CS109A Introduction to Data Science:

Homework 3 - Forecasting Bike Sharing Usage

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

In [61]:
```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A/master/content/styles/cs109.css").text
HTML(styles)
```

Out[61]:

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:

Gildas Bah, Michel Atoudem Kana

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**. 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).

Use only the libraries below:

In [0]:
```
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
```

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 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)

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)

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).

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 [0]:
```
# your code here
bikes_df = pd.read_csv('https://raw.githubusercontent.com/michelkana/c
s109a/master/BSS_hour_raw.csv')
```

In [64]:
```
bikes_df.head()
```

Out[64]:

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

In [65]:
```
# your code here
bikes_df.describe()
```

Out[65]:

	season	hour	holiday	weekday	workingday	v
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.
mean	2.501640	11.546752	0.028770	3.003683	0.682721	1.4252
std	1.106918	6.914405	0.167165	2.005771	0.465431	0.6393
min	1.000000	0.000000	0.000000	0.000000	0.000000	1.0000
25%	2.000000	6.000000	0.000000	1.000000	0.000000	1.0000
50%	3.000000	12.000000	0.000000	3.000000	1.000000	1.0000
75%	3.000000	18.000000	0.000000	5.000000	1.000000	2.0000
max	4.000000	23.000000	1.000000	6.000000	1.000000	4.0000

In [66]:
```
# your code here
bikes_df.dtypes
```

Out[66]:
```
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
```

Analysis of variables and ranges

From the structure of the data, the variable `dteday` is an object, which will be converted to reflect the business rules.

The variables season, hour, holiday, weekday, workingday, and weather are of the data type integer. However, they are descriptors that provide context the data generating process. Although their range is being provided in this descriptive results they are factor variables that will enable to subset the data.

The variables temp, atemp, hum, windspeed are all decimal suggesting that some operational business rules were used to derive them. These rules were not provided to us and we cannot verify that they were correctly implemented. We will assume they were correctly computed. Except for **temp**, which has a minimum of 0.02, they all have reach their minimum at 0.00. They all reach their maximum at 1, except for windspeed, which reaches a miximum of 0.85.

The variables casual and registered are the main variables of interest. They are count data represented as positive integers. The operational business rule for counting casual and registered were not provided. Subject to verification with the business process owner, we will assume the business rule for deciding counting records into the casual variable and for counting records into the registered variable are well defined.

It can also be noted that there a good deal of left skwedness in data as evidence by all of the instances where the mean is less that the median. For example, temp, atemp, hum, and windspeed all have means which are less than the median at 0.496987 < 0.50, 0.475775 < 0.48, 0.627229 < 0.63, and 0.190098 < 0.1904, respectively. Casual and registered have mean greater than the median suggesting that their distribution will be right skewed.

Finally, we can infer from the description of the attributes of the data model, we have four types of attributes:

1. Date and time attributes, and predictor variables
2. Weather related attributes, and predictor variables
3. Atmospheric conditions, and predictor variables
4. Customer count attributes, and the response variables

In what follows, the impact of these attributes of the business model will be studied and recommendations will be provided in a report.

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.

The following is the conversion of **dteday** attribute into a datetime object:

In [67]:
```
# Casting bike dteday as Pandas datetime
bikes_df.dteday = pd.to_datetime(bikes_df.dteday, format='%Y-%m-%d')
bikes_df.dtypes
```

Out[67]:
```
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).

The following is the creationg of three new features:

In [0]:
```
bikes_df['year'] = bikes_df.dteday.dt.year - 2011
bikes_df['month'] = bikes_df['dteday'].dt.month
bikes_df['counts'] = bikes_df.casual + bikes_df.registered
```

In [69]:
```
# Check to ensure feature are corrctly created
bikes_df.head()
```

Out[69]:

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

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 occurence, an error in the data collection, or a significant event (an online search of those date(s) might help).

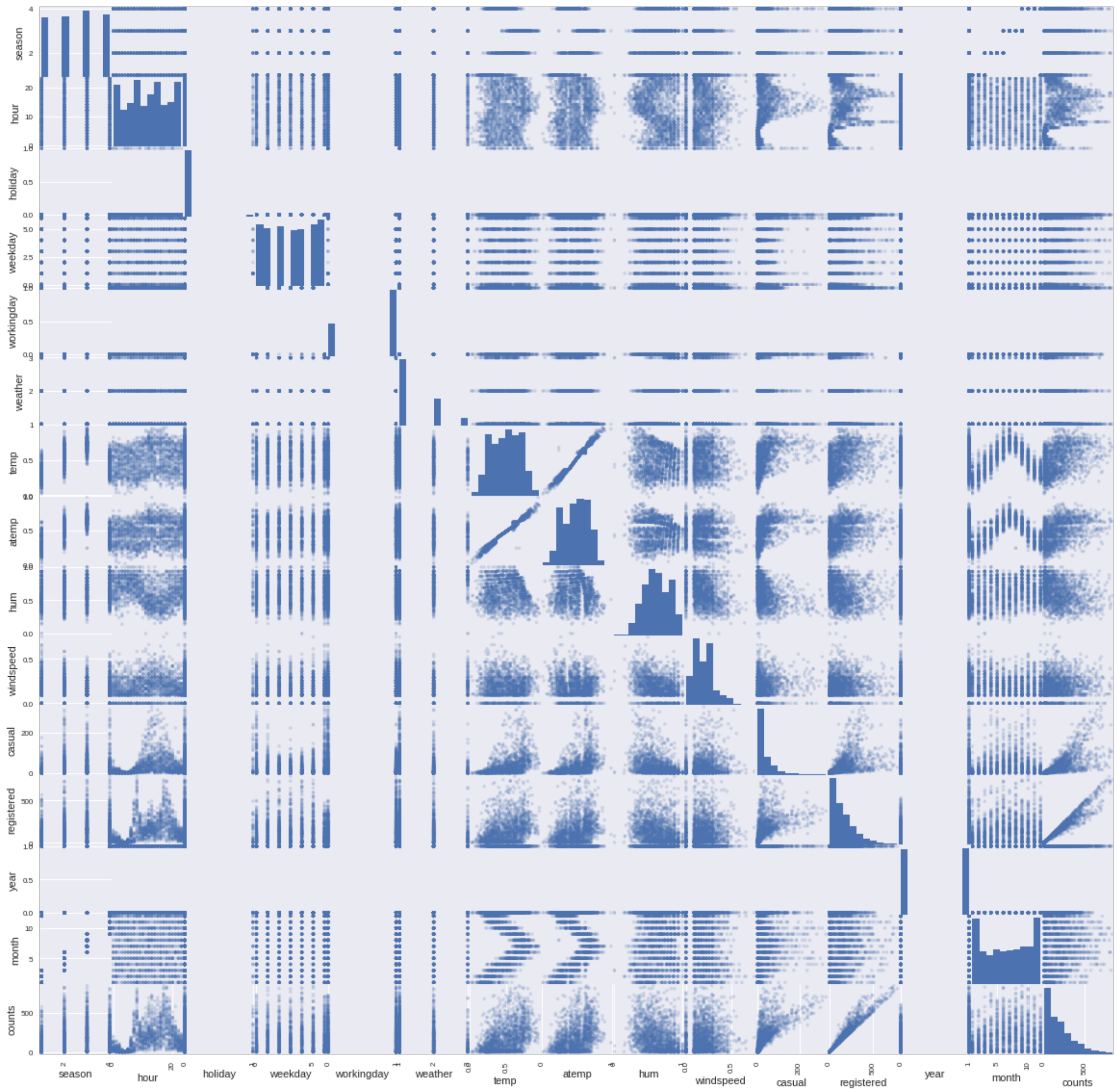
Answers

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]

The following is the scatter plot of all of the features in the data model:

In [70]:
```
bikes_df_5 = bikes_df.sample(frac=0.1)
sc = pd.plotting.scatter_matrix(bikes_df_5, alpha=0.2, figsize=(20, 20
))
plt.suptitle("inter-dependencies among all predictors", fontsize=14)
plt.show()
```

inter-dependencies among all predictors



Comments about the scatter plot matrix

From the scatter plot matrix, we can see two kinds of distributions:

1. Histogram for each of the numerical predictors, on the diagonal:
2. Scatter plots for pairs of variables in the upper and lower triangles of the matrix

The histograms confirm our initial intuition about the skedness in the data. That the distibutions are skewed and this will taken into account when imposing a linear model on the dataset.

From the scatter plots, no clear pattern emerges from the pairwise association. We have to look the pair of association individually in order to discern the associations.

The effect of various variable on the response will be shown and discussed in this study.

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.

The goal is of this study is to help predict the hourly demand for rental bikes. As a result, the data model must be summarized at the granularity of the hour as follows:

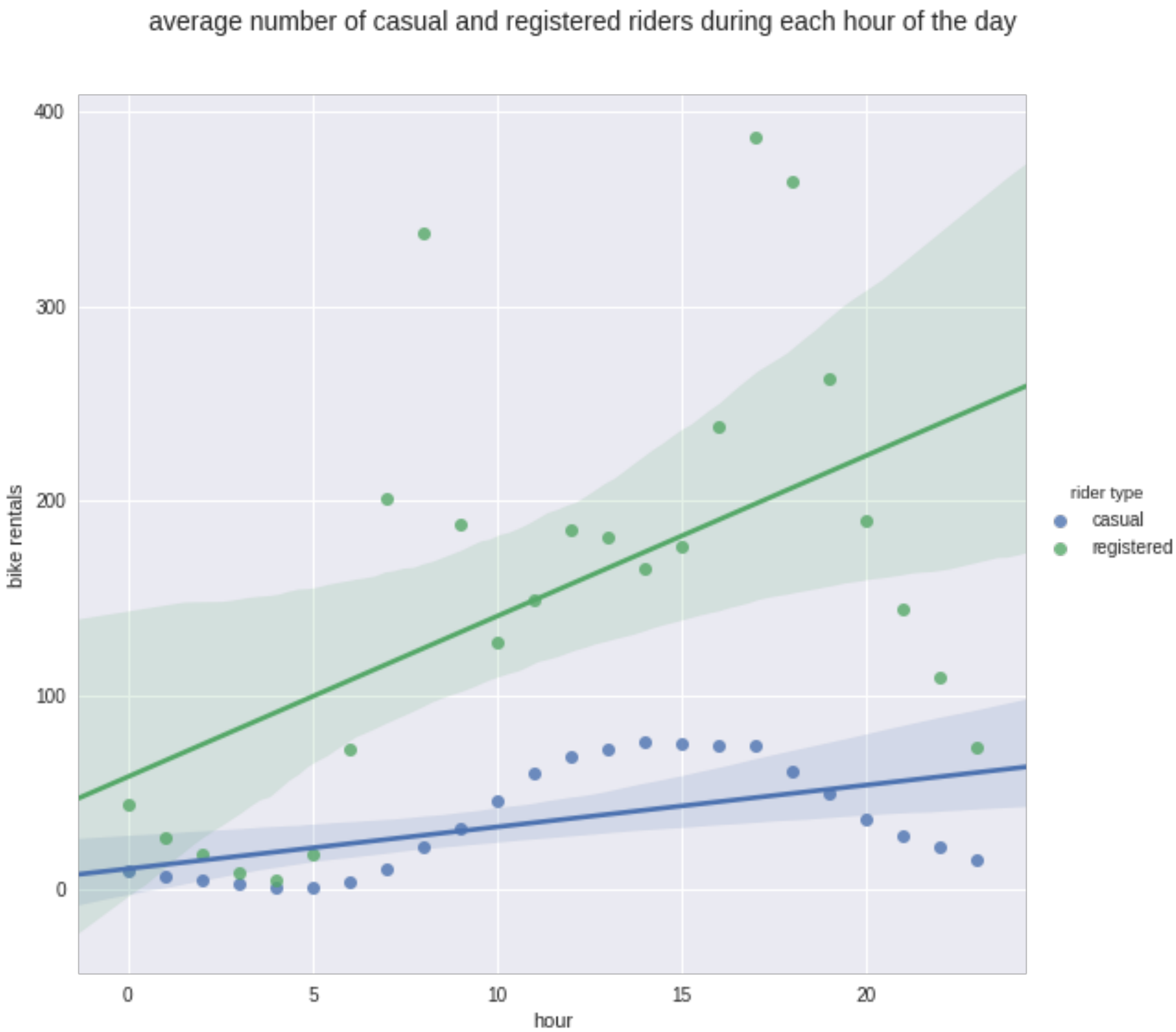
In [71]:
```
hourly_bikes_df = bikes_df.groupby('hour').agg({
    'casual': np.mean,
    'registered': np.mean
})
hourly_bikes_df = hourly_bikes_df.round()
hourly_bikes_df.casual = hourly_bikes_df.casual.astype('int32')
hourly_bikes_df.registered = hourly_bikes_df.registered.astype('int32'
)
hourly_bikes_df.head()
```

Out[71]:

	casual	registered
hour		
0	10	44
1	7	27
2	5	18
3	3	9
4	1	5

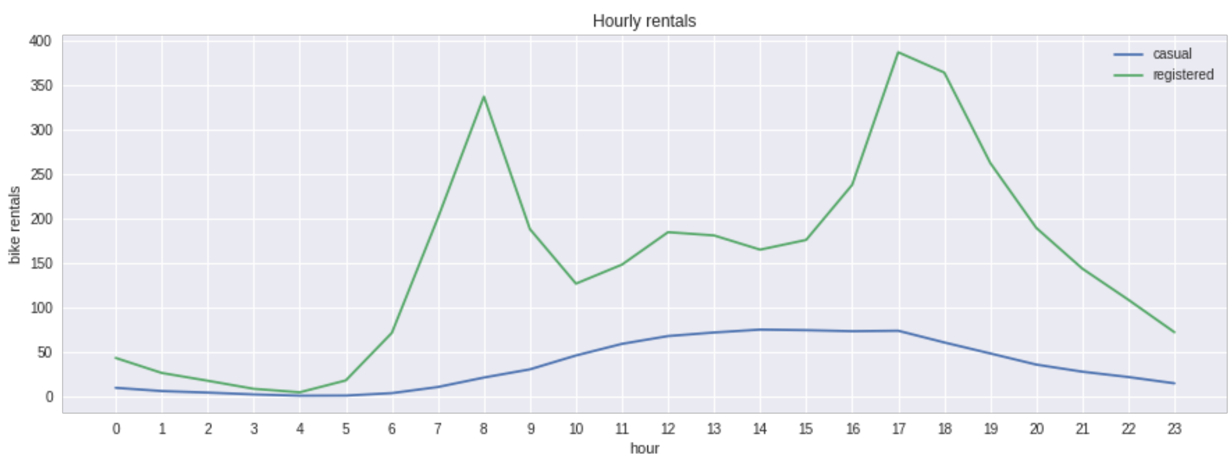
The following figures show the average variation of the demand for rental bikes for a cycle of 24 hours in Washington DC from 2011 and 2012.

In [72]:
```
tidy_hourly_bikes_df = (
    hourly_bikes_df.stack() # pull the columns into row variables
      .to_frame() # convert the resulting Series to a DataFrame
      .reset_index() # pull the resulting MultiIndex into the columns
      .rename(columns={0: 'bike rentals', 'level_1': 'rider type'}) #
rename the unnamed column
)

