
',
               '2011-01-01', '2011-01-01', '2011-01-01', '2011-01-01
',
               '2011-01-01', '2011-01-01',
               ...
               '2012-12-31', '2012-12-31', '2012-12-31', '2012-12-31
',
               '2012-12-31', '2012-12-31', '2012-12-31', '2012-12-31
',
               '2012-12-31', '2012-12-31'],
              dtype='datetime64[ns]', name='dteday', length=17379, f
req=None)
```

1.3 Create three new columns ...

In [5]:

```
# add counts column

bikes_df['counts'] = bikes_df['casual'] + bikes_df['registered']

# add month column

bikes_df['month'] = pd.to_datetime(bikes_df.index).month

# add year column

bikes_df['years'] = pd.to_datetime(bikes_df.index).year

bikes_df["year"] = [0 if yr  == 2011 else 1 for yr in bikes_df["years"]]

# delete years column to leave just year

bikes_df = bikes_df.drop('years', axis=1)

bikes_df.head()
```

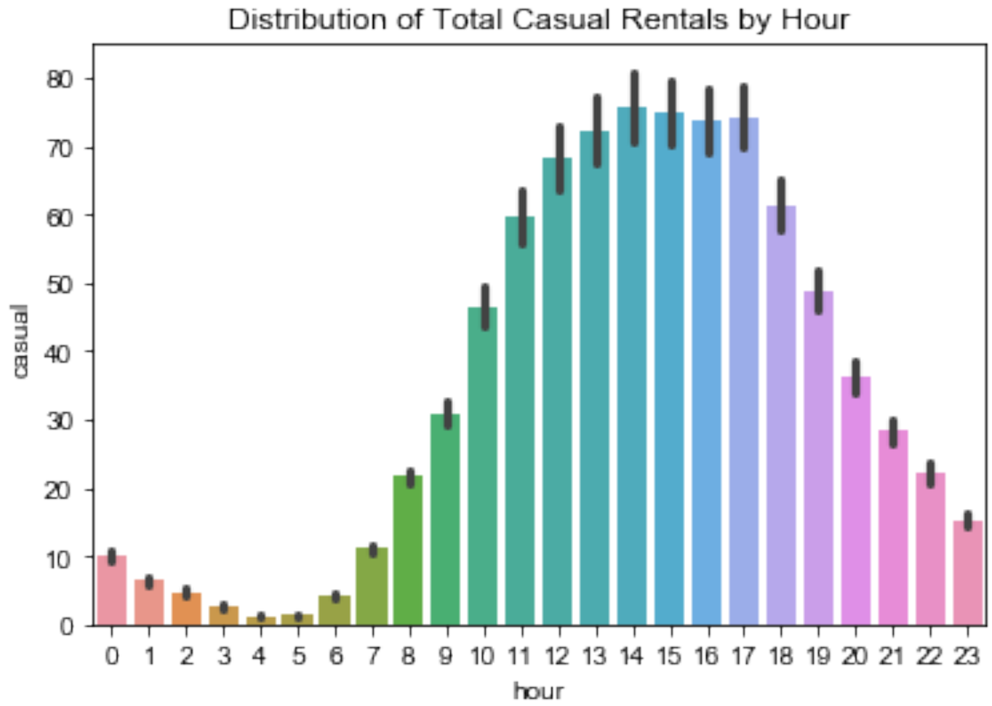
Out[5]:

	season	hour	holiday	weekday	workingday	weather	temp	atemp	hum	wir
dteday										
2011-01-01	1	0	0	6	0	1	0.24	0.2879	0.81	0.0
2011-01-01	1	1	0	6	0	1	0.22	0.2727	0.80	0.0
2011-01-01	1	2	0	6	0	1	0.22	0.2727	0.80	0.0
2011-01-01	1	3	0	6	0	1	0.24	0.2879	0.75	0.0
2011-01-01	1	4	0	6	0	1	0.24	0.2879	0.75	0.0

1.4 Use visualization to inspect and comment on how casual rentals and registered rentals vary with the hour.

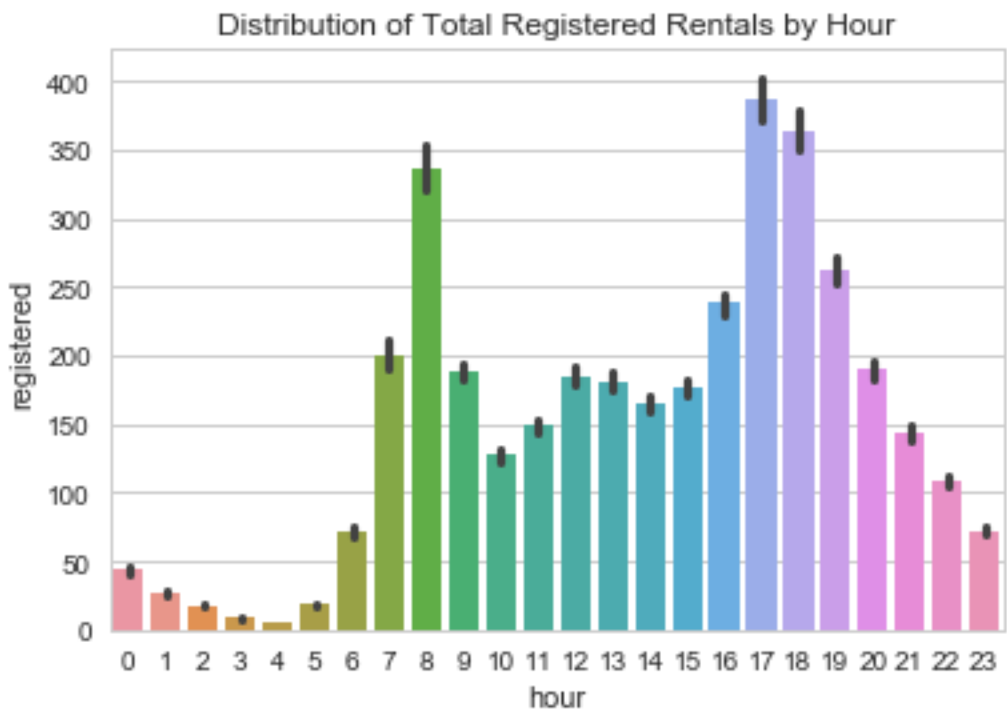
In [6]:

```
plt.title("Distribution of Total Casual Rentals by Hour")
sns.set(style="whitegrid")
ax = sns.barplot(x="hour", y="casual", data=bikes_df)
```



In [7]:

```
plt.title("Distribution of Total Registered Rentals by Hour")
sns.set(style="whitegrid")
ax = sns.barplot(x="hour", y="registered", data=bikes_df)
```



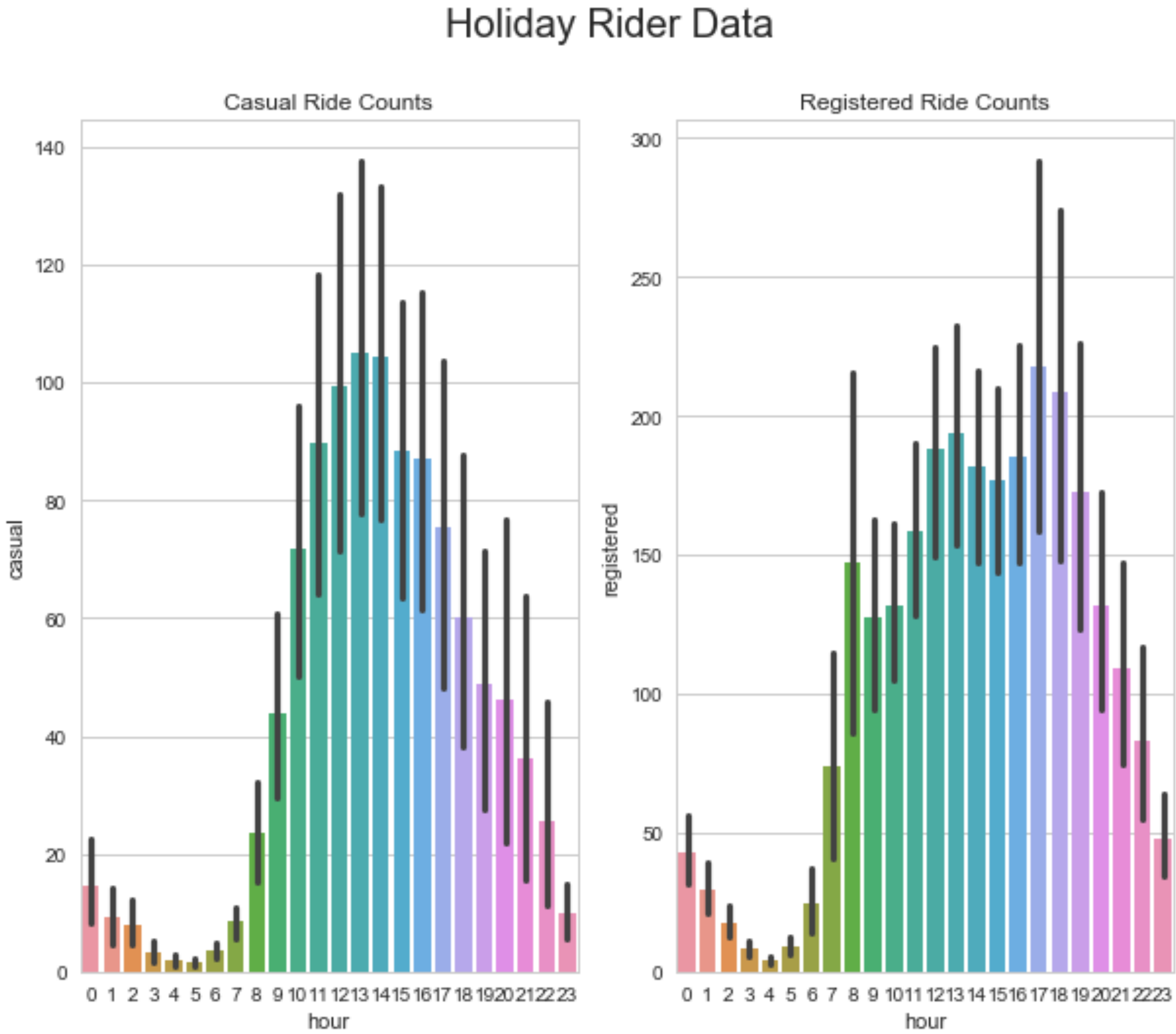
Comparing the two graphs above, registered users tend to follow the typical work schedule. Whereas the casual users are less prevelant in the morning. They seem to be common during the second half of the day; maybe going to lunch/dinner/out and then coming back home or biking around the city.

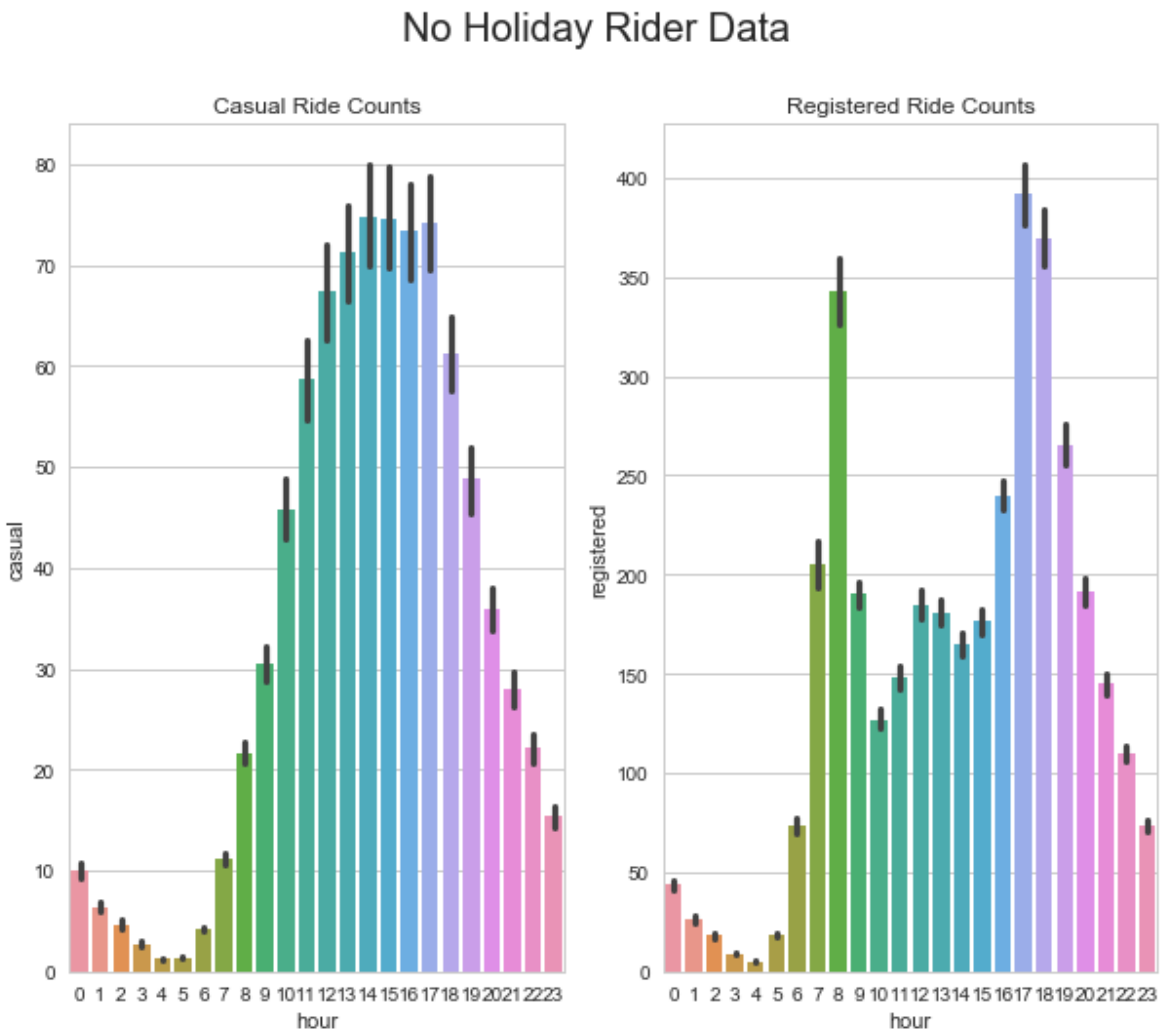
1.5 Use the variable holiday to show how holidays affect the relationship in question 1.4. What do you observe?

In [8]:

```
#holiday (casual + registered)

holiday = bikes_df[(bikes_df['holiday'] == 1)]

fig, ax = plt.subplots(1, 2, figsize=(10,8))
sns.set(style="whitegrid")
sns.barplot(x="hour", y="casual", data=holiday, ax=ax[0])
ax[0].set_title('Casual Ride Counts')
ax[1].set_title('Registered Ride Counts')
sns.barplot(x="hour", y="registered", data=holiday,ax=ax[1])
fig.suptitle('Holiday Rider Data', fontsize=20);
fig.show()

#no holiday (casual + registered)

no_holiday = bikes_df[(bikes_df['holiday'] == 0)]

fig, ax = plt.subplots(1, 2, figsize=(10,8))
sns.set(style="whitegrid")
sns.barplot(x="hour", y="casual", data=no_holiday, ax=ax[0])
sns.barplot(x="hour", y="registered", data=no_holiday,ax=ax[1])
ax[0].set_title('Casual Ride Counts')
ax[1].set_title('Registered Ride Counts')
fig.suptitle('No Holiday Rider Data', fontsize=20);
fig.show()
```

```
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```


Holiday Rider Data
Casual Ride Counts
Registered Ride Counts
casual
registered
hour
hour
0
20
40
60
80
100
120
140
0
50
100
150
200
250
300
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

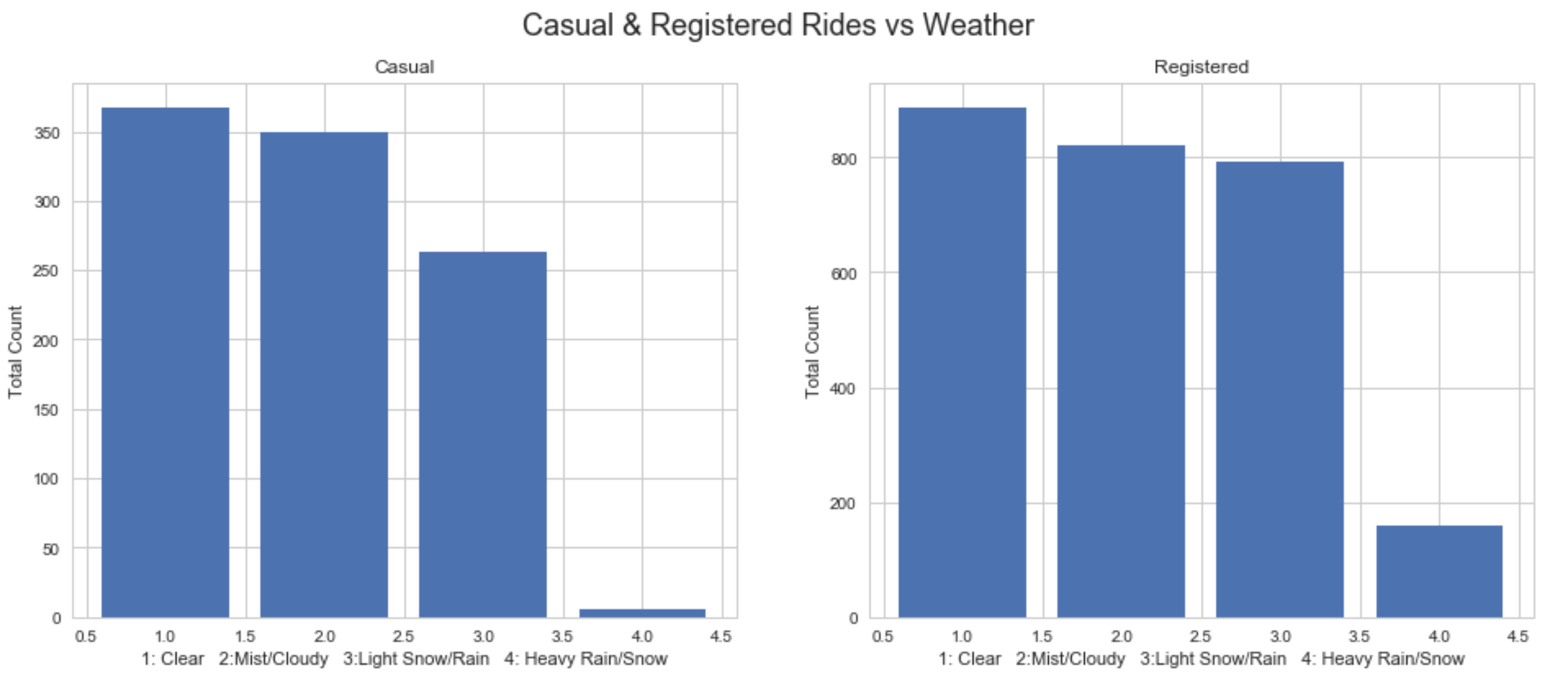


Looking at the graphs above, we can observe a few trends. The first thing that stood out to me was that there was a higher overall rider count for registered users during not holiday days, whereas for the casual riders, it was the oppposite. There was a large spike in the rider count for casual users during the holidays. Less registered users are using the service during the holidays. During the work days, the registered riders follows the workday. There is spike for the commute to work and then around spike during after work; whereas for the casual rides, it is a consistent rider count for the second half of the day even though there is less people using it. Intutiviely, this makes sense. Registered rides tend to use the service more during workdays to get to work, whereas casual rider usage will increase during holidays because they need something to do.

1.6 Use visualization to show how weather affects casual and registered rentals. What do you observe?

In [9]:

```
# 1: clear/few clouds/cloudy
# 2: mist+cloud/mist+broken clouds/mist
# 3: light snow, light rain + thudnerstorm
# 4: heavy rain/thunder/snow+fog

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(16,6))
ax1.bar(bikes_df['weather'], bikes_df['casual'])
ax1.set_xlabel("1: Clear   2:Mist/Cloudy   3:Light Snow/Rain   4: Heavy Rain/Sno
w")
ax2.set_xlabel("1: Clear   2:Mist/Cloudy   3:Light Snow/Rain   4: Heavy Rain/Sno
w")
ax1.set_ylabel("Total Count")
ax2.set_ylabel("Total Count")
ax1.set_title('Casual')
ax2.set_title('Registered')
ax2.bar(bikes_df['weather'], bikes_df['registered'])
f.suptitle('Casual & Registered Rides vs Weather', fontsize=18);
```



From looking at the two graphs above, both graphs exhibit a similar overall behaviour. As the weather gets worse (1 -> 4), there is less total rides. However, one important thing I noticed was that registered riders used the service more when the weather was poor; this makes sense because I would assume that people who use it everyday, tend to use it everyday. Another (obvious) obsevration is that casual users dont use the service when the weather is really bad out. This makes perfect sense. Overall, ride counts for the registered customers is significantly higher than casual in all weather conditions.

Question 2: Explore Seasonality on Bike Ridership.

Seasonality and weather

Now let's examine the effect of weather and time of the year. For example, you want to see how ridership varies with season of the year.

2.1 Make a new dataframe with the following subset of attributes from the previous dataset and with each entry being **ONE** day:

- `dteday`, the timestamp for that day (fine to set to noon or any other time)
- `weekday`, the day of the week
- `weather`, the most severe weather that day
- `season`, the season that day falls in
- `temp`, the average temperature
- `atemp`, the average atemp that day
- `windspeed`, the average windspeed that day
- `hum`, the average humidity that day
- `casual`, the **total** number of rentals by casual users
- `registered`, the **total** number of rentals by registered users
- `counts`, the **total** number of rentals

Name this dataframe `bikes_by_day` and use it for all of Question 2.

2.2 How does **season** affect the number of bike rentals for **casual riders** or **registered riders** per day? Use the variable `season` for this question. Comment on your observations.

2.3 What percentage of rentals are made by casual riders or registered riders for each day of the week? Comment on any patterns you see and give a possible explanation.

2.4 How is the **distribution of total number of bike rentals** different for sunny days vs cloudy days?

2.5 Visualize how the **total number of rides** per day varies with the **season**. Do you see any **outliers**? (We define an outlier as a value 1.5 times the IQR above the 75th percentile or 1.5 times the IQR below the 25th percentiles. This is the same rule used by pyplot's boxplot function). If you see any outliers, identify those dates and investigate if they are a chance occurence, an error in the data collection, or an important event.

HINT

- Use .copy() when creating the new dataframe, so you leave the original untouched. We will come back to it later.
- Use .groupby() to creat the new dataframe. You will have to make some choice on how to aggregate the variables.

Answers

2.1 Make a new dataframe with the following subset ...

In [10]:

```
df = bikes_df.copy()

df2 = df.groupby('dteday')
df2_mean = df2['weekday','season','temp','atemp','windspeed','hum'].agg(np.mean)
df2_max = df2['weather'].agg(np.max)
df2_count = df2['casual','registered'].agg(np.sum)

weekday = df2_mean['weekday']
weather = df2_max
season = df2_mean['season']
temp = df2_mean['temp']
atemp = df2_mean['atemp']
windspeed = df2_mean['windspeed']
hum = df2_mean['hum']
casual = df2_count['casual']
registered = df2_count['registered']
counts = casual + registered

rows = list(zip(df2_mean.index, weekday.values, weather.values, season.values, t
emp.values, atemp.values, \
          windspeed.values, hum.values, casual.values, registered.values, counts
.values))

bikes_by_day = pd.DataFrame(rows, columns=['dteday','weekday','weather','season'
,'temp','atemp', \
                            'windspeed','hum','casual','registered','counts'])

bikes_by_day.head()
```

Out[10]:

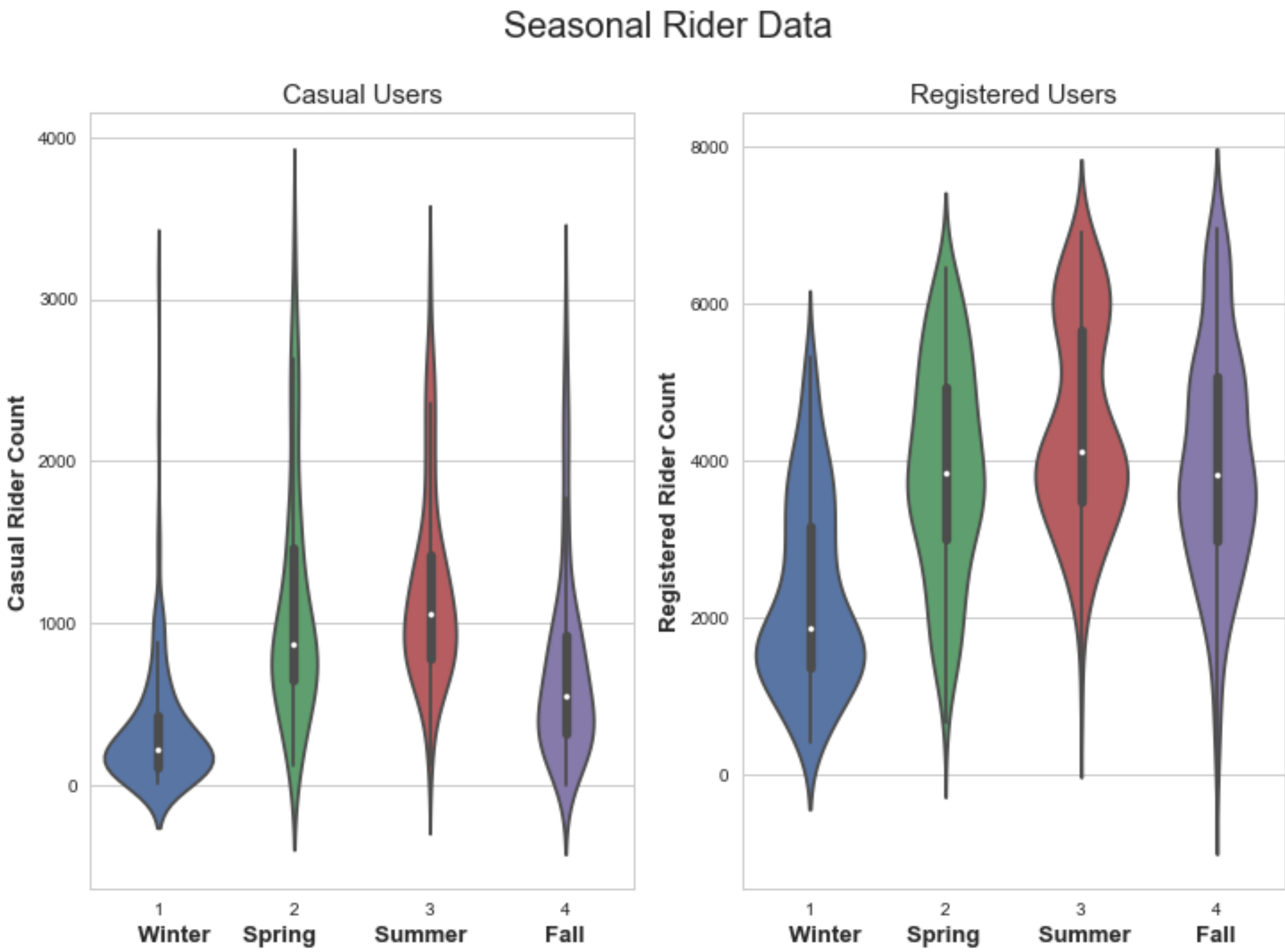
	dteday	weekday	weather	season	temp	atemp	windspeed	hum	casua
0	2011-01-01	6	3	1	0.344167	0.363625	0.160446	0.805833	331
1	2011-01-02	0	3	1	0.363478	0.353739	0.248539	0.696087	131
2	2011-01-03	1	1	1	0.196364	0.189405	0.248309	0.437273	120
3	2011-01-04	2	2	1	0.200000	0.212122	0.160296	0.590435	108
4	2011-01-05	3	1	1	0.226957	0.229270	0.186900	0.436957	82

