



2019

# Data Science and AI

Module 3  
Part 1:

---

## Exploratory Data Analysis (EDA)

---



# Agenda: Module 3 Part 1

- Introduction to EDA
- Data cleaning & profiling
- Assessing data quality
- Data rejection & imputation
- Exploring & visualising continuous data
- Exploring & visualising categorical data
- Temporal data
- Geographic data



# Python Fundamentals

- Where does data come from?
- What does data look like?
- What is Exploratory Data Analysis?
- Where does EDA fit in the data science pipeline?

# Where does data come from?

- databases
  - data marts
  - data warehouses
- transaction systems
  - cloud
  - mainframes
- distributed file systems
  - Hadoop
- APIs
- scanned documents
- websites
  - downloads of datasets, posts, conversations, etc.
  - web scrapers
- subscribed feeds
  - news
  - IoT devices
- multimedia hosts
  - images
  - video
  - audio
- ?

# What does data look like?

- database tables
- reports & extracts
- spreadsheets & workbooks
- structured & semi-structured files
- streams
- encoded files
- bitmaps
- ?

# What is Exploratory Data Analysis?

*everything we do with a candidate dataset ...*

- after it has been rendered essentially ***usable***
- before we start developing analytics and models that address our original problem
- to determine whether it will make a useful ***proxy*** for understanding the phenomenon we are interested in.

*where does it fit?*

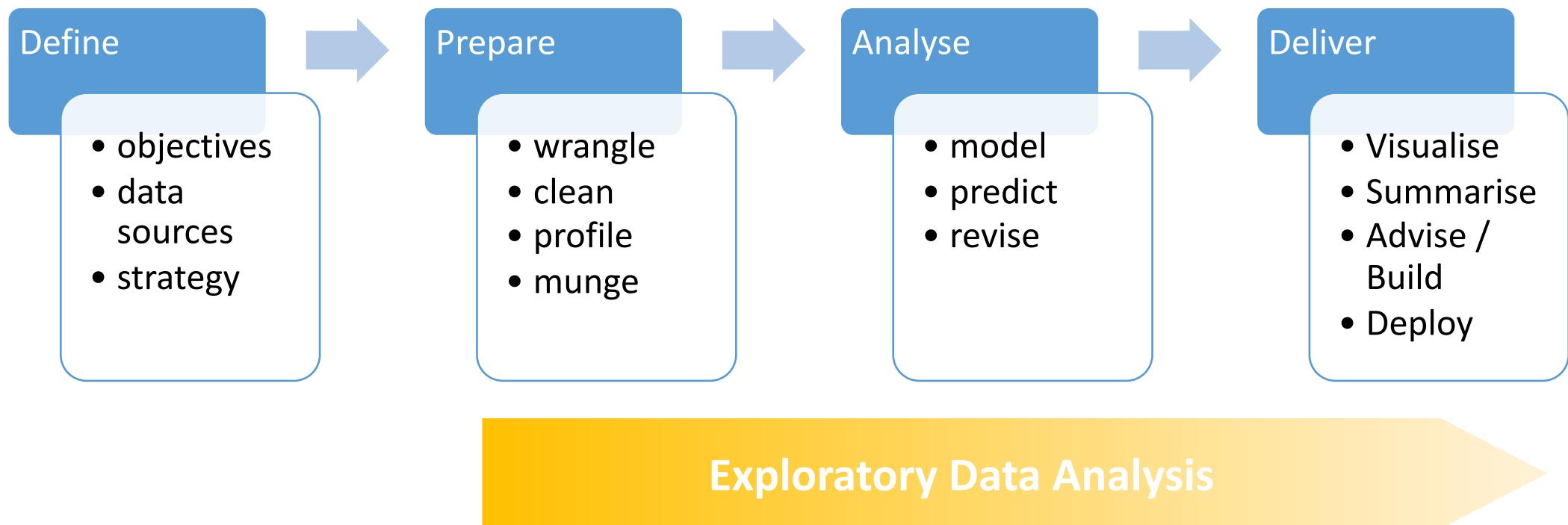
- (within the data science pipeline)

# How do we make a dataset “usable”?

- *wrangling*
  - sourcing, loading, and precleaning the data so we can see what it really looks like
  - fixing critical issues
- *profiling and cleaning*
  - understanding the essential characteristics of the data
  - applying preliminary transformations to confer context and meaning
  - implementing strategies for missing and invalid data
- *munging*
  - reshaping the data to prepare it for analysis

# Where does EDA fit?

- (within in the data science pipeline)





# Data Cleaning & Profiling

- Preliminary data cleaning
- Basic data profiling
- Assessing data quality
- Data rejection and imputation

# Data Cleaning & Profiling

*def:* Data profiling

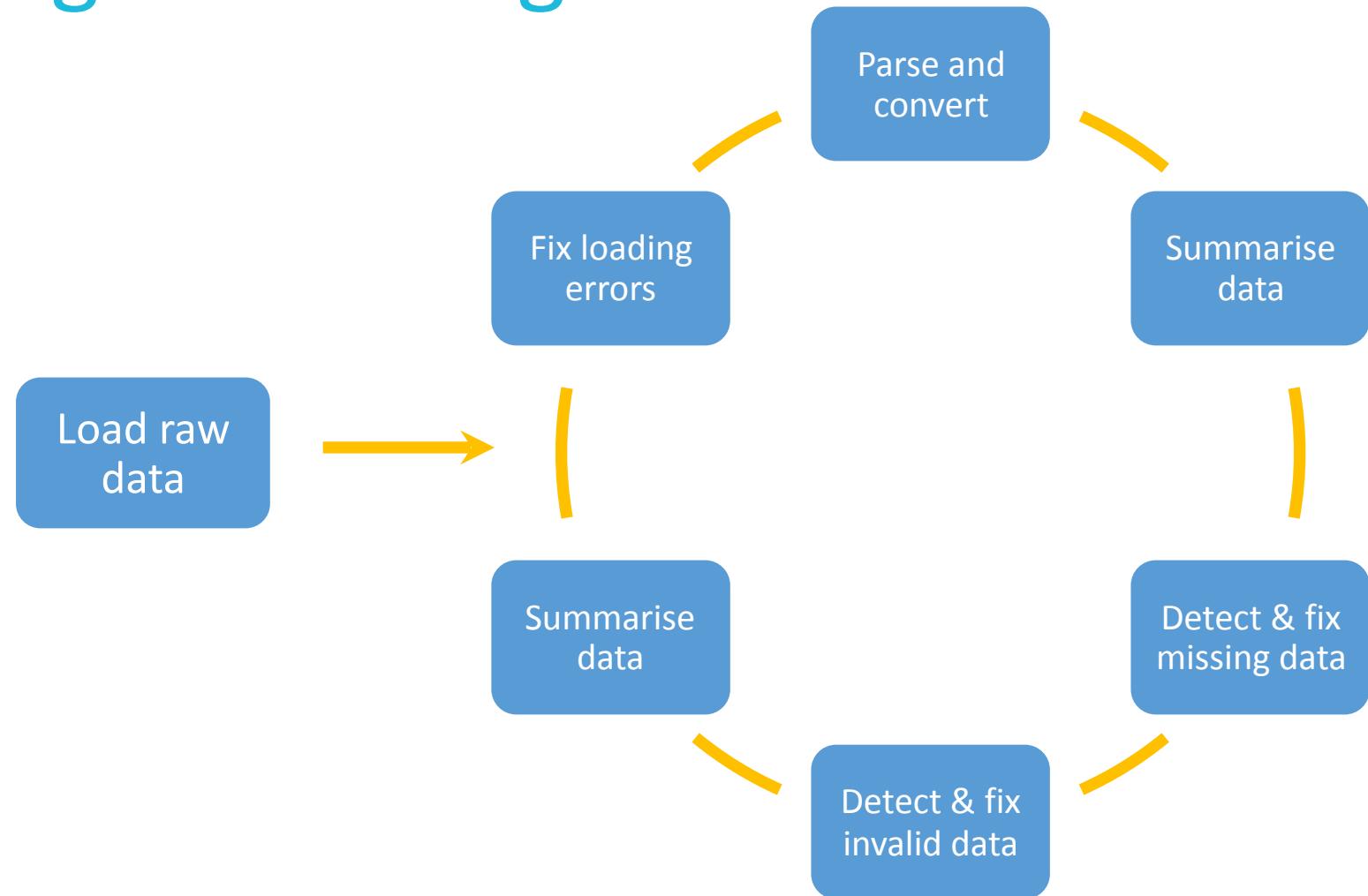
- examining the characteristics of the dataset
  - data types
  - data ranges (continuous) & categories
- identifying issues with the data

*def:* Data cleaning

- making the data usable (preparing it for analysis)
  - reformatting
  - data type conversion
  - dealing with dirty data

# Data Cleaning & Profiling

*... is iterative*



# Data Cleaning & Profiling – Details

## Load raw data

- from source system
    - database
    - HFS
    - flat file
    - spreadsheet / workbook
    - semi-structured file (JSON, XML, HTML)
    - API
    - stream (feed, IoT)
    - web scraper
    - scanned text



# Data Cleaning & Profiling – Details

## Fix loading errors

- missing delimiters
  - e.g. badly written mainframe extracts that suppress trailing commas for empty fields
- unexpected delimiters
  - e.g. ‘|’ or tab character used in “CSV” file
- illegal characters
  - e.g. ‘\u’ is normally interpreted as indicating Unicode  
may need to suppress default behaviour of function used to load the data
- missing control characters
  - EOL
  - EOF
- other?

# Data Cleaning & Profiling – Details

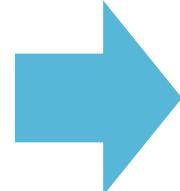
## Parse and convert

- formatted date strings to dates
  - d/m/y, m/d/y, dd/mm/yyyy, dd-mmm-yyyy, day names, month names, ...
- formatted time strings to times
  - AM/PM vs 24-hr
  - time zone conversions
- formatted date+time strings to datetimes
- string to int, string to float
- proprietary formats
  - binary, octal, hexadecimal

# Data Cleaning & Profiling – Details

What to do when data conversions fail?

- implement a *try* block
  - to catch format conversion failures
- use transformations that can handle missing values
  - or deal with missing values first
- document conversion failures
  - these are *limitations* that should be addressed when interpreting the results of analysis



```
def try_parse_int(s, base=10, val=None):  
    try:  
        return int(s, base)  
    except ValueError:  
        return val
```

# Data Cleaning & Profiling – Details

## Detect & fix missing values

- drop rows
- replace with NA
- impute values
  - mean, median, mode
    - of entire column
    - of similar data (grouped by other fields)
  - nearest neighbour
    - assign value from closest point (according to a suitable distance metric)

# Data Cleaning & Profiling – Details

Dealing with missing or bad data

- replace with NA
- impute values
  - out of range
    - too small: set to minimum possible value?
    - too large: set to maximum possible value?
- drop rows
  - impossible values (e.g. out of domain)
    - length = green: drop?
    - salary = -1: drop?
- drop columns
  - too many missing or invalid samples

# Data Cleaning & Profiling – Details

Summarise data

- counts of missing values
- counts of invalid values
- statistical parameters of distribution
  - continuous variables
    - bin frequencies
    - mean, median, maximum, minimum
  - categorical variables
    - category frequencies
      - most frequent (mode), least frequent

# Assessing Data Quality

- accuracy, reliability (veracity)
- currency, relevance (value)
- missing and invalid values
  - overall
  - by column
  - by row

*issues:*

- can we afford to throw out rows with missing data?
- how will imputation of missing/invalid data affect the outcome?

# Assessing Data Quality with Python

*let df be a Pandas DataFrame object*

- view the first few rows: `df.head(), df.head(nrows)`
- check for missing values: `df.isnull(), df.isnull().sum()`
- pairwise correlations: `df.corr()`
- (continuous) value ranges: `df.min(), df.max()`
- (discrete) value counts: `df.value_counts()`
- summary: `df.describe()`
- pandas profiling: `pandas_profiling.ProfileReport`
- pydqc: `pydqc`

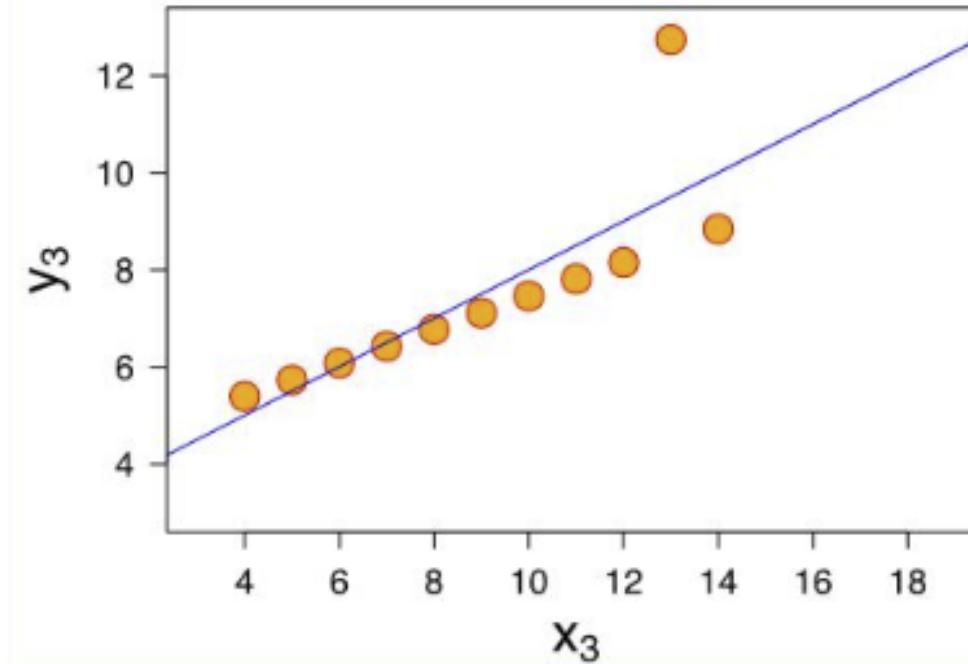
# Lab 3.1.1: Data Wrangling and Munging

- Purpose:
  - To explore Python methods for wrangling, munging, and profiling datasets
- Materials:
  - ‘Lab 3.1.1.ipynb’

