S-109A Introduction to Data Science

Homework 3 - Forecasting Bike Sharing Usage

Harvard University
Summer 2018
Instructors: Pavlos Protopapas, Kevin Rader

INSTRUCTIONS

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

Names of people you have worked with goes here:

Nikita Roy

In [625]:
```
from IPython.core.display import HTML
def css_styling(): styles = open("cs109.css", "r").read(); return HTML(styles)
css_styling()
```

Out[625]:

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.

Take as many short rides as you want while your pass is active. Passes and memberships include unlimited trips under 30 minutes.

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.

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

Overview

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

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

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

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

Use only the libraries below:

In [626]:
```
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 add some features that will help us with the analysis and then separate it into training and test sets. Each row in this file contains 12 attributes and each entry represents one hour of a 24-hour day with its weather, etc, and the number of rental rides for that day divided in categories according to if they were made by registered or casual riders. Those attributes are the following:

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

General Hints

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

Resources

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

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

Learn your Domain and Perform a bit of Feature Engineering

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

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

1.3 Create three new columns in the dataframe:

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

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

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

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

Answers

1.1 Load the dataset from the csv file ...

In [627]:
```
# Load dataframe
bikes_df = pd.read_csv("data/BSS_hour_raw.csv")
bikes_df.describe()
```

Out[627]:

	season	hour	holiday	weekday	workingday	we
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.00
mean	2.501640	11.546752	0.028770	3.003683	0.682721	1.425283
std	1.106918	6.914405	0.167165	2.005771	0.465431	0.639357
min	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2.000000	6.000000	0.000000	1.000000	0.000000	1.000000
50%	3.000000	12.000000	0.000000	3.000000	1.000000	1.000000
75%	3.000000	18.000000	0.000000	5.000000	1.000000	2.000000
max	4.000000	23.000000	1.000000	6.000000	1.000000	4.000000

The temp and atemp ranges seem to be suspect, as they appear to be normalized and not in degrees Celsius. The means of season, workingday, weekday and holiday cannot be interpreted. The range of season, weekday are innapropriate, as they are categorical variables.

1.2 Notice that the variable in column ....

In [628]:
```
#Convert dteday into datetime
bikes_df["dteday"] = pd.to_datetime(bikes_df["dteday"], yearfirst = True)

#print head
bikes_df.head()
```

Out[628]:

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

1.3 Create three new columns ...

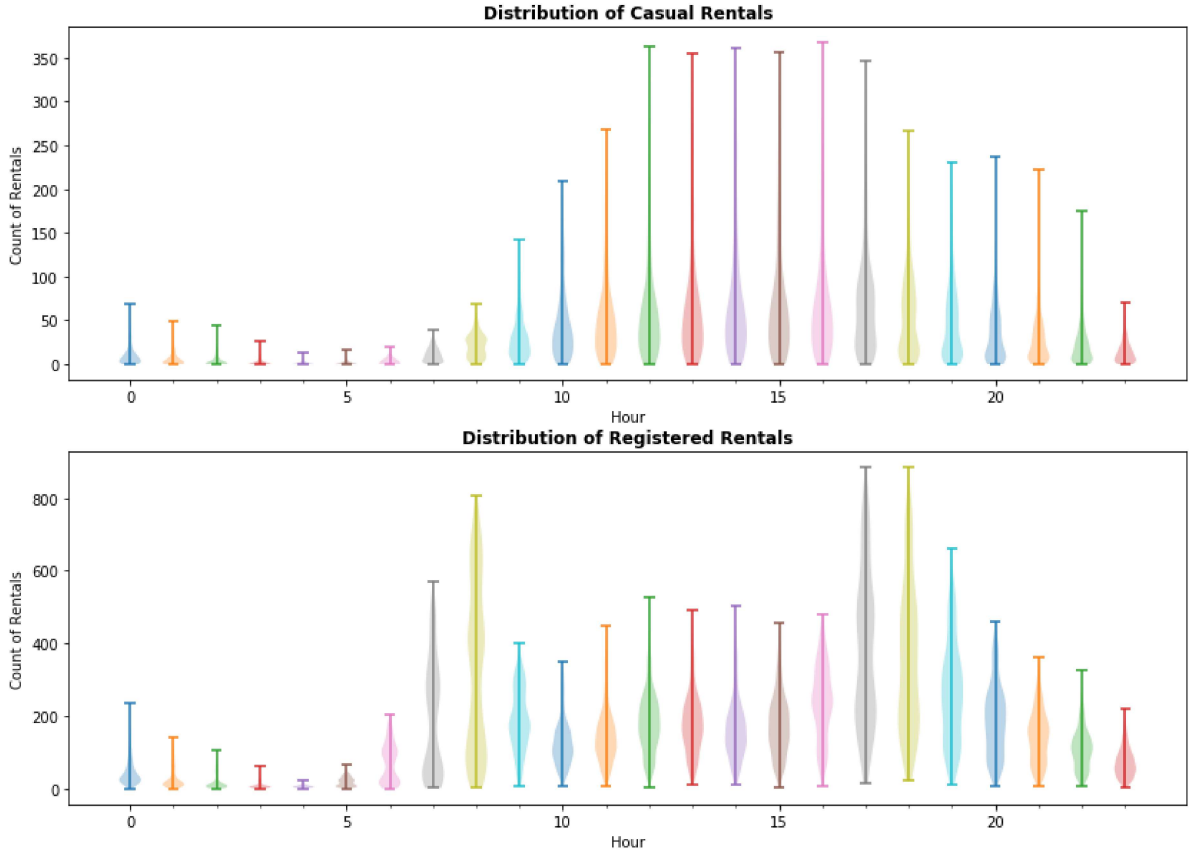
In [629]:
```
# Create year and month columns
bikes_df["year"] = bikes_df["dteday"].dt.year - 2011
bikes_df["month"] = bikes_df["dteday"].dt.month

# make new dataframe column for counts
bikes_df["counts"] = bikes_df["casual"] + bikes_df["registered"]
bikes_df.head()
```

Out[629]:

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

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

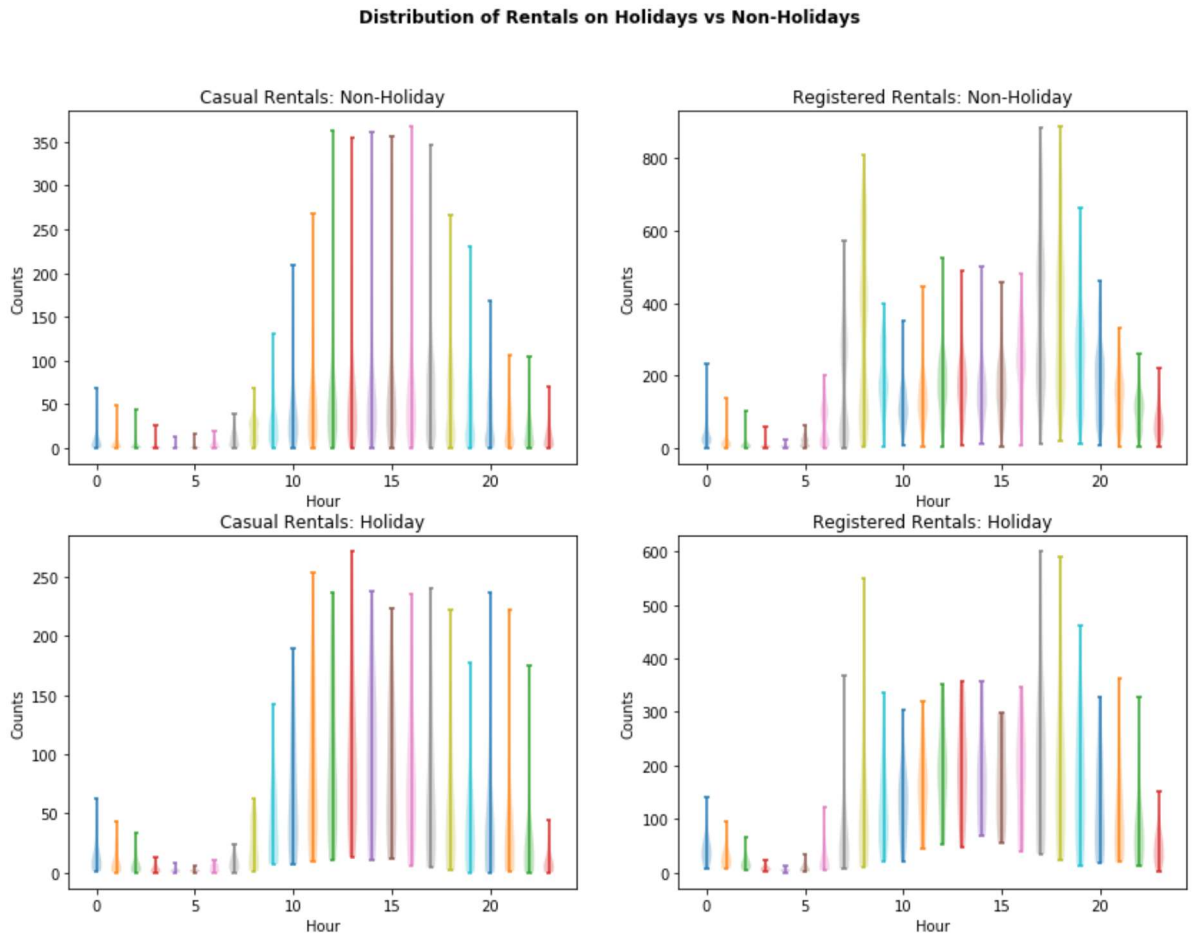
In [630]:
```
# initialize figure
fig, ax = plt.subplots(2,1, figsize=(14,10))

# plot several violin plots
cur_x_pos = 0

# for each position, violinplot the rental counts for casual and registered bi
kes
for cur_hour, cur_df in bikes_df.groupby('hour'):
    ax[0].violinplot(cur_df['casual'], positions=[cur_x_pos])
    ax[1].violinplot(cur_df['registered'], positions=[cur_x_pos])

    # increment the position
    cur_x_pos+=1

# get the hour for each position and set the x-ticks
hours = [x[0] for x in bikes_df.groupby('hour')]
ax[0].set_xticks(range(len(hours)), hours)
ax[1].set_xticks(range(len(hours)), hours);

