Week 3: Visualising One Variable

Visual Data Analytics

University of Sydney

Outline

- Nominal/Ordinal Data
  - Bar
  - Lollipop
  - Pie/donut
- Numeric data
  - Box plot
  - Histograms
  - Kernel density

Motivation

- Understand the distribution of a variable
  - Find outliers
  - Find multi-modality
  - Find skew
- Understanding the distribution is about generating interesting questions for further analysis.
- Thinking probabilistically is about thinking about distributions and not just the mean.

Examples

- We will use two dataets that can be directly loaded from the `seaborn` package.
  - The `taxis` dataset with data on pickup and drop off locations, fares, payment type etc., in New York City.
  - The `diamonds` dataset with information on size, cut clarity, price, etc. of diamonds.
- These contain categorical (nominal and ordinal) and numeric variables.

Categorical variables

The bar chart

- Categories displayed on one axis (usually x).
- The *frequency* of each observation is displayed on the other axis (usually y).
- The frequency is mapped to the *length* of each bar.
- For this reason always include zero on the y axis.

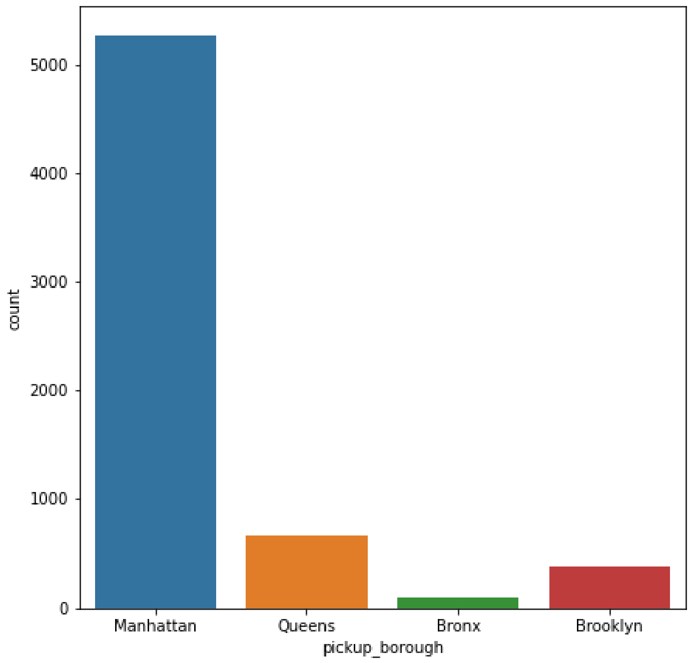
Taxis data

```python
import seaborn as sns
taxisdat = sns.load_dataset('taxis')
taxisdat
```

```
##                     pickup             dropoff  ...  pickup_borough  dropoff_
## 0    2019-03-23 20:21:09 2019-03-23 20:27:24  ...       Manhattan        Ma
## 1    2019-03-04 16:11:55 2019-03-04 16:19:00  ...       Manhattan        Ma
## 2    2019-03-27 17:53:01 2019-03-27 18:00:25  ...       Manhattan        Ma
## 3    2019-03-10 01:23:59 2019-03-10 01:49:51  ...       Manhattan        Ma
## 4    2019-03-30 13:27:42 2019-03-30 13:37:14  ...       Manhattan        Ma
## ...                  ...                 ...  ...             ...
## 6428 2019-03-31 09:51:53 2019-03-31 09:55:27  ...       Manhattan        Ma
## 6429 2019-03-31 17:38:00 2019-03-31 18:34:23  ...          Queens
## 6430 2019-03-23 22:55:18 2019-03-23 23:14:25  ...        Brooklyn         B
## 6431 2019-03-04 10:09:25 2019-03-04 10:14:29  ...        Brooklyn         B
## 6432 2019-03-13 19:31:22 2019-03-13 19:48:02  ...        Brooklyn         B
##
## [6433 rows x 14 columns]
```

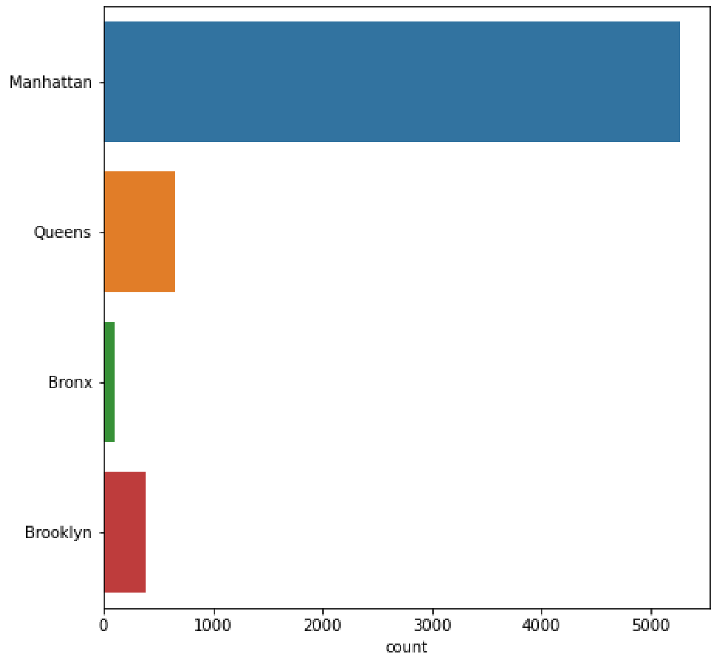
Bar plot of pick up borough

```
sns.countplot(data = taxisdat, x='pickup_borough')
```



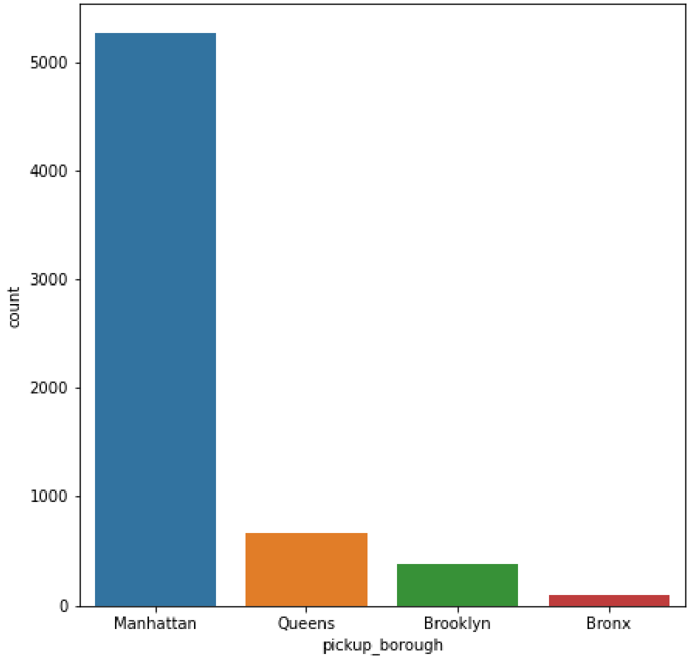
Change orientation

```
sns.countplot(data = taxisdat, y='pickup_borough')
```



Order by frequency

```
sns.countplot(data = taxisdat, x='pickup_borough', order = taxisdat['pi
```



Data are nominal - this is fine.

Ordinal data

- For nominal data it is suitable, to order according to frequency.
- This is not the case for ordinal data
- Always order according to categories of the variable.
- Diamonds dataset has clarity as an ordinal variable
  - Categories ordered as IF, VVS1, VVS2, VS1, VS2, SI1, SI2, I1.