# Outliers

*def:* an observation that is distant from other observations in the sample

- measurement inaccuracy
- measurement errors
  - incl. recording errors
- unusual system behaviour
- external phenomena



# Outlier Detection in 1 Dimension

## *extreme value analysis*

- outliers are defined by statistical tests based on mean & variance of sample
  - Z-test
  - Student's  $t$ -test
- mark points with low score as outliers

## *probabilistic & statistical models*

- based on assumed distribution of data
  - calculate probability that each point belongs to the distribution
  - mark points with low probability as outliers

# Outlier Detection in Multiple Dimensions

## *linear models*

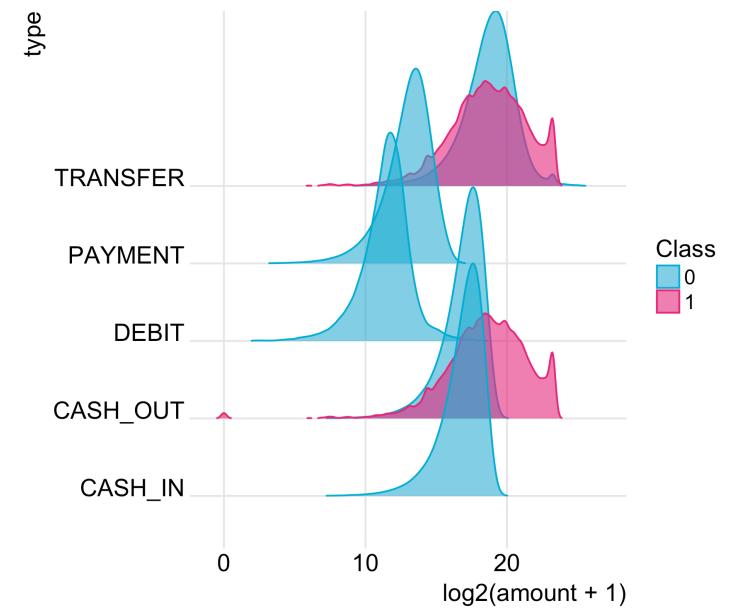
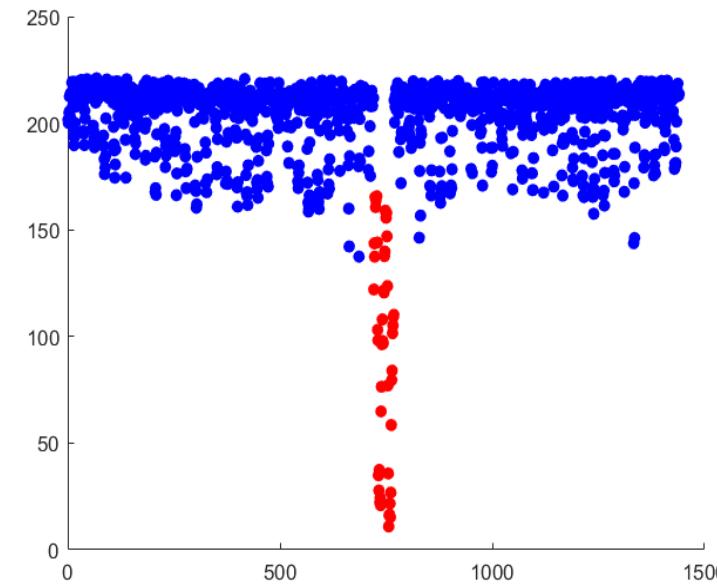
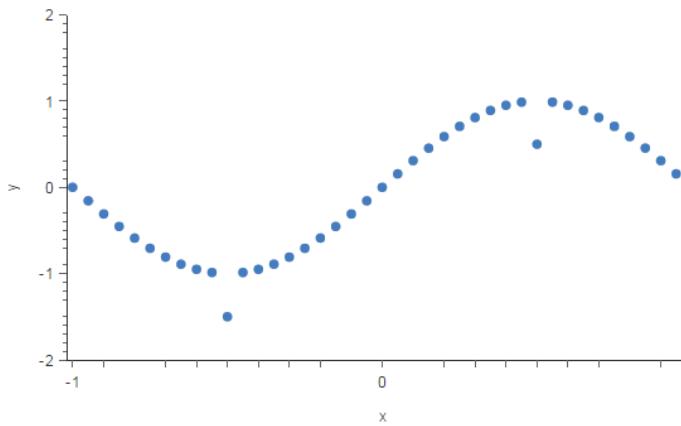
- reduce data to lower-dimensional spaces
- calculate distance from each point to a reference hyperplane
- mark points with largest distance as outliers
- similar concept to *principal component analysis* (PCA)

## *proximity-based models*

- define a distance metric and apply to each pair of points
- mark points that are more isolated as outliers
- examples: *cluster analysis*, *density-based analysis*, *nearest-neighbour analysis*

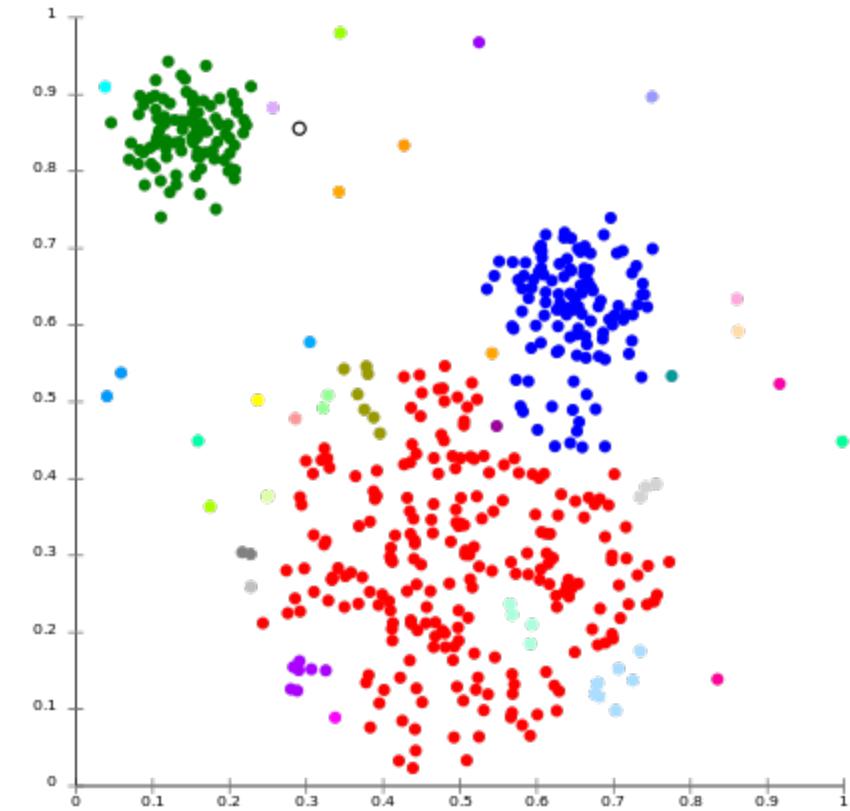
# Outlier Detection – cont'd

- outliers vs. anomalies
  - if unsure, analyse data with *and* without the outliers



# Outlier Detection – cont'd

- outliers may not be obvious in one dimension
  - some points may only get separated from the mainstream when looking at several dimensions at once
  - may indicate subsets of behaviour (“classes”)



# Discussion

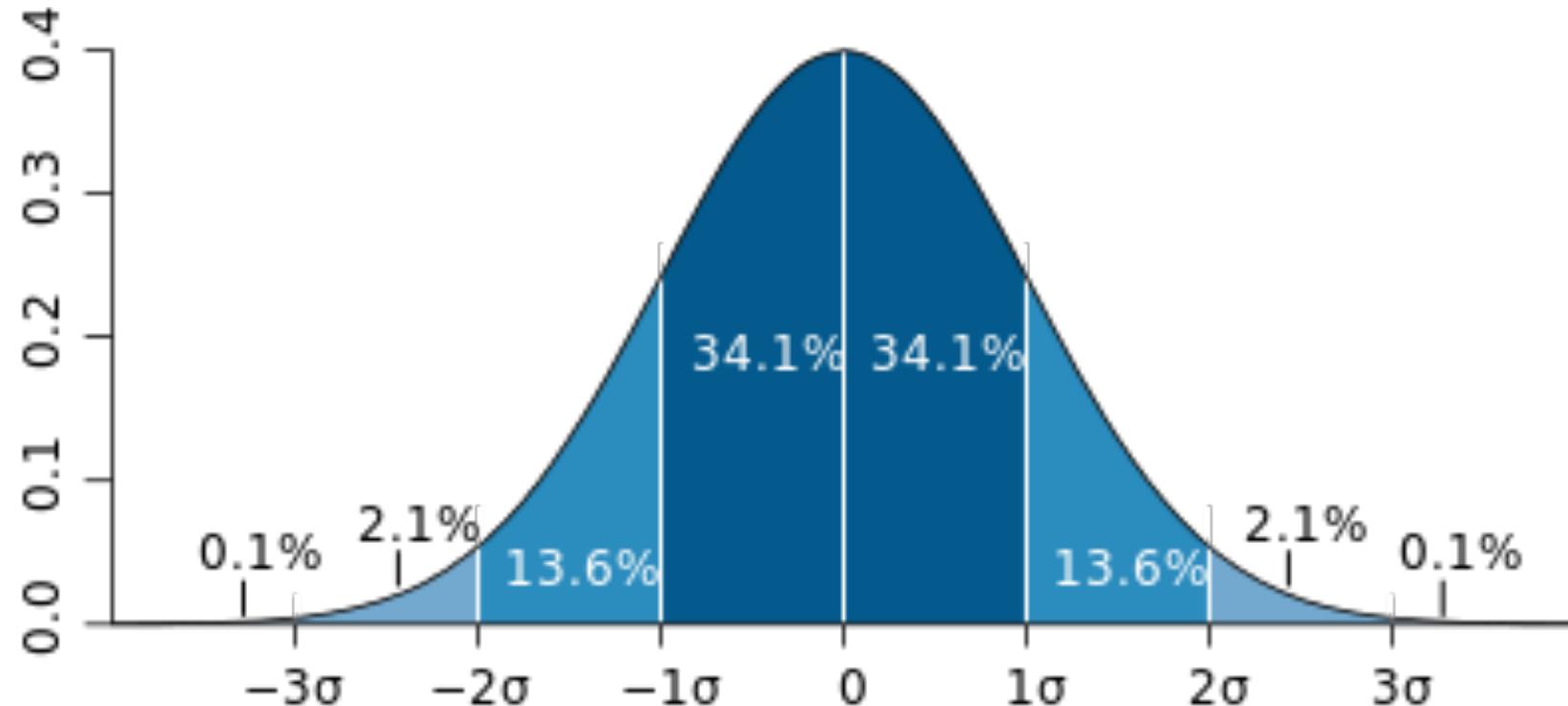
# HOMEWORK

- finish Lab 3.1.1

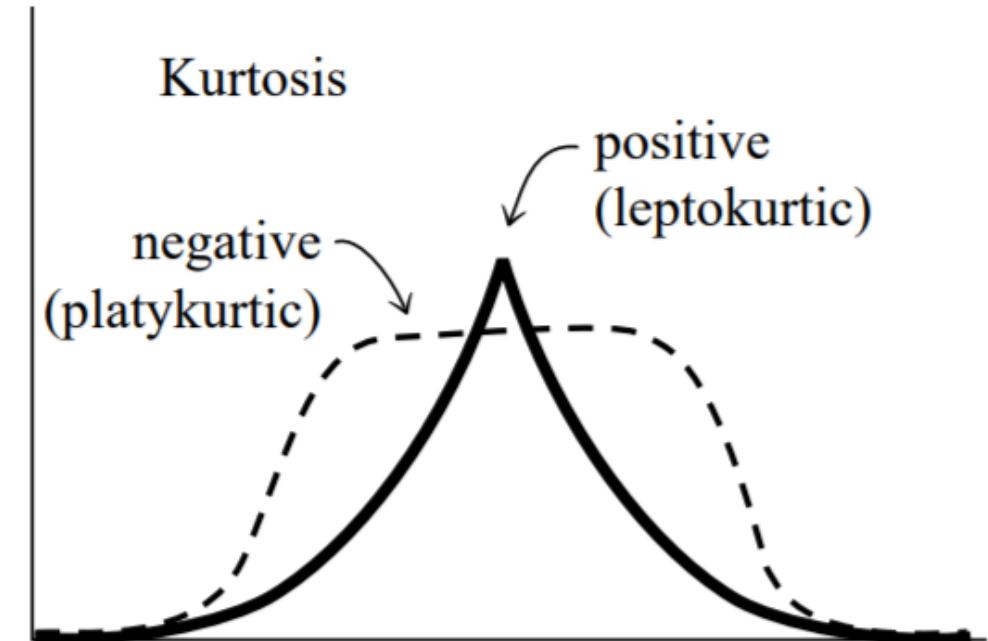
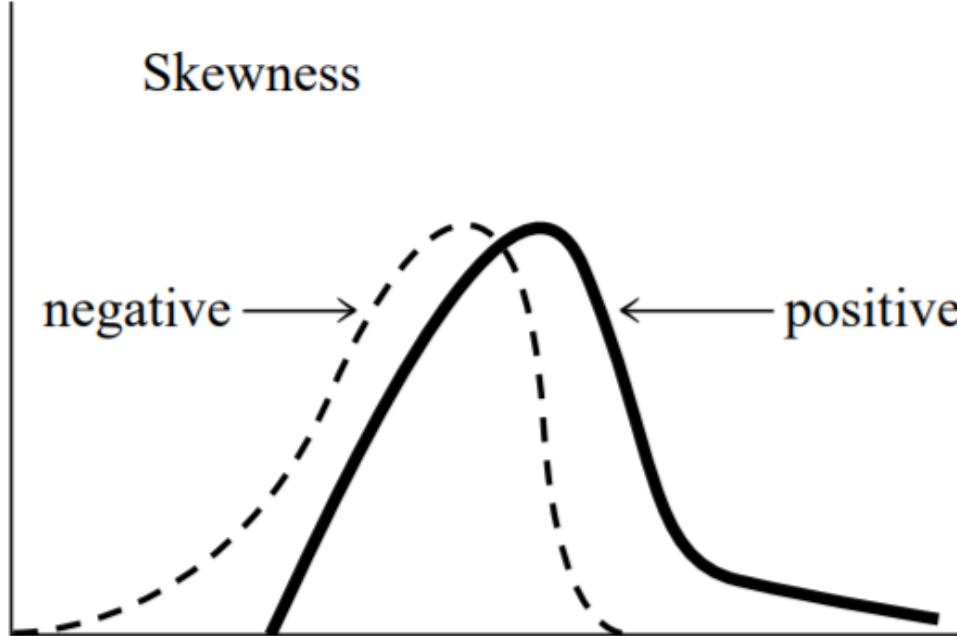
# Continuous Data