lm = sns.lmplot(x = 'hour', y = 'bike rentals', hue = 'rider type', da
ta = tidy_hourly_bikes_df, size = 8)

lm.fig.subplots_adjust(top=0.9)
a = lm.fig.suptitle("average number of casual and registered riders du
ring each hour of the day", fontsize=14)
```



Even better than mere linear trends, the following figure tells the story of the customers, their habit, and their needs:

In [73]:
```
hourly_bikes_df = bikes_df.groupby('hour').agg({'casual': np.mean,'reg
istered': np.mean})
def plot_casual_vs_registered(data):
    nb_plots = len(data)
    fig, ax = plt.subplots(1, nb_plots, figsize=(15,5))
    for i, d in enumerate(data):
        if nb_plots > 1:
            f = ax[i]
        else:
            f = ax
        f.set_xlabel('hour')
        f.set_ylabel('bike rentals')
        f.set_xticks(range(24))
        f.set_title(d['title'])
        d['frame'].plot(ax=f)


plot_casual_vs_registered([{'title':'Hourly rentals', 'frame': hourly_
bikes_df}])
#plt.savefig('https://raw.githubusercontent.com/michelkana/cs109a/mast
er/fig1.png', bbox_inches='tight')
```



As it can be seen, the blue line plots the habits, the needs, and the demand of casual bike renters. On average, these bikers ask for bikes mostly between 11:am and 5:pm. Their demand reaches it lowest points between 4 and 5 in the morning, when it is essentially zero. It should also be noted that the qunatity demanded reaches a maximum of about between 60 and 80 rides from 12 in the afternoon to 5 in the afternoon, when it begins to ebb. This pattern is consistent with the pattern of customers who do not strong commitments, perharps, they are tourists or people on vacation.

The green line plots the story of the committed professional. After all, they are registered bikers whith a demand for bike that spkies twice daily: first spike is around 8: am and the second is around 5: pm. This pattern suggests that these customers are office workers who must get to work by 8 in the morning and leave by 5 in the afternoon.

The degree to which date and time of the day, day of the week, season, temperature and other atmospheric conditions affect the demand for rental bikes remains to be seen.

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

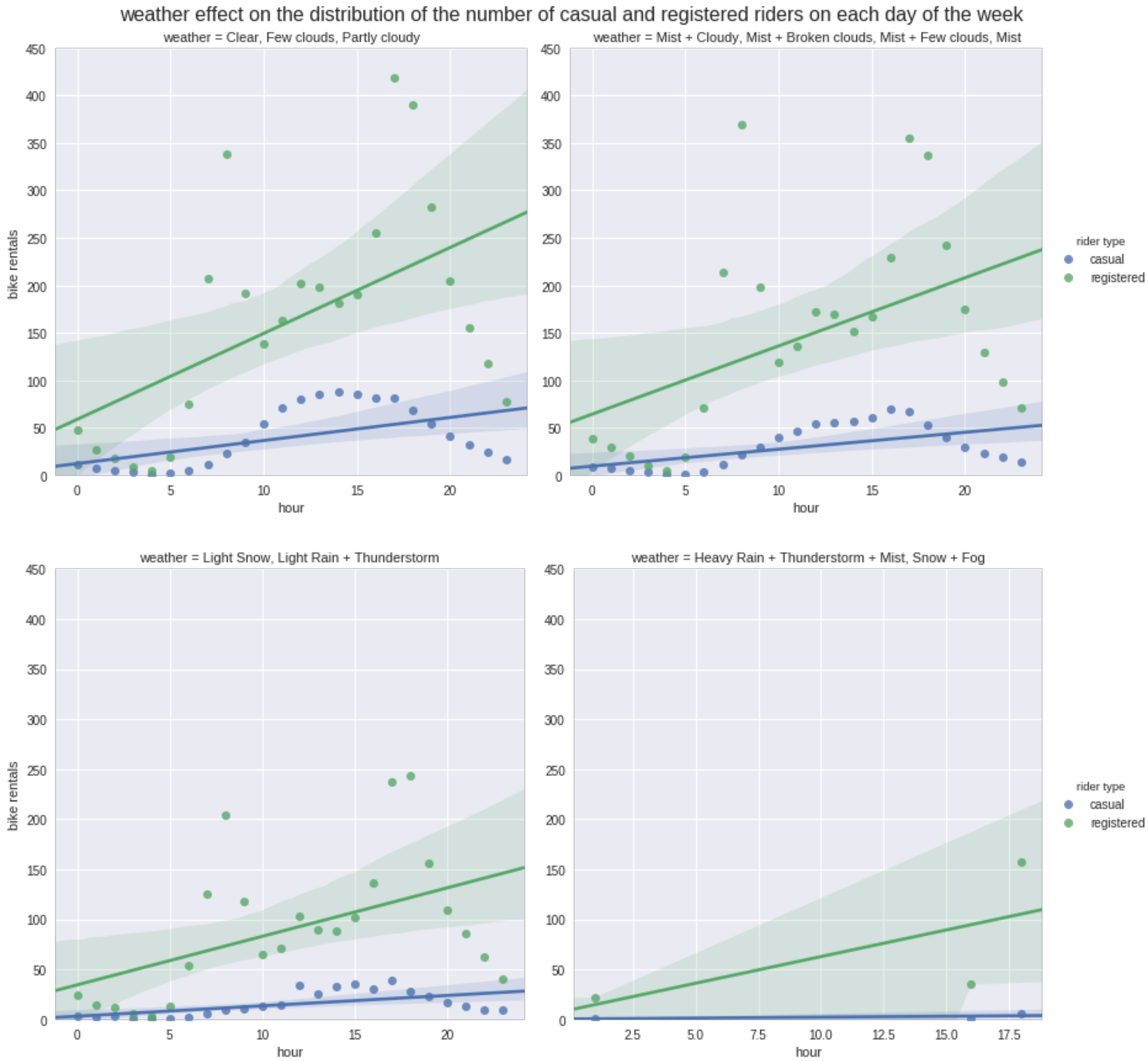
In [74]:
```
# Summaring the data by hour and by weather as follows:
weather_hourly_bikes_df = bikes_df.groupby(['hour', 'weather']).agg({
    'casual': np.mean,
    'registered': np.mean
})
weather_hourly_bikes_df = weather_hourly_bikes_df.round()
weather_hourly_bikes_df.casual = weather_hourly_bikes_df.casual.astype
('int32')
weather_hourly_bikes_df.registered = weather_hourly_bikes_df.registere
d.astype('int32')
weather_hourly_bikes_df.head()
```

Out[74]:

		casual	registered
hour	**weather**		
0	**1**	11	48
	2	9	39
	3	4	25
1	**1**	7	27
	2	7	29

In [75]:
```
# Computing the graphs
tidy_weather_hourly_bikes_df = (
    weather_hourly_bikes_df.stack() # pull the columns into row variab
les
      .to_frame() # convert the resulting Series to a DataFrame
      .reset_index() # pull the resulting MultiIndex into the columns
      .rename(columns={0: 'bike rentals', 'level_2': 'rider type', 'we
ather': 'weather_code'}) # rename the unnamed column
)
tidy_weather_hourly_bikes_df['weather'] = tidy_weather_hourly_bikes_df
.weather_code
tidy_weather_hourly_bikes_df.weather = tidy_weather_hourly_bikes_df.we
ather.replace(1, 'Clear, Few clouds, Partly cloudy')
tidy_weather_hourly_bikes_df.weather = tidy_weather_hourly_bikes_df.we
ather.replace(2, 'Mist + Cloudy, Mist + Broken clouds, Mist + Few clou
ds, Mist')
tidy_weather_hourly_bikes_df.weather = tidy_weather_hourly_bikes_df.we
ather.replace(3, 'Light Snow, Light Rain + Thunderstorm')
tidy_weather_hourly_bikes_df.weather = tidy_weather_hourly_bikes_df.we
ather.replace(4, 'Heavy Rain + Thunderstorm + Mist, Snow + Fog')