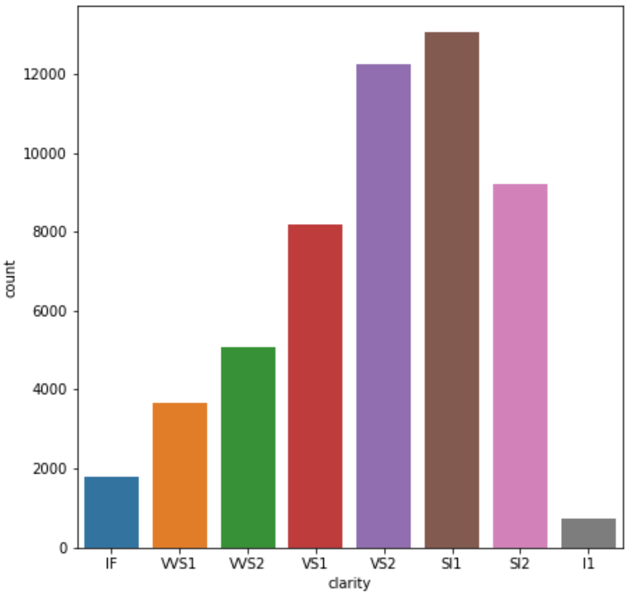
Diamonds data

```
diam = sns.load_dataset('diamonds')
diam
```

```
##          carat        cut color clarity  depth  table  price     x     y   
## 0         0.23      Ideal     E     SI2   61.5   55.0    326  3.95  3.98  2.4
## 1         0.21    Premium     E     SI1   59.8   61.0    326  3.89  3.84  2.3
## 2         0.23       Good     E     VS1   56.9   65.0    327  4.05  4.07  2.3
## 3         0.29    Premium     I     VS2   62.4   58.0    334  4.20  4.23  2.6
## 4         0.31       Good     J     SI2   63.3   58.0    335  4.34  4.35  2.7
## ...        ...        ...   ...     ...    ...    ...    ...   ...   ...   ..
## 53935     0.72      Ideal     D     SI1   60.8   57.0   2757  5.75  5.76  3.5
## 53936     0.72       Good     D     SI1   63.1   55.0   2757  5.69  5.75  3.6
## 53937     0.70  Very Good     D     SI1   62.8   60.0   2757  5.66  5.68  3.5
## 53938     0.86    Premium     H     SI2   61.0   58.0   2757  6.15  6.12  3.7
## 53939     0.75      Ideal     D     SI2   62.2   55.0   2757  5.83  5.87  3.6
## 
## [53940 rows x 10 columns]
```

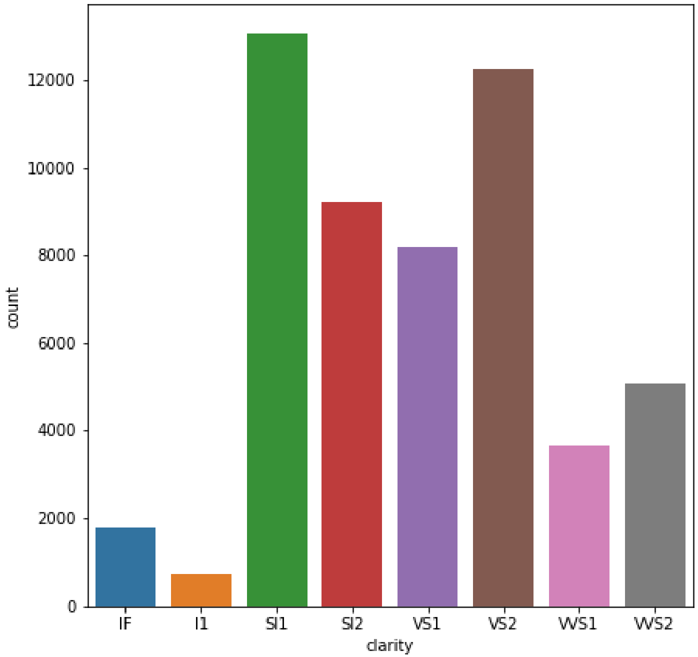
Ordinal

```python
diam = sns.load_dataset('diamonds')
sns.countplot(data=diam,x='clarity')
```



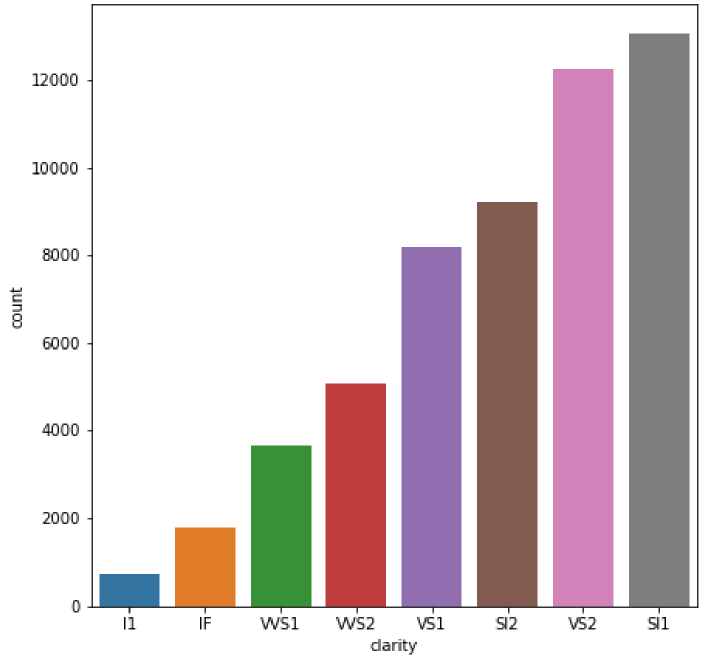
Categories ordered by levels of variable - this is fine.

Incorrect plot



Incorrect. Categories in alphabetical order.

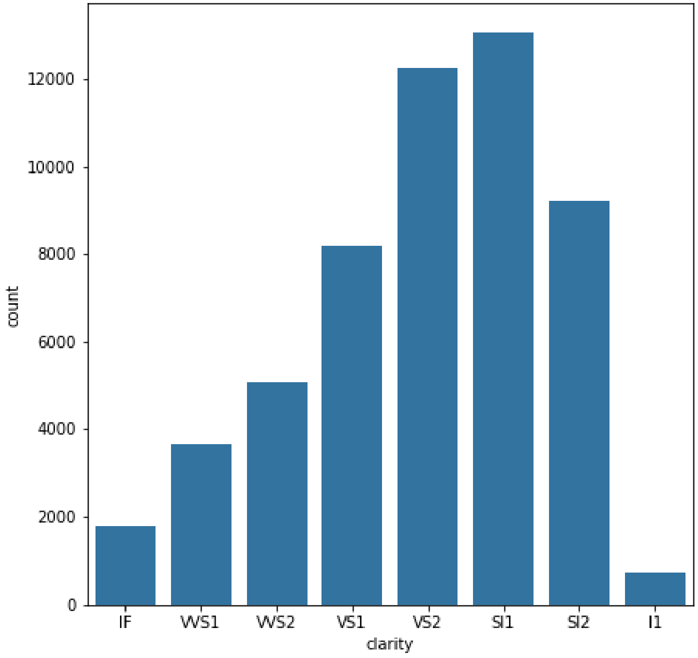
Incorrect plot



Incorrect. Ordered by frequency.

Single color

```
diam = sns.load_dataset('diamonds')
sns.countplot(data=diam,x='clarity',color='tab:blue')
```



Coloring

- Although by default categories have different colors this is not strictly necessary.
  - The default behaviour may be different for different package versions.
- Arguably it is distracting, especially when there are more categories.
- Later on we will use color to display data
  - For example grouping by a second variable and mapping that to color.
- This will be covered later on.

Lollipop charts

- If there are
  - A large number of categories,
  - If the categories all have similar frequencies,
- then consider using a lollipop chart.
- This can be done with some data munging using `value_counts` and the `stem` function in `matplotlib`.

Data preparation

For simpler graph, will only consider pickup in Bronx

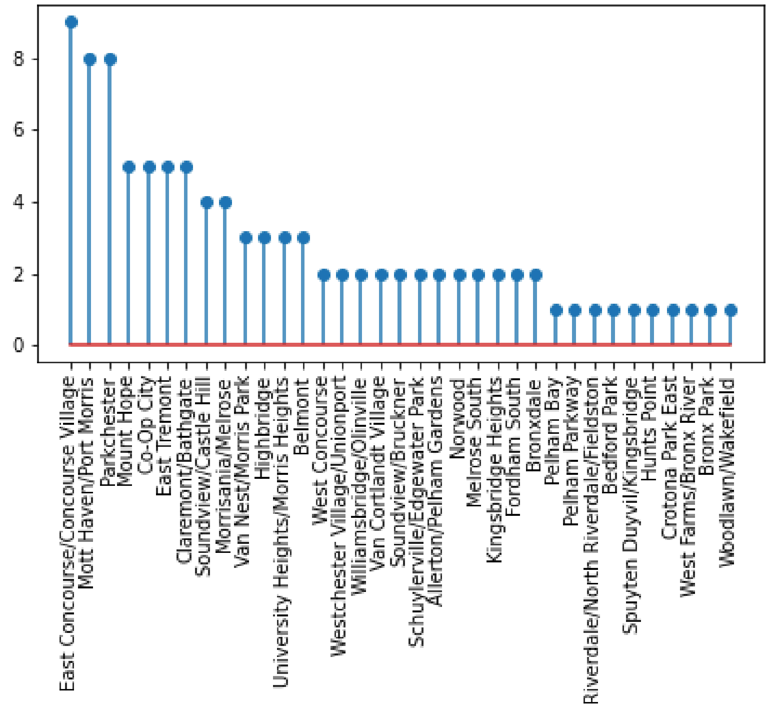
```
freq = taxisdat[taxisdat['pickup_borough']=='Bronx'].value_counts('pick
freq
```

```
## pickup_zone
## East Concourse/Concourse Village       9
## Mott Haven/Port Morris                 8
## Parkchester                            8
## Mount Hope                             5
## Co-Op City                             5
## East Tremont                           5
## Claremont/Bathgate                     5
## Soundview/Castle Hill                  4
## Morrisania/Melrose                     4
## Van Nest/Morris Park                   3
## Highbridge                             3
## University Heights/Morris Heights      3
## Belmont                                3
```

