

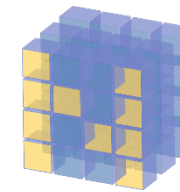
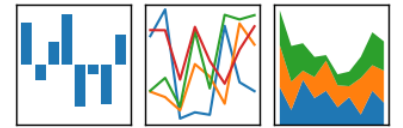
# Data Science in Python

Oxford Biomedical Data Science Training Programme

# Python Data Science

- Data science – analysis and visualisation of data
- Pandas is a Python package for data analysis
  - Gives **R**-like functionality to Python
  - Built using the numerical analysis package **NumPy**
  - N-dimensional data structures
  - Fast matrix algebra calculations
- Plotting in Python
  - Many packages built on Matplotlib
  - Seaborn
- Jupyter notebook / Jupyter Lab

pandas  
 $y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



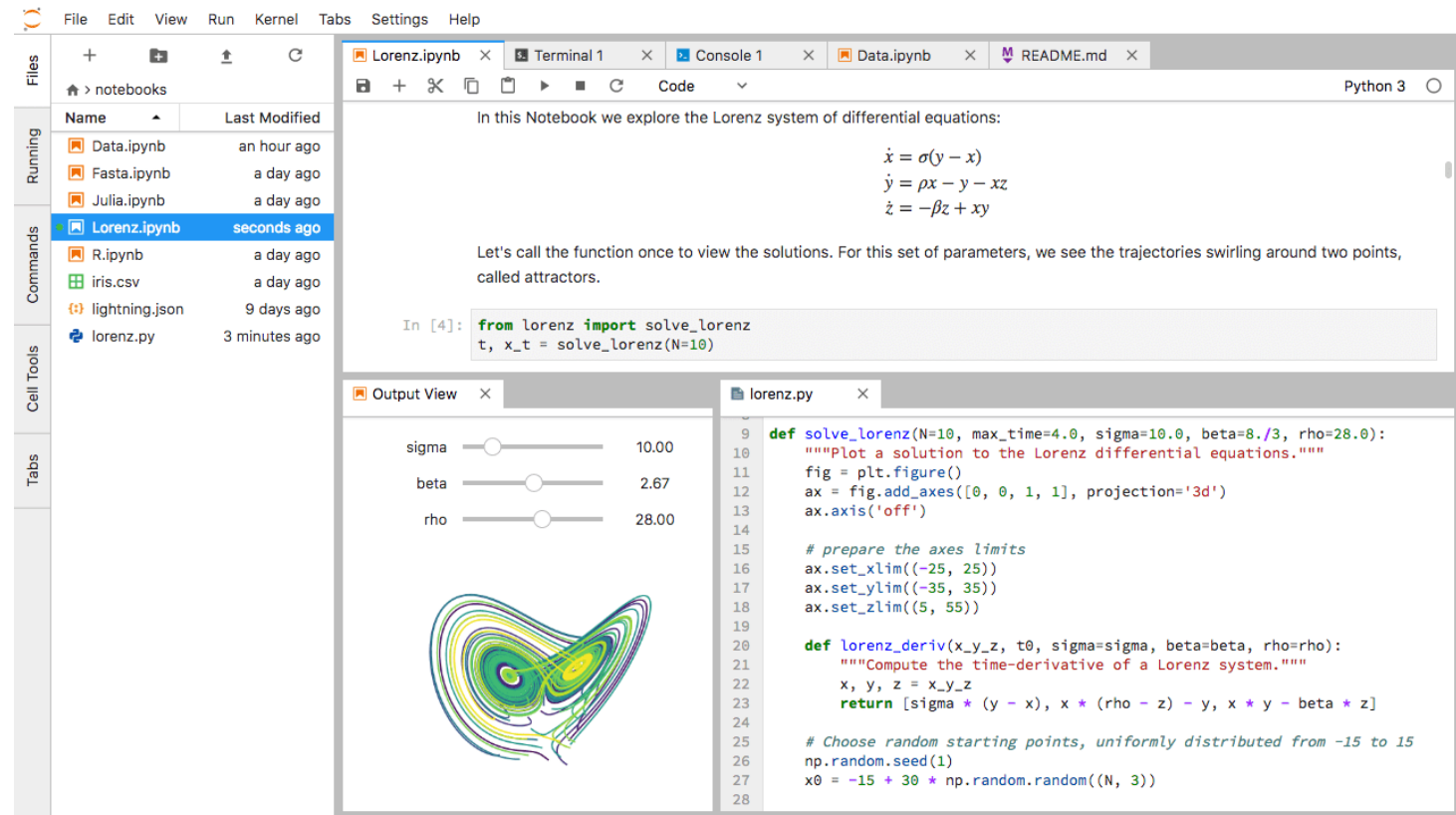
NumPy

matplotlib

# Jupyter Lab



- A lab book for exploratory data analysis
- Combines live code, equations, visualizations & narrative text
- Runs in your browser
- Build reports
- Shareable

