

cs109a_hw3_109

October 1, 2018

1 CS109A Introduction to Data Science:

1.1 Homework 3 - Forecasting Bike Sharing Usage

Harvard University Fall 2018 Instructors: Pavlos Protopapas, Kevin Rader

In [1]: *#RUN THIS CELL*

```
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/master/
HTML(styles)
```




Out[1]: <IPython.core.display.HTML object>

1.1.1 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.
- As much as possible, try and stick to the hints and functions we import at the top of the homework, as those are the ideas and tools the class supports and is aiming to teach. And if a problem specifies a particular library you're required to use that library, and possibly others from the import list.

Names of people you have worked with goes here:

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

 Unlock Pick up a bike at one of hundreds of stations around the metro DC area. See bike availability on the System Map or mobile app .	 Ride Take as many short rides as you want while your pass is active. Passes and memberships include unlimited trips under 30 minutes.	 Return End a ride by returning your bike to any station. Push your bike firmly into an empty dock and wait for the green light to make sure it's locked.
---	--	--

bike_sharing

1.1.2 Overview

You are hired by the administrators of the [Capital Bikeshare 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**. Your task is to prepare a short report summarizing your findings and make recommendations.

The predicted hourly demand could be used for planning the number of bikes that need to be available in the system at any given hour of the day. 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 to predict bike usage. 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 all available information given to you.

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).

1.1.3 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 pandas.plotting import scatter_matrix

import seaborn as sns

%matplotlib inline

In [3]: plt.style.use('seaborn')
```

1.2 Data Exploration & Preprocessing, Multiple Linear Regression, Subset Selection

1.2.1 Overview

The initial data set is provided in the file `data/BSS_hour_raw.csv`. You will first add features that will help with the analysis and then separate the data into training and test sets. Each row in this file represents the number of rides by registered users and casual users in a given hour of a specific

date. There are 12 attributes in total describing besides the number of users the weather if it is a holiday or not etc:

- `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)

1.2.2 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!

1.2.3 Resources

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html

Question 1: Data Read-In and Cleaning

In this section, we read in the data and begin one of the most important analytic steps: verifying that the data is what it claims to be.

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, 1 for 2012, etc. - `month` with 1 through 12, with 1 denoting January. - `counts` with the total number of bike rentals for that **hour** (this is the response variable for later).

1.2.4 Answers

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?

In [4]: # your code here

```
df = pd.read_csv('./data/BSS_hour_raw.csv')
df.head()
```

Out [4]:

	dteday	season	hour	holiday	weekday	workingday	weather	temp	\
0	2011-01-01	1	0	0	6	0	1	0.24	
1	2011-01-01	1	1	0	6	0	1	0.22	
2	2011-01-01	1	2	0	6	0	1	0.22	
3	2011-01-01	1	3	0	6	0	1	0.24	
4	2011-01-01	1	4	0	6	0	1	0.24	

	atemp	hum	windspeed	casual	registered
0	0.2879	0.81	0.0	3	13
1	0.2727	0.80	0.0	8	32
2	0.2727	0.80	0.0	5	27
3	0.2879	0.75	0.0	3	10
4	0.2879	0.75	0.0	0	1

In [5]: # your code here

```
df.describe()
```

Out [5]:

	season	hour	holiday	weekday	workingday	\
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	
mean	2.501640	11.546752	0.028770	3.003683	0.682721	
std	1.106918	6.914405	0.167165	2.005771	0.465431	
min	1.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	6.000000	0.000000	1.000000	0.000000	
50%	3.000000	12.000000	0.000000	3.000000	1.000000	
75%	3.000000	18.000000	0.000000	5.000000	1.000000	
max	4.000000	23.000000	1.000000	6.000000	1.000000	

	weather	temp	atemp	hum	windspeed	\
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	
mean	1.425283	0.496987	0.475775	0.627229	0.190098	
std	0.639357	0.192556	0.171850	0.192930	0.122340	
min	1.000000	0.020000	0.000000	0.000000	0.000000	
25%	1.000000	0.340000	0.333300	0.480000	0.104500	
50%	1.000000	0.500000	0.484800	0.630000	0.194000	
75%	2.000000	0.660000	0.621200	0.780000	0.253700	
max	4.000000	1.000000	1.000000	1.000000	0.850700	

	casual	registered
count	17379.000000	17379.000000

mean	35.676218	153.786869
std	49.305030	151.357286
min	0.000000	0.000000
25%	4.000000	34.000000
50%	17.000000	115.000000
75%	48.000000	220.000000
max	367.000000	886.000000

```
In [6]: # your code here
df.dtypes
```

```
Out[6]: dteday          object
season          int64
hour           int64
holiday         int64
weekday         int64
workingday      int64
weather         int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual          int64
registered     int64
dtype: object
```

Your answer here

Observations: * It is strange that temp and atemp have maximums of 1 and are never negative. I am going to assume that these values have been standardized to this range. * It is unclear what units windspeed is measured in. * dteday should have a type that represents that it is a date, not just a general "object"

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.

```
In [7]: # your code here
df['dteday'] = pd.to_datetime(df['dteday'], format='%Y-%m-%d')
df.dtypes
```

```
Out[7]: dteday          datetime64[ns]
season          int64
hour           int64
holiday         int64
weekday         int64
workingday      int64
weather         int64
temp           float64
atemp          float64
```

```

hum                float64
windspeed          float64
casual              int64
registered          int64
dtype: object

```

1.3 Create three new columns in the dataframe:

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

```

In [8]: # your code here
df['year'] = df['dteday'].apply(lambda x : x.year)
df['month'] = df['dteday'].apply(lambda x : x.month)
df['counts'] = df.casual + df.registered

```

```

In [9]: # your code here
df.head()

```

```

Out[9]:
   dteday  season  hour  holiday  weekday  workingday  weather  temp  \
0 2011-01-01      1    0        0         6           0         1  0.24
1 2011-01-01      1    1        0         6           0         1  0.22
2 2011-01-01      1    2        0         6           0         1  0.22
3 2011-01-01      1    3        0         6           0         1  0.24
4 2011-01-01      1    4        0         6           0         1  0.24

   atemp  hum  windspeed  casual  registered  year  month  counts
0  0.2879  0.81         0.0        3          13  2011     1      16
1  0.2727  0.80         0.0        8          32  2011     1      40
2  0.2727  0.80         0.0        5          27  2011     1      32
3  0.2879  0.75         0.0        3          10  2011     1      13
4  0.2879  0.75         0.0        0           1  2011     1       1

```

Question 2: Exploratory Data Analysis.

In this question, we continue validating the data, and begin hunting for patterns in ridership that shed light on who uses the service and why.

2.1 Use pandas' `scatter_matrix` command to visualize the inter-dependencies among all predictors in the dataset. Note and comment on any strongly related variables. [This will take several minutes to run. You may wish to comment it out until your final submission, or only plot a randomly-selected 10% of the rows]

2.2 Make a plot showing the *average* number of casual and registered riders during each hour of the day. `.groupby` and `.aggregate` should make this task easy. Comment on the trends you observe.

2.3 Use the variable `weather` to show how each weather category affects the relationships in question 2.2. What do you observe?

2.4 Make a new dataframe with the following subset of attributes from the previous dataset and with each entry being just **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 (normalized)
- `atemp`, the average atemp that day (normalized)
- `windspeed`, the average windspeed that day (normalized)
- `hum`, the average humidity that day (normalized)
- `casual`, the **total** number of rentals by casual users
- `registered`, the **total** number of rentals by registered users
- `counts`, the **total** number of rentals of that day

Name this dataframe `bikes_by_day`.

Make a plot showing the *distribution* of the number of casual and registered riders on each day of the week.

2.5 Use `bikes_by_day` to visualize how the distribution of **total number of rides** per day (casual and registered riders combined) varies with the **season**. Do you see any **outliers**? Here we use the pyplot's boxplot function definition of an outlier as any value 1.5 times the IQR above the 75th percentile or 1.5 times the IQR below the 25th percentiles. If you see any outliers, identify those dates and investigate if they are a chance occurrence, an error in the data collection, or a significant event (an online search of those date(s) might help).

1.2.5 Answers

2.1 Use pandas' `scatter_matrix` command to visualize the inter-dependencies among all predictors in the dataset. Note and comment on any strongly related variables. [This will take several minutes to run. You may wish to comment it out until your final submission, or only plot a randomly-selected 10% of the rows]

```
In [10]: # your code here
         plot = pd.scatter_matrix(df, figsize = (14, 14))
         plot
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel/__main__.py:2: FutureWarning:
from ipykernel import kernelapp as app
```

```
Out[10]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x10c5bb9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11ca61f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11ca8f9e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cac14a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cae6f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cae6f60>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cb414a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cb66f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cb939e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cbc64a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cbecf28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cc9c9e8>],
```