Lollipop plot (code)

```
import matplotlib.pyplot as plt
plt.stem(freq)
plt.xticks(range(0,len(freq.index)), freq.index, rotation='vertical')
plt.show()
```

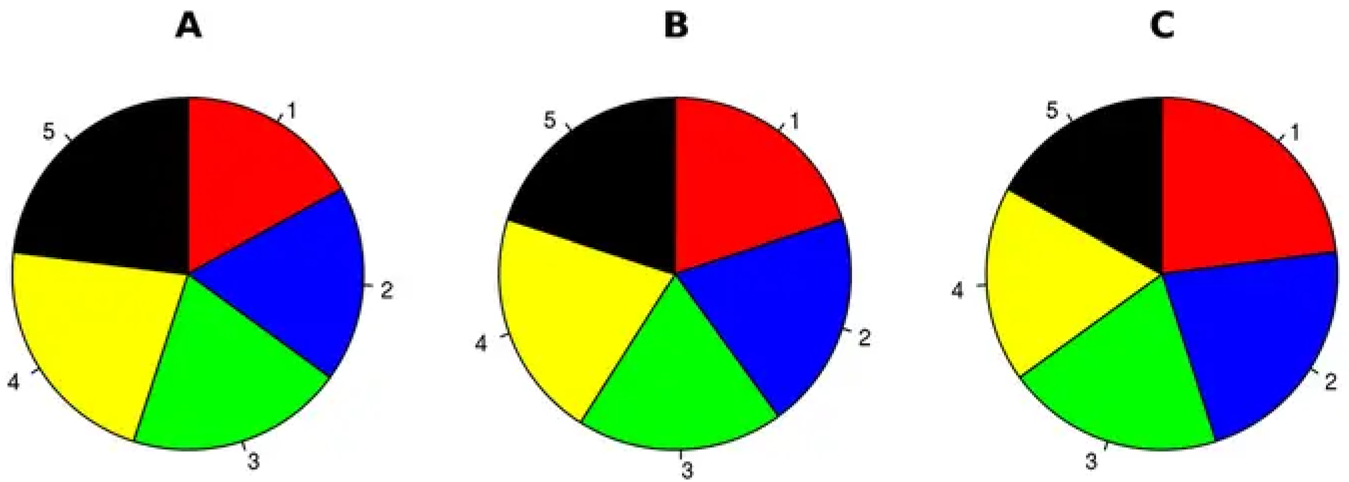
Lollipop plot (output)



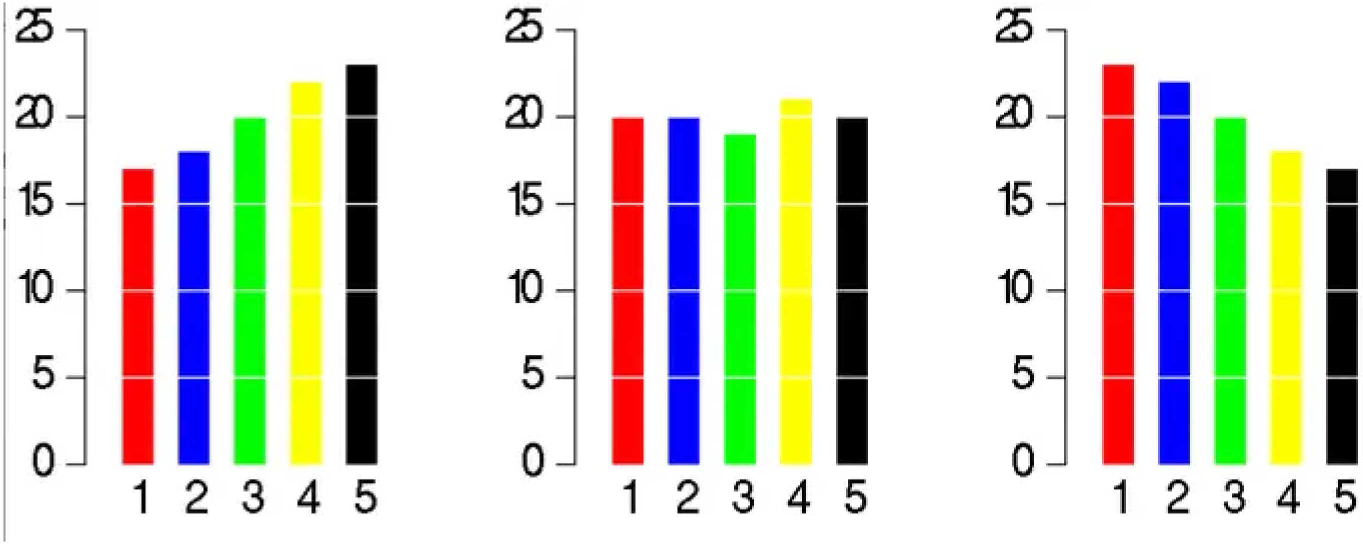
Pie charts

- Pie charts are considered to be poor practice by visualisation experts since
  - It is difficult to compare sizes of angles.
  - It is difficult to make comparisons unless categories are close.
  - They do not handle large numbers of categories.
- Following examples come from a Business Insider article by Walt Hickey.