# fill in the remaining labels
ax[0].set_xlabel("Hour")
ax[1].set_xlabel("Hour")
ax[0].set_ylabel("Count of Rentals")
ax[1].set_ylabel("Count of Rentals")
ax[0].set_title("Distribution of Casual Rentals", fontweight="bold")
ax[1].set_title("Distribution of Registered Rentals", fontweight="bold");
```



The registered rental distribution is bimodal around the commuting hours (8 am, 5-6pm), and the max number of rentals per hour exceeds 800 bikes. Casual rentals are centered around the daytime hours (12:00 pm to 5:00 pm) with a maxium number of rentals not exceeding 400 bikes per hour.

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

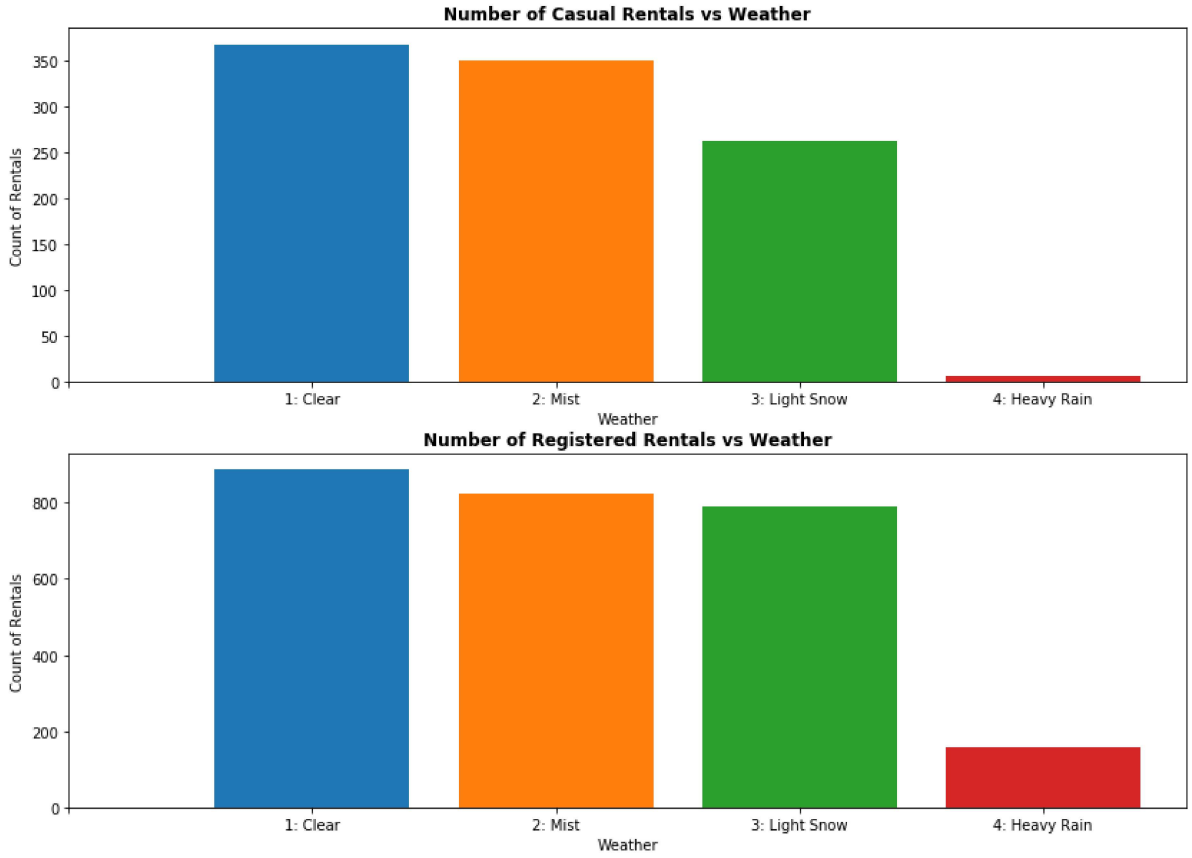
In [631]:
```
# initialize figure
fig, ax = plt.subplots(2,2, figsize=(14,10))


# plot several violin plots
cur_x_pos = 0

# for each position, violinplot the various rental counts for holidays and non
-holidays
for cur_day, cur_d in bikes_df.groupby('holiday'):

    # Set variable for subplot titles
    if(cur_day == 0):
        day = "Non-Holiday"
    else:
        day = "Holiday"

    # Create subplots of casual and registered rides for each holiday subgroup
    for cur_hour, cur_df in cur_d.groupby('hour'):
        ax[cur_day][0].violinplot(cur_df['casual'], positions=[cur_x_pos])
        ax[cur_day][1].violinplot(cur_df['registered'], positions=[cur_x_pos])

        # Set Labels
        ax[cur_day][0].set_xlabel("Hour")
        ax[cur_day][1].set_xlabel("Hour")
        ax[cur_day][0].set_ylabel("Counts")
        ax[cur_day][1].set_ylabel("Counts")
        ax[cur_day][0].set_title("Casual Rentals: {0}".format(day))
        ax[cur_day][1].set_title("Registered Rentals: {0}".format(day))
        cur_x_pos+=1

    cur_x_pos=0

fig.suptitle("Distribution of Rentals on Holidays vs Non-Holidays", fontweight
="bold");
```



On holidays, there is a decrease in casual and registered rentals. 350 is the highest casual rental count per hour observed on non-holidays. while 250 is the highest casual rental count per hour observed on holidays. More than 800 is the highest registered rental count per hour observed on non-holidays, while 600 is the highest casual rental count per hour observed on holidays.

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

In [632]:
```
# initialize figure
fig, ax = plt.subplots(2,1, figsize=(14,10))

# for each weather, barplot the casual and registered values of that group
for cur_weath, cur_df in bikes_df.groupby('weather'):
    ax[0].bar(cur_weath, cur_df['casual'])
    ax[1].bar(cur_weath, cur_df['registered'])
    # increment the x-position

# get the name of each position and set the x ticks
weaths = ["", "1: Clear", "2: Mist", "3: Light Snow", "4: Heavy Rain"]
ax[0].set_xticks(range(len(weaths)), weaths)
ax[1].set_xticks(range(len(weaths)), weaths)
ax[0].set_xticklabels(weaths)
ax[1].set_xticklabels(weaths);

# fill in the remaining labels
ax[0].set_xlabel("Weather")
ax[1].set_xlabel("Weather")
ax[0].set_ylabel("Count of Rentals")
ax[1].set_ylabel("Count of Rentals")
ax[0].set_title("Number of Casual Rentals vs Weather", fontweight="bold")
ax[1].set_title("Number of Registered Rentals vs Weather", fontweight="bold");
```



The proportion of registered rentals during category 3 and4 weather, the most severe weather, is much larger relative to the total number of rentals than the proportion of casual rentals during category 3and 4 weather versus the total number of casual rentals. However, the most rentals for both casual and clear rentals are on clear days.

Question 2: Explore Seasonality on Bike Ridership.

Seasonality and weather

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

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

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

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

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

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

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

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

HINT

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

Answers

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

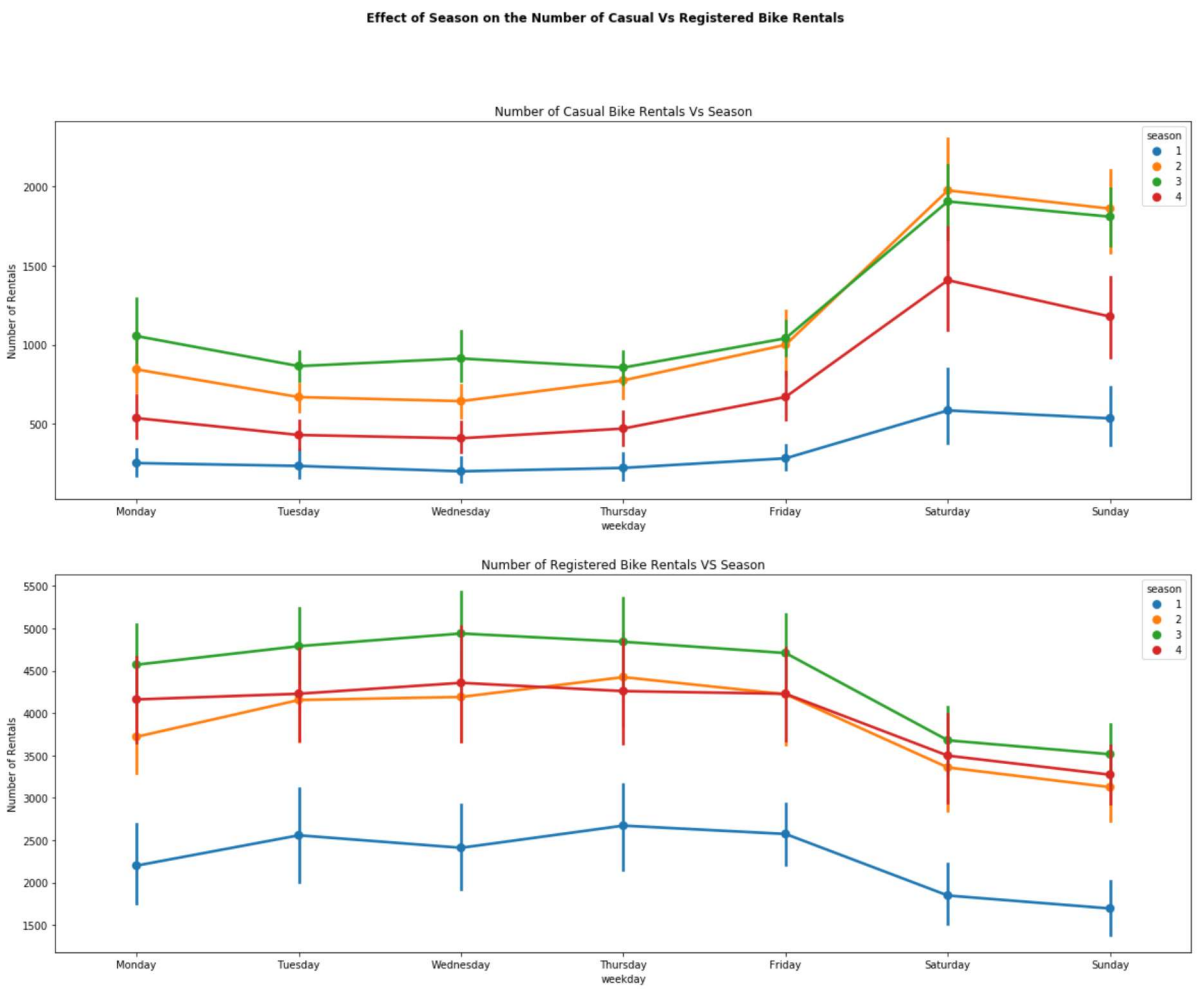
In [633]:
```
# make a copy of bikes_df
bikes_temp = bikes_df.copy()