```

<matplotlib.axes._subplots.AxesSubplot object at 0x11ccc94a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11ca2bf28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cc1a9e8>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x11cc474a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x10c5e1f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11224c9e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11227a4a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cc76f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cce39e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cd124a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cd36f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cd659e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cd954a8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11cdb9ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x11db519b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21366470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2138cef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c213b99b0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x1c213e8470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2140def0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c214399b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2146a470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21490ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c214be9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c214ed470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21512ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2153f9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2156f470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21593ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c215c09b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c215f2470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21617ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c216469b0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x1c21675470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21698ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c216c69b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c216f5470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2171bef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2174a9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21779470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2179fef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c217cc9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c217fa470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c21820ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2184e9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2187d470>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2253bef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c225699b0>],

```

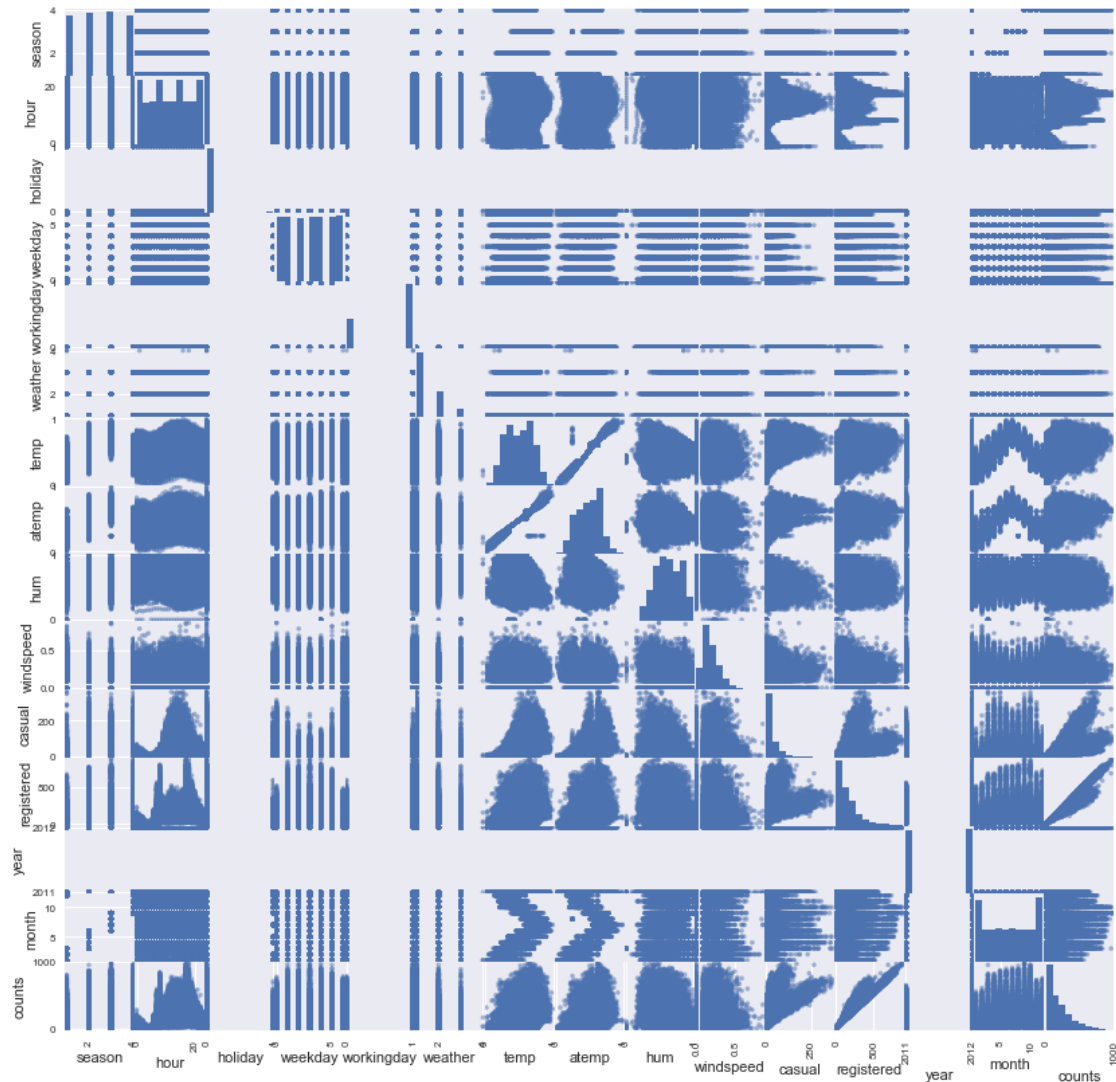

[illegible]

[illegible]

```

<matplotlib.axes._subplots.AxesSubplot object at 0x1c23e0d400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23e32e80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23e61940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23e91400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23eb6e80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23ee3940>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x1c23f11400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23f38e80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23f66940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23f95400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23fbae80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c23fe7940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c24015400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2403de80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2406a940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2409a400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c240bde80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c240ed940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2411c400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c24140e80>,
<matplotlib.axes._subplots.AxesSubplot object at 0x1c2416e940>]],
dtype=object)

```



your answer here

There are some collinearities in the data. Notably, temp vs. atemp vs. month vs. season. There seems to be a lot of structure in the data relating counts to various times (hour, holiday, weekday, workingday, month).

2.2 Make a plot showing the *average* number of casual and registered riders during each hour of the day. `.groupby` and `.aggregate` should make this task easy. Comment on the trends you observe.

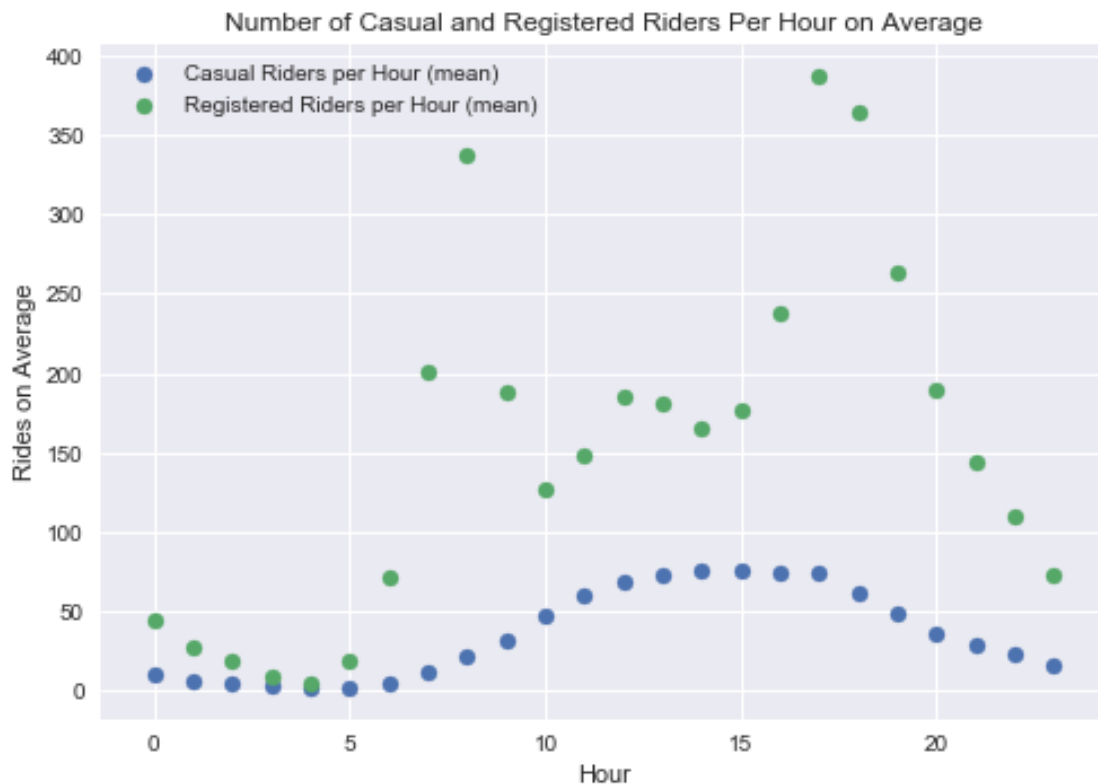
```
In [11]: # your code here
rides_per_hour = df.groupby('hour').agg({
    "casual" : 'mean',
    'registered' : 'mean'
})
rides_per_hour.head()
```

```
Out[11]:
```

hour	casual	registered
0	10.158402	43.739669
1	6.504144	26.871547
2	4.772028	18.097902
3	2.715925	9.011478
4	1.253945	5.098996

```
In [12]: plt.scatter(rides_per_hour.index, rides_per_hour.casual, label = "Casual Riders per Hour")
plt.scatter(rides_per_hour.index, rides_per_hour.registered, label = "Registered Riders per Hour")
plt.xlabel('Hour')
plt.ylabel('Rides on Average')
plt.legend()
plt.title('Number of Casual and Registered Riders Per Hour on Average')
```

```
Out[12]: Text(0.5,1,'Number of Casual and Registered Riders Per Hour on Average')
```



your answer here

Most casual rides occur in the middle of the day, whereas most registered riders take trips during rush hour (7-9am, 4-7pm). The rides are periodic.

2.3 Use the variable weather to show how each weather category affects the relationships in question 2.2. What do you observe?