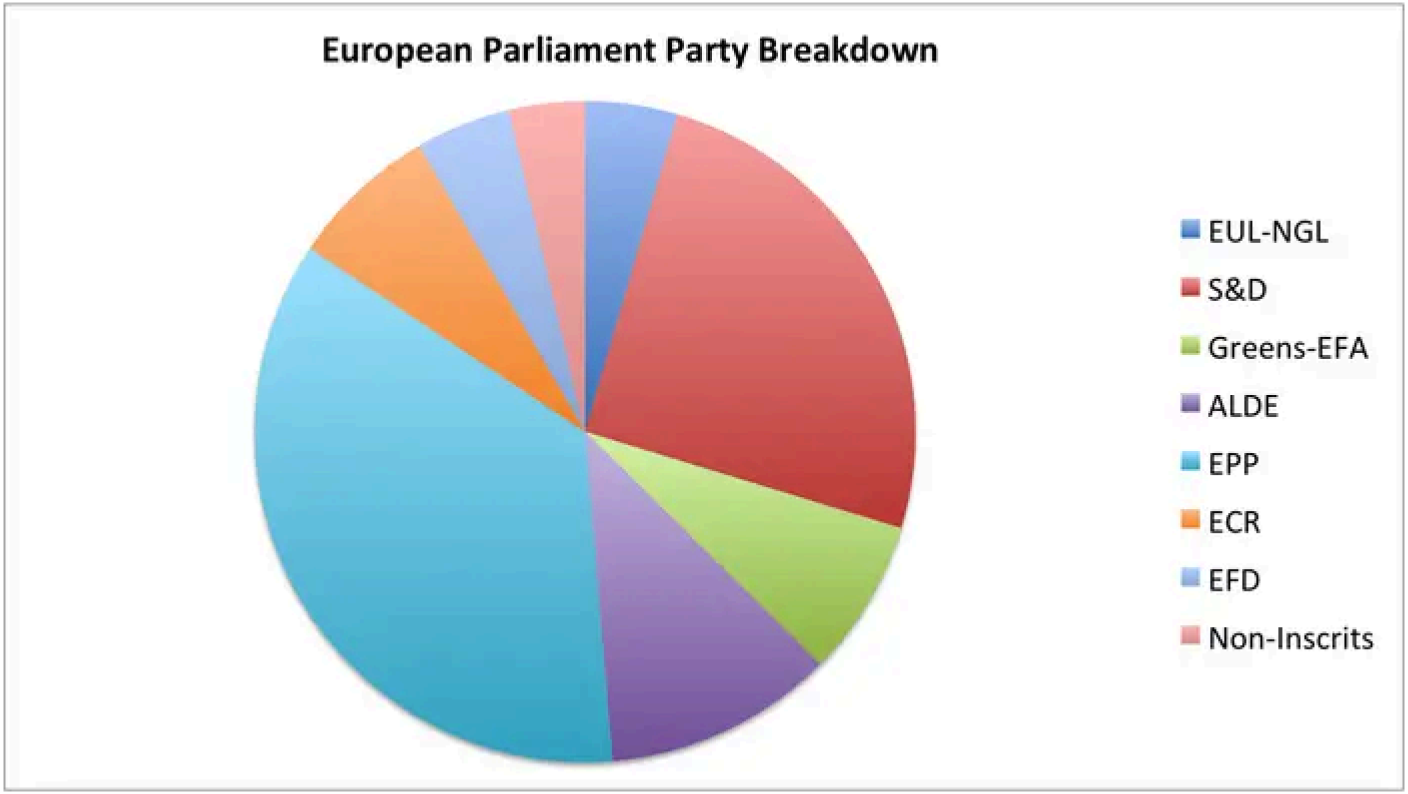
Pie chart



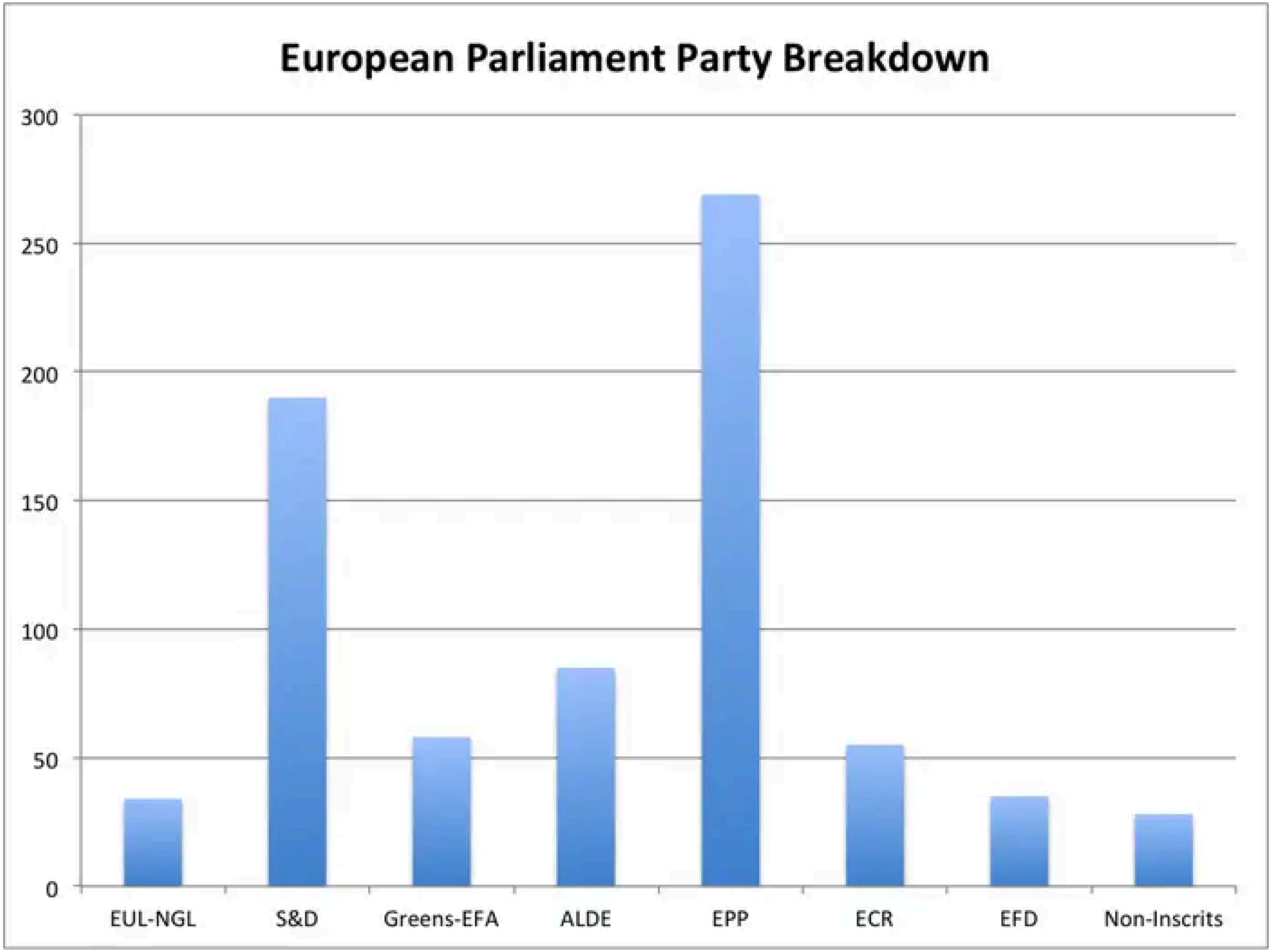
Bar chart



Pie chart

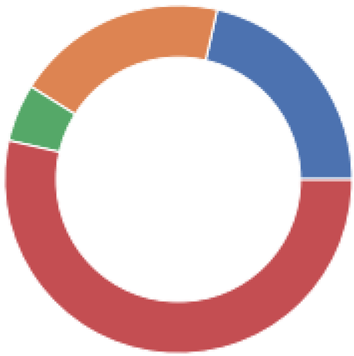

European Parliament Party Breakdown
EUL-NGL
S&D
Greens-EFA
ALDE
EPP
ECR
EFD
Non-Inscrits

Bar chart


European Parliament Party Breakdown
300
250
200
150
100
50
0
EUL-NGL
S&D
Greens-EFA
ALDE
EPP
ECR
EFD
Non-Inscrits

How to do pie charts

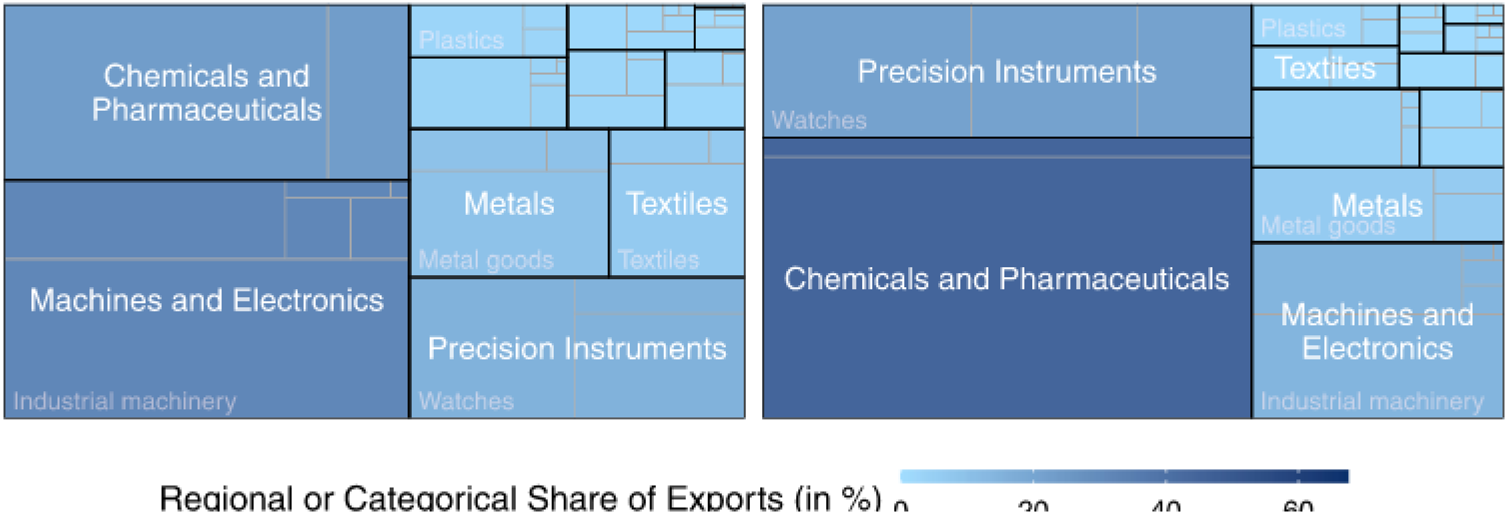
- If you absolutely MUST do a pie chart a guide can be found at this link.
- A donut chart is a pie chart with a hole. It is even worse than a pie chart.