# add a weekday column
bikes_temp['weekday'] = bikes_temp["dteday"].dt.dayofweek

# using groupby dteday, aggregate the data for each column in the group
bikes_by_day = bikes_temp.groupby('dteday').agg({
    'dteday': np.max,
    '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,
    'counts': np.sum
})

# print the head
bikes_by_day.head()
```

Out[633]:

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	ca
dteday									
2011-01-01	2011-01-01	5	3	1	0.344167	0.363625	0.160446	0.805833	33
2011-01-02	2011-01-02	6	3	1	0.363478	0.353739	0.248539	0.696087	13
2011-01-03	2011-01-03	0	1	1	0.196364	0.189405	0.248309	0.437273	12
2011-01-04	2011-01-04	1	2	1	0.200000	0.212122	0.160296	0.590435	10
2011-01-05	2011-01-05	2	1	1	0.226957	0.229270	0.186900	0.436957	82

2.2 How does season affect the number of bike ...

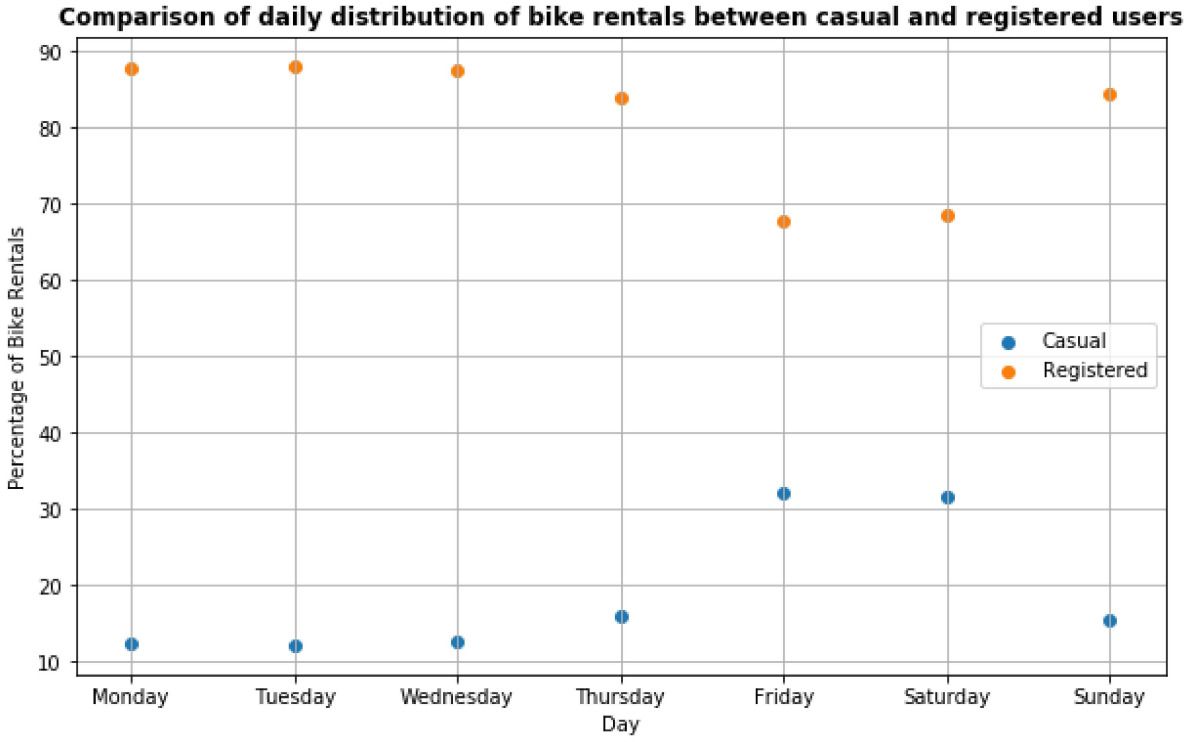
In [634]:
```
# initialize the figure
fig2, ax = plt.subplots(2,1, figsize = (20,15))

# Set suptitle, use seaborn to create two pointplots with error bars
fig2.suptitle("Effect of Season on the Number of Casual Vs Registered Bike Rentals", fontweight="bold")
sns.pointplot(data = bikes_by_day[['weekday', 'casual', 'season']], x = 'weekday', y = 'casual', hue = 'season', ax = ax[0])
sns.pointplot(data = bikes_by_day[['weekday', 'registered', 'season']], x = 'weekday', y = 'registered', hue = 'season', ax = ax[1])

# set labels
ax[0].set_title("Number of Casual Bike Rentals Vs Season")
ax[1].set_title("Number of Registered Bike Rentals VS Season")
ax[0].set_ylabel("Number of Rentals")
ax[1].set_ylabel("Number of Rentals")

# set x tick labels tp the days of the week
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
ax[0].set_xticklabels(days)
ax[1].set_xticklabels(days);
```



The most casual and registered rentals occur during summer, and the least casual and registered rentals occur during winter. However, spring and fall had similar registered rental rates while fall had a lower casual rental rate than spring.

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

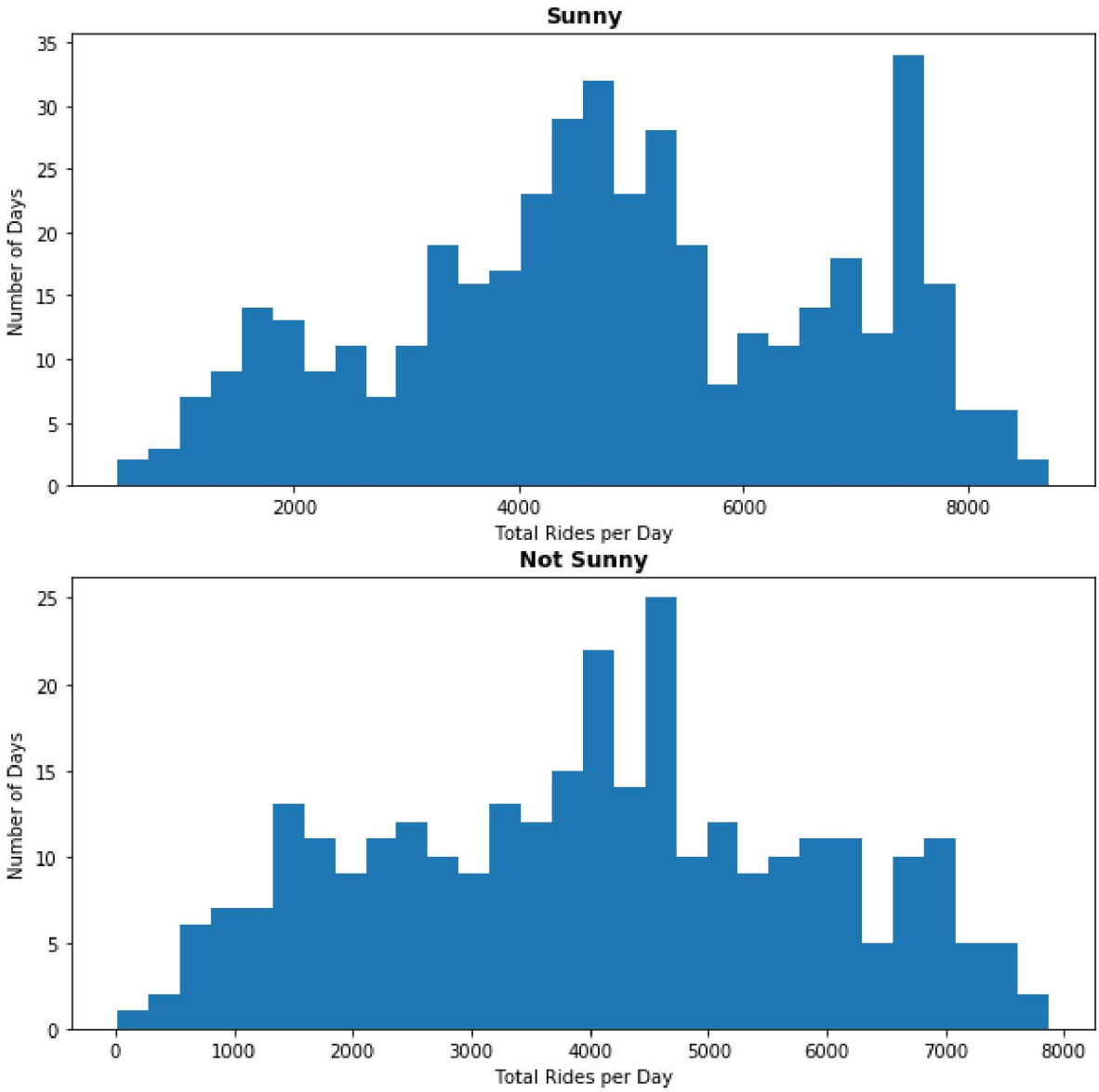
In [635]:
```
# compute the percentage rentals for casual and registered riders
percent_c = bikes_df.groupby(['weekday'])['casual'].sum().values
percent_r = bikes_df.groupby(['weekday'])['registered'].sum().values

# create percent dataframe
percent = pd.DataFrame()
percent['Weekday'] = bikes_by_day['weekday'].unique()
percent['Casual'] = percent_c
percent['Percent_c'] = (percent_c/(percent_c + percent_r))*100
percent['Registered'] = percent_r
percent['Percent_r'] = (percent_r/(percent_c + percent_r))*100
percent = percent.sort_values(by= ['Weekday'])