A screenshot of the Jupyter Lab web interface. The interface is divided into several panes. On the left, a sidebar shows a file explorer with a list of notebooks and files, including 'Data.ipynb', 'Fasta.ipynb', 'Julia.ipynb', 'Lorenz.ipynb' (selected), 'R.ipynb', 'iris.csv', 'lightning.json', and 'lorenz.py'. The main area is split into three sections. The top section is a code editor showing a notebook cell with text and mathematical equations for the Lorenz system. The bottom-left section is an 'Output View' showing a 3D plot of the Lorenz attractor with sliders for parameters sigma, beta, and rho. The bottom-right section is a code editor showing the Python code for the Lorenz system simulation.

File Edit View Run Kernel Tabs Settings Help

Files

notebooks

Name	Last Modified
Data.ipynb	an hour ago
Fasta.ipynb	a day ago
Julia.ipynb	a day ago
Lorenz.ipynb	seconds ago
R.ipynb	a day ago
iris.csv	a day ago
lightning.json	9 days ago
lorenz.py	3 minutes ago

Running

Commands

Cell Tools

Tabs

Lorenz.ipynb

In this Notebook we explore the Lorenz system of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

Let's call the function once to view the solutions. For this set of parameters, we see the trajectories swirling around two points, called attractors.

In [4]: `from lorenz import solve_lorenz  
t, x_t = solve_lorenz(N=10)`

Output View

sigma: 10.00  
beta: 2.67  
rho: 28.00

lorenz.py

```
9 def solve_lorenz(N=10, max_time=4.0, sigma=10.0, beta=8./3, rho=28.0):  
10     """Plot a solution to the Lorenz differential equations."""  
11     fig = plt.figure()  
12     ax = fig.add_axes([0, 0, 1, 1], projection='3d')  
13     ax.axis('off')  
14  
15     # prepare the axes limits  
16     ax.set_xlim((-25, 25))  
17     ax.set_ylim((-35, 35))  
18     ax.set_zlim((5, 55))  
19  
20     # Compute the time-derivative of a Lorenz system.  
21     def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):  
22         x, y, z = x_y_z  
23         return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]  
24  
25     # Choose random starting points, uniformly distributed from -15 to 15  
26     np.random.seed(1)  
27     x0 = -15 + 30 * np.random.random((N, 3))  
28
```

<https://jupyterlab.readthedocs.io/en/stable/>

# Tunneling Jupyter Lab

- ssh to cgath1
- cd to working directory
- Load Python Conda environment
- Start Jupyter Lab
  - jupyter lab --no-browser --port=xxxx
  - Everyone use a different port number
- New ssh to cgath1 - nb
- Copy jupyter URL to local browser
  - http://localhost:<port>/lab

## **.ssh/config**

```
Host      nb
hostname  cgath1
User      <username>
ExitOnForwardFailure yes
ServerAliveInterval 300
ServerAliveCountMax 2
RequestTTY no
ProxyCommand
           ssh <username>@cgatui.imm.ox.ac.uk nc %h %p
LocalForward localhost:<port> localhost:<port>
```

# NumPy



- Array
  - N-dimensional
  - elements of the same type
- Many numeric datatypes
  - int64, float64
- mathematical functions operate elementwise on arrays
  - operator overloads
  - functions
- Fast matrix algebra operations
- Slicing and indexing

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum
print(x + y)
print(np.add(x, y))

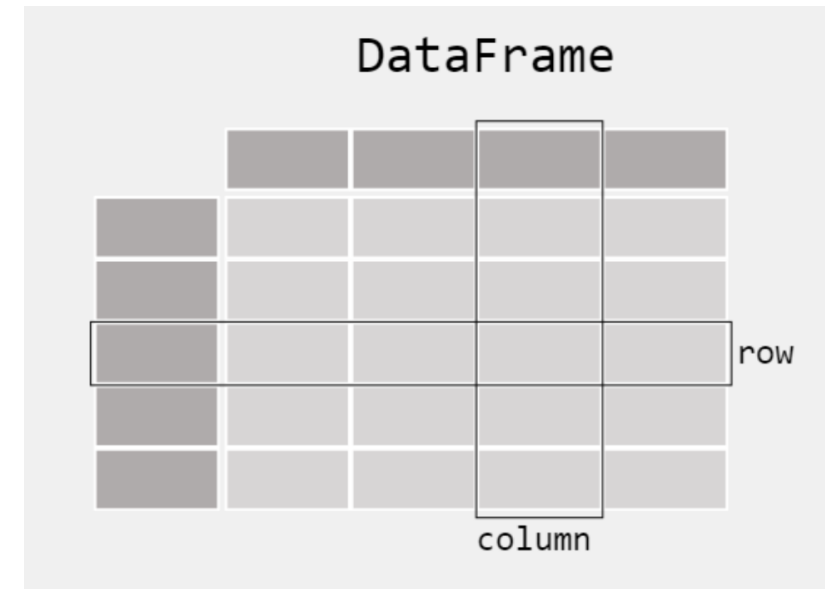
# Transpose attribute
print(x.T)

# Slicing
b = x[:2, 1:3]
```

# Pandas



- Dataframes in Python
- Built on numpy arrays
- Adds row and column metadata



[https://pandas.pydata.org/docs/user\\_guide/index.html](https://pandas.pydata.org/docs/user_guide/index.html)  
[https://pandas.pydata.org/docs/getting\\_started/comparison/comparison\\_with\\_r.html](https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_r.html)

# Pandas

- **Easy to convert** Python and NumPy data structures into **DataFrame** objects
- Robust IO from **flat files** (CSV & delimited), Excel, databases, HDF5 format
- Size mutability: columns can be **inserted and deleted** from DataFrames
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** data sets
- Easy handling of **missing data** (represented as NaN)
- Vectorised numerical and string **operations**
- Intuitive **merging** and **joining** data sets
- Powerful **group by** functionality to perform split-apply-combine operations
- Flexible **reshaping** and pivoting of data sets

# Pandas Series

- One-dimensional vector
- Built on NumPy array
- Includes axis labels
- Can be sliced like a list
- Can be indexed by label
- Not only numeric data
  - Any Python Object
  - Single data type

```
import numpy as np
import pandas as pd
```

```
# pandas creates default integer index
S = pd.Series([1, 2, 3, 4, 5, 6, 7, 8])
```

```
# Add our own index
s = pd.Series(np.random.randn(5),
              index=['a', 'b', 'c', 'd', 'e'])
```

```
# index & name are attributes of Series class
s.index
s.name
```

```
# Slicing
s[:3]          # list like slicing
s['a']         # Index using labels
s[[4, 3, 1]]  # array-based indexing
```

```
s + s # element-wise addition
s * 2 # element-wise multiplication
```



# Pandas Dataframe

- Two-dimensional
- Like R dataframe
- Includes axis labels
- Create from any 2d Python data structure
  - Dict of Series / dict of dicts
  - List of dicts / dict of lists
  - Dict of tuples
- Not only numeric data
  - Any Python Object
- Different data type per column

```
# Dataframe from dictionary of series
d = {'one': pd.Series([1., 2., 3.],
                      index=['a', 'b', 'c']),
      'two': pd.Series([1., 2., 3., 4.],
                      index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
```

```
# attributes
df.index    # row labels
df.columns  # column labels
df.shape    # dataframe size
```

```
# Viewing dataframes
df.head()
df.tail()
```

```
# Summary
df.info()
```

# Column Addition & Deletion

```
# Adding columns
```

```
df['three'] = df['one'] * df['two']
```

```
df['flag'] = df['one'] > 2
```

```
# Assign - like dplyr mutate
```

```
df.assign(four = df['one'] / df['two'])
```

```
# Deleting columns
```

```
del df['flag']
```

```
four = df.pop('four')
```

# Slicing / Selection

- Slicing like Python lists
- Also optimised pandas methods – loc/iloc
- By label or position (0-based, exclusive)
- Can use lists of labels / indexes
- Boolean indexing

Operation	Syntax	Result
Select column	df[col]	Series
Select row by label	df.loc[label]	Series
Select row by integer location	df.iloc[loc]	Series
Slice rows	df[5:10]	DataFrame
Select rows by boolean vector	df[bool_vec]	DataFrame

```
# Select single column
df['one']
# Select multiple columns (list)
df[['two', 'one']] # any order

# Slice data frame rows / cols
df[:3]
df.loc['b',:]
df.loc['c':, 'one':'three']
df.iloc[2:]
df.iloc[1:3, 2:3]

# Boolean indexing
# select rows on column values
df[df.one > 1]
# select values on condition
df[df > 2]
```

# Categorical Variables

- Category column type in dataframe
- Like factor in R
- Categories can be ordered
- Sorting is per order in the categories, not lexical order
- Grouping by a categorical column also shows empty categories

```
df["biotype"] = df["raw_biotype"].astype("category")  
df["biotype"].cat.categories = ["protein_coding", "rRNA", "lincRNA"]
```

# Missing data

- Pandas uses `np.nan` to represent missing data
- By default not included in computations
- Drop or fill NAs
- Create a Boolean mask where values are nan

```
# drop rows with nan  
df.dropna(how='any')
```

```
# Replace nan with specific value  
df.fillna(value=5)
```

```
# Create Boolean mask  
pd.isna(df)
```

# Operations

- Operations with scalars are element-wise
- Boolean operators
- Descriptive statistics
  - mean()
- Numpy functions
- Matrix algebra

```
# Element-wise mathematical operations  
df * 5  
df **3
```

```
# Boolean logic  
df1 & df2
```

```
# Numpy functions  
np.log(df)
```

```
# Statistical summary  
df.describe()
```

```
# statistical functions  
df.mean() # per column  
df.mean(1) # per row
```

```
# Matrix algebra  
df.T # transpose of matrix  
df.dot(df) # matrix multiplication
```

# Apply

- Apply functions along an axis of a data frame
- Like apply in R

```
# apply to entire dataframe  
df.apply(np.sqrt)
```

```
# apply to cols  
df.apply(np.sum, axis=0)
```

```
# apply to rows  
df.apply(np.sum, axis=1)
```

# Sorting Dataframes

```
# Sort by single column
```

```
df.sort_values(by=['col1'])
```

```
# Sort by multiple columns
```

```
df.sort_values(by=['col1', 'col2'])
```

```
# Descending order
```

```
df.sort_values(by='col1', ascending=False)
```

```
# NAs first
```

```
df.sort_values(by='col1', ascending=False, na_position='first')
```



# Combining datasets

- Linux like concatenation - `concat`
- SQL style merges
  - `merge()`
  - `join()`
- Much faster than R

```
# Concatenate rows
pd.concat([df[:3], df[3:7], df[7:]])
```

```
# Merge (inner by default)
result = pd.merge(df1, df2, on='key')
result = pd.merge(df1, df2, on='key',
                  how='left')
```

```
# Join on index
result = df1.join(df2, how='outer')
# Join on key
result = df1.join(df2, on='key')
```

Merge method	SQL Join Name	Description
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTERJOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

# Grouping

- Group by
  - **Split** the data into groups based on some criteria
  - **Apply** a function to each group independently
  - **Combine** the results into a data structure

```
# Group rows by categorical variable in column A
# Then calculate the sum for each group
# Produces a dataframe where rows are groups in A
# and columns are sum of each numerical column
df.groupby('A').sum()
```

# Reshaping

- Many plotting packages require tidy data
- melt = melt / gather in R
- pivot = cast / spread in R

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766

country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

Melt

```
melted = df.melt(id_vars=['country'], var_name='year', value_name='cases')  
# Unmelt  
df = melted.pivot(index='country', columns='year')
```

# Loading and Saving Data

- Read from / write to many formats

- CSV
- TSV
- Excel
- HDF5
- Databases

```
# Read and write to CSV
pd.read_csv('foo.csv')
df.to_csv('foo.csv')
```

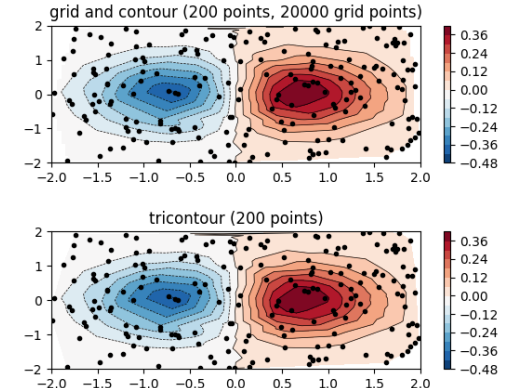
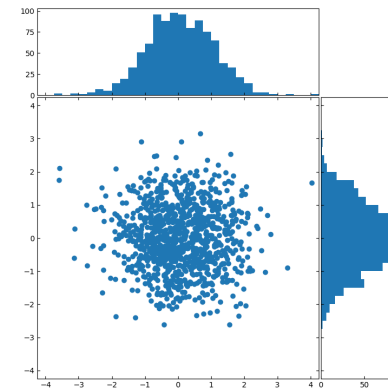
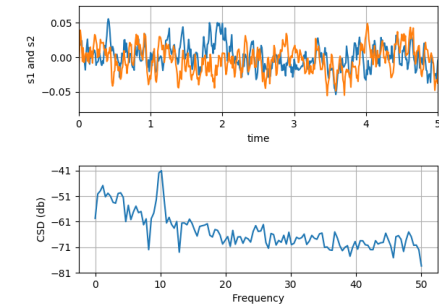
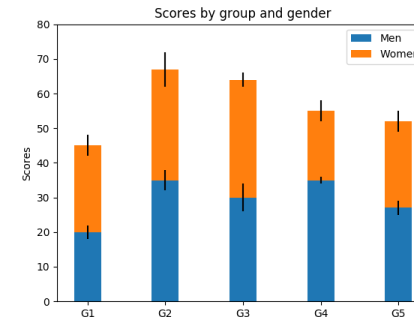
```
# tsv
pd.read_csv('foo.tsv', sep='\t', header=0)
pd.read_table(fpath) # tab default
```

```
# Excel
pd.read_excel('foo.xlsx', 'Sheet1',
              index_col=None, na_values=['NA'])
df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

```
# HDF5
pd.read_hdf('foo.h5', 'df')
df.to_hdf('foo.h5', 'df')
```

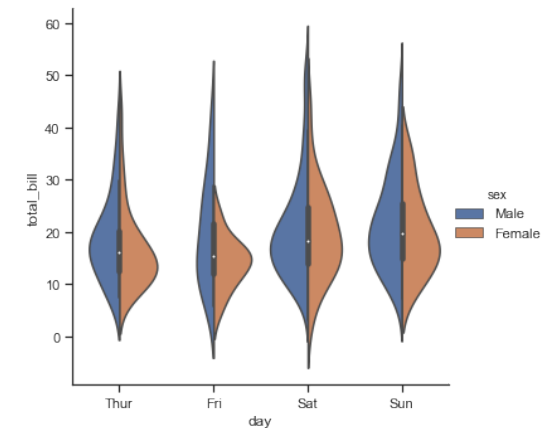
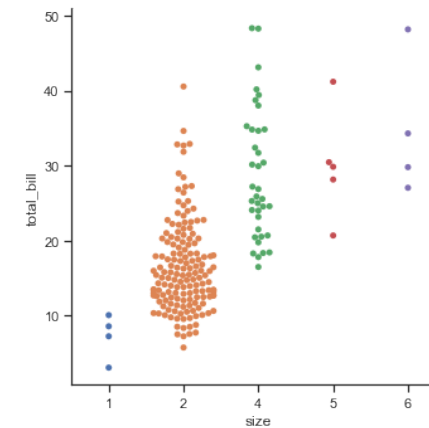
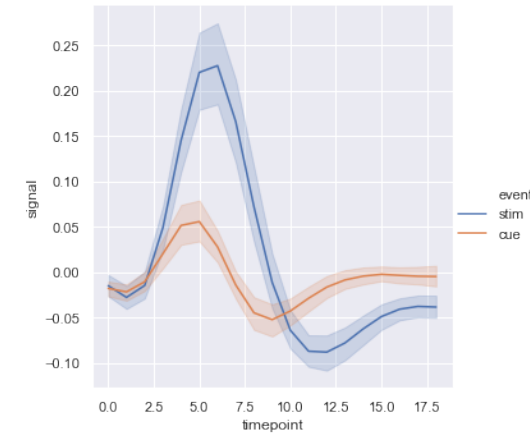
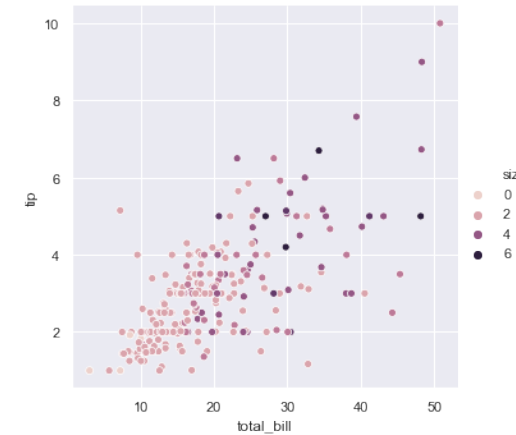
# Plotting in Python

- Matplotlib plotting library
- Inspired by Matlab plotting
- Very fine control of plotting
  - Every line, shape & character
- Extensive capabilities
- Multiple interfaces
  - Pyplot
  - Object oriented



# Seaborn

- Statistical graphics package for Python
- Built on top of Matplotlib
- Closely integrated with Pandas
  - Works with **tidy data**
- Built-in themes & colour palettes
- High-level plotting
- Can still control detail with Matplotlib

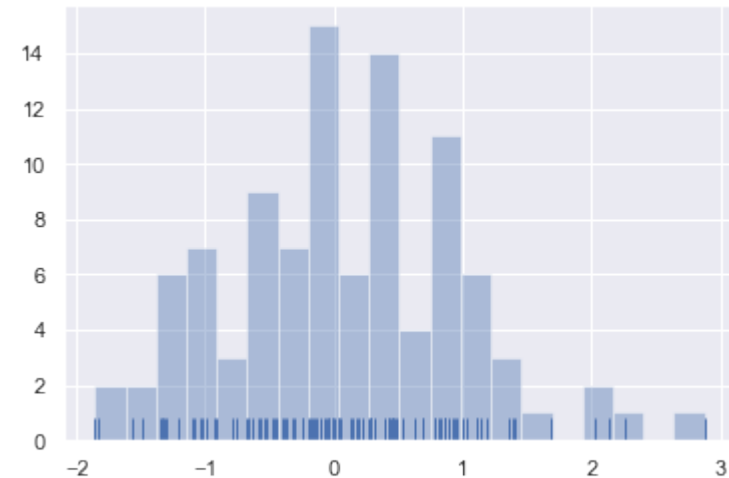
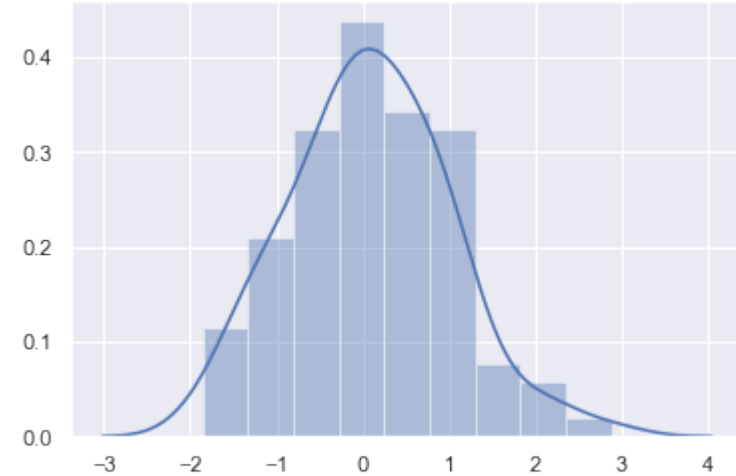


# Visualizing the distribution of a dataset

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

x = np.random.normal(size=100)
sns.distplot(x)

# Select number of bins
# Remove kernel density estimation
# Add rug plot
sns.distplot(x, bins=20, kde=False, rug=True)
```



# Scatterplot example

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(style="darkgrid")

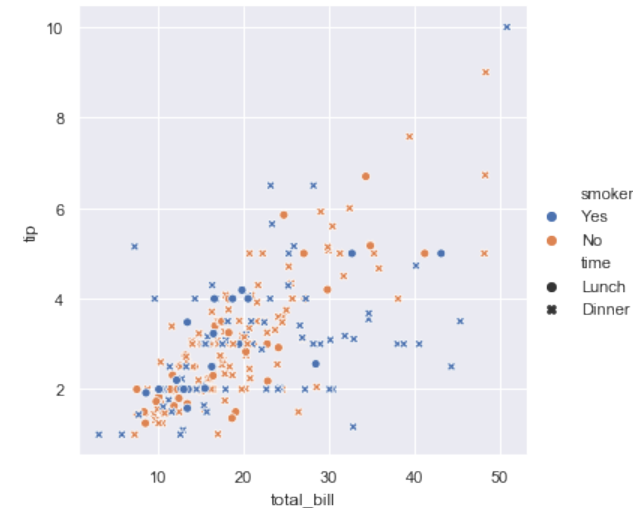