Treemaps

- Even bar charts can struggle when the number of categories is truly huge.
- One way to handle this is using a *treemap*.
- See this example
- These are particularly well suited when categories follow a hierarchy.
- The following example considers Swiss exports that are classified into 12 categories and 48 subcategories.

Swiss Exports



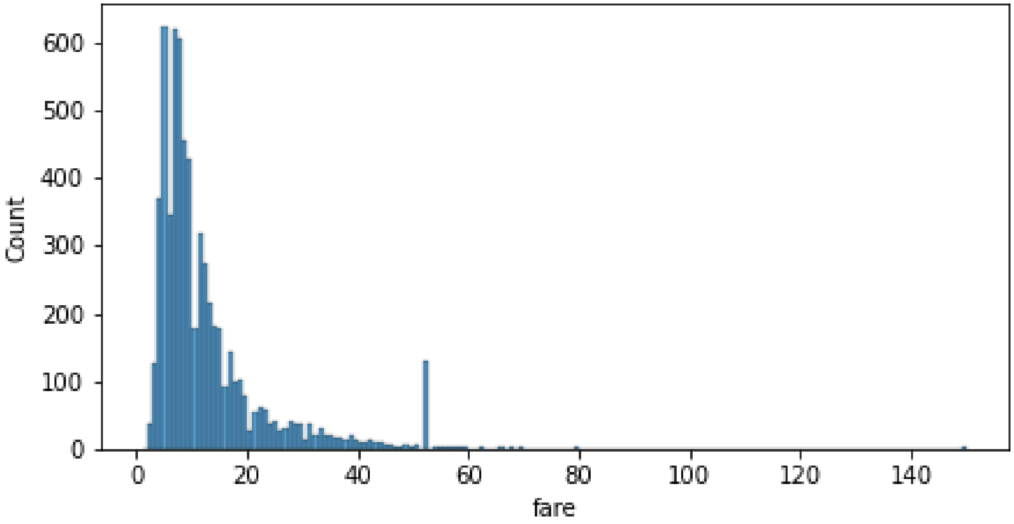
Numerical Data

Histogram

- The equivalent of a bar chart for numerical data is a histogram.
- The area of each bar represents the frequency within a certain interval.
- If all bars have equal width then frequency is mapped to the length of the bars too.
- Zero should always be included on the y axis (but not necessarily x axis).

Histogram

```
sns.histplot(taxisdat['fare'])
```

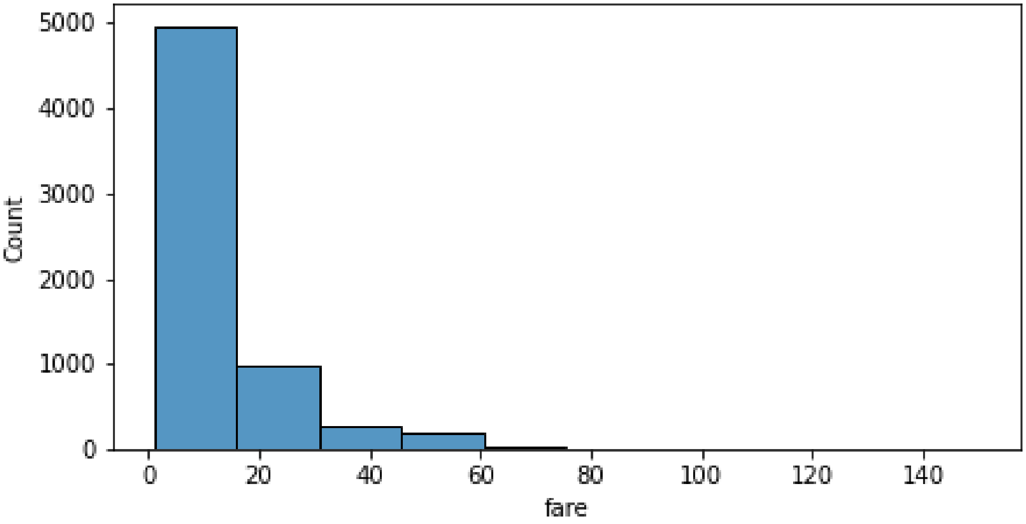


What do we see?

- Right skew
  - Should we use mean or median as measure of central tendency?
- A few big outliers.
- A spike (second 'mode') at around $50
  - Could represent a fixed fee (e.g. from airport).

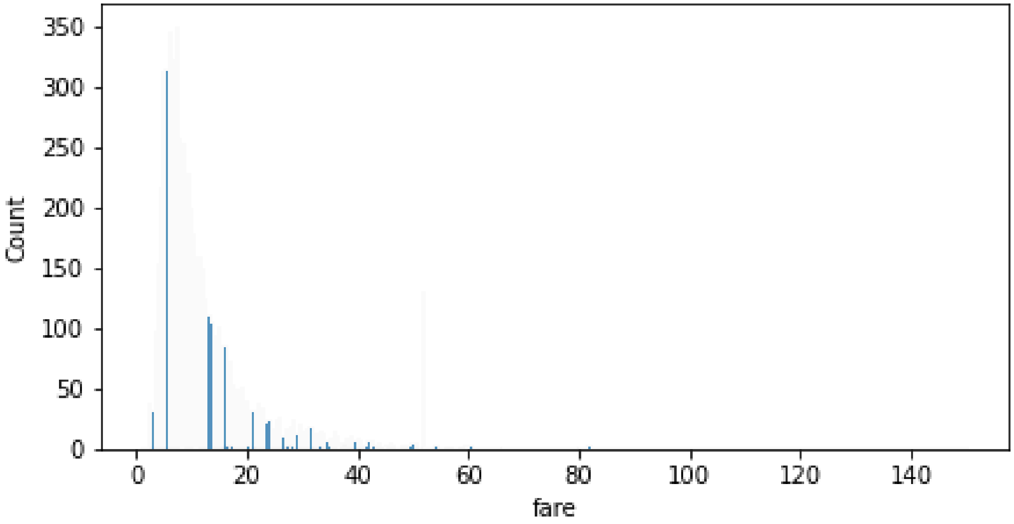
Change number of bins

```
sns.histplot(taxisdat['fare'], bins=10)
```



Change number of bins

```
sns.histplot(taxisdat['fare'], bins=2000)
```



Lessons

- By having too many (or too few) bins we can miss out on important features of data.
- In the above example the spike of fares around $50 is not seen when the number of bins is changed.
- In general default choice of bin number is good, however it is always a good idea to experiment.

Kernel density estimate (KDE)

- A kernel density estimates the *probability density function (pdf)* of data.
- For data $x_1, x_2, \ldots, x_n$ the KDE is given by

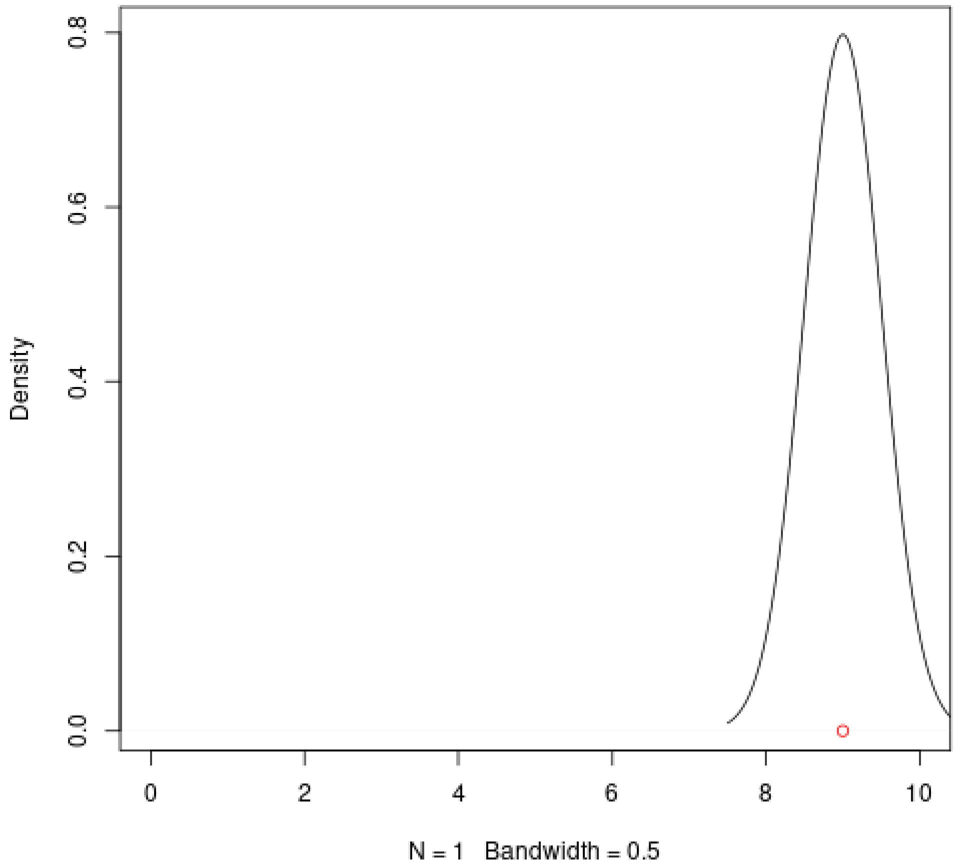
$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} K_h(x - x_i)$$

- The function $K_h(.)$ is called the *kernel*.
- Can take many forms.
- The function depends on a bandwidth h (to be explained soon).

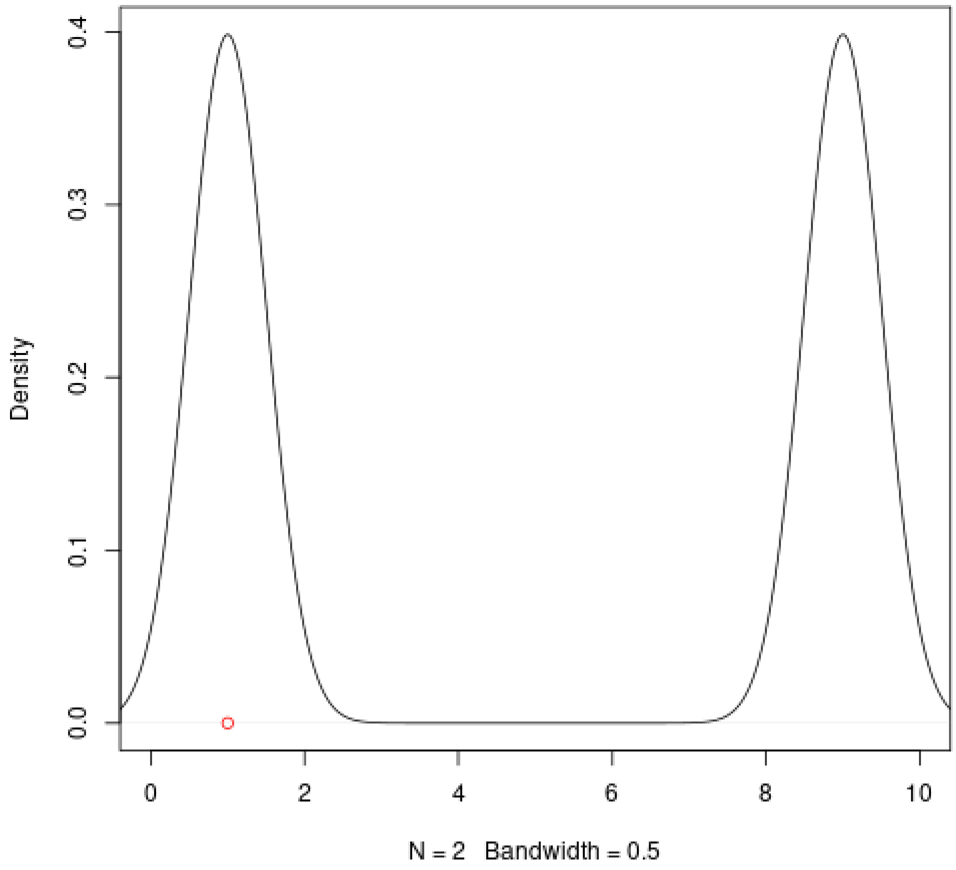
Intuition behind KDE

- If I observe a point in some location, that evidence supports that there is probability that a point comes from a nearby region.
- Imagine I drop a mountain of sand at the location I observe the data point.
- The shape of the sand is the kernel function.
- If I repeat this for n observations the result is the KDE.

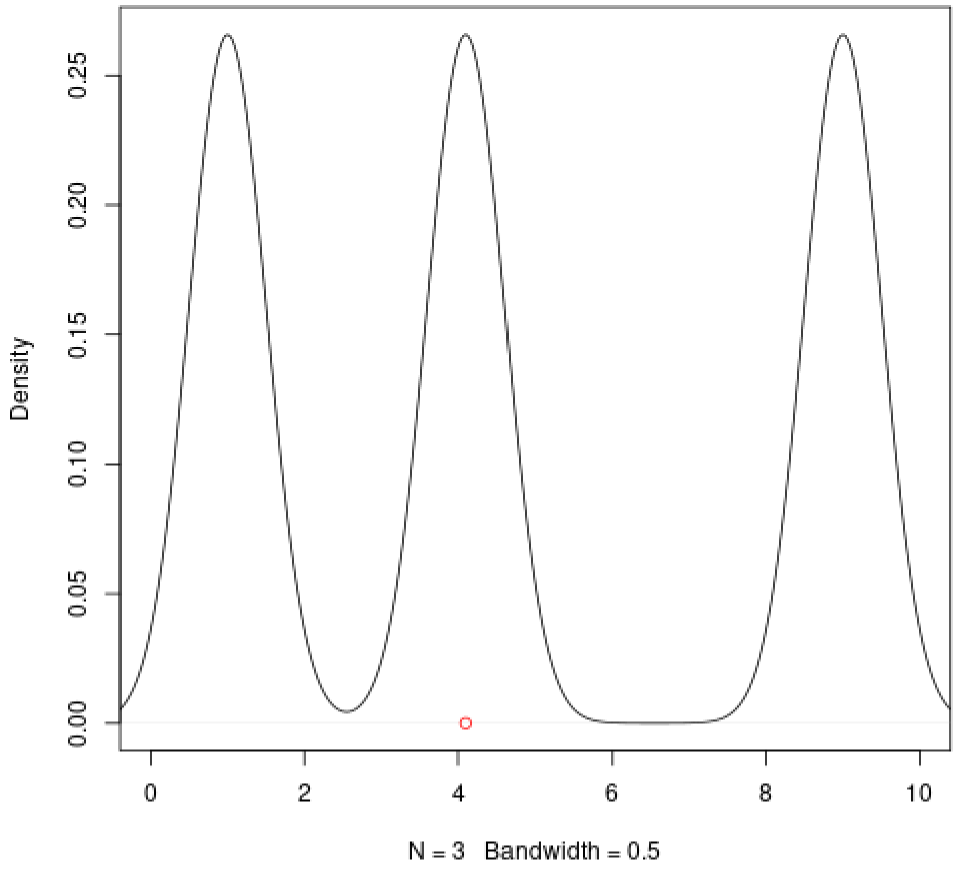
KDE (n-1)



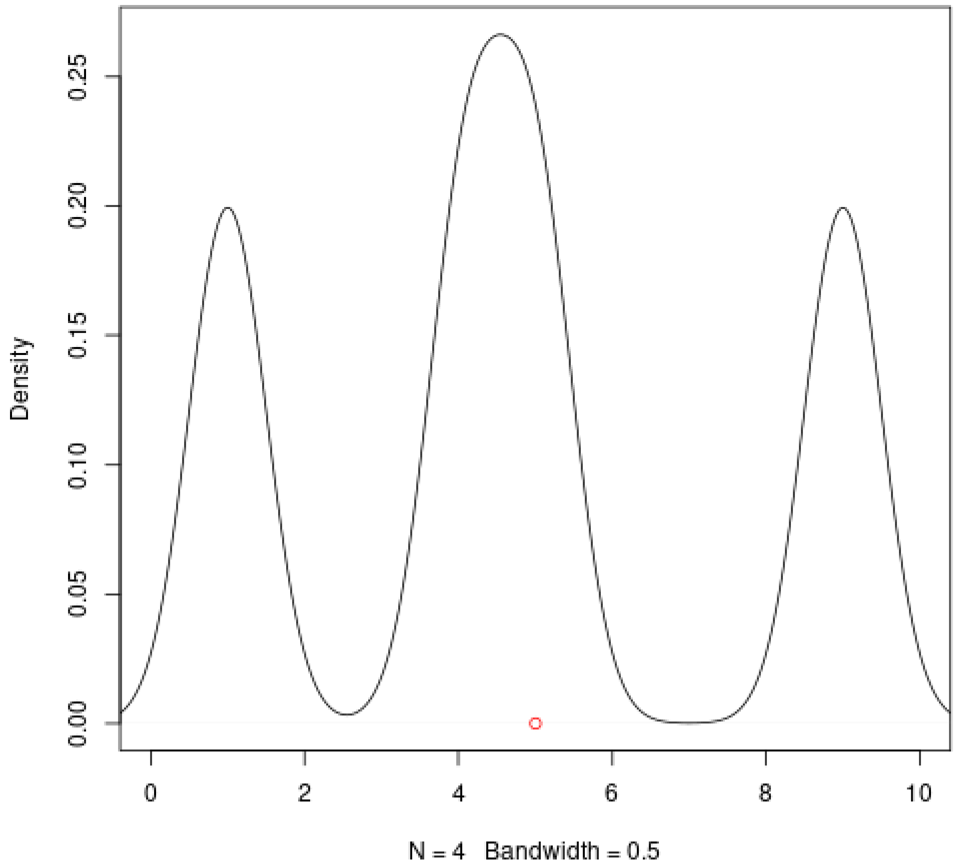
KDE (n=2)



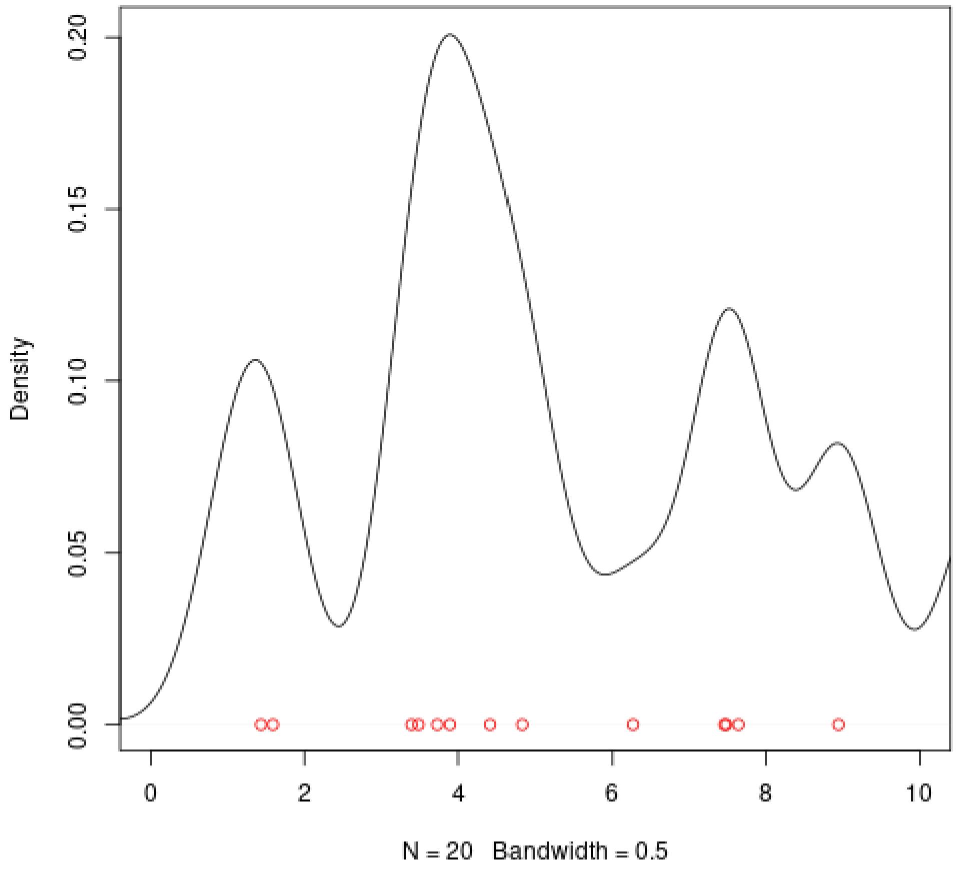
KDE (n=3)



KDE (n=4)



KDE (n=20)

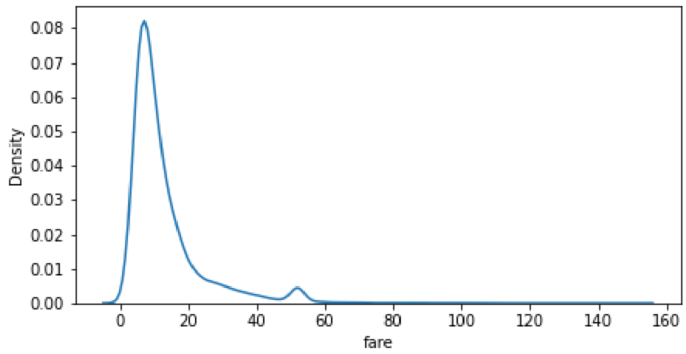


The bandwidth

- The bandwidth h controls whether the mountain of sand is 'peaked' or 'flat' .
- For small bandwidth the mountain of sand is more peaked and the KDE is more wiggly.
- For large bandwidth the mountain of sand is more flat and the KDE is more smooth.
- This is similar to the role of the number of bins in the histogram.
- There are sensible defaults used by visualisation packages.

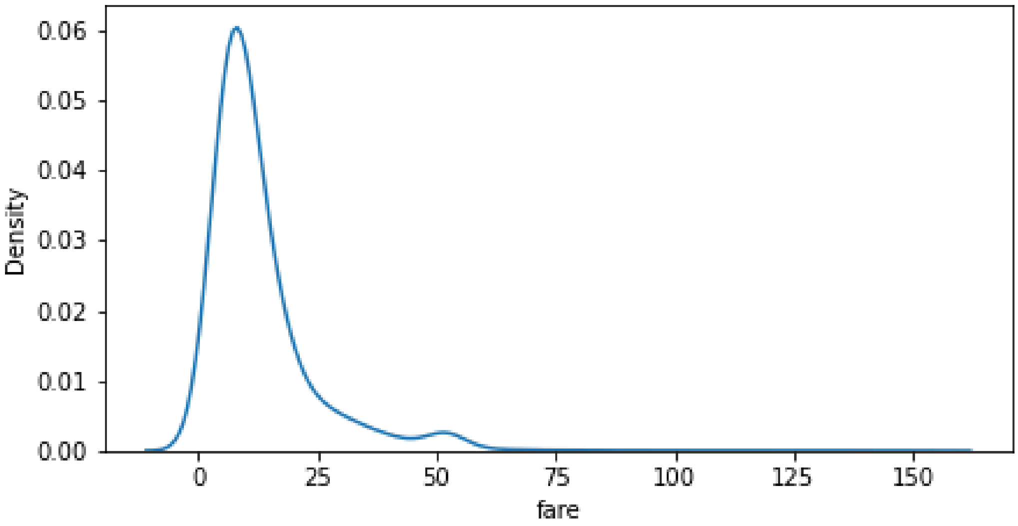
KDE plot

```
sns.kdeplot(taxisdat['fare'])
```



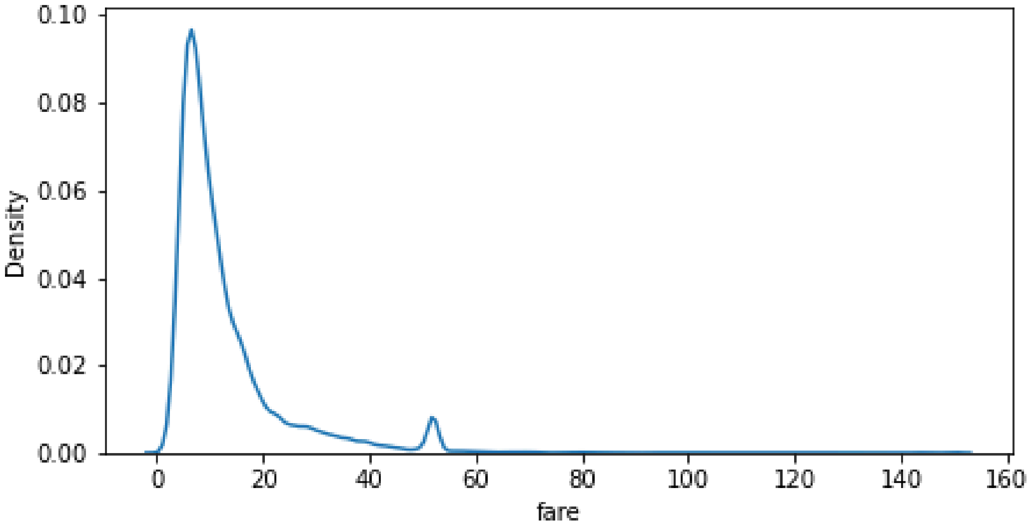
KDE plot (double default BW)

```
sns.kdeplot(taxisdat['fare'], bw_adjust = 2)
```



KDE plot (half default BW)

```
sns.kdeplot(taxisdat['fare'], bw_adjust = 0.5)
```

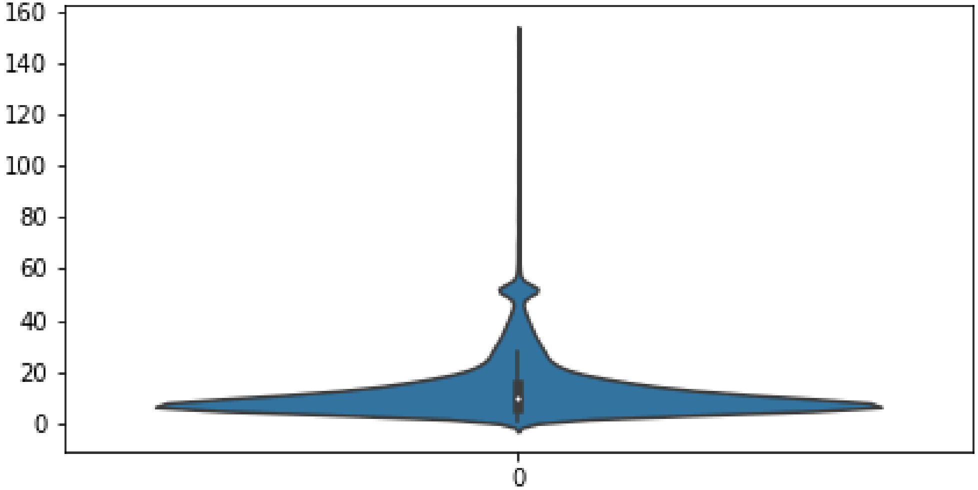


Violin plot

- A violin plot mirrors a KDE and fills it in.
- It is particularly useful for making comparisons of density according to a grouping variable.
- We will cover this next week.

Violin plot

```python
sns.violinplot(taxisdat['fare'])
```

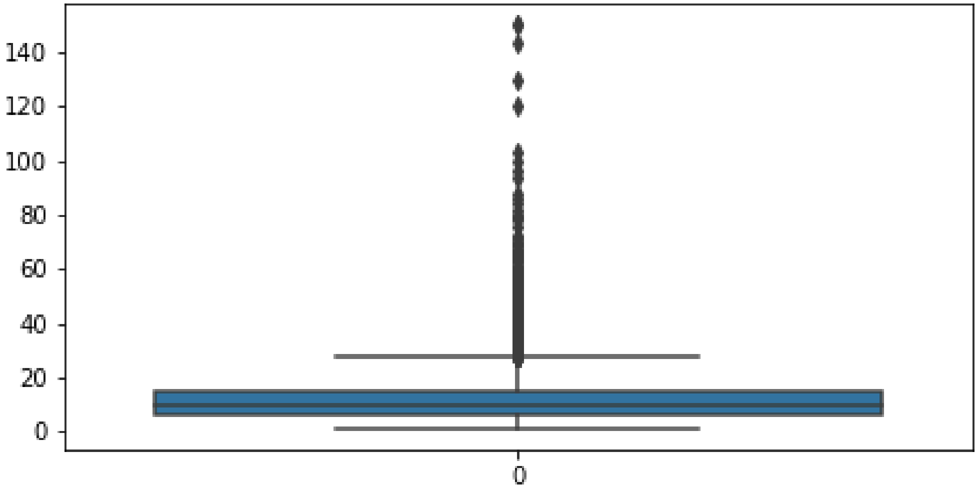


Boxplot

- Inside the violin plot is a boxplot.
- The boxplot is a summary of five statistics
  - Median
  - First Quartile
  - Third Quartile
  - Minimum
  - Maximum

Box plot

```
sns.boxplot(taxisdat['fare'])
```



Fences

- For most implementations, a boxplot actually shows an upper and lower fence rather than the maximum and minimum.
- The upper (lower) fence is given by the third (first) quartile plus (minus) 1.5 times the IQR.
- The maximum (minimum) is shown instead if it is less (greater) than the upper (lower) fence.

Boxplots v KDE

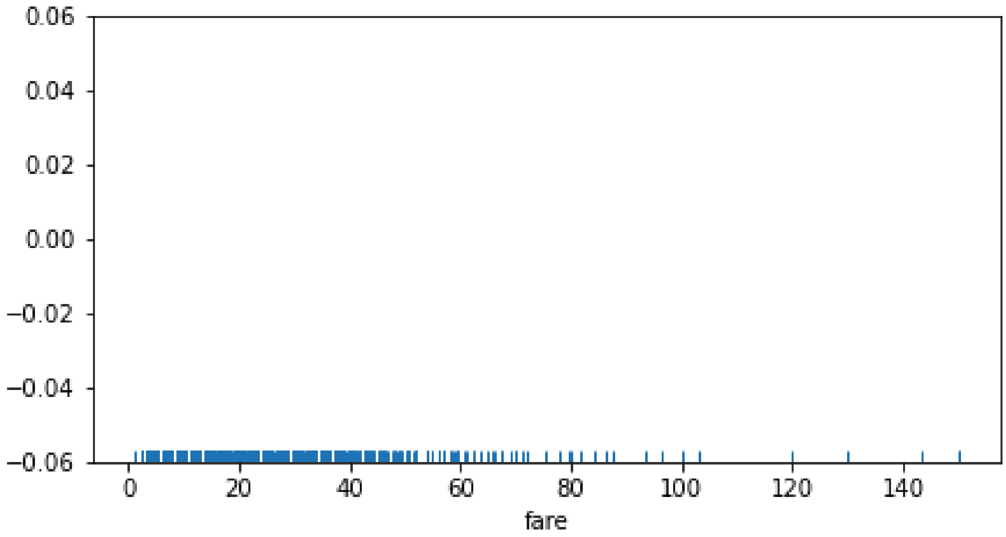
- Note that in this example the spike at around $50 is lost in the boxplot.
- However it is clearer that there are four outliers above $110.
- There is no right and wrong answer, it all depends on what you are trying to visualise.

Rug plot

- The final plot we will consider is a rug plot.
- The rug plot can highlight outliers.
- It is harder to understand the shape of the distribution using a rug plot, especially for large sample sizes.
- As a univariate plot, a jittered rug plot (strip plot) works better.

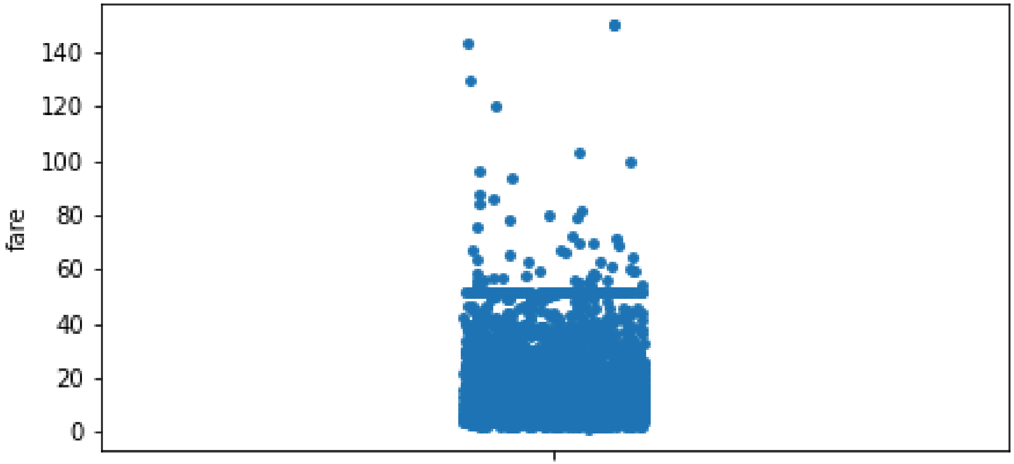
Rug plot

```
sns.rugplot(taxisdat['fare'])
```



Rug plot (jittered)

```
sns.stripplot(y=taxisdat['fare'])
```



Wrap-up

Conclusions

- Univariate plots are useful for
  - Understanding distribution of a variable
  - Finding outliers
  - Finding frequent values
  - Seeing whether data are skewed.
- Always remember that univariate plots generate questions. To answer these questions requires domain knowledge and further analysis.

Questions