# create scatterplot
fig, ax = plt.subplots(1,1, figsize = (10,6))
ax.scatter(percent['Weekday'], percent['Percent_c'], label = "Casual")
ax.scatter(percent['Weekday'], percent['Percent_r'], label = "Registered")

# set labels
days = ['','Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
 'Sunday']
ax.set_xticklabels(days)
ax.set_xlabel("Day")
ax.set_ylabel("Percentage of Bike Rentals")
ax.set_title("Comparison of daily distribution of bike rentals between casual
 and registered users", fontweight = 'bold')
ax.grid(True)
ax.legend();
```



The days with the highest percentage of registered riders are Monday and Tuesday. Wednesday, Thursday, and Sunday also have registered percentages above 80 %. However, on Friday and Saturday, the percentage of registered rentals drops to below 65%, possibly because many registered bikers are commuters that travel on the weekends, and many casual bikers possibly have more time for leisure activities such as biking and visiting Washington DC on Friday and Saturday. There are probably more tourists in Washington DC on Friday and Saturday.

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

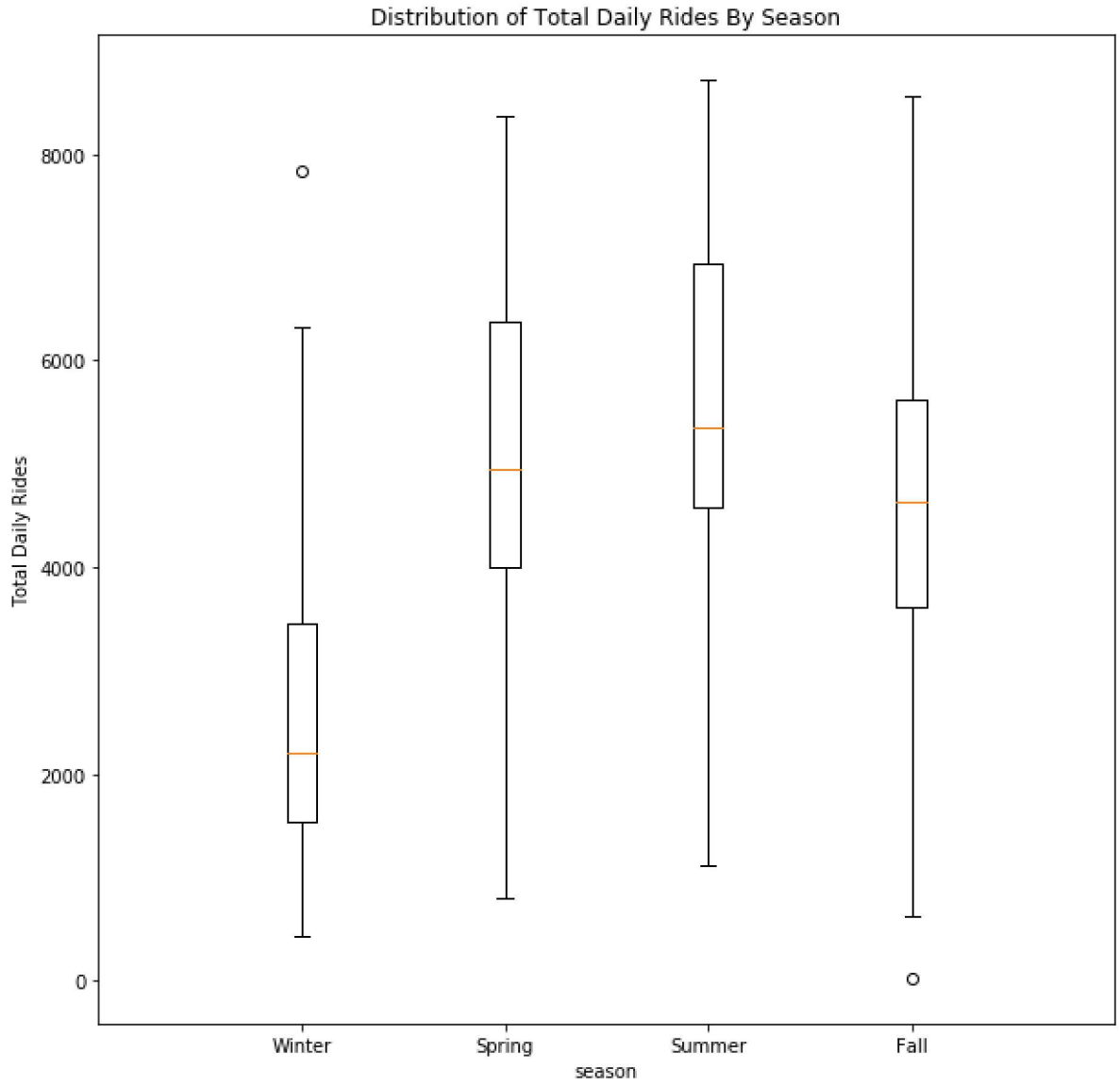
In [636]:
```
# create a dataframe to store if weather is sunny or not.
# we decided that category 1 in weather would be considered sunny due to the l
ack of precipitation
# and potential lack of clouds.
bikes_by_day_sunny = bikes_by_day.copy()
bikes_by_day_sunny["weather"] = bikes_by_day_sunny['weather'].apply(lambda x:
int((x-1)/2))

# initialize the figure
fig, ax = plt.subplots(2,1, figsize=(10,10))

# Create subplots
for cur_weather, cur_df in bikes_by_day_sunny.groupby("weather"):

    # Store title string
    if cur_weather == 0:
        weather = "Sunny"
    else:
        weather = "Not Sunny"

    # Create histogram
    ax[cur_weather].hist(cur_df["counts"], bins=30)

    # Set labels
    ax[cur_weather].set_xlabel("Total Rides per Day")
    ax[cur_weather].set_ylabel("Number of Days")
    ax[cur_weather].set_title(weather, fontweight = "bold")
```



The distribution of sunny days appears to be bimodal around ~5000 and ~7500, while the distribution of not-sunny days appears to be unimodal around approximately ~4500 days. The max bin height for sunny days is ~35 days, while for not sunny days it is only ~25 days

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

In [637]:
```
# initialize figure
fig3, ax = plt.subplots(1,1, figsize=(10,10))

# initialize x-position
cur_x_pos = 0

# Create boxplots for all four seasons
for cur_position, cur_df in bikes_by_day.groupby('season'):
    box_output = ax.boxplot(cur_df['counts'], positions=[cur_x_pos])

    # increment x position
    cur_x_pos+=1

# get the name of each position and set the x ticks
plt.xticks([0,1,2,3], ["Winter", "Spring", "Summer", "Fall"]);
ax.set_xlim(-1,len(seasons))