```
In [13]: # your code here
rides_per_hour = df.groupby(['hour', 'weather']).agg({
    "casual" : 'mean',
    'registered' : 'mean'
})
rides_per_hour.reset_index(inplace=True)
rides_per_hour.head()
```

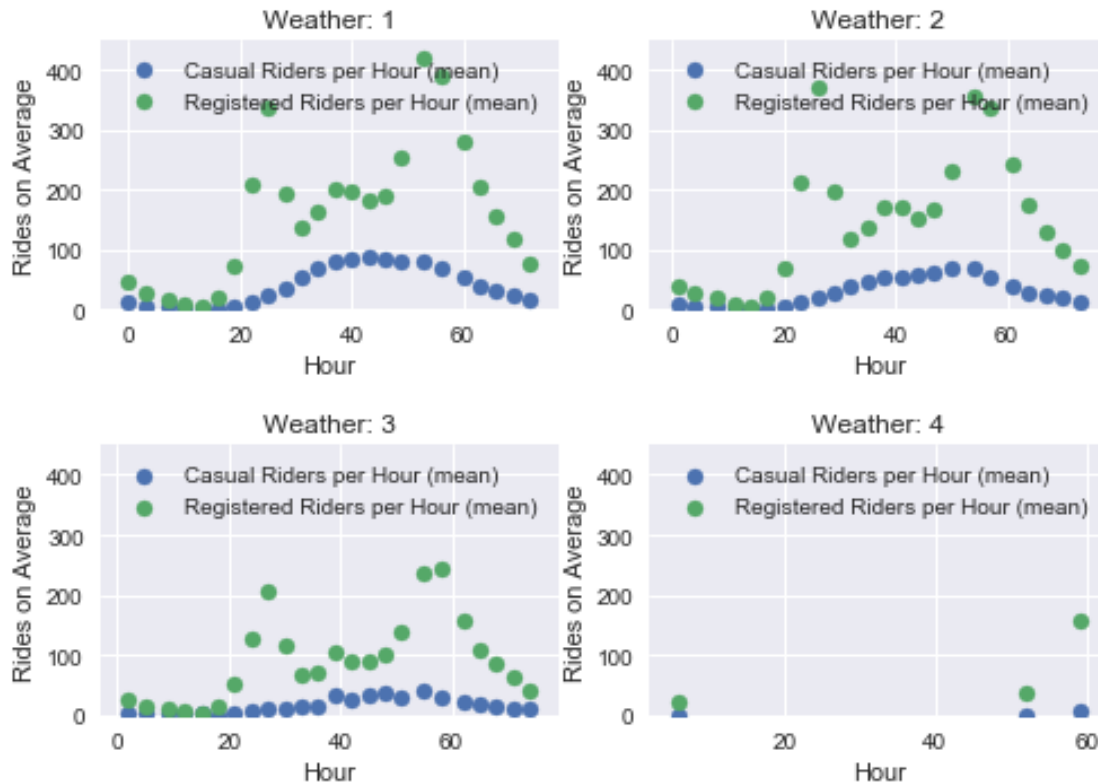
```
Out[13]:
```

	hour	weather	casual	registered
0	0	1	11.429448	47.732106
1	0	2	8.648649	38.583784
2	0	3	3.576923	24.538462
3	1	1	6.951020	27.444898
4	1	2	6.536313	29.005587

```
In [14]: # your code here
fig, ax = plt.subplots(2,2)
for id1, row in enumerate(ax):
    for id2, axes in enumerate(row):
        weather = 2*(id1)+ (id2)+1
        df_weather = rides_per_hour[rides_per_hour.weather == weather]
        axes.scatter(df_weather.index, df_weather.casual, label = "Casual Riders per Hour")
        axes.scatter(df_weather.index, df_weather.registered, label = "Registered Riders per Hour")
        axes.set_xlabel('Hour')
        axes.set_ylabel('Rides on Average')
        axes.set_ylim(0,450)
        axes.legend()
        axes.set_title('Weather: {}'.format(weather))
fig.suptitle("Number of Casual and Registered Riders Per Hour on Average ")
fig.subplots_adjust(hspace = 0.5)
fig.show()
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/matplotlib/figure.py:457: UserWarning: 
  "matplotlib is currently using a non-GUI backend, "
```

Number of Casual and Registered Riders Per Hour on Average



your answer here

Though the relative frequencies seem similar, the number of rides overall decreases as the weather increases from 1 to 4

2.4 Make a new dataframe with the following subset of attributes from the previous dataset and with each entry being just 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 (normalized)
- `atemp`, the average atemp that day (normalized)
- `windspeed`, the average windspeed that day (normalized)
- `hum`, the average humidity that day (normalized)
- `casual`, the **total** number of rentals by casual users
- `registered`, the **total** number of rentals by registered users
- `counts`, the **total** number of rentals of that day

Name this dataframe `bikes_by_day`.

Make a plot showing the *distribution* of the number of casual and registered riders on each day of the week.

In [15]: # your code here

```
bikes_by_day = df.groupby('dteday').agg({
    'weekday' : 'min',
    'weather' : 'max',
    'season' : 'min',
    'temp' : 'mean',
    'atemp' : 'mean',
    'windspeed' : "mean",
    'hum' : "mean",
    "casual": "sum",
    'registered': "sum",
    "counts": "sum"
})
bikes_by_day.reset_index(inplace=True)
bikes_by_day.head(10)
```

```
Out[15]:
```

	dteday	weekday	weather	season	temp	atemp	windspeed	\
0	2011-01-01	6	3	1	0.344167	0.363625	0.160446	
1	2011-01-02	0	3	1	0.363478	0.353739	0.248539	
2	2011-01-03	1	1	1	0.196364	0.189405	0.248309	
3	2011-01-04	2	2	1	0.200000	0.212122	0.160296	
4	2011-01-05	3	1	1	0.226957	0.229270	0.186900	
5	2011-01-06	4	2	1	0.204348	0.233209	0.089565	
6	2011-01-07	5	3	1	0.196522	0.208839	0.168726	
7	2011-01-08	6	3	1	0.165000	0.162254	0.266804	
8	2011-01-09	0	1	1	0.138333	0.116175	0.361950	
9	2011-01-10	1	2	1	0.150833	0.150888	0.223267	

	hum	casual	registered	counts
0	0.805833	331	654	985
1	0.696087	131	670	801
2	0.437273	120	1229	1349
3	0.590435	108	1454	1562
4	0.436957	82	1518	1600
5	0.518261	88	1518	1606
6	0.498696	148	1362	1510
7	0.535833	68	891	959
8	0.434167	54	768	822
9	0.482917	41	1280	1321

In [16]: # Set up the matplotlib figure

```
fig, axes = plt.subplots(7, 1, figsize=(7, 14), sharex=True, sharey = True)
sns.despine(left=True)
for day in range(7):
    bikes_by_weekday = bikes_by_day[bikes_by_day.weekday == day]
    sns.distplot(bikes_by_weekday.casual, kde = False, norm_hist = True, label = "Casual")
```