2.2 How does season affect the number of bike

In [11]:

```
# 1 = winter, 2 = spring, 3 = summer, 4 = fall

fig, ax = plt.subplots(1, 2, figsize=(12,8))
a = sns.violinplot(x="season", y="casual", data=bikes_by_day, ax=ax[0])
b = sns.violinplot(x="season", y="registered", data=bikes_by_day, ax=ax[1])
ax[0].set_title('Casual Users', fontsize=15)
ax[1].set_title('Registered Users', fontsize=15)
fig.suptitle('Seasonal Rider Data', fontsize=20);
a.set_xlabel('Winter     Spring         Summer            Fall', fontsize=13, fo
ntweight='bold')
a.set_ylabel('Casual Rider Count', fontsize=13, fontweight='bold')
b.set_ylabel('Registered Rider Count', fontsize=13, fontweight='bold')
b.set_xlabel('Winter       Spring         Summer          Fall', fontsize=13, fo
ntweight='bold')
fig.show()

# also: https://stackoverflow.com/questions/49065837/customize-the-axis-label-in
-seaborn-jointplot
#resource used: https://stackoverflow.com/questions/43131274/how-do-i-plot-two-c
ountplot-graphs-side-by-side-in-seaborn
```

```
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



First observation I had was that the casual rider count is still lower for the casuals than registered riders. Winter has very low distribution for casual riders; makes sense because winter time, you'll see only the registered users use the service more and this is shown by the higher distribution in the registered figure. Casual users are more consious about the season, whereas registered riders aren't. Highest casual count is in the spring time, versus Fall for the registered riders.

2.3 What percentage of rentals are made by casual riders or registered riders ...

In [12]:

```
sun = bikes_by_day[(bikes_by_day['weekday'] == 0)]
sun1 = (sun['casual'].sum())/(sun['counts'].sum())*100
sun2 = 100 - sun1

mon = bikes_by_day[(bikes_by_day['weekday'] == 1)]
mon1 = (mon['casual'].sum())/(mon['counts'].sum())*100
mon2 = 100 - mon1

tues = bikes_by_day[(bikes_by_day['weekday'] == 2)]
tues1 = (tues['casual'].sum())/(tues['counts'].sum())*100
tues2 = 100 - tues1

wed = bikes_by_day[(bikes_by_day['weekday'] == 3)]
wed1 = (wed['casual'].sum())/(wed['counts'].sum())*100
wed2 = 100 - wed1

thurs = bikes_by_day[(bikes_by_day['weekday'] == 4)]
thurs1 = (thurs['casual'].sum())/(thurs['counts'].sum())*100
thurs2 = 100 - thurs1

fri = bikes_by_day[(bikes_by_day['weekday'] == 5)]
fri1 = (fri['casual'].sum())/(fri['counts'].sum())*100
fri2 = 100 - fri1

sat = bikes_by_day[(bikes_by_day['weekday'] == 6)]
sat1 = (fri['casual'].sum())/(sat['counts'].sum())*100
sat2 = 100 - sat1

day_of_week = ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday')
casual = (sun1, mon1, tues1, wed1, thurs1, fri1, sat1)
reg = (sun2, mon2, tues2, wed2, thurs2, fri2, sat2)

percent = pd.DataFrame(list(zip(day_of_week, casual, reg)), columns=['Day of Wee
k', 'casual', 'registered'])
percent['casual'] = percent['casual'].astype(str) + '%'
percent['registered'] = percent['registered'].astype(str) + '%'

percent
```

Out[12]:

	Day of Week	casual	registered
0	Sunday	31.64694939722134%	68.35305060277867%
1	Monday	15.539743975341548%	84.46025602465845%
2	Tuesday	12.330396560287694%	87.6696034397123%
3	Wednesday	12.116952190898175%	87.88304780910183%
4	Thursday	12.661852717889555%	87.33814728211044%
5	Friday	16.039279198015542%	83.96072080198445%
6	Saturday	16.374393845213653%	83.62560615478634%

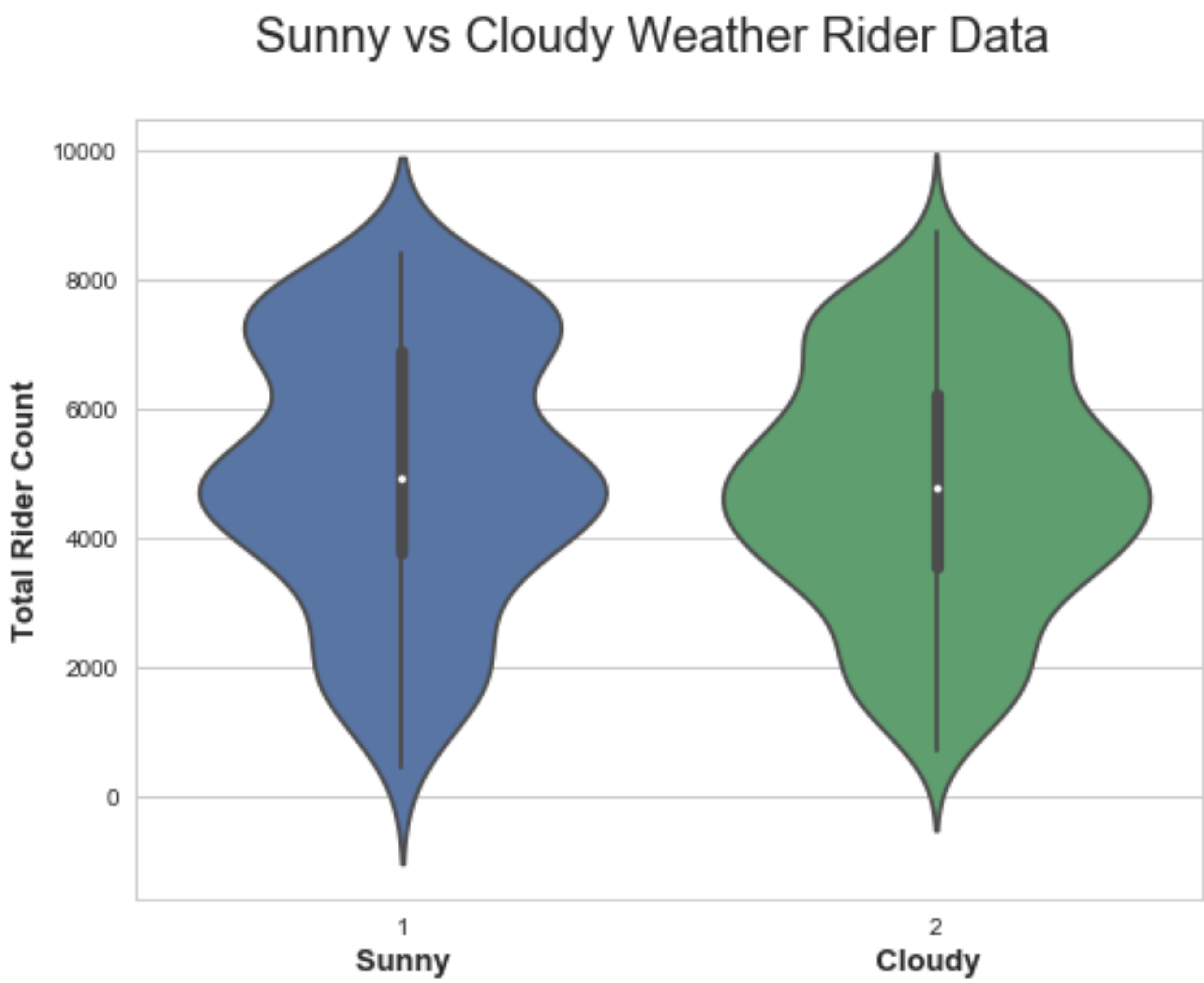
The first obvious pattern is that registered users tend to use the service signicantly more than casual users during the work week of monday through firday. Then on Saturday, there is a large amount of registered users using the service, but then that drops sharply on sunday. Sunday tens to be a day when casual users use it more (31%!). As the day of the week progresses for the casual user, the casual ridership percentage increases, whereas for registrted users, it is the opposite because it drops during the weekend, especially Sunday. One possible explanation is that casual users are trying to go out and explore with the bikes, whereas registered users already use the service, so they might be seeking out alternative transportation options and/or going longer distances, where bike transportation might not be good enough or pratical.

2.4 How is the distribution of total number of bike rentals different ...

In [13]:

```
# cloudy = 2, sunny = 1

sunny = bikes_by_day[(bikes_by_day['weather'] == 1)]

cloudy = bikes_by_day[(bikes_by_day['weather'] == 2)]

frames = [sunny, cloudy]

sunny_cloudy = pd.concat(frames)

fig, ax = plt.subplots(figsize=(8,6))
a = sns.violinplot(x="weather", y="counts", data=sunny_cloudy)
fig.suptitle('Sunny vs Cloudy Weather Rider Data', fontsize=20);
a.set_xlabel('Sunny                                                Cloudy', fon
tsize=13, fontweight='bold')
a.set_ylabel('Total Rider Count', fontsize=13, fontweight='bold')
fig.show()
```

```
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```



Looking at the violin plot above, it can be said that the distribution for the sunny days is different from the cloudy days. The distribtuion for the sunny days is "top heavy," which means that there are more days where riders took out the bikes during sunny days, versus the cloudy days, whhere the distrubition was lower. The means for both data sets is very similar, but the sunny total rider count mean is slightly higher. There were less cloudy days where a lot of people went out and used the bicycles.

2.5 Visualize how the total number of rides per day ...

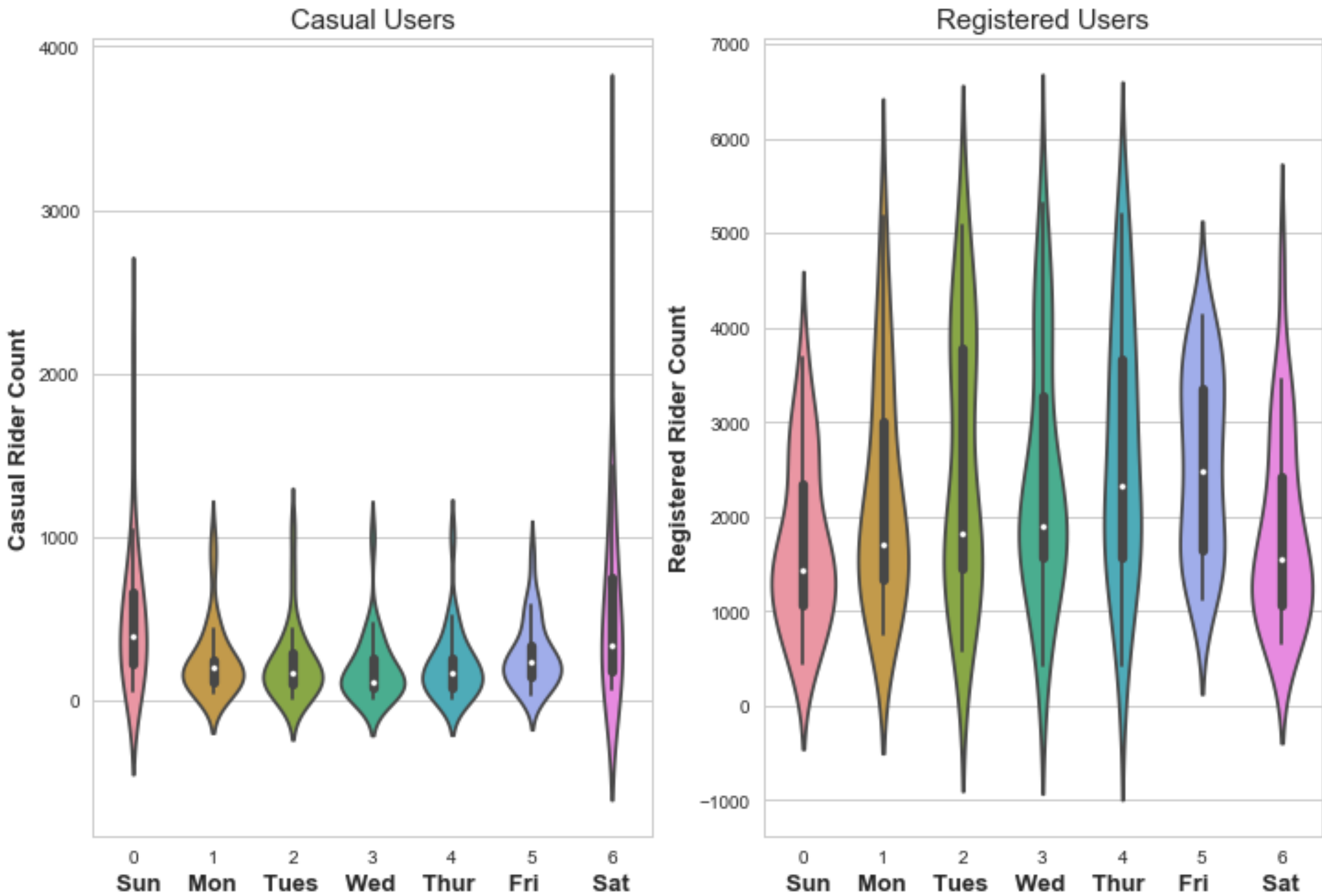
In [14]:

```
# 1 = winter, 2 = spring, 3 = summer, 4 = fall



season1 = bikes_by_day[(bikes_by_day['season'] == 1)]
season2 = bikes_by_day[(bikes_by_day['season'] == 2)]
season3 = bikes_by_day[(bikes_by_day['season'] == 3)]
season4 = bikes_by_day[(bikes_by_day['season'] == 4)]


fig, ax = plt.subplots(1, 2, figsize=(12,8))
a = sns.violinplot(x="weekday", y="casual", data=season1, ax=ax[0])
b = sns.violinplot(x="weekday", y="registered", data=season1, ax=ax[1])
ax[0].set_title('Casual Users', fontsize=15)
ax[1].set_title('Registered Users', fontsize=15)
fig.suptitle('Winter Seasonal Rider Data by day', fontsize=20);
a.set_xlabel('Sun     Mon     Tues     Wed     Thur     Fri         Sat', fontsize=13
, fontweight='bold')
a.set_ylabel('Casual Rider Count', fontsize=13, fontweight='bold')
b.set_ylabel('Registered Rider Count', fontsize=13, fontweight='bold')
b.set_xlabel('Sun     Mon     Tues     Wed     Thur     Fri         Sat', fontsize=13
, fontweight='bold')
fig.show()

fig, ax = plt.subplots(1, 2, figsize=(12,8))
a = sns.violinplot(x="weekday", y="casual", data=season2, ax=ax[0])
b = sns.violinplot(x="weekday", y="registered", data=season2, ax=ax[1])
ax[0].set_title('Casual Users', fontsize=15)
ax[1].set_title('Registered Users', fontsize=15)
fig.suptitle('Spring Seasonal Rider Data by day', fontsize=20);
a.set_xlabel('Sun     Mon     Tues     Wed     Thur     Fri         Sat', fontsize=13
, fontweight='bold')
a.set_ylabel('Casual Rider Count', fontsize=13, fontweight='bold')
b.set_ylabel('Registered Rider Count', fontsize=13, fontweight='bold')
b.set_xlabel('Sun     Mon     Tues     Wed     Thur     Fri         Sat', fontsize=13
, fontweight='bold')
fig.show()

fig, ax = plt.subplots(1, 2, figsize=(12,8))
a = sns.violinplot(x="weekday", y="casual", data=season3, ax=ax[0])
b = sns.violinplot(x="weekday", y="registered", data=season3, ax=ax[1])
ax[0].set_title('Casual Users', fontsize=15)
ax[1].set_title('Registered Users', fontsize=15)
```