# fill in the remaining labels
ax.set_xlabel("season")
ax.set_ylabel("Total Daily Rides")
ax.set_title("Distribution of Total Daily Rides By Season");
```



In [638]:
```
# locate and print the first outlier in the boxplot
selector1 = bikes_by_day['season']== 1
selector2 = bikes_by_day['counts'] > 7000
rows_of_interest = np.logical_and(selector1, selector2)
bikes_by_day.loc[rows_of_interest,:]
```

Out[638]:

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	ca
dteday									
2012-03-17	2012-03-17	5	2	1	0.514167	0.505046	0.110704	0.755833	31

This outlier is due to the St. Patrick's Day holiday, leading to more rentals.

In [639]:
```
# locate and print the second outlier in the boxplot
selector1 = bikes_by_day['season']== 4
selector2 = bikes_by_day['counts'] < 100
rows_of_interest = np.logical_and(selector1, selector2)
bikes_by_day.loc[rows_of_interest,:]
```

Out[639]:

	dteday	weekday	weather	season	temp	atemp	windspeed	hum	casual	regis
dteday										
2012-10-29	2012-10-29	0	3	4	0.44	0.4394	0.3582	0.88	2	20

This outlier is due to Hurricane Sandy closing down much of the city.

Question 3: Prepare the data for Regression

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

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

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

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

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

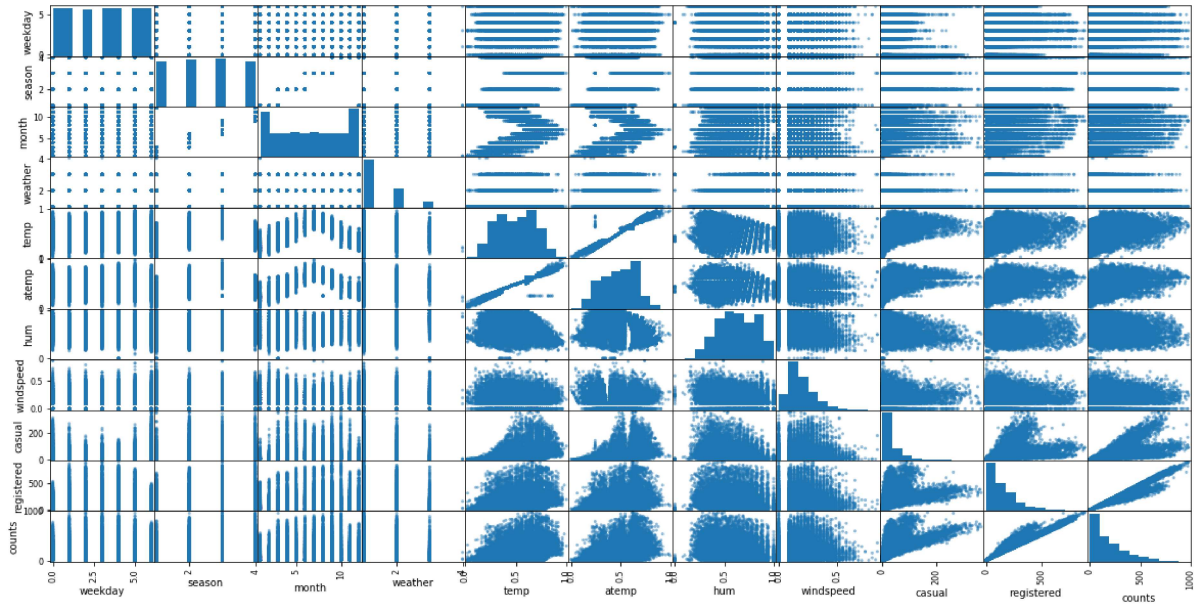
Hints:

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

Answers

3.1 Visualize and describe inter-dependencies ...

In [640]:
```
# create a scatter matrix for the specified columns
scatter_matrix(bikes_df.loc[:, ["weekday", "season", "month", "weather", "tem
p", "atemp", "hum", "windspeed", "casual", "registered", "counts"]], figsize=(
20,10));
```



atemp and temp have a strong positive relationship. There is a strong potential cosine relationship between temp/atemp and month. There is also a strong positive relationship between registered/casual and counts. temperature and casual have a weaker positive relationship, but interestingly temperature and registered don not appear to have a visible relationship. The distributions of windspeed, weather, casual, registered, and counts are skewed to the right.

3.2 Convert the categorical attributes ....

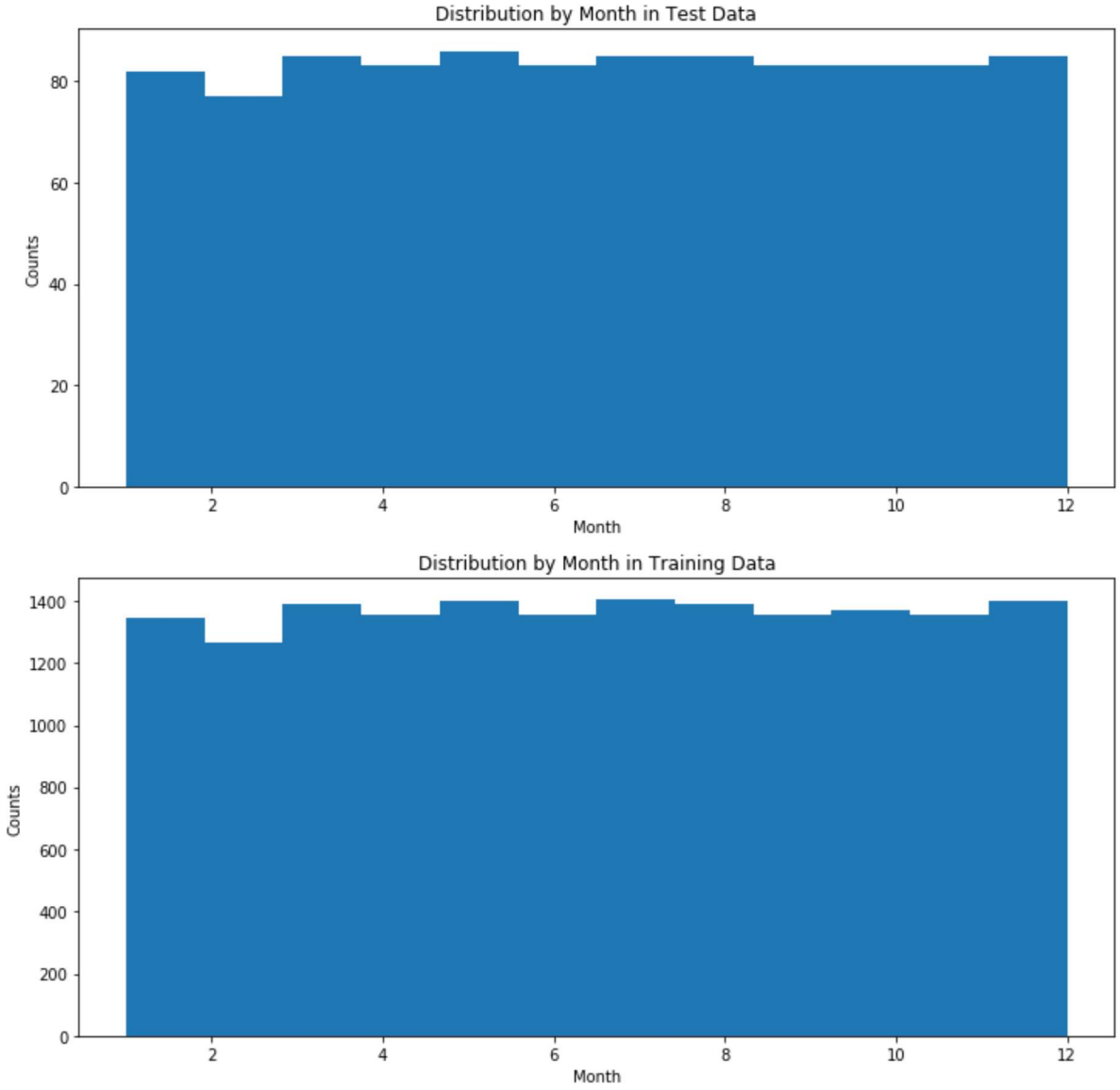
In [641]:
```
# one-hot encode the categorical variables season, holiday, weekday, weather,and month into multipple binary attributes
bikes_df_recoded  = pd.get_dummies(bikes_df, columns=["season", "holiday", "weekday", "weather", "month"], drop_first=True)

bikes_df_recoded.head()
```

Out[641]:

	dteday	hour	workingday	temp	atemp	hum	windspeed	casual	registered	year	co
0	2011-01-01	0	0	0.24	0.2879	0.81	0.0	3	13	0	16
1	2011-01-01	1	0	0.22	0.2727	0.80	0.0	8	32	0	40
2	2011-01-01	2	0	0.22	0.2727	0.80	0.0	5	27	0	32
3	2011-01-01	3	0	0.24	0.2879	0.75	0.0	3	10	0	13
4	2011-01-01	4	0	0.24	0.2879	0.75	0.0	0	1	0	1

3.3 Split the initial bikes_df dataset...

In [642]:
```
# store the length of the bikes_df
bikes_size = len(bikes_df)

# splits the bikes_df dataset into train and tests sets in a stratified fashio
n
try:
    train_data, test_data = train_test_split(bikes_df, test_size = 1000/bikes_
size, stratify=bikes_df["month"])

# if an exception is raised, the missing rows are dealt with accordingly
except:
    missing_rows = np.isnan(bikes_df["month"])
    print("Exception raised for missing row")
    league_df = league_df.dropna(subset=['month'])
    train_data, test_data = train_test_split(bikes_df, test_size = 1000/bikes_
size, stratify=bikes_df["month"])

# initialize the figure
fig, ax = plt.subplots(2,1, figsize=(12,12))

# plot bar charts of month distribution fot train and test data to show sample
s are stratified
ax[0].hist(test_data["month"], bins=12)
ax[1].hist(train_data["month"], bins=12)

# set labels
ax[0].set_xlabel("Month")
ax[1].set_xlabel("Month")
ax[0].set_ylabel("Counts")
ax[1].set_ylabel("Counts")
ax[0].set_title("Distribution by Month in Test Data")
ax[1].set_title("Distribution by Month in Training Data");
```



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

In [643]:
```
# read the files into dataframes
BSS_train = pd.read_csv("data/BSS_train.csv")
BSS_test = pd.read_csv("data/BSS_test.csv")
```

In [644]:
```
# check BSS_train for accuracy
BSS_train.head()
```

Out[644]:

	Unnamed: 0	dteday	hour	holiday	year	workingday	temp	atemp	hum	windspeed	c
0	0	2011-01-01	0	0	0	0	0.24	0.2879	0.81	0.0	3
1	1	2011-01-01	1	0	0	0	0.22	0.2727	0.80	0.0	8
2	2	2011-01-01	2	0	0	0	0.22	0.2727	0.80	0.0	5
3	3	2011-01-01	3	0	0	0	0.24	0.2879	0.75	0.0	3
4	4	2011-01-01	4	0	0	0	0.24	0.2879	0.75	0.0	0

In [645]:
```
# check BSS_test for accuracy
BSS_test.head()
```

Out[645]:

	Unnamed: 0	dteday	hour	holiday	year	workingday	temp	atemp	hum	windspeed	c
0	6	2011-01-01	6	0	0	0	0.22	0.2727	0.80	0.0000	2
1	9	2011-01-01	9	0	0	0	0.32	0.3485	0.76	0.0000	8
2	20	2011-01-01	20	0	0	0	0.40	0.4091	0.87	0.2537	1
3	33	2011-01-02	10	0	0	0	0.36	0.3485	0.81	0.2239	7
4	35	2011-01-02	12	0	0	0	0.36	0.3333	0.66	0.2985	2

In [646]:
```
# Drop the dteday, casual, registered, and Unnamed: 0 columns
BSS_train = BSS_train.drop(columns=["dteday", "casual", "registered","Unnamed:
 0"])
BSS_test = BSS_test.drop(columns=["dteday", "casual", "registered", "Unnamed:
 0"])
```

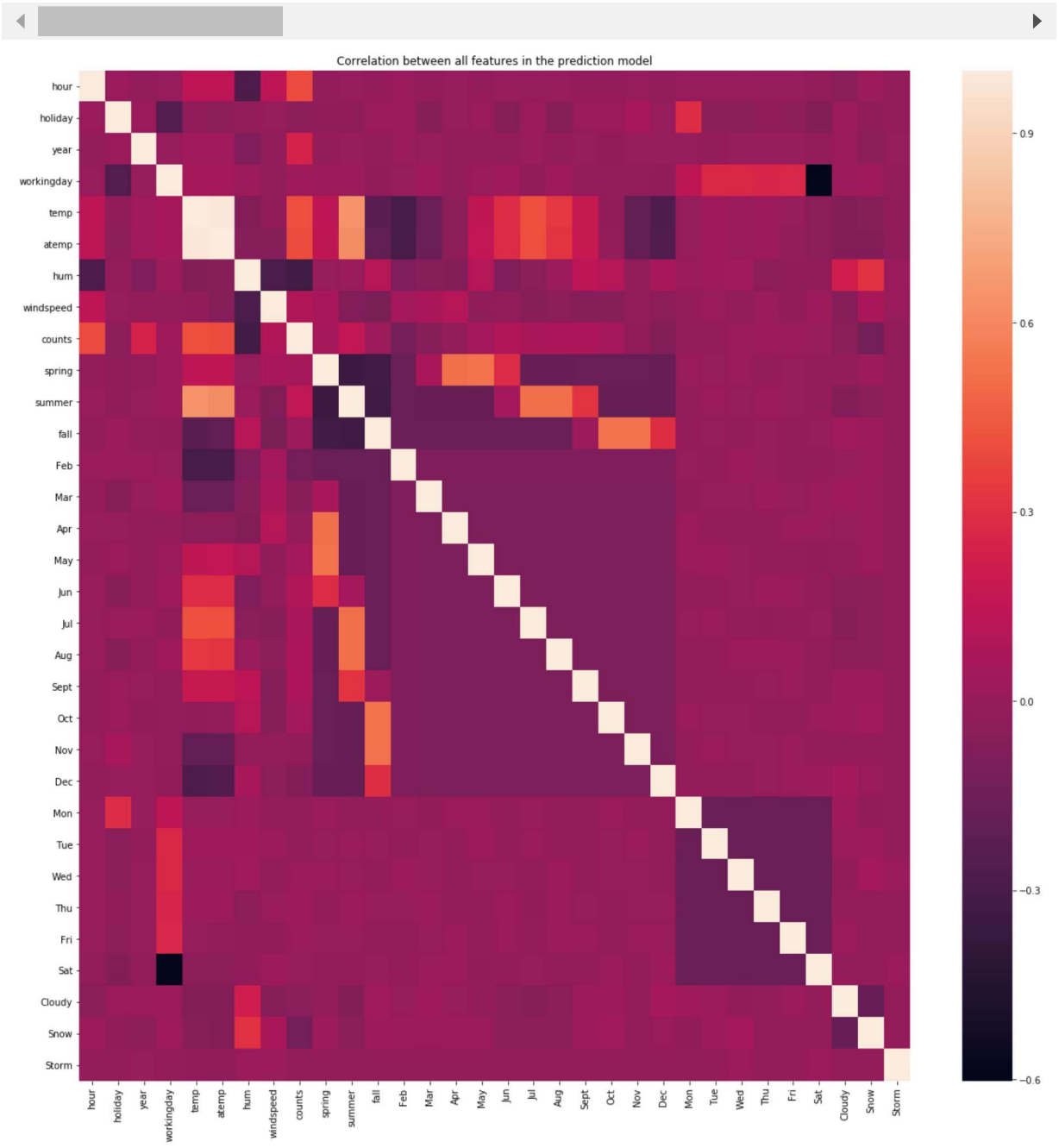
3.5 Calculate the Pearson correlation ....

In [647]:
```
# store the correlation matrix
corr = BSS_train.corr(method = 'pearson')