- Statistics of sample distributions
  - deeper dive: mean, variance, skewness, kurtosis
- Exploring and visualising sample variables
  - histograms
  - box & whisker plots
  - violin plots
- Outlier detection

# Mean & Variance

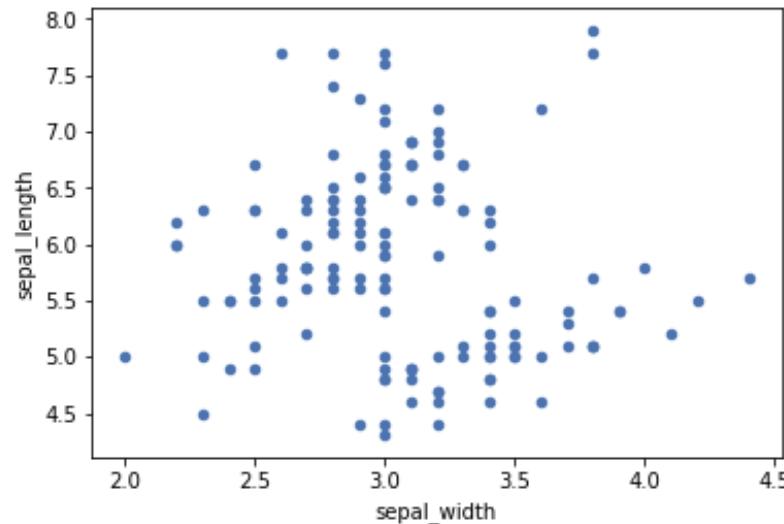


# Skewness and Kurtosis

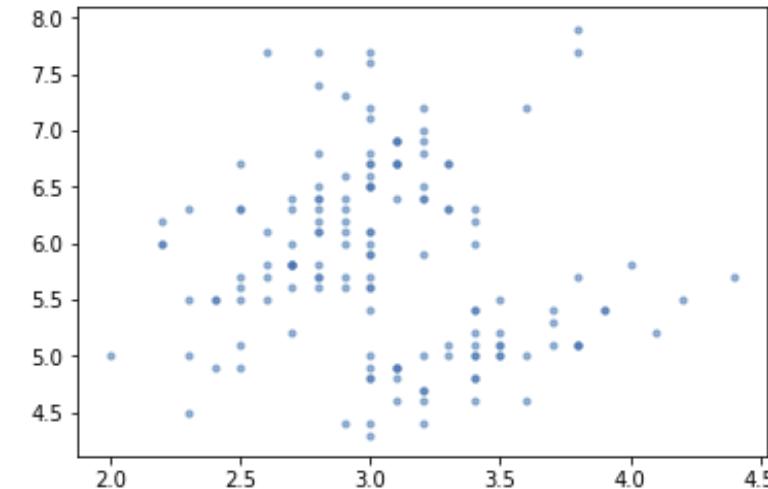


# Scatterplot

- shows a 2D relationship within the dataset by plotting one column against another



```
df.plot(kind='scatter', x='sepal_width', y='sepal_length')
```

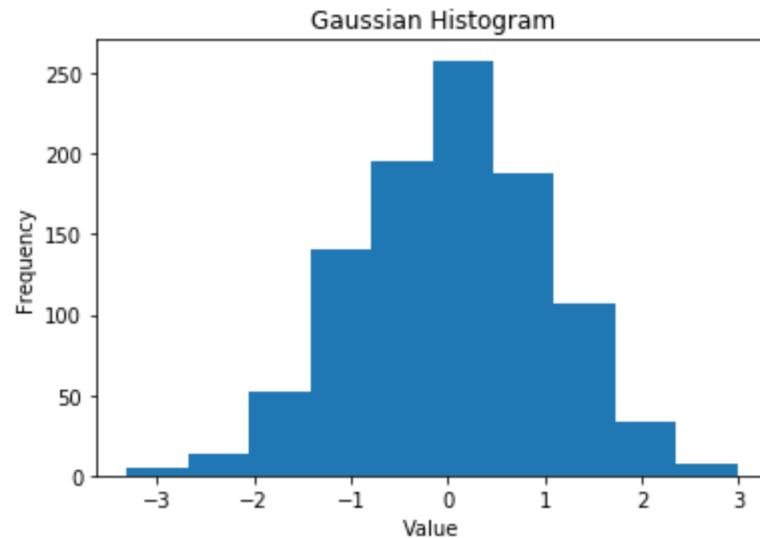


```
plt.scatter(df['sepal_width'], df['sepal_length'], s = 10,  
           linewidths = 1, alpha = 0.5)
```

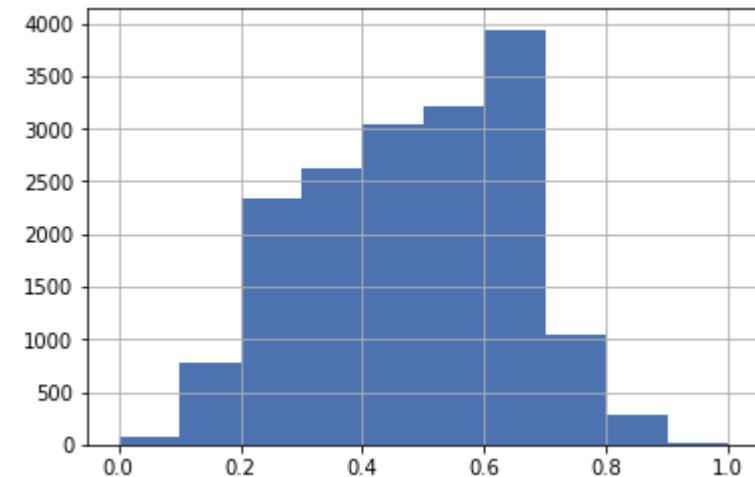
[https://matplotlib.org/api/\\_as\\_gen/matplotlib.pyplot.scatter.html](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html)

# Histogram

shows the properties of the data sample distribution with no loss of information



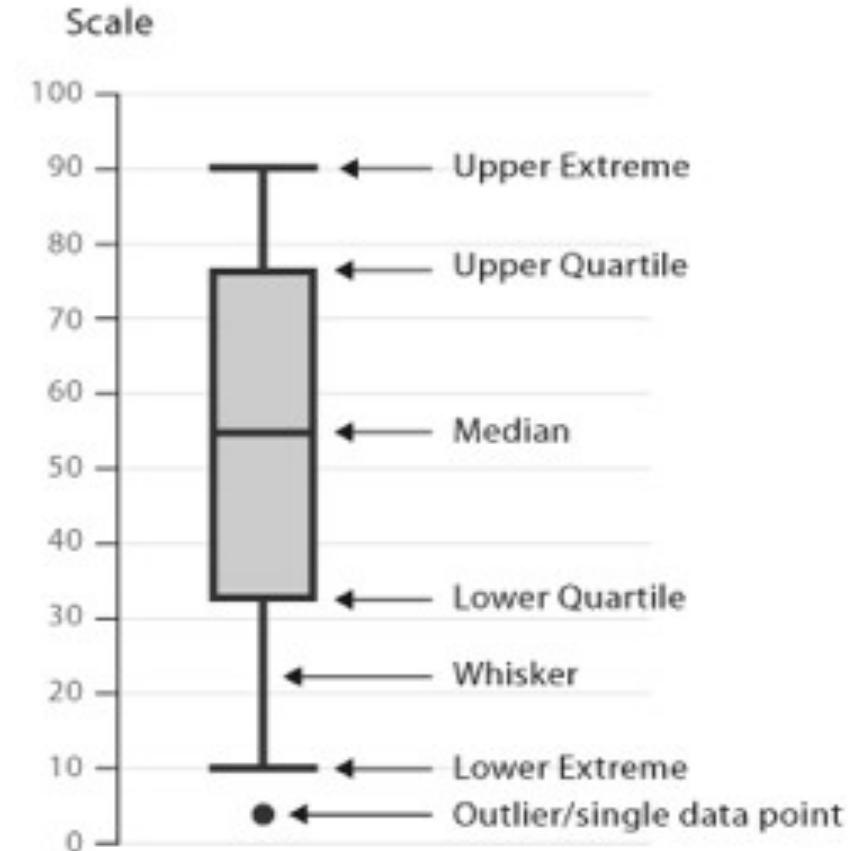
```
plt.hist(y)
plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")
```



```
df['temp'].hist()
```

# Box & Whisker Plots

- shows multiple features of sample distribution
  - median
  - interquartile range
  - 10<sup>th</sup>, 90<sup>th</sup> percentiles



# Box & Whisker Plots

```
# get 50 random numbers normally  
distributed about -1:
```

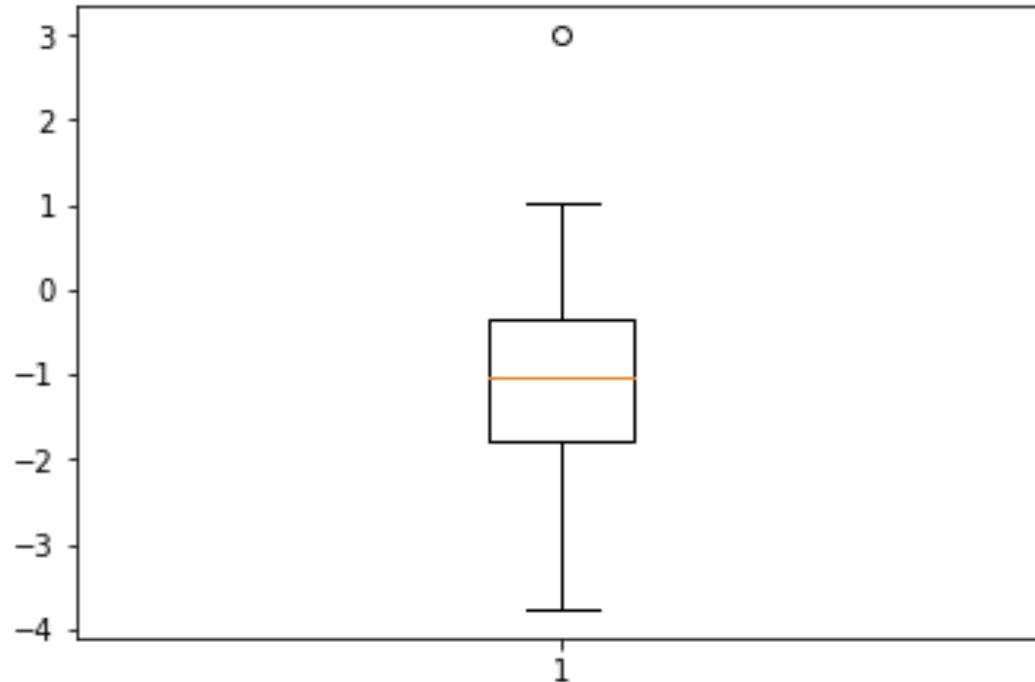
```
y = np.random.randn(50) - 1
```

```
# create an outlier:
```

```
y [49] = 3
```

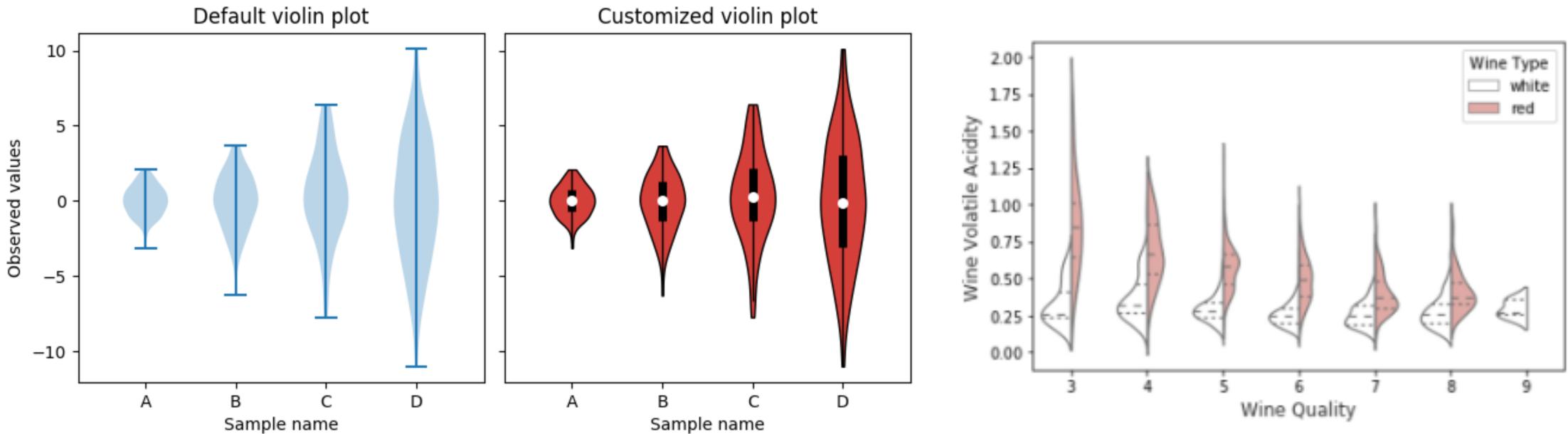
```
# plot box & whiskers:
```

```
plt.boxplot(y)
```



# Violin Plots

- shows the sample distribution itself



[https://matplotlib.org/gallery/statistics/customized\\_violin.html](https://matplotlib.org/gallery/statistics/customized_violin.html)

# Quantiles

- quantiles are popular in reporting because they help to create a sense of what is “normal”
  - *90% of calls last less than 3 minutes, 22 seconds*
  - *80% of revenue was derived from 22% of the product range*

```
1 bikes['atemp'].quantile(0.5)
```

```
0.4848
```

```
1 bikes['atemp'].quantile((0.1, 0.25, 0.5, 0.75, 0.9))
```

```
0.10    0.2424
0.25    0.3333
0.50    0.4848
0.75    0.6212
0.90    0.6970
Name: atemp, dtype: float64
```

- quantiles are cumulative
  - e.g. 80<sup>th</sup> percentile is a subset of 90<sup>th</sup> percentile

Q: what would a plot of all possible quantiles represent?

