mlp-week01

January 20, 2021

1 Machine Learning in Python - Workshop 1

As with any other programming language, the best way to learn Python and its machine learning libraries is to play with them, so follow the steps below and ask for help from a tutor if you get stuck.

Today's workshop is mostly about refamiliarizing everyone with some of the core libraries we will be using for data management and visualization. If you don't remember how to do something we would generally advise that you do the following: start by taking a look a the package documentation, then ask your classmates, and if you are still stuck then ask for help from a tutor.

1.1 1. Pandas

This course will assume that you have some basic familiarity with the **pandas** library, and now is a good time to go back and review the relevant materials from Python Programming and the pandas documentation.

For this workshop we will review a small part of **pandas** by working with a sample of data of Airbnb listings in Edinburgh. These data are included in the `listings.csv` file which should be available along with this notebook.

The data set includes the following variables:

- `id` - ID number of the listing
- `price` - Price, in GBP, for one night stay
- `neighbourhood` - Neighbourhood listing is located in
- `accommodates` - Number of people listing accommodates
- `bathrooms` - Number of bathrooms
- `bedrooms` - Number of bedrooms
- `beds` - Number of beds (which can be different than the number of bedrooms)
- `review_scores_rating` - Average rating of property
- `number_of_reviews` - Number of reviews
- `listing_url` - Listing URL

We will read in these data using pandas with the following code,

[1]:
```
import pandas as pd

d = pd.read_csv("listings.csv")
```

```
d
```

```
[1]:              id  price neighbourhood  accommodates  bathrooms  bedrooms  beds  \
     0         15420   80.0      New Town             2        1.0       1.0   1.0
     1         24288  115.0     Southside             4        1.5       2.0   2.0
     2         38628   46.0           NaN             2        1.0       0.0   2.0
     3         44552   32.0         Leith             2        1.0       1.0   1.0
     4         47616  100.0     Southside             2        1.0       1.0   1.0
     …           …    …             …           …         …        …    …
     13240  36061175   95.0      New Town             3        1.0       1.0   2.0
     13241  36061191    NaN     Tollcross             3        1.0       1.0   2.0
     13242  36061722    NaN      Old Town             5        2.0       2.0   4.0
     13243  36061940   47.0           NaN             2        1.0       2.0   2.0
     13244  36066014   35.0           NaN             2        2.5       1.0   1.0

            review_scores_rating  number_of_reviews  \
     0                      99.0                283
     1                      92.0                199
     2                      94.0                 52
     3                      93.0                184
     4                      98.0                 32
     …                    …                 …
     13240                   NaN                  0
     13241                   NaN                  0
     13242                   NaN                  0
     13243                   NaN                  0
     13244                   NaN                  0

                                   listing_url
     0         https://www.airbnb.com/rooms/15420
     1         https://www.airbnb.com/rooms/24288
     2         https://www.airbnb.com/rooms/38628
     3         https://www.airbnb.com/rooms/44552
     4         https://www.airbnb.com/rooms/47616
     …                                    …
     13240  https://www.airbnb.com/rooms/36061175
     13241  https://www.airbnb.com/rooms/36061191
     13242  https://www.airbnb.com/rooms/36061722
     13243  https://www.airbnb.com/rooms/36061940
     13244  https://www.airbnb.com/rooms/36066014

     [13245 rows x 10 columns]
```

Note here we print out the pandas dataframe object by returning it at the end of the cell, generally when we want to output something in a notebook it is better to use an explicit `print` function call but in this case we want to take advantage of Jupyter's ability to nicely display the pandas data frame output.

Below are a couple of quick exercises to re-familiarize yourself with pandas.

1.1.1 Exercise 1

How many observations are included in this data set?

[2]:
```python
len(d.index)
```

[2]:
```
13245
```

[3]:
```python
d.shape[0]
```

[3]:
```
13245
```

1.1.2 Exercise 2

How many different neighborhoods are represented in these data?

[4]:
```python
d["neighbourhood"].nunique()
```

[4]:
```
13
```

[5]:
```python
d["neighbourhood"].unique()
```

[5]:
```
array(['New Town', 'Southside', nan, 'Leith', 'Old Town', 'West End',
       'Haymarket', 'Morningside', 'Newington', 'Marchmont',
       'Cannonmills', 'Tollcross', 'Bruntsfield', 'Stockbridge'],
      dtype=object)
```

[6]:
```python
d["neighbourhood"].describe()
```

[6]:
```
count     10951
unique       13
top       Leith
freq       3134
Name: neighbourhood, dtype: object
```

1.1.3 Exercise 3

What is the mean and the median price per night of an Airbnb in Edinburgh?

[7]:
```python
print(
    "Mean :", d[["price"]].mean()[0], "\n" # We subset here to get a scalar
    "Med  :", d[["price"]].median()[0]
)
```

```
Mean : 97.2108692319485
Med  : 75.0
```

[8]:
```
d["price"].describe()
```

[8]:
```
count    13046.000000
mean        97.210869
std         86.031393
min          0.000000
25%         49.000000
50%         75.000000
75%        110.000000
max        999.000000
Name: price, dtype: float64
```

1.1.4 Exercise 4

Calculate a new column called `beds_per_bedroom` which is the number of beds divided by the number bedrooms for a listing. For this new column report the 2.5th and 97.5th percentile.

[9]:
```
# If the following fails, make sure the correct version of pandas is loaded (e.
 ↪g. not 1.0.5)
d = d.assign(
    beds_per_bedroom = d.beds / d.bedrooms
).replace(
    float("inf"), float("nan") # this handles cases where # bedrooms is 0.
)

d["beds_per_bedroom"]
```

[9]:
```
0        1.0
1        1.0
2        NaN
3        1.0
4        1.0
        …
13240    2.0
13241    2.0
13242    2.0
13243    1.0
13244    1.0
Name: beds_per_bedroom, Length: 13245, dtype: float64
```

[10]:
```
d.beds_per_bedroom.quantile(q=(0.025,0.975))
```

```
[10]: 0.025    1.0
      0.975    2.5
      Name: beds_per_bedroom, dtype: float64
```

1.2 2. Visualization

For this course we will be using a combination of the libraries **seaborn** and **matplotlib** for the purposes of visualization. The former is actually built using the latter, and is designed to specifically provide a high-level interface for creating statistical graphics.

We will set up some initial configuration details using **matplotlib** to determine the size and resolution of the plots that will be shown in the notebook.

```
[11]: %matplotlib inline

      import matplotlib as mpl
      import matplotlib.pyplot as plt
      import seaborn as sns

      plt.rcParams['figure.figsize'] = (8,5)
      plt.rcParams['figure.dpi'] = 80
```

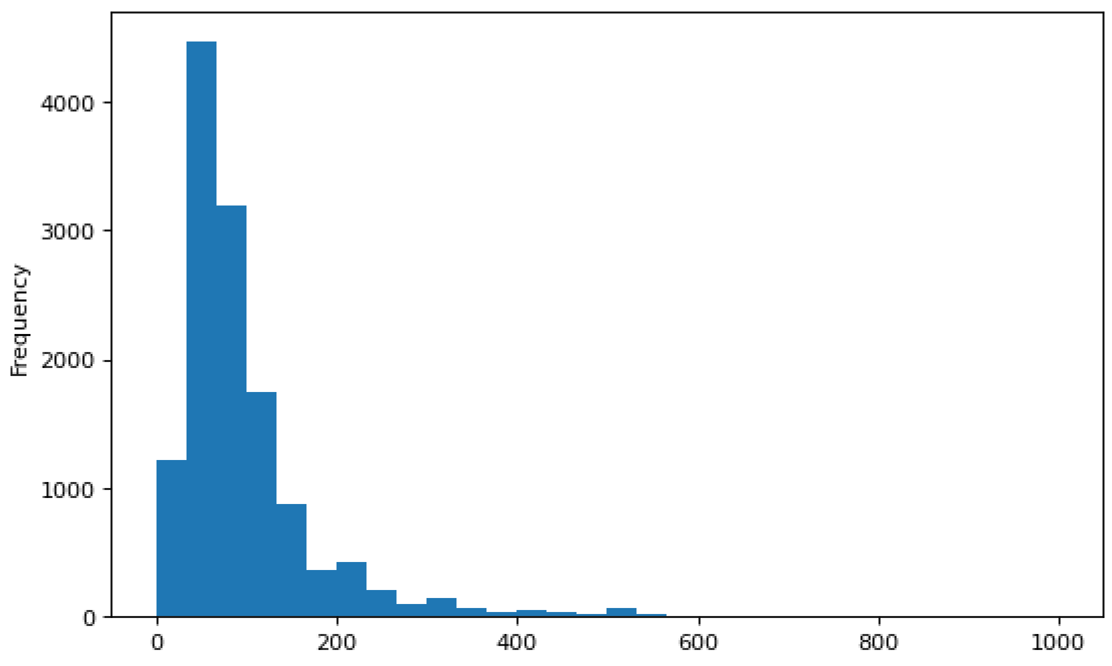
and then we can use pandas and seaborn to visualize the Airbnb data.

1.2.1 2.1 Univariate plots

For example if we want to examine the distribution of the rental prices we can use pandas as follows,

```
[12]: d["price"].plot.hist(bins=30)
```

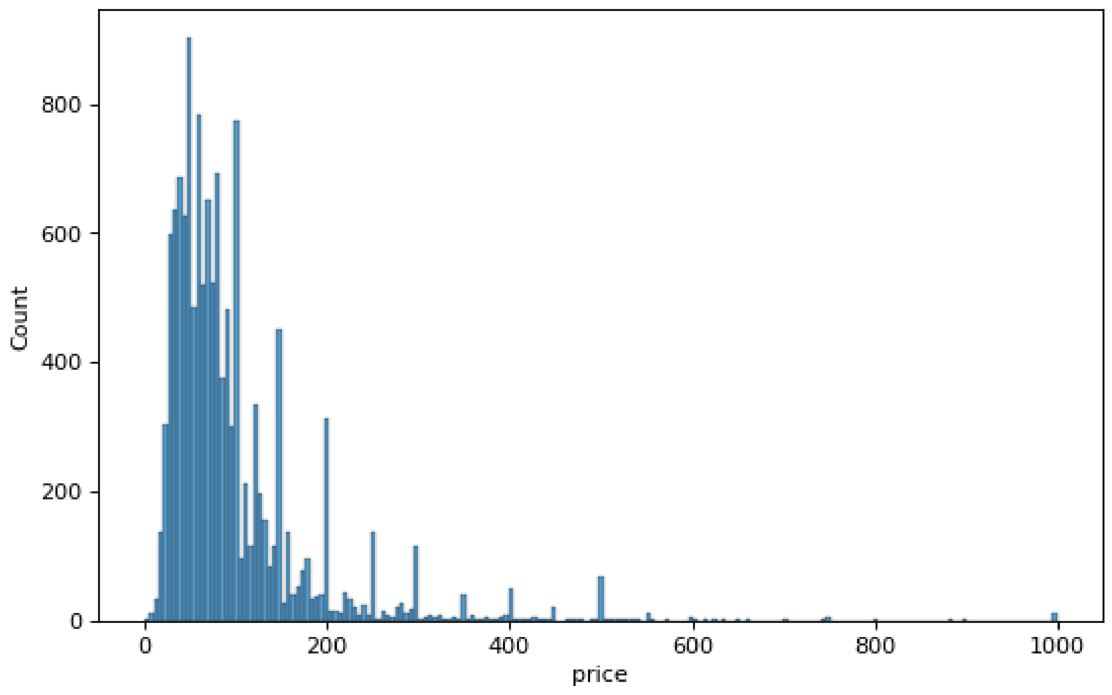
```
[12]: <AxesSubplot:ylabel='Frequency'>
```



We can generate a similar plot using seaborn via the `histplot` function.

```
[13]: sns.histplot(d["price"])
```

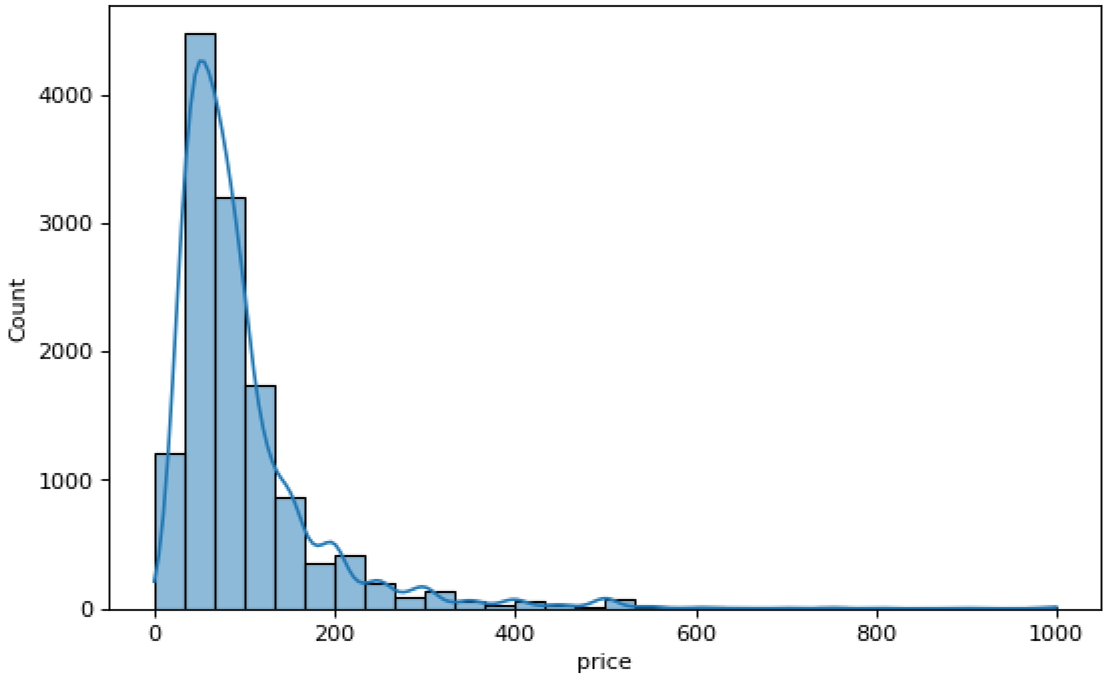
```
[13]: <AxesSubplot:xlabel='price', ylabel='Count'>
```



Like most plots within seaborn, `histplot` includes a large number of arguments which we can use to adjust the plotting behavior. He we adjust the number of bins and add a kernel density estimate to our plot.

[14]:
```
sns.histplot(d["price"], kde=True, bins=30)
```

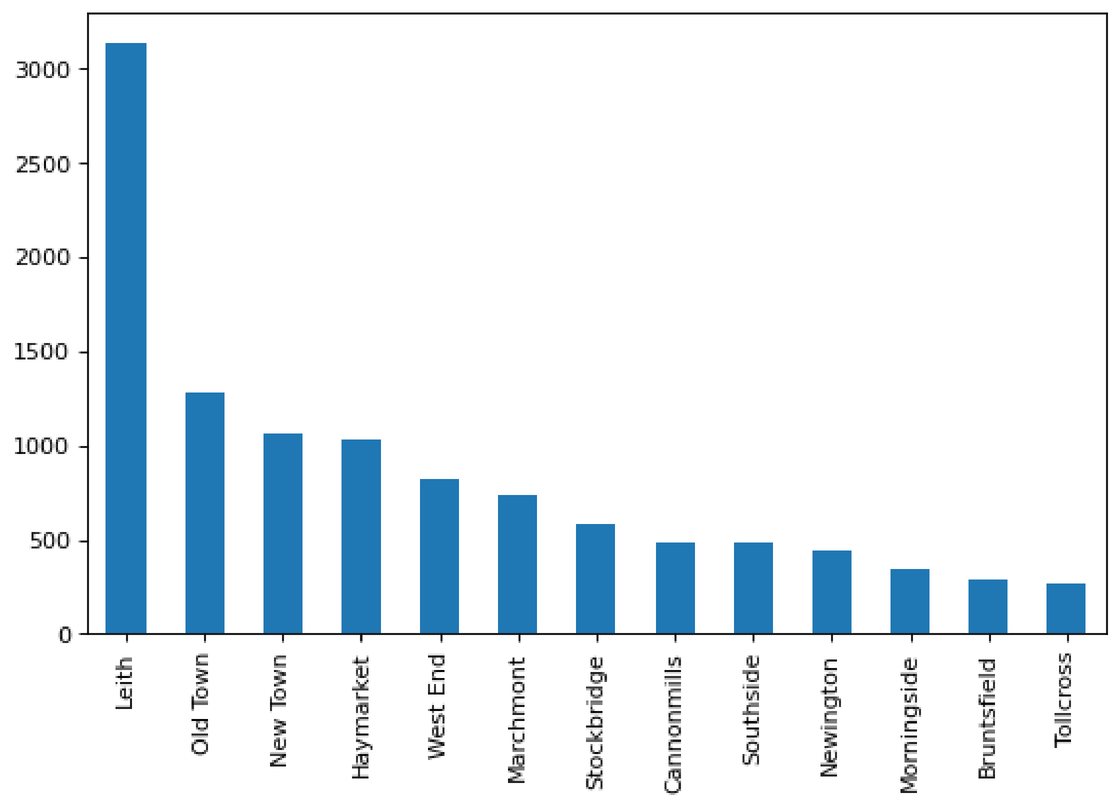
[14]:
```
<AxesSubplot:xlabel='price', ylabel='Count'>
```



We can also examine the distribution of categorical variables by creating a bar plot. This is possible with pandas but somewhat clunky as we have to take care of transforming the variable into the underlying counts of the levels before creating the bar plot.

[15]:
```
d["neighbourhood"].value_counts().plot(kind="bar")
```

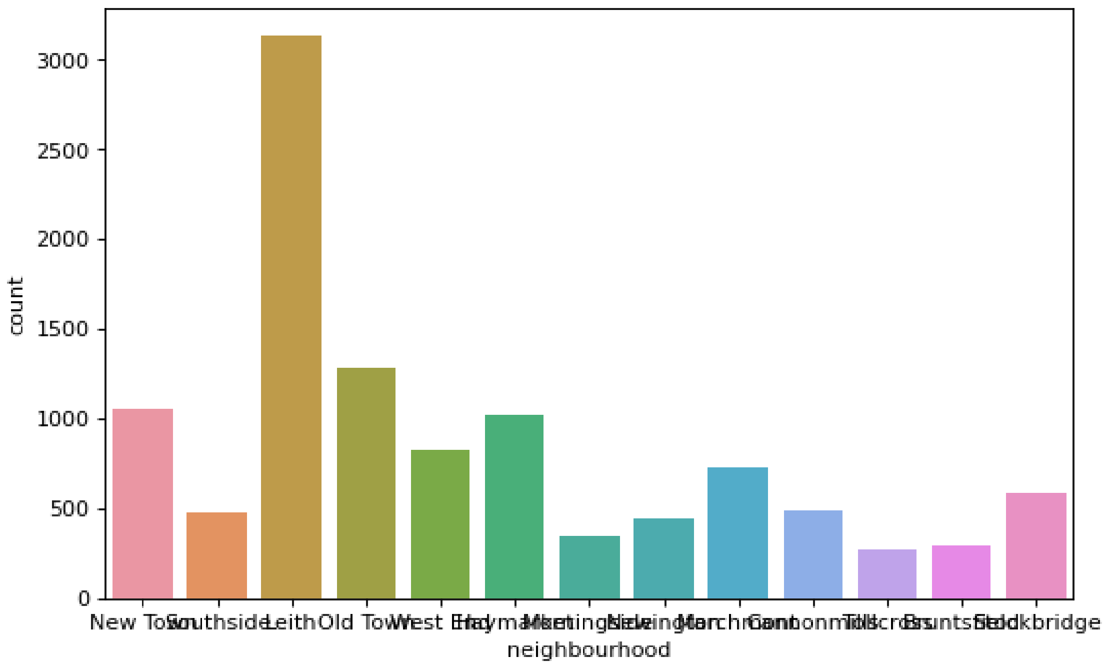
[15]:
```
<AxesSubplot:>
```



A similar plot can be created with seaborn using the `catplot` or `countplot` functions,

[16]:
```
sns.countplot(x="neighbourhood", data=d)
```

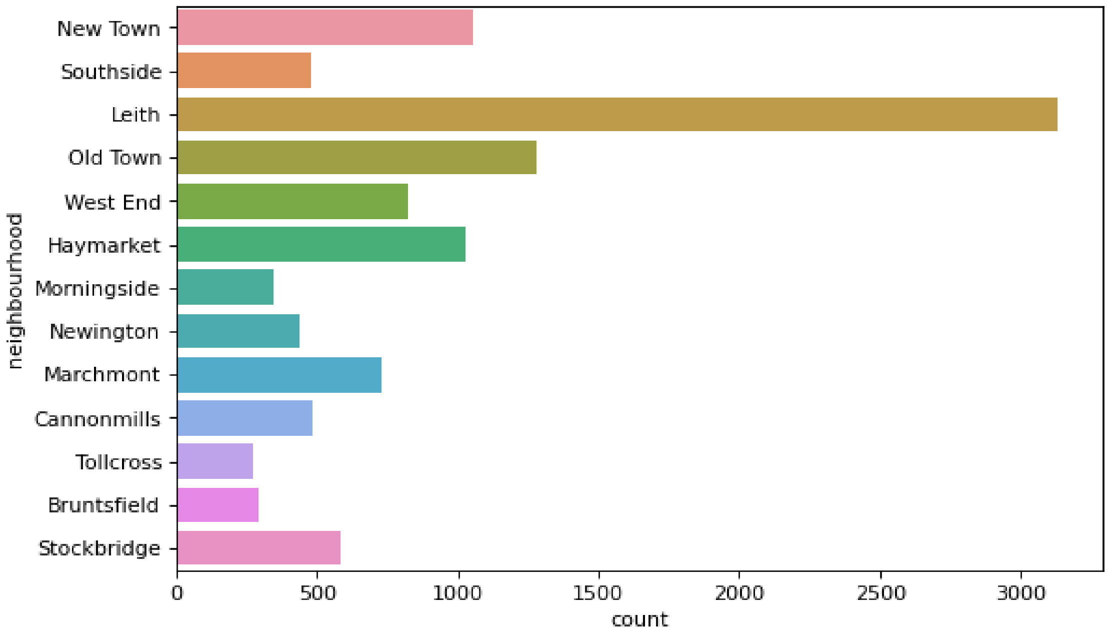
[16]:
```
<AxesSubplot:xlabel='neighbourhood', ylabel='count'>
```



Note that the x-axis labels are overploting making it nearly impossible to read them, one quick fix is to rotate the plot by putting the catergories on the y-axis which can be done as follows,

[17]:
```
sns.countplot(y="neighbourhood", data=d)
```

[17]:
```
<AxesSubplot:xlabel='count', ylabel='neighbourhood'>
```

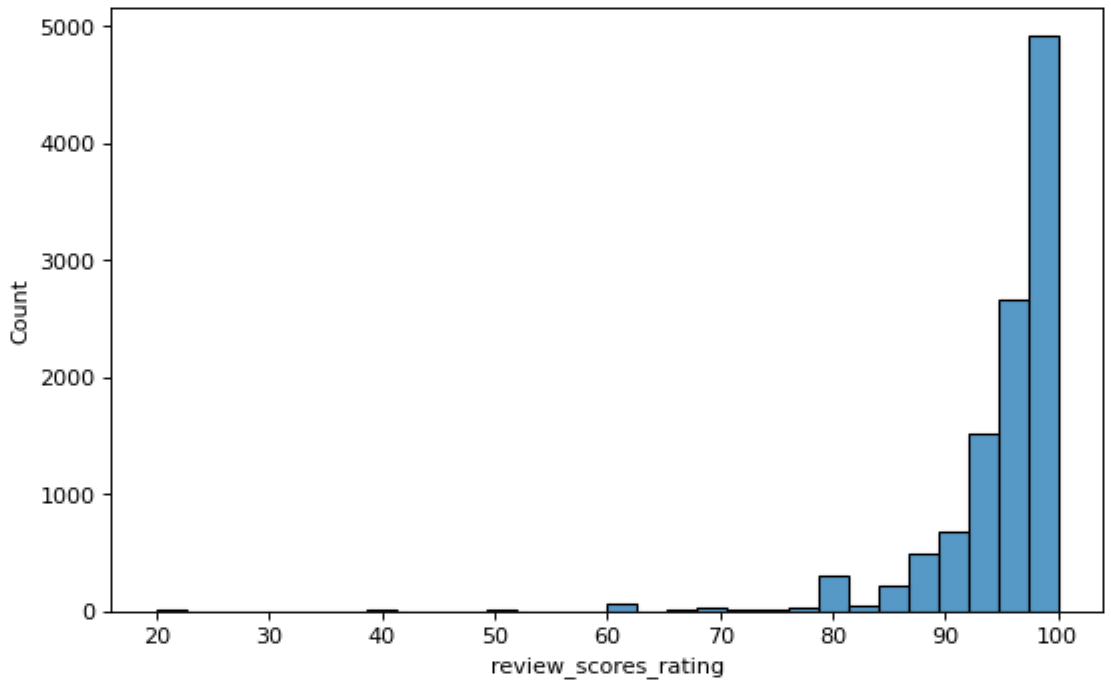


1.2.2 Exercise 5

Create a plot and describe the distribution of the `review_scores_rating` variable.

```
[18]: sns.histplot(d["review_scores_rating"].dropna(), bins=30)
```

```
[18]: <AxesSubplot:xlabel='review_scores_rating', ylabel='Count'>
```



The reviews are a left skewed unimodal distribution with a mode near 100.

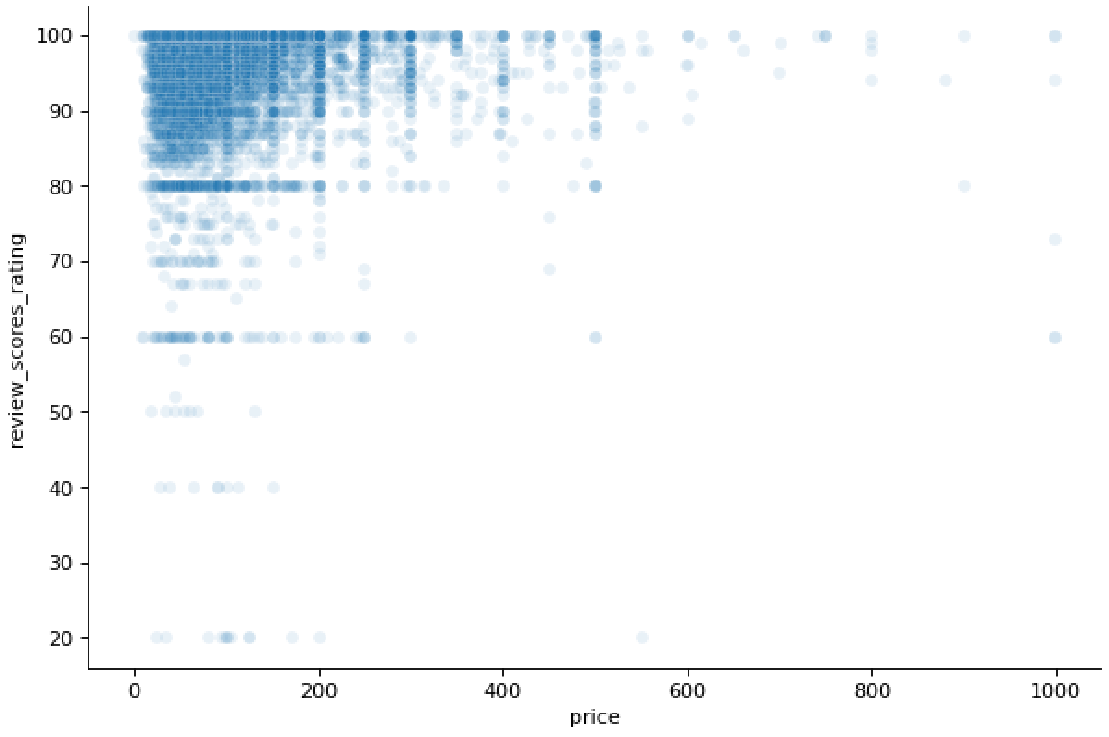
1.3 Multivariate plots

Seaborn also includes a number of functions for visualizing bivariate and multivariate relationships within a data set. The two primary high level functions are `relplot` and `catplot` for plotting numeric or categorical variable relationships respectively.

For example to create a scatter plot of `price` vs `review_scores_rating` we can use `relplot` as follows,

[19]:
```python
sns.relplot(
    x = "price",
    y = "review_scores_rating",
    data = d,
    aspect = 1.5,
    alpha = 0.1
)
```

[19]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2b82b1d950>
```

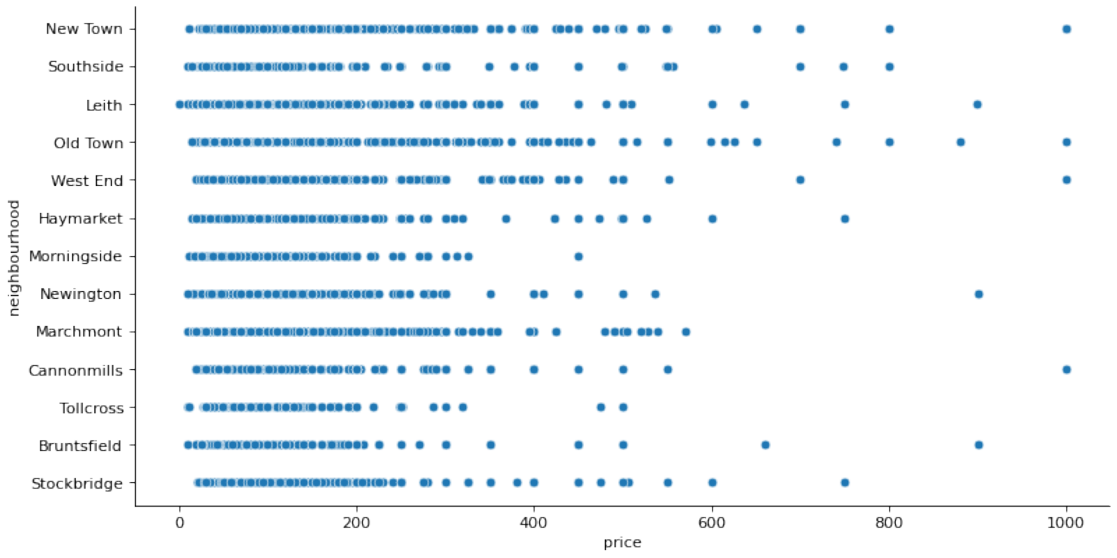


We use the `aspect` argument to adjust the aspect ratio of the plot, making it 1.5 times as wide as it is tall and the `alpha` argument to reduce issues with the over-plotting of points.

Note that `relplot` can also be used with categorical data, the function only determines the type of plot that will be created (i.e. a scatter or line plot).

[20]:
```python
sns.relplot(
    x = "price",
    y = "neighbourhood",
    data = d,
    aspect = 2
)
```

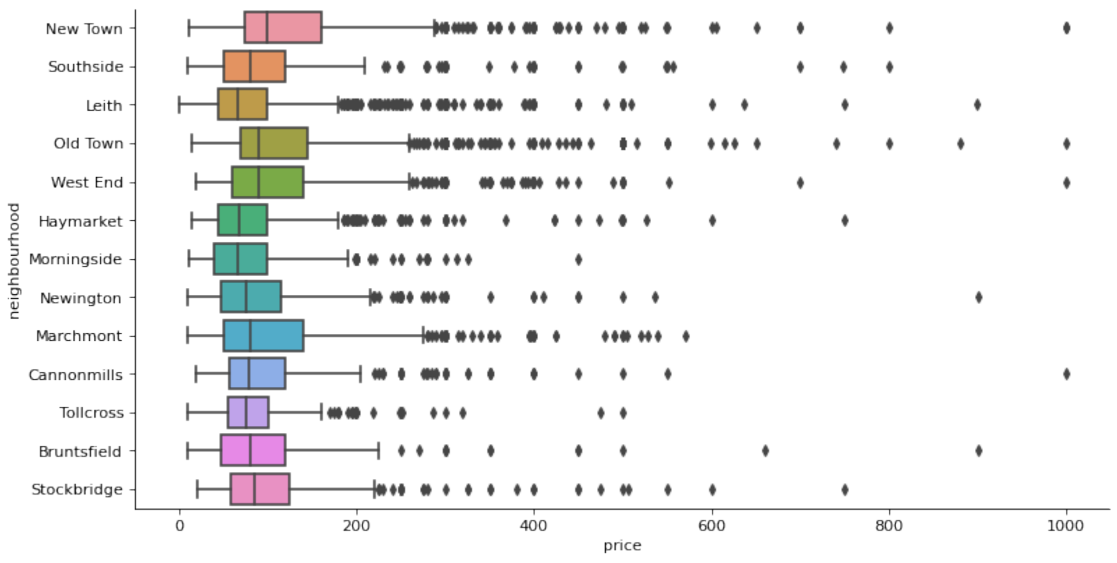
[20]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2b82d3aad0>
```



`catplot` alternatively deals with plots that involve at least one categorical variable (e.g. boxplots, swarm plots, bar plots, etc.). The type of plot is determined by the `kind` argument that is passed to the function. You can try changing this in the cell below and see how it affects the plot. Try values like: `"violin"`, `"bar"`, `"strip"`, or `"point"`.

```
[21]: sns.catplot(
          x = "price",
          y = "neighbourhood",
          kind = "box",
          data = d,
          aspect = 2
      )
```

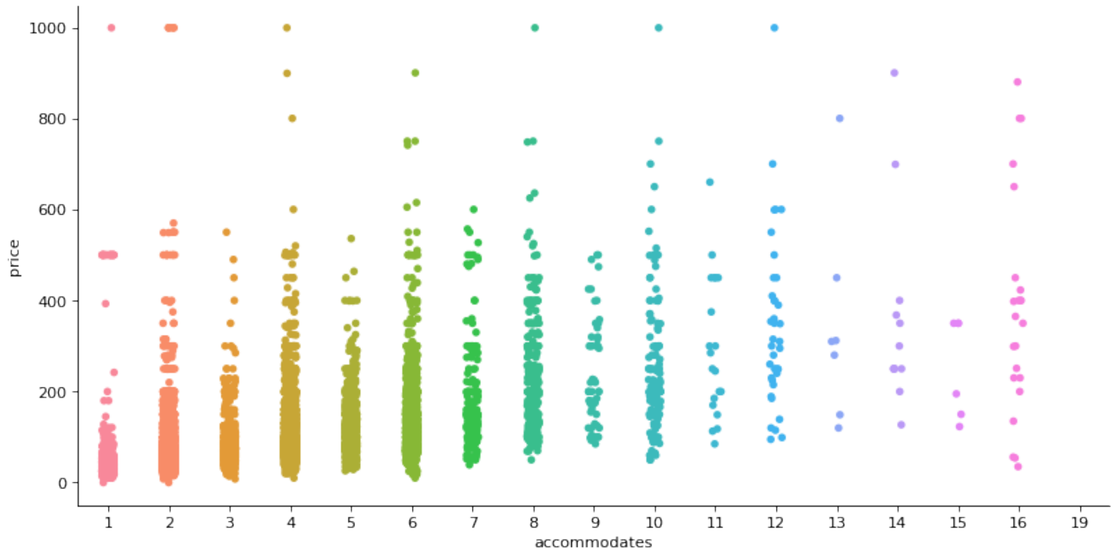
```
[21]: <seaborn.axisgrid.FacetGrid at 0x7f2b82944e50>
```



Just like `relplot` there is not a requirement that both `x` and `y` arguments be categorical variables, but note that when using two numeric variables the `x` variable will be treated as the categorical variable for plotting purposes.

[22]:
```
sns.catplot(
    y = "price",
    x = "accommodates",
    data = d,
    aspect = 2
)
```

[22]:
```
<seaborn.axisgrid.FacetGrid at 0x7f2b829606d0>
```

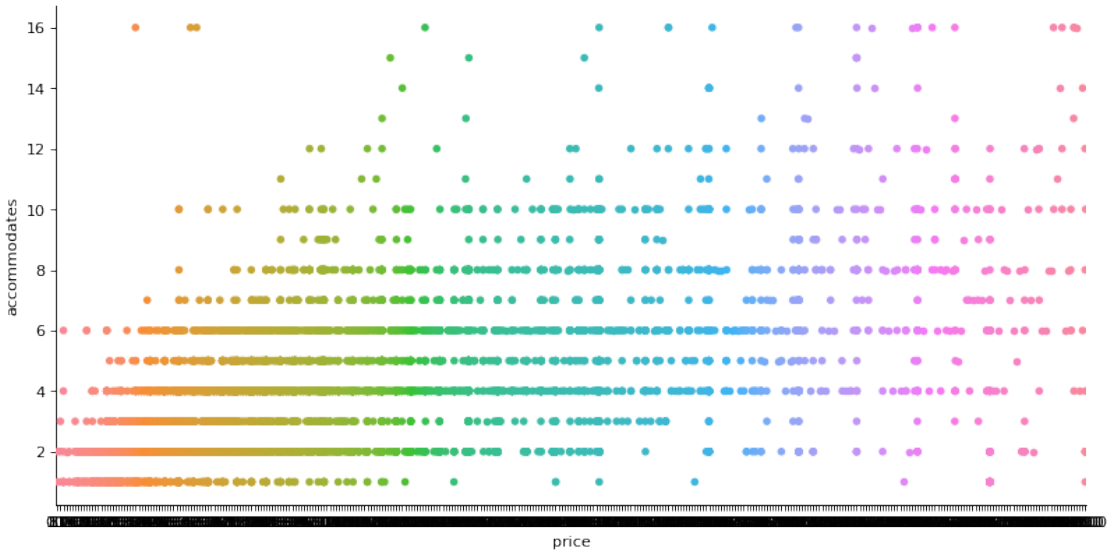


1.3.1 Exercise 6

What happens if you rerun the cell above with the `x` and `y` arguments swapped? To make this behavior even more clear try changing the `kind` to `"box"` for both plots.

```
[23]: sns.catplot(
          x = "price",
          y = "accommodates",
          data = d,
          aspect = 2
      )
```

```
[23]: <seaborn.axisgrid.FacetGrid at 0x7f2b82874210>
```



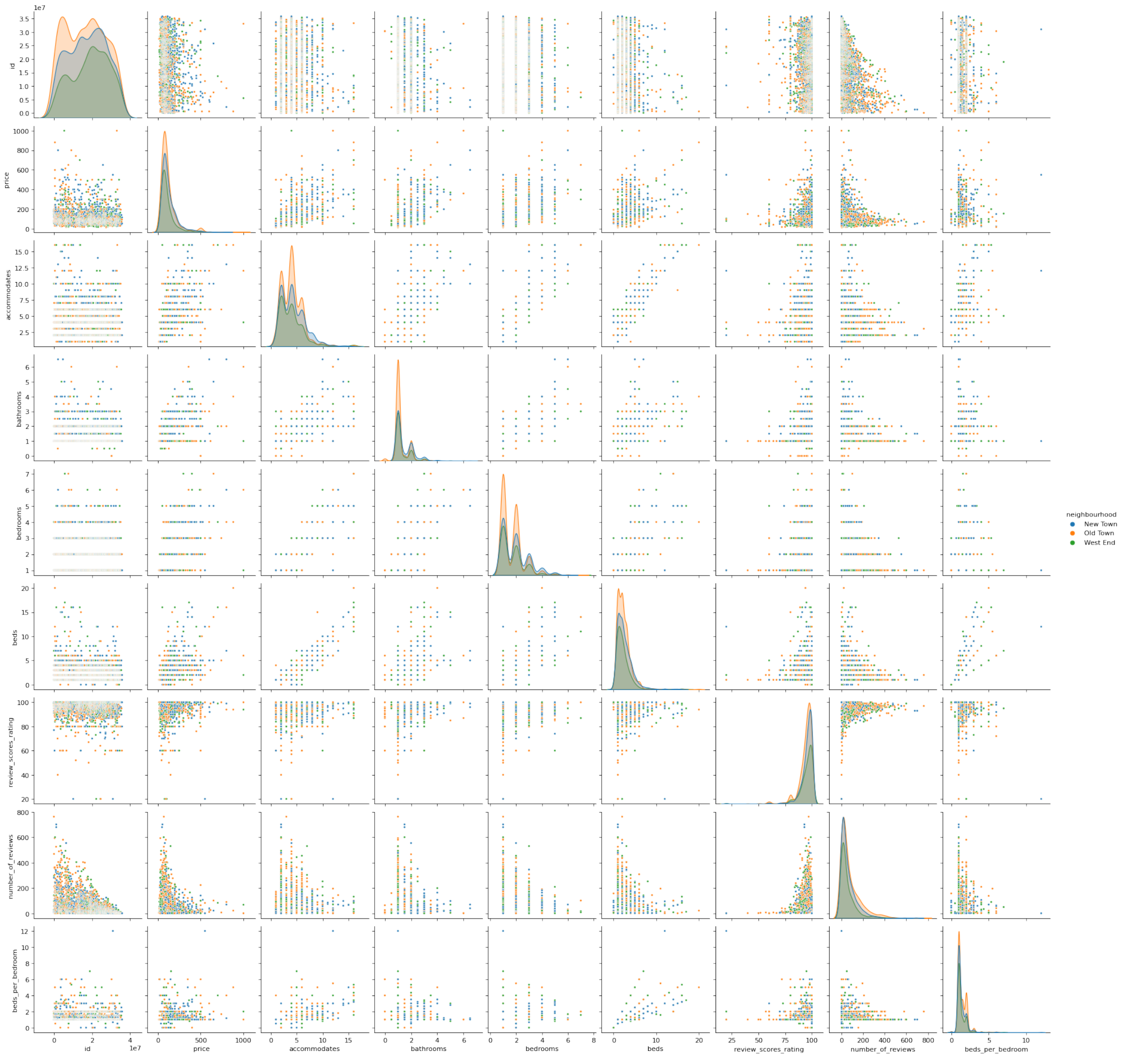
All of the x values are treated as different categories, and are given their own ticks making the x-axis unreadable.

Finally, one other useful tool provided by seaborn is its ability to generate a pairs plot for examining the relationship between many numeric variables at the same time. Here we subset the original data to only include neighbourhoods in the city center and then create a pairs plot for the numeric variables.

```
[24]: center = d.query('neighbourhood in ["New Town", "Old Town", "West End"]')
```

```
sns.pairplot(center.dropna(), hue="neighbourhood", markers=".")
```

[24]: `<seaborn.axisgrid.PairGrid at 0x7f2b872ebb10>`



Hint - if you get an error when running the above code make sure that you have not accidently introduced `Inf` values when you constructed the `beds_per_bedroom` column in **Exercise 4**.

1.3.2 Exercise 7

Pick several other neighbourhoods that are of interest to you and create a pairs plot for them. Is there anything interesting revealed by your plot?

[ ]:
```
student = d.query('neighbourhood in ["Marchmont", "Morningside", "Newington"]')
sns.pairplot(student.dropna(), hue="neighbourhood", markers=".")
```

```
[ ]: <seaborn.axisgrid.PairGrid at 0x7f2b82a99550>
```

There are not any obvious patterns, there does not seem to be much difference betweent the three neighborhoods.

Additional information, documentation, and examples can be found at the seaborn website. The tutorial and gallery sections are of particular use for new users.

1.4 3. Competing the worksheet

At this point you have hopefully been able to complete all the preceeding exercises. Now is a good time to check the reproducibility of this document by restarting the notebook's kernel and rerunning all cells in order.

Once that is done and you are happy with everything, you can then run the following cell to generate your PDF.

```
[ ]: !jupyter nbconvert --to pdf mlp-week01.ipynb
```

Once generated this PDF can be submitted to Gradescope under the `mlp-week01` assignmnet. This must be done by January 20th at 5 pm in order to receive credit for this workshop. See the getting started with Gradescope screencast for the necessary steps for both individual and team submissions.