```

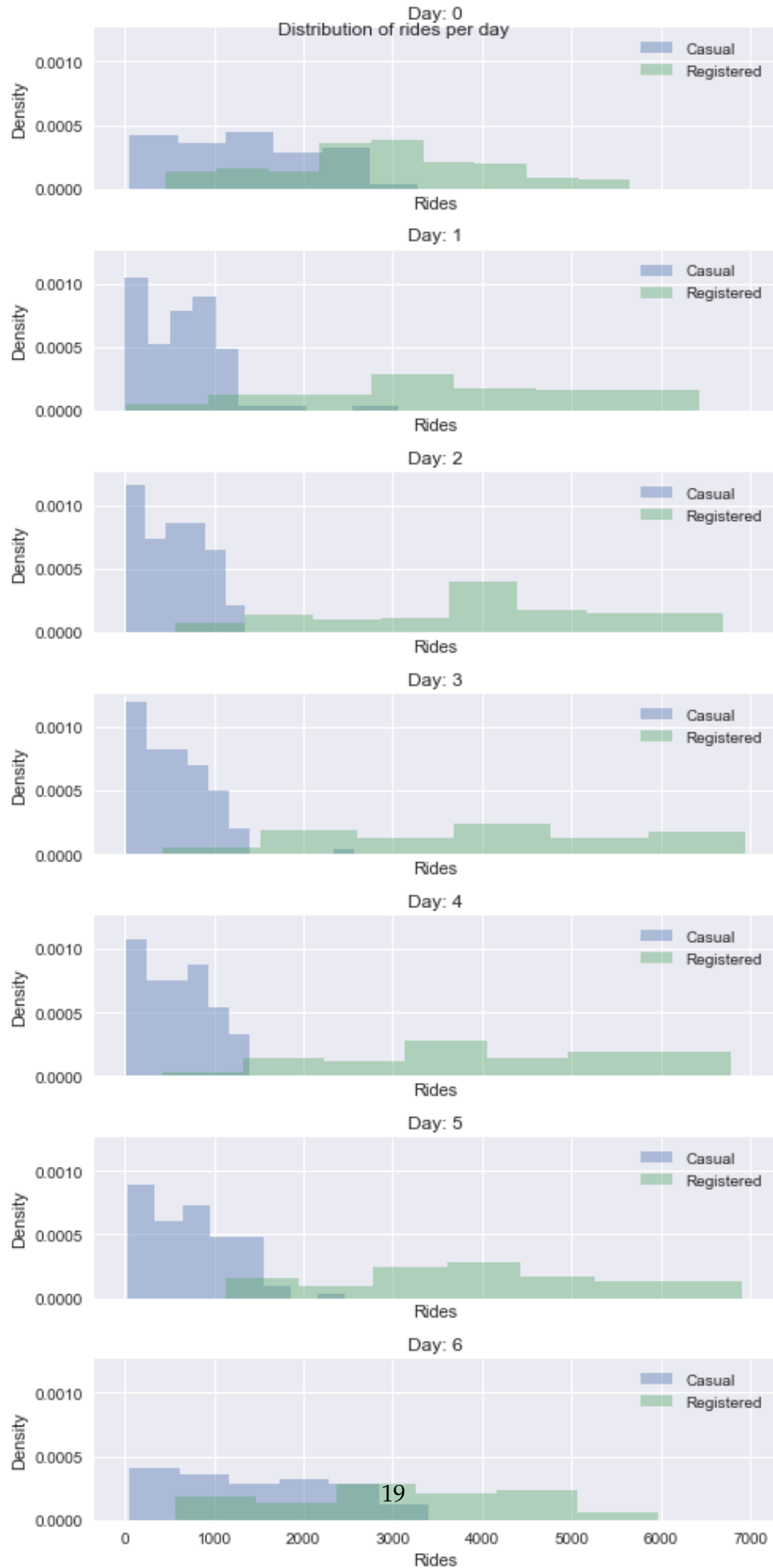
sns.distplot(bikes_by_weekday.registered, kde = False, norm_hist = True, label = '
axes[day].set_xlabel('Rides')
axes[day].set_ylabel('Density')
axes[day].set_title('Day: {}'.format(day))
axes[day].legend()
fig.suptitle("Distribution of rides per day")
fig.tight_layout()
fig.show()

```

```

/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/scipy/stats/stats.py:1713:
    return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/matplotlib/figure.py:457: U
"matplotlib is currently using a non-GUI backend, "

```



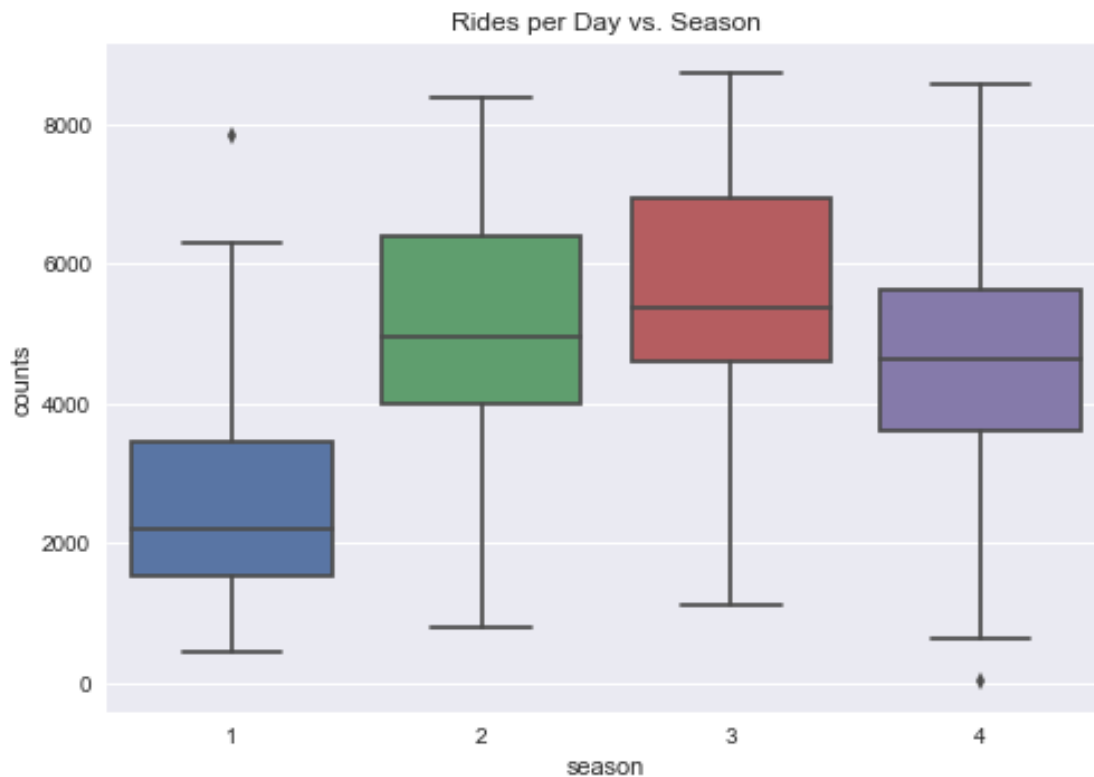
your answer here

Observations: * The casual riders increase on the weekend * The registered riders increase on weekdays

2.5 Use `bikes_by_day` to visualize how the distribution of total number of rides per day (casual and registered riders combined) varies with the season. Do you see any outliers? Here we use the `pyplot`'s `boxplot` function definition of an outlier as any value 1.5 times the IQR above the 75th percentile or 1.5 times the IQR below the 25th percentiles. If you see any outliers, identify those dates and investigate if they are a chance occurrence, an error in the data collection, or a significant event (an online search of those date(s) might help).

In [17]: *# your code here*

```
sns.boxplot(bikes_by_day.season, bikes_by_day.counts)
plt.title('Rides per Day vs. Season')
plt.show()
```



In [18]: *# your code here*

```
bikes_by_day[bikes_by_day.season == 1][bikes_by_day.counts > 7000]
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel/__main__.py:2: Use
from ipykernel import kernelapp as app
```

```
Out[18]:
```

	dteday	weekday	weather	season	temp	atemp	windspeed	\
441	2012-03-17	6	2	1	0.514167	0.505046	0.110704	

	hum	casual	registered	counts
441	0.755833	3155	4681	7836

```
In [19]: # your code here
```

```
bikes_by_day[bikes_by_day.season == 4][bikes_by_day.counts < 100]
```

```
/Users/joshfeldman/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel/__main__.py:2: Use
from ipykernel import kernelapp as app
```

```
Out[19]:
```

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	\
667	2012-10-29	1	3	4	0.44	0.4394	0.3582	0.88	

	casual	registered	counts
667	2	20	22

your answer here

The first anomaly on March 17th, 2012 might be explained by the fact that it was both a hot day and St. Patrick's day.

The second anomaly on October 29th, 2012 might be explained by the fact that it was during Hurricane Sandy.

Question 3: Prepare the data for Regression

In order to build and evaluate our regression models, a little data cleaning is needed. In this problem, we will explicitly create binary variables to represent the categorical predictors, set up the train-test split in a careful way, remove ancillary variables, and do a little data exploration that will be useful to consider in the regression models later.

3.1 Using `bikes_df`, with hourly data about rentals, convert the categorical attributes ('season', 'month', 'weekday', 'weather') into multiple binary attributes using **one-hot encoding**.

3.2 Split the updated `bikes_df` dataset in a train and test part. Do this in a 'stratified' fashion, ensuring that all months are equally represented in each set. Explain your choice for a splitting algorithm.

3.3 Although we asked you to create your train and test set, 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 files `data/BSS_train.csv` and `data/BSS_test.csv`. Read these two files into dataframes `BSS_train` and `BSS_test`, respectively. Remove the `dteday` column from both the train and the test dataset (its format cannot be used for analysis). Also, remove any predictors that would make predicting the count trivial. Note we gave more meaningful names to the one-hot encoded variables.

Answers

3.1 Using `bikes_df`, with hourly data about rentals, convert the categorical attributes ('season', 'month', 'weekday', 'weather') into multiple binary attributes using **one-hot encoding**.