tidy_weather_hourly_bikes_df
g = sns.lmplot(x = 'hour', y = 'bike rentals', hue = 'rider type', col
= 'weather',
           data = tidy_weather_hourly_bikes_df[(tidy_weather_hourly_bi
kes_df.weather_code==1) |
                                               (tidy_weather_hourly_bi
kes_df.weather_code==2)],
           size = 6, sharex=False, sharey=False)
g.set(ylim=(0, 450))
g.fig.subplots_adjust(top=0.9)
g.fig.suptitle('weather effect on the distribution of the number of ca
sual and registered riders on each day of the week', fontsize=16)
g = sns.lmplot(x = 'hour', y = 'bike rentals', hue = 'rider type', col
= 'weather',
           data = tidy_weather_hourly_bikes_df[(tidy_weather_hourly_bi
kes_df.weather_code==3) |
                                               (tidy_weather_hourly_bi
kes_df.weather_code==4)],
           size = 6, sharex=False, sharey=False)
g.set(ylim=(0, 450))
```

Out[75]: <seaborn.axisgrid.FacetGrid at 0x7f9a491183c8>



The above figures that the there is a persistent difference in the pattern of demand for rental bikes between registered and casual bikers. However, the patterns are similar in the following weather conditions:

- Light snow or rain or thunderstorm
- Clear, few clouds, or partly cloudy
- Mist or cloudy, mist or broken clouds, mist + few clouds, or simply mist

The demand for rental bike is almost non-existent when the weather condition is

- Heavy rain, thunderstorm, mist, snow, or fog

The demand from registered customers should be taken as suspect here. The trend is misleading and has come about as a results of some outliers. In particular, the outlier occuring aorubd 5:pm could well be from registered customers wanting to get back home ahead of nasty weather conditions.

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`.

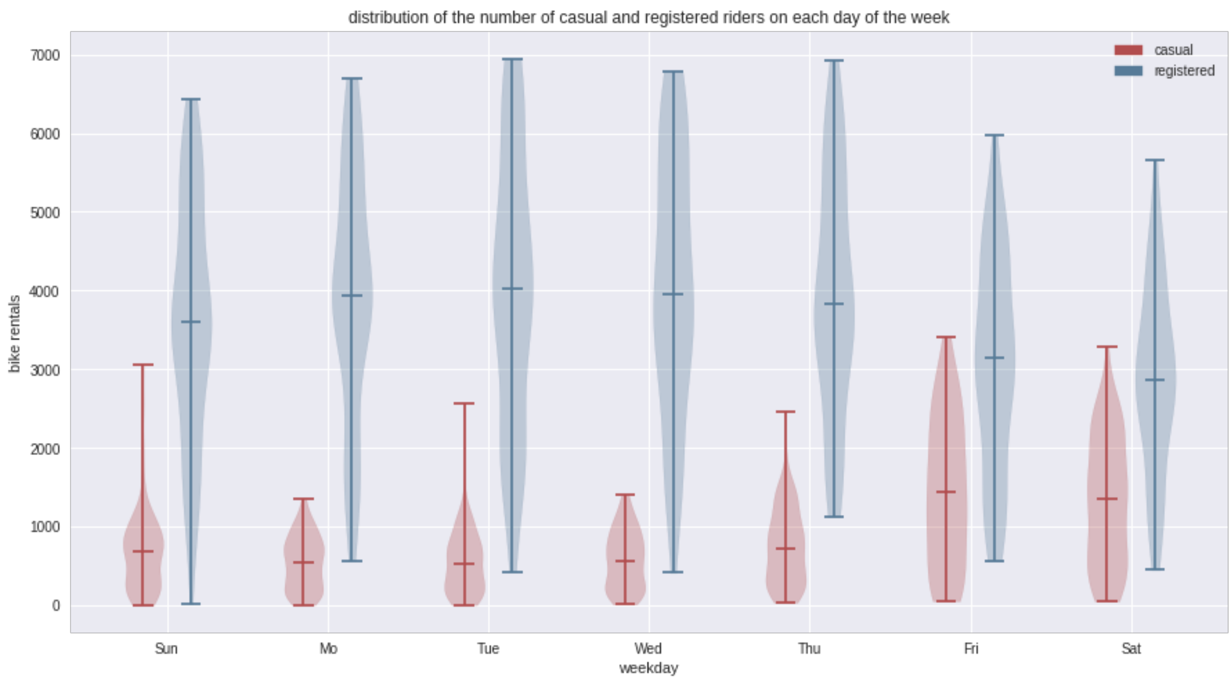
Make a plot showing the *distribution* of the number of casual and registered riders on each day of the week.

In [0]:
```
bikes_by_day = bikes_df.groupby(['dteday']).agg({
        'weekday': np.max,
        'weather': np.max,
        'season': np.max,
        'temp': np.mean,
        'atemp': np.mean,
        'windspeed': np.mean,
        'hum': np.mean,
        'casual': np.sum,
        'registered': np.sum,
    }).reset_index()


bikes_by_day.dteday = bikes_by_day.dteday.astype(str) + " 12:00"
bikes_by_day.dteday = pd.to_datetime(bikes_by_day.dteday, format='%Y-%m-%d %H:%M')
bikes_by_day['weekday'] = bikes_by_day.dteday.dt.weekday
bikes_by_day['counts'] = bikes_by_day.casual + bikes_by_day.registered
```

In [0]:
```
bikes_by_weekday = bikes_by_day.groupby(['weekday']).agg({
    'casual': np.sum,
    'registered': np.sum
})
```

In [78]:
```
fig, ax = plt.subplots(1, 1, figsize=(15,8))
weekdays = [0, 1, 2, 3, 4, 5, 6]
rider_types = ['casual', 'registered']
positions_array = np.arange(len(weekdays))
colors = sns.color_palette("Set1", n_colors=len(weekdays), desat=.5)
sns.palplot(colors)
fake_handles = []

for i, rider_type in enumerate(rider_types):
    offset = .15 * (-1 if i == 0 else 1)
    violin = ax.violinplot([
        bikes_by_day[bikes_by_day['weekday'] == wd][rider_type].values
        for wd in weekdays
    ], positions=positions_array + offset, widths=.25, showmedians=Tru
e, showextrema=True)

    # Set the color
    color = colors[i]
    for part_name, part in violin.items():
        if part_name == 'bodies':
            for body in violin['bodies']:
                body.set_color(color)
        else:
            part.set_color(color)
    fake_handles.append(mpatches.Patch(color=color))

ax.legend(fake_handles, rider_types)
ax.set_xticks(positions_array, weekdays)
ax.set_xticklabels(['', 'Sun', 'Mo', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'
])
ax.set_xlabel("weekday")
ax.set_ylabel("bike rentals");
ax.set_title('distribution of the number of casual and registered ride
rs on each day of the week')
```