# initialize and create the heatmap
fig, ax = plt.subplots(figsize=(20,20))
ax = sns.heatmap(corr)

# set the title
ax.set_title("Correlation between all features in the prediction model");

# print out the variables which have a positive correlation with bike rentals
print("The following variables have a positive correlation with bike rentals:"
)

# use the given conditional to identify the variables
for row in corr.index[corr['counts'] > 0]:
    if row != 'counts':
        print("   {0}".format(row))

# print the correlation matrix
corr
```

The following variables have a positive correlation with bike rentals:
- hour
- year
- workingday
- temp
- atemp
- windspeed
- spring
- summer
- fall
- May
- Jun
- Jul
- Aug
- Sept
- Oct
- Wed
- Thu
- Fri
- Sat

Out[647]:

	hour	holiday	year	workingday	temp	atemp	hum
hour	1.000000	0.005028	-0.010900	0.002024	0.140745	0.136311	-0.274146
holiday	0.005028	1.000000	0.011641	-0.253523	-0.026372	-0.030072	-0.010085
year	-0.010900	0.011641	1.000000	-0.000840	0.038813	0.037635	-0.086934
workingday	0.002024	-0.253523	-0.000840	1.000000	0.056547	0.055666	0.020164
temp	0.140745	-0.026372	0.038813	0.056547	1.000000	0.987408	-0.071756
atemp	0.136311	-0.030072	0.037635	0.055666	0.987408	1.000000	-0.053984
hum	-0.274146	-0.010085	-0.086934	0.020164	-0.071756	-0.053984	1.000000
windspeed	0.139770	-0.000291	-0.008300	-0.009580	-0.018421	-0.058252	-0.286629
counts	0.394167	-0.028252	0.243886	0.029534	0.406155	0.401119	-0.328232
spring	-0.003675	-0.026538	-0.003334	0.022044	0.142863	0.151023	0.002175
summer	0.003857	-0.025676	0.002648	0.018794	0.645391	0.622339	0.010583
fall	-0.006450	0.016316	-0.013063	-0.016104	-0.220062	-0.200820	0.122103
Feb	0.005606	0.010534	0.007331	0.001543	-0.297576	-0.296865	-0.086556
Mar	-0.003834	-0.052837	-0.000196	0.026703	-0.166973	-0.163844	-0.057093
Apr	0.003251	0.003616	-0.002221	-0.006576	-0.042255	-0.032615	-0.067723
May	-0.003347	0.007875	-0.006394	0.011599	0.154599	0.159369	0.105133
Jun	0.001247	-0.052197	-0.001702	0.030723	0.292131	0.284454	-0.084841
Jul	-0.001340	0.010926	0.006461	-0.015422	0.412723	0.408329	-0.049786
Aug	0.004635	-0.052886	-0.003298	0.034786	0.335480	0.311650	0.010451
Sept	-0.004525	0.005165	-0.000132	-0.009376	0.186166	0.180722	0.138100
Oct	-0.002319	0.010816	-0.016538	-0.012953	-0.015389	-0.003395	0.105310
Nov	0.001024	0.064028	0.000390	-0.011617	-0.200615	-0.190762	-0.008004
Dec	-0.008559	0.003484	-0.000215	-0.019465	-0.276243	-0.267651	0.063829
Mon	0.002398	0.288895	0.003327	0.147598	-0.000895	0.004153	0.013808
Tue	0.002814	-0.045881	-0.003703	0.268793	0.025038	0.027500	0.032100
Wed	0.000877	-0.046017	0.001554	0.272720	0.016462	0.014942	0.042039
Thu	-0.000289	-0.023697	0.003746	0.260472	0.021504	0.021189	-0.042152
Fri	-0.000661	-0.027916	-0.000422	0.263344	0.000682	-0.007918	-0.024094
Sat	-0.007787	-0.071661	-0.010093	-0.602438	-0.036657	-0.037245	-0.019439
Cloudy	-0.050553	0.007805	0.013921	0.022523	-0.071525	-0.068782	0.221191
Snow	0.020257	-0.021242	-0.038232	0.034080	-0.062334	-0.068844	0.309075
Storm	-0.005253	-0.002083	-0.000110	-0.004647	-0.019716	-0.021537	0.016521



temp and atemp have a correlation coefficient of 0.987408. No other non-trivial correlations are greater than 0.7.

Question 4: Multiple Linear Regression

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

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

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

Answers

4.1 Use statsmodels to fit a ...

In [648]:
```
def build_model1_design_mat(df):
    """ Builds a desing matrix for the desired model

    Args:
        df: The matrix to be manipulated

    Returns:
        y: The response variables for the design matrix
        sm.add_constnat(design_mat): the fitted design matrix
    """
    #first, split off the target
    y = df['counts']

    ## Build a design matrix for this model
    design_mat = df.copy()

    # drop counts from the design matrix
    design_mat= design_mat.drop(columns="counts")

    # return the values
    return y, sm.add_constant(design_mat)

# initialize the training values needed for the model
y_train, train_design = build_model1_design_mat(BSS_train)

# create a fitted model with the design matrix
fitted_sm_ols = OLS(endog = y_train, exog = train_design).fit()

# Initialize the test values needed for the model
y_test, test_design = build_model1_design_mat(BSS_test)

# store the predicted y-values for R^2 score calculation
y_train_predict = fitted_sm_ols.predict(train_design)
y_test_predict = fitted_sm_ols.predict(test_design)

# Store and print the R^2 values
train_r2 = r2_score(y_train, y_train_predict)
test_r2 = r2_score(y_test, y_test_predict)
print("\033[1mR^2\nTraining:\033[0m {0}\n\033[1mTest:\033[0m {1}".format(train
_r2, test_r2))
```

```
R^2
Training: 0.4065387827969087
Test: 0.40638554757102263
```

4.2 Find out which of estimated coefficients ...

In [649]:
```
fitted_sm_ols.summary()
```

Out[649]:

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:	Tue, 17 Jul 2018	**Prob (F-statistic):**	0.00
Time:	23:52:33	**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

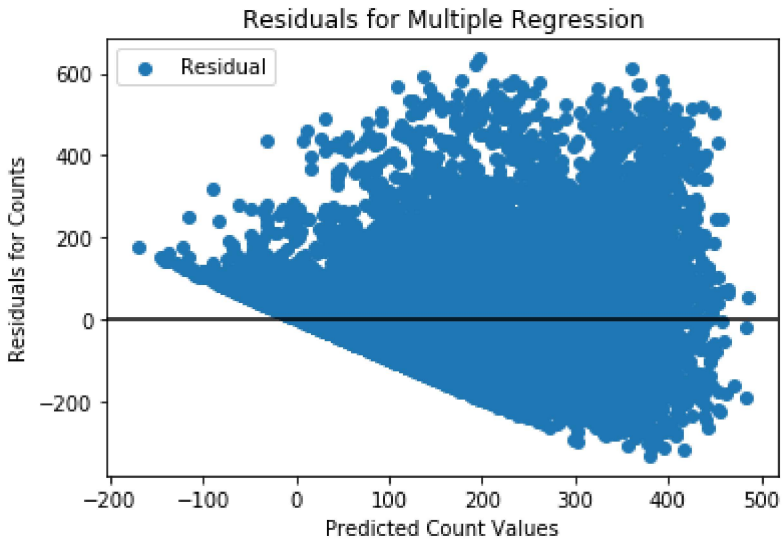
Omnibus:	2831.359	**Durbin-Watson:**	0.755
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5657.789
Skew:	1.224	**Prob(JB):**	0.00
Kurtosis:	4.943	**Cond. No.**	1.17e+16

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

The predictors hour, holiday, year, workingday, temp, hum, windspeed, spring, summer, fall, Jun, Jul, Sat, Cloudy, and Snow are all statistically significant based on the p-value criteria. It should be noted that there are multiple warnings: standard errors in the summary table are calculated assuming that the covariance matrix of the errors is correctly specified. Also, the smallest eigenvalue is quite large (1.4e-20), which might indicate that there are strong multicollinearity problems or that thte design matrix is singular.

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

In [650]:
```
# Create the residual plot
residual = y_train - y_train_predict
plt.scatter(y_train_predict, residual, label="Residual")
plt.axhline(0, color='k')