```
In [20]: # your code here
```

```
bikes_df = pd.get_dummies(df, columns = ['season', 'month', 'weekday', 'weather'], dr
bikes_df.columns
```

```
Out[20]: Index(['dteday', 'hour', 'holiday', 'workingday', 'temp', 'atemp', 'hum',
               'windspeed', 'casual', 'registered', 'year', 'counts', 'season_2',
               'season_3', 'season_4', 'month_2', 'month_3', 'month_4', 'month_5',
               'month_6', 'month_7', 'month_8', 'month_9', 'month_10', 'month_11',
               'month_12', 'weekday_1', 'weekday_2', 'weekday_3', 'weekday_4',
               'weekday_5', 'weekday_6', 'weather_2', 'weather_3', 'weather_4'],
              dtype='object')
```

```
In [21]: bikes_df.head()
```

```
Out[21]:
```

	dteday	hour	holiday	workingday	temp	atemp	hum	windspeed	\
0	2011-01-01	0	0	0	0.24	0.2879	0.81	0.0	
1	2011-01-01	1	0	0	0.22	0.2727	0.80	0.0	
2	2011-01-01	2	0	0	0.22	0.2727	0.80	0.0	
3	2011-01-01	3	0	0	0.24	0.2879	0.75	0.0	
4	2011-01-01	4	0	0	0.24	0.2879	0.75	0.0	

	casual	registered	...	month_12	weekday_1	weekday_2	weekday_3	\
0	3	13	...	0	0	0	0	
1	8	32	...	0	0	0	0	
2	5	27	...	0	0	0	0	
3	3	10	...	0	0	0	0	
4	0	1	...	0	0	0	0	

	weekday_4	weekday_5	weekday_6	weather_2	weather_3	weather_4
0	0	0	1	0	0	0
1	0	0	1	0	0	0
2	0	0	1	0	0	0
3	0	0	1	0	0	0
4	0	0	1	0	0	0

[5 rows x 35 columns]

3.2 Split the updated bikes_df dataset in a train and test part. Do this in a 'stratified' fashion, ensuring that all months are equally represented in each set. Explain your choice for a splitting algorithm.

```
In [22]: bikes_df['month'] = df.month
         train, test = train_test_split(bikes_df,
                                       test_size = 0.2,
                                       shuffle = True,
                                       stratify = bikes_df['month'],
                                       random_state = 42
                                       )
         train = train.drop(columns = ['month'])
         test = test.drop(columns = ['month'])
```

your answer here I stratified the train test split method by months by setting the stratify argument to month. This algorithm preserves the original distribution of months in the training and test sets.

3.3 Although we asked you to create your train and test set, 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 files `data/BSS_train.csv` and `data/BSS_test.csv`. Read these two files into dataframes `BSS_train` and `BSS_test`, respectively. Remove the `dteday` column from both the train and the test dataset (its format cannot be used for analysis). Also, remove any predictors that would make predicting the count trivial. Note we gave more meaningful names to the one-hot encoded variables.

In [23]: *# your code here*

```
BSS_train = pd.read_csv('./data/BSS_train.csv')
BSS_test = pd.read_csv('./data/BSS_test.csv')
BSS_train.head()
```

```
Out[23]:
```

	Unnamed: 0	dteday	hour	holiday	year	workingday	temp	atemp	\
0	0	2011-01-01	0	0	0	0	0.24	0.2879	
1	1	2011-01-01	1	0	0	0	0.22	0.2727	
2	2	2011-01-01	2	0	0	0	0.22	0.2727	
3	3	2011-01-01	3	0	0	0	0.24	0.2879	
4	4	2011-01-01	4	0	0	0	0.24	0.2879	

	hum	windspeed	...	Dec	Mon	Tue	Wed	Thu	Fri	Sat	Cloudy	Snow	\
0	0.81	0.0	...	0	0	0	0	0	0	1	0	0	
1	0.80	0.0	...	0	0	0	0	0	0	1	0	0	
2	0.80	0.0	...	0	0	0	0	0	0	1	0	0	
3	0.75	0.0	...	0	0	0	0	0	0	1	0	0	
4	0.75	0.0	...	0	0	0	0	0	0	1	0	0	

	Storm
0	0
1	0
2	0
3	0
4	0

[5 rows x 36 columns]

In [24]: `BSS_train.columns`

```
Out[24]: Index(['Unnamed: 0', 'dteday', 'hour', 'holiday', 'year', 'workingday', 'temp',
               'atemp', 'hum', 'windspeed', 'casual', 'registered', 'counts', 'spring',
               'summer', 'fall', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
               'Sept', 'Oct', 'Nov', 'Dec', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat',
               'Cloudy', 'Snow', 'Storm'],
              dtype='object')
```

In [25]: *# your code here*

```
columns_to_drop = ['Unnamed: 0', 'casual', 'registered', 'dteday']

BSS_train = BSS_train.drop(columns=columns_to_drop)
BSS_test = BSS_test.drop(columns=columns_to_drop)
```

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) to predict counts, and report its R^2 score on the train and test sets.

4.2 Examine the estimated coefficients and report which ones are statistically significant at a significance level of 5% (p-value < 0.05). You should see some strange values, such as July producing 93 fewer rentals, all else equal, than January.

4.3 To diagnose the model, make two plots: first a histogram of the residuals, and second a plot of the residuals of the fitted model $e = y - \hat{y}$ as a function of the predicted value \hat{y} . Draw a horizontal line denoting the zero residual value on the Y-axis. What do the plots reveal about the OLS assumptions (linearity, constant variance, and normality)?

4.4 Perhaps we can do better via a model with polynomial terms. Build a dataset `X_train_poly` from `X_train` with added x^2 terms for temp, hour, and humidity. Are these polynomial terms important? How does predicted ridership change as each of temp, hour, and humidity increase?

4.5 The strange coefficients from 4.2 could also come from *multicollinearity*, where one or more predictors capture the same information as existing predictors. Why can multicollinearity lead to erroneous coefficient values? Create a temporary dataset `X_train_drop` that drops the following 'redundant' predictors from `X_train`: `workingday` `atemp` `spring` `summer` and `fall`. Fit a multiple linear regression model to `X_train_drop`. Are the estimates more sensible in this model?

1.2.6 Answers

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

```
In [26]: # your code here
y_train = BSS_train['counts']
X_train = sm.add_constant(BSS_train.drop(columns=['counts']))
model = OLS(y_train, X_train).fit()

In [27]: # your code here
y_train_pred = model.predict(X_train)

print("R squared (train): ", r2_score(y_train, y_train_pred))

y_test = BSS_test['counts']
X_test = sm.add_constant(BSS_test.drop(columns=['counts']))
y_test_pred = model.predict(X_test)

print("R squared (test): ", r2_score(y_test, y_test_pred))
```

```
R squared (train):  0.4065387827969087
```

```
R squared (test):  0.40638554757102274
```

4.2 Examine the estimated coefficients and report which ones are statistically significant at a significance level of 5% (p-value < 0.05). You should see some strange values, such as July producing 93 fewer rentals, all else equal, than January.

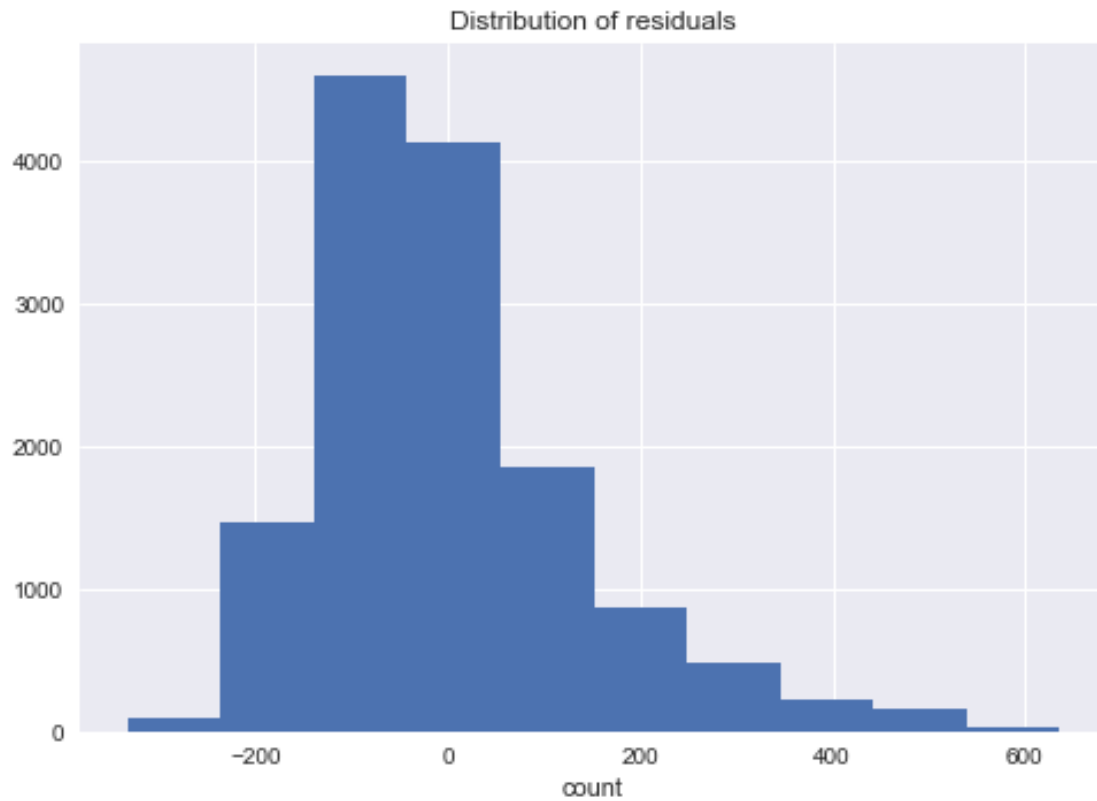

```
In [28]: # your code here
print("Significant coefficients (p < 0.05)")
print(model.pvalues[model.pvalues < 0.05])
```

Significant coefficients (p < 0.05)

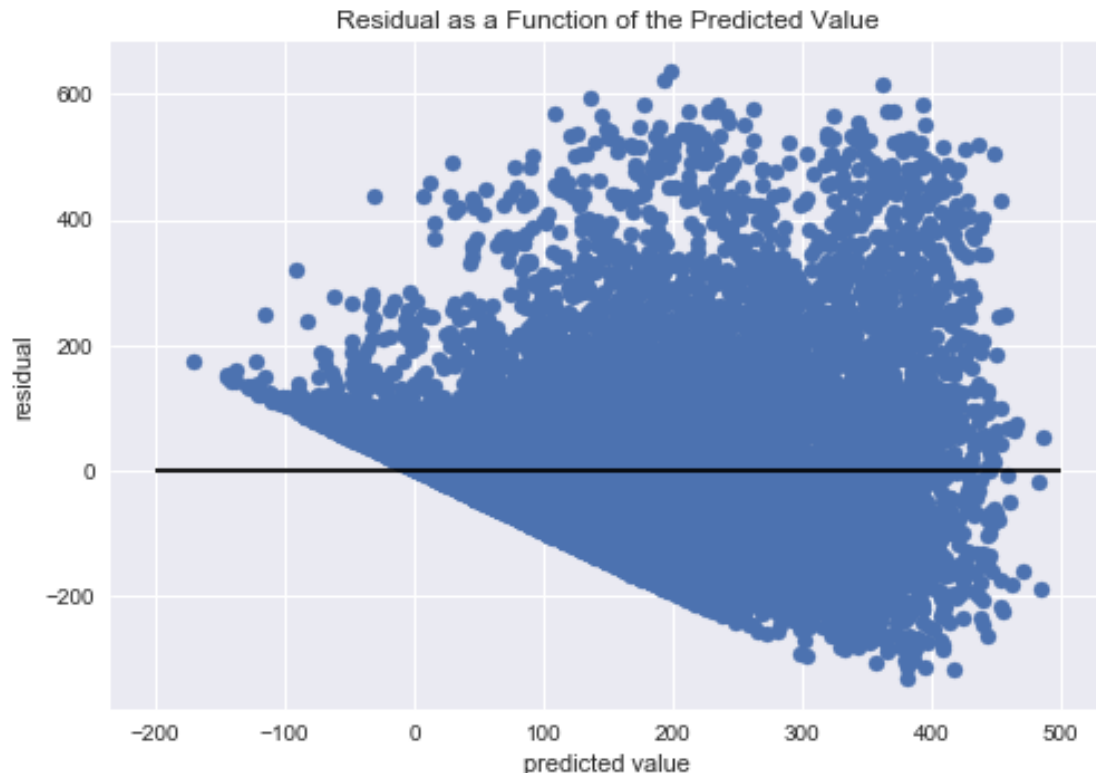
```
const      1.470264e-02
hour       0.000000e+00
holiday    6.095043e-03
year       6.205883e-218
workingday 3.905740e-05
temp       4.767468e-14
hum        2.797780e-149
windspeed  3.628163e-02
spring     6.082058e-09
summer     7.609902e-04
fall       6.106365e-20
Apr        2.640964e-05
May        1.592599e-03
Jun        8.447047e-10
Jul        1.110753e-14
Aug        5.685359e-07
Nov        6.619949e-03
Sat        6.490550e-04
Cloudy     1.926802e-02
Snow       4.454966e-09
dtype: float64
```

4.3 To diagnose the model, make two plots: first a histogram of the residuals, and second a plot of the residuals of the fitted model $e = y - \hat{y}$ as a function of the predicted value \hat{y} . Draw a horizontal line denoting the zero residual value on the Y-axis. What do the plots reveal about the OLS assumptions (linearity, constant variance, and normality)?

```
In [29]: # your code here
plt.hist(y_train - y_train_pred)
plt.xlabel('residuals')
plt.xlabel('count')
plt.title('Distribution of residuals')
plt.show()
```



```
In [30]: plt.scatter(y_train_pred,y_train-y_train_pred)
plt.hlines(0,-200,500)
plt.xlabel('predicted value')
plt.ylabel('residual')
plt.title("Residual as a Function of the Predicted Value")
plt.show()
```



your answer here

Linearity: The data might not be linear because, when we examine the histogram of the residuals, we see that the distribution is not symmetric. The distribution is right skewed, meaning that the data is likely convex.

Constant Variance: The variance is not constant because, when we examine the plot of the residuals vs. predicted value, we see that the residuals have higher variance for higher predicted values. If the variance were constant, we would expect the variance of the residuals to be equally distributed at all predicted values.

Normality: The distribution of residuals is not normal because it is asymmetric. This means that our normality assumption also does not hold

4.4 Perhaps we can do better via a model with polynomial terms. Build a dataset `X_train_poly` from `X_train` with added x^2 terms for temp, hour, and humidity. Are these polynomial terms important? How does predicted ridership change as each of temp, hour, and humidity increase?

```
In [31]: # your code here
         # your code here
         BSS_train_poly = BSS_train.copy()
         BSS_train_poly['temp_squared'] = BSS_train_poly.temp ** 2
         BSS_train_poly['hour_squared'] = BSS_train_poly.hour ** 2
         BSS_train_poly['hum_squared'] = BSS_train_poly.hum ** 2

         y_train_poly = BSS_train_poly['counts']
```

```
X_train_poly = sm.add_constant(BSS_train_poly.drop(columns=['counts']))
poly_model = OLS(y_train_poly, X_train_poly).fit()
```

```
poly_model.summary()
```

```
Out[31]: <class 'statsmodels.iolib.summary.Summary'>
```

```
"""
```

OLS Regression Results

=====						
Dep. Variable:	counts	R-squared:	0.501			
Model:	OLS	Adj. R-squared:	0.500			
Method:	Least Squares	F-statistic:	421.8			
Date:	Mon, 01 Oct 2018	Prob (F-statistic):	0.00			
Time:	16:48:07	Log-Likelihood:	-87102.			
No. Observations:	13903	AIC:	1.743e+05			
Df Residuals:	13869	BIC:	1.745e+05			
Df Model:	33					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

const	-185.2131	14.016	-13.214	0.000	-212.687	-157.739
hour	39.5786	0.662	59.777	0.000	38.281	40.876
holiday	-13.0061	6.056	-2.148	0.032	-24.877	-1.135
year	81.0305	2.199	36.854	0.000	76.721	85.340
workingday	13.2894	2.524	5.265	0.000	8.342	18.237
temp	132.7247	58.298	2.277	0.023	18.452	246.997
atemp	67.4957	43.532	1.550	0.121	-17.833	152.824
hum	11.8636	36.114	0.329	0.743	-58.925	82.652
windspeed	-6.9100	9.920	-0.697	0.486	-26.354	12.534
spring	43.7116	6.805	6.424	0.000	30.374	57.049
summer	33.9087	8.066	4.204	0.000	18.098	49.720
fall	72.1937	6.878	10.497	0.000	58.712	85.675
Feb	1.6487	5.538	0.298	0.766	-9.207	12.504
Mar	9.5583	6.304	1.516	0.129	-2.798	21.914
Apr	-10.7152	9.238	-1.160	0.246	-28.824	7.393
May	-2.7388	9.789	-0.280	0.780	-21.926	16.449
Jun	-23.0368	9.922	-2.322	0.020	-42.485	-3.588
Jul	-53.5230	11.163	-4.795	0.000	-75.405	-31.642
Aug	-23.6944	10.965	-2.161	0.031	-45.188	-2.201
Sept	10.9055	9.887	1.103	0.270	-8.475	30.286
Oct	2.8452	9.262	0.307	0.759	-15.309	20.999
Nov	-16.5926	8.897	-1.865	0.062	-34.032	0.846
Dec	-6.9106	7.078	-0.976	0.329	-20.784	6.963
Mon	-2.4620	2.731	-0.901	0.367	-7.816	2.892
Tue	-3.8629	2.943	-1.313	0.189	-9.631	1.905
Wed	2.1275	2.920	0.729	0.466	-3.596	7.851
Thu	-0.2540	2.922	-0.087	0.931	-5.981	5.473

Fri	4.7347	2.923	1.620	0.105	-0.995	10.465
Sat	16.7983	4.021	4.178	0.000	8.917	24.679
Cloudy	-8.4327	2.680	-3.146	0.002	-13.687	-3.179
Snow	-47.3269	4.583	-10.327	0.000	-56.310	-38.344
Storm	35.5800	90.246	0.394	0.693	-141.315	212.475
temp_squared	109.4437	36.470	3.001	0.003	37.958	180.930
hour_squared	-1.3570	0.027	-50.567	0.000	-1.410	-1.304
hum_squared	-108.7057	28.944	-3.756	0.000	-165.440	-51.971

```
=====
Omnibus:                2946.543    Durbin-Watson:                0.890
Prob(Omnibus):           0.000    Jarque-Bera (JB):            6094.033
Skew:                    1.253    Prob(JB):                    0.00
Kurtosis:                5.059    Cond. No.                    1.35e+16
=====
```

Warnings:

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.56e-24. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

your answer here

Temperature:

$$y = \dots + 109t^2 + 132t + \dots$$

y is a convex function of temperature.

Hour:

$$y = \dots - 1.35h^2 + 39h + \dots$$

y is a concave function of hour.

Humidity

$$y = \dots - 108h^2 + 11h + \dots$$

y is a concave function of humidity

4.5 The strange coefficients from 4.2 could also come from *multicollinearity*, where one or more predictors capture the same information as existing predictors. Why can multicollinearity lead to erroneous coefficient values? Create a temporary dataset `X_train_drop` that drops the following 'redundant' predictors from `X_train`: `workingday` `atemp` `spring` `summer` and `fall`. Fit a multiple linear regression model to `X_train_drop`. Are the estimates more sensible in this model?