```
Out[78]: Text(0.5,1,'distribution of the number of casual and registered ride
         rs on each day of the week')
```

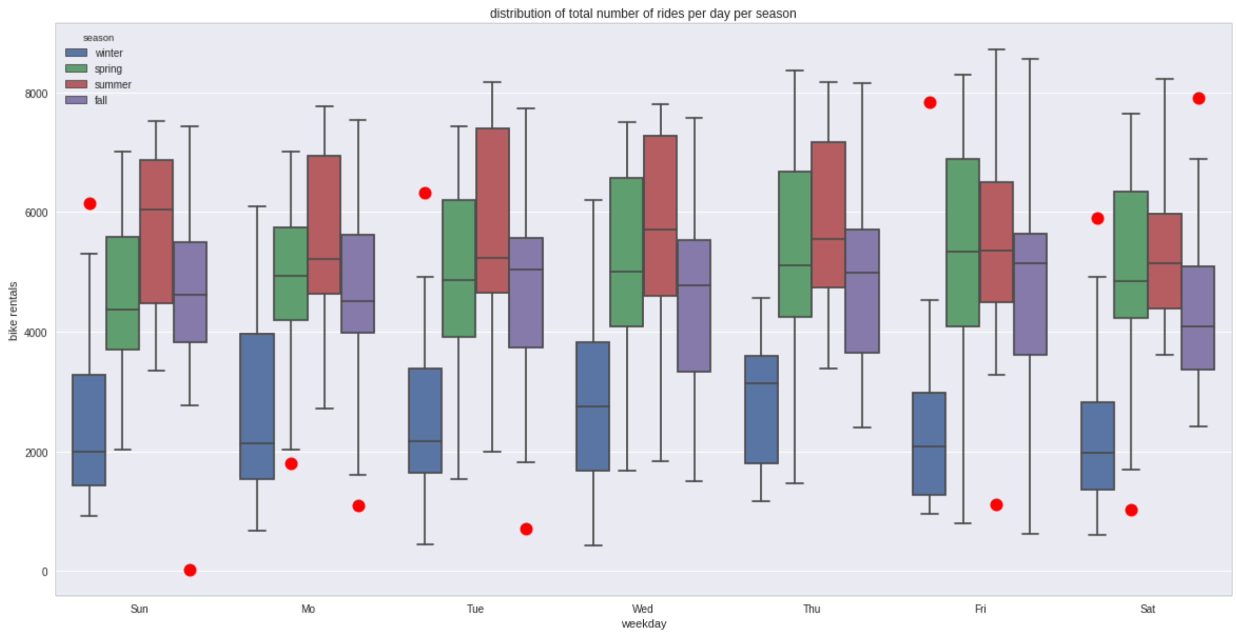


Here again, there is a difference in the distribution by biker type based on the day of the week. For registers customers, the demand is nearly uniform between from Monday to Thursday and is slightly lower on fridays, perharps because of some office workers from from home on Fridays.

For casual bikers, the demand picks on the weekends. This pattern also strengthen our hunch that the customers are vacationers or tourists.

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 occurence, an error in the data collection, or a significant event (an online search of those date(s) might help).

In [79]:
```
bikes_by_day.season = bikes_by_day.season.replace(1, 'winter').replace
(2, 'spring').replace(3, 'summer').replace(4, 'fall')
fig, ax = plt.subplots(figsize=(20,10))
flierprops = dict(marker='o', markerfacecolor='r', markersize=12,
                  linestyle='none', markeredgecolor='g')
sns.boxplot(x="weekday", y="counts", hue="season", data=bikes_by_day,
flierprops=flierprops, ax=ax)
ax.set_title('distribution of total number of rides per day per season
')
ax.set_xticklabels(['Sun', 'Mo', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'])
ax.set_ylabel('bike rentals')
plt.show()
```



The above figure shows the outliers as read dots. The following are some additional research showing why these outliers are observed.

Explanation about the outliers

The following outliers were found by visual inspection of the boxplot above.

In [80]:
```
# your code here
outlier1 = bikes_by_day.loc[(bikes_by_day.weekday==0) & (bikes_by_day.
season=='winter') & (bikes_by_day.counts>6000)]
outlier2 = bikes_by_day.loc[(bikes_by_day.weekday==0) & (bikes_by_day.
season=='fall') & (bikes_by_day.counts<100)]
outlier3 = bikes_by_day.loc[(bikes_by_day.weekday==1) & (bikes_by_day.
season=='spring') & (bikes_by_day.counts<2000)]
outlier4 = bikes_by_day.loc[(bikes_by_day.weekday==1) & (bikes_by_day.
season=='fall') & (bikes_by_day.counts<1100)]
outlier5 = bikes_by_day.loc[(bikes_by_day.weekday==2) & (bikes_by_day.
season=='winter') & (bikes_by_day.counts>6000)]
outlier6 = bikes_by_day.loc[(bikes_by_day.weekday==2) & (bikes_by_day.
season=='fall') & (bikes_by_day.counts<1000)]
outlier7 = bikes_by_day.loc[(bikes_by_day.weekday==5) & (bikes_by_day.
season=='winter') & (bikes_by_day.counts>6000)]
outlier8 = bikes_by_day.loc[(bikes_by_day.weekday==5) & (bikes_by_day.
season=='summer') & (bikes_by_day.counts<2000)]
outlier9 = bikes_by_day.loc[(bikes_by_day.weekday==6) & (bikes_by_day.
season=='winter') & (bikes_by_day.counts>5000)]
outlier10 = bikes_by_day.loc[(bikes_by_day.weekday==6) & (bikes_by_day
.season=='spring') & (bikes_by_day.counts<1500)]
outlier11 = bikes_by_day.loc[(bikes_by_day.weekday==6) & (bikes_by_day
.season=='fall') & (bikes_by_day.counts>6500)]
outlier1.append(outlier2).append(outlier3).append(outlier4).append(out
lier5).append(outlier6).append(outlier7).append(outlier8).append(outli
er9).append(outlier10).append(outlier11).sort_values(by=['dteday'])
```

Out[80]:

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	ca
94	2011-04-05 12:00:00	1	3	spring	0.414167	0.398350	0.388067	0.642083	1
238	2011-08-27 12:00:00	5	3	summer	0.680000	0.635556	0.375617	0.850000	2
340	2011-12-07 12:00:00	2	3	fall	0.410000	0.400246	0.266175	0.970417	5
438	2012-03-14 12:00:00	2	2	winter	0.572500	0.548617	0.115063	0.507083	9
441	2012-03-17 12:00:00	5	2	winter	0.514167	0.505046	0.110704	0.755833	3

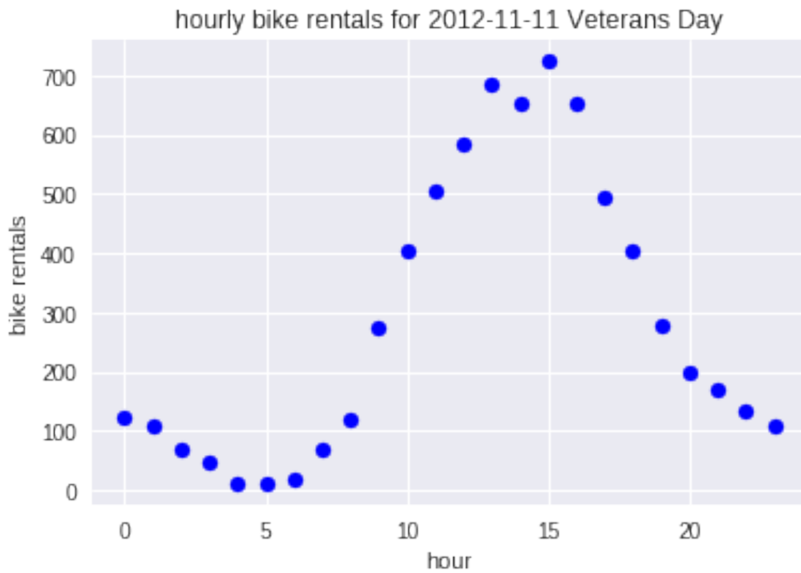
442	2012-03-18 12:00:00	6	3	winter	0.472500	0.464000	0.126883	0.810000	2
443	2012-03-19 12:00:00	0	2	winter	0.545000	0.532821	0.162317	0.728750	9
477	2012-04-22 12:00:00	6	3	spring	0.396667	0.389504	0.344546	0.835417	1
631	2012-09-23 12:00:00	6	2	fall	0.529167	0.518933	0.223258	0.467083	2
638	2012-09-30 12:00:00	6	3	fall	0.526667	0.517663	0.134958	0.583333	2
652	2012-10-14 12:00:00	6	2	fall	0.521667	0.508204	0.278612	0.640417	2
659	2012-10-21 12:00:00	6	1	fall	0.464167	0.456429	0.166054	0.510000	2
667	2012-10-29 12:00:00	0	3	fall	0.440000	0.439400	0.358200	0.880000	2
668	2012-10-30 12:00:00	1	3	fall	0.318182	0.309909	0.213009	0.825455	8
680	2012-11-11 12:00:00	6	1	fall	0.420833	0.421713	0.127500	0.659167	2

Investigation of the outliers

An online search finds 2012-11-11 as to be the Veterans Day. Probably many of veterans checkout bikes for driving through Washington DC. This would explain the unexpected high count of bikes rental.

In [81]:
```
# your code here
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-11-11'].counts
, label='2012-11-11', color='b')
plt.title('hourly bike rentals for 2012-11-11 Veterans Day')
plt.xlabel('hour')
plt.ylabel('bike rentals')
```

Out[81]:
```
Text(0,0.5,'bike rentals')
```

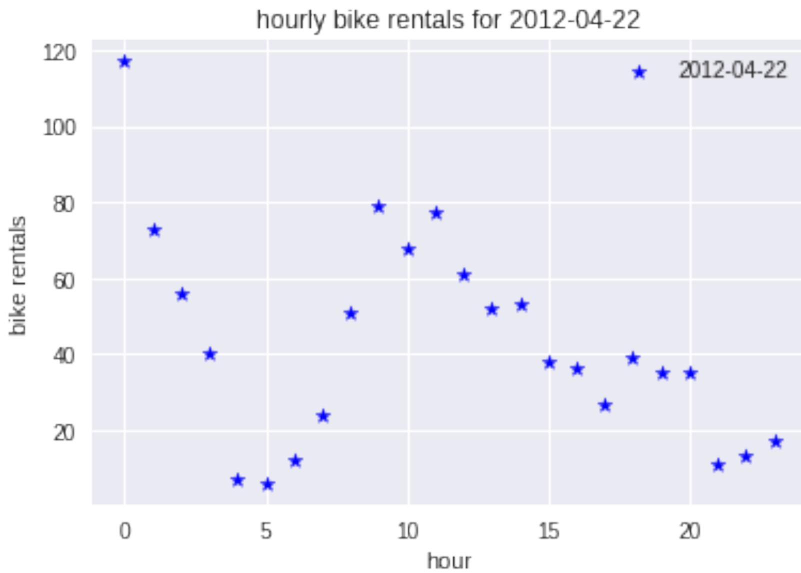


Weather conditions and the occurence of outliers

The weather on 2012-04-22 in Washington DC (Thunderstorm) probably explains the sudden drop in bike rentals.

In [82]:
```
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-04-22'].counts
, label='2012-04-22', color='b', marker='*')
plt.title('hourly bike rentals for 2012-04-22')
plt.xlabel('hour')
plt.ylabel('bike rentals')
plt.legend()
```

Out[82]: <matplotlib.legend.Legend at 0x7f9a4bebaa58>

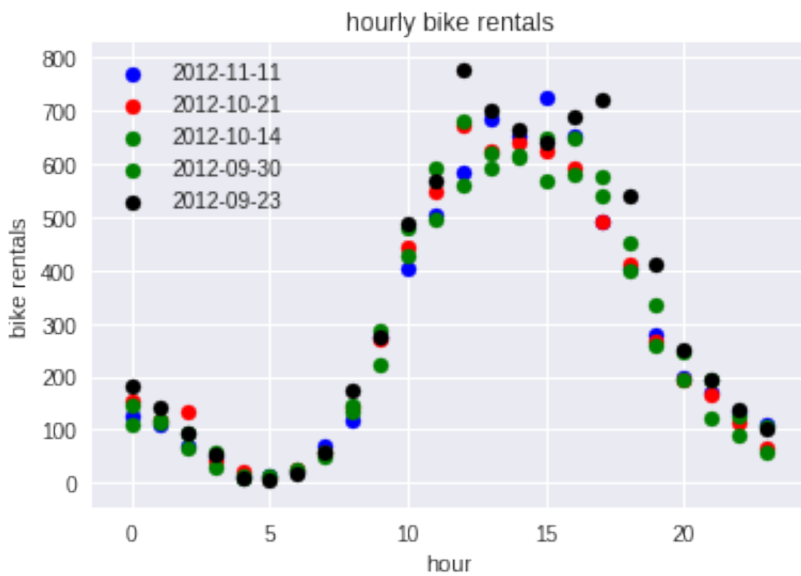


Duplicated data and outliers

The plot of hourly bike rentals on the remaining dates shows a strong correlation to the data from 2012-11-11, this indicates a measurement abnormality.

In [83]:
```
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-11-11'].counts
, label='2012-11-11', color='b')
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-10-21'].counts
, label='2012-10-21', color='r')
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-10-14'].counts
, label='2012-10-14', color='g')
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-09-30'].counts
, label='2012-09-30', color='g')
plt.scatter(range(24),bikes_df[bikes_df.dteday == '2012-09-23'].counts
, label='2012-09-23', color='black')
plt.title('hourly bike rentals')
plt.xlabel('hour')
plt.ylabel('bike rentals')
plt.legend()
```

Out[83]: <matplotlib.legend.Legend at 0x7f9a4ceb5390>



Italicized text Missing records

The data for 2012-10-30 and 201-10-29 show records missing for some bunch of hours, indicating a measurment abnormality.

your answer here

In [84]:
```
bikes_df[(bikes_df.dteday == '2012-10-30') | (bikes_df.dteday == '2012
-10-29')]
```

Out[84]:

	dteday	season	hour	holiday	weekday	workingday	weather	temp	atemp	hu
15883	2012-10-29	4	0	0	1	1	3	0.44	0.4394	0.
15884	2012-10-30	4	13	0	2	1	3	0.30	0.2727	0.
15885	2012-10-30	4	14	0	2	1	3	0.30	0.2727	0.
15886	2012-10-30	4	15	0	2	1	3	0.30	0.2879	0.
15887	2012-10-30	4	16	0	2	1	3	0.30	0.2879	0.
15888	2012-10-30	4	17	0	2	1	3	0.30	0.2879	0.
15889	2012-10-30	4	18	0	2	1	3	0.30	0.3030	0.
15890	2012-10-30	4	19	0	2	1	2	0.50	0.4848	0.
15891	2012-10-30	4	20	0	2	1	2	0.30	0.2879	0.
15892	2012-10-30	4	21	0	2	1	2	0.30	0.3182	0.
15893	2012-10-30	4	22	0	2	1	1	0.30	0.3030	0.
15894	2012-10-30	4	23	0	2	1	1	0.30	0.3030	0.

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 he 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 [0]:
```
# your code here
bikes_df_encoded = pd.get_dummies(bikes_df, columns=['season', 'month', 'weekday', 'weather'], drop_first=True)
```

In [0]:
```
bikes_df_encoded = bikes_df_encoded.rename(columns={'season_2':'spring', 'season_3':'summer', 'season_4':'fall',
                                                    'month_2':'February', 'month_3':'March', 'month_4':'April', 'month_5':'May', 'month_6':'June', 'month_7':'July', 'month_8':'August', 'month_9':'September', 'month_10':'October', 'month_11':'November', 'month_12':'December',
                                                    'weekday_1':'Mon', 'weekday_2':'Tue','weekday_3':'Wed', 'weekday_4':'Thu','weekday_5':'Fri', 'weekday_6':'Sat'})
```

In [27]:
```
#your code here
bikes_df_encoded.head()
```

Out[27]:

	dteday	hour	holiday	workingday	temp	atemp	hum	windspeed	casual	registere
0	2011-01-01	0	0	0	0.24	0.2879	0.81	0.0	3	13
1	2011-01-01	1	0	0	0.22	0.2727	0.80	0.0	8	32
2	2011-01-01	2	0	0	0.22	0.2727	0.80	0.0	5	27
3	2011-01-01	3	0	0	0.24	0.2879	0.75	0.0	3	10
4	2011-01-01	4	0	0	0.24	0.2879	0.75	0.0	0	1

5 rows × 35 columns

In [87]:
```
#your code here
bikes_df_encoded.columns
```

Out[87]:
```
Index(['dteday', 'hour', 'holiday', 'workingday', 'temp', 'atemp', '
hum', 'windspeed', 'casual', 'registered', 'year', 'counts', 'spring
', 'summer', 'fall', 'February', 'March', 'April', 'May', 'June', 'J
uly', 'August', 'September', 'October', 'November', 'December', 'Mon
', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'weather_2', 'weather_3', 'wea
ther_4'], dtype='object')
```

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.

Explanation

We use sklearn train_test_split function with the parameter stratify. In order to have an equal representation of each of the 24 months, we select the year and the binary attributes representing the months.

- We catch exceptions in case of missing data.
- We use tthe standard 20% for test data set.

In [0]:
```
# your code here
months_list = ['year', 'February', 'March', 'April', 'May', 'June', 'J
uly', 'August', 'September', 'October', 'November', 'December']
try:
    # Stratify doesn't work if a value is missing
    train_data, test_data = train_test_split(bikes_df_encoded, test_si
ze = 0.2, stratify=bikes_df_encoded[months_list])
except:
    # Drop missing lines
    print("Lines with missing values dropped")
    bikes_df_encoded = bikes_df_encoded.dropna(subset=months_list)
    train_data, test_data = train_test_split(bikes_df_encoded, test_si
ze = 0.2, stratify=bikes_df_encoded[months_list])
```

Explanation

The stratification produces a test set where all 24 months are represented equally, with 144 ± 4 data points. Similarly we have 579 ± 17 data points per month in the training set.

In [89]:
```
# test data - monthly distribution after stratification
test_data_monthly_count = test_data.groupby(months_list).aggregate(['c
ount'])['hour']
print("Mean: ", np.mean(test_data_monthly_count), "standard deviation:
", np.std(test_data_monthly_count))
# percentage of test data over each month
test_data_monthly_count/test_data.shape[0]
```

