CS109A Introduction to Data Science:

Homework 3 - Forecasting Bike Sharing Usage

Harvard University
Fall 2018
Instructors: Pavlos Protopapas, Kevin Rader

In [4]:
```
#RUN THIS CELL
import requests
from IPython.core.display import HTML
styles = requests.get("https://raw.githubusercontent.com/Harvard-IACS/2018-CS109A
HTML(styles)
```

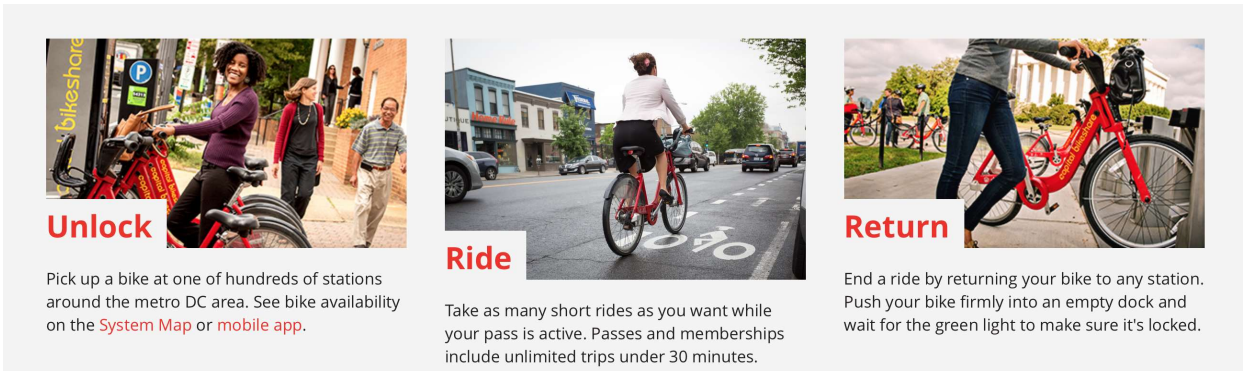
Out[4]:

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:

Type *Markdown* and LaTeX: α^2



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 [5]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt

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

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

from pandas.plotting import scatter_matrix

import seaborn as sns


%matplotlib inline
```

In [6]:
```
# define figure defaults
mpl.rc('axes', labelsize=14, titlesize=14)
mpl.rc('figure', figsize=[8,6], titlesize=16)
mpl.rc('legend', fontsize=14)
mpl.rc('lines', linewidth=2, color='k')
mpl.rc('xtick', labelsize=14)
mpl.rc('ytick', labelsize=14)
```

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 day (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 [7]:
```
# read in data
bikes_df = pd.read_csv("data/BSS_hour_raw.csv")
bikes_df.describe()
```

Out[7]:

	season	hour	holiday	weekday	workingday	weather	
count	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.000000	17379.
mean	2.501640	11.546752	0.028770	3.003683	0.682721	1.425283	0.
std	1.106918	6.914405	0.167165	2.005771	0.465431	0.639357	0.
min	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.
25%	2.000000	6.000000	0.000000	1.000000	0.000000	1.000000	0.
50%	3.000000	12.000000	0.000000	3.000000	1.000000	1.000000	0.
75%	3.000000	18.000000	0.000000	5.000000	1.000000	2.000000	0.
max	4.000000	23.000000	1.000000	6.000000	1.000000	4.000000	1.

There are a few issues with the data:

1. `workingday` is somewhat redundant, since it seems just to specify weekday 1-5
2. `tempurature` is supposed to be in degrees celcius, but the minand max values are 0-1, suggesting the temperature is normalized to some range
3. columns such as holiday or workingday are binary variables and could be `boolean` rather than `int64`
4. the `dteday` column is stored as a pandas object rather than datetime object

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

In [26]:
```
bikes_df.dteday = pd.to_datetime(bikes_df.dteday, yearfirst=True)
```

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 day (this is the response variable for later).

In [10]:
```
bikes_df['year'] = bikes_df.dteday.apply(lambda x: x.year)
bikes_df['month'] = bikes_df.dteday.apply(lambda x: x.month)
bikes_df['count'] = [row["casual"] + row["registered"] for i, row in bikes_df.ite
```

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

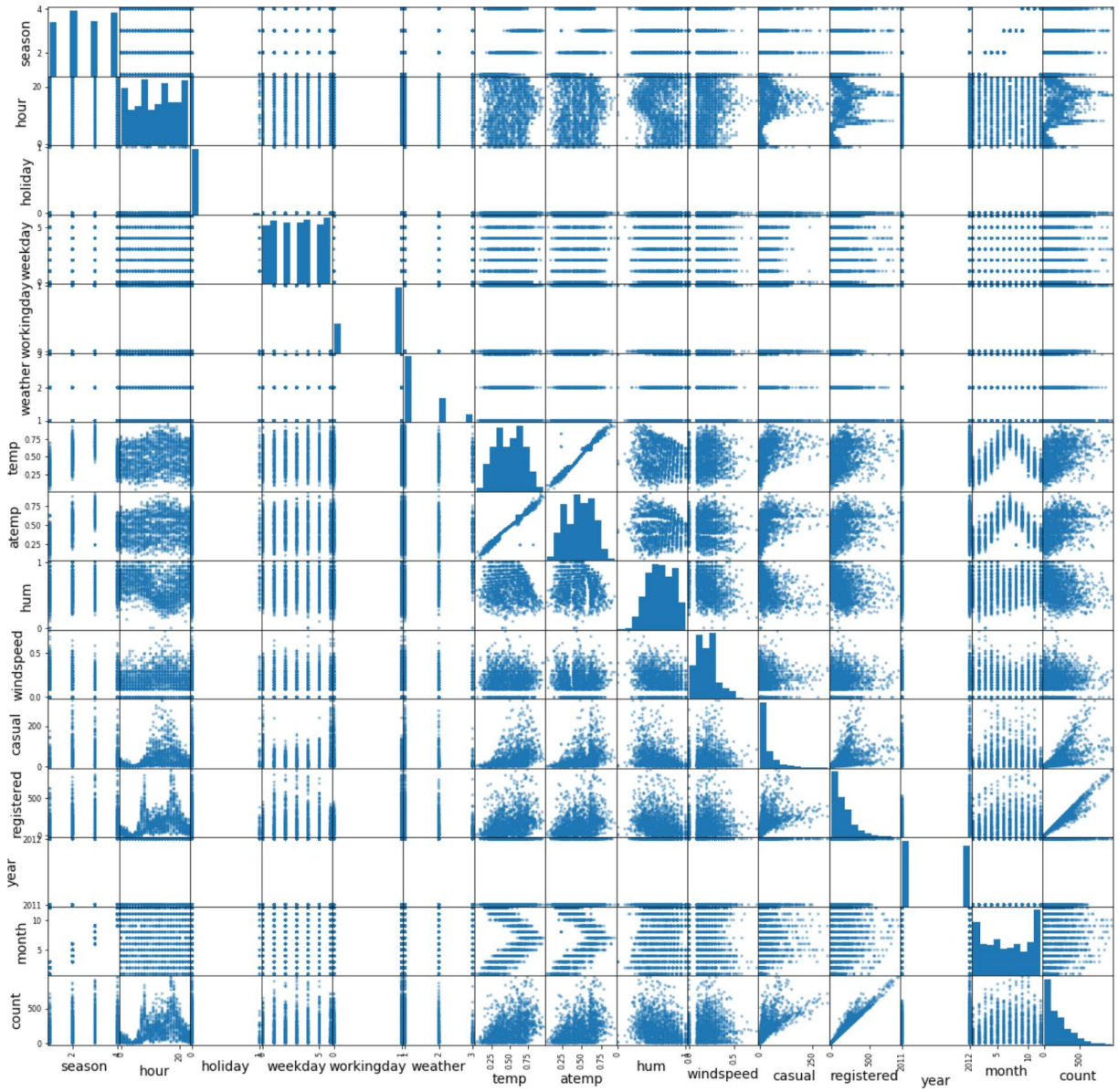
percentile or 1.5 times the IQR below the 25th percentiles. If you see any outliers, identify those dates and investigate if they are a chance occurence, an error in the data collection, or a significant event (an online search of those date(s) might help).

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]

In [12]:
```
# get a random subset of the rows
rand_sample = bikes_df.sample(frac=0.1)
```

In [13]:
```
scatter_matrix(rand_sample, figsize=(20,20))
plt.show()
```



Positive correlations:

1. temp and atemp (these contain nearly the same information)
2. Casual/registered users and total count
3. Casual users and registered users
4. Temp/atemp and casual/registered/total counts

Negative correlations:

1. Humidity and casual/registered/total counts
2. Windspeed and casual/registered/total counts

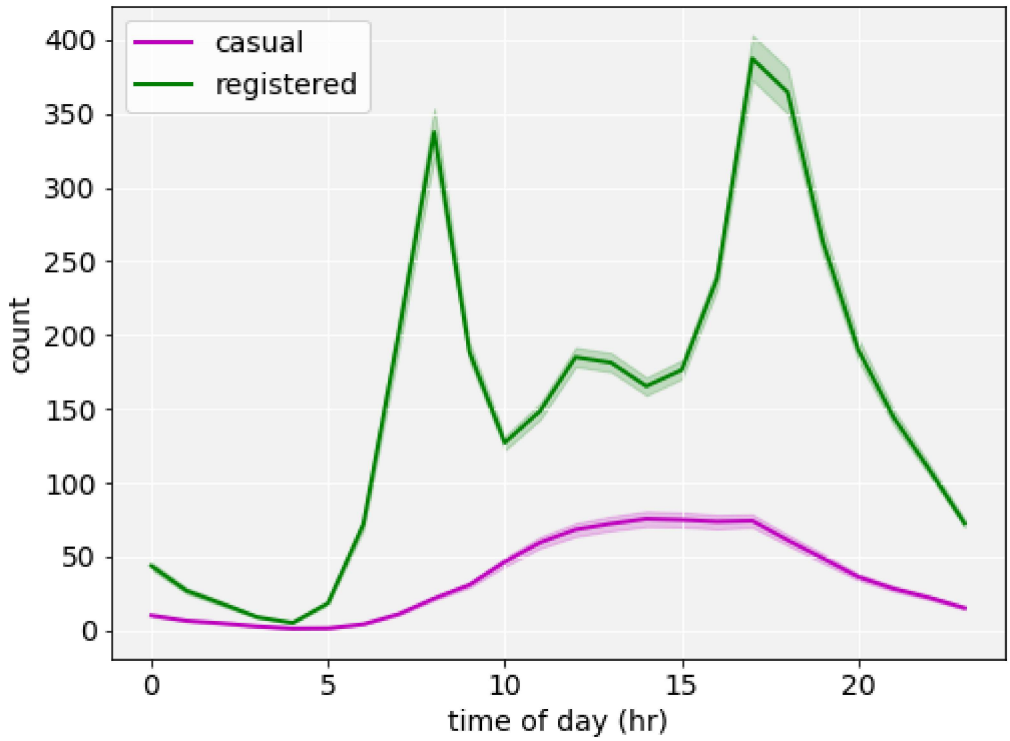
Other relationships:

1. casual/registered/total counts vary by hour

2. temp/atemp/humidity vary by hour
3. temp/atemp vary by month

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

In [25]:
```
ax = sns.lineplot(x="hour", y="casual", data=bikes_df, label="casual", color='m')
sns.lineplot(x="hour", y="registered", data=bikes_df, label="registered", color='
plt.xlabel('time of day (hr)')
plt.ylabel('count')
plt.legend()
plt.legend()
plt.grid(color='w')
plt.gca().set_facecolor([.95,.95,.95])
```

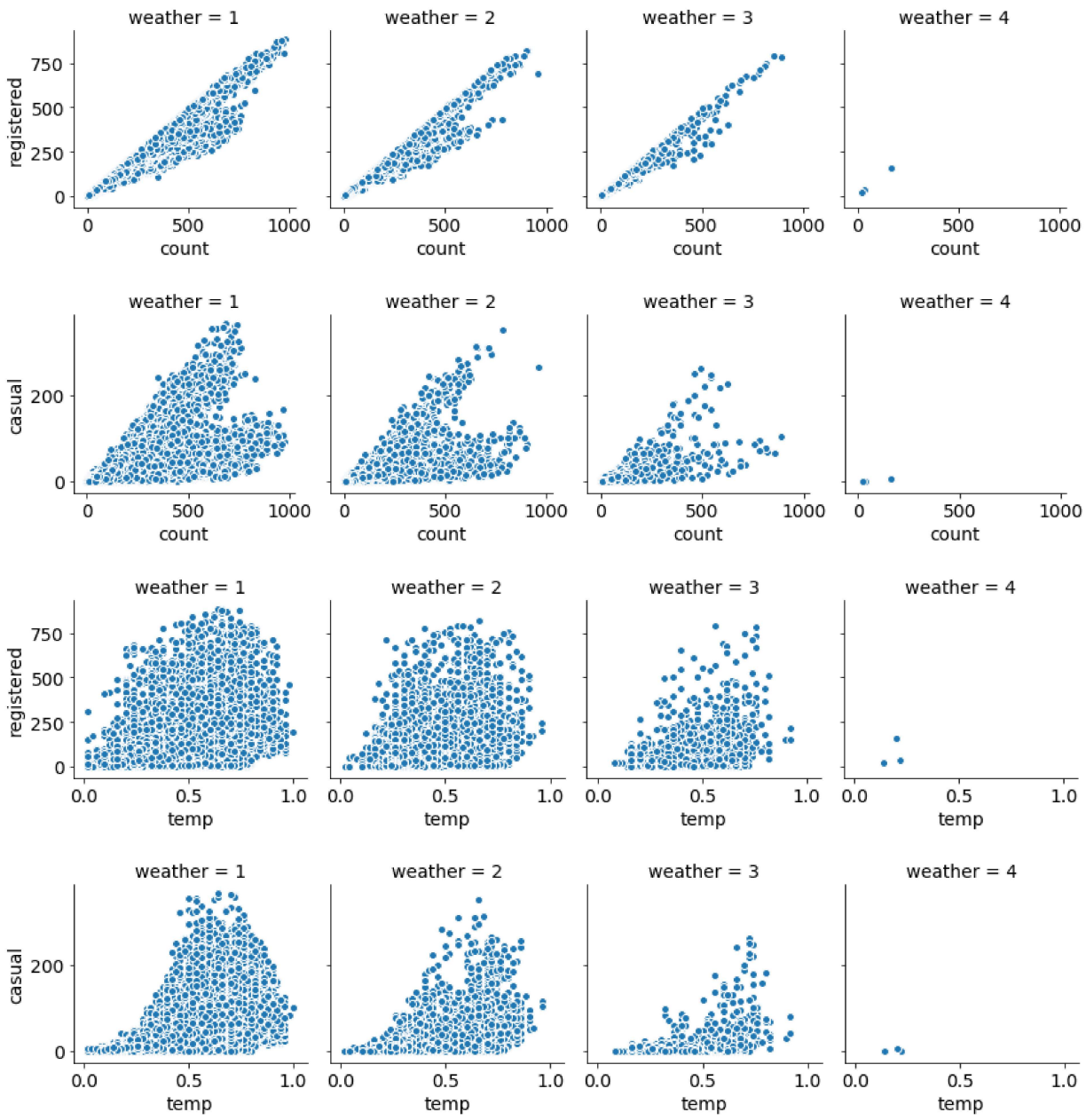


Registered users seem to peak in the morning and late afternoon, as is consistent with work commute. Casual users peak in the midday, suggesting that casual users likely use the bikes for joy rides.

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

In [15]:
```
def facet_stack(df, col, x=None, rows=None):

    if rows is None:
        rows = df.columns
    if x is None:
        x = df.columns

    axes_handles = np.empty((0,0))
    for i in x:
        for r in rows:
            g = sns.FacetGrid(df, col=col)
            g.map(plt.scatter, i, r, edgecolor='w')
            axes_handles = np.append(axes_handles, g.axes)

    return(axes_handles)
```

In [16]:
```
ah = facet_stack(bikes_df, 'weather', x=['count','temp'], rows=['registered','cas
```



The trends don't really seem to change qualitatively by weather category, but there is a clear reduction in all bike rides with increasingly inclimate weather.

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` .

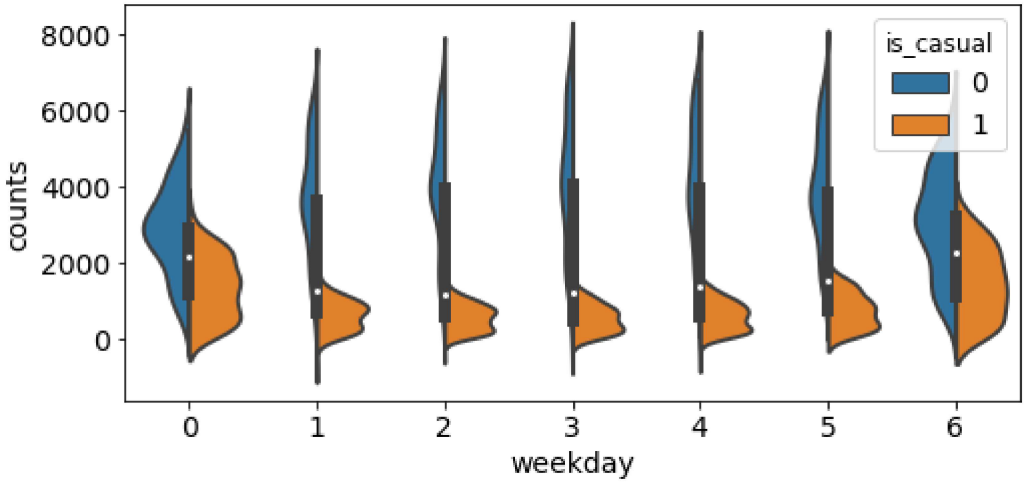
Make a plot showing the *distribution* of the number of casual and registered riders on each day of the week.

In [17]:
```
norm = preprocessing.MinMaxScaler()

# group the data by dteday to get unique dates
by_dteday = bikes_df.groupby("dteday")

# create dictionary with new measurements
by_day_dict = {
    "dteday": by_dteday.dteday.unique(),
    "weekday": by_dteday.weekday.mean(),
    "weather": by_dteday.weather.max(),
    "season": by_dteday.season.mean(),
    "temp": np.squeeze(norm.fit_transform(by_dteday.temp.mean().values.reshape(-1
    "atemp": np.squeeze(norm.fit_transform(by_dteday.atemp.mean().values.reshape(
    "windspeed": np.squeeze(norm.fit_transform(by_dteday.windspeed.mean().values.
    "hum": np.squeeze(norm.fit_transform(by_dteday.hum.mean().values.reshape(-1,1
    "casual": by_dteday.casual.sum(),
    "registered": by_dteday.registered.sum(),
    "counts": by_dteday["count"].sum(),

}

# intialize new data frame from dictionary
bikes_by_day = pd.DataFrame.from_dict(by_day_dict)
```

In [15]:
```
def hist_bygroup(df, grp_name: str, cols, figsize=None):

    # group data
    grp_df = df.groupby(grp_name)

    if type(cols) is not list:
        cols = list(cols)

    # get subplot dimensions
    ncol = len(grp_df.groups.keys())
    sp_cols = np.ceil(np.sqrt(ncol))
    sp_rows = np.ceil(ncol/sp_cols)

    if figsize is not None:
        plt.figure(figsize=figsize)

    i=1
    for key, grp in grp_df:
        plt.subplot(sp_rows,sp_cols,i)
        grp = grp_df.get_group(key)

        for col in cols:
            plt.hist(grp[col], label=col, alpha=0.5)

        plt.legend()
        plt.xlabel("fraction total")
        if key is not str:
            key = str(key)
        plt.title("%s = %s"% (grp_name,key))
        i+=1

    plt.subplots_adjust(hspace=0.5)
```

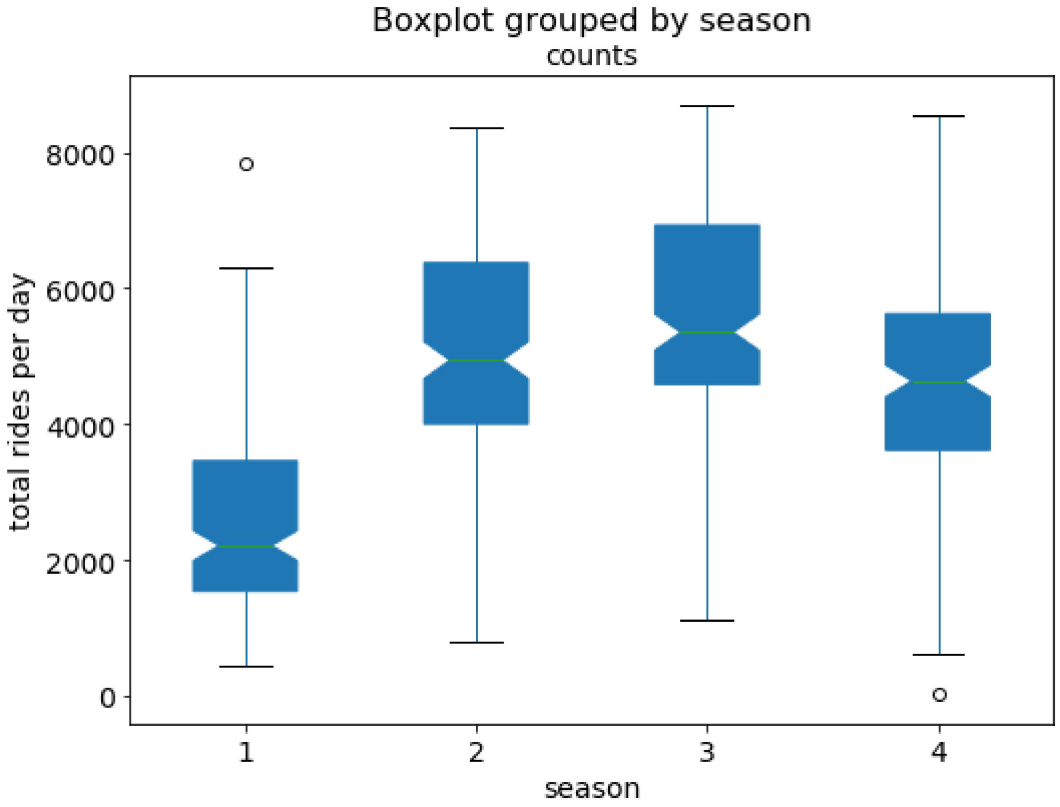
In [28]:
```
# create new temp df for visualization
counts = bikes_by_day[["casual","registered"]].values
weekday = bikes_by_day[["weekday","weekday"]].values.flatten(order="F").reshape(-
is_casual = np.concatenate((\
                np.full((counts.shape[0],1),True),\
                np.full((counts.shape[0],1),False)),\
                axis=1).flatten(order="F")
counts = counts.flatten(order="F")
vis_df = pd.DataFrame(data = \
                      np.concatenate((counts.reshape(-1,1),is_casual.reshape(-1,1
                      columns = ["counts","is_casual","weekday"])
```

In [30]:
```
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
sns.violinplot(x='weekday', y="counts", hue="is_casual", split=True, data=vis_df)
```



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 [31]:
```
#g = sns.FacetGrid(bikes_by_day, col='season')
#g.map(plt.hist, "counts")
bp = bikes_by_day.boxplot(column='counts',by='season', notch=True, patch_artist=T
bp.set_xlabel("season")
bp.set_ylabel("total rides per day");
```



In [19]:
```
from scipy.stats import iqr
by_season = bikes_by_day.groupby("season")
season_count_iqrs = by_season['counts'].apply(iqr).values
season_count_percentiles = by_season['counts'].apply(lambda x: np.percentile(x,(2
```

In [20]:
```
outlier_dates = []

i=0
for key, season in by_season:
    grp = by_season.get_group(key)
    lower_bound = season_count_percentiles[i][0] - season_count_iqrs[i]*1.5
    upper_bound = season_count_percentiles[i][1] + season_count_iqrs[i]*1.5
    is_outlier = np.array([[\
                            grp.counts.values < lower_bound],\
                           [grp.counts.values > upper_bound]\
                          ]).reshape(-1,2).squeeze()
    is_outlier = is_outlier.any(axis=1)
    outlier_dates.append(grp.dteday[is_outlier])
    i+=1
```

In [21]:
```
for date in outlier_dates:
    if date.values.size > 0:
        print(date.values)
```

```
[array(['2012-12-24T00:00:00.000000000'], dtype='datetime64[ns]')]
[array(['2011-11-24T00:00:00.000000000'], dtype='datetime64[ns]')]
```

There are two outliers in the dataset:

1. Nov. 24, 2011 (Thanksgiving)
2. Dec. 24, 2012 (Christmas Eve)

Since these dates are holidays, these are most likely genuine outliers and not recording errors.

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 [32]:
```
bikes_df_recoded = pd.get_dummies(bikes_df, \
                                  columns=['season', 'month', 'weekday', 'weather
                                  drop_first=True)
```

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

In [23]:
```
train_df, test_df = train_test_split(bikes_df, test_size = 0.2, stratify=bikes_df
```

Using the `stratify` keyword-argument tells the train_test_split function to ensure that our categories of month are all equally represented in both the training and test data. This is particularly important if some of the categories are rare, meaning that its likely that a specific month(s) could be over or under-represented in the split data sets. This could bias our coefficient estimates our estimation of the goodness of fit.

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 [34]:
```
def intialize_BSS_dfs():

    train_df = pd.read_csv("data/BSS_train.csv")
    test_df = pd.read_csv("data/BSS_test.csv")
    train_df = train_df.drop(columns=['Unnamed: 0', 'dteday', 'casual', 'register
    test_df = test_df.drop(columns=['Unnamed: 0', 'dteday', 'casual', 'registered

    return(train_df, test_df)
```

In [35]:
```
train_df, test_df = intialize_BSS_dfs()
```

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 [37]:
```
def train_test_score(train_df, test_df):

    train_mat = train_df.drop("counts", axis=1).values
    test_mat = test_df.drop("counts", axis=1).values

    X = sm.add_constant(train_mat)
    model = OLS(train_df.counts.values, X)
    model = model.fit()
    pred_counts = model.predict(sm.add_constant(test_mat))
    print('training data:\t r_squared = %.2f' % model.rsquared)
    print('test data:\t r_squared = %.2f' % r2_score(test_df.counts.values, pred_

    return(model)
```

In [38]:
```
bike_ols = train_test_score(train_df, test_df)
```

```
training data:   r_squared = 0.41
test data:       r_squared = 0.41
```

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

In [39]:
```
def coefficient_summary(df, model):
    p_vals = model.pvalues[1:]
    col_names = df.drop("counts", axis=1).columns
    coefs = model.params[1:]

    param_dicts = list([])
    for i in range(len(coefs)):
        new_param = {
            "name": col_names[i],
            "coefficient": coefs[i],
            "p_value": p_vals[i],
            "significant": p_vals[i] < 0.05,
        }
        param_dicts.append(new_param)

    param_df = pd.DataFrame.from_dict(param_dicts)
    return(param_df)
```

In [29]:
```
coef_df = coefficient_summary(train_df, bike_ols)
coef_df.head(len(coef_df))
```

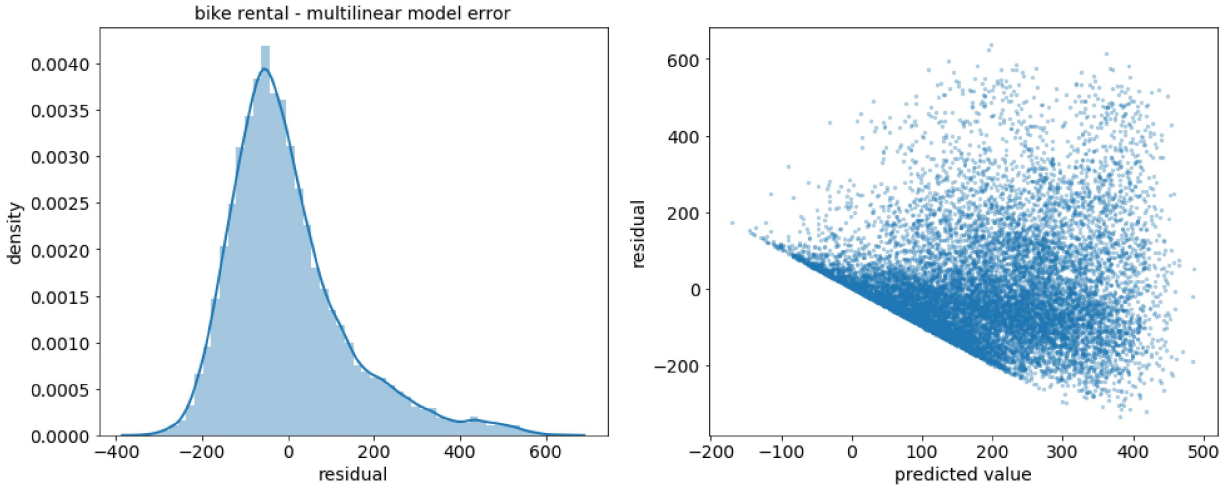
Out[29]:

	coefficient	name	p_value	significant
0	7.221382	hour	0.000000e+00	True
1	-18.095798	holiday	6.095043e-03	True
2	76.351903	year	6.205883e-218	True
3	11.317765	workingday	3.905740e-05	True
4	333.248241	temp	4.767468e-14	True
5	74.631221	atemp	1.062978e-01	False
6	-205.495864	hum	2.797780e-149	True
7	22.516841	windspeed	3.628163e-02	True
8	43.154064	spring	6.082058e-09	True
9	29.542551	summer	7.609902e-04	True
10	68.595334	fall	6.106365e-20	True
11	-7.643028	Feb	2.001503e-01	False
12	-11.673687	Mar	7.987186e-02	False
13	-41.524384	Apr	2.640964e-05	True
14	-33.292706	May	1.592599e-03	True
15	-65.803934	Jun	8.447047e-10	True
16	-93.480482	Jul	1.110753e-14	True
17	-59.208113	Aug	5.685359e-07	True
18	-16.051747	Sept	1.290517e-01	False
19	-16.160150	Oct	1.014216e-01	False
20	-25.873181	Nov	6.619949e-03	True
21	-10.204251	Dec	1.801759e-01	False
22	-2.660099	Mon	3.717564e-01	False
23	-6.142458	Tue	5.551230e-02	False
24	2.296440	Wed	4.706312e-01	False
25	-3.161076	Thu	3.209638e-01	False
26	2.889160	Fri	3.644519e-01	False
27	14.945929	Sat	6.490550e-04	True
28	6.786806	Cloudy	1.926802e-02	True
29	-28.285890	Snow	4.454966e-09	True
30	42.356939	Storm	6.667970e-01	False

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 [40]:
```
# create residual histogram plot
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.distplot(bike_ols.resid)
plt.xlabel("residual")
plt.ylabel("density")
plt.title("bike rental - multilinear model error")

# plot residual as a function of the predicted value
plt.subplot(1,2,2)
pred_counts = bike_ols.predict(sm.add_constant(train_df.drop("counts", axis=1).va
plt.scatter(pred_counts, (train_df.counts.values - pred_counts), s=5, alpha=0.3)
plt.xlabel("predicted value")
plt.ylabel("residual");
```



The assumption of linearity is not a good one. Qualitatively, the plot seems to show that the sign, magnitude, and variance of the error changes as a function of the predicted value. This is not what we would expect from a model that modeled the trend well with a uniform amount of error.

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

In [31]:
```
def add_poly_terms(df):
    # initialize new columns and add to train_df
    tmp_df = pd.DataFrame(data=df[["temp","hour","hum"]].values**2,\
                          columns=['temp_squared','hour_squared','hum_squared'],\
                          index=df.index)
    return(pd.concat([df, tmp_df], axis=1))
```

In [32]:
```
train_df, test_df = intialize_BSS_dfs()
train_df = add_poly_terms(train_df)
test_df = add_poly_terms(test_df)
```

In [33]:
```
X_train_poly = train_test_score(train_df, test_df)
```

```
training data:   r_squared = 0.50
test data:       r_squared = 0.50
```

In [34]:
```
coef_df = coefficient_summary(train_df, X_train_poly)
coef_df.head(len(coef_df))
```

Out[34]:

	coefficient	name	p_value	significant
0	39.578636	hour	0.000000e+00	True
1	-13.006135	holiday	3.176951e-02	True
2	81.030524	year	9.565654e-284	True
3	13.289427	workingday	1.422985e-07	True
4	132.724741	temp	2.282248e-02	True
5	67.495690	atemp	1.210488e-01	False
6	11.863600	hum	7.425350e-01	False
7	-6.910041	windspeed	4.860598e-01	False
8	43.711567	spring	1.371673e-10	True
9	33.908698	summer	2.641910e-05	True
10	72.193720	fall	1.117931e-25	True
11	1.648668	Feb	7.659367e-01	False
12	9.558311	Mar	1.294633e-01	False
13	-10.715221	Apr	2.461292e-01	False
14	-2.738812	May	7.796453e-01	False
15	-23.036803	Jun	2.025860e-02	True
16	-53.523006	Jul	1.647195e-06	True
17	-23.694352	Aug	3.072606e-02	True
18	10.905473	Sept	2.700522e-01	False
19	2.845152	Oct	7.586952e-01	False
20	-16.592637	Nov	6.220001e-02	False
21	-6.910556	Dec	3.288940e-01	False
22	-2.461952	Mon	3.674201e-01	False
23	-3.862897	Tue	1.892845e-01	False
24	2.127476	Wed	4.662491e-01	False
25	-0.254026	Thu	9.307218e-01	False
26	4.734692	Fri	1.053284e-01	False
27	16.798310	Sat	2.957339e-05	True
28	-8.432666	Cloudy	1.658110e-03	True
29	-47.326888	Snow	6.566553e-25	True
30	35.579966	Storm	6.933990e-01	False
31	109.443691	temp_squared	2.696299e-03	True
32	-1.357004	hour_squared	0.000000e+00	True

	coefficient	name	p_value	significant
33	-108.705709	hum_squared	1.735534e-04	True

Ridership increases with temperature_squared and decreases with humidity_squared and hour_squared.

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?

Multicolinearity occurs when two or more predictors are highly correlated. If they are strongly correlated enough, it means that the relationship between the correlated predictors and the response can largely be captured with a single coeficient. This can lead to erroneous coeficient values because it can lead to over or under stating the importance of correlated predictors.

In [35]:
```
# re-initialize dataframes
train_df, test_df = intialize_BSS_dfs()
train_df = add_poly_terms(train_df)
test_df = add_poly_terms(test_df)
```

In [36]:
```
# drop colinear predictors
train_df = train_df.drop(["workingday","atemp","spring","summer","fall"], axis=1)
test_df = test_df.drop(["workingday","atemp","spring","summer","fall"], axis=1)
```

In [37]:
```
X_train_drop = train_test_score(train_df, test_df)
```

```
training data:   r_squared = 0.50
test data:       r_squared = 0.49
```

In [38]:
```
# report parameter summary
coef_df = coefficient_summary(train_df, X_train_drop)
coef_df.head(len(coef_df))
```

Out[38]:

	coefficient	name	p_value	significant
0	39.645146	hour	0.000000e+00	True
1	-32.600969	holiday	1.643900e-06	True
2	80.814108	year	4.298388e-280	True
3	231.986479	temp	2.436789e-10	True
4	8.502294	hum	8.116272e-01	False
5	-17.628074	windspeed	6.678293e-02	False
6	0.696760	Feb	9.003513e-01	False
7	22.626452	Mar	1.192497e-04	True
8	30.304500	Apr	2.016278e-06	True
9	36.752614	May	3.048655e-07	True
10	13.129147	Jun	9.080194e-02	False
11	-23.169576	Jul	6.067791e-03	True
12	5.179251	Aug	5.180612e-01	False
13	50.298726	Sept	1.040450e-11	True
14	71.329003	Oct	2.443495e-27	True
15	53.542969	Nov	7.617469e-20	True
16	38.699257	Dec	4.651711e-12	True
17	11.402539	Mon	6.587941e-03	True
18	9.078461	Tue	2.606063e-02	True
19	15.043204	Wed	2.081938e-04	True
20	12.948724	Thu	1.517306e-03	True
21	16.867237	Fri	3.374560e-05	True
22	16.018655	Sat	7.329899e-05	True
23	-8.569233	Cloudy	1.457626e-03	True
24	-48.350132	Snow	1.041643e-25	True
25	34.759629	Storm	7.015054e-01	False
26	76.000369	temp_squared	3.445621e-02	True
27	-1.359223	hour_squared	0.000000e+00	True
28	-103.877675	hum_squared	2.837248e-04	True

The parameter estimate for the months are much less negative than before. Most likely this is because the season parameters were capturing much of the effect of those parameters. The coefficient estimate for temp increased. This makes sense because the coefficients for temp and

atemp before were both strongly positive and were highly correlated to each other. With atemp removed from the model, temp did not have to share the effect of temperature with another parameter.

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 [39]:
```
def get_model_BIC(x_data, y_data):

    model = OLS(y_data, sm.add_constant(x_data))
    model = model.fit()
    return(model.bic)
```

In [40]:
```
def best_kth_predictor(response, pred_confirmed, pred_candidates):

    pred_confirmed = pred_confirmed.values.reshape(pred_confirmed.shape[0],-1)
    BICs = np.empty(pred_candidates.shape[1])
    for i, col in enumerate(pred_candidates.columns):
        new_design_mat = np.concatenate((pred_confirmed,
                                         pred_candidates[col].values.reshape(-1,1)
        BICs[i] = get_model_BIC(new_design_mat,response)

    best_idx = np.argsort(BICs)
    return(pred_candidates.columns[best_idx[0]], BICs[best_idx[0]])
```

In [41]:
```
def forward_selection(response_data, pred_df):

    all_pred = pred_df.columns
    select_pred = []
    best_k_params_BIC = np.empty(len(all_pred))

    for k in range(len(all_pred)):
        curr_pred = list(set(select_pred) ^ set(all_pred))
        kth_pred, kth_BIC = best_kth_predictor(response_data,\
                                               pred_df[select_pred],\
                                               pred_df[curr_pred])
        best_k_params_BIC[k] = kth_BIC
        select_pred.append(kth_pred)

    return(select_pred, best_k_params_BIC)
```

In [42]:
```
train_df, test_df = intialize_BSS_dfs()
train_df = add_poly_terms(train_df)
test_df = add_poly_terms(test_df)
```

In [43]:
```
# do forward parameter selection
ordered_predictors, BICs = forward_selection(train_df.counts.values, train_df.drc
```

In [44]:
```
# find model with minimum BIC
idx_sort = np.argsort(BICs)
best_predictors = ordered_predictors[:idx_sort[0]+1]
print("Best model has %i predictors (in order of importance): \n" % len(best_prec
print(*best_predictors, sep="\n")
```

```
Best model has 12 predictors (in order of importance):

temp
hour
hour_squared
year
hum_squared
fall
Jul
Snow
spring
Sept
holiday
Cloudy
```

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.

`Spring` and `Fall` were not dropped from the model. One plausible reason why these two predictors stayed in the model is because most of the spring and fall months (their colinear predictors), were removed from the model.

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 [45]:
```
# re-intialize dataframes
train_df, test_df = intialize_BSS_dfs()
train_df = add_poly_terms(train_df)
test_df = add_poly_terms(test_df)
```

In [46]:
```
# report polynomial model score
X_train_poly = train_test_score(train_df, test_df)
```

```
training data:   r_squared = 0.50
test data:       r_squared = 0.50
```

In [47]:
```
# restrict dataframes to best parameters
best_predictors.append("counts")
train_df = train_df[best_predictors]
test_df = test_df[best_predictors]

# report scores for best k parameters
best_k_model = train_test_score(train_df, test_df)
```

```
training data:   r_squared = 0.50
test data:       r_squared = 0.49
```

The R^2 values are essentially unchanged between the two models ($R^2 \approx 0.5$)

Written Report to the Administrators [20 pts]

Question 6

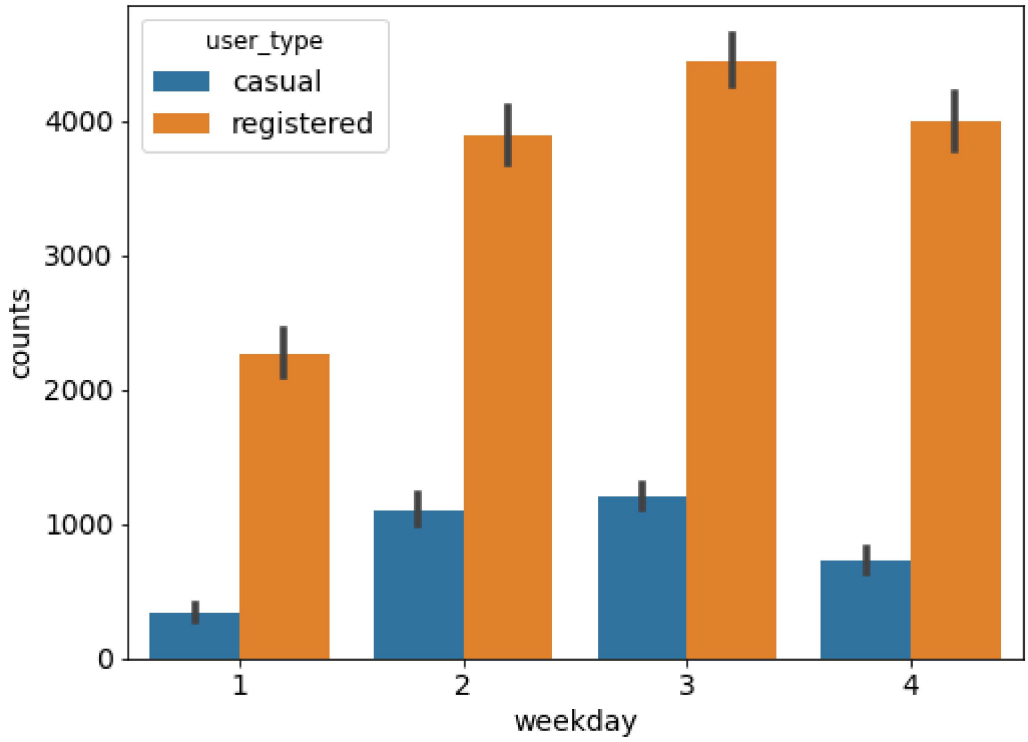
Write a short repost stating some of your findings on how the administrators can increase the bike share system's revenue. You might want to include suggestions such as what model to use to predict ridership, what additional services to provide, or when to give discounts, etc. Include your report as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

Answers 6

your answer here

In [109]:
```
sns.barplot(x="weekday", y="counts", data=vis_df, hue="user_type")
```

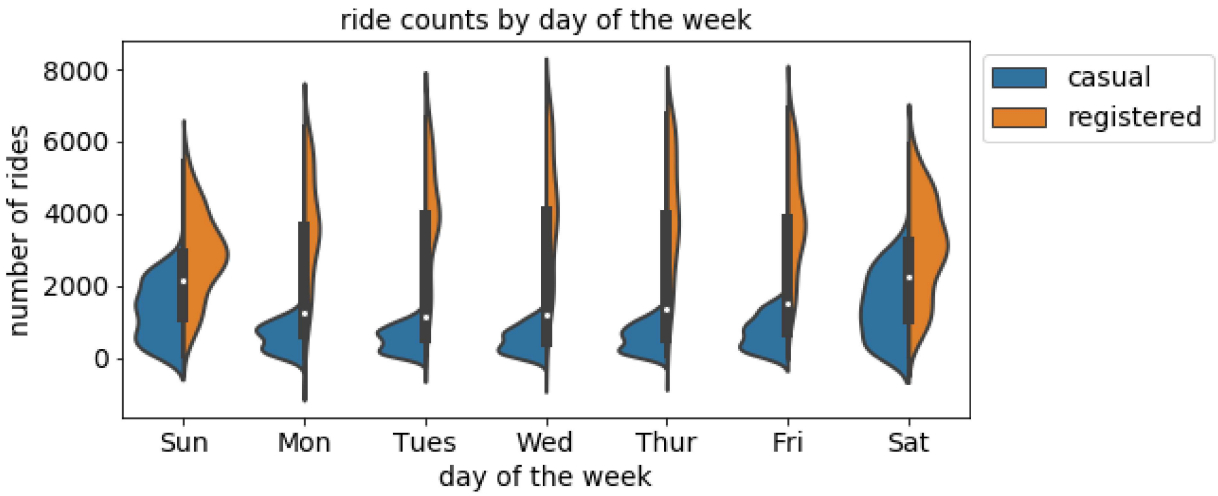
Out[109]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e5e0b4ee80>
```



In [82]:
```
# create new temp df for visualization
counts = bikes_by_day[["casual","registered"]].values
weekday = bikes_by_day[["weekday","weekday"]].values.flatten(order="F")
user = ["casual"] * counts.shape[0]
reg = ["registered"] * counts.shape[0]
user.extend(reg)
counts = counts.flatten(order="F")
data = {
    "counts": counts,
    "user_type": user,
    "weekday": weekday,
}
vis_df = pd.DataFrame.from_dict(data)
```

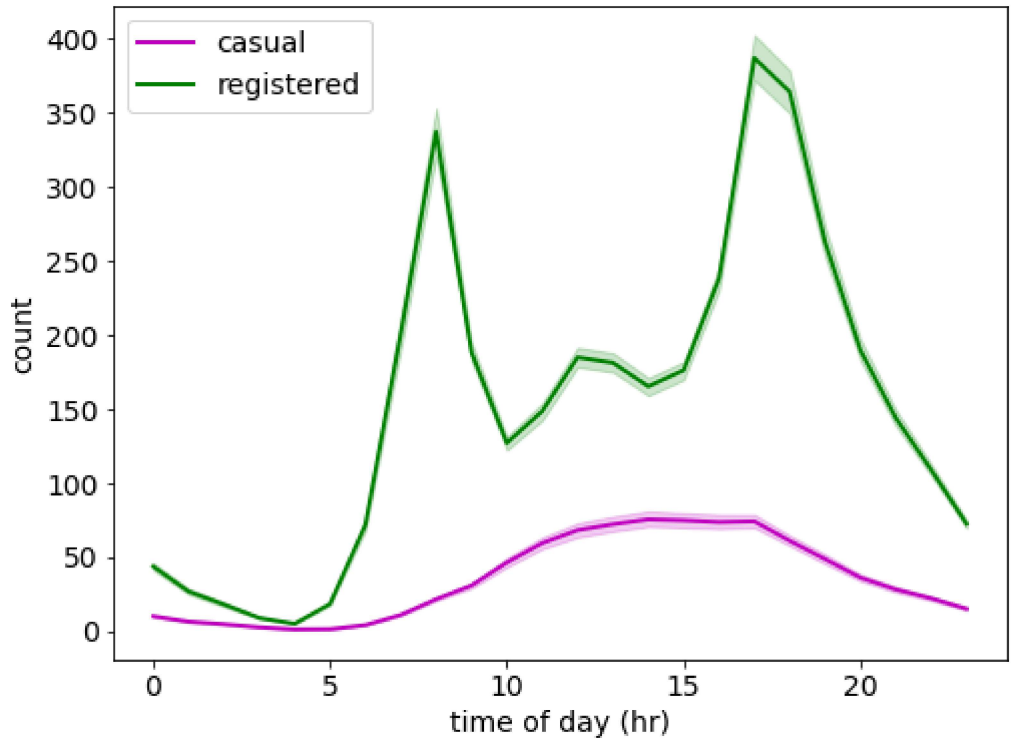
In [103]:
```
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
ax = sns.violinplot(x='weekday', y="counts", hue="user_type", split=True, data=vi
ax.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.xlabel("day of the week")
plt.ylabel("number of rides")
plt.xticks(np.arange(7),("Sun","Mon","Tues","Wed","Thur","Fri","Sat"))
plt.title("ride counts by day of the week")
```

Out[103]: Text(0.5,1,'ride counts by day of the week')



In [24]:
```
ax = sns.lineplot(x="hour", y="casual", data=bikes_df, label="casual", color='m')
sns.lineplot(x="hour", y="registered", data=bikes_df, label="registered", color='
plt.xlabel('time of day (hr)')
plt.ylabel('count')
plt.legend()
```

Out[24]:
```
<matplotlib.legend.Legend at 0x1494c42aa58>
```



In [49]:
```
# create new temp df for visualization
counts = bikes_by_day[["casual","registered"]].values
season = bikes_by_day[["season","season"]].values.flatten(order="F")
user = ["casual"] * counts.shape[0]
reg = ["registered"] * counts.shape[0]
user.extend(reg)
counts = counts.flatten(order="F")
data = {
    "counts": counts,
    "user_type": user,
    "season": season,
}
vis_df = pd.DataFrame.from_dict(data)
```

In [56]:
```
# first plot
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
ax = sns.barplot(x="season", y="counts", data=vis_df, hue="user_type")
plt.xticks(np.arange(4),("Winter","Spring","Summer","Fall"))
ax.legend(loc='upper left', bbox_to_anchor=(1,1))

plt.subplot(2,1,2)
ax = sns.violinplot(x='season', y="counts", hue="user_type", split=True, data=vis
ax.legend(loc='upper left', bbox_to_anchor=(1,1))
plt.xlabel("season")
plt.ylabel("number of rides")
plt.xticks(np.arange(4),("Winter","Spring","Summer","Fall"))
th = plt.title("ride counts by season")
plt.subplots_adjust(hspace=0.5)
```