

Week 4: Visualising Two Variables

Visual Data Analytics

University of Sydney



Outline

- Categorical v Categorical
 - Side by side bar
- Categorical v Numerical
 - Side by side boxplots
- Numeric v Numerical
 - Scatter plot

Motivation

- Understand the distributions of two variables
 - Find outliers
 - Find multi-modality
 - Find skew
- Also understand the relationship between the variables

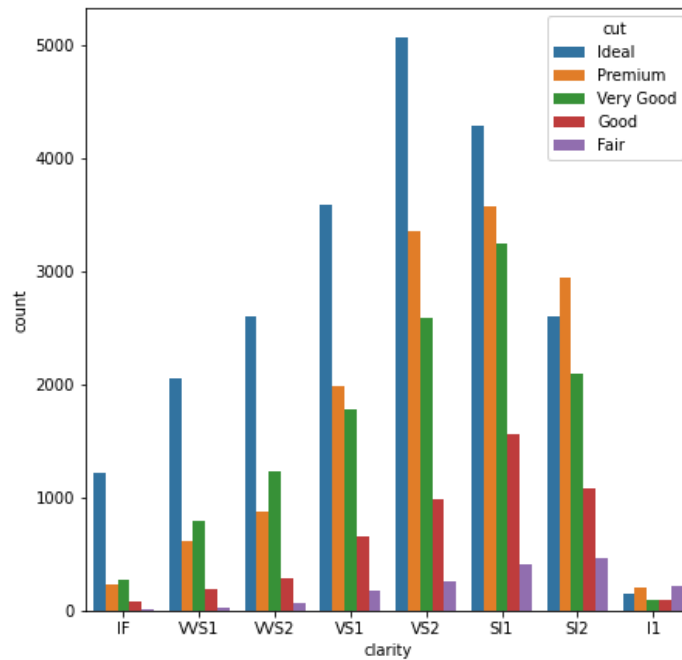
Categorical v Categorical variables

Bar charts

- To visualise two nominal/ordinal variables bar charts can still be used to plot the frequency of each category.
- There are two options
 - Grouped bar charts
 - Stacked bar charts
- We will investigate using the taxis and diamonds data also used last week.

Grouped bar chart

```
sns.countplot(data=diam, x='clarity', hue='cut')
```



Grouped bar chart

- Can easily make comparisons such as
 - Fair diamonds are relatively more common for worse levels of clarity (e.g. SI2)
 - For IF diamonds, these overwhelmingly have an ideal cut.
- Harder to make a comparison for
 - The most common category of clarity.

Stacked bar chart

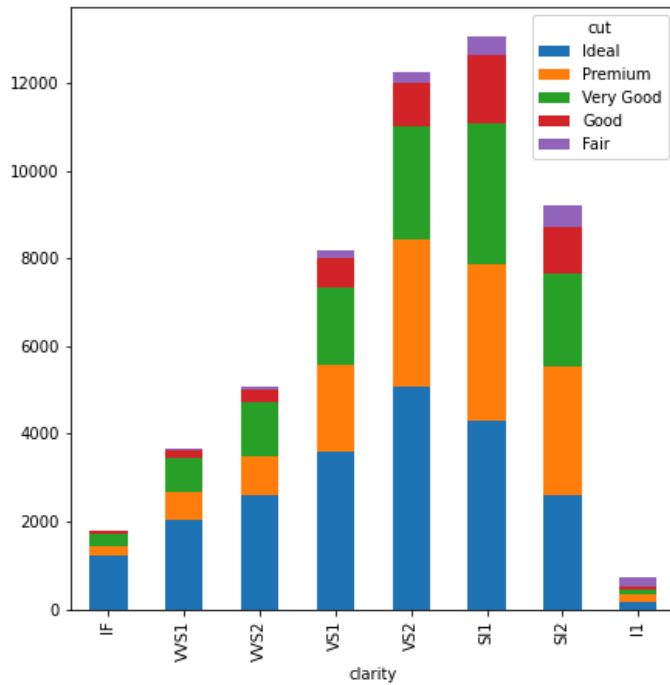
First must prepare data by creating cross tab

```
diamct = pd.crosstab(diam['clarity'],diam['cut'])  
diamct
```

## cut	Ideal	Premium	Very Good	Good	Fair
## clarity					
## IF	1212	230	268	71	9
## VVS1	2047	616	789	186	17
## VVS2	2606	870	1235	286	69
## VS1	3589	1989	1775	648	170
## VS2	5071	3357	2591	978	261
## SI1	4282	3575	3240	1560	408
## SI2	2598	2949	2100	1081	466
## I1	146	205	84	96	210

Stacked bar chart

```
diamct.plot(kind = 'bar', stacked = True)
```



Stacked bar chart

- Can easily see
 - Frequencies of categories for clarity
 - Relative frequency of cut within a single category of clarity
- Harder to make a comparison for
 - Frequency of cut across categories of clarity (e.g. hard to quickly tell if there are more Good/VS2 or Good/SI2)

Percentage stacked bar chart

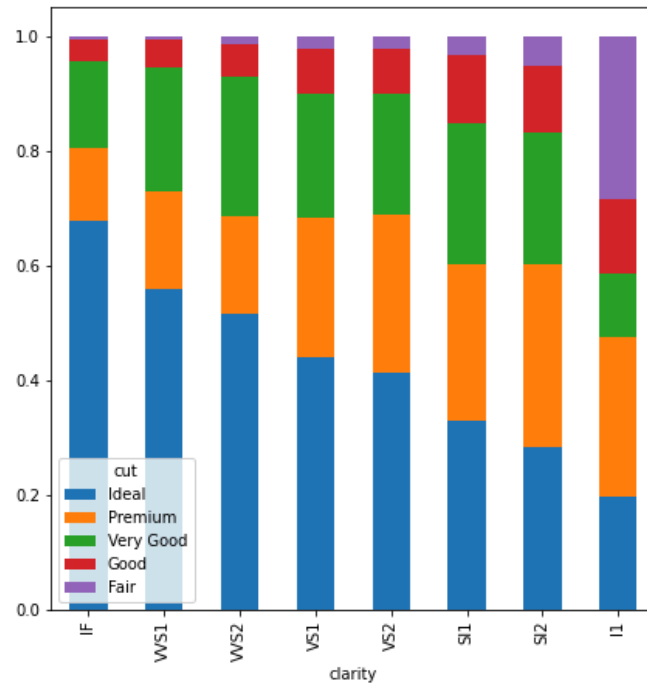
First must prepare data

```
diamct = pd.crosstab(diam['clarity'],diam['cut'], normalize='index')  
diamct
```

## cut	Ideal	Premium	Very Good	Good	Fair
## clarity					
## IF	0.677095	0.128492	0.149721	0.039665	0.005028
## VVS1	0.560055	0.168536	0.215869	0.050889	0.004651
## VVS2	0.514410	0.171733	0.243782	0.056455	0.013620
## VS1	0.439236	0.243422	0.217232	0.079305	0.020805
## VS2	0.413689	0.273862	0.211372	0.079785	0.021292
## SI1	0.327746	0.273632	0.247991	0.119403	0.031228
## SI2	0.282576	0.320753	0.228410	0.117577	0.050685
## I1	0.197031	0.276653	0.113360	0.129555	0.283401

Stacked bar chart

```
diamct.plot(kind = 'bar', stacked = True)
```



Percentage stacked bar chart

- Can easily see
 - The relationship between cut and clarity
- Harder to see
 - Anything about the distribution of the categories

Bar charts

- Three options
 - Grouped
 - Stacked
 - Percentage stacked
- No one is right or wrong, it all depends on what you are trying to communicate.

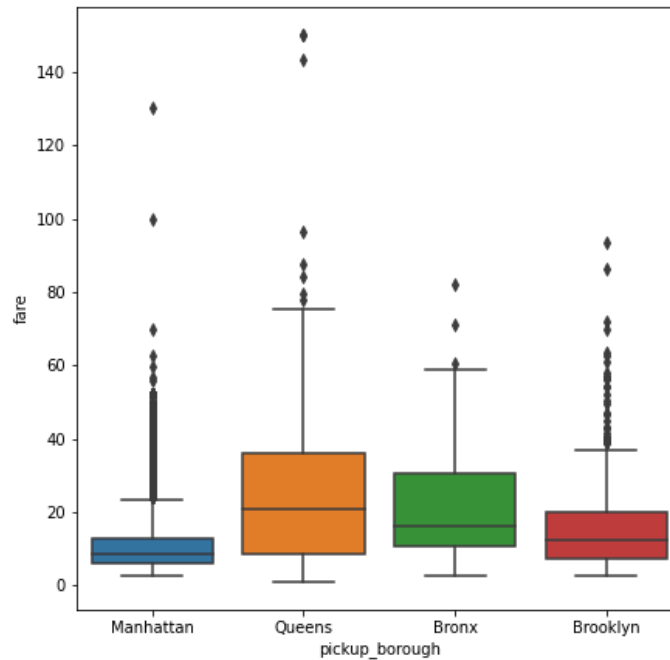
Categorical v Numeric

Side-by-side

- Will consider side-by-side
 - Boxplots
 - Violin plots
 - (Jittered) rug plots
- Seaborn syntax well suited to these

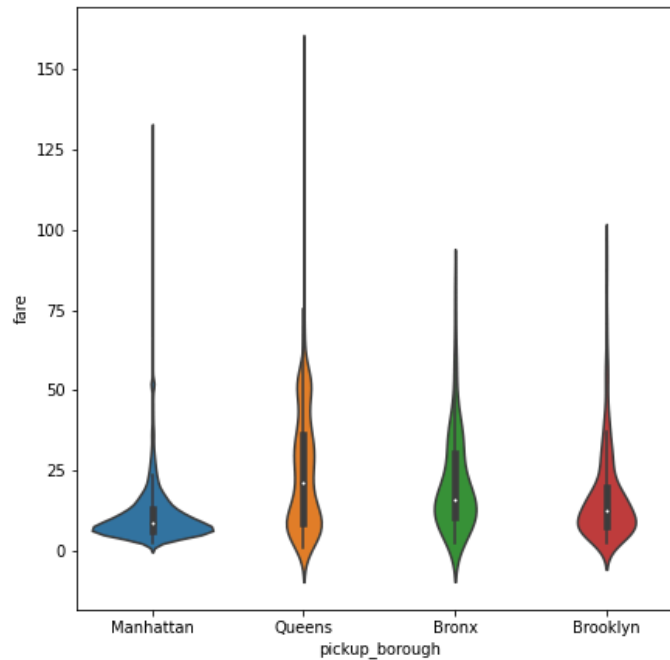
Side by side boxplots

```
sns.boxplot(data = taxisdat, y = 'fare', x = 'pickup_borough')
```



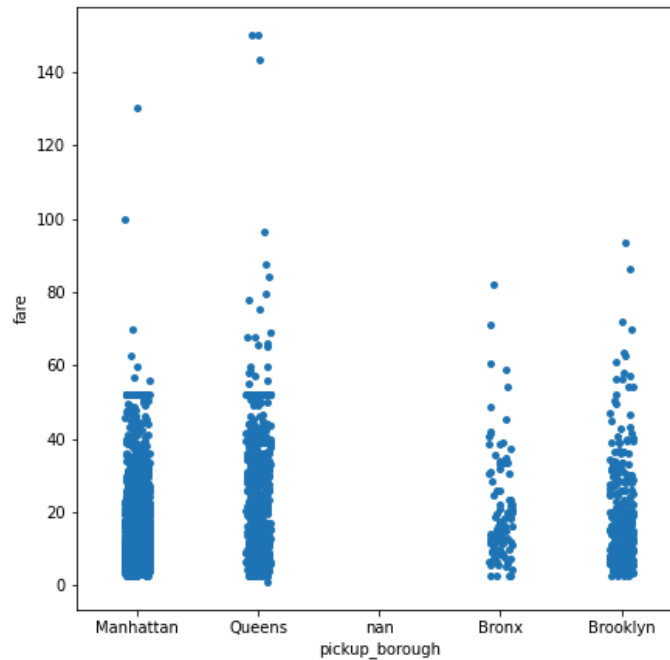
Side by side violin plots

```
sns.violinplot(data = taxisdat, y = 'fare', x = 'pickup_borough')
```



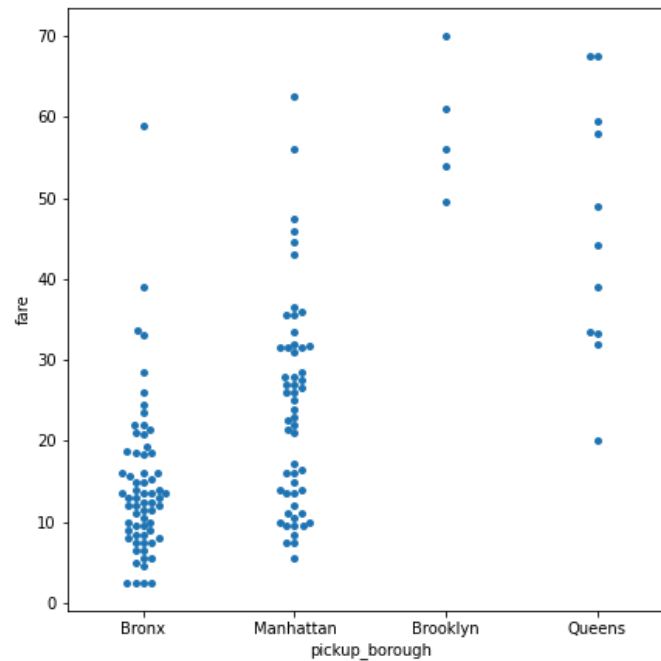
Side by side strip plots

```
sns.stripplot(data = taxisdat, y = 'fare', x = 'pickup_borough')
```



Side by side swarm plots (slow)

```
sns.swarmplot(data = taxisdat[taxisdat['dropoff_borough']=='Bronx'], y :
```



Different plots

- Boxplots clearly show that Manhattan has the lowest median fare and less volatility in the fare.
- Violin plots show three modes in the Queens distribution (cannot be seen using boxplots).
- Strip plots show that frequent value at around \$50 occurs for Manhattan and Queens.
- No single correct plot, use the plot that tells your story.

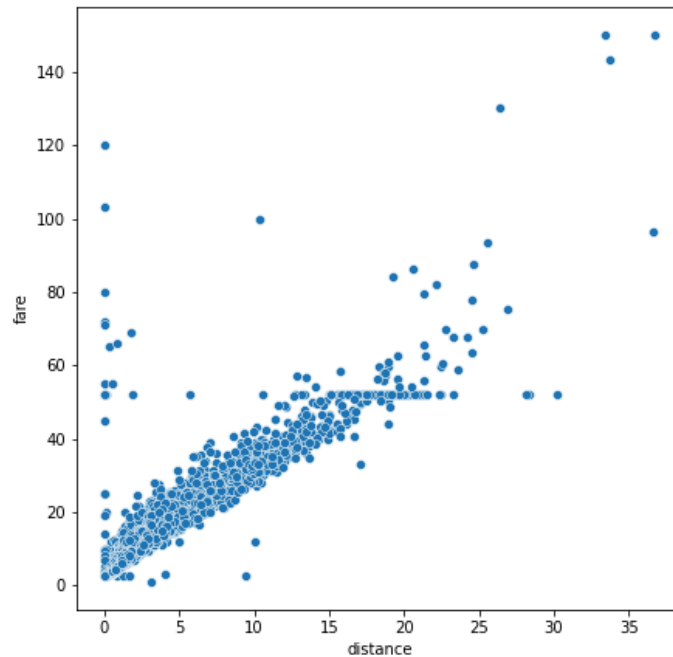
Numerical v Numerical

Scatter plot

- The scatter plot plots one numerical variable on the x axis and the other on the y axis.
- It is a very common plot and useful for
 - Finding outliers
 - Understanding the relationship between two variables.
- We will also look at
 - Log transformations
 - Strategies for dealing with big data.

Scatterplot

```
sns.scatterplot(data = taxisdat, y = 'fare', x = 'distance')
```

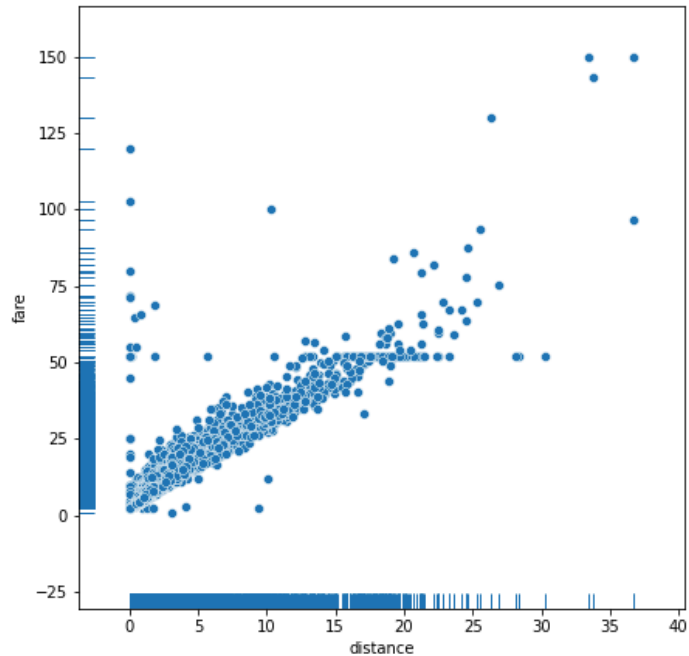


What do we see?

- Can still see certain distributional characteristics
 - Frequent values around \$50
 - Outliers
- Relationship is positive and linear
- There are some observations with low distance but high fares.
 - This could not be seen by only looking at each distribution individually.

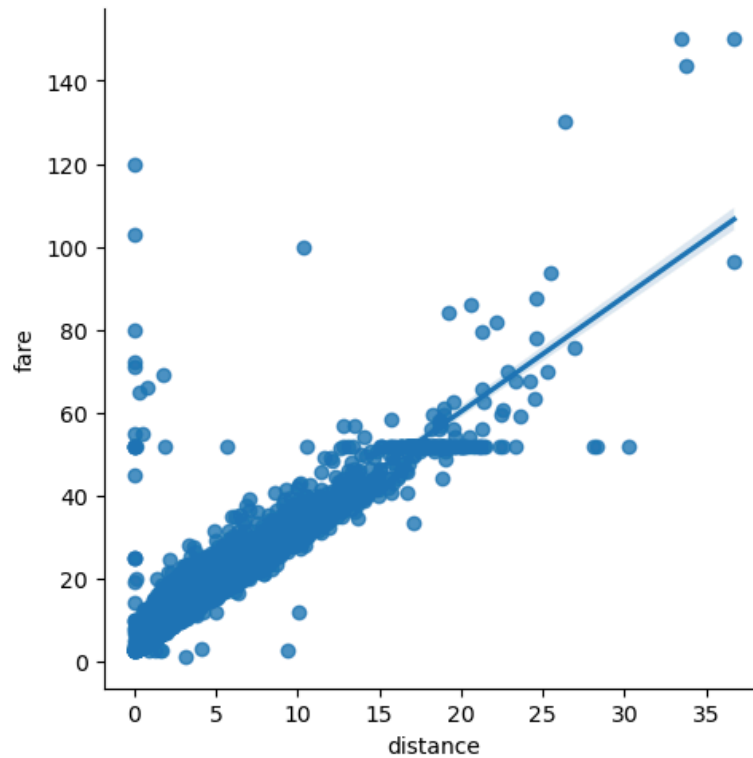
Adding rug plot

```
sns.scatterplot(data = taxisdat, y = 'fare', x = 'distance')  
sns.rugplot(data = taxisdat, y = 'fare', x = 'distance')
```



Adding regression line

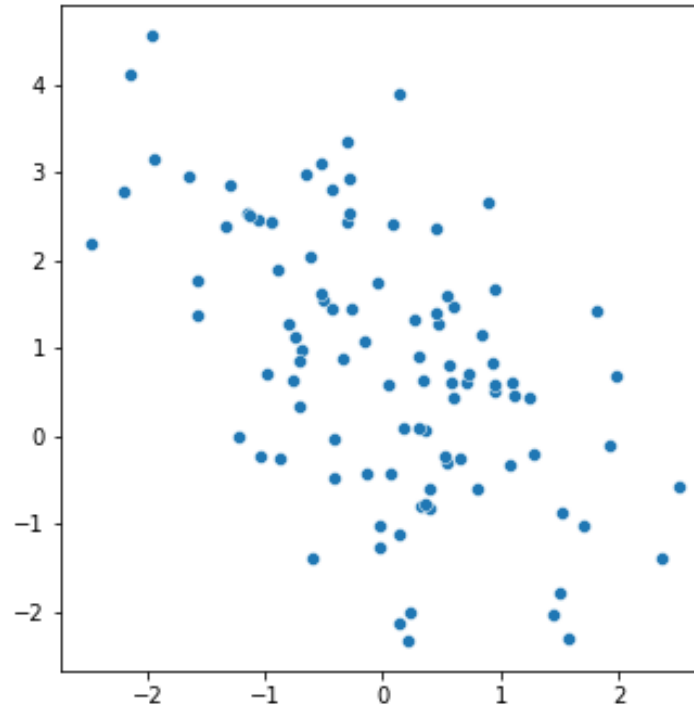
```
sns.lmplot(data = taxisdat, y = 'fare', x = 'distance')
```



Regression

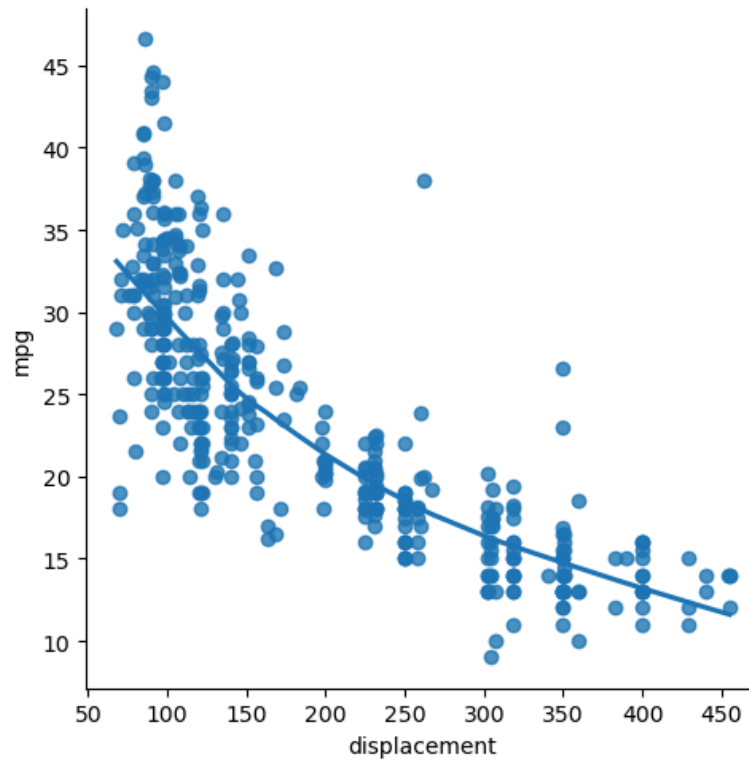
- We usually think of x as an independent variable and y as a dependent variable.
- Even if you do not include regression lines, if there are natural choices of independent and dependent variable then use them on the x and y axes respectively.
- For instance, in this example, distance determines price rather than the other way around.
- Always remember that correlation does not imply causality.

Example of negative correlation



Example of non-linear relationship

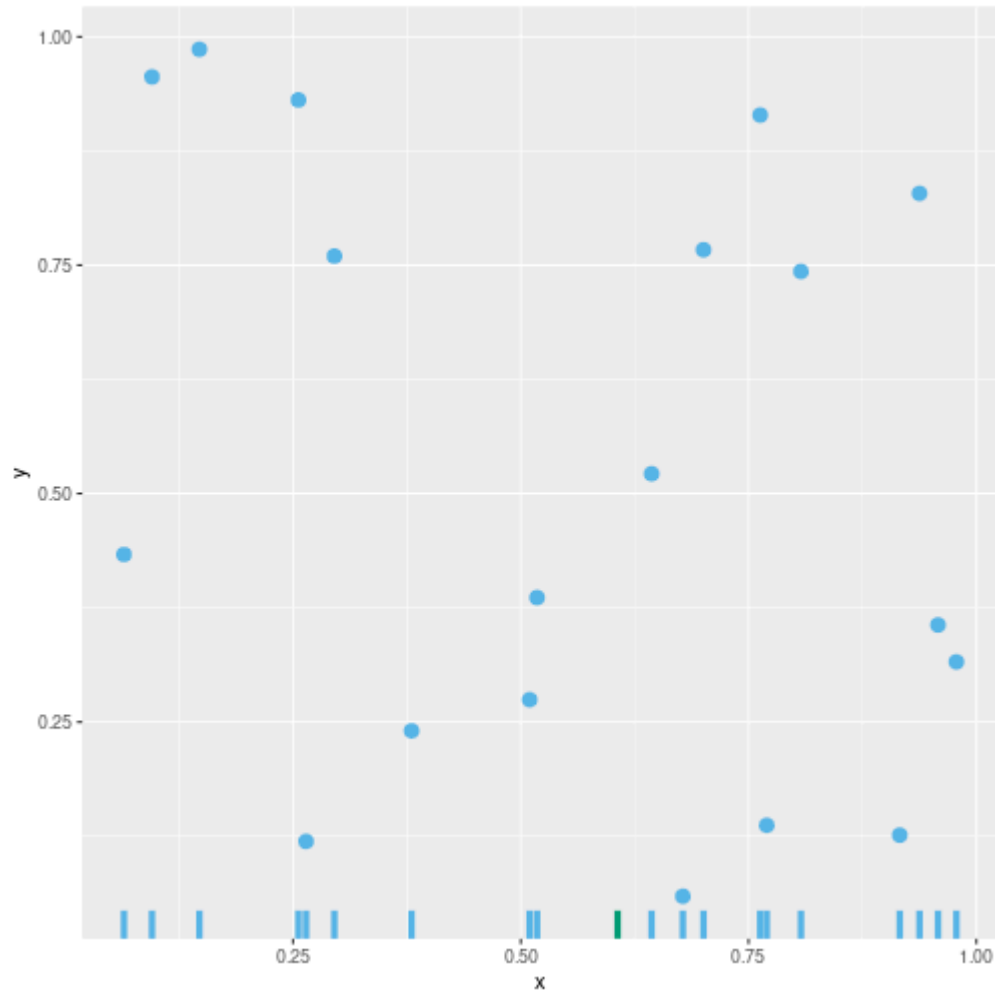
```
mpg = sns.load_dataset('mpg')  
sns.lmplot(data = mpg, y = 'mpg', x = 'displacement', lowess=True)
```



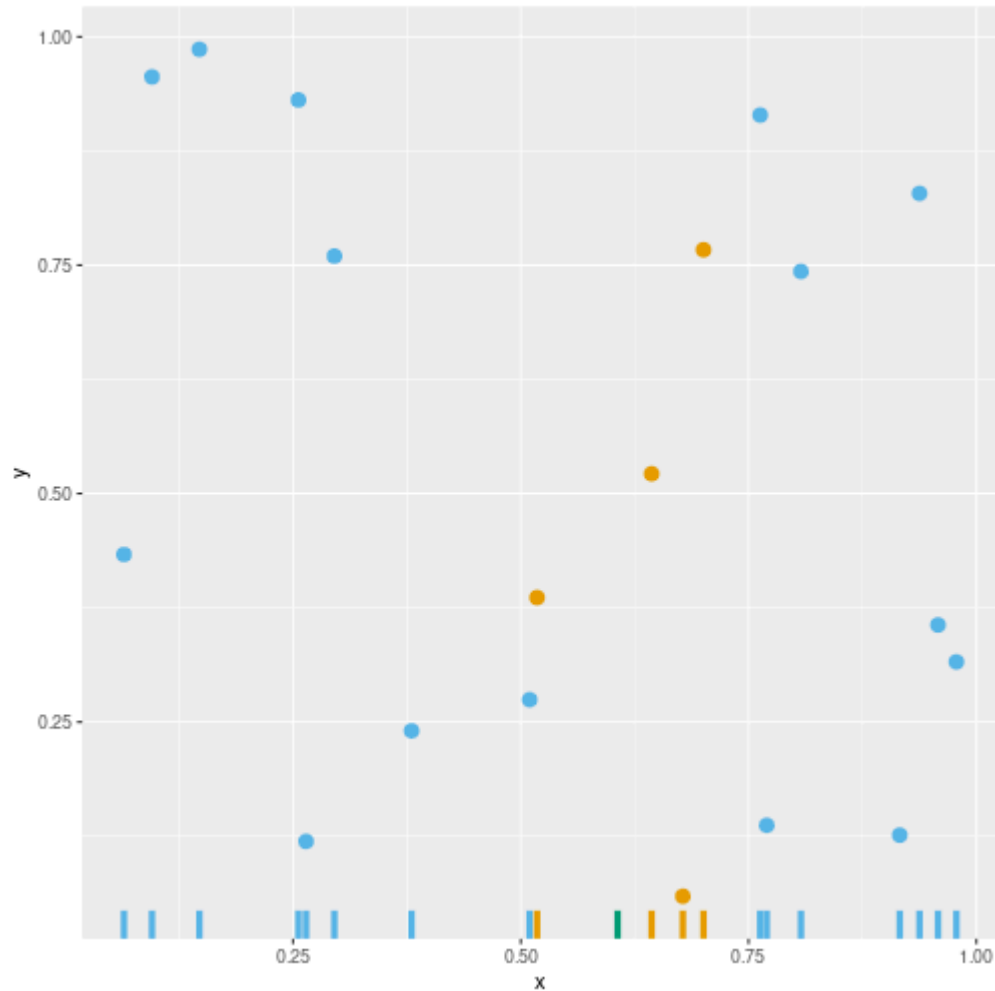
Smooth fit

- The fitted line comes from the LOcal Weighted Estimated Scatterplot Smoothing (LOWESS) method.
- This technique combines the idea of nearest neighbours with regression.
- Nearest neighbours are found (along the x-axis)
- A constant, linear or quadratic regression is fit to the nearest neighbours.

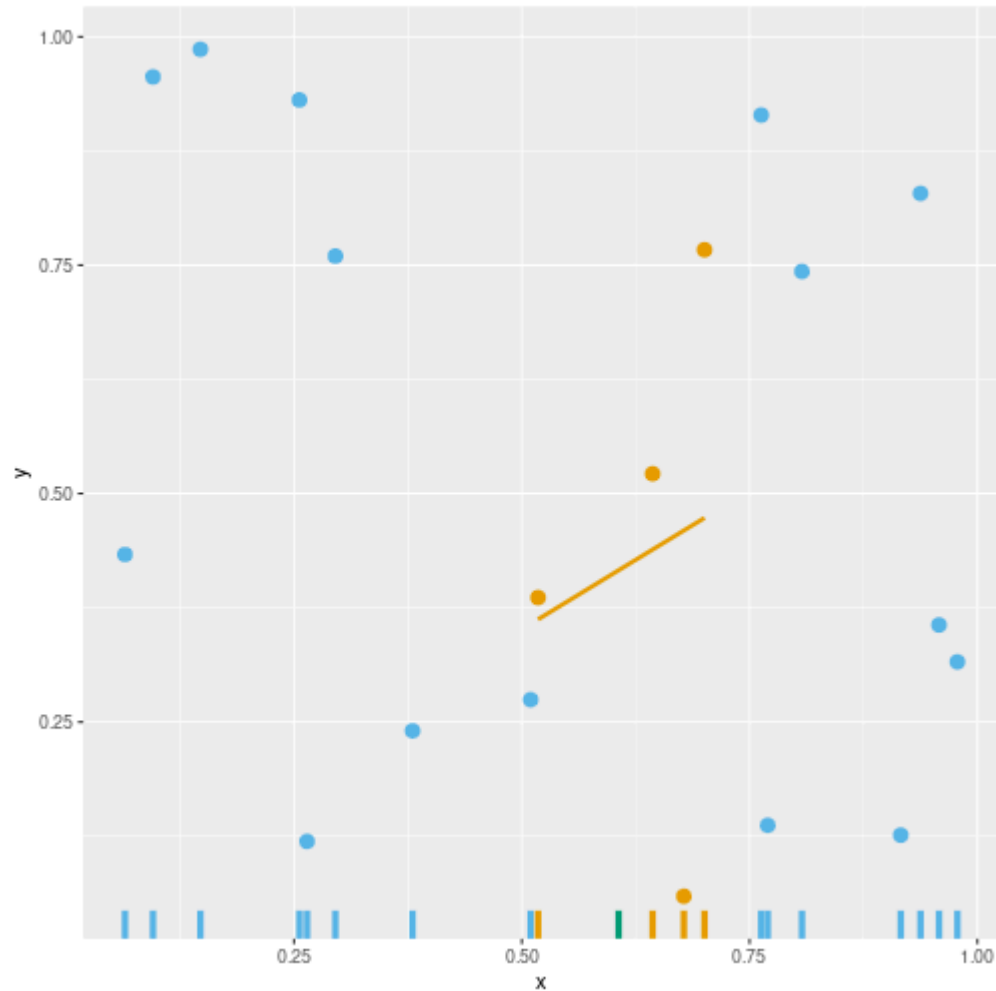
Nearest Neighbours



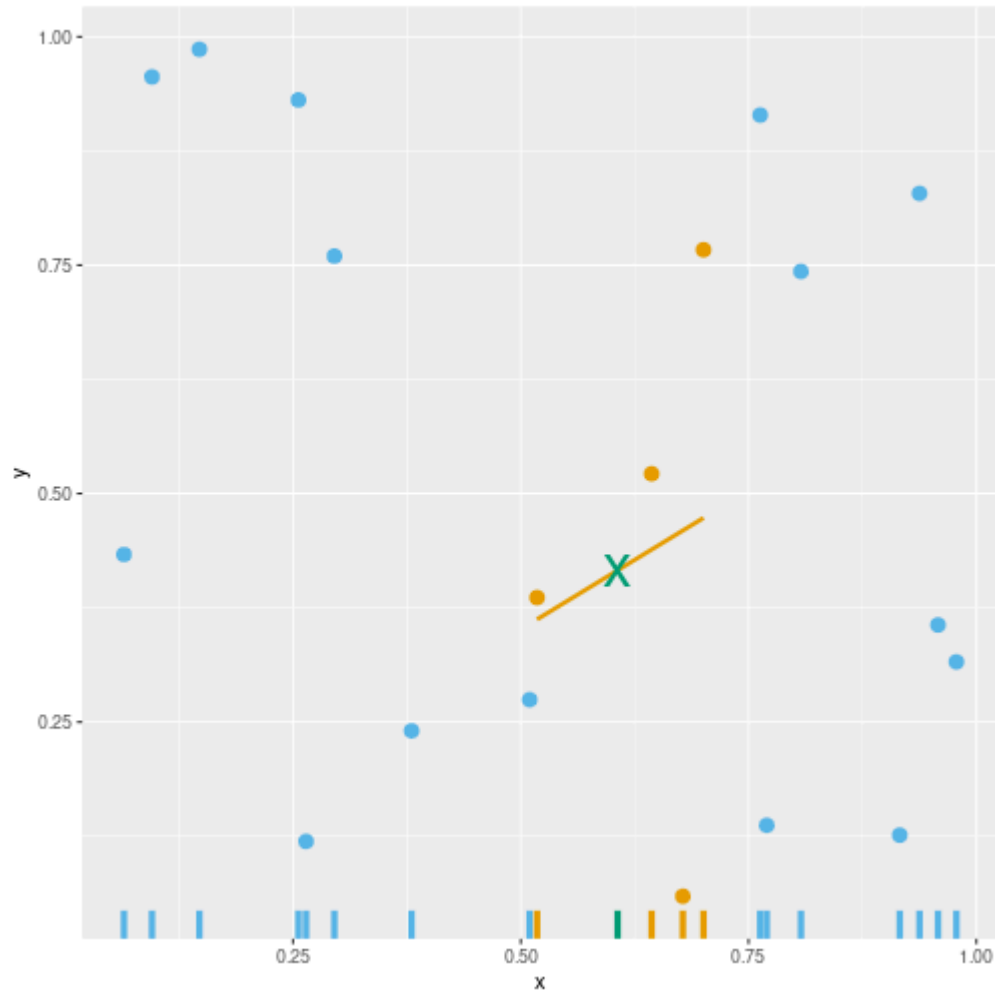
Nearest Neighbours



Local fit



Nearest Neighbours



Details

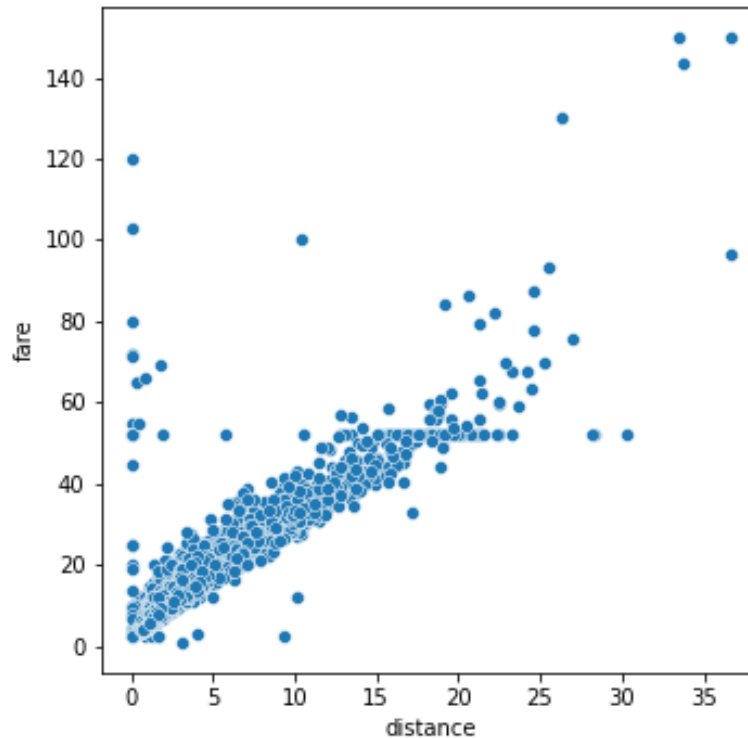
- In the LOWESS algorithm, the smoothing parameter we estimate a regression function ($\hat{y} = f(x)$) rather than a regression line.
- The smoothness of this function can be tuned by changing the number of nearest neighbours.
- Local regressions estimated by weighted least squares so that closer neighbours are given more influence.
- LOWESS is not ideal for large datasets.

Log transformations

- For positively skewed data it is often worth looking at a log transformation
- It particularly makes sense for relationships best understood in terms of percentage changes
 - For example prices change by 1%, demand changes by 5%.
- This transformation only works for positive data.

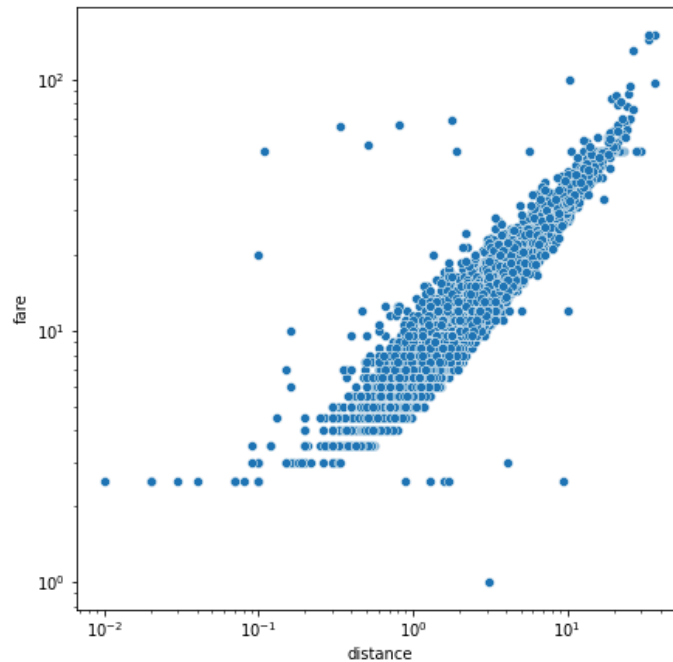
Without transformation

```
sns.scatterplot(data = taxisdat, y = 'fare', x = 'distance')
```



With transformation

[None, None]



Using log scales

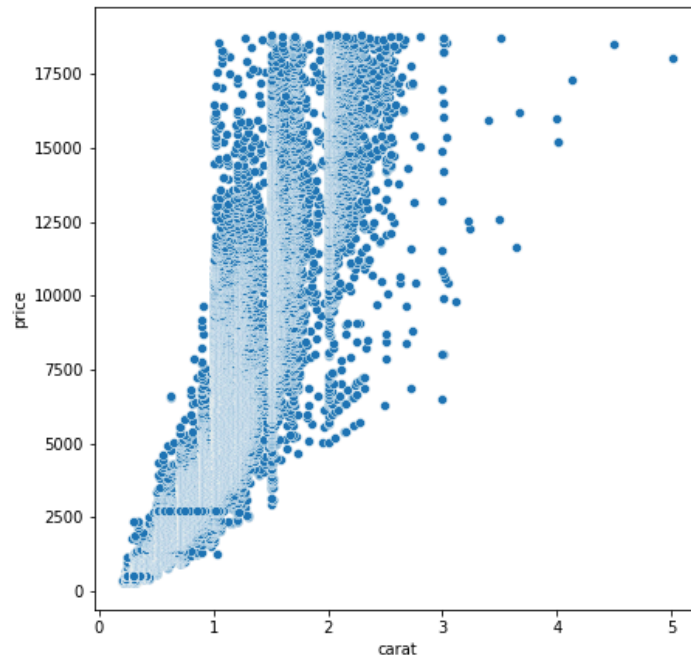
- Before using the log scale the values in the bottom left corner were too close to one another.
- By using the log-log scale these values can be seen more clearly.
- One thing to notice is that the fare has clusters around certain values.
- You will cover using log scales in a tutorial.

Overplotting

- With datasets with a large number of observations, points are plotted on top of one another.
- This is known as *overplotting*
- There are a few strategies to avoid this
 - Smaller points.
 - Transparent points.
 - Contour plot (of 2D KDE).
 - Binning.
- Can demonstrate using diamonds data (53940 observations).

Scatterplot

```
sns.scatterplot(data = diam, y = 'price', x = 'carat')
```

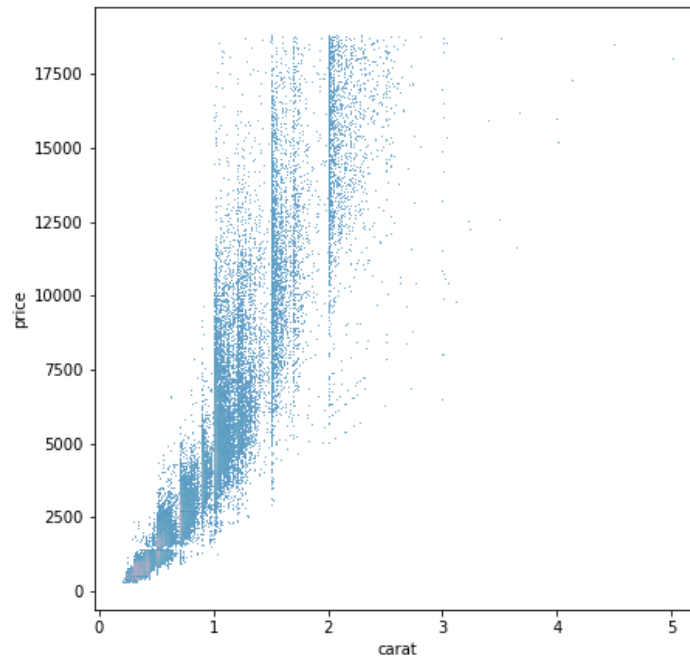


Overplotting

- There are so many points in the region towards the bottom left of the plot that points are on top of one another.
- This causes issues with how the plot is rendered.
- It is difficult to tell where any modes are.
- We will construct the same plot but with different strategies to deal with overplotting.

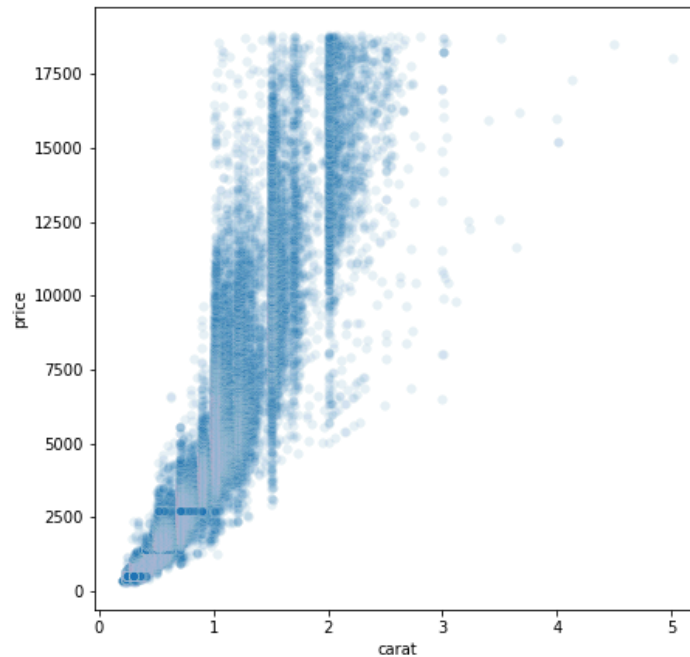
Smaller points

```
sns.scatterplot(data = diam, y = 'price', x = 'carat', s=1)
```



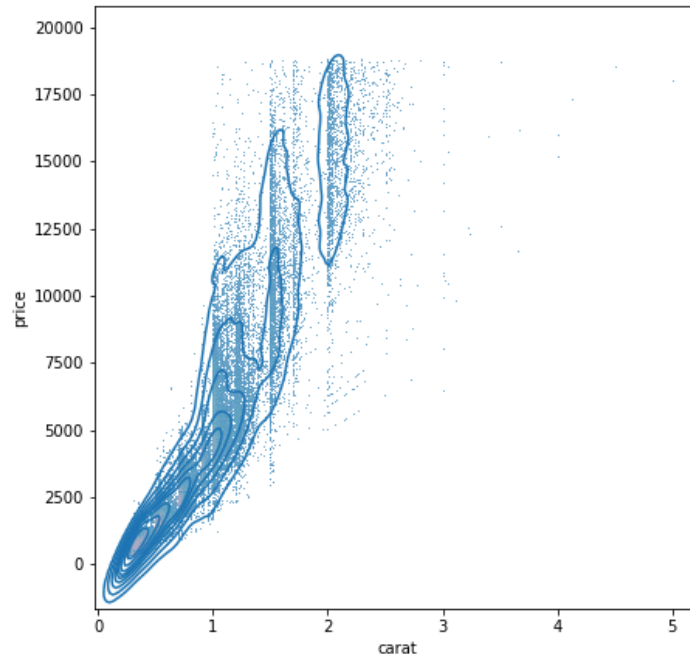
Transparent points

```
sns.scatterplot(data = diam, y = 'price', x = 'carat', alpha=0.1)
```



Add KDE contours

```
sns.scatterplot(data = diam, y = 'price', x = 'carat', s=1)  
sns.kdeplot(data = diam, y = 'price', x = 'carat')
```

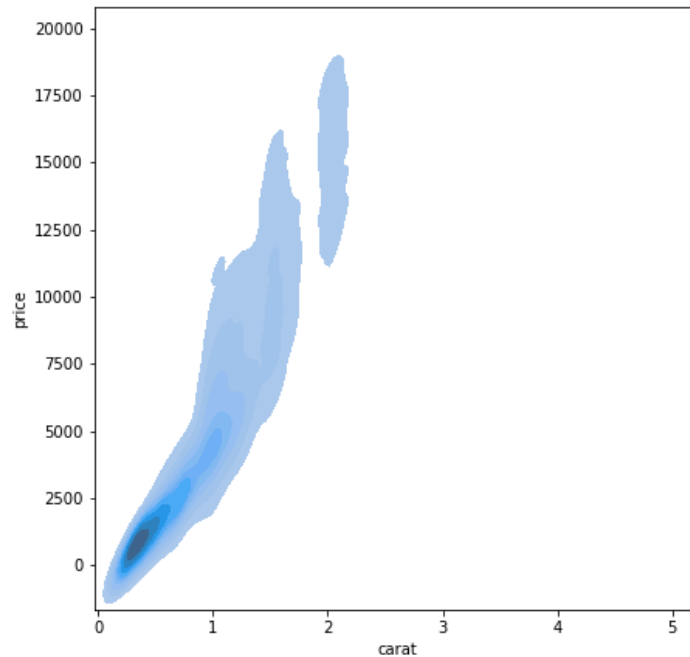


Two dimensional KDE

- Kernel function now takes a vector as an input and returns a single number as an output.
- Think of each kernel as a pile of sand on a table, and the value of the function is the height of the sand.
- The contour lines give all the values of price and carat that evaluate to the same density value.
- If any of you are familiar with elevation maps used in geography (e.g for mountain climbers), this is the same idea.

With shading

```
sns.kdeplot(data = diam, y = 'price', x = 'carat', shade = True)
```

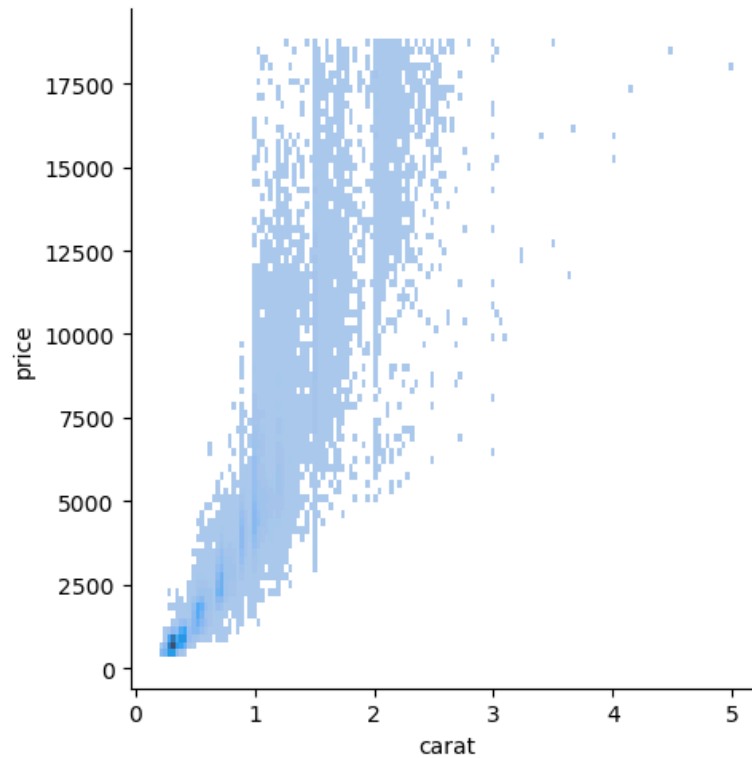


Binning

- Recall a KDE is a smooth version of a histogram.
- We can plot a "2D histogram" by binning the data into rectangles and using color to represent the frequency of observations in each rectangle.
- This can be useful for locating modes in bivariate data.

Binning

```
sns.displot(data = diam, y = 'price', x = 'carat')
```



Discussion

- The advantage of binning (compared to a KDE) is that you can still see outliers
- However the KDE is a "smoother" plot that can still be overlaid with a scatterplot.
- Again there is no right or wrong plot, it depends on the story you want to tell.

Wrap-up

Conclusions

- Visualising two variables helps us to understand about the distribution of the variables including
 - Modes
 - Outliers
- In addition we can understand more about the relationship between the two variables
 - Direction of relationship.
 - Whether relationship is linear
- Log transformations can be useful to help visualise variables.
- With big data, important to avoid overplotting.

Questions