```
Mean:  count    144.833333
dtype: float64 standard deviation:  count    4.588633
dtype: float64
```

Out[89]:

year	February	March	April	May	June	July	August	September	October	Novembe
0	0	0	0	0	0	0	0	0	0	0
										1
									1	0
								1	0	0
							1	0	0	0
						1	0	0	0	0
					1	0	0	0	0	0
				1	0	0	0	0	0	0
			1	0	0	0	0	0	0	0
		1	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
										1
									1	0
								1	0	0
							1	0	0	0
						1	0	0	0	0
					1	0	0	0	0	0
				1	0	0	0	0	0	0
			1	0	0	0	0	0	0	0
		1	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0

In [90]:

```
# train data - monthly distribution after stratification
train_data_monthly_count = train_data.groupby(months_list).aggregate([
'count'])['hour']
print("Mean: ", np.mean(train_data_monthly_count), "standard deviation
: ", np.std(train_data_monthly_count))
# percentage of train data over each month
train_data_monthly_count/train_data.shape[0]
```

```
Mean:  count    579.291667
dtype: float64 standard deviation:  count    17.903629
dtype: float64
```

Out[90]:

year	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**Novembe**
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
										1
									1	**0**
								1	**0**	**0**
							1	**0**	**0**	**0**
						1	**0**	**0**	**0**	**0**
					1	**0**	**0**	**0**	**0**	**0**
				1	**0**	**0**	**0**	**0**	**0**	**0**
			1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
										1
									1	**0**
								1	**0**	**0**
							1	**0**	**0**	**0**
						1	**0**	**0**	**0**	**0**
					1	**0**	**0**	**0**	**0**	**0**
				1	**0**	**0**	**0**	**0**	**0**	**0**
			1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
		1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
	1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

In [91]:
```
# orginal data - monthly distribution before stratification
bikes_df.groupby(['year', 'month']).aggregate(['count'])['hour']/bikes
_df.shape[0]
```

Out[91]:

		count
year	**month**	
0	**1**	0.039588
	2	0.037344
	3	0.042005
	4	0.041372
	5	0.042810
	6	0.041429
	7	0.042810
	8	0.042062
	9	0.041257
	10	0.042753
	11	0.041372
	12	0.042638
1	**1**	0.042638
	2	0.039818
	3	0.042753
	4	0.041314
	5	0.042810
	6	0.041429
	7	0.042810
	8	0.042810
	9	0.041429
	10	0.040739
	11	0.041314
	12	0.042695

Additional notes on our stratistification procedure.

By stratifying the data, we seek to ensure that the proportional distribution of the data is maintain first among the year 2011 and 2012 and also reflect the monthly variations.

The above extract step was take to ensure that the relative frequencies by year and by month are maintain in the test and the train 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 he 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 [0]:
```
# read CSV into dataframe
BSS_train = pd.read_csv('https://raw.githubusercontent.com/michelkana/
cs109a/master/BSS_train.csv')
BSS_test = pd.read_csv('https://raw.githubusercontent.com/michelkana/c
s109a/master/BSS_test.csv')
```

In [0]:
```
# remove 'casual' and 'registered' as they make predicting 'counts' tr
ivial
# remove 'Unnamed: 0' because it is not needed for regression
select_columns = ['dteday', 'casual', 'registered', 'Unnamed: 0']
BSS_train = BSS_train.drop(columns = select_columns)
BSS_test = BSS_test.drop(columns = select_columns)
```

In [95]:
```
# check the data frame
BSS_test.columns
```

Out[95]:
```
Index(['hour', 'holiday', 'year', 'workingday', 'temp', 'atemp', 'hu
m', 'windspeed', 'counts', 'spring', 'summer', 'fall', 'Feb', 'Mar',
'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon
', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm'], dt
ype='object')
```

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 *multicolinearity*, where one or more predictors capture the same information as existing predictors. Why can multicolinearity 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?

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 [0]:
```
# create the predictors list
predictors = list(BSS_train.columns)
predictors.remove('counts')
```

In [0]:
```
# prepare the design matrix with all predictors
design_mat_train = BSS_train[predictors]
design_mat_train = sm.add_constant(design_mat_train)
design_mat_test = BSS_test[predictors]
design_mat_test = sm.add_constant(design_mat_test)
```

In [98]:
```
# fit a linear regression model on train set
fitted_model_train = OLS(BSS_train.counts, design_mat_train, hasconst=
True).fit()
fitted_model_train.summary()
```

Out[98]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.407
Model:	OLS	**Adj. R-squared:**	0.405
Method:	Least Squares	**F-statistic:**	316.8
Date:	Thu, 04 Oct 2018	**Prob (F-statistic):**	0.00
Time:	02:14:24	**Log-Likelihood:**	-88306.
No. Observations:	13903	**AIC:**	1.767e+05
Df Residuals:	13872	**BIC:**	1.769e+05
Df Model:	30		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-21.0830	8.641	-2.440	0.015	-38.020	-4.146
hour	7.2214	0.184	39.144	0.000	6.860	7.583
holiday	-18.0958	6.597	-2.743	0.006	-31.027	-5.165
year	76.3519	2.380	32.084	0.000	71.687	81.017
workingday	11.3178	2.751	4.114	0.000	5.926	16.710
temp	333.2482	44.162	7.546	0.000	246.684	419.812
atemp	74.6312	46.207	1.615	0.106	-15.940	165.202
hum	-205.4959	7.801	-26.343	0.000	-220.786	-190.205
windspeed	22.5168	10.753	2.094	0.036	1.439	43.595
spring	43.1541	7.417	5.818	0.000	28.615	57.693
summer	29.5426	8.773	3.367	0.001	12.346	46.739
fall	68.5953	7.492	9.156	0.000	53.911	83.280
Feb	-7.6430	5.966	-1.281	0.200	-19.336	4.050
Mar	-11.6737	6.665	-1.752	0.080	-24.737	1.390

Apr	-41.5244	9.878	-4.204	0.000	-60.886	-22.163
May	-33.2927	10.543	-3.158	0.002	-53.958	-12.628
Jun	-65.8039	10.716	-6.141	0.000	-86.809	-44.799
Jul	-93.4805	12.086	-7.734	0.000	-117.171	-69.789
Aug	-59.2081	11.832	-5.004	0.000	-82.401	-36.015
Sept	-16.0517	10.575	-1.518	0.129	-36.780	4.676
Oct	-16.1602	9.865	-1.638	0.101	-35.497	3.177
Nov	-25.8732	9.527	-2.716	0.007	-44.547	-7.199
Dec	-10.2043	7.614	-1.340	0.180	-25.128	4.719
Mon	-2.6601	2.978	-0.893	0.372	-8.498	3.177
Tue	-6.1425	3.208	-1.915	0.056	-12.430	0.145
Wed	2.2964	3.183	0.721	0.471	-3.943	8.536
Thu	-3.1611	3.185	-0.993	0.321	-9.404	3.082
Fri	2.8892	3.186	0.907	0.364	-3.355	9.133
Sat	14.9459	4.382	3.411	0.001	6.357	23.535
Cloudy	6.7868	2.900	2.341	0.019	1.103	12.470
Snow	-28.2859	4.819	-5.870	0.000	-37.731	-18.841
Storm	42.3569	98.377	0.431	0.667	-150.475	235.189

Omnibus:	2831.359	**Durbin-Watson:**	0.755
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5657.789
Skew:	1.224	**Prob(JB):**	0.00
Kurtosis:	4.943	**Cond. No.**	6.90e+15

In [0]:

```
# OLS Linear Regression model training predictions
ols_predicted_counts_train = fitted_model_train.predict(design_mat_tra
in)

# OLS Linear Regression model test predictions
ols_predicted_counts_test = fitted_model_train.predict(design_mat_test
)
```

In [100]:
```
# report R2 score

r2_score_train = r2_score(BSS_train[['counts']].values, ols_predicted_
counts_train)
r2_score_test = r2_score(BSS_test[['counts']].values, ols_predicted_co
unts_test)

print("R^2 score for training set: {:.4}".format(r2_score_train))
print("R^2 score for test set: {:.4}".format(r2_score_test))
```

```
R^2 score for training set: 0.4065
R^2 score for test set: 0.4064
```

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.

The following coefficient are statiscally significant at 5%, including the constant term:

In [101]:
```
fitted_model_train.pvalues[fitted_model_train.pvalues<0.05].sort_value
s()
```

Out[101]:
```
hour           0.000000e+00
year          6.205883e-218
hum           2.797780e-149
fall           6.106365e-20
Jul            1.110753e-14
temp           4.767468e-14
Jun            8.447047e-10
Snow           4.454966e-09
spring         6.082058e-09
Aug            5.685359e-07
Apr            2.640964e-05
workingday     3.905740e-05
Sat            6.490550e-04
summer         7.609902e-04
May            1.592599e-03
holiday        6.095043e-03
Nov            6.619949e-03
const          1.470264e-02
Cloudy         1.926802e-02
windspeed      3.628163e-02
dtype: float64
```

The following are the non-signicant terms at the 5% level of significance.

In [102]:
```
# less significant predictors
fitted_model_train.pvalues[fitted_model_train.pvalues>=0.05]
```

Out[102]:
```
atemp    0.106298
Feb      0.200150
Mar      0.079872
Sept     0.129052
Oct      0.101422
Dec      0.180176
Mon      0.371756
Tue      0.055512
Wed      0.470631
Thu      0.320964
Fri      0.364452
Storm    0.666797
dtype: float64
```

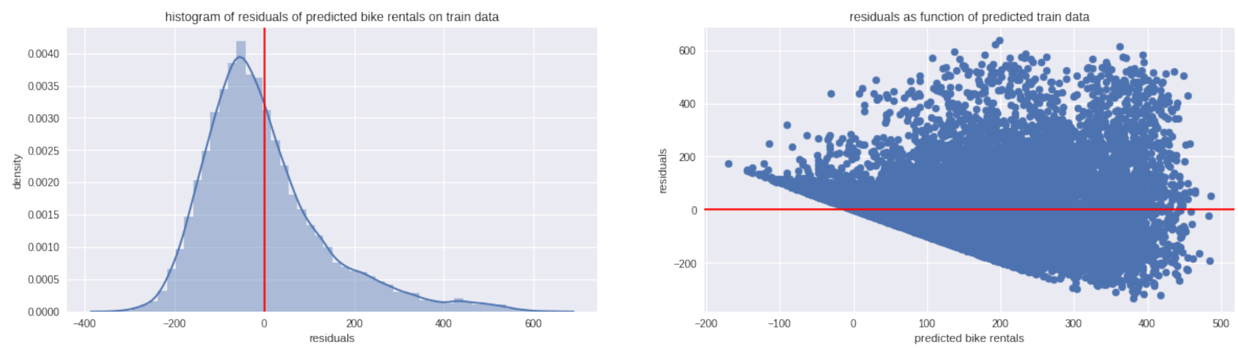
Indeed, as shown above, there tends to significantly be on average 93 bike rental in July than in January. This suspicous result should be expected. After all, the have shown that the distribution of the demand for bike is not normal. The metric used for this output is the p-value with the assumption of normality, which does not hold here. This result should not be taken seriously.

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 [0]:
```
sorted_BSS_train = BSS_train.sort_values(['counts'])
sorted_design_mat_train = sorted_BSS_train[predictors]
sorted_design_mat_train = sm.add_constant(sorted_design_mat_train)

sorted_fitted_model_train = OLS(sorted_BSS_train.counts, sorted_design
_mat_train, hasconst=True).fit()
sorted_ols_predicted_counts_train = sorted_fitted_model_train.predict(
sorted_design_mat_train)
```

In [0]:
```
# calculate residuals
residuals = sorted_BSS_train['counts'] - sorted_ols_predicted_counts_t
rain
```

In [105]:
```
# your code here
fig, ax = plt.subplots(1,2,figsize=(20,5))
sns.distplot(residuals, ax=ax[0])
ax[0].set_title('histogram of residuals of predicted bike rentals on t
rain data')
ax[0].set_xlabel('residuals')
ax[0].set_ylabel('density')
ax[0].axvline(0, 0, 1, color='r')
ax[1].scatter(sorted_ols_predicted_counts_train, residuals)
ax[1].set_title('residuals as function of predicted train data')
ax[1].set_xlabel('predicted bike rentals')
ax[1].set_ylabel('residuals')
ax[1].axhline(0, 0, 1, color='r')
plt.show()
```



Analysis of residuals

The histogram of residuals is not centered at zero. Most residuals are found on the negative side. Since they are not randomly dispersed around the horizontal axis in the second graph, a simple linear regression model might not be appropriate for the data. The residuals are not randomly distributed around zero, therefore the assumption about the normality of the distribution is not correct. There is drift of the variance towards the bottom, on the other hand, residuals show an increasing trend, the assumption of constant variance is not likely to be true.

In summary we noted the follwoing three points about the predicted values as compared to the actual values: :

1. Data points are too far from the average: this give us the solution to our concerns about the coefficients such as -93 for July.
2. The increasing variance with respect to the predictor
3. The shape of the density of the residual hints at the need for a more complex model

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 [0]:
```
# Create 2nd order polynomial terms
X_train_poly = BSS_train.copy()
X_train_poly["temp_squared"] = X_train_poly.temp ** 2
X_train_poly["hour_squared"] = X_train_poly.hour ** 2
X_train_poly["hum_squared"] = X_train_poly.hum ** 2
X_test_poly = BSS_test.copy()
X_test_poly["temp_squared"] = X_test_poly.temp ** 2
X_test_poly["hour_squared"] = X_test_poly.hour ** 2
X_test_poly["hum_squared"] = X_test_poly.hum ** 2
```

In [107]:
```
# prepare predictors
predictors_poly = ['hour', 'holiday', 'workingday', 'temp', 'atemp', '
hum',
       'windspeed', 'spring', 'summer', 'fall', 'Feb', 'Mar', 'Apr',
       'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon',
'Tue',
       'Wed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm', 'temp_sq
uared', 'hour_squared', 'hum_squared']
design_mat_train_poly = X_train_poly[predictors_poly]
design_mat_train_poly = sm.add_constant(design_mat_train_poly)

# fit a multilinear regression model
fitted_model_train_poly = OLS(X_train_poly.counts, design_mat_train_po
ly, hasconst=True).fit()
fitted_model_train_poly.summary()
```

Out[107]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.452
Model:	OLS	**Adj. R-squared:**	0.451
Method:	Least Squares	**F-statistic:**	357.5
Date:	Thu, 04 Oct 2018	**Prob (F-statistic):**	0.00
Time:	02:14:54	**Log-Likelihood:**	-87752.
No. Observations:	13903	**AIC:**	1.756e+05
Df Residuals:	13870	**BIC:**	1.758e+05
Df Model:	32		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-170.0037	14.680	-11.581	0.000	-198.778	-141.229
hour	38.4605	0.693	55.497	0.000	37.102	39.819
holiday	-12.6657	6.346	-1.996	0.046	-25.104	-0.227
workingday	12.1305	2.645	4.587	0.000	6.947	17.314
temp	346.6924	60.780	5.704	0.000	227.554	465.830
atemp	12.3483	45.585	0.271	0.786	-77.005	101.702
hum	76.0973	37.796	2.013	0.044	2.013	150.182
windspeed	-24.1249	10.382	-2.324	0.020	-44.475	-3.775
spring	46.3718	7.129	6.504	0.000	32.397	60.346
summer	34.7578	8.452	4.113	0.000	18.191	51.324
fall	70.3279	7.206	9.759	0.000	56.203	84.453
Feb	-4.7651	5.800	-0.822	0.411	-16.134	6.603
Mar	-4.2284	6.593	-0.641	0.521	-17.152	8.695
Apr	-31.2172	9.662	-3.231	0.001	-50.157	-12.278
May	-27.2006	10.233	-2.658	0.008	-47.259	-7.142
Jun	-50.3522	10.367	-4.857	0.000	-70.673	-30.031
Jul	-77.8098	11.676	-6.664	0.000	-100.697	-54.923
Aug	-49.7040	11.466	-4.335	0.000	-72.178	-27.230
Sept	-10.8007	10.341	-1.044	0.296	-31.071	9.470
Oct	-15.4695	9.690	-1.596	0.110	-34.463	3.524
Nov	-27.7146	9.317	-2.975	0.003	-45.976	-9.453
Dec	-14.2818	7.413	-1.927	0.054	-28.812	0.249
Mon	-2.3373	2.862	-0.817	0.414	-7.947	3.273
Tue	-4.1252	3.083	-1.338	0.181	-10.169	1.918
Wed	2.4544	3.059	0.802	0.422	-3.542	8.451
Thu	-0.5991	3.062	-0.196	0.845	-6.600	5.402
Fri	4.0719	3.063	1.329	0.184	-1.932	10.076
Sat	15.3448	4.212	3.643	0.000	7.088	23.602

Cloudy	-5.0215	2.807	-1.789	0.074	-10.523	0.480
Snow	-42.3194	4.800	-8.817	0.000	-51.728	-32.911
Storm	50.5439	94.558	0.535	0.593	-134.802	235.890
temp_squared	-14.1642	38.051	-0.372	0.710	-88.749	60.420
hour_squared	-1.3254	0.028	-47.160	0.000	-1.380	-1.270
hum_squared	-184.9746	30.250	-6.115	0.000	-244.268	-125.681

Omnibus:	3521.050	**Durbin-Watson:**	0.807
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8613.821
Skew:	1.401	**Prob(JB):**	0.00
Kurtosis:	5.649	**Cond. No.**	1.35e+16

Importance of polynomial terms

The polynomial term hour_squared and hum_squared are significant at 5% and they are both equal to 0.00.

temp's coefficient is 346, this indicates that ridership increases by 0. 346 bikes when temp increases by a unit, assuming that nothing else changes. The predictor *temp* is significant with p-value 0, within the confidence interval [227 - 465] which is acceptable.

hour's coefficient is 38, this indicates that ridership increases by ca. 38 bikes when time passes during the day, assuming that nothing else changes. The predictor *hour* is very significant with p-value 0, within a small confidence interval [37 - 39] which is acceptable.

hum's coefficient is 76, this indicates that ridership increases by ca. 38 bikes when humidity increases, assuming that nothing else changes. The predictor *hum* is less but still significant with p-value 0.004 within a confidence interval [2 - 150] which is acceptable.

The confidence intervals for all three predictors does not contain zero. This increase their significance for the ridership prediction. R2 score of the model is 0.450, which is an improvement compared to 0.407 obtained without polynomial terms.

The following plots illustrate how ridership changes as a function of the predicted temp, hour and hum.

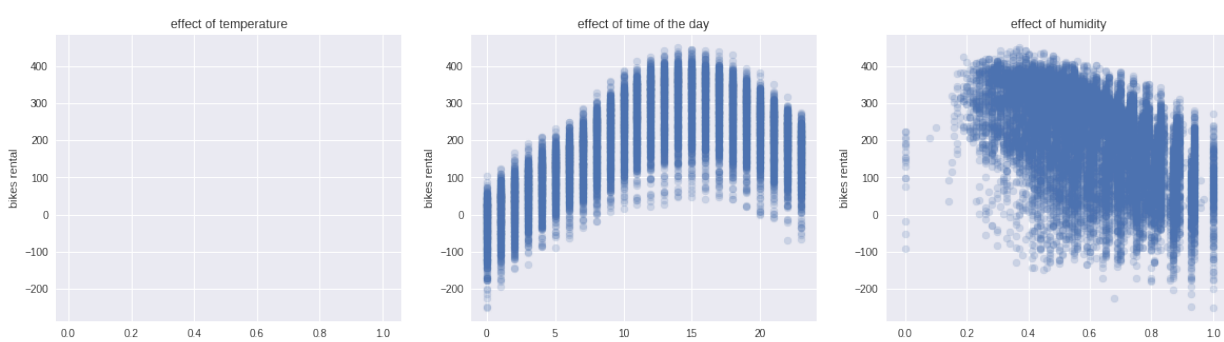
The followoing changes are to be noted: Hour 7.1120 increases to 38.4605 temp 237.8634 increase to 46.6924 hum -209.8955 increase to 76.0973

predict bike rentals using the new polynomial model on training set

In [0]:
```
def sorted_model(p):
    sorted_X_train_poly = X_train_poly.copy()
    sorted_X_train_poly = sorted_X_train_poly.sort_values([p])
    sorted_design_mat_train_poly = sorted_X_train_poly[predictors_poly
]
    sorted_design_mat_train_poly = sm.add_constant(sorted_design_mat_t
rain_poly)
    sorted_fitted_model_train_poly = OLS(sorted_X_train_poly.counts, s
orted_design_mat_train_poly, hasconst=True).fit()
    predicted_counts_poly = sorted_fitted_model_train_poly.predict(sor
ted_design_mat_train_poly)
    return {'observed': sorted_X_train_poly[p], 'predicted': predicted
_counts_poly}
```

In [111]:
```
temp_model = sorted_model('temp')
hour_model = sorted_model('hour')
humidity_model = sorted_model('hum')

fig, ax = plt.subplots(1,3,figsize=(20,5))
ax[0].scatter(temp_model['observed'], temp_model['predicted'], alpha=0
.)
ax[0].set_xlabel('temp')
ax[0].set_ylabel('bikes rental')
ax[0].set_title('effect of temperature')

ax[1].scatter(hour_model['observed'], hour_model['predicted'], alpha=0
.2)
ax[1].set_xlabel('hour')
ax[1].set_ylabel('bikes rental')
ax[1].set_title('effect of time of the day')

ax[2].set_xlabel('humidity')
ax[2].set_ylabel('bikes rental')
ax[2].set_title('effect of humidity')
ax[2].scatter(humidity_model['observed'], humidity_model['predicted'],
alpha=0.2)
```

Out[111]: <matplotlib.collections.PathCollection at 0x7f9a52323eb8>



4.5 The strange coefficients from 4.2 could also come from *multicolinearity*, where one or more predictors capture the same information as existing predictors. Why can multicolinearity 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 [112]:

```
# your code here
design_mat_train_poly_drop = design_mat_train_poly.copy()
design_mat_train_poly_drop = design_mat_train_poly.drop(columns = ['wo
rkingday', 'atemp', 'spring', 'summer', 'fall'])
fitted_model_train_poly_clean = OLS(X_train_poly.counts, design_mat_tr
ain_poly_drop, hasconst=True).fit()
fitted_model_train_poly_clean.summary()
```

Out[112]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.447
Model:	OLS	**Adj. R-squared:**	0.446
Method:	Least Squares	**F-statistic:**	400.8
Date:	Thu, 04 Oct 2018	**Prob (F-statistic):**	0.00
Time:	02:15:30	**Log-Likelihood:**	-87813.
No. Observations:	13903	**AIC:**	1.757e+05
Df Residuals:	13874	**BIC:**	1.759e+05
Df Model:	28		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-171.0970	14.675	-11.659	0.000	-199.863	-142.331
hour	38.5168	0.696	55.379	0.000	37.154	39.880
holiday	-30.5131	7.119	-4.286	0.000	-44.468	-16.558
temp	387.9643	38.082	10.188	0.000	313.319	462.609
hum	65.1888	37.321	1.747	0.081	-7.964	138.342
windspeed	-31.3573	10.061	-3.117	0.002	-51.079	-11.636
Feb	-5.7315	5.824	-0.984	0.325	-17.147	5.684
Mar	9.9429	6.145	1.618	0.106	-2.103	21.989

Apr	12.5709	6.656	1.889	0.059	-0.476	25.617
May	15.6356	7.488	2.088	0.037	0.958	30.313
Jun	-11.1305	8.099	-1.374	0.169	-27.006	4.745
Jul	-45.7358	8.816	-5.188	0.000	-63.017	-28.454
Aug	-18.3993	8.364	-2.200	0.028	-34.793	-2.006
Sept	29.7606	7.716	3.857	0.000	14.637	44.884
Oct	51.5769	6.858	7.521	0.000	38.135	65.019
Nov	40.7049	6.128	6.642	0.000	28.692	52.717
Dec	30.1671	5.849	5.157	0.000	18.702	41.632
Mon	10.2991	4.394	2.344	0.019	1.687	18.912
Tue	7.6816	4.271	1.798	0.072	-0.691	16.054
Wed	14.3871	4.246	3.388	0.001	6.064	22.710
Thu	11.5267	4.274	2.697	0.007	3.148	19.905
Fri	15.3557	4.258	3.606	0.000	7.010	23.702
Sat	14.6520	4.229	3.465	0.001	6.363	22.941
Cloudy	-5.0295	2.817	-1.786	0.074	-10.551	0.492
Snow	-43.2187	4.818	-8.970	0.000	-52.663	-33.775
Storm	50.3011	94.959	0.530	0.596	-135.832	236.435
temp_squared	-40.1818	37.482	-1.072	0.284	-113.653	33.289
hour_squared	-1.3271	0.028	-47.046	0.000	-1.382	-1.272
hum_squared	-174.1359	29.892	-5.825	0.000	-232.729	-115.543

Omnibus:	3525.953	**Durbin-Watson:**	0.800
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8606.649
Skew:	1.404	**Prob(JB):**	0.00
Kurtosis:	5.640	**Cond. No.**	2.05e+04

Are the estimates more sensible in this model?

The model score improves slightly from 0.4394891374613129 to 0.435860652616913. Reviewing the changes in the coefficients, we see that the model has the greatest impact on Tuesday increasing 2,024% but this result is not significant at 5%.

After dropping workingday, atemp, spring, summer and fall, the model coefficients appear to be more sensible.

The month's coefficients are now mostly positive. The confidence interval of most of them does not include zero any more and it became narrower. More p-values are below 0.05. The corrrelation with the weekdays makes more sense.

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?

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 [0]:
```
# your code here

predictors_all = ['hour', 'holiday', 'workingday', 'temp', 'atemp', 'h
um',
       'windspeed', 'spring', 'summer', 'fall', 'Feb', 'Mar', 'Apr',
       'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon',
'Tue',
       'Wed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm', 'temp_sq
uared', 'hour_squared', 'hum_squared']
```

In [0]:
```
# your code here
def linear_regression_train(s):
    design_mat_train_poly_subset = X_train_poly[s]
    design_mat_train_poly_subset = sm.add_constant(design_mat_train_po
ly_subset)
    fitted_model_train_poly_subset = OLS(X_train_poly.counts, design_m
at_train_poly_subset).fit()
    return fitted_model_train_poly_subset

def linear_regression_test(s):
    design_mat_test_poly_subset = X_test_poly[s]
    design_mat_test_poly_subset = sm.add_constant(design_mat_test_poly
_subset)
    fitted_model_test_poly_subset = OLS(X_test_poly.counts, design_mat
_test_poly_subset).fit()
    return fitted_model_test_poly_subset

def select_next_predictor(current_predictors, candidate_predictors):
    model = linear_regression_train(current_predictors)
    min_bic = model.bic
    best_predictor = None
    for predictor in candidate_predictors:
        current_predictors_copy = current_predictors.copy()
        current_predictors_copy.append(predictor)
        model = linear_regression_train(current_predictors_copy)
        if model.bic < min_bic:
```

```
                best_predictor = predictor
                min_bic = model.bic
        if best_predictor != None:
            current_predictors.append(best_predictor)
            return True
        else:
            return False

candidates_predictors = ['holiday', 'workingday', 'temp', 'atemp', 'hu
m',
       'windspeed', 'spring', 'summer', 'fall', 'Feb', 'Mar', 'Apr',
       'May', 'Jun', 'Jul', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec', 'Mon',
'Tue',
       'Wed', 'Thu', 'Fri', 'Sat', 'Cloudy', 'Snow', 'Storm', 'temp_sq
uared', 'hour_squared', 'hum_squared']

# forward predictors selection using BIC on train data

selected_predictors = ['hour']
selected_predictors_sets = []
while len(candidates_predictors)>0:
    if select_next_predictor(selected_predictors, candidates_predictor
s):
        candidates_predictors.remove(selected_predictors[-1])
        selected_predictors_sets.append(selected_predictors.copy())
    else:
        break

# comparison of predictors set and model selection using BIC on test d
ata
model = linear_regression_test(['hour'])
min_bic = model.bic
for predictors_set in selected_predictors_sets:
  model = linear_regression_test(predictors_set)
  if model.bic < min_bic:
      selected_predictors = predictors_set
      min_bic = model.bic
```

In [115]:

```
# subsets of predictors which where selected using forward step-wise s
election on train data
selected_predictors_sets
```

```
Out[115]: [['hour', 'hour_squared'],
           ['hour', 'hour_squared', 'temp'],
           ['hour', 'hour_squared', 'temp', 'hum_squared'],
           ['hour', 'hour_squared', 'temp', 'hum_squared', 'fall'],
           ['hour', 'hour_squared', 'temp', 'hum_squared', 'fall', 'Jul'],
           ['hour', 'hour_squared', 'temp', 'hum_squared', 'fall', 'Jul', 'Sno
          w'],
           ['hour',
            'hour_squared',
            'temp',
            'hum_squared',
            'fall',
            'Jul',
            'Snow',
            'spring'],
           ['hour',
            'hour_squared',
            'temp',
            'hum_squared',
            'fall',
            'Jul',
            'Snow',
            'spring',
            'Sept'],
           ['hour',
            'hour_squared',
            'temp',
            'hum_squared',
            'fall',
            'Jul',
            'Snow',
            'spring',
            'Sept',
            'holiday'],
           ['hour',
            'hour_squared',
            'temp',
            'hum_squared',
            'fall',
            'Jul',
            'Snow',
            'spring',
            'Sept',
            'holiday',
            'Jun']]
```

In [116]:
```
# set of predictors which where selected using models comparison (bic)
on test data
selected_predictors
```

Out[116]:
```
['hour',
 'hour_squared',
 'temp',
 'hum_squared',
 'fall',
 'Jul',
 'Snow',
 'spring',
 'Sept',
 'holiday',
 'Jun']
```

In [117]:
```
# Standardizing the predictors in order to make them unitless for fair
er comparisons:
X_train_poly_normalized = (X_train_poly-X_train_poly.mean())/X_train_p
oly.std()
X_test_poly_normalized =(X_test_poly-X_test_poly.mean())/X_test_poly.s
td()
X_test_poly_normalized.head()
```

Out[117]:

	hour	holiday	year	workingday	temp	atemp	hum	windsp
0	-0.813782	-0.165778	-0.988985	-1.495278	-1.447916	-1.192514	0.937306	-1.5394
1	-0.379065	-0.165778	-0.988985	-1.495278	-0.927100	-0.750438	0.729086	-1.5394
2	1.214900	-0.165778	-0.988985	-1.495278	-0.510447	-0.397010	1.301692	0.5119
3	-0.234159	-0.165778	-0.988985	-1.495278	-0.718774	-0.750438	0.989361	0.2709
4	0.055653	-0.165778	-0.988985	-1.495278	-0.718774	-0.839086	0.208535	0.8741

5 rows × 35 columns

In [118]:
```
# fit the best model on train data using the set of predictors which w
here selected using models comparison (bic) on test data
best_model = linear_regression_train(selected_predictors)
best_model.summary()
```

Out[118]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.449
Model:	OLS	**Adj. R-squared:**	0.449
Method:	Least Squares	**F-statistic:**	1030.
Date:	Thu, 04 Oct 2018	**Prob (F-statistic):**	0.00
Time:	02:16:01	**Log-Likelihood:**	-87786.
No. Observations:	13903	**AIC:**	1.756e+05
Df Residuals:	13891	**BIC:**	1.757e+05
Df Model:	11		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-140.7030	5.361	-26.245	0.000	-151.212	-130.194
hour	38.1747	0.673	56.763	0.000	36.857	39.493
hour_squared	-1.3134	0.027	-47.810	0.000	-1.367	-1.260
temp	332.6262	7.690	43.256	0.000	317.553	347.699
hum_squared	-123.0671	5.483	-22.445	0.000	-133.814	-112.320
fall	55.5861	2.938	18.921	0.000	49.828	61.344
Jul	-32.8398	5.053	-6.500	0.000	-42.743	-22.936
Snow	-43.1307	4.449	-9.695	0.000	-51.851	-34.411
spring	25.8946	3.068	8.441	0.000	19.881	31.908
Sept	26.9924	4.565	5.913	0.000	18.044	35.940
holiday	-25.6510	6.740	-3.806	0.000	-38.863	-12.439
Jun	-15.6397	4.642	-3.369	0.001	-24.738	-6.542

Omnibus:	3546.069	**Durbin-Watson:**	0.801
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	8757.176
Skew:	1.406	**Prob(JB):**	0.00
Kurtosis:	5.684	**Cond. No.**	1.98e+03

In [119]:
```
print("selected predictors: ", selected_predictors)
print("eliminated predictors: ", set(predictors_all)- set(selected_pre
dictors))
print("R2: ", best_model.rsquared)
print("BIC: ", best_model.bic)
```

```
selected predictors:  ['hour', 'hour_squared', 'temp', 'hum_squared'
, 'fall', 'Jul', 'Snow', 'spring', 'Sept', 'holiday', 'Jun']
eliminated predictors:  {'Apr', 'summer', 'Sat', 'May', 'Dec', 'hum'
, 'atemp', 'Nov', 'Wed', 'Tue', 'Cloudy', 'Mar', 'Feb', 'Oct', 'Mon'
, 'Storm', 'Aug', 'temp_squared', 'workingday', 'Thu', 'Fri', 'winds
peed'}
R2:  0.4493269812445594
BIC:  175686.6398322799
```

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.

The forward selection elimiate additional predictors as reported above.

The predictors *fall* and *spring* were not dropped. When added during forward selection, they decreased the BIC metric and were retained. This behavior can be explained by looking at the estimates, p-values and confidence intervals of those predictors in the question 4.4 just before they were removed. Both predictors were significant (p = 0) with relatively small confidence intervals. Now they are even more significant in our best model with forward selection.

All coefficients are significant ($p<0.05$) within very small confidence intervals, neither of them containing zero. Their standard error is low compared to previous models. The overall bic is small and R2 score is acceptable (0.449).

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 [0]:
```
# calculare R2 of both polynomial (4.4) model and best model (5.1) on
test and train data
def evaluate_model(model, predictors):
    # linear regression using the 'best' model on test data
    design_mat_test_poly = X_test_poly[predictors]
    design_mat_test_poly = sm.add_constant(design_mat_test_poly)
    predicted_counts_test = model.predict(design_mat_test_poly)
    r2_test = r2_score(X_test_poly['counts'], predicted_counts_test)
    # linear regression using the 'best' model on train data
    design_mat_train_poly = X_train_poly[predictors]
    design_mat_train_poly = sm.add_constant(design_mat_train_poly)
    predicted_counts_train = model.predict(design_mat_train_poly)
    r2_train = r2_score(X_train_poly['counts'], predicted_counts_train
)
    return (r2_train, r2_test)
```

In [124]:
```
print ("R2 score for the fitted step-wise model on (train, test) data
is: " + str(evaluate_model(best_model, selected_predictors)))
```

```
R2 score for the fitted step-wise model on (train, test) data is: (0
.4493269812445593, 0.4386824653131691)
```

In [125]:
```
print ("R2 score for the 4.4 polynomial model on (train, test) data is
: " + str(evaluate_model(fitted_model_train_poly, predictors_poly)))
```

```
R2 score for the 4.4 polynomial model on (train, test) data is: (0.4
520259183519364, 0.4394891374613128)
```

The step-wise model was fitted on the train data set with a R2 score 0.449. The polynomial model was fitted on the same data set with a R2 score 0.452.

The fitted step-wise model was used on test data for validation with a R2 score 0.438. When we predict ridership using the polynomial model on test data, we get a R2 score 0.439.

Written Report to the Administrators [20 pts]

Question 6

Write a short repost 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.

The following is the name of the report file:

hw3_summary_report_Michel_Kana_Gildas_Bah.pdf

Answers 6

your answer here

your answer here

In [0]:
```
#your code here
```

your answer here