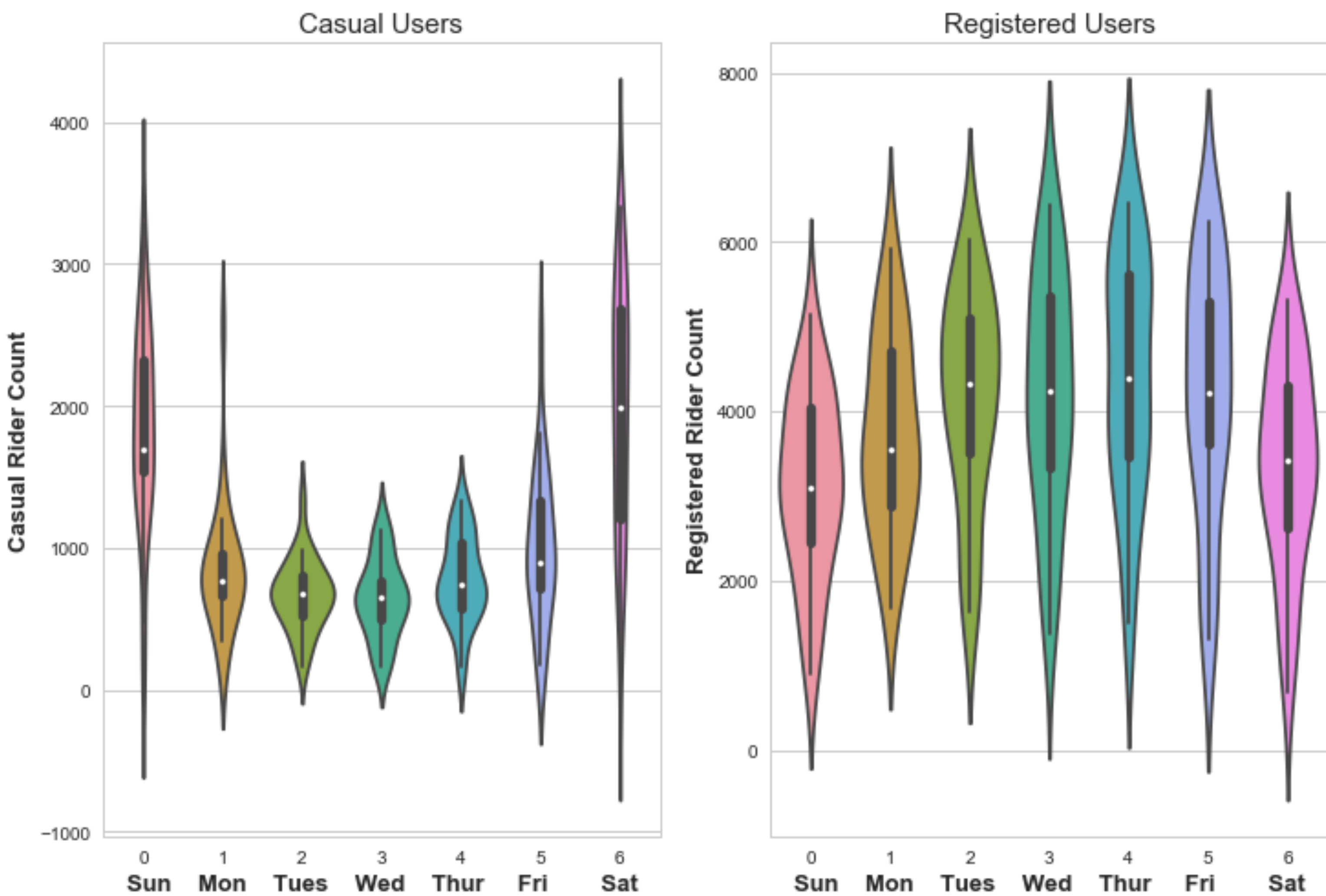
```
fig.suptitle('Summer Seasonal Rider Data by day', fontsize=20);

a.set_xlabel('Sun    Mon    Tues    Wed    Thur     Fri        Sat', fontsize=13
, fontweight='bold')
a.set_ylabel('Casual Rider Count', fontsize=13, fontweight='bold')
b.set_ylabel('Registered Rider Count', fontsize=13, fontweight='bold')
b.set_xlabel('Sun    Mon    Tues    Wed    Thur     Fri        Sat', fontsize=13
, fontweight='bold')
fig.show()

fig, ax = plt.subplots(1, 2, figsize=(12,8))
a = sns.violinplot(x="weekday", y="casual", data=season4, ax=ax[0])
b = sns.violinplot(x="weekday", y="registered", data=season4, ax=ax[1])
ax[0].set_title('Casual Users', fontsize=15)
ax[1].set_title('Registered Users', fontsize=15)
fig.suptitle('Fall Seasonal Rider Data by day', fontsize=20);
a.set_xlabel('Sun    Mon    Tues    Wed    Thur     Fri        Sat', fontsize=13
, fontweight='bold')
a.set_ylabel('Casual Rider Count', fontsize=13, fontweight='bold')
b.set_ylabel('Registered Rider Count', fontsize=13, fontweight='bold')
b.set_xlabel('Sun    Mon    Tues    Wed    Thur     Fri        Sat', fontsize=13
, fontweight='bold')
fig.show()
```

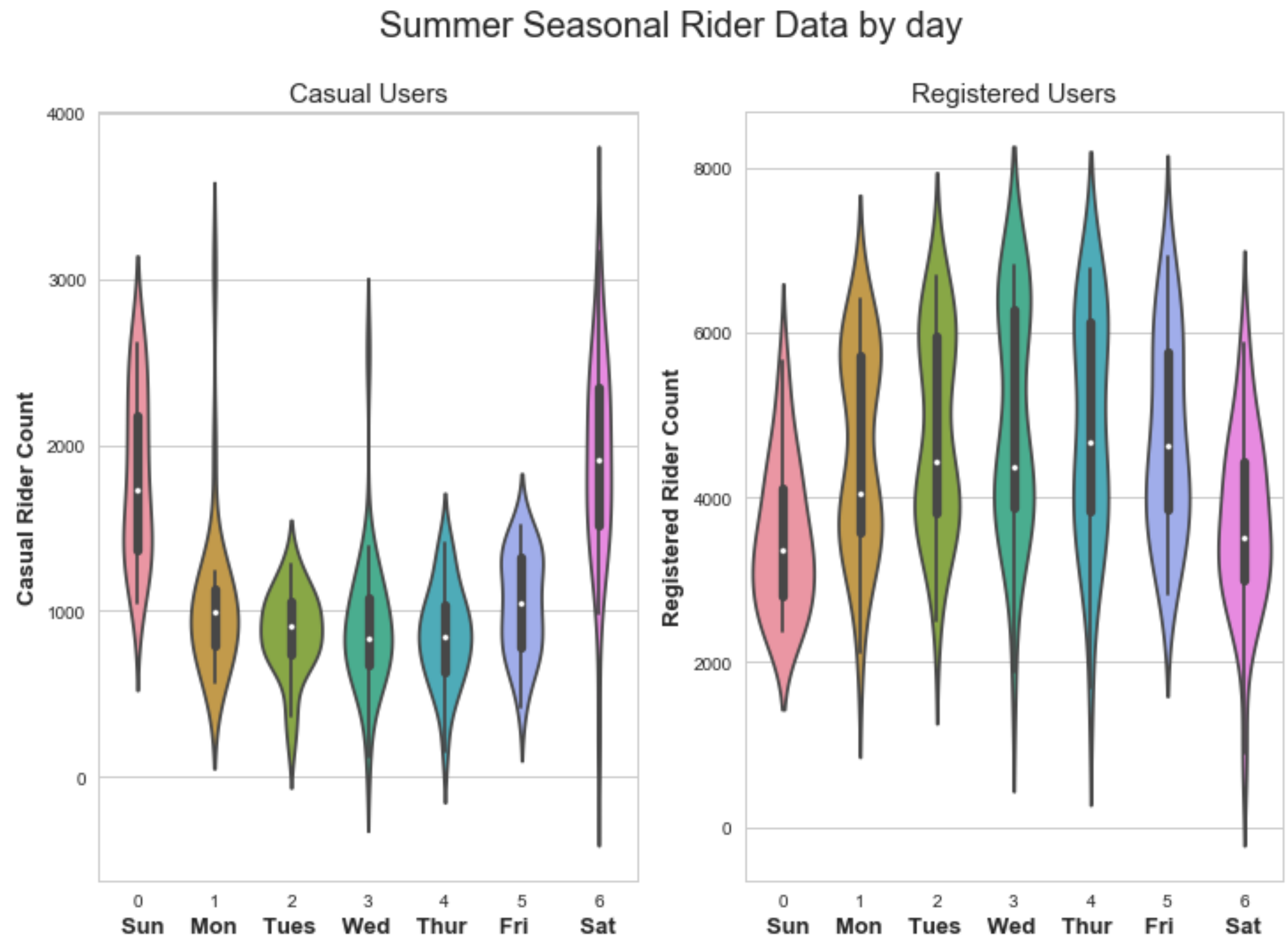
```
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```

Winter Seasonal Rider Data by day

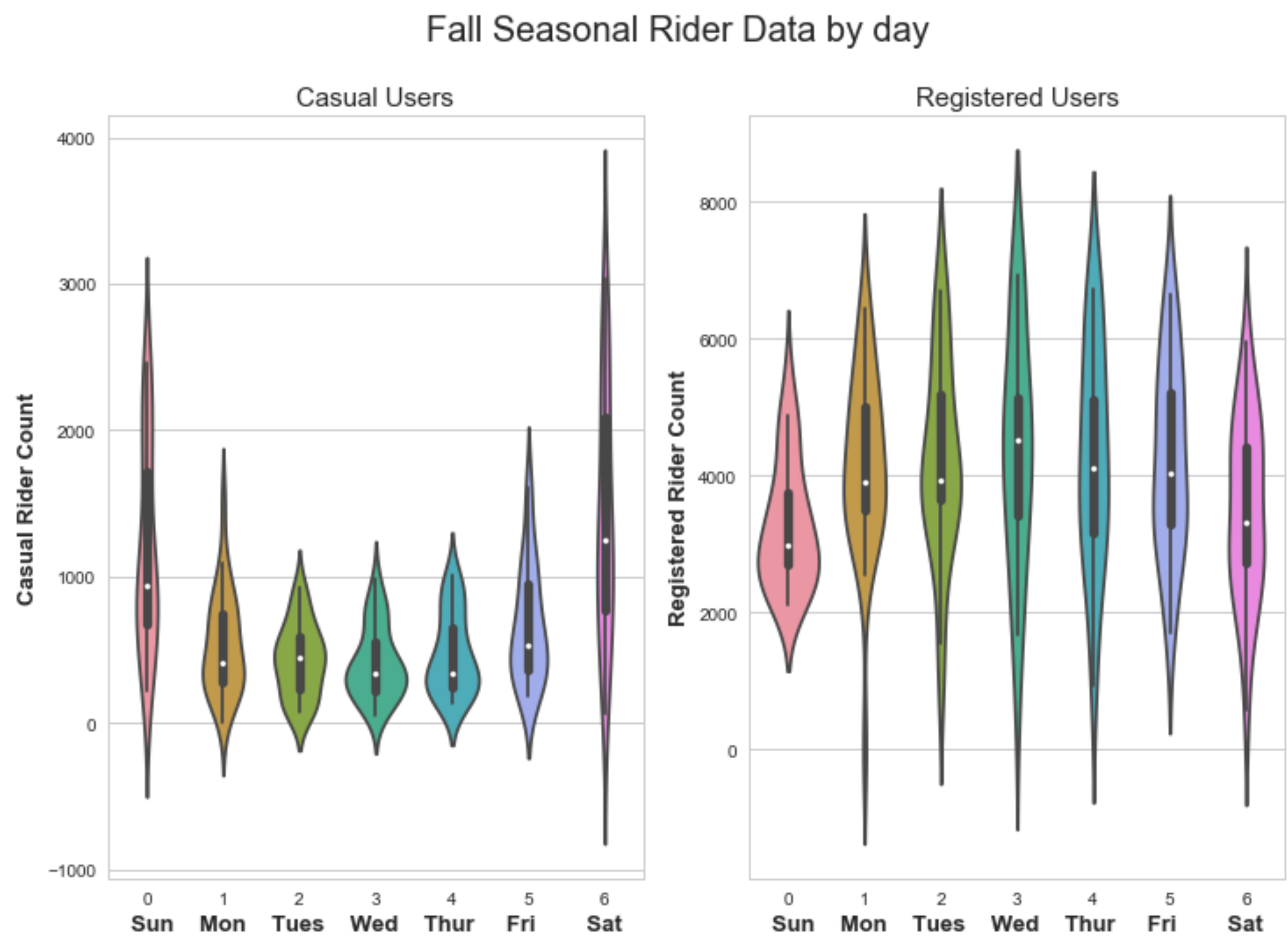


Spring Seasonal Rider Data by day





Summer Seasonal Rider Data by day
Casual Users
Registered Users
Casual Rider Count
Registered Rider Count
0 1 2 3 4 5 6
Sun Mon Tues Wed Thur Fri Sat



Fall Seasonal Rider Data by day
Casual Users
Registered Users
Casual Rider Count
Registered Rider Count
0 1 2 3 4 5 6
Sun Mon Tues Wed Thur Fri Sat

From the graphs, I see a few outliers. For example, there are some winter days (first graph) where casual rider count is almost as high as even the registered users. Same with Sunday. Looking at the summer graph, there is an anomaly with the monday and saturday where the monday one has a high point and then saturday where it is also very low. From the graphs, I really don't see any outliers with the summer days for both sets of data. For the fall, there are a few outliers, for example on sunday, the distribution graph is showing towards -1000. If I had to make a guess to why this happened, it could be possible holidays, such as christmas or new years for the winter set, or even an error in the data collection; for example, what does a "ride" actually mean? Some people might unhook the bike and then immediately put it back on the rack and that would count as one or the people running the service could be testing the mechanisms for the winter.

Question 3: Prepare the data for Regression

3.1 Visualize and describe inter-dependencies among the following variables: `weekday`, `season`, `month`,`weather`, `temp`, `atemp`, `hum`, `windspeed`, `casual`,`registered`, `counts`. Note and comment on any strongly related variables.

3.2 Convert the categorical attributes into multiple binary attributes using **one-hot encoding**.

3.3 Split the initial `bikes_df` dataset (with hourly data about rentals) into train and test sets. Do this in a 'stratified' fashion, ensuring that all months are equally represented in each set. Explain your choice for a splitting algorithm. We ask you to create your train and test sets, but for consistency and easy checking we ask that, for the rest of this problem set, you use the train and test set provided in the question below.

3.4 Read `data/BSS_train.csv` and `data/BSS_test.csv` into dataframes `BSS_train` and `BSS_test`, respectively. After checking your train and test datasets for accuracy, remove the `dteday` column from both train and test dataset. We do not need it, and its format cannot be used for analysis. Also, remove any predictors that would make predicting the `count` trivial.

3.5 Calculate the **Pearson correlation** coefficients between all the features. Visualize the matrix using a heatmap. Which predictors have a positive correlation with the number of bike rentals? For categorical attributes, you should use each binary predictor resulting from one-hot encoding to compute their correlations. Identify pairs of predictors with collinearity >0.7.

Hints:

- You may use the `np.corrcoef` function to compute the correlation matrix for a data set (do not forget to transpose the data matrix). You may use `plt.pcolor` function to visualize the correlation matrix.

Answers

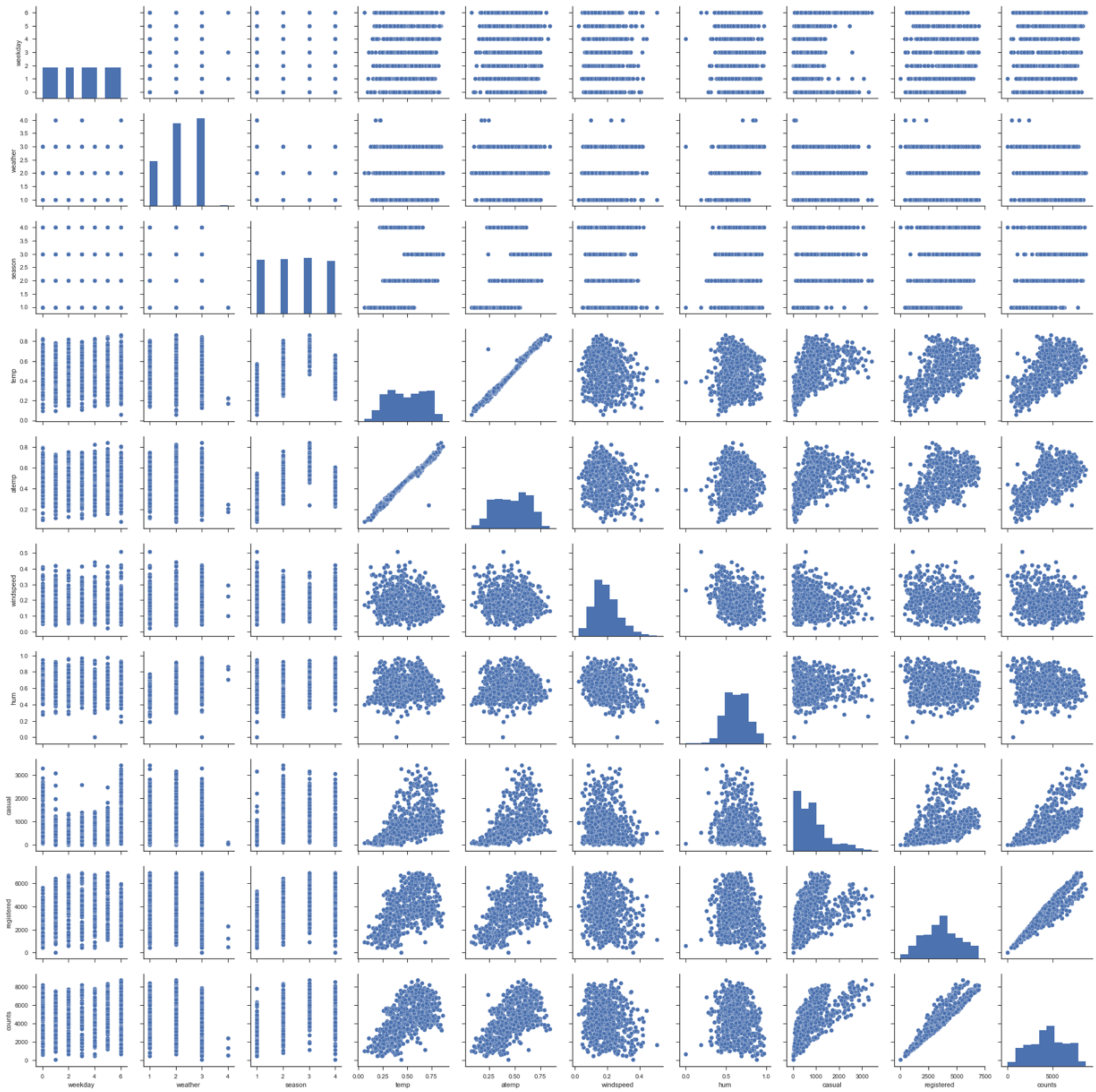
3.1 Visualize and describe inter-dependencies ...

In [15]:

```
sns.set(style="ticks")
df = bikes_by_day
sns.pairplot(df)
```

Out[15]:

<seaborn.axisgrid.PairGrid at 0x1c192d7240>



Strongly related variables

- temp and atemp: this is straight forward. They are directly related to each other
- registered/casual and counts: they are directly related to each other. We need the registered variable to do counts
- casual/registered and temp/atemp: as temp goes up, the number of rides goes up
- We don't see any linear correlation between the weekday/weather/season to casual/registered.

3.2 Convert the categorical attributes ....

In [16]:

```
#categorical attributes = weekday, weather, season

df_hot = bikes_by_day.copy()

one = pd.get_dummies(df_hot.weekday, prefix='weather').iloc[:, 1:]

two = pd.get_dummies(df_hot.weekday, prefix='weekday').iloc[:, 1:]

three = pd.get_dummies(df_hot.season, prefix='season').iloc[:, 1:]

# add them to data frame
df_hot = pd.concat([df_hot, one, two, three], axis = 1)

# resource used: https://www.youtube.com/watch?v=0s_1IsROgDc

df_hot.head()
```

Out[16]:

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	casua
0	2011-01-01	6	3	1	0.344167	0.363625	0.160446	0.805833	331
1	2011-01-02	0	3	1	0.363478	0.353739	0.248539	0.696087	131
2	2011-01-03	1	1	1	0.196364	0.189405	0.248309	0.437273	120
3	2011-01-04	2	2	1	0.200000	0.212122	0.160296	0.590435	108
4	2011-01-05	3	1	1	0.226957	0.229270	0.186900	0.436957	82

5 rows × 26 columns

pd head above shows one-hot encoding

3.3 ....

In [17]:

```
traindf, testdf = train_test_split(bikes_df,
                                   stratify = pd.to_datetime(bikes_df.index).mon
th)

print(traindf.shape, testdf.shape)

# resource used: https://stackoverflow.com/questions/29438265/stratified-train-t
est-split-in-scikit-learn
# and http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.t
rain_test_split.html
```

```
(13034, 15) (4345, 15)
```

The splitting algoritm used was sklearn's train_test_split. The test size was left to the default setting, which is 0.25. Random state was also left to be default. I did stratify the dats set. I did this so that all months would equally represented in each set; which in this case, I did it with the pd.to_datetime command. I decided to leave the default settings because according to the documentation, it will adapt the random state and train size with the sample set. I printed out the testdf to check if the stratifcation occured, and it did. We do this to balance the variables, so that one month is not represented more in the test set than the other.

3.4 Read `data/BSS_train.csv` and `data/BSS_test.csv` into ...

In [18]:

```
BBS_train = pd.read_csv("data/BSS_train.csv", index_col=0)
BBS_train = BBS_train.drop('dteday', axis=1)
BBS_train = BBS_train.drop('casual', axis=1)
BBS_train = BBS_train.drop('registered', axis=1)

BBS_test = pd.read_csv("data/BSS_test.csv", index_col=0)
BBS_test = BBS_test.drop('dteday', axis=1)
BBS_test = BBS_test.drop('casual', axis=1)
BBS_test = BBS_test.drop('registered', axis=1)

BBS_train.head()
```

Out[18]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed	counts	spring	...
0	0	0	0	0	0.24	0.2879	0.81	0.0	16	0	...
1	1	0	0	0	0.22	0.2727	0.80	0.0	40	0	...
2	2	0	0	0	0.22	0.2727	0.80	0.0	32	0	...
3	3	0	0	0	0.24	0.2879	0.75	0.0	13	0	...
4	4	0	0	0	0.24	0.2879	0.75	0.0	1	0	...

5 rows × 32 columns

In [19]:

```
BBS_test.head()
```

Out[19]:

	hour	holiday	year	workingday	temp	atemp	hum	windspeed	counts	spring	...
6	6	0	0	0	0.22	0.2727	0.80	0.0000	2	0	...
9	9	0	0	0	0.32	0.3485	0.76	0.0000	14	0	...
20	20	0	0	0	0.40	0.4091	0.87	0.2537	36	0	...
33	10	0	0	0	0.36	0.3485	0.81	0.2239	53	0	...
35	12	0	0	0	0.36	0.3333	0.66	0.2985	93	0	...

5 rows × 32 columns

Displaying the head of the pd frames shows that the columns needed to be dropped were dropped out.

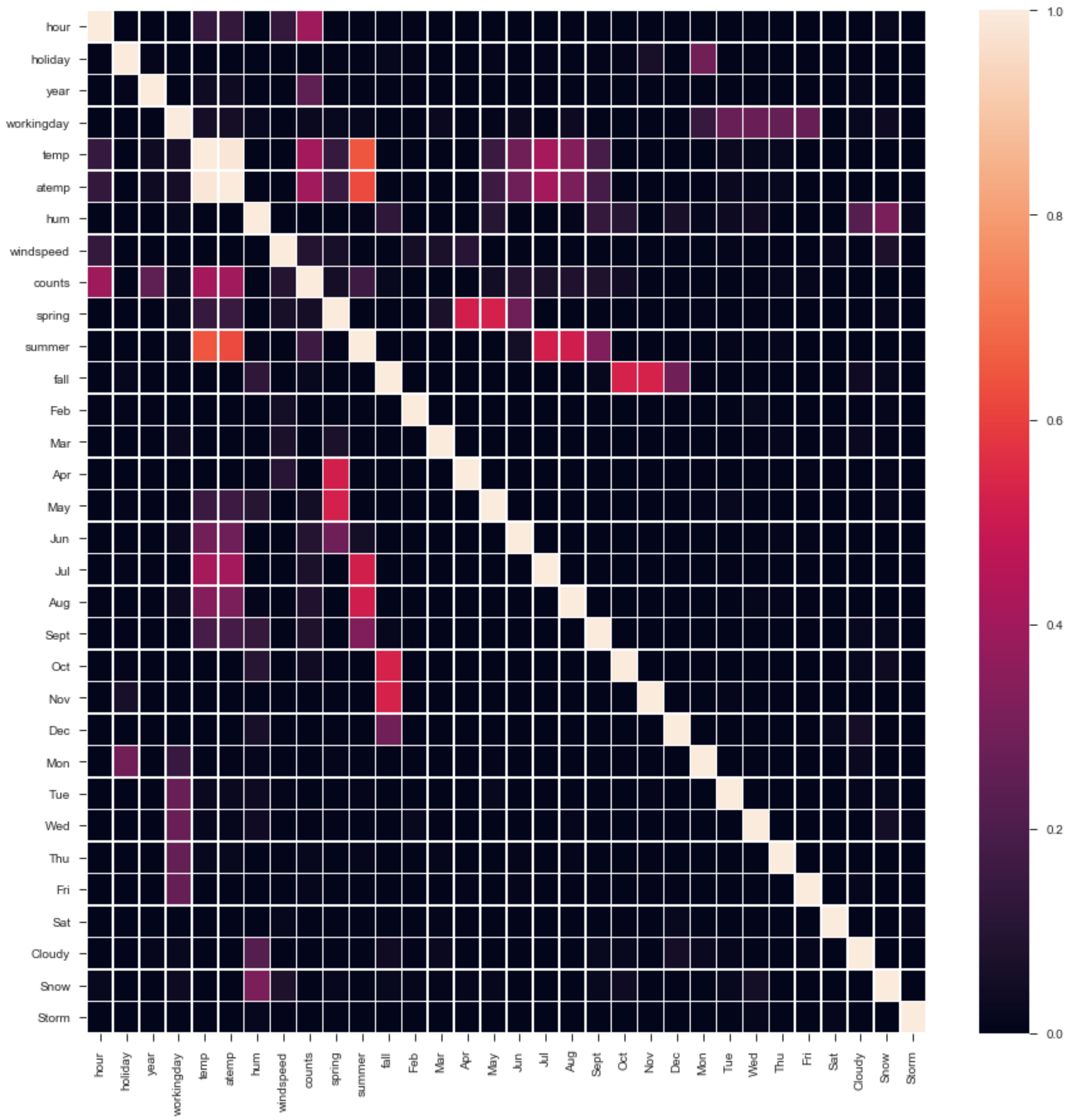
3.5 Calculate the Pearson correlation ....

In [20]:

```
pearson = BBS_train.corr(method='pearson')

fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(pearson, vmin=0, vmax=1, linewidths=.5)



# resource: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFr
ame.corr.html
# also used: https://seaborn.pydata.org/generated/seaborn.heatmap.html
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c31dd0c18>
```



In [21]:

```
print(pearson)
```

```
                hour   holiday      year  workingday      temp      a
temp  \
hour        1.000000  0.005028 -0.010900    0.002024  0.140745  0.13
6311
holiday     0.005028  1.000000  0.011641   -0.253523 -0.026372 -0.03
0072
year       -0.010900  0.011641  1.000000   -0.000840  0.038813  0.03
7635
workingday  0.002024 -0.253523 -0.000840    1.000000  0.056547  0.05
5666
temp        0.140745 -0.026372  0.038813    0.056547  1.000000  0.98
7408
atemp       0.136311 -0.030072  0.037635    0.055666  0.987408  1.00
```

```
0000
hum        -0.274146 -0.010085 -0.086934    0.020164 -0.071756 -0.05
3984
windspeed   0.139770 -0.000291 -0.008300   -0.009580 -0.018421 -0.05
8252
counts      0.394167 -0.028252  0.243886    0.029534  0.406155  0.40
1119
spring     -0.003675 -0.026538 -0.003334    0.022044  0.142863  0.15
1023
summer      0.003857 -0.025676  0.002648    0.018794  0.645391  0.62
2339
fall       -0.006450  0.016316 -0.013063   -0.016104 -0.220062 -0.20
0820
Feb         0.005606  0.010534  0.007331    0.001543 -0.297576 -0.29
6865
Mar        -0.003834 -0.052837 -0.000196    0.026703 -0.166973 -0.16
3844
Apr         0.003251  0.003616 -0.002221   -0.006576 -0.042255 -0.03
2615
May        -0.003347  0.007875 -0.006394    0.011599  0.154599  0.15
9369
Jun         0.001247 -0.052197 -0.001702    0.030723  0.292131  0.28
4454
Jul        -0.001340  0.010926  0.006461   -0.015422  0.412723  0.40
8329
Aug         0.004635 -0.052886 -0.003298    0.034786  0.335480  0.31
1650
Sept       -0.004525  0.005165 -0.000132   -0.009376  0.186166  0.18
0722
Oct        -0.002319  0.010816 -0.016538   -0.012953 -0.015389 -0.00
3395
Nov         0.001024  0.064028  0.000390   -0.011617 -0.200615 -0.19
0762
Dec        -0.008559  0.003484 -0.000215   -0.019465 -0.276243 -0.26
7651
Mon         0.002398  0.288895  0.003327    0.147598 -0.000895  0.00
4153
Tue         0.002814 -0.045881 -0.003703    0.268793  0.025038  0.02
7500
Wed         0.000877 -0.046017  0.001554    0.272720  0.016462  0.01
4942
Thu        -0.000289 -0.023697  0.003746    0.260472  0.021504  0.02
1189
Fri        -0.000661 -0.027916 -0.000422    0.263344  0.000682 -0.00
7918
Sat        -0.007787 -0.071661 -0.010093   -0.602438 -0.036657 -0.03
7245
Cloudy     -0.050553  0.007805  0.013921    0.022523 -0.071525 -0.06
8782
Snow        0.020257 -0.021242 -0.038232    0.034080 -0.062334 -0.06
8844
Storm      -0.005253 -0.002083 -0.000110   -0.004647 -0.019716 -0.02
1537
```

```
                 hum  windspeed    counts    spring    ...
Dec  \
hour       -0.274146   0.139770  0.394167 -0.003675    ...    -0.008
559
holiday    -0.010085  -0.000291 -0.028252 -0.026538    ...     0.003
484
year       -0.086934  -0.008300  0.243886 -0.003334    ...    -0.000
215
workingday  0.020164  -0.009580  0.029534  0.022044    ...    -0.019
465
temp       -0.071756  -0.018421  0.406155  0.142863    ...    -0.276
243
atemp      -0.053984  -0.058252  0.401119  0.151023    ...    -0.267
651
hum         1.000000  -0.286629 -0.328232  0.002175    ...     0.063
829
windspeed  -0.286629   1.000000  0.093981  0.063466    ...    -0.033
493
counts     -0.328232   0.093981  1.000000  0.058418    ...    -0.080
273
spring      0.002175   0.063466  0.058418  1.000000    ...    -0.177
812
summer      0.010583  -0.076725  0.159319 -0.344110    ...    -0.180
484
fall        0.122103  -0.097907  0.022531 -0.331343    ...     0.292
842
Feb        -0.086556   0.054643 -0.122671 -0.168382    ...    -0.088
315
Mar        -0.057093   0.071739 -0.056147  0.071130    ...    -0.092
917
Apr        -0.067723   0.109152 -0.003708  0.515742    ...    -0.091
705
May         0.105133  -0.022095  0.050471  0.525460    ...    -0.093
433
Jun        -0.084841  -0.007841  0.094448  0.282217    ...    -0.091
792
Jul        -0.049786  -0.057224  0.073333 -0.178139    ...    -0.093
433
Aug         0.010451  -0.034667  0.085847 -0.177319    ...    -0.093
003
Sept        0.138100  -0.059577  0.080225 -0.174845    ...    -0.091
705
Oct         0.105310  -0.054899  0.046657 -0.175755    ...    -0.092
182
Nov        -0.008004  -0.012009 -0.018979 -0.174845    ...    -0.091
705
Dec         0.063829  -0.033493 -0.080273 -0.177812    ...     1.000
000
Mon         0.013808  -0.002639 -0.009796  0.009869    ...    -0.003
846
Tue         0.032100   0.008295 -0.004783  0.000596    ...    -0.012
107
```

```
Wed          0.042039  -0.009246  0.006535  0.007562    ...    -0.011
563
Thu         -0.042152   0.005465  0.018731 -0.003677    ...    -0.001
501
Fri         -0.024094  -0.014696  0.015080  0.002232    ...     0.004
747
Sat         -0.019439   0.018772  0.000878 -0.006187    ...     0.022
950
Cloudy       0.221191  -0.052600 -0.050171 -0.007358    ...     0.055
270
Snow         0.309075   0.077704 -0.130511  0.018801    ...     0.006
285
Storm        0.016521   0.006998 -0.010548 -0.006984    ...    -0.003
663

                  Mon        Tue        Wed        Thu        Fri        S
at  \
hour         0.002398  0.002814  0.000877 -0.000289 -0.000661 -0.0077
87
holiday      0.288895 -0.045881 -0.046017 -0.023697 -0.027916 -0.0716
61
year         0.003327 -0.003703  0.001554  0.003746 -0.000422 -0.0100
93
workingday   0.147598  0.268793  0.272720  0.260472  0.263344 -0.6024
38
temp        -0.000895  0.025038  0.016462  0.021504  0.000682 -0.0366
57
atemp        0.004153  0.027500  0.014942  0.021189 -0.007918 -0.0372
45
hum          0.013808  0.032100  0.042039 -0.042152 -0.024094 -0.0194
39
windspeed   -0.002639  0.008295 -0.009246  0.005465 -0.014696  0.0187
72
counts      -0.009796 -0.004783  0.006535  0.018731  0.015080  0.0008
78
spring       0.009869  0.000596  0.007562 -0.003677  0.002232 -0.0061
87
summer      -0.011030  0.008069  0.001234  0.011869  0.002578 -0.0073
04
fall        -0.004069  0.001465 -0.007450  0.001937 -0.005451  0.0038
56
Feb          0.002116 -0.004972  0.019576 -0.000863 -0.008879 -0.0069
79
Mar         -0.015639  0.003531  0.004663  0.014924  0.002643  0.0084
91
Apr          0.006661 -0.007717 -0.008599 -0.005223  0.007892  0.0049
36
May          0.009041  0.021465  0.001911 -0.002654 -0.010405 -0.0146
95
Jun         -0.006302 -0.003426  0.003796  0.011072  0.010662 -0.0034
23
Jul          0.001653  0.007434 -0.008316 -0.014480 -0.001570  0.0013
45
```

```
Aug         -0.002486 -0.004094  0.012513  0.009510  0.005388 -0.0136
87
Sept        -0.009097 -0.002467 -0.007115  0.002284  0.006396 -0.0017
29
Oct          0.006220 -0.000628 -0.000135 -0.004116 -0.013410  0.0066
84
Nov         -0.006846  0.008034  0.003272  0.004536  0.006396 -0.0054
31
Dec         -0.003846 -0.012107 -0.011563 -0.001501  0.004747  0.0229
50
Mon          1.000000 -0.164210 -0.166736 -0.164015 -0.164892 -0.1671
71
Tue         -0.164210  1.000000 -0.166835 -0.164113 -0.164990 -0.1672
70
Wed         -0.166736 -0.166835  1.000000 -0.166637 -0.167527 -0.1698
43
Thu         -0.164015 -0.164113 -0.166637  1.000000 -0.164794 -0.1670
72
Fri         -0.164892 -0.164990 -0.167527 -0.164794  1.000000 -0.1679
65
Sat         -0.167171 -0.167270 -0.169843 -0.167072 -0.167965  1.0000
00
Cloudy       0.029514  0.015133 -0.016018 -0.006510  0.011969 -0.0035
98
Snow        -0.019048  0.019807  0.051982 -0.009931 -0.008018 -0.0093
42
Storm       -0.004859 -0.004862  0.012103 -0.004856 -0.004882  0.0120
59

               Cloudy      Snow     Storm
hour        -0.050553  0.020257 -0.005253
holiday      0.007805 -0.021242 -0.002083
year         0.013921 -0.038232 -0.000110
workingday   0.022523  0.034080 -0.004647
temp        -0.071525 -0.062334 -0.019716
atemp       -0.068782 -0.068844 -0.021537
hum          0.221191  0.309075  0.016521
windspeed   -0.052600  0.077704  0.006998
counts      -0.050171 -0.130511 -0.010548
spring      -0.007358  0.018801 -0.006984
summer      -0.067390 -0.045399 -0.007089
fall         0.041356  0.019541 -0.006826
Feb          0.004064  0.014755 -0.003469
Mar          0.026649  0.008799 -0.003650
Apr          0.002123  0.012040 -0.003602
May         -0.003425  0.016225 -0.003670
Jun         -0.047537 -0.035450 -0.003605
Jul         -0.060840 -0.044396 -0.003670
Aug         -0.042487 -0.026939 -0.003653
Sept         0.021759  0.022460 -0.003602
Oct          0.020561  0.037537 -0.003621
Nov         -0.005017 -0.005010 -0.003602
Dec          0.055270  0.006285 -0.003663
```

```
Mon          0.029514 -0.019048 -0.004859
Tue          0.015133  0.019807 -0.004862
Wed         -0.016018  0.051982  0.012103
Thu         -0.006510 -0.009931 -0.004856
Fri          0.011969 -0.008018 -0.004882
Sat         -0.003598 -0.009342  0.012059
Cloudy       1.000000 -0.178340 -0.007117
Snow        -0.178340  1.000000 -0.003605
Storm       -0.007117 -0.003605  1.000000

[32 rows x 32 columns]
```

From looking at the heatmap, with the number of bike rentals, the predictors that have a positive correlation are: hour, temp, atemp, and year.

The pairs that have a colinearity of ~ > 0.7 are:

- summer vs temp/atemp
- april/may vs counts
- jul/aug vs summer
- oct/nov vs fall

Question 4: Multiple Linear Regression

4.1 Use statsmodels to fit a multiple linear regression model to the training set using all the predictors (no interactions or polynomial terms), and report its R^2 score on the train and test sets.

4.2 Find out which of estimated coefficients are statistically significant at a significance level of 5% (p-value < 0.05). Comment on the results.

4.3 Make a plot of residuals of the fitted model $e = y - \hat{y}$ as a function of the predicted value $\hat{y}$. Note that this is slightly different from the residual plot for simple linear regression. Draw a horizontal line denoting the zero residual value on the Y-axis. Does the plot reveal a non-linear relationship between the predictors and response? What does the plot convey about the variance of the error terms?

Answers

4.1 Use statsmodels to fit a ...

In [22]:

```
X = BBS_train[['hour', 'holiday', 'year', 'workingday', 'temp', 'atemp', 'hum','
spring', 'summer', 'fall',
              'Feb', 'Mar', 'Apr', 'May','Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'No
v', 'Dec', 'Mon', 'Tue', 'Wed',
              'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm']]

y = BBS_train['counts']

X = sm.add_constant(X)
model = sm.OLS(y, X).fit()



test_x = BBS_test[['hour', 'holiday', 'year', 'workingday', 'temp', 'atemp', 'hu
m','spring', 'summer', 'fall',
              'Feb', 'Mar', 'Apr', 'May','Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'No
v', 'Dec', 'Mon', 'Tue', 'Wed',
              'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm']]
test_y = BBS_test['counts']

r2_test = r2_score(test_y, model.predict(sm.add_constant(test_x)))

print('R2 for train set:', model.rsquared)
print('R2 for test model:', r2_test)

# resource: https://blog.datarobot.com/multiple-regression-using-statsmodels
```

```
R2 for train set: 0.40635120125670743
R2 for test model: 0.40595089350757974
```

4.2 Find out which of estimated coefficients ...

In [25]:

```
# report coefficients that have p value < 0.05

model.summary()
```

Out[25]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.406
Model:	OLS	**Adj. R-squared:**	0.405
Method:	Least Squares	**F-statistic:**	327.4
Date:	Wed, 18 Jul 2018	**Prob (F-statistic):**	0.00
Time:	22:28:12	**Log-Likelihood:**	-88308.

No. Observations:	13903	**AIC:**	1.767e+05
Df Residuals:	13873	**BIC:**	1.769e+05
Df Model:	29		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-14.0773	7.968	-1.767	0.077	-29.695	1.541
hour	7.2341	0.184	39.229	0.000	6.873	7.596
holiday	-18.4174	6.596	-2.792	0.005	-31.346	-5.488
year	76.1531	2.378	32.022	0.000	71.492	80.815
workingday	11.3008	2.751	4.108	0.000	5.908	16.693
temp	356.4330	42.757	8.336	0.000	272.624	440.242
atemp	51.1617	44.832	1.141	0.254	-36.715	139.039
hum	-209.4844	7.566	-27.689	0.000	-224.314	-194.655
spring	43.1702	7.418	5.820	0.000	28.630	57.711
summer	29.2452	8.773	3.333	0.001	12.049	46.442
fall	67.6397	7.479	9.045	0.000	52.981	82.299
Feb	-7.6642	5.966	-1.285	0.199	-19.359	4.031
Mar	-11.6669	6.666	-1.750	0.080	-24.732	1.399
Apr	-41.2879	9.878	-4.180	0.000	-60.651	-21.925
May	-34.1679	10.536	-3.243	0.001	-54.819	-13.516
Jun	-67.2084	10.697	-6.283	0.000	-88.175	-46.242
Jul	-95.1258	12.062	-7.886	0.000	-118.770	-71.482
Aug	-60.6874	11.813	-5.138	0.000	-83.842	-37.533
Sept	-16.8872	10.568	-1.598	0.110	-37.603	3.828
Oct	-16.0654	9.866	-1.628	0.103	-35.405	3.274
Nov	-25.3482	9.525	-2.661	0.008	-44.018	-6.678
Dec	-9.9804	7.614	-1.311	0.190	-24.904	4.943
Mon	-2.6003	2.978	-0.873	0.383	-8.438	3.238
Tue	-6.0797	3.208	-1.895	0.058	-12.367	0.208
Wed	2.1858	3.183	0.687	0.492	-4.053	8.425
Thu	-3.2486	3.185	-1.020	0.308	-9.492	2.995

Fri	2.6262	3.184	0.825	0.409	-3.614	8.866
Sat	14.9864	4.382	3.420	0.001	6.397	23.576
Cloudy	7.0039	2.898	2.417	0.016	1.323	12.685
Snow	-26.7320	4.762	-5.614	0.000	-36.066	-17.398
Storm	44.5767	98.383	0.453	0.650	-148.268	237.421

Omnibus:	2832.667	**Durbin-Watson:**	0.756
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5661.391
Skew:	1.224	**Prob(JB):**	0.00
Kurtosis:	4.943	**Cond. No.**	1.17e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.87e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [24]:

```
model.pvalues < 0.05
```

Out[24]:

```
const          False
hour            True
holiday         True
year            True
workingday      True
temp            True
atemp          False
hum             True
spring          True
summer          True
fall            True
Feb            False
Mar            False
Apr             True
May             True
Jun             True
Jul             True
Aug             True
Sept           False
Oct            False
Nov             True
Dec            False
Mon            False
Tue            False
Wed            False
Thu            False
Fri            False
Sat             True
Cloudy          True
Snow            True
Storm          False
dtype: bool
```

20 different estimated coefficients are statistically significant at a significance level of 5%. These are the ones listed above (that say **True**). Specifically, the estimated coeff for the hour, holiday, year, workingday, temp, hum, spring, summer, fall, apr, may, jun, july, aug, nov, sat, cloudy, and snow variables are statisically significant. This means that the likelihood of a relationship between these variables and the total ride count is caused by something other than chance.

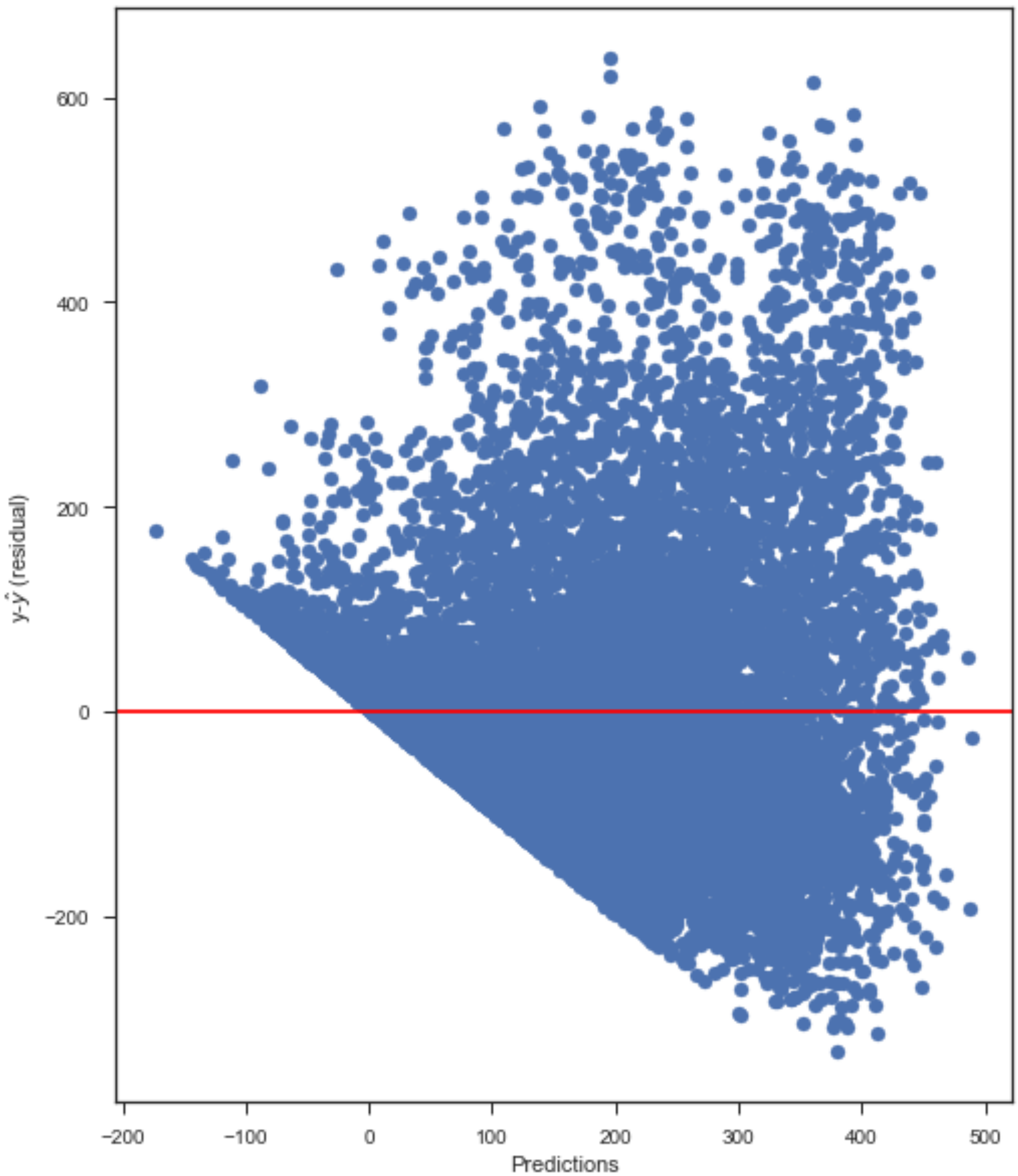
resource: https://measuringu.com/statistically-significant/ (https://measuringu.com/statistically-significant/)

4.3 Make a plot of residuals of the fitted ...

In [26]:

```
# e- y_true - y_pred

y_true = y
y_pred = model.predict()

residual = (y_true - y_pred)

fig, ax = plt.subplots(1,1, figsize=(8,10))
ax.scatter(y_pred, residual, label="training data")
plt.axhline(0, color='red')
ax.set_xlabel('Predictions')
ax.set_ylabel('y-$\hat{y}$ (residual)')
fig.suptitle('Scatter Plot of Residual from Multiple Linear Regression', fontsiz
e=16);
```

Scatter Plot of Residual from Multiple Linear Regression



The residual plot sort of shows a negative linear relationship between the predictors and response because it is decreasing as the predictions increase. On the topic of the variance of error terms, the plot is showing a tendency to predict negatively as the predictions increase. It is more accuarate as the predictions get smaller.

Question 5: Subset Selection

5.1 Implement forward step-wise selection to select a minimal subset of predictors that are related to the response variable:

We require that you implement the method **from scratch**. You may use the Bayesian Information Criterion (BIC) to choose the subset size in each method.

5.2 Do these methods eliminate one or more of the colinear predictors (if any) identified in Question 3.5? If so, which ones. Briefly explain (3 or fewer sentences) why you think this may be the case.

5.3 Fit the linear regression model using the identified subset of predictors to the training set. How do the test R^2 scores for the fitted models compare with the model fitted in Question 4 using all predictors?

Answers

5.1 Implement forward step-wise ....

In [27]:

```
# added one by one to subset x to find the best combination. only add if pvalue
< 0.05

subset_x = BBS_train[['hour', 'holiday', 'year', 'workingday', 'temp', 'hum', 's
pring', 'summer', 'fall',
                     'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Nov', 'Sat', 'Cloudy',
'Snow']]

# did not add 12 of them, if R2 decreased after adding, dropped from list

subset_y = BBS_train['counts']

# resource: http://www.biostat.jhsph.edu/~iruczins/teaching/jf/ch10.pdf
```

Implemented it one by one. I went through and tried to add more predictors into the subset_x and then calcaulte it. This was using forward step-wise selection one by one, and I came up with this combination of predictors.

5.2 Do these methods eliminate ...

From 3.5, some of the predictors that were identified were: winter, sunday, fall, and casual.

This model already had the casual predictor taken out, so that wasn't an issue. The forward step-wise predictor correctly identified the other predictors, such as winter and sunday, which had a few outliers from 3.5. Taking those out and my R2 squared score actually went up. The only difference between the two models that I saw was that 3.5 said I should remove the fall predictor, but in this model, when I tried that, it lowered the r2 sqaured tremendously. I think this is the case because 3.5 was a very rudamentary way of visually identifiing outliers versus computationally doing it in 5.2.

5.3 In each case, fit linear regression ...

In [28]:

```
X = sm.add_constant(subset_x)
sub = sm.OLS(subset_y, X).fit()

sub.summary()
```

Out[28]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.406
Model:	OLS	**Adj. R-squared:**	0.405
Method:	Least Squares	**F-statistic:**	526.9
Date:	Wed, 18 Jul 2018	**Prob (F-statistic):**	0.00
Time:	22:42:14	**Log-Likelihood:**	-88314.
No. Observations:	13903	**AIC:**	1.767e+05
Df Residuals:	13884	**BIC:**	1.768e+05
Df Model:	18		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-15.9596	7.302	-2.186	0.029	-30.272	-1.647
hour	7.2706	0.183	39.681	0.000	6.911	7.630
holiday	-20.4296	7.596	-2.689	0.007	-35.319	-5.540
year	76.3896	2.375	32.163	0.000	71.734	81.045
workingday	9.9561	3.422	2.909	0.004	3.248	16.664

temp	391.6092	11.640	33.642	0.000	368.792	414.426
hum	-209.6542	7.390	-28.370	0.000	-224.140	-195.169
spring	37.9566	6.385	5.945	0.000	25.442	50.471
summer	22.3995	6.216	3.603	0.000	10.215	34.584
fall	61.6811	4.295	14.361	0.000	53.262	70.100
Apr	-27.7004	6.969	-3.975	0.000	-41.361	-14.040
May	-19.6031	7.128	-2.750	0.006	-33.574	-5.632
Jun	-51.3459	6.397	-8.026	0.000	-63.886	-38.806
Jul	-77.4199	6.258	-12.370	0.000	-89.687	-65.152
Aug	-43.9252	6.126	-7.171	0.000	-55.933	-31.918
Nov	-12.0988	5.174	-2.338	0.019	-22.240	-1.957
Sat	14.8292	4.379	3.387	0.001	6.246	23.412
Cloudy	7.0353	2.887	2.437	0.015	1.376	12.694
Snow	-27.1796	4.732	-5.744	0.000	-36.455	-17.904

Omnibus:	2809.628	**Durbin-Watson:**	0.755
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5577.654
Skew:	1.218	**Prob(JB):**	0.00
Kurtosis:	4.921	**Cond. No.**	159.

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [29]:

```
print('R2 from step-forward selection for train set:', sub.rsquared)
print('R2 for train set from Q4:', model.rsquared)
```

```
R2 from step-forward selection for train set: 0.40587568903028415
R2 for train set from Q4: 0.40635120125670743
```

R2 from this question's method is similar.

Question 6: Polynomial Regression

We will now try to improve the performance of the regression model by including higher-order polynomial terms.

6.1 For each continuous predictor X_j, include additional polynomial terms X_j^2, X_j^3, and X_j^4, and fit a polynomial regression model to the expanded training set. How does the R^2 of this model on the test set compare with that of the linear model fitted in the previous question? Using a t-tests, find out which of the estimated coefficients for the polynomial terms are statistically significant at a significance level of 5%.

In [30]:

```
# cts predictors = numeric variables that have infinite # of values in any bound
ed interval
# predictors used: temp, atemp, hum, year, hour. All others do not fit the defii
ntion of a cts predictor

y_train = BBS_train['counts']
y_test = BBS_test['counts']
x_train = BBS_train[['temp', 'atemp', 'hum', 'year', 'hour', ]]
x_test = BBS_test[['temp', 'atemp', 'hum', 'year','hour']]

# Have to transform to 4th degree for every term

transform = PolynomialFeatures(degree=4)
new_features = transform.fit_transform(x_train)

x_reg = sm.add_constant(new_features)
polymodel = sm.OLS(y_train, x_reg).fit()

polymodel.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.599
Model:	OLS	**Adj. R-squared:**	0.596
Method:	Least Squares	**F-statistic:**	198.0
Date:	Wed, 18 Jul 2018	**Prob (F-statistic):**	0.00
Time:	22:42:50	**Log-Likelihood:**	-85584.
No. Observations:	13903	**AIC:**	1.714e+05
Df Residuals:	13798	**BIC:**	1.722e+05
Df Model:	104		

Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	194.4786	192.397	1.011	0.312	-182.646	571.603
x1	-5379.5635	3000.204	-1.793	0.073	-1.13e+04	501.244
x2	5447.2103	3069.617	1.775	0.076	-569.656	1.15e+04
x3	-730.6966	791.054	-0.924	0.356	-2281.269	819.876
x4	-25.3176	36.312	-0.697	0.486	-96.494	45.858
x5	-55.9071	14.504	-3.855	0.000	-84.337	-27.477
x6	1.495e+04	1.91e+04	0.781	0.435	-2.25e+04	5.24e+04
x7	-3.097e+04	4.06e+04	-0.762	0.446	-1.11e+05	4.86e+04
x8	2.162e+04	9660.473	2.238	0.025	2685.029	4.06e+04
x9	-262.7620	659.262	-0.399	0.690	-1555.006	1029.481
x10	242.9517	175.177	1.387	0.165	-100.419	586.323
x11	1.523e+04	2.24e+04	0.680	0.497	-2.87e+04	5.92e+04
x12	-2.061e+04	1e+04	-2.053	0.040	-4.03e+04	-933.292
x13	261.4639	687.532	0.380	0.704	-1086.193	1609.120
x14	-339.7512	181.278	-1.874	0.061	-695.080	15.578
x15	794.9902	1300.806	0.611	0.541	-1754.767	3344.747
x16	161.4201	163.480	0.987	0.323	-159.024	481.864
x17	174.0084	35.696	4.875	0.000	104.039	243.977
x18	-26.3682	36.337	-0.726	0.468	-97.594	44.858
x19	-3.1522	2.761	-1.141	0.254	-8.565	2.261
x20	5.2414	0.997	5.259	0.000	3.288	7.195
x21	-1.41e+05	8.38e+04	-1.683	0.092	-3.05e+05	2.32e+04
x22	4.406e+05	2.8e+05	1.574	0.116	-1.08e+05	9.89e+05
x23	-7.109e+04	3.15e+04	-2.257	0.024	-1.33e+05	-9356.653
x24	2969.7000	7041.333	0.422	0.673	-1.08e+04	1.68e+04
x25	130.8530	652.895	0.200	0.841	-1148.910	1410.616
x26	-4.382e+05	3.19e+05	-1.374	0.169	-1.06e+06	1.87e+05
x27	1.271e+05	6.72e+04	1.891	0.059	-4633.011	2.59e+05
x28	-5861.6714	1.51e+04	-0.388	0.698	-3.55e+04	2.38e+04
x29	-274.1814	1372.278	-0.200	0.842	-2964.034	2415.671

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
x30	-2.259e+04	1.17e+04	-1.926	0.054	-4.56e+04	395.677
x31	1155.7617	2199.497	0.525	0.599	-3155.550	5467.074
x32	-481.0418	317.597	-1.515	0.130	-1103.575	141.492
x33	-263.2050	659.289	-0.399	0.690	-1555.501	1029.091
x34	-21.8144	43.801	-0.498	0.618	-107.671	64.042
x35	-6.3392	7.854	-0.807	0.420	-21.733	9.055
x36	1.411e+05	1.23e+05	1.150	0.250	-9.95e+04	3.82e+05
x37	-5.795e+04	3.8e+04	-1.525	0.127	-1.32e+05	1.65e+04
x38	3239.3957	8209.456	0.395	0.693	-1.29e+04	1.93e+04
x39	158.9597	758.488	0.210	0.834	-1327.780	1645.699
x40	2.189e+04	1.24e+04	1.760	0.078	-2483.886	4.63e+04
x41	-1671.7867	2357.972	-0.709	0.478	-6293.732	2950.159
x42	625.3652	335.269	1.865	0.062	-31.807	1282.538
x43	263.0103	687.622	0.382	0.702	-1084.822	1610.842
x44	35.6373	46.223	0.771	0.441	-54.966	126.240
x45	17.8317	8.525	2.092	0.036	1.121	34.542
x46	-551.7716	1105.093	-0.499	0.618	-2717.905	1614.362
x47	-138.7703	294.344	-0.471	0.637	-715.725	438.184
x48	-86.3212	35.174	-2.454	0.014	-155.267	-17.376
x49	160.3155	163.447	0.981	0.327	-160.062	480.694
x50	1.7604	7.647	0.230	0.818	-13.228	16.749
x51	-13.7373	1.378	-9.969	0.000	-16.438	-11.036
x52	-26.3721	36.337	-0.726	0.468	-97.598	44.854
x53	-3.1485	2.762	-1.140	0.254	-8.562	2.264
x54	0.8940	0.188	4.765	0.000	0.526	1.262
x55	-0.1495	0.039	-3.839	0.000	-0.226	-0.073
x56	-2.585e+04	2.48e+04	-1.042	0.298	-7.45e+04	2.28e+04
x57	2.614e+05	1.56e+05	1.677	0.093	-4.41e+04	5.67e+05
x58	4.537e+04	3.22e+04	1.411	0.158	-1.77e+04	1.08e+05
x59	1.032e+05	5.64e+04	1.830	0.067	-7344.690	2.14e+05
x60	18.6455	767.123	0.024	0.981	-1485.019	1522.310

x61	-7.003e+05	4.27e+05	-1.641	0.101	-1.54e+06	1.36e+05
x62	-6.463e+04	9.41e+04	-0.687	0.492	-2.49e+05	1.2e+05
x63	-3.415e+05	1.89e+05	-1.811	0.070	-7.11e+05	2.81e+04
x64	412.8136	2217.864	0.186	0.852	-3934.501	4760.128
x65	3.597e+04	1.42e+04	2.527	0.012	8064.929	6.39e+04
x66	-1477.0209	1.28e+04	-0.115	0.908	-2.66e+04	2.36e+04
x67	-309.6330	416.886	-0.743	0.458	-1126.785	507.519
x68	2969.7024	7041.333	0.422	0.673	-1.08e+04	1.68e+04
x69	35.1444	395.326	0.089	0.929	-739.748	810.037
x70	-2.5977	7.250	-0.358	0.720	-16.810	11.614
x71	7.101e+05	4.44e+05	1.601	0.110	-1.6e+05	1.58e+06
x72	-1.326e+04	1.09e+05	-0.122	0.903	-2.26e+05	2e+05
x73	3.716e+05	2.11e+05	1.763	0.078	-4.17e+04	7.85e+05
x74	-1334.0062	2675.533	-0.499	0.618	-6578.415	3910.402
x75	-6.016e+04	2.89e+04	-2.081	0.037	-1.17e+05	-3499.387
x76	7756.1587	2.88e+04	0.269	0.788	-4.88e+04	6.43e+04
x77	490.8522	874.227	0.561	0.574	-1222.751	2204.455
x78	-5861.6710	1.51e+04	-0.388	0.698	-3.55e+04	2.38e+04
x79	-42.1831	871.527	-0.048	0.961	-1750.495	1666.129
x80	24.4760	14.925	1.640	0.101	-4.780	53.732
x81	6149.6538	5478.469	1.123	0.262	-4588.890	1.69e+04
x82	-3667.6936	2799.549	-1.310	0.190	-9155.191	1819.804
x83	336.3379	198.092	1.698	0.090	-51.950	724.626
x84	1155.7606	2199.496	0.525	0.599	-3155.551	5467.073
x85	-8.6291	73.143	-0.118	0.906	-152.000	134.742
x86	0.8193	5.622	0.146	0.884	-10.201	11.839
x87	-263.2053	659.289	-0.399	0.690	-1555.501	1029.091
x88	-21.8137	43.801	-0.498	0.618	-107.670	64.043
x89	1.7086	2.210	0.773	0.440	-2.624	6.041
x90	-0.0094	0.181	-0.052	0.959	-0.365	0.346
x91	-2.468e+05	1.58e+05	-1.565	0.118	-5.56e+05	6.23e+04

x92	3.248e+04	4.69e+04	0.693	0.488	-5.94e+04	1.24e+05
x93	-1.336e+05	7.87e+04	-1.697	0.090	-2.88e+05	2.07e+04
x94	799.0665	1201.984	0.665	0.506	-1556.986	3155.119
x95	2.599e+04	1.62e+04	1.602	0.109	-5803.796	5.78e+04
x96	-6993.3972	1.64e+04	-0.427	0.669	-3.91e+04	2.51e+04
x97	-172.9566	494.086	-0.350	0.726	-1141.433	795.519
x98	3239.3952	8209.456	0.395	0.693	-1.29e+04	1.93e+04
x99	16.9555	481.625	0.035	0.972	-927.094	961.005
x100	-17.0032	9.079	-1.873	0.061	-34.799	0.793
x101	-6703.9386	5943.196	-1.128	0.259	-1.84e+04	4945.534
x102	5145.4506	3082.379	1.669	0.095	-896.431	1.12e+04
x103	-376.5447	212.875	-1.769	0.077	-793.809	40.719
x104	-1671.7869	2357.972	-0.709	0.478	-6293.733	2950.159
x105	-17.2225	81.819	-0.210	0.833	-177.600	143.155
x106	-5.3222	6.255	-0.851	0.395	-17.582	6.938
x107	263.0100	687.622	0.382	0.702	-1084.822	1610.842
x108	35.6355	46.223	0.771	0.441	-54.968	126.239
x109	-2.2893	2.439	-0.939	0.348	-7.069	2.491
x110	-0.4206	0.200	-2.104	0.035	-0.812	-0.029
x111	381.7725	425.757	0.897	0.370	-452.770	1216.315
x112	-282.7143	268.797	-1.052	0.293	-809.593	244.165
x113	-20.3638	16.330	-1.247	0.212	-52.372	11.645
x114	-138.7698	294.344	-0.471	0.637	-715.724	438.185
x115	10.8558	9.213	1.178	0.239	-7.203	28.914
x116	5.0698	0.683	7.424	0.000	3.731	6.408
x117	160.3155	163.447	0.981	0.327	-160.063	480.694
x118	1.7596	7.647	0.230	0.818	-13.229	16.748
x119	-0.5191	0.282	-1.843	0.065	-1.071	0.033
x120	0.2680	0.024	11.345	0.000	0.222	0.314
x121	-26.3720	36.337	-0.726	0.468	-97.598	44.854
x122	-3.1486	2.762	-1.140	0.254	-8.562	2.264

x123	0.8828	0.188	4.702	0.000	0.515	1.251
x124	-0.0586	0.008	-7.078	0.000	-0.075	-0.042
x125	0.0012	0.001	1.822	0.068	-9.43e-05	0.003

Omnibus:	3489.942	**Durbin-Watson:**	0.935
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	10113.866
Skew:	1.312	**Prob(JB):**	0.00
Kurtosis:	6.251	**Cond. No.**	1.25e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 9.15e-19. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [33]:

```
print('R2 value:', polymodel.rsquared)
```

```
R2 value: 0.5988251844985915
```

In [34]:

```
print(polymodel.params)
```

```
const        194.478590
x1         -5379.563481
x2          5447.210291
x3          -730.696575
x4           -25.317642
x5           -55.907054
x6         14947.579485
x7        -30966.590859
x8         21620.869413
x9          -262.762042
x10          242.951699
x11        15234.919204
x12       -20606.345455
x13          261.463927
x14         -339.751207
x15          794.990213
x16          161.420080
x17          174.008405
x18          -26.368153
x19           -3.152224
x20            5.241419
```

```
x21     -141032.071057
x22      440600.388662
x23      -71087.300917
x24        2969.700031
x25         130.852976
x26     -438197.914257
x27      127130.421726
x28       -5861.671415
x29        -274.181362
             ...
x96       -6993.397181
x97        -172.956566
x98        3239.395244
x99          16.955460
x100        -17.003225
x101      -6703.938565
x102       5145.450648
x103       -376.544717
x104      -1671.786906
x105        -17.222467
x106         -5.322175
x107        263.010046
x108         35.635514
x109         -2.289276
x110         -0.420578
x111        381.772487
x112       -282.714335
x113        -20.363841
x114       -138.769764
x115         10.855825
x116          5.069788
x117        160.315498
x118          1.759641
x119         -0.519068
x120          0.268017
x121        -26.371976
x122         -3.148576
x123          0.882827
x124         -0.058614
x125          0.001244
Length: 126, dtype: float64
```

t for df=104 is: 1.9830, t values for each coeff are listed two cells above this.

This model's R2 is significantly higher, 0.59 vs 0.406.

Written Report to the Administrators

Question 7

Write a short summary report, intended for the administrators of the company, to address two major points (can be written as two large paragraphs):

1. How to predict ridership well (which variables are important, when is ridership highest/lowest, etc.).
2. Suggestions on how to increase the system revenue (what additional services to provide, when to give discounts, etc.).

Include your report below. The report should not be longer than 300 words and should include a maximum of 3 figures.

Answers

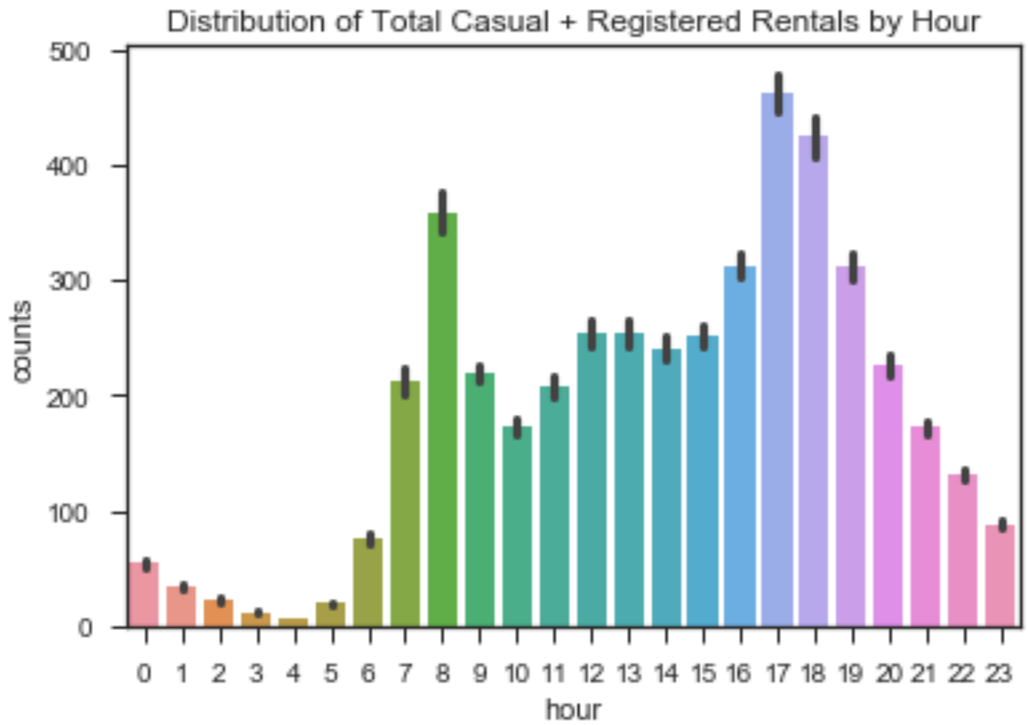
7

Predicting ridership can be difficult, but with the use of one technique, I believe that it might be possible. The first method I tried to use to predict ridership was a simple multiple linear regression, where all the different predictors were used. That only gave us an R2 score of around 0.41, which isn't very good, as we are trying to get as close to 1 as possible. The second technique I used was a regression model with higher-order polynomial terms. Using this, I was able to increase the R2 score to 0.598, which is as high as I was able to get it. To do this, continuous predictors had to be used. In this case, they were temp, atemp, hum, year, and hour. Using this and the graphs in the rest of the report, we will able to see a few different trends in ridership, such as casual riders increased tremendously in the weekends and during the nicer months, whereas registered rider counts stayed about the same year round.

Using all of the data in this report, I came up with two different scenarios that could increase system revenue. The first one would be to charge riders less at later times of the day, whether it is a registered or casual user. This might push them towards trying out the bikes. A second suggestion to increase ridership would be to implement surge pricing, similar to Uber. For example, increase prices during morning commute or afternoon commutes. On the other hand, during the off months, such as the Fall or winter days, lower the price below the regular rate to get people to use the bikes. The charts below show how the ridership decreased based on season and how there is a surge of users during morning commute times.

In [35]:

```
plt.title("Distribution of Total Casual + Registered Rentals by Hour")
sns.set(style="whitegrid")
ax = sns.barplot(x="hour", y="counts", data=bikes_df)
```



In [77]:

```
fig, ax = plt.subplots(figsize=(8,6))
a = sns.violinplot(x="season", y="counts", data=bikes_by_day)
fig.suptitle('Total Rider Data by Weather (by day)', fontsize=20);
a.set_xlabel(' Winter                              Summer                  Spring
Fall', fontsize=13, fontweight='bold')
a.set_ylabel('Total Rider Count', fontsize=13, fontweight='bold')
fig.show()
```

```
/Users/sishiryeety/anaconda3/lib/python3.6/site-packages/matplotlib/
figure.py:459: UserWarning: matplotlib is currently using a non-GUI
backend, so cannot show the figure
  "matplotlib is currently using a non-GUI backend, "
```