```
In [32]: # your code here
# your code here
y_train = BSS_train['counts']
X_train = sm.add_constant(BSS_train.drop(columns=['counts', 'workingday', 'atemp', 'spring', 'summer', 'fall']))
model = OLS(y_train, X_train).fit()
model.summary()
```

Out[32]: <class 'statsmodels.iolib.summary.Summary'>

"""

OLS Regression Results

```

=====
Dep. Variable:          counts    R-squared:                0.402
Model:                  OLS      Adj. R-squared:           0.401
Method:                 Least Squares    F-statistic:            358.3
Date:                  Mon, 01 Oct 2018    Prob (F-statistic):      0.00
Time:                  16:48:07    Log-Likelihood:         -88363.
No. Observations:      13903    AIC:                    1.768e+05
Df Residuals:          13876    BIC:                    1.770e+05
Df Model:              26
Covariance Type:       nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	-20.0627	8.541	-2.349	0.019	-36.805	-3.321
hour	7.2378	0.185	39.095	0.000	6.875	7.601
holiday	-35.8906	7.395	-4.854	0.000	-50.385	-21.396
year	76.3039	2.389	31.945	0.000	71.622	80.986
temp	406.2359	13.279	30.593	0.000	380.208	432.264
hum	-201.5103	7.800	-25.835	0.000	-216.799	-186.221
windspeed	11.9668	10.448	1.145	0.252	-8.512	32.446
Feb	-7.6897	5.986	-1.285	0.199	-19.422	4.043
Mar	2.8889	6.158	0.469	0.639	-9.182	14.960
Apr	1.0237	6.594	0.155	0.877	-11.902	13.950
May	7.2426	7.613	0.951	0.341	-7.680	22.165
Jun	-30.6611	8.346	-3.674	0.000	-47.020	-14.302
Jul	-67.7620	9.062	-7.477	0.000	-85.525	-49.999
Aug	-34.2712	8.628	-3.972	0.000	-51.183	-17.359
Sept	20.6406	7.882	2.619	0.009	5.191	36.090
Oct	50.7025	6.823	7.431	0.000	37.329	64.076
Nov	42.3211	6.111	6.926	0.000	30.344	54.299
Dec	34.2134	5.952	5.748	0.000	22.546	45.881
Mon	9.2907	4.570	2.033	0.042	0.333	18.248
Tue	4.7929	4.442	1.079	0.281	-3.914	13.500
Wed	13.2143	4.417	2.992	0.003	4.557	21.871
Thu	8.0051	4.445	1.801	0.072	-0.708	16.718
Fri	13.0474	4.429	2.946	0.003	4.367	21.728
Sat	14.1461	4.397	3.217	0.001	5.528	22.764
Cloudy	6.7192	2.909	2.310	0.021	1.018	12.421
Snow	-29.1668	4.828	-6.041	0.000	-38.631	-19.703
Storm	40.3125	98.759	0.408	0.683	-153.267	233.893

```

=====
Omnibus:                2850.389    Durbin-Watson:           0.749
Prob(Omnibus):          0.000    Jarque-Bera (JB):        5702.134
Skew:                   1.231    Prob(JB):                0.00
Kurtosis:               4.944    Cond. No.                 1.13e+03
=====

```

```
=====
Warnings:
```

```
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
[2] The condition number is large, 1.13e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

your answer here The output is slightly more sensible although July is still negative. Multicollinearity can lead to bad coefficient estimates because (1) the matrix inversion necessary to calculate the coefficients is unstable and (2) the standard error on these coefficients increase because it there isn't a unique minimization of the loss function.

Question 5: Subset Selection

Perhaps we can automate finding a good set of predictors. This question focuses on forward stepwise selection, where predictors are added to the model one by one.

5.1 Implement forward step-wise selection to select a minimal subset of predictors that are related to the response variable. Run your code on the richest dataset, `X_train_poly`, and determine which predictors are selected.

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

Note: Implementing from scratch means you are not allowed to use a solution provided by a Python library, such as `sklearn` or use a solution you found on the internet. You have to write all of the code on your own. However you MAY use the `model.bic` attribute implemented in `statsmodels`.

5.2 Does forward selection eliminate one or more of the colinear predictors we dropped in Question 4.5 (workingday atemp spring summer and fall)? If any of the five predictors are not dropped, explain why.

5.3 Fit the linear regression model using the identified subset of predictors to the training set. How do the train and test R^2 scores for this fitted step-wise model compare with the train and test R^2 scores from the polynomial model fitted in Question 4.4?

1.2.7 Answers

5.1 Implement forward step-wise selection to select a minimal subset of predictors that are related to the response variable. Run your code on the richest dataset, `X_train_poly`, and determine which predictors are selected.

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

Note: Implementing from scratch means you are not allowed to use a solution provided by a Python library, such as `sklearn` or use a solution you found on the internet. You have to write all of the code on your own. However you MAY use the `model.bic` attribute implemented in `statsmodels`.

```
In [33]: # your code here
def forward_selection(X,y):
    included_columns = []
```

```

remaining_columns = list(X.columns)
best_overall_bic = float("inf")
for idx in range(len(X.columns)):
    best_bic = float("inf")
    for col in remaining_columns:
        columns_to_try = included_columns.copy()
        columns_to_try.append(col)
        model = OLS(y, X[columns_to_try]).fit()
        if best_bic > model.bic:
            best_bic = model.bic
            best_col = col
            best_model = model
    included_columns.append(best_col)
    remaining_columns.remove(best_col)
    if best_overall_bic > best_bic:
        best_overall_bic = best_bic
        best_overall_model = best_model
        best_columns = included_columns.copy()
return best_overall_model, best_columns

```

```

In [34]: # your code here
         best_model, best_columns = forward_selection(X_train_poly, y_train)

```

```

In [35]: print("Covariates with lowest BIC:")
         print(best_columns)

```

Covariates with lowest BIC:

```
['temp', 'hour', 'hum_squared', 'hour_squared', 'year', 'const', 'fall', 'Jul', 'Snow', 'spring']
```

5.2 Does forward selection eliminate one or more of the colinear predictors we dropped in Question 4.5 (workingday atemp spring summer and fall)? If any of the five predictors are not dropped, explain why. *your answer here*

All of the predictors except spring are dropped. Spring wasn't dropped because the variable(s) it was collinear with was(were) dropped instead.

5.3 Fit the linear regression model using the identified subset of predictors to the training set. How do the train and test R^2 scores for this fitted step-wise model compare with the train and test R^2 scores from the polynomial model fitted in Question 4.4?

```

In [36]: # your code here
         y_train = BSS_train['counts']
         y_test = BSS_test['counts']

         BSS_test_poly = BSS_test.copy()
         BSS_test_poly['temp_squared'] = BSS_test_poly.temp ** 2
         BSS_test_poly['hour_squared'] = BSS_test_poly.hour ** 2
         BSS_test_poly['hum_squared'] = BSS_test_poly.hum ** 2

```



```
X_test_poly = sm.add_constant(BSS_test_poly.drop(columns=['counts']))
```

```
print("R squared (stepwise selection, train): ",r2_score(y_train, poly_model.predict(X_train))
print("R squared (stepwise selection, test): ",r2_score(y_test, poly_model.predict(X_test))
print("R squared (stepwise selection, train): ",r2_score(y_train, best_model.predict(X_train))
print("R squared (stepwise selection, test): ",r2_score(y_test, best_model.predict(X_test))
```

```
R squared (stepwise selection, train): 0.5009041530600604
R squared (stepwise selection, test): 0.49653116472922565
R squared (stepwise selection, train): 0.4979274979859656
R squared (stepwise selection, test): 0.4943103235558135
```

```
In [37]: poly_model.summary()
```

```
Out[37]: <class 'statsmodels.iolib.summary.Summary'>
```

```
"""
```

OLS Regression Results

```
=====
Dep. Variable:          counts    R-squared:                0.501
Model:                  OLS      Adj. R-squared:           0.500
Method:                 Least Squares    F-statistic:           421.8
Date:                  Mon, 01 Oct 2018    Prob (F-statistic):      0.00
Time:                  16:48:10    Log-Likelihood:        -87102.
No. Observations:      13903    AIC:                   1.743e+05
Df Residuals:          13869    BIC:                   1.745e+05
Df Model:               33
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-185.2131	14.016	-13.214	0.000	-212.687	-157.739
hour	39.5786	0.662	59.777	0.000	38.281	40.876
holiday	-13.0061	6.056	-2.148	0.032	-24.877	-1.135
year	81.0305	2.199	36.854	0.000	76.721	85.340
workingday	13.2894	2.524	5.265	0.000	8.342	18.237
temp	132.7247	58.298	2.277	0.023	18.452	246.997
atemp	67.4957	43.532	1.550	0.121	-17.833	152.824
hum	11.8636	36.114	0.329	0.743	-58.925	82.652
windspeed	-6.9100	9.920	-0.697	0.486	-26.354	12.534
spring	43.7116	6.805	6.424	0.000	30.374	57.049
summer	33.9087	8.066	4.204	0.000	18.098	49.720
fall	72.1937	6.878	10.497	0.000	58.712	85.675
Feb	1.6487	5.538	0.298	0.766	-9.207	12.504
Mar	9.5583	6.304	1.516	0.129	-2.798	21.914
Apr	-10.7152	9.238	-1.160	0.246	-28.824	7.393
May	-2.7388	9.789	-0.280	0.780	-21.926	16.449
Jun	-23.0368	9.922	-2.322	0.020	-42.485	-3.588

Jul	-53.5230	11.163	-4.795	0.000	-75.405	-31.642
Aug	-23.6944	10.965	-2.161	0.031	-45.188	-2.201
Sept	10.9055	9.887	1.103	0.270	-8.475	30.286
Oct	2.8452	9.262	0.307	0.759	-15.309	20.999
Nov	-16.5926	8.897	-1.865	0.062	-34.032	0.846
Dec	-6.9106	7.078	-0.976	0.329	-20.784	6.963
Mon	-2.4620	2.731	-0.901	0.367	-7.816	2.892
Tue	-3.8629	2.943	-1.313	0.189	-9.631	1.905
Wed	2.1275	2.920	0.729	0.466	-3.596	7.851
Thu	-0.2540	2.922	-0.087	0.931	-5.981	5.473
Fri	4.7347	2.923	1.620	0.105	-0.995	10.465
Sat	16.7983	4.021	4.178	0.000	8.917	24.679
Cloudy	-8.4327	2.680	-3.146	0.002	-13.687	-3.179
Snow	-47.3269	4.583	-10.327	0.000	-56.310	-38.344
Storm	35.5800	90.246	0.394	0.693	-141.315	212.475
temp_squared	109.4437	36.470	3.001	0.003	37.958	180.930
hour_squared	-1.3570	0.027	-50.567	0.000	-1.410	-1.304
hum_squared	-108.7057	28.944	-3.756	0.000	-165.440	-51.971

Omnibus:	2946.543	Durbin-Watson:	0.890
Prob(Omnibus):	0.000	Jarque-Bera (JB):	6094.033
Skew:	1.253	Prob(JB):	0.00
Kurtosis:	5.059	Cond. No.	1.35e+16

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 4.56e-24. This might indicate that there are strong multicollinearity problems or that the design matrix is singular.
"""

In [38]: best_model.summary()

Out[38]: <class 'statsmodels.iolib.summary.Summary'>
"""

OLS Regression Results						
=====						
Dep. Variable:	counts	R-squared:	0.498			
Model:	OLS	Adj. R-squared:	0.497			
Method:	Least Squares	F-statistic:	1148.			
Date:	Mon, 01 Oct 2018	Prob (F-statistic):	0.00			
Time:	16:48:10	Log-Likelihood:	-87144.			
No. Observations:	13903	AIC:	1.743e+05			
Df Residuals:	13890	BIC:	1.744e+05			
Df Model:	12					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

```

-----
temp          310.9097      6.890      45.125      0.000      297.404      324.415
hour          39.5749      0.647      61.177      0.000      38.307      40.843
hum_squared   -97.3713      5.534     -17.595      0.000     -108.219     -86.524
hour_squared  -1.3582      0.026     -51.420      0.000      -1.410      -1.306
year          80.3153      2.182      36.804      0.000      76.038      84.593
const        -189.1223      5.283     -35.795      0.000     -199.479    -178.766
fall          56.0162      2.797      20.026      0.000      50.533      61.499
Jul          -25.6107      4.679      -5.473      0.000     -34.783     -16.438
Snow         -48.1081      4.447     -10.819      0.000     -56.824     -39.392
spring        25.8171      2.910       8.873      0.000      20.114      31.520
Sept         29.2489      4.302       6.798      0.000      20.816      37.682
holiday       -27.4291      6.432      -4.265      0.000     -40.037     -14.822
Cloudy        -8.4814      2.666      -3.181      0.001     -13.707      -3.256
=====
Omnibus:                2990.921   Durbin-Watson:                0.884
Prob(Omnibus):           0.000   Jarque-Bera (JB):            6264.394
Skew:                    1.265   Prob(JB):                     0.00
Kurtosis:                5.102   Cond. No.                     1.85e+03
=====

```

Warnings:

```

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.85e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""

```

your answer here The performance of the two models are similar, but the model with covariates chosen through forward stepwise selection has much lower standard errors on the coefficients. This means that the degree to which each factor contributes to the outcome is much more certain, making this model more interpretable.

2 Written Report to the Administrators [20 pts]

Question 6

Write a short report stating some of your findings on how the administrators can increase the bike share system's revenue. You might want to include suggestions such as what model to use to predict ridership, what additional services to provide, or when to give discounts, etc. Include your report as a pdf file in canvas. The report should not be longer than one page (300 words) and should include a maximum of 5 figures.

Answers 6 Recommendation 1: Incentivize the casual riders to redistribute the bikes for the registered riders.

One of the challenges of bike share programs is ensuring that stations have enough docking stations and bikes for the supply and demand. Although we do not have access to the individual station data, there are nevertheless some actions you can take to ensure everyone can access/return a bike when they need.

A key trend in your data is in the average number of riders per hour for both the casual and registered users (See Fig 1). The registered riders use the system to presumably commute to and from work. Peak times are 7am-9am and 4pm-7pm. The casual riders, who are most likely tourists, use the service most in between those times (11am-6pm).

Since the casual riders may have less restricted schedules, you could incentivize them to move the bikes around during the day to prepare for evening rush hour. This could be done through financial discounts or suggested bike routes that may help them see interesting parts of DC while conveniently guiding them to return the bikes to where they're needed.

```
In [39]: # your code here
# your code here
rides_per_hour = df.groupby('hour').agg({
    "casual" : 'mean',
    'registered' : 'mean'
})
rides_per_hour.head()
plt.scatter(rides_per_hour.index, rides_per_hour.casual, label = "Casual Riders per Hour")
plt.scatter(rides_per_hour.index, rides_per_hour.registered, label = "Registered Riders per Hour")
plt.xlabel('Hour')
plt.ylabel('Rides on Average')
plt.legend()
plt.title('Fig 1: Number of Casual and Registered Riders Per Hour on Average')
plt.show()
```



Recommendation 2: Using a Predictive Model to Help Registered Users Know Where to Get and Return Bikes

We have developed a predictive model that explains about half of the variance in your data (test set $R^2 = 0.494$). The model is a linear regression on most of your dataset with some additional terms added.

One way in which you could use this model is to help registered riders know when to pick up and return their bikes during rush hour. If the model is predicting that the service will become heavily used within the next hour, you could send alerts to riders urging them to leave earlier. Conversely, if you expect the service to quiet down in the next hour, you could encourage the rider to wait a bit, thus making it more likely that a bike will be available. This could distribute use during your peak hours, meaning that it will be less likely that any station will be totally full or empty.

Additionally, if you expect a day to be very popular with riders, you could set up a casual employment model (like uber) where individuals would move bikes to where they're most needed. While this likely would not be cost effective every day, the predictive model would help you know when to spend money on bike-movers.

your answer here

Recommendation 3: Investigate Boosting Ridership in Cooler Months

We used a method called forward stepwise variable selection to choose the variables that are most associated with number of rides per hour (see table below). The coef column represents the degree to which the covariate is associated with the number of rides. The greatest association is with the temp or temperature covariate. One potential opportunity is to see if there are ways to encourage people to bike as the temperature begins to drop.

A final insight is that the year parameter is 80. This means that if the present trend continues, you can expect an increase of 80 rides/hour per year. The future is bright for the bike share!

```
In [40]: best_model.summary().tables[1]
```

```
Out[40]: <class 'statsmodels.iolib.table.SimpleTable'>
```