> the cumulative probability function

# Discretisation

- suppose want to look at intervals (“bins”) instead?

```
pandas.cut(x, bins, right=True, labels=None, retbins=False, precision=3,  
           include_lowest=False, duplicates='raise')
```

```
pandas.cut(df['temp'], bins = 4).head()
```

```
(0.25, 0.5]  
(0.25, 0.5]  
(0.5, 0.75]  
(0.25, 0.5]  
(0.5, 0.75]  
..
```

- continuous data can be sorted into specified bins
- **bins** can be a vector of ‘cut’ boundaries (for asymmetric bins)
- bin counts can be plotted as a bar chart (discrete version of histogram)

# Continuous $n$ -Dimensional Data

## marginal distribution

- the distribution of the entire sample of a given variable from a multivariate sample
- ignores presence of other ( $n-1$ ) covariates

## conditional distribution

- the distribution of a given variable *contingent on* values of other ( $n-1$ ) covariates
- for a pair of covariates X, Y
  - joint distribution:  $\Pr(X = x, Y = y)$
  - conditional distribution:  $\Pr(X = x \mid Y = y)$      *Y has been “marginalised out”*

# Pairwise Correlations in $n$ -Dimensional Data

computes correlation between every pair of columns in a matrix or DataFrame:

```
1 iris.corr()
```

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

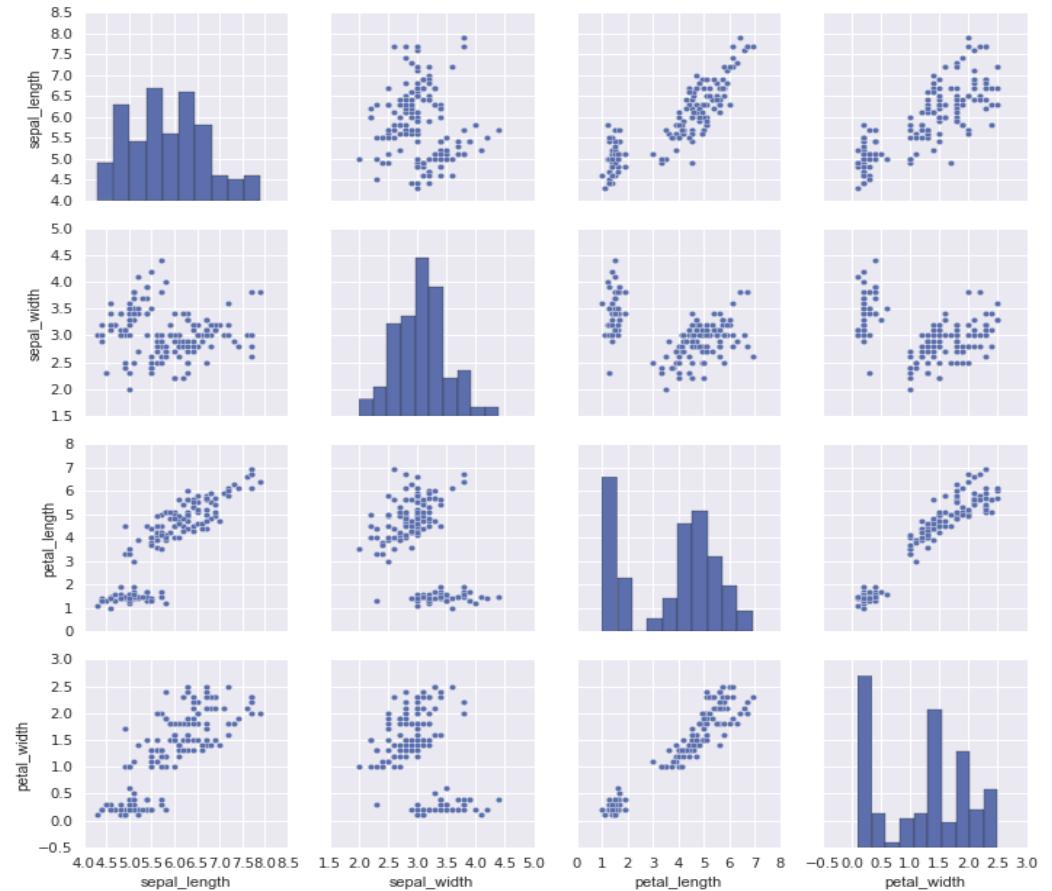
- only the figures below (or above) the main diagonal are needed
- uses Pearson's correlation by default

# Pairwise Correlations in $n$ -Dimensional Data – cont'd

can visualise correlations as a **pair plot**

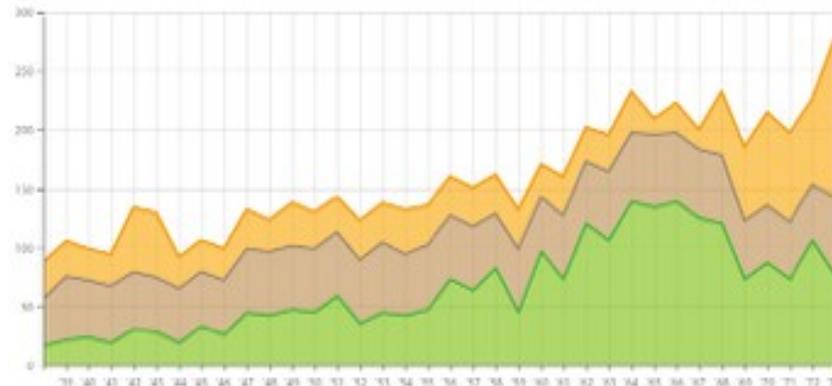
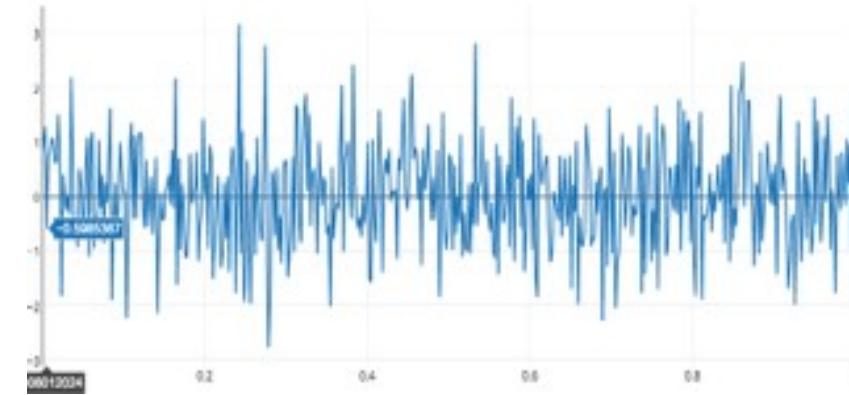
```
import seaborn as sns
```

```
sns.pairplot(iris)
```

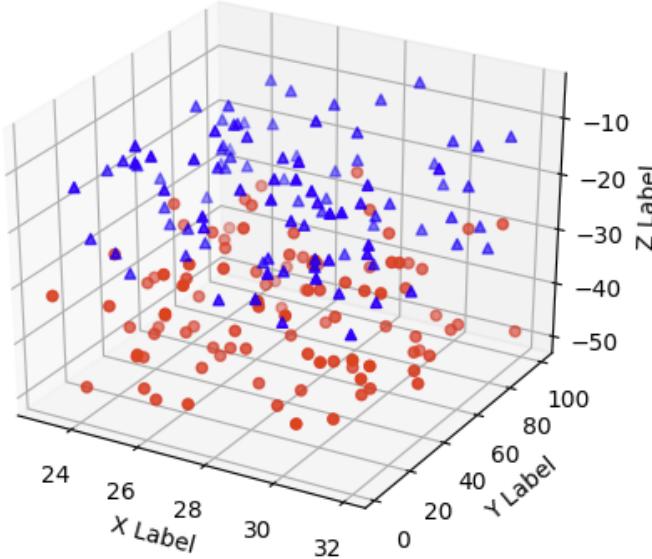


# Visualising 2-Dimensional Data

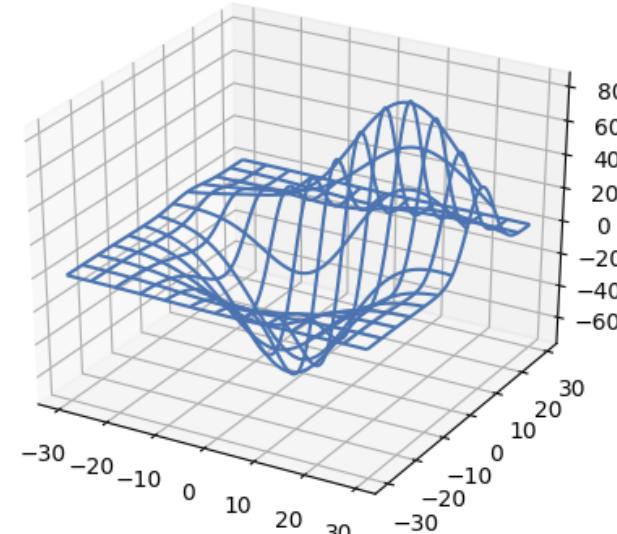
- scatterplot
- line chart
- bar chart (binned horizontal axis)
- stacked area chart
- *many variations of these*