# add labels
plt.title("Residuals for Multiple Regression")
plt.ylabel("Residuals for Counts")
plt.xlabel("Predicted Count Values")
plt.legend();
```



The plot reveals a non linear relationship since there is a strong pattern in the residual plot. The variance of the error terms predict a negative count value when there appears to be an asymptote at 0. As the count values increase, the variance of the error terms increase.

Question 5: Subset Selection

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

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

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

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

Answers

5.1 Implement forward step-wise ....

In [651]:
```
# initialize a dataframe storing all of the predictors
complete_list_predictors = BSS_train.copy().drop(["counts"], axis=1)


def get_bic(X_train, y_trn = y_train) -> float:
    """Returns the BIC score

    Args:
        X_train: the matrix of predictor variables
        y_trn: the response variables, default set to y_train

    Returns:
        the calculated BIC value for the given data
    """

    # add constant to data matrix
    X_train = sm.add_constant(X_train)

    # return the BIC value
    return OLS(endog = y_trn, exog = X_train).fit().bic

# initialize a dataframe storing the remaining predictors
remaining_predictors = complete_list_predictors.copy()

# initialize variables
best_model_bic = np.inf
best_model = []
best = []

# iterate through for each predictor in complete_list_predictors
for i in range(len(complete_list_predictors.columns)):

    # reset the best bic for i number of predictors to infinity
    best_bic = np.inf

    # iterate through for each remaining predictor not yet selected
    for col in range(len(remaining_predictors.columns)):

        # Get the BIC value for the selected predictor list with the current p
redictor added
        curr_col = remaining_predictors.columns[col]
        curr_bic = get_bic(complete_list_predictors[best_model + [curr_col]])

        # If the BIC is lower than previous BIC's involving the same number of
 predictors,
        # store the BIC and variable name
        if curr_bic < best_bic:
            best_bic = curr_bic
            best_col = curr_col

    # After iterating through the remaining predictors, store the best predict
or found in the selected subset
    best_model.append(best_col)

    # Drop the newly selected predictor from the datarame of remaining predict
ors
```

```
    remaining_predictors = remaining_predictors.drop(columns= best_col)

    # If the BIC for the model with col amount of predictors is the lowest so
 far
    if  best_bic < best_model_bic:

        # store the model and bic as the best overall
        best = best_model.copy()
        best_model_bic = best_bic

# print the overall results
print("The forward step-selection method selects the following subset of predi
ctors:\n {0}".format(best))
```

```
The forward step-selection method selects the following subset of predictors:
 ['temp', 'hour', 'year', 'hum', 'fall', 'Jul', 'Snow', 'Aug', 'Jun', 'holida
y', 'spring']
```

5.2 Do these methods eliminate ...

This method eliminated atemp, one of the two collinear predictors. This may be because the stepwise method's predictor by predictor approach illuminates the lack of utility of the the second collinear predictor once the first one is added to the subset.

5.3 In each case, fit linear regression ...

In [652]:
```
# Store an X_train of the subset
X_train = sm.add_constant(BSS_train[best])

# Fit the model with the subset data
subset_model = OLS(endog = BSS_train["counts"], exog = X_train).fit()

# Print the test R^2
test_r2 = r2_score(BSS_test["counts"], a.predict(sm.add_constant(BSS_test[best
]])))
print("\033[1mTest R^2:\033[0m {0}".format(test_r2))
```

```
Test R^2: 0.40469087705369655
```

The fitted model using all predictors has an R^2 value of ~ 0.406, while the fitted model using the predictors from the stepwise selection has an R^2 value of ~0.405. The R^2 from the stepwise selection is much better because it uses much less predictors and has an almost identical R^2 value.

Question 6: Polynomial Regression

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

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

In [653]:
```
def build_model2_design_mat(df):
    """Builds a design matrix and separates the response variables

    Args:
        df: the data to be split and used

    Returns:
        y: the response variables
        sm.add_constant(design_mat): the design matrix of predictors
    """
    # first,  split off the target
    y = df['counts'].copy()

    # Build a design matrix for this model
    design_mat = df.copy().drop(["counts"], axis=1)

    # For the determined continuous variables, add quadratic, cubic and quarti
c terms
    for col in ["temp", "atemp", "hum", "windspeed"]:
        for i in [2, 3, 4]:
            design_mat[("{0}_{1}".format(col, i))] = design_mat[col] ** i

    # return the values
    return y, sm.add_constant(design_mat)

# store a copy of BSS_train
train_df = BSS_train.copy()

# initialize the training values needed for the polynomial regression model
Y, BSS_train_polynomials = build_model2_design_mat(train_df)

# Create a fiited model for the training data
poly_model = sm.OLS(Y, BSS_train_polynomials).fit()

# initialize the test values needed for the polynomial regression model
y, BSS_tests = build_model2_design_mat(BSS_test.copy())

# store and print R^2 values
test_r2 = r2_score(y, poly_model.predict(BSS_tests))
train_r2 = poly_model.rsquared
print("\033[1mR^2\nTraining:\033[0m {0}\n\033[1mTest:\033[0m {1}".format(train
_r2, test_r2))

