# MSDS 430 Module 5 Exploratory Data Analysis (EDA)

**In this assignment you will read through the notebook and complete the exercises. Once you are satisfied with the results, submit your notebook and html/PDF file to Google Classroom. Your files should include all output, i.e. run each cell and save your file before submitting.**

**Research project problem statement:** A brewery has a number of signature beers that they produce and they want to expand their production in to different styles of beer. Listed below are some of the questions that the brewery wants to investigate. - They know what their market likes in beer, but what does the market in general rate highest? - What are the top breweries based on ratings making? - What are the top styles? - What else are you able to tell them about the top rated beers? - They are thinking about a seasonal beer but are not sure if seasonal beers are rated highly? - ... You will use a number of EDA techniques to answer these questions and many other questions.

**In many of the problems you will see #TODO statements added as comments on the code cell provided. You will want to be sure to complete each of these as indicated to avoid losing points.**

**Installing Python Packages from a Jupyter Notebook**

**Python Data Science Handbook**

In [3]:
```
# https://jakevdp.github.io/blog/2017/12/05/installing-python-packages-from-jupyter/
import sys
!conda update --yes --prefix {sys.prefix} seaborn
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working...

Updating seaborn is constricted by

anaconda -> requires seaborn==0.11.2=pyhd3eb1b0_0

If you are sure you want an update of your package either try `conda update --all` or
install a specific version of the package you want using `conda install <pkg>=<versio
n>`

done

# All requested packages already installed.
```

In [2]:
```
# load up modules
import pandas as pd
import numpy as np
import re
import seaborn as sns

# set up notebook to display multiple output in one cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
%matplotlib inline
```

# Part 1: Load the Datasets

- We are going to work with the following two sets of data:
  1. 'open-beer-database.csv'
  2. 'beer_reviews.csv'

Source 1: https://data.opendatasoft.com/explore/dataset/open-beer-database%40public-us/table/

Source 2: https://www.kaggle.com/rdoume/beerreviews

## Load 'open_beer_database.csv' and take a peek at the data

- Load the data in the file "open_beer_database.csv" into a dataframe and save it to the a variable named `beers`.
- View the data as follows:
  the total numbers of rows and columns
  the first (or last) few rows
  the column names
- Generate descriptive statistics of the dataframe using the pandas `describe()` method.
- Use `boxplots` to detect outliers.

In [4]:
```
# columns are separated by semicolons
beers = pd.read_csv('open_beer_database.csv',sep=';')

# what is the shape of the data?
beers.shape

# what are its columns
beers.columns

# look at first five records
beers.head()
```

Out[4]:
```
(5973, 22)
```

Out[4]:
```
Index(['Name', 'id', 'brewery_id', 'cat_id', 'style_id', 'Alcohol By Volume',
       'International Bitterness Units', 'Standard Reference Method',
       'Universal Product Code', 'filepath', 'Description', 'add_user',
       'last_mod', 'Style', 'Category', 'Brewer', 'Address', 'City', 'State',
       'Country', 'Coordinates', 'Website'],
      dtype='object')
```

Out[4]:

| | Name | id | brewery_id | cat_id | style_id | Alcohol By Volume | International Bitterness Units | Standard Reference Method | Universal Product Code | filepath |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | Porter | 716 | 842 | 2 | 25 | 0.0 | 0.0 | 0.0 | 0.0 | NaN |
| **1** | Possession Porter | 723 | 445 | 2 | 25 | 5.6 | 0.0 | 0.0 | 0.0 | NaN |
| **2** | Maibock | 736 | 1124 | 7 | 90 | 0.0 | 0.0 | 0.0 | 0.0 | NaN |
| **3** | Free Bike Amber | 742 | 1151 | 3 | 33 | 4.5 | 0.0 | 0.0 | 0.0 | NaN |
| **4** | Oatmeal Stout | 961 | 691 | 3 | 42 | 0.0 | 0.0 | 0.0 | 0.0 | NaN |

5 rows × 22 columns

In [5]:
```
beers.describe()

beers.describe(include=['O'])
```

Out[5]:

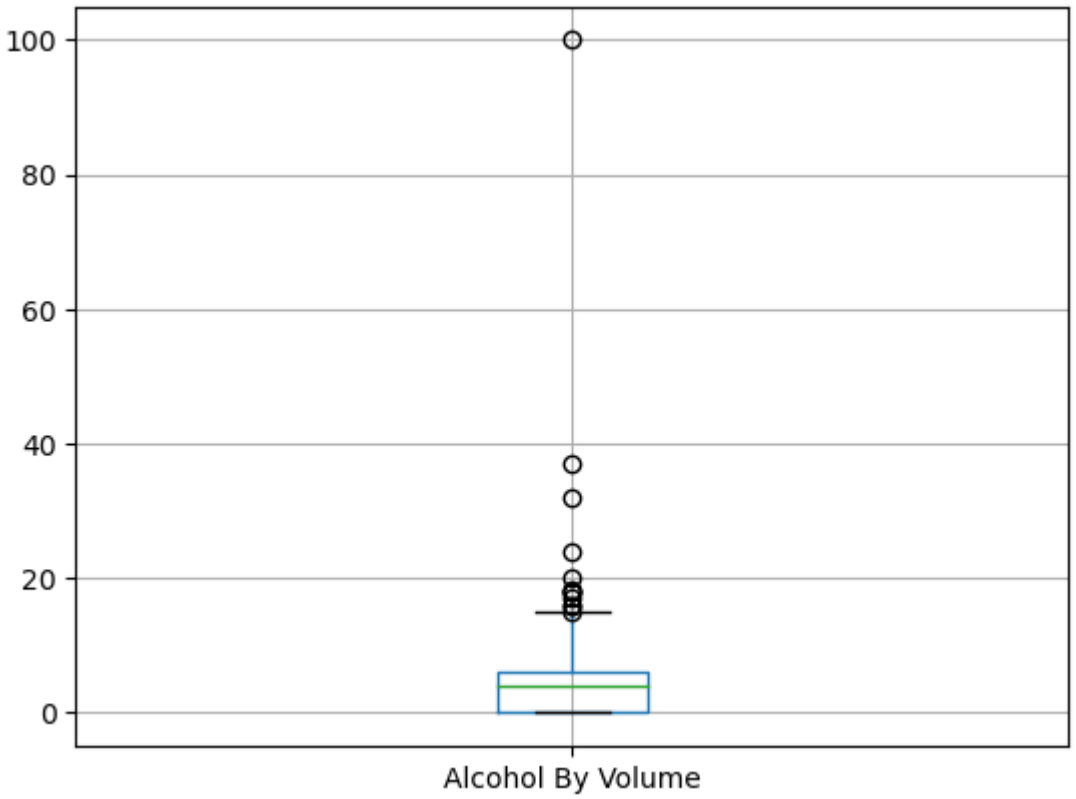
| | Alcohol By Volume | International Bitterness Units | Standard Reference Method | Universal Product Code |
|---|---|---|---|---|
| **count** | 5948.000000 | 5948.000000 | 5948.000000 | 5.944000e+03 |
| **mean** | 3.379987 | 0.121553 | 0.046738 | 1.445144e+06 |
| **std** | 3.846887 | 2.408826 | 1.193208 | 5.569431e+07 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000e+00 |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000e+00 |
| **50%** | 4.000000 | 0.000000 | 0.000000 | 0.000000e+00 |
| **75%** | 6.000000 | 0.000000 | 0.000000 | 0.000000e+00 |
| **max** | 99.989998 | 93.000000 | 47.000000 | 2.147484e+09 |

Out[5]:

| | Name | id | brewery_id | cat_id | style_id | filepath | Description | add_user | last_mod |
|---|---|---|---|---|---|---|---|---|---|
| **count** | 5963 | 5973 | 5963 | 5950 | 5949 | 25 | 2046 | 5930 | 590 |
| **unique** | 5050 | 5926 | 1332 | 16 | 73 | 25 | 2034 | 35 | 7 |
| **top** | Pale Ale | 642 | 858 | 3 | -1 | bluepoint-oktoberfest.png | 0 | 0 | 2010-07-22T13:00:00-07:0 |
| **freq** | 48 | 2 | 57 | 2014 | 1478 | 1 | 4 | 5829 | 583 |

In [6]:
```
beers.boxplot(column='Alcohol By Volume')
```

Out[6]:
```
<AxesSubplot:>
```




## Load `beer_reviews.csv` and take a peek at the data

- Load the data in the file "beer_reviews.csv" into a dataframe and save it to a variable named `beer_reviews`.
- View the data as follows:
  ```
  the total numbers of rows and columns
  the first (or last) few rows
  the column names
  ```

**Problem 1 (5 pts.)**: Read 'beer_reviews.csv" into Python just like we did with the 'open-beer-database.csv' file and assign the dataframe to the variable *beer_reviews*. Note that this time the fields in the csv are separated by commas (the default separator).

Also (a) display the shape, (b) the first five records, (c) the columns of *beer_reviews* and (d) the descriptive statistics. Finally, (e) create a boxplot to detect outliers in each of the review rating categories.

In [7]:
```
# TODO:  read 'beer_reviews.csv" into Python and assign the dataframe to the variable
beer_reviews = pd.read_csv('beer_reviews.csv')

# TODO: determine the shape of the data
beer_reviews.shape

# TODO: display the first five records
beer_reviews.head()
```

```
# TODO: use describe() to display descriptive statistics about the data
beer_reviews.describe()
```

Out[7]:
```
(342381, 13)
```

Out[7]:

| | brewery_id | brewery_name | review_time | review_overall | review_aroma | review_appearance | review_ |
|---|---|---|---|---|---|---|---|
| **0** | 1075 | Caldera Brewing Company | 1325524659 | 3.0 | 3.5 | 3.5 | |
| **1** | 1075 | Caldera Brewing Company | 1318991115 | 3.5 | 3.5 | 3.5 | |
| **2** | 1075 | Caldera Brewing Company | 1306276018 | 3.0 | 2.5 | 3.5 | |
| **3** | 1075 | Caldera Brewing Company | 1316025612 | 3.0 | 3.0 | 2.5 | Bee |
| **4** | 1075 | Caldera Brewing Company | 1325478004 | 4.5 | 4.5 | 3.0 | L |

Out[7]:

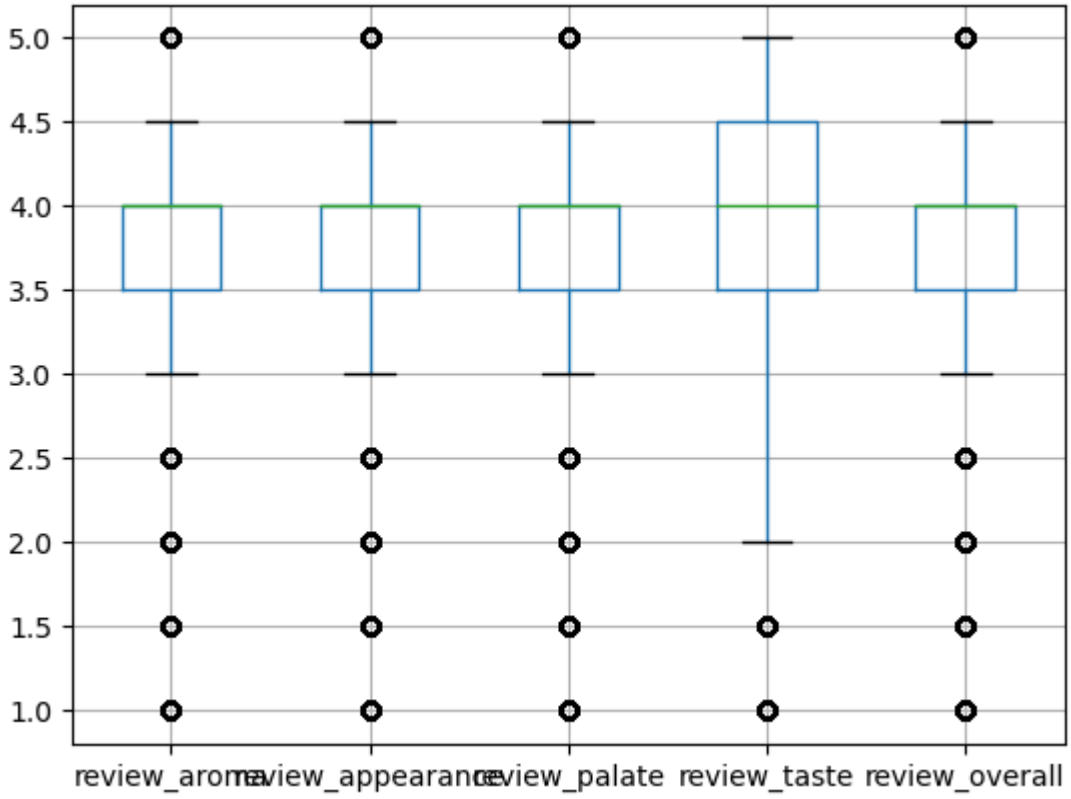
| | brewery_id | review_time | review_overall | review_aroma | review_appearance | review_palate |
|---|---|---|---|---|---|---|
| **count** | 342381.000000 | 3.423810e+05 | 342381.000000 | 342381.000000 | 342381.000000 | 342381.000000 |
| **mean** | 5032.652741 | 1.309746e+09 | 3.812731 | 3.776194 | 3.872972 | 3.770125 |
| **std** | 7374.499484 | 9.385856e+06 | 0.676551 | 0.666907 | 0.584289 | 0.650984 |
| **min** | 1.000000 | 1.293858e+09 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| **25%** | 192.000000 | 1.301459e+09 | 3.500000 | 3.500000 | 3.500000 | 3.500000 |
| **50%** | 718.000000 | 1.309664e+09 | 4.000000 | 4.000000 | 4.000000 | 4.000000 |
| **75%** | 9897.000000 | 1.317781e+09 | 4.000000 | 4.000000 | 4.000000 | 4.000000 |
| **max** | 28003.000000 | 1.326285e+09 | 5.000000 | 5.000000 | 5.000000 | 5.000000 |

In [8]:
```
# TODO: list the columns
beer_reviews.columns
```

Out[8]:
```
Index(['brewery_id', 'brewery_name', 'review_time', 'review_overall',
       'review_aroma', 'review_appearance', 'review_profilename', 'beer_style',
       'review_palate', 'review_taste', 'beer_name', 'beer_abv',
       'beer_beerid'],
      dtype='object')
```

In [9]:
```
# TODO: create a boxplot for aroma, appearance, palate, taste, and overall
beer_reviews.boxplot(column=['review_aroma','review_appearance','review_palate','revie
```

Out[9]:
```
<AxesSubplot:>
```




# Part 2: Inspect the Data

## Using `info( )` to inspect the two dataframes

- For each dataframe, we use the pandas function `info()` to display the column names (variables), their data types, and the number of non-null values in each column.

- We will also see what common variables (containing the same information) the two dataframes share.

**Observations from the output:**

- The columns **Style** and **Category** in the `beers` dataframe have a lot of missing values. However, the beer style also appears in the `beer_reviews` dataframe as **beer_style** column with no missing values.

- The `beer_reviews` dataframe contains many missing `beer_abv` values while `beers` only misses a few of the `Alcohol By Volume` values.

- The columns **filepath**, **Description** and **Website** from the `beers` dataframe miss most of their values and, therefore, they should be deleted from the `beers` dataframe or ignored

in the study of the data.

In [10]:
```
beers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5973 entries, 0 to 5972
Data columns (total 22 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Name                            5963 non-null   object 
 1   id                              5973 non-null   object 
 2   brewery_id                      5963 non-null   object 
 3   cat_id                          5950 non-null   object 
 4   style_id                        5949 non-null   object 
 5   Alcohol By Volume               5948 non-null   float64
 6   International Bitterness Units  5948 non-null   float64
 7   Standard Reference Method       5948 non-null   float64
 8   Universal Product Code          5944 non-null   float64
 9   filepath                        25 non-null     object 
 10  Description                     2046 non-null   object 
 11  add_user                        5930 non-null   object 
 12  last_mod                        5900 non-null   object 
 13  Style                           4466 non-null   object 
 14  Category                        4466 non-null   object 
 15  Brewer                          5948 non-null   object 
 16  Address                         5191 non-null   object 
 17  City                            5921 non-null   object 
 18  State                           5624 non-null   object 
 19  Country                         5948 non-null   object 
 20  Coordinates                     5746 non-null   object 
 21  Website                         2879 non-null   object 
dtypes: float64(4), object(18)
memory usage: 1.0+ MB
```

**Problem 2 (1 pt.)**: Apply the *info( )* method to *beer_reviews* like we did with the *beers* DataFrame object.

In [11]:
```
# TODO: use the info() method to inspect the data
beer_reviews.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 342381 entries, 0 to 342380
Data columns (total 13 columns):
 #   Column              Non-Null Count   Dtype  
---  ------              --------------   -----  
 0   brewery_id          342381 non-null  int64  
 1   brewery_name        342376 non-null  object 
 2   review_time         342381 non-null  int64  
 3   review_overall      342381 non-null  float64
 4   review_aroma        342381 non-null  float64
 5   review_appearance   342381 non-null  float64
 6   review_profilename  342381 non-null  object 
 7   beer_style          342381 non-null  object 
 8   review_palate       342381 non-null  float64
 9   review_taste        342381 non-null  float64
 10  beer_name           342381 non-null  object 
 11  beer_abv            331557 non-null  float64
 12  beer_beerid         342381 non-null  int64  
dtypes: float64(6), int64(3), object(4)
memory usage: 34.0+ MB
```

## Using `isnull( ).sum( )` to display the number of missing values in each column

- For each dataframe, we use `isnull().sum()` to displays the number of missing values in each column.
- `df.isnull()` returns a copy of the dataframe `df` with the null (i.e. `NaN`) values replaced by `True` and the nonnull values replaced by `False`. Since the Boolean values `True` and `False` are implemented as `1` and `0`, respectively, in Python, the sum of the Boolean values in each column of `df` (obtained by applying `sum()`) is the number of missing values in each column of `df`.

**Observations from the output:**

- The **Style** column in the `beers` dataframe has **1507** missing values. But the **beer_style** column in the `beer_reviews` dataframe doesn't have any missing values.

- The `beer_reviews` dataframe has **10,824** missing `beer_abv` values while `beers` has only **25** missing `Alcohol By Volume` values.

In [12]:
```
beers.isnull().head()
beers.isnull().sum()
beer_reviews.isnull().sum()
```

Out[12]:

| | Name | id | brewery_id | cat_id | style_id | Alcohol By Volume | International Bitterness Units | Standard Reference Method | Universal Product Code | filepath | . |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | False | False | False | False | False | False | False | False | False | True | . |
| **1** | False | False | False | False | False | False | False | False | False | True | . |
| **2** | False | False | False | False | False | False | False | False | False | True | . |
| **3** | False | False | False | False | False | False | False | False | False | True | . |
| **4** | False | False | False | False | False | False | False | False | False | True | . |

5 rows × 22 columns

Out[12]:
```
Name                              10
id                                 0
brewery_id                        10
cat_id                            23
style_id                          24
Alcohol By Volume                 25
International Bitterness Units    25
Standard Reference Method         25
Universal Product Code            29
filepath                        5948
Description                     3927
add_user                          43
last_mod                          73
Style                           1507
Category                        1507
Brewer                            25
Address                          782
City                              52
State                            349
Country                           25
Coordinates                      227
Website                         3094
dtype: int64
```

Out[12]:
```
brewery_id                0
brewery_name              5
review_time               0
review_overall            0
review_aroma              0
review_appearance         0
review_profilename        0
beer_style                0
review_palate             0
review_taste              0
beer_name                 0
beer_abv              10824
beer_beerid               0
dtype: int64
```

## Using `unique( )` to find the *different* beers in each dataframe

- Note that the number of rows in the `beers` dataframe is not the number of different beers since some of the beers are listed multiple times. To find the different beers included in the `beers` dataframe, we use the `unique()` function on the **Name** variable (column). Similar, we use `nunique()` to find the number of different beer names.

- Similarly, we use the `nunique()` function on the `beer_name` variable in the beer_reviews dataframe to find the number of different beers reviewed in the dataframe.

**Observations from the output:**

- From the results below, we see that `beer_reviews` contains more than 5 times as many beers as `beers` contains.

In [13]:
```
beers.Name.unique().shape
beer_reviews.beer_name.unique().shape  # or beer_reviews.beer_name.nunique()
```

Out[13]: (5051,)

Out[13]: (27437,)

## Using `nunique( )` to find the number of *different* beers missing abv

- When we select column `col` of a dataframe `df`, the result, `df.col`, is a a one dimensional data structure called a `Series`. The `Series` method `unique()` returns the different values in a series. Similarly, `nunique()` returns the *number* of different values in a series.
- If `col` is a column in `df`, then `df.col.isnull()` replaces null values in that column, actually series `df.col`, by `True` and the nonnull values by `False`. This allows us to use `df.col.isnull` as a "filter" to select the rows of the series `df.col` with null values: `df.col[df.col.isnull()]`. This is sometimes referred to as `Boolean filtering`.
- Recall that the `beer_reviews` dataframe has **10,824** missing `beer_abv` values. But how many different beers with missing abv in the dataframe?

- From the output in Problem 3, we will see there are **4,307** beers with missing abv.

## Selecting rows with Boolean filtering

How do we select the rows where values in one or more columns are missing? Easy, we use "Boolean filtering". Below we find that there are 52 rows with missing values in the City column. We create a new dataframe `df` with just these 52 rows.

In [14]:
```
beers.City.isnull().sum()
```

```
df = beers[beers.City.isnull()]
df.shape
```

Out[14]:
```
52
```

Out[14]:
```
(52, 22)
```

**Problem 3 (2 pts.):** Use Boolean filtering to select the rows in the *beer_reviews* dataframe where the *beer_abv* values are missing and save this new dataframe to a variable named *missing_abv*. Also determine the number of different beers with missing abv values.

In [15]:
```
# TODO: use Boolean filtering to select rows where beer_abv values are missing and ass
beer_reviews.beer_abv.isnull()
# TODO: use the nunique() method to determine the number of different beers with missi
beer_reviews.beer_abv.isnull().nunique()
```

Out[15]:
```
0         False
1         False
2         False
3         False
4         False
          ...
342376    False
342377    False
342378    False
342379    False
342380    False
Name: beer_abv, Length: 342381, dtype: bool
```

Out[15]:
```
2
```

# Part 3: Prepare the Data

**For the data preparation part, we will conduct the following three tasks:**

1. Eliminating columns that contain no useful information
2. Transforming time data to the standard datetime format
3. Handling missing values

**How to handle missing values**

- Here are the two common methods:
    1. Deleting rows with missing values
    2. Replacing missing values with a special value such Mean/Median/Mode or an unique category

**How we handle the missing brewery names in the `beer_reviews` dataframe**

**We will use the following functions to accomplish our tasks:**

1. Dropping the unusable columns using `drop()`
2. Transforming time data using `to_datetime()`

3. Filling missing values using `replace()` and `fillna()`
4. Dropping missing values using `dropna()`

## Using `drop( )` to eliminate columns

- Let's eliminate the columns we know for sure we will not use.

- From the `beers` dataframe, we will drop the following columns: **id**, **brewery_id**, **Standard Reference Method**, **Universal Product Code**, **filepath**, **Description**, **add_user**, **last_mod**, and **Website**. After we drop these columns we will save the new dataframe as `beers2` and use the new dataframe from now on.

- From the `beer_reviews` dataframe, we will drop the two columns: **brewery_id** and **beer_beerid**. After we drop the columns, we will save the new dataframe as `beer_reviews2` and use the new dataframe from now on.

- We can eliminate more later if EDA shows they will not help our analysis.

In [16]:
```
col_to_drop = ['id','brewery_id','Standard Reference Method','Universal Product Code',
               'filepath', 'Description', 'add_user','last_mod', 'Website']

beers2 = beers.drop(columns=col_to_drop, inplace=False)
beers2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5973 entries, 0 to 5972
Data columns (total 13 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   Name                            5963 non-null   object 
 1   cat_id                          5950 non-null   object 
 2   style_id                        5949 non-null   object 
 3   Alcohol By Volume               5948 non-null   float64
 4   International Bitterness Units  5948 non-null   float64
 5   Style                           4466 non-null   object 
 6   Category                        4466 non-null   object 
 7   Brewer                          5948 non-null   object 
 8   Address                         5191 non-null   object 
 9   City                            5921 non-null   object 
 10  State                           5624 non-null   object 
 11  Country                         5948 non-null   object 
 12  Coordinates                     5746 non-null   object 
dtypes: float64(2), object(11)
memory usage: 606.8+ KB
```

**Problem 4 (2 pts.):** Drop the "brewery_id" and "beer_beerid" columns from *beer_reviews* and save the new dataframe as "beer_reviews2". Finally, use **info( )** to display a concise summary of the new dataframe.

In [17]:
```
# TODO: create a new dataframe called 'beer_reviews2' that uses the drop() method to r
```

```
col_to_drop = ['brewery_id','beer_beerid']
beer_reviews2 = beer_reviews.drop(columns=col_to_drop, inplace = False)

# TODO: use the info() method to display a summary of 'beer_reviews2'
beer_reviews2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 342381 entries, 0 to 342380
Data columns (total 11 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   brewery_name        342376 non-null  object
 1   review_time         342381 non-null  int64
 2   review_overall      342381 non-null  float64
 3   review_aroma        342381 non-null  float64
 4   review_appearance   342381 non-null  float64
 5   review_profilename  342381 non-null  object
 6   beer_style          342381 non-null  object
 7   review_palate       342381 non-null  float64
 8   review_taste        342381 non-null  float64
 9   beer_name           342381 non-null  object
 10  beer_abv            331557 non-null  float64
dtypes: float64(6), int64(1), object(4)
memory usage: 28.7+ MB
```

## Using `to_datetime( )` to transform "unix time" data to the standard date values

- Note that the time data in the `review_time` column are "Unix times" so we need to covert them to the standard datetime values.

- We will add a new column named `review_date` to hold the transformed `review_time`.

- We will also use the pandas `value_counts()` function to display all the unique values with their counts.

In this example we show how to convert integer values in a column that represent the number of seconds that have elapsed since the "Unix epoch", i.e. `00:00:00 UTC on 1 January 1970`. Unix time is widely used in operating systems and file formats (including `beer_reviews.csv`) and one should now how to convert these values to a more useful representation. Luckily, pandas provides some methods for doing this conversion starting with `pd.to_datetime`.
See https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.to_datetime.html.
The values returned by `pd.to_datetime` are of type `Timestamp`. We then use the `dt.date` Series method to extract the date part of Timestamps without timezone information.

See https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.dt.date.html

First we create a series with "Unix time" values.

In [18]:
```
time_series = pd.Series([1234817823, 1235915097, 1235916604, 1234725145, 1293735206])
time_series
```

Out[18]:
```
0    1234817823
1    1235915097
2    1235916604
3    1234725145
4    1293735206
dtype: int64
```

The pd.to_datetime(...) methods converts such a value to a Timestamp object.

In [19]:
```
pd.to_datetime(1234817823,unit='s')  # The integer represents seconds elapsed..
```

Out[19]:
```
Timestamp('2009-02-16 20:57:03')
```

This method can be applied to all the values in the series at once...

In [20]:
```
pd.to_datetime(time_series, unit='s')
```

Out[20]:
```
0   2009-02-16 20:57:03
1   2009-03-01 13:44:57
2   2009-03-01 14:10:04
3   2009-02-15 19:12:25
4   2010-12-30 18:53:26
dtype: datetime64[ns]
```

We can follow it by dt.date to extract just the date from the timestamp (dropping the time).

In [21]:
```
pd.to_datetime(time_series, unit='s').dt.date
```

Out[21]:
```
0    2009-02-16
1    2009-03-01
2    2009-03-01
3    2009-02-15
4    2010-12-30
dtype: object
```

**Problem 5 (3 pts.):** Convert the values in the "review_time" column of the "beer_reviews2" dataframe to dates. Create a new "review_date" column in "beer_reviews2" with these date values and then drop the "review_time" column.

Use "value_counts( )" to display the 5 dates with the most reviews.

In [22]:
```
# TODO:  from the beer_reviews2 dataframe, create a new column 'review_date' that conv
#        'review_time' column to the standard datetime YYYY-MM-DD
beer_reviews2['review_date'] = pd.to_datetime(beer_reviews2.review_time, unit = 's')


# TODO: use the drop method to remove the review_time column
#beer_reviews.drop(columns='review_time', inplace = True)


# TODO: use value_counts() to display the 5 dates with the most reviews
beer_reviews2['review_date'].value_counts()
```

Out[22]:
```
2011-11-20 23:19:18    3
2011-08-31 22:46:40    3
2011-03-21 00:58:11    3
2011-04-08 23:46:30    3
2011-07-23 03:54:50    3
                      ..
2011-11-27 06:50:24    1
2011-02-21 14:12:12    1
2011-03-04 23:52:55    1
2011-03-08 11:42:16    1
2011-10-02 23:44:13    1
Name: review_date, Length: 339622, dtype: int64
```

## Using `drop( )` to drop rows with missing names in `beers2`

- We will drop the null values from the columns: **Name**, **Brewer**, **Style**, **Category**

- Before we do this, we want to know how many rows with missing values in these columns and display these rows.

In [23]:
```
beers2.Name.isnull().sum()
missing_beers = beers2[beers2.Name.isnull()][['Name','Brewer','Style','Category']]
missing_beers
```

Out[23]:
```
10
```

Out[23]:

| | Name | Brewer | Style | Category |
|---|---|---|---|---|
| **627** | NaN | NaN | NaN | NaN |
| **983** | NaN | NaN | NaN | NaN |
| **1593** | NaN | NaN | NaN | NaN |
| **2368** | NaN | NaN | NaN | NaN |
| **2761** | NaN | NaN | NaN | NaN |
| **3214** | NaN | NaN | NaN | NaN |
| **5250** | NaN | NaN | NaN | NaN |
| **5680** | NaN | NaN | NaN | NaN |
| **5803** | NaN | NaN | NaN | NaN |
| **5917** | NaN | NaN | NaN | NaN |

In [24]:
```
# Dropped the 10 rows...
beers2.shape
beers2 = beers2.drop(missing_beers.index)
beers2.shape
```

Out[24]:
```
(5973, 13)
```

Out[24]:
```
(5963, 13)
```

# Dealing with missing brewery names in `beer_reviews2`

## Step 1: Get the names of beers in the reviews with missing brewery names.

- First we use `isnull().sum()` on the `brewer_name` column to find the number of beers with missing brewery names.
- Then we display the different names of these beers using `unique()`.

**Observations from the output:**

- There are **4** different beers with missing brewery names:
  - Breakaway IPA
  - Caboose Oatmeal Stout
  - Engel Keller Dunkel WRONG BREWERY SEE CRAILSHEIMER
  - Engel Tyrolian Bräu WRONG BREWERY SEE SCHWABISCH GMUND
- The brewery names appears to be stored with the beer names in the last two cases. In these cases, we check if there are any reviews where the brewery name contains CRAILSHEIMER or SCHWABISCH GMUND (or SCHWABISCH or GMUND). If so then we may have found the missing breweries we are looking for. Otherwise, we have no choice but to drop the reviews.
- For the first two cases we check if there are other reviews for the same beer which have the brewery present and use it to repair the missing brewery names in the reviews for the beer that are missing it.

In [25]:
```
# Find the number of beers with missing brewery names:
beer_reviews2.brewery_name.isnull().sum()

# Display the names of these beers:
beer_reviews_missing = beer_reviews2[beer_reviews2.brewery_name.isnull()]
beer_reviews_missing.beer_name.value_counts()
```

Out[25]:
```
5
```

Out[25]:
```
Breakaway IPA                                          2
Engel Tyrolian Bräu WRONG BREWERY SEE SCHWABISCH GMUND  1
Engel Keller Dunkel  WRONG BREWERY SEE CRAILSHEIMER     1
Caboose Oatmeal Stout                                  1
Name: beer_name, dtype: int64
```

## Step 2: Find the brewery associated with "CRAILSHEIMER" and use it to replace the missing brewery name.

- First we want to see how many beers whose name contains the substring **"CRAILSHEIMER"** in the `beer_reviews_missing` dataframe.
- Then we search the breweries whose name contains the substring **"CRAILSHEIMER"** in the `beer_reviews2` dataframe. In the searching we ignore case by setting `flags=re.IGNORECASE`.

**Observations from the output:**

- We found the missing brewery name: `'Crailsheimer Engelbräu'`. Set the name of the brewery of the `Engel Keller Dunkel  WRONG BREWERY SEE CRAILSHEIMER` beer to that.

In [26]:
```
#  beer_reviews_missing[beer_reviews_missing.beer_name.str.contains('CRAILSHEIMER')]
beer_reviews2.brewery_name.str.contains('CRAILSHEIMER',flags=re.IGNORECASE,na=False).s
```

Out[26]:
```
1
```

In [27]:
```
df=beer_reviews2[beer_reviews2.brewery_name.str.contains('CRAILSHEIMER',flags=re.IGNOR
df
```

Out[27]:

| | brewery_name | review_time | review_overall | review_aroma | review_appearance | review_profile |
|---|---|---|---|---|---|---|
| **188066** | Crailsheimer Engelbräu | 1320340873 | 3.0 | 3.0 | 3.0 | ja |

In [28]:
```
df.brewery_name.iloc[0]
```

Out[28]:
```
'Crailsheimer Engelbräu'
```

In [29]:
```
index_of_missing =\
beer_reviews2[beer_reviews2.beer_name.str.contains('CRAILSHEIMER',flags=re.IGNORECASE,
beer_reviews2.loc[index_of_missing]
beer_reviews2.at[index_of_missing,'brewery_name'] = df.brewery_name.iloc[0]
beer_reviews2.loc[index_of_missing]
```

Out[29]:
```
brewery_name                                                  NaN
review_time                                            1298078926
review_overall                                                3.0
review_aroma                                                  3.0
review_appearance                                             3.0
review_profilename                                      Ochsenblut
beer_style                                  Keller Bier / Zwickel Bier
review_palate                                                 2.0
review_taste                                                  3.0
beer_name             Engel Keller Dunkel  WRONG BREWERY SEE CRAILSH...
beer_abv                                                      5.3
review_date                                   2011-02-19 01:28:46
Name: 144790, dtype: object
```

Out[29]:
```
brewery_name                                    Crailsheimer Engelbräu
review_time                                                 1298078926
review_overall                                                    3.0
review_aroma                                                      3.0
review_appearance                                                 3.0
review_profilename                                          Ochsenblut
beer_style                                  Keller Bier / Zwickel Bier
review_palate                                                     2.0
review_taste                                                      3.0
beer_name             Engel Keller Dunkel  WRONG BREWERY SEE CRAILSH...
beer_abv                                                          5.3
review_date                                       2011-02-19 01:28:46
Name: 144790, dtype: object
```

## Step 3: Remove **WRONG BREWERY SEE CRAILSHEIMER** from the beer name.

- Replace the substring: **WRONG BREWERY SEE CRAILSHEIMER** in the beer name by **Crailsheimer Engelbräu**.

In [30]:
```
beer_reviews2.beer_name.replace(regex=[" WRONG BREWERY SEE CRAILSHEIMER"], value="",ir
beer_reviews2.loc[index_of_missing]
```

Out[30]:
```
brewery_name              Crailsheimer Engelbräu
review_time                           1298078926
review_overall                               3.0
review_aroma                                 3.0
review_appearance                            3.0
review_profilename                    Ochsenblut
beer_style            Keller Bier / Zwickel Bier
review_palate                                2.0
review_taste                                 3.0
beer_name                   Engel Keller Dunkel
beer_abv                                     5.3
review_date                 2011-02-19 01:28:46
Name: 144790, dtype: object
```

In [31]:
```
# Display the beer names with missing brewer names:
beer_reviews_missing = beer_reviews2[beer_reviews2.brewery_name.isnull()]
beer_reviews_missing.beer_name.unique()
```

Out[31]:
```
array(['Engel Tyrolian Bräu WRONG BREWERY SEE SCHWABISCH GMUND',
       'Breakaway IPA', 'Caboose Oatmeal Stout'], dtype=object)
```

## Step 4: Dealing with "SCHWABISCH GMUND" ...

- We will repeat what we did in the step 3:
- First we want to see how many beers whose name contains the substring **'schwabisch'** in the `beer_reviews_missing` dataframe.

- Then we search the breweries whose name contains the substring **'schwabisch'** in the `beer_reviews2` dataframe.

**Observations from the output:**

- Only one beer name contains 'schwabisch' and no brewery name contains the substring. Thus we should drop the row with missing brewery name and beer name containing

'schwabisch'.

In [32]:
```
beer_reviews2.beer_name.str.contains('schwabisch',flags=re.IGNORECASE).sum() # output
beer_reviews2.beer_name.str.contains('gemund',flags=re.IGNORECASE).sum()     # output
```

Out[32]:
```
1
```

Out[32]:
```
0
```

In [33]:
```
beer_reviews_missing.beer_name.str.contains('schwabisch',flags=re.IGNORECASE).sum()
```

Out[33]:
```
1
```

In [34]:
```
beer_reviews_missing.beer_name.value_counts()
```

Out[34]:
```
Breakaway IPA                                              2
Engel Tyrolian Bräu WRONG BREWERY SEE SCHWABISCH GMUND     1
Caboose Oatmeal Stout                                      1
Name: beer_name, dtype: int64
```

In [35]:
```
beer_reviews_missing[beer_reviews_missing.beer_name.str.lower().str.contains('schwabis
```

Out[35]:

| | brewery_name | review_time | review_overall | review_aroma | review_appearance | review_profile |
|---|---|---|---|---|---|---|
| **142780** | NaN | 1301022066 | 2.0 | 2.5 | 2.5 | Kn |

In [36]:
```
beer_reviews2.brewery_name.str.contains('schwabisch',flags=re.IGNORECASE).sum()
```

Out[36]:
```
0
```

In [37]:
```
beer_reviews2.shape[0]     # number of records
beer_reviews2 = beer_reviews2.drop(142780) # index of the row we want to drop
beer_reviews2.shape[0] # check that is is one less row...
```

Out[37]:
```
342381
```

Out[37]:
```
342380
```

## Step 5: Dealing with "Caboose Oatmeal Stout"...

In [38]:
```
beer_reviews2[beer_reviews2.beer_name.str.contains("Caboose Oatmeal Stout",flags=re.IC
```

Out[38]:

| | brewery_name | review_time | review_overall | review_aroma | review_appearance | review_profile |
|---|---|---|---|---|---|---|
| **183063** | American Brewing Company | 1325535022 | 4.0 | 4.0 | 4.0 | timtw |
| **183064** | American Brewing Company | 1324594647 | 4.5 | 4.5 | 4.5 | o |
| **183065** | American Brewing Company | 1321151918 | 4.5 | 3.5 | 3.5 | lo |
| **183066** | American Brewing Company | 1319084458 | 4.0 | 4.0 | 4.0 | flagm |
| **183067** | American Brewing Company | 1312754960 | 4.0 | 4.0 | 4.0 | barleywin |
| **183068** | American Brewing Company | 1311824406 | 4.5 | 4.5 | 4.5 | |
| **183069** | American Brewing Company | 1298865577 | 4.0 | 4.0 | 4.0 | bee |
| **302053** | NaN | 1320995408 | 5.0 | 4.5 | 4.0 | |

In [39]:
```
beer_reviews2.loc[302053]

beer_reviews2.loc[302053,'brewery_name']= "American Brewing Company"

beer_reviews2.loc[302053]
```

Out[39]:
```
brewery_name                          NaN
review_time                    1320995408
review_overall                        5.0
review_aroma                          4.5
review_appearance                     4.0
review_profilename                  Docer
beer_style                American Stout
review_palate                         4.5
review_taste                          4.5
beer_name           Caboose Oatmeal Stout
beer_abv                              7.0
review_date           2011-11-11 07:10:08
Name: 302053, dtype: object
```

Out[39]:
```
brewery_name          American Brewing Company
review_time                         1320995408
review_overall                             5.0
review_aroma                               4.5
review_appearance                          4.0
review_profilename                       Docer
beer_style                    American Stout
review_palate                              4.5
review_taste                               4.5
beer_name               Caboose Oatmeal Stout
beer_abv                                   7.0
review_date               2011-11-11 07:10:08
Name: 302053, dtype: object
```

## Step 6: Dealing with "Breakaway IPA"...

- Looks like all the reviews of **"Breakaway IPA"** contain `NaN` values for the **brewery_name**. So we drop such reviews from our dataframe.

In [40]:
```
import re
bb_reviews = beer_reviews2[beer_reviews2.beer_name.str.contains("Breakaway IPA",flags=
bb_reviews
```

Out[40]:

| | brewery_name | review_time | review_overall | review_aroma | review_appearance | review_profile |
|---|---|---|---|---|---|---|
| **302051** | NaN | 1323314674 | 4.5 | 4.5 | 3.5 | |
| **302052** | NaN | 1320989774 | 3.5 | 4.0 | 3.5 | |

## Step 7: Drop these two rows.

In [41]:
```
beer_reviews2.shape[0]     # number of records
beer_reviews2 = beer_reviews2.drop([302051,302052]) # index of the row we want to drop
beer_reviews2.shape[0] # check that is is one less row..
```

Out[41]:
```
342380
```

Out[41]:
```
342378
```

## Step 8: We drop all rows with NaN values

We check for any missing values in `beer_reviews2`. It looks like there are quite a few of them in the `beer_abv` column (but still only 0.3% of the value in the column). We drop all the rows with NaN values

In [42]:
```
beer_reviews2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 342378 entries, 0 to 342380
Data columns (total 12 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   brewery_name        342378 non-null  object
 1   review_time         342378 non-null  int64
 2   review_overall      342378 non-null  float64
 3   review_aroma        342378 non-null  float64
 4   review_appearance   342378 non-null  float64
 5   review_profilename  342378 non-null  object
 6   beer_style          342378 non-null  object
 7   review_palate       342378 non-null  float64
 8   review_taste        342378 non-null  float64
 9   beer_name           342378 non-null  object
 10  beer_abv            331554 non-null  float64
 11  review_date         342378 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(6), int64(1), object(4)
memory usage: 34.0+ MB
```

**Problem 6 (3 pts.):** Drop all rows with missing *beer_abv* values. Display the shape and info.

In [43]:
```
beer_reviews2.dropna(subset=['beer_abv'], inplace=True)
beer_reviews2.shape
beer_reviews2.info()
```

Out[43]:
```
(331554, 12)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 331554 entries, 0 to 342380
Data columns (total 12 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   brewery_name        331554 non-null  object
 1   review_time         331554 non-null  int64
 2   review_overall      331554 non-null  float64
 3   review_aroma        331554 non-null  float64
 4   review_appearance   331554 non-null  float64
 5   review_profilename  331554 non-null  object
 6   beer_style          331554 non-null  object
 7   review_palate       331554 non-null  float64
 8   review_taste        331554 non-null  float64
 9   beer_name           331554 non-null  object
 10  beer_abv            331554 non-null  float64
 11  review_date         331554 non-null  datetime64[ns]
dtypes: datetime64[ns](1), float64(6), int64(1), object(4)
memory usage: 32.9+ MB
```

# Part 4: Analyze the Data

1. How many total reviews are there? How many different beers were reviewed?
2. How many different beer styles are there in the reviews? What are the five most common?
3. What are the top 5 cities in terms of the number of different beers they produce? Use the "beers" dataframe to answer this question and then create a series (top5_breweries) of the

breweries in these 5 cities. Use this series to obtain the dataframe (top5_reviews) of reviews for breweries in those cities.

4. For these top 5 cities, how many of each style does it produce? Find the average overall rating of each style to find the most popular style. (This is a bit of cheat since I am using two different data sets for the two parts of the question...)

**Problem 7 (2 pts.):** How many total reviews are there? How many different beers were reviewed?

In [56]:
```
# TODO: find the total number of reviews in the beer_reviews2
print(f"Total number of reviews: {len(beer_reviews2.axes[0])}")

# TODO: find the number of *different* beers in beer_reviews2
print(f"Total number of different beers reviewed: {beer_reviews2.beer_name.nunique()}"
```

```
Total number of reviews: 331554
Total number of different beers reviewed: 23476
```

**Problem 8 (2 pts.):** How many different beer styles are there in the reviews? What are the five most common?

In [57]:
```
# TODO: find the number of different beer styles in beer_reviews2
print(f"Total number of different beers reviewed: {beer_reviews2.beer_style.nunique()}
```

```
Total number of different beers reviewed: 104
```

In [63]:
```
# TODO: display the five most common beer styles in beer_reviews2
print(f"Total number of different beers reviewed: {beer_reviews2.beer_style.sort_value
```

```
Total number of different beers reviewed: 92262    Witbier
Name: beer_style, dtype: object
```

**Save your updated dataframe to a csv file called "beer_reviews_final.csv" for use in next week's assignment.**

In [65]:
```
beer_reviews2.to_csv('beer_reviews_final.csv',index=False)
```

What are the top 5 cities in terms of the number of different beers they produce? We'll use the `beers` dataframe to answer this question and then create a series `top5_breweries` of the breweries from these 5 cities. Then, we'll use this series to obtain the dataframe `top5_reviews` of reviews for breweries in those cities.

In [66]:
```
# using the 'beers' dataframe, determine the top 5 cities based on the number of beers
# display the results
top5 = beers.City.value_counts().head()
top5

# create and display a series named 'top5_breweries' of the breweries from the top 5 c
top5_breweries = beers[beers.City.isin(top5.index)].Brewer
top5_breweries
```

```
# using 'top5_breweries', create a dataframe called 'top5_reviews' that contains revie
# the top 5 cities

top5_reviews = beer_reviews[beer_reviews.brewery_name.isin(top5_breweries)]

# display the first 5 records of 'top5_reviews'
top5_reviews.head()
```

Out[66]:
```
Denver       110
Seattle      106
Portland      69
Chicago       65
Anchorage     65
Name: City, dtype: int64
```

Out[66]:
```
28                            Shipyard Brewing - Portland
32                          Elysian Brewing - TangleTown
54      Pyramid Alehouse, Brewery and Restaurant - Sea...
68                                   Redhook Ale Brewery
76                             Bull & Bush Pub & Brewery
                              ...
5829                           Midnight Sun Brewing Co.
5862                               Great Divide Brewing
5864                               Metropolitan Brewing
5906                     Heavenly Daze Brewery and Grill
5918                                River West Brewing
Name: Brewer, Length: 415, dtype: object
```

Out[66]:

| | brewery_id | brewery_name | review_time | review_overall | review_aroma | review_appearance | re |
|---|---|---|---|---|---|---|---|
| **58509** | 68 | Flying Dog Brewery | 1302051417 | 4.0 | 3.5 | 3.0 | |
| **58510** | 68 | Flying Dog Brewery | 1301969276 | 4.5 | 4.0 | 4.0 | |
| **58521** | 68 | Flying Dog Brewery | 1299961528 | 3.5 | 2.5 | 3.0 | |
| **59845** | 68 | Flying Dog Brewery | 1299895015 | 2.5 | 2.0 | 4.0 | |
| **60312** | 68 | Flying Dog Brewery | 1322628792 | 3.5 | 3.5 | 3.5 | |

For these top 5 cities, how many of each style does it produce? We'll show the 10 most popular styles and find the average overall rating of each style to find the most popular style. Finally, we'll display the 10 most highly rated styles.

In [67]:
```
# from the top 5 cities in the previous problem, display the 10 most popular styles pr
# and the number produced for each style
beers[beers.City.isin(top5.index)].Style.value_counts()[:10]
```

Out[67]:
```
American-Style Pale Ale              34
American-Style India Pale Ale        29
American-Style Lager                 27
American-Style Stout                 24
American-Style Amber/Red Ale         21
Porter                               21
Imperial or Double India Pale Ale    16
American-Style Brown Ale             15
German-Style Oktoberfest             12
Other Belgian-Style Ales             10
Name: Style, dtype: int64
```

In [68]:
```
# find the average overall rating for each of the styles in the previous step and disp
# rated styles in descending order
top5_style_ratings =\
top5_reviews.groupby('beer_style')['review_overall'].mean().reset_index()
top5_style_ratings.sort_values(by='review_overall',ascending=False)[:10]
```

Out[68]:

| | beer_style | review_overall |
|---|---|---|
| **43** | Flanders Oud Bruin | 4.666667 |
| **59** | Russian Imperial Stout | 4.500000 |
| **23** | Berliner Weissbier | 4.333333 |
| **64** | Scottish Ale | 4.250000 |
| **70** | Wheatwine | 4.250000 |
| **44** | Foreign / Export Stout | 4.178571 |
| **35** | English Barleywine | 4.174107 |
| **51** | Maibock / Helles Bock | 4.125000 |
| **57** | Quadrupel (Quad) | 4.071429 |
| **21** | Belgian Strong Dark Ale | 4.058824 |

Now we'll create a dataframe showing the average overall review and number of beer reviews for each beer style for the top 5 cities. Then we'll select those syles for which there are more then 10 reviews and display the 10 most highly rated styles. *Unlike the last step, we are ignoring styles which did not receive "enough" reviews.*

In [70]:
```
# create a dataframe called 'top5_style_counts'
top5_style_counts = \
top5_reviews.beer_style.value_counts().reset_index(name='number of reviews').rename(cc
top5_style_counts
```

Out[70]:

| | beer_style | number of reviews |
|---|---|---|
| **0** | American Double / Imperial IPA | 482 |
| **1** | American IPA | 481 |
| **2** | American Porter | 480 |
| **3** | Baltic Porter | 297 |
| **4** | Belgian IPA | 293 |
| **...** | ... | ... |
| **68** | Weizenbock | 2 |
| **69** | Rye Beer | 1 |
| **70** | Dortmunder / Export Lager | 1 |
| **71** | Milk / Sweet Stout | 1 |
| **72** | Bière de Champagne / Bière Brut | 1 |

73 rows × 2 columns

In [71]:
```
top5_style_ratings2 = \
pd.merge(top5_style_ratings,top5_style_counts)
top5_style_ratings2
```

Out[71]:

| | beer_style | review_overall | number of reviews |
|---|---|---|---|
| **0** | Altbier | 3.974359 | 39 |
| **1** | American Amber / Red Ale | 3.447977 | 173 |
| **2** | American Amber / Red Lager | 3.750000 | 2 |
| **3** | American Barleywine | 3.872093 | 43 |
| **4** | American Black Ale | 3.985075 | 67 |
| **...** | ... | ... | ... |
| **68** | Vienna Lager | 3.866667 | 15 |
| **69** | Weizenbock | 4.000000 | 2 |
| **70** | Wheatwine | 4.250000 | 2 |
| **71** | Winter Warmer | 3.430921 | 152 |
| **72** | Witbier | 3.642857 | 98 |

73 rows × 3 columns

In [72]:
```
top5_style_ratings2[top5_style_ratings2['number of reviews']>10].sort_values(by='revie
```

Out[72]:

| | beer_style | review_overall | number of reviews |
|---|---|---|---|
| **44** | Foreign / Export Stout | 4.178571 | 28 |
| **35** | English Barleywine | 4.174107 | 112 |
| **57** | Quadrupel (Quad) | 4.071429 | 14 |
| **21** | Belgian Strong Dark Ale | 4.058824 | 51 |
| **22** | Belgian Strong Pale Ale | 4.043478 | 23 |
| **9** | American Double / Imperial Stout | 4.008671 | 173 |
| **18** | Baltic Porter | 3.986532 | 297 |
| **4** | American Black Ale | 3.985075 | 67 |
| **0** | Altbier | 3.974359 | 39 |
| **58** | Rauchbier | 3.952381 | 21 |