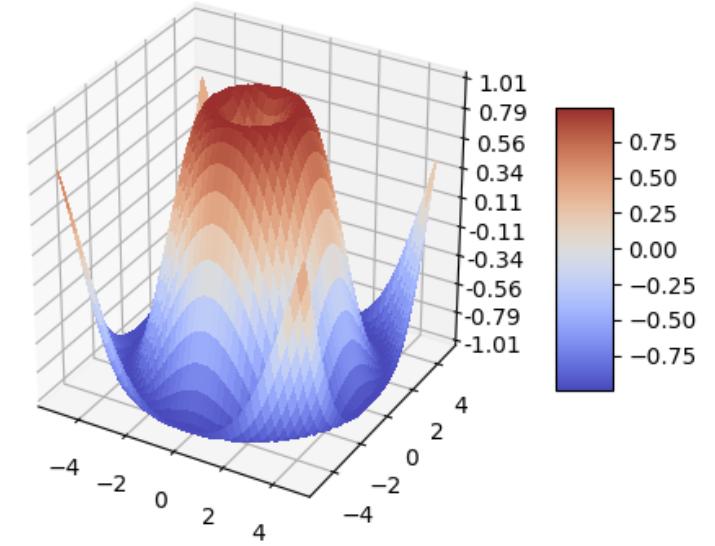
# Visualising 3-Dimensional Data



3D Scatterplot



Wireframe Plot

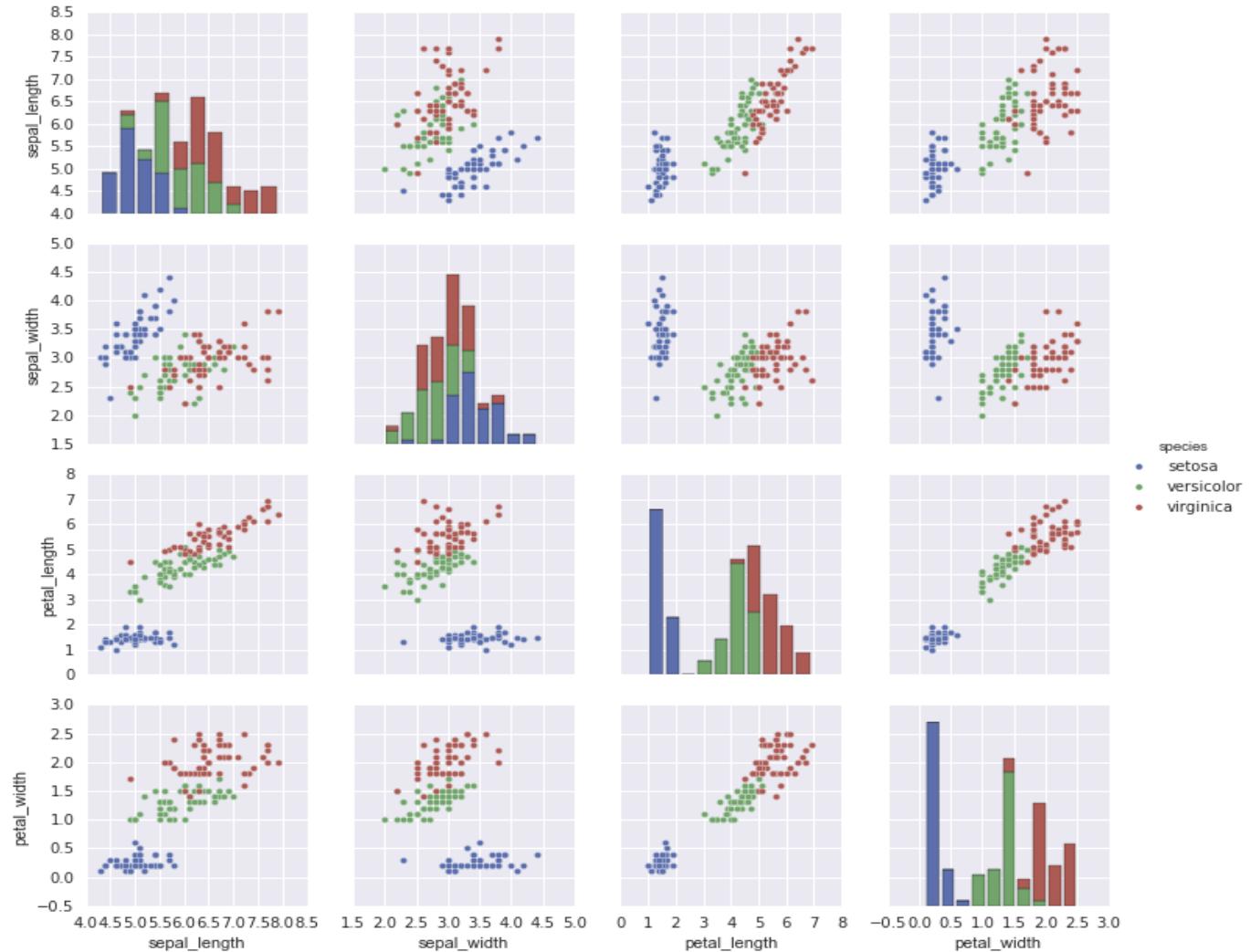


Surface Plot

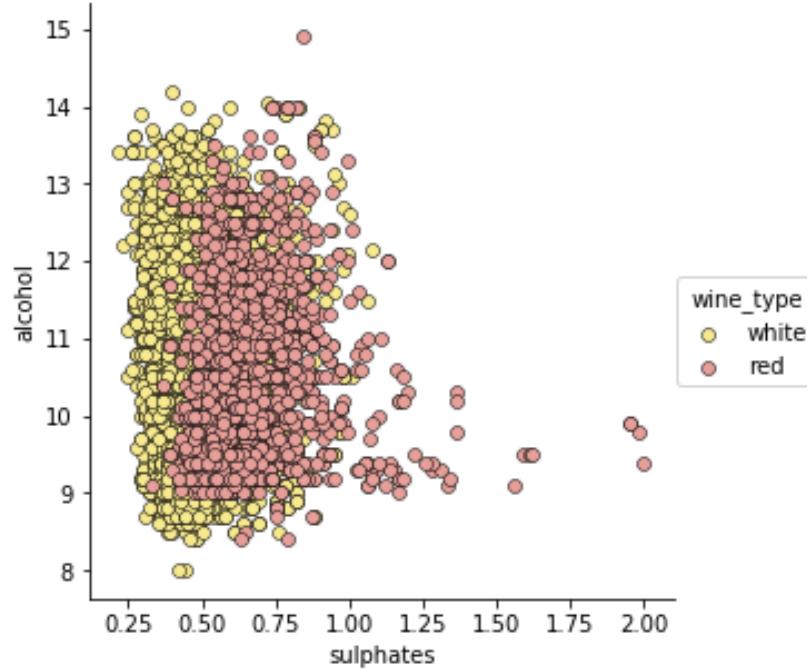
[https://matplotlib.org/mpl\\_toolkits/mplot3d](https://matplotlib.org/mpl_toolkits/mplot3d)

# Visualising 3 Dimensions – cont'd

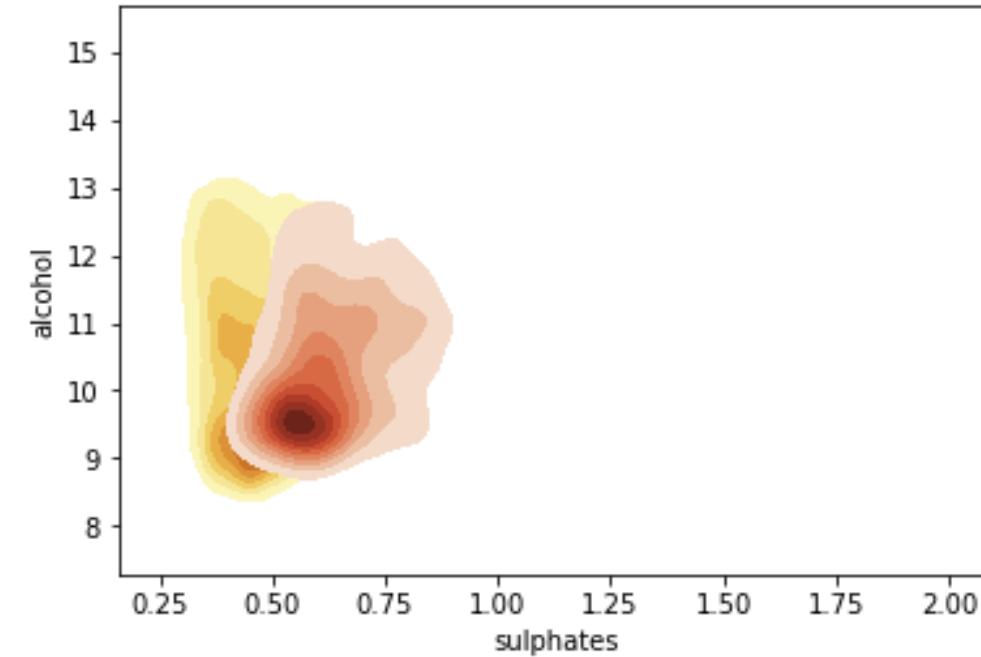
- adding colour allows stratification by a categorical variable (usually called a “class”)



# Visualising 3 Dimensions – cont'd



using colour in a scatterplot



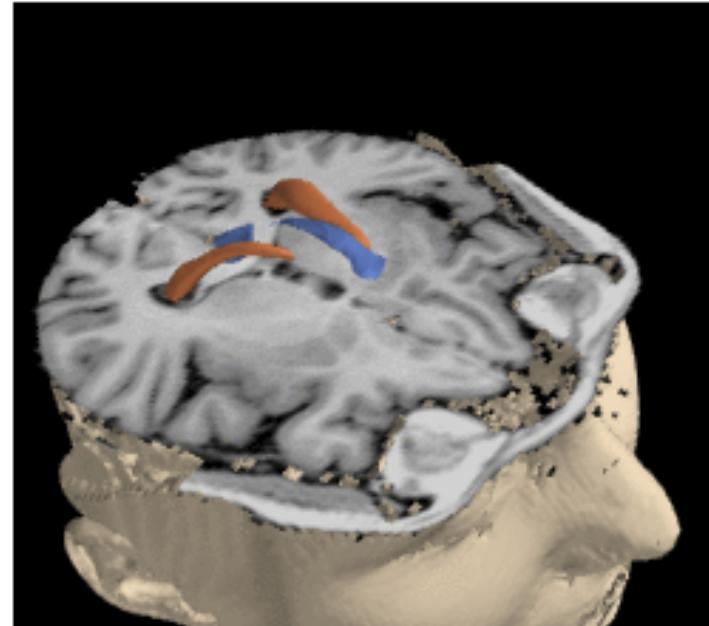
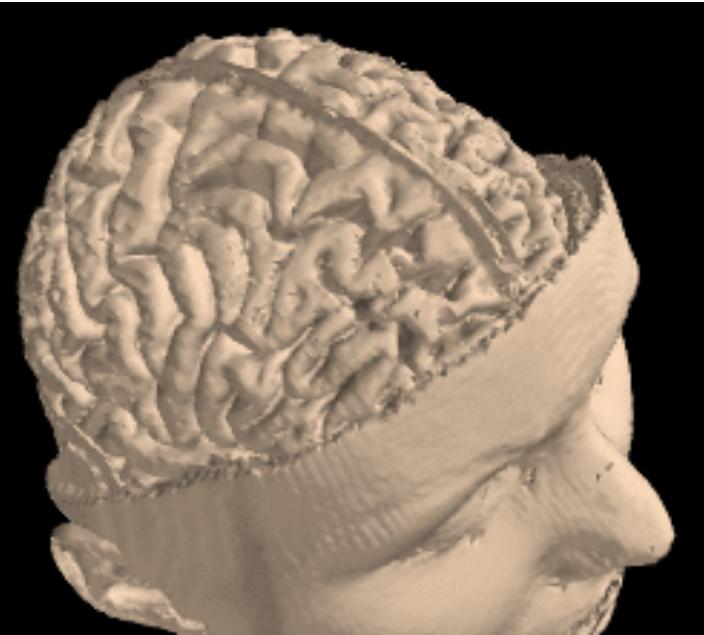
using colour and hue in a contour plot

<https://towardsdatascience.com/the-art-of-effective-visualization-of-multi-dimensional-data-6c7202990c57>

# Visualising 3 Dimensions – cont'd

## Slicing

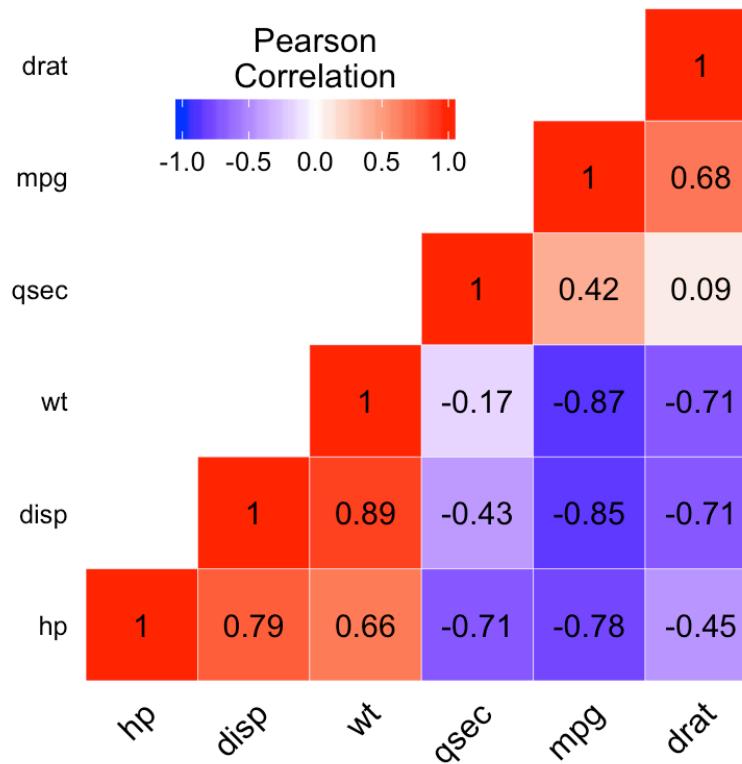
- reduce dimensionality by viewing a plane
- does not have to be parallel to a dimensional axis



<http://zulko.github.io/blog/2014/11/29/data-animations-with-python-and-moviepy/>

# Visualising 3 Dimensions – cont'd

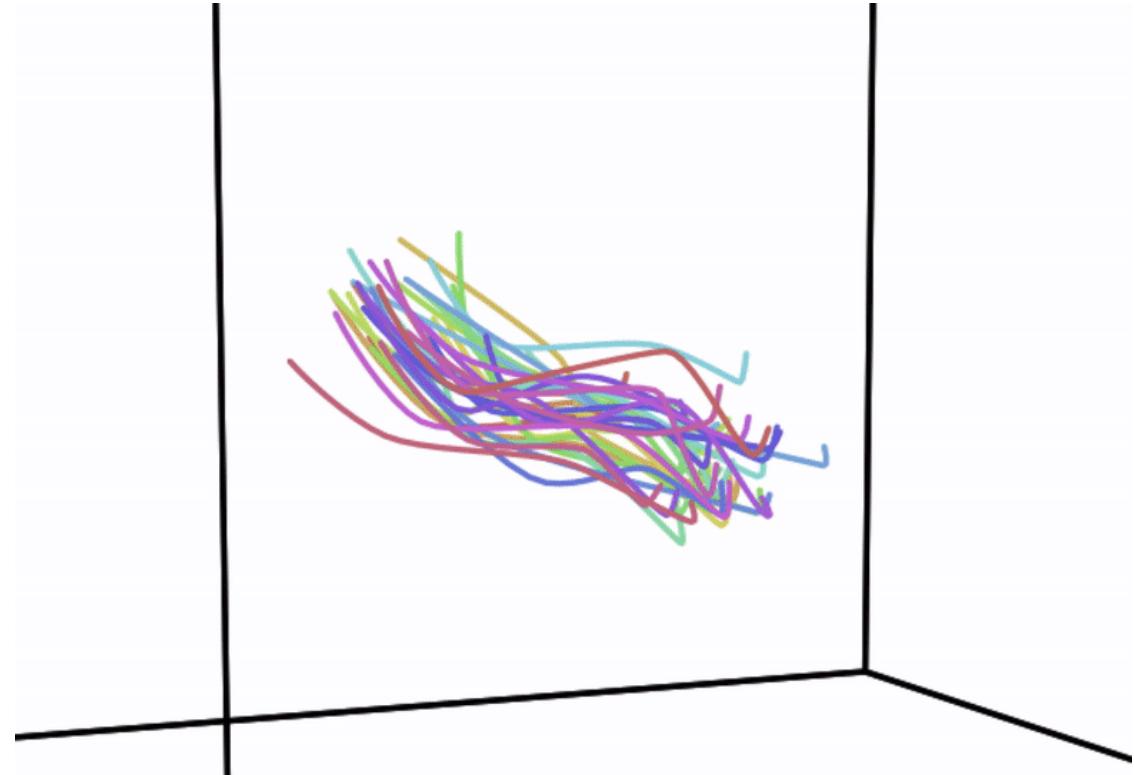
- heat map



# Visualising > 3 Dimensions

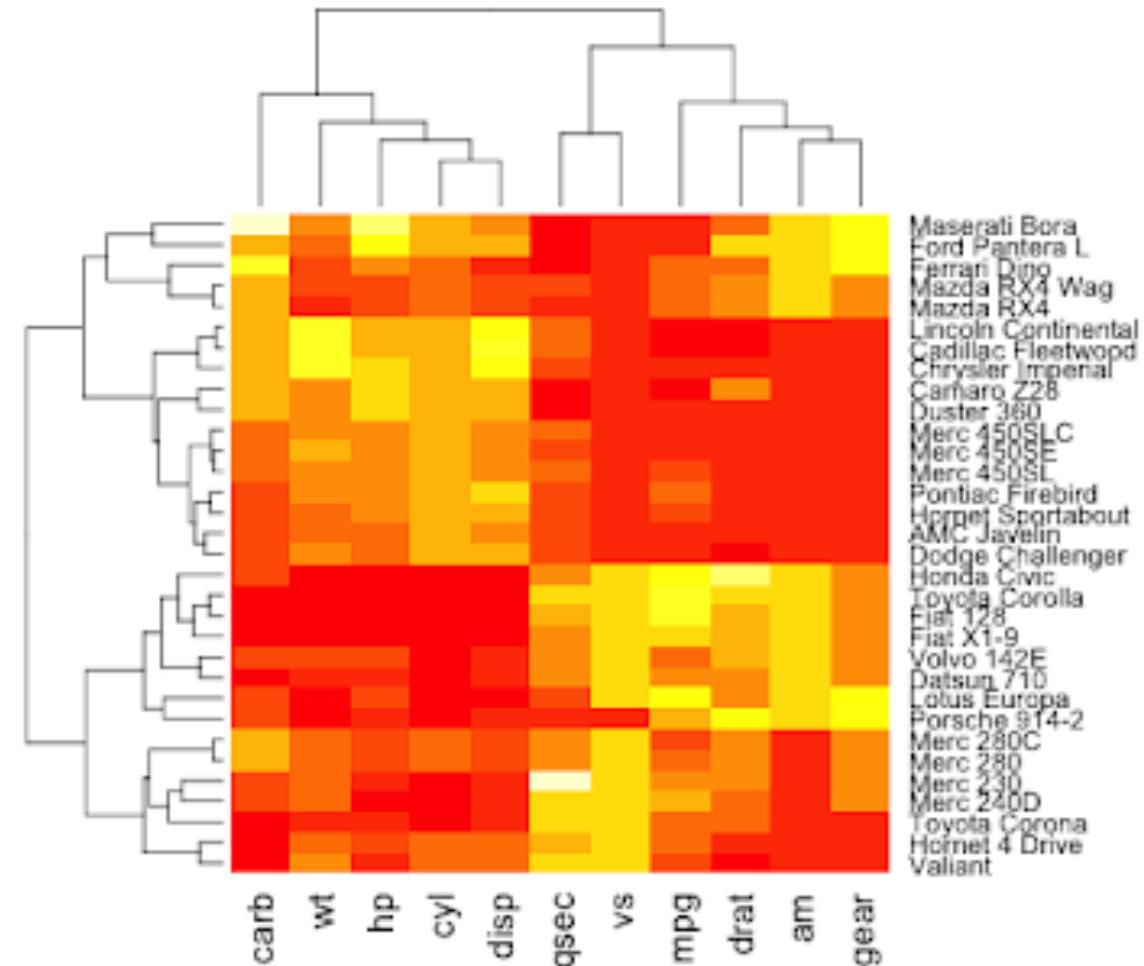
- dimensional reduction
  - e.g. to animated trajectories

<https://hypertools.readthedocs.io/en/latest/>



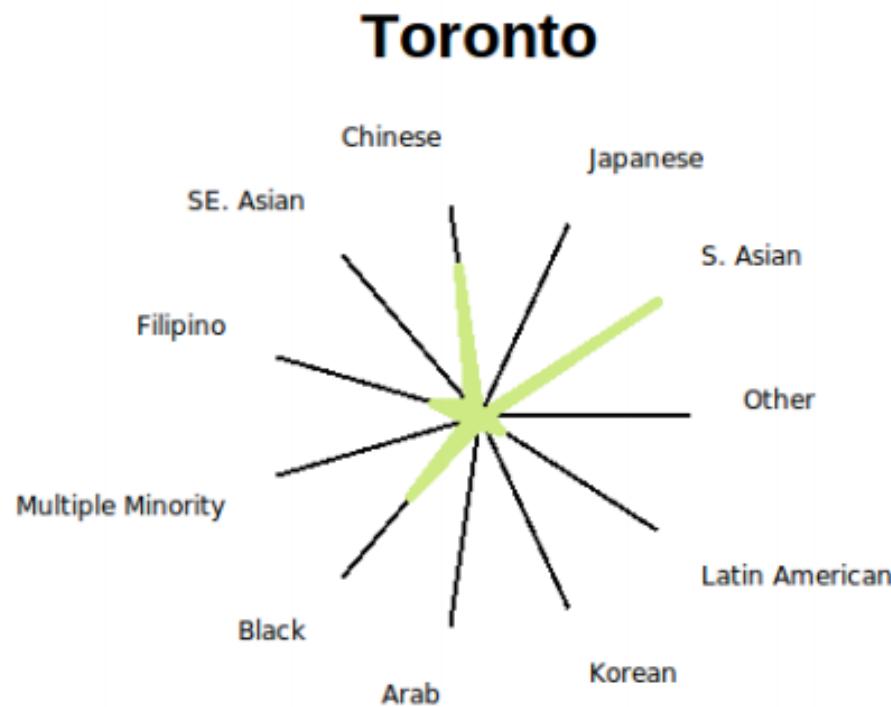
# Visualising > 3 Dimensions – cont'd

- N-dimensional correlation of categorical dimensions using a heat map
  - vertical axis = cars
  - horizontal = attributes (dimensions)

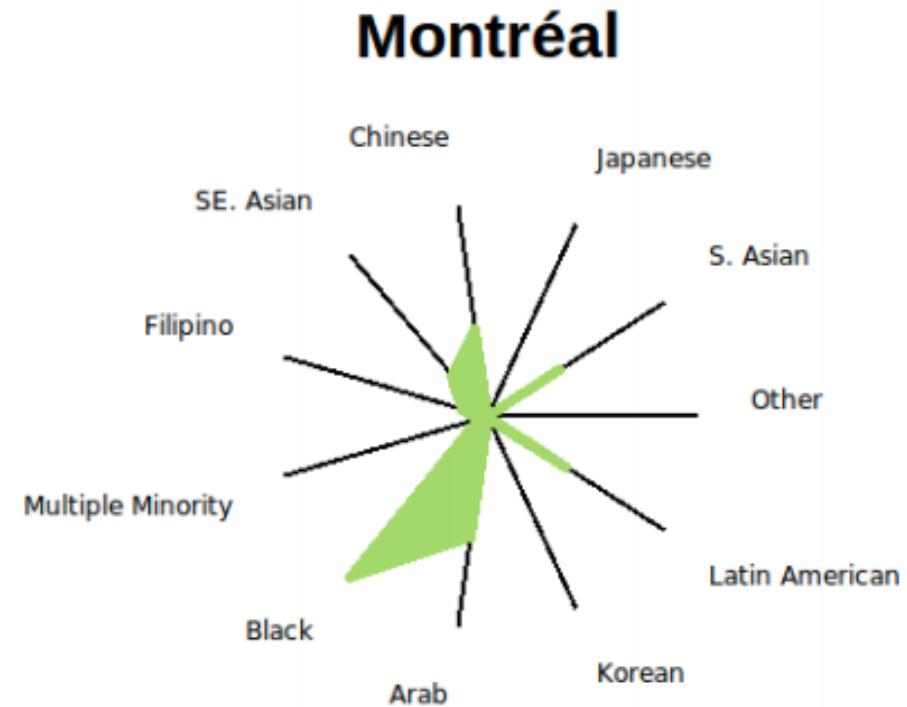


# Visualising > 3 Dimensions – cont'd

- radar chart



(a) Largest English speaking city



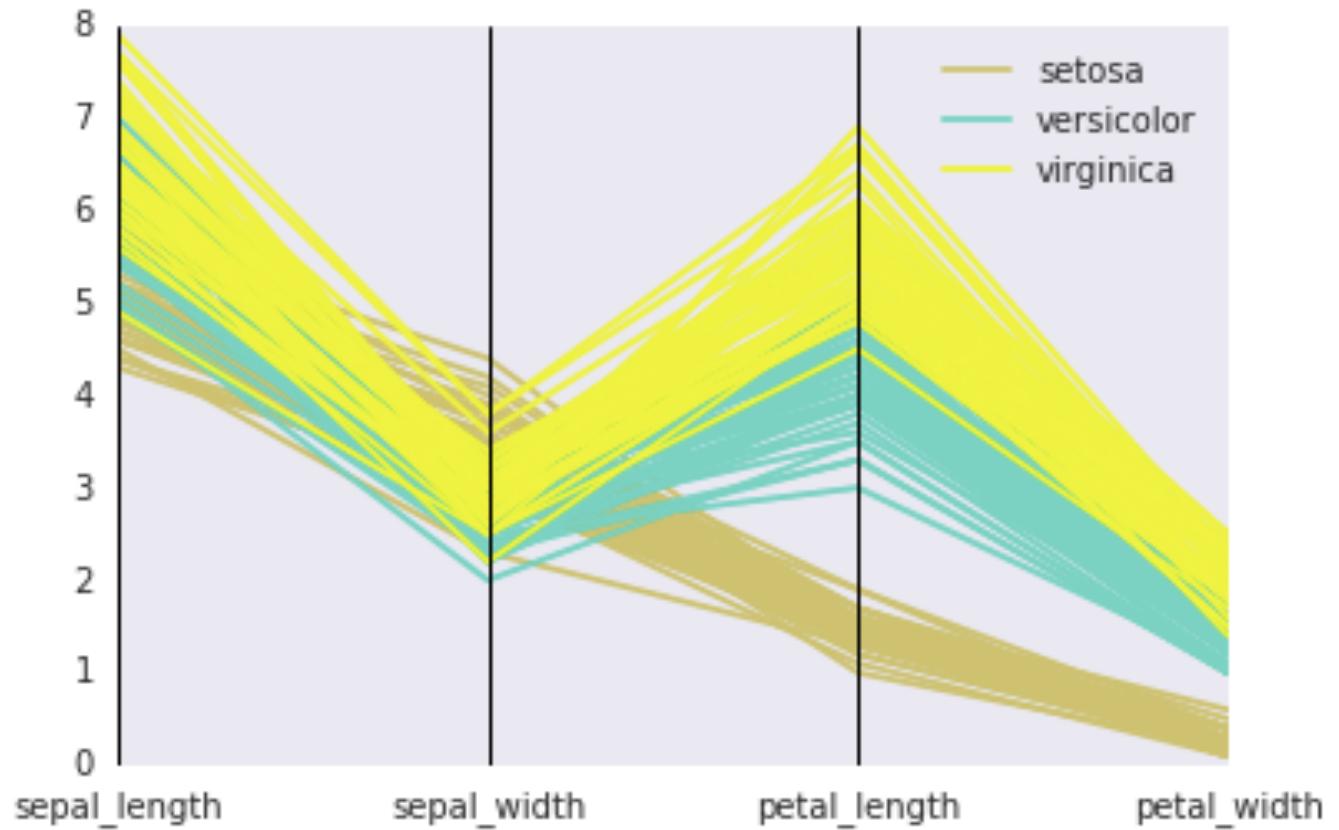
(b) Largest French speaking city

# Visualising > 3 Dimensions – cont'd

- parallel coordinates
  - can show multiple variables of same scale
  - especially useful for repeated measures
    - each variable is a time point in a longitudinal study

```
from pandas.tools.plotting import  
parallel_coordinates
```

```
parallel_coordinates(iris, 'species')
```

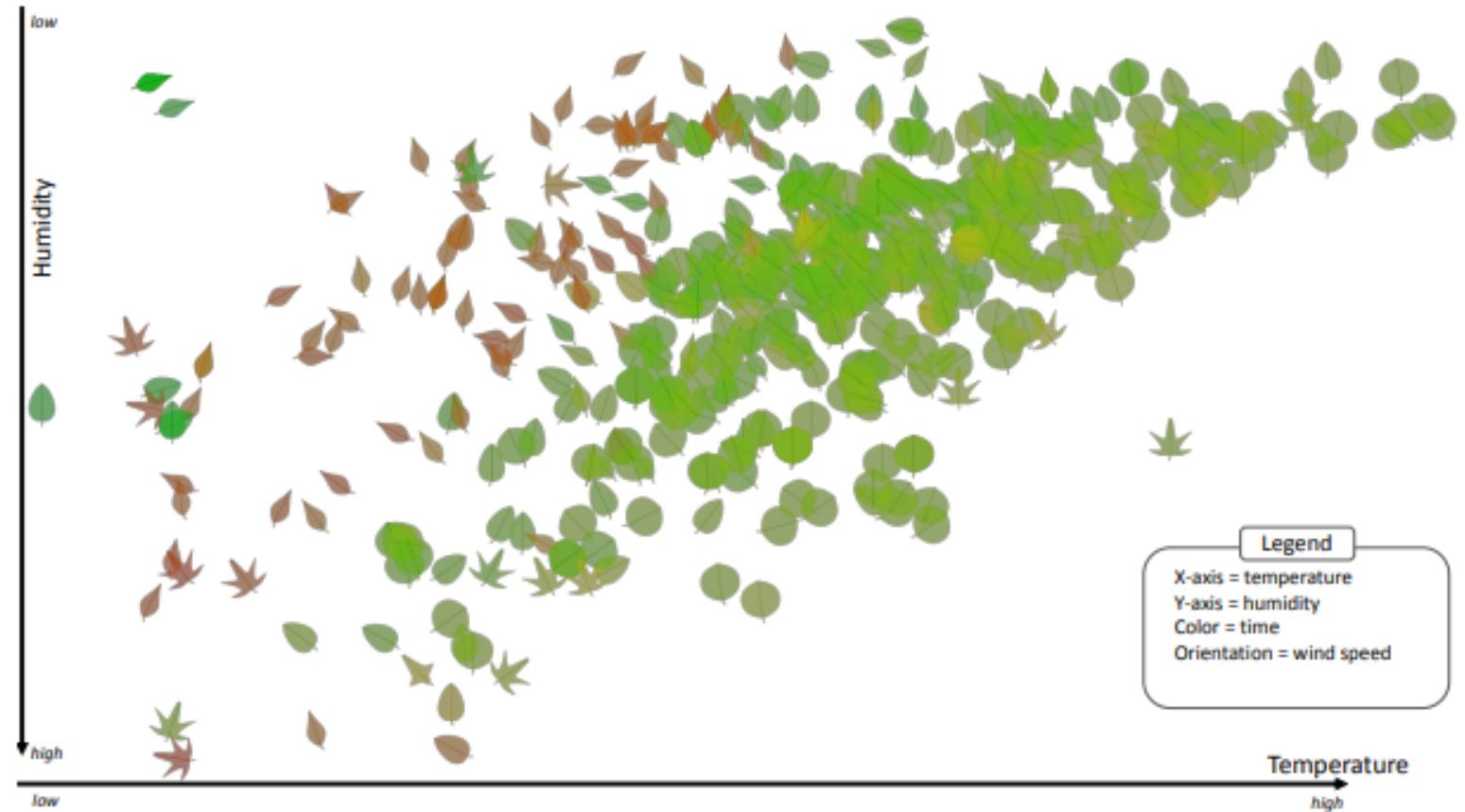


# Visualising > 3 Dimensions – cont'd

*scatterplot with glyphs*

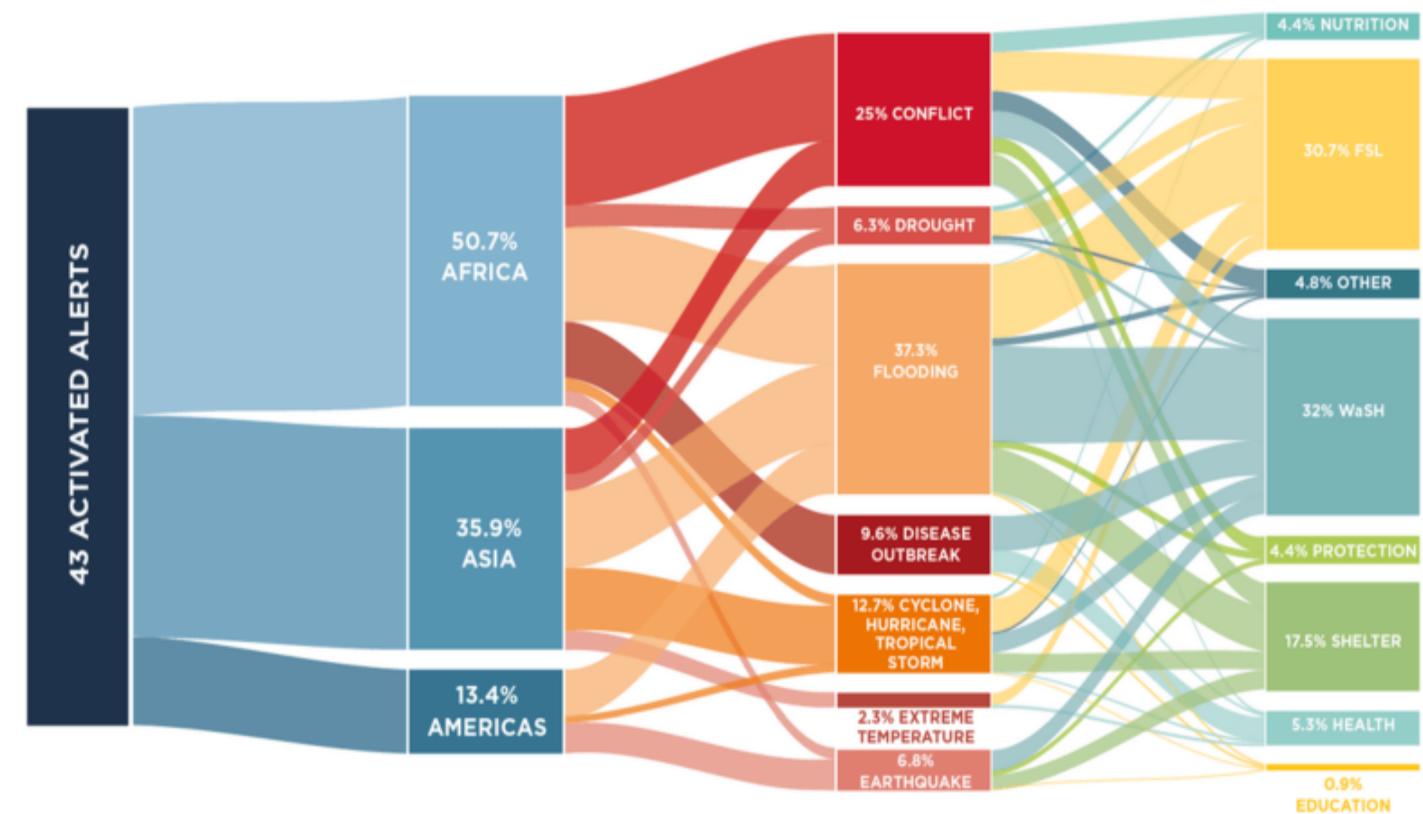
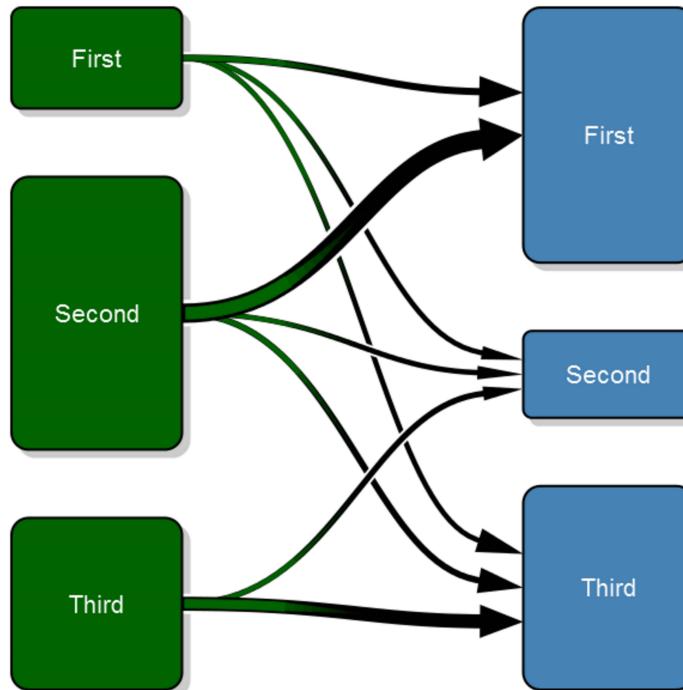
options for  
encoding glyphs:

- size
- colour
- intensity
- transparency
- shape
- texture



# Sankey Diagram

*state changes, class transitions, redistributions*



# Categorical Data

- Statistics of discrete distributions
  - class frequencies
- Exploring and visualising sample variables
  - bar plots
  - pie / donut charts
- Outlier detection

# Marginal Distributions of Discrete Variables

```
# donut chart recipe ===
```

```
# The slices will be ordered and plotted counter-clockwise.
```

```
data = [0.27, 0.67, 0.06]
```

```
labels = 'Low', 'Medium', 'High'
```

```
colors = ['yellowgreen', 'gold', 'lightskyblue']
```

```
plt.pie
```

```
(data, explode=(0,0), labels=labels, colors=colors, autopct='%.1f%%', shadow=False)
```

```
#draw a circle at the center of pie to make it look like a donut:
```

```
centre_circle = plt.Circle((0,0), 0.5, fc='white', linewidth=1.25)
```

```
fig = plt.gcf()
```

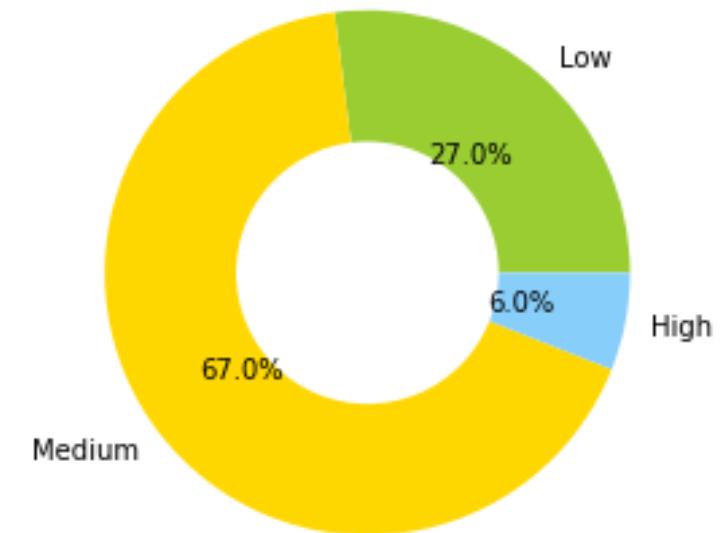
```
fig.gca().add_artist(centre_circle)
```

```
# Set aspect ratio to be equal so that pie is drawn as a circle:
```

```
plt.axis('equal')
```

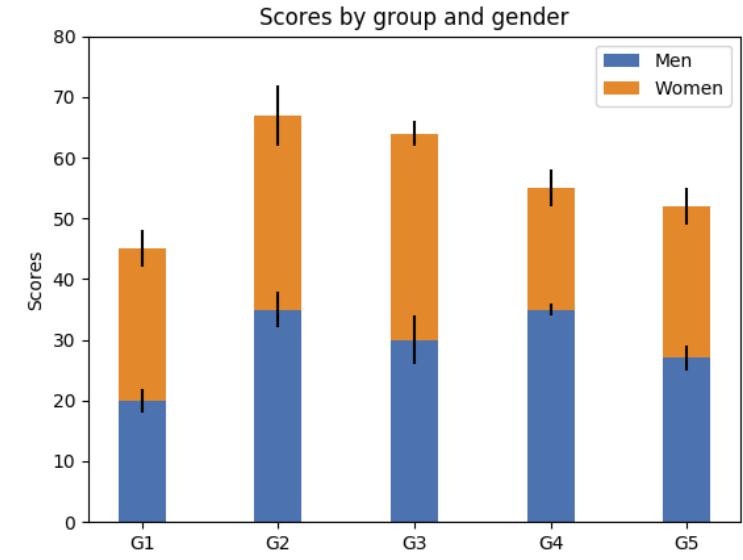
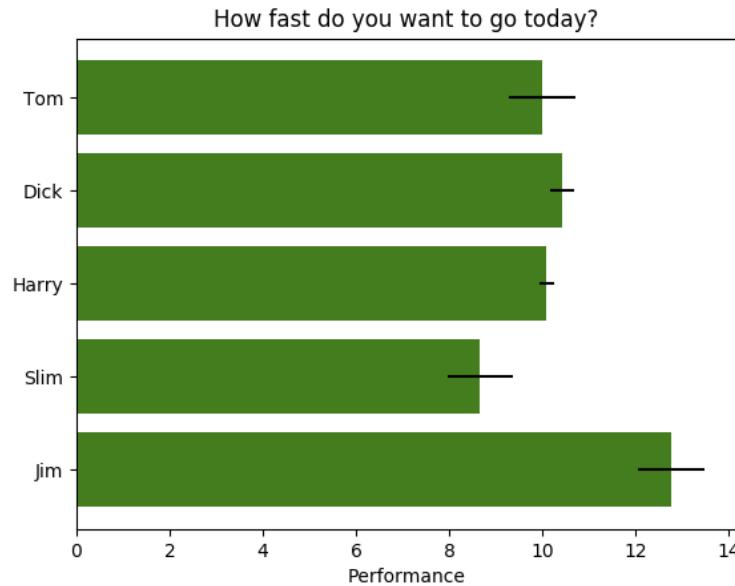
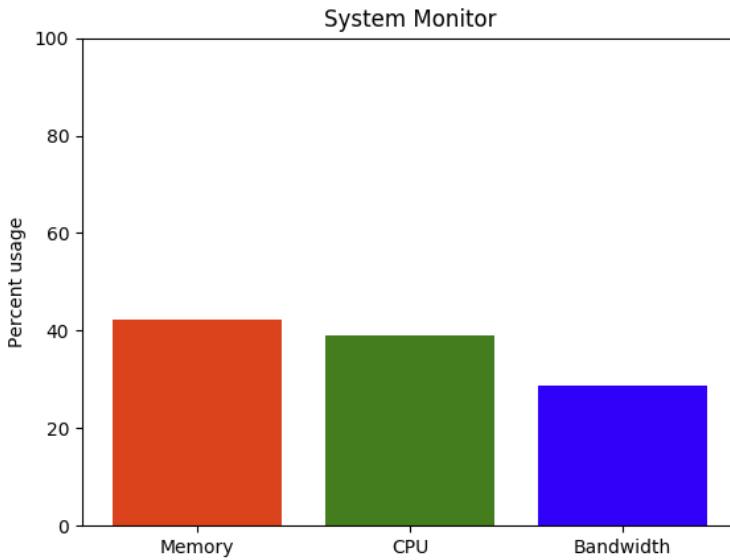
```
plt.show()
```

Income Bracket	
Low	0.27
Medium	0.67
High	0.06



# Bar Plots

- styles:
  - horizontal, vertical
  - grouped, stacked



# Conditional Distributions of Discrete Variables

- **contingency tables**
  - 2D:
    - var1 = rows, var 2 = columns
  - 3D:
    - var3 = planes (1 table for each value of var3)
  - > 3D:
    - multi-dimensional arrays
      - can be represented in code even if we can't visualise them

# Lab 3.1.2: Data Profiling

- Purpose:
  - To explore Python methods for exploring and summarising datasets
- Materials:
  - ‘Lab 3.1.2.ipynb’

# Exploring Large Datasets

- randomised sampling

```
1 bikes.sample(5)
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
9870	9871	2012-02-21	1	1	2	7	0	2	1	1	0.22	0.2727	0.64	0.0000	6	273	279
16419	16420	2012-11-21	4	1	11	21	0	3	1	1	0.36	0.3788	0.50	0.0000	8	97	105
6558	6559	2011-10-05	4	0	10	20	0	3	1	1	0.52	0.5000	0.77	0.1642	18	228	246
15577	15578	2012-10-16	4	1	10	6	0	2	1	1	0.42	0.4242	0.67	0.1642	4	168	172
16855	16856	2012-12-10	4	1	12	2	0	1	1	2	0.38	0.3939	0.94	0.1045	2	3	5

- repeated sampling
  - collect a number of random subsets from the sample population
  - analyse each subset
  - aggregate the results

# The Central Limit Theorem

- Suppose we take  $n$  samples from a distribution and compute the mean  $\bar{x}_k$  of each sample

then, as  $n \rightarrow \infty$

- the set of  $\bar{x}_k$  approaches a normal distribution
- the mean of  $\bar{x}_k$  approaches the mean of the original distribution

$$\lim_{n \rightarrow \infty} \frac{1}{n} \sum_{k=1}^n \bar{x}_k = \mu$$

- implication:

*by repeated resampling of a non-normal distribution, we can apply all (?) the statistical methods that were designed for normal distributions (as long as the samples are independent and identically distributed)*

# Lab 3.1.3: The Central Limit Theorem

- Purpose:
  - To test the central limit theorem by experiment
- Materials:
  - ‘Lab 3.1.3.ipynb’



# Time Series

- What is a time series?
- How are time series represented in Python?

# Time Series

*def:* a sequence of data points representing the state of a system over time

classes of time series:

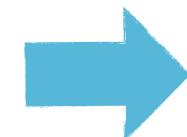
- temporally deterministic
  - periodic
    - pattern repeats at equal intervals
  - aperiodic
    - state at time  $t_k$  is influenced by state at time  $t_{k-1}$  but there is no repeating pattern
- stochastic
  - state at time  $t_k$  is unrelated to state at time  $t_{k-1}$

# Programming with Time Series

- timebase is usually regular
  - seconds, days, or years (typically)
    - may need to cope with leap years
  - no gaps
    - may need to impute or assign NA for missing time points

*example (Pandas):*

```
index = pd.DatetimeIndex(['2014-07-04', '2014-08-04',
                           '2015-07-04', '2015-08-04'])
data = pd.Series([0, 1, 2, 3], index=index)
data
```



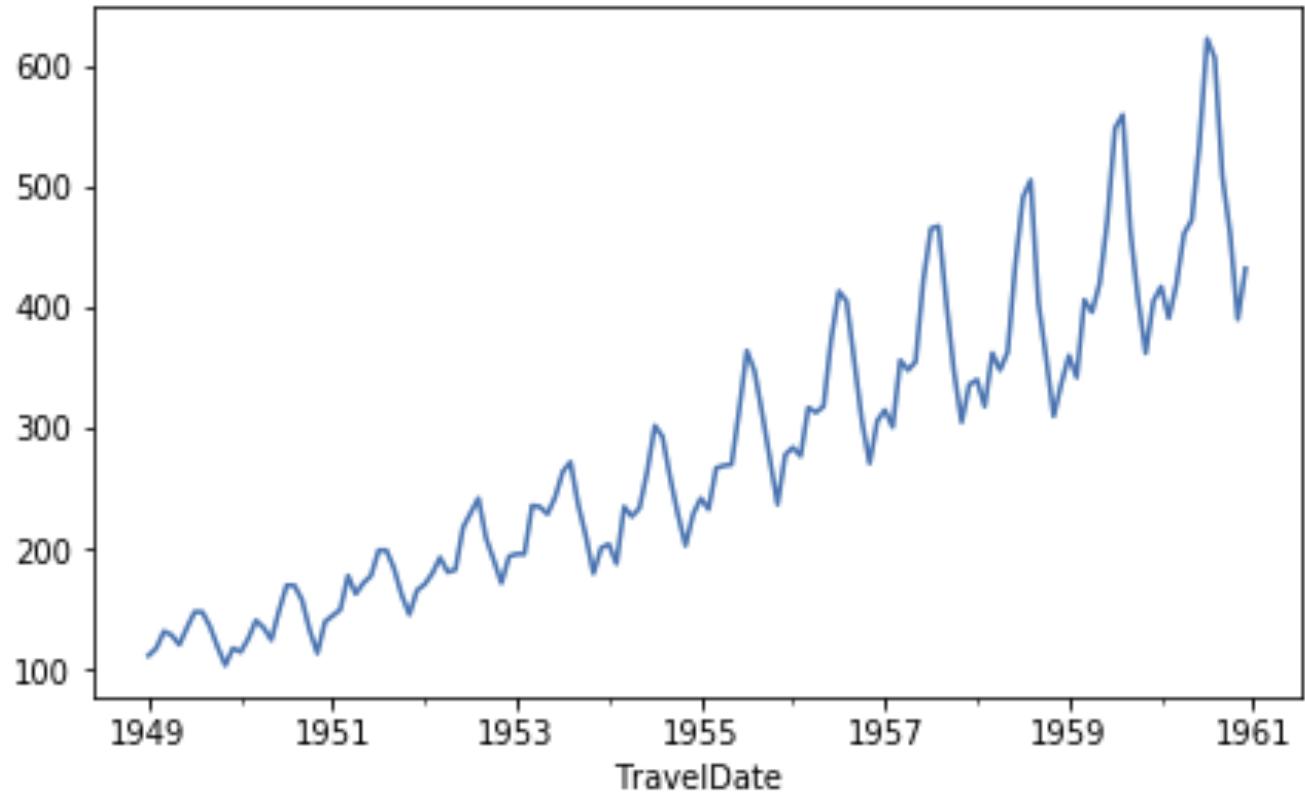
2014-07-04	0
2014-08-04	1
2015-07-04	2
2015-08-04	3
<b>dtype:</b>	<b>int64</b>

# Visualising Time Series

## Static time series

- convert DataFrame to Pandas time series
- timebase is an index of the DataFrame
- default axis labelling is aware of timebase

`ts.plot()`

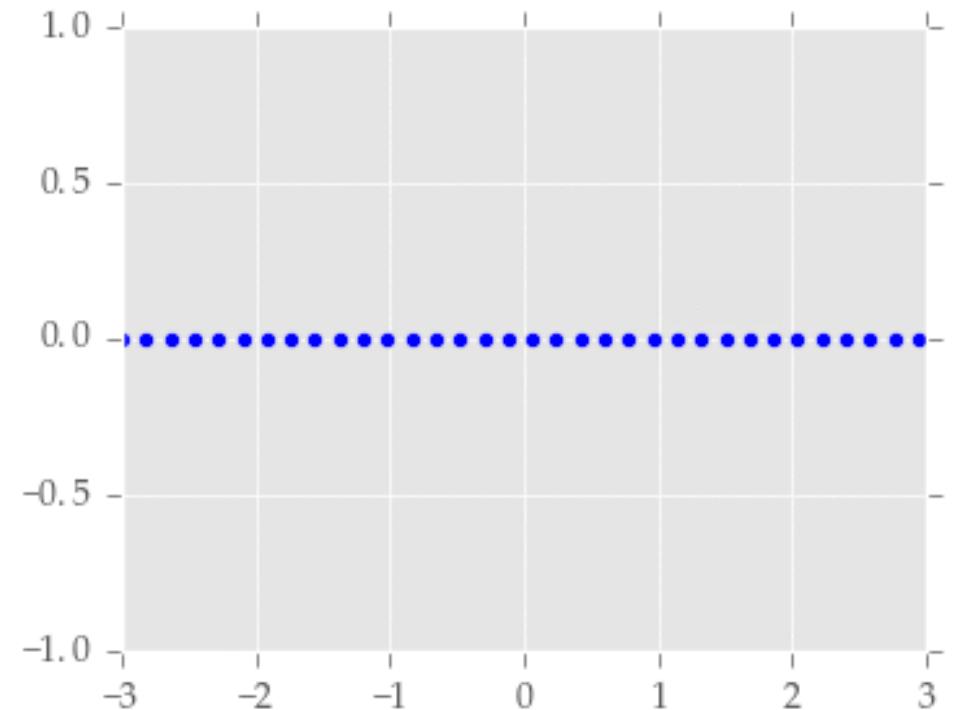


# Visualising Time Series – cont'd

## Animated time series

- compact way to present the evolution of a dataset over time
- reveals temporospatial patterns more effectively than static images
- provides cues for interpreting 3<sup>rd</sup> dimension within the 2D projection

[https://brushingupscience.com/2016/06/21/  
matplotlib-animations-the-easy-way/](https://brushingupscience.com/2016/06/21/matplotlib-animations-the-easy-way/)



# Geospatial Data

- How are geospatial data organised?
- Tools for exploring geospatial data
- Visualising geospatial data in Python

# Geospatial Data Formats

- GIS
  - range of open (standard) and proprietary formats
    - raster, vector, grid
    - metadata
- typically
  - a list with nested structure
- arrays / lists
  - coordinates
  - attributes
    - built-in (e.g. elevation)
    - user-defined (e.g. derived statistics)

# Geospatial Data Formats – cont'd

## *shape files*

- most common
- a set of file types
  - SHP = feature geometry
  - SHX = shape index position
  - DBF = attribute data
  - PRJ = projection
  - XML = additional metadata
  - SBN = spatial index

# Geospatial Data Formats – cont'd

## *Keyhole Markup Language*

- primarily used for Google Earth
- .KMZ/.KML

## *Open Streetmap*

- largest crowdsourcing GIS data project of the planet Earth
- .OSM

## *GeoJSON*

- open standard format designed for representing simple geographical features
- .geojson

# Tools for Exploring Geospatial Data

- interactive maps/ APIs
- base map may be featureless
  - add *tiles* to display features
    - street map
    - topography
    - satellite view
- data organised, rendered in *layers*
- ability to overlay image data from other sources
  - weather
    - satellite view
    - simulations



ArcGIS



# Geospatial Libraries for Python

## Shapely

- manipulation of geometric objects

## Fiona

- read/write vector file formats (e.g. shapefiles or geojson)
- projection conversions

## Geopandas

- all of the above

## Rasterio

- read/write raster file formats

## GDAL

- more complex manipulations of raster and vector data

# Geospatial Libraries for Python – cont'd

*other useful libraries*

- rasterstats
- scikit-image
- scikit-learn
- folium
- descartes
- xarray

# Visualising Geospatial Data

## Geoplot

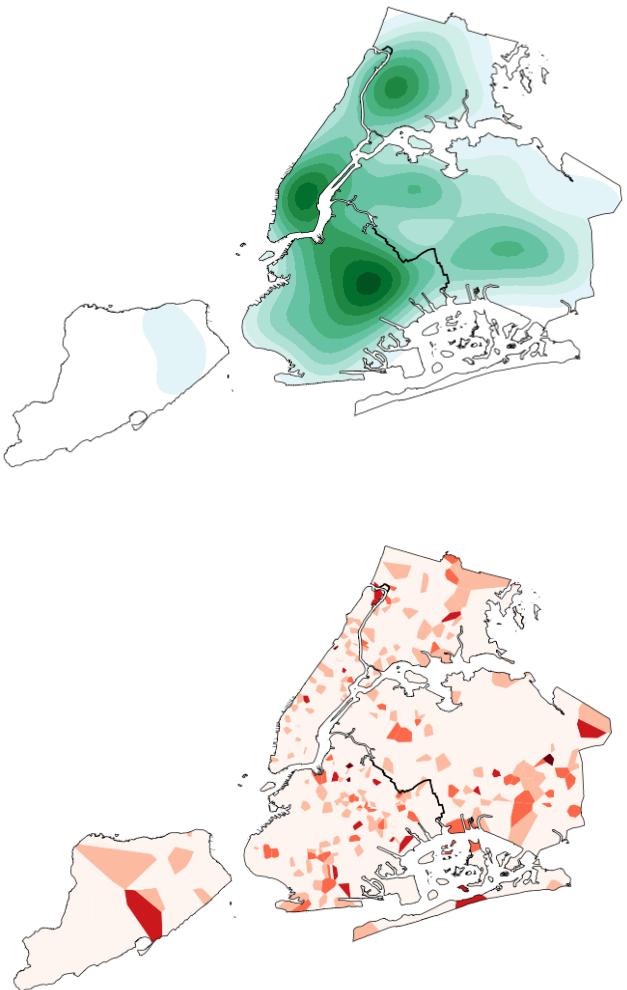
- works with GeoPandas

## DataMaps

- interactive SVG maps using D3.js

## Carto

## Leaflet



# Discussion

- Questions?

# HOMEWORK

1. Load the ‘titanic’ dataset into a DataFrame
2. Using a Jupyter Notebook, explore/profile the data using the techniques covered in this module
3. Upload your Notebook to your GitHub repo and share the link with the course instructors.