# Print the summary for the polynomial regression model
poly_model.summary()
```

```
R^2
Training: 0.42230805166587093
Test: 0.4202791276225242
```

Out[653]:

OLS Regression Results

Dep. Variable:	counts	**R-squared:**	0.422
Model:	OLS	**Adj. R-squared:**	0.421
Method:	Least Squares	**F-statistic:**	241.2
Date:	Tue, 17 Jul 2018	**Prob (F-statistic):**	0.00
Time:	23:52:47	**Log-Likelihood:**	-88119.
No. Observations:	13903	**AIC:**	1.763e+05
Df Residuals:	13860	**BIC:**	1.766e+05
Df Model:	42		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-170.9866	38.451	-4.447	0.000	-246.356	-95.617
hour	7.0982	0.183	38.815	0.000	6.740	7.457
holiday	-19.9687	6.527	-3.059	0.002	-32.763	-7.175
year	79.1663	2.383	33.217	0.000	74.495	83.838
workingday	11.5480	2.721	4.244	0.000	6.214	16.882
temp	-98.4860	500.457	-0.197	0.844	-1079.449	882.477
atemp	972.1010	562.472	1.728	0.084	-130.421	2074.623
hum	808.4325	233.754	3.458	0.001	350.243	1266.622
windspeed	-4.8944	71.765	-0.068	0.946	-145.564	135.776
spring	42.1280	7.348	5.733	0.000	27.724	56.532
summer	21.2722	8.703	2.444	0.015	4.213	38.331
fall	67.7929	7.434	9.120	0.000	53.222	82.364
Feb	6.6054	6.034	1.095	0.274	-5.222	18.433
Mar	12.1869	6.872	1.773	0.076	-1.283	25.657
Apr	-18.8035	9.975	-1.885	0.059	-38.356	0.749
May	-24.9693	10.567	-2.363	0.018	-45.681	-4.257
Jun	-69.8240	10.710	-6.519	0.000	-90.817	-48.831
Jul	-92.0230	12.059	-7.631	0.000	-115.660	-68.386
Aug	-64.1045	11.886	-5.393	0.000	-87.404	-40.805
Sept	-7.2505	10.710	-0.677	0.498	-28.243	13.742
Oct	8.3218	10.013	0.831	0.406	-11.305	27.948

Nov	-0.5760	9.667	-0.060	0.952	-19.525	18.373
Dec	9.2113	7.752	1.188	0.235	-5.984	24.407
Mon	-4.8463	2.945	-1.645	0.100	-10.619	0.927
Tue	-6.9005	3.172	-2.175	0.030	-13.118	-0.683
Wed	1.0496	3.148	0.333	0.739	-5.122	7.221
Thu	-2.2895	3.153	-0.726	0.468	-8.469	3.890
Fri	4.5660	3.159	1.445	0.148	-1.626	10.758
Sat	15.5005	4.330	3.580	0.000	7.013	23.988
Cloudy	7.2913	2.872	2.539	0.011	1.662	12.921
Snow	-24.9330	4.967	-5.019	0.000	-34.670	-15.196
Storm	31.0141	97.153	0.319	0.750	-159.419	221.447
temp_2	-3464.1586	1723.151	-2.010	0.044	-6841.767	-86.551
temp_3	8584.3969	2391.413	3.590	0.000	3896.904	1.33e+04
temp_4	-5050.0400	1143.548	-4.416	0.000	-7291.548	-2808.532
atemp_2	-373.5642	2015.394	-0.185	0.853	-4324.009	3576.881
atemp_3	-605.5645	2929.063	-0.207	0.836	-6346.924	5135.795
atemp_4	114.8517	1469.938	0.078	0.938	-2766.425	2996.128
hum_2	-2395.6325	698.409	-3.430	0.001	-3764.609	-1026.656
hum_3	2306.1400	874.057	2.638	0.008	592.870	4019.410
hum_4	-767.2521	386.578	-1.985	0.047	-1524.997	-9.507
windspeed_2	1221.7297	507.532	2.407	0.016	226.899	2216.561
windspeed_3	-3680.9342	1262.120	-2.916	0.004	-6154.861	-1207.008
windspeed_4	2662.6197	987.316	2.697	0.007	727.347	4597.892

Omnibus:	2862.081	**Durbin-Watson:**	0.771
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	5852.821
Skew:	1.224	**Prob(JB):**	0.00
Kurtosis:	5.028	**Cond. No.**	1.16e+16

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.88e-26. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

According to the t-tests, all of the 2nd, 3rd, and 4th degree polynomial terms are statistically significant at a level of 5% except atemp2, atemp3 and atemp4. The first degree polynomial terms for temp, atemp, and windspeed are also not statistically significant. It should be noted, however, that there are warnings for potential strong mulitcollinearity problems.

Written Report to the Administrators

Question 7

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

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

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

Answers

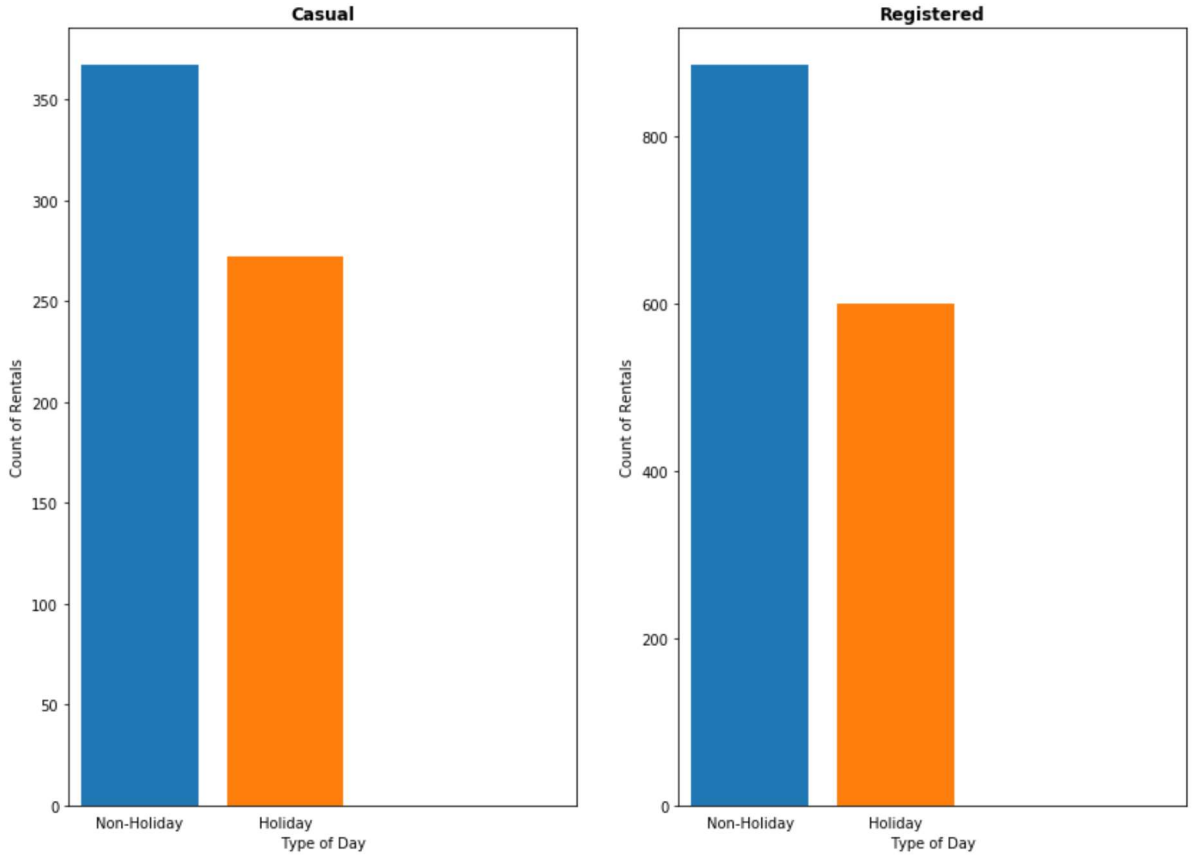
7

The polynomial regression model can be used to forecast the bike sharing demand. The time of day, if it's a holiday, day of week (Tuesday or Saturday), season and weather are important variables in predicting the bike ridership. In Figure 1, we can see that bike ridership decreases when it is a since a majority of our riders are commuters. According to Figure 2, the time of day has an effect on the ride counts with the peaks in bike rentals being during the morning and evening commute times. As seen in Figure 3, bike ridership appears to be the lowest during winter season and during bad weather, when there is snow or a thunderstorm.

A significant proportion of our riders on holidays are casual rentals and therefore to increase revenues, we would want these casual rentals to become registered rentals and buy a pass. Therefore, holidays would be a good time to increase promotions of the monthly and annual passes. Looking at the distribution of bike rentals over the week in Figure 3, the casual bike rentals appear to always peak on the weekends. This could be due to tourists who are visiting or also locals who don't have the membership since they do not use the bikes often. A promotion strategy would be to offer 10% off a casual renter's next ride if they are using the bikes for the first time in order to encourage them to use the bikes again. Also, there appears to be a certain population that may not be using bikes to commute during the weekdays but instead use them on weekends. In order to encourage this subset to become registered rentals, we could offer them a biweekly month pass where a rider is able to use twice a week for the month.

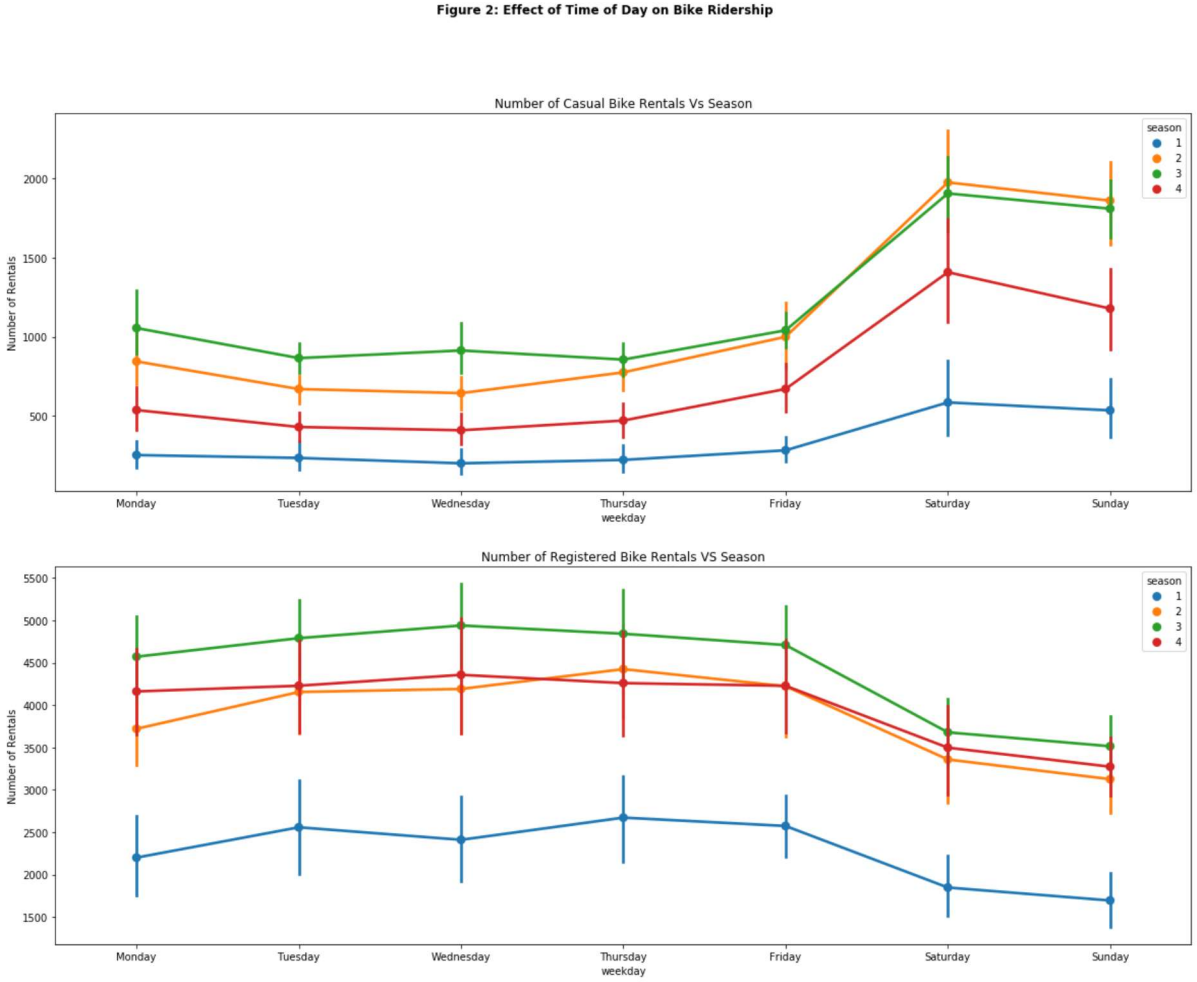
In [654]:
```
# initialize figure
fig, ax = plt.subplots(1,2, figsize=(14,10))

# for holiday and non-holidays, barplot the casual and registered values of th
at group
for cur_holi, cur_df in bikes_df.groupby('holiday'):
    ax[0].bar(cur_holi, cur_df['casual'])
    ax[1].bar(cur_holi, cur_df['registered'])

# get the name of each position and set the x ticks
holidays = ["", "Non-Holiday", "", "Holiday"]
ax[0].set_xticks(range(len(holidays)), holidays)
ax[1].set_xticks(range(len(holidays)), holidays)
ax[0].xaxis.set_ticks_position('none')
ax[1].xaxis.set_ticks_position('none')
ax[0].set_xticklabels(holidays)
ax[1].set_xticklabels(holidays)

# fill in the remaining labels
ax[0].set_xlabel("Type of Day")
ax[1].set_xlabel("Type of Day")
ax[0].set_ylabel("Count of Rentals")
ax[1].set_ylabel("Count of Rentals")
ax[0].set_title("Casual", fontweight="bold")
ax[1].set_title("Registered", fontweight="bold")
fig.suptitle("Figure 1: Bike Ridership on Holidays");
```

Figure 1: Bike Ridership on Holidays



In [655]:
```
# Print figure 2
fig2.suptitle("Figure 2: Effect of Time of Day on Bike Ridership", fontweight
= "bold")
fig2
```

Out[655]:



In [656]:
```
# Print figure 3
fig3.suptitle("Figure 3: Bike Ridership According to Seasons")
fig3
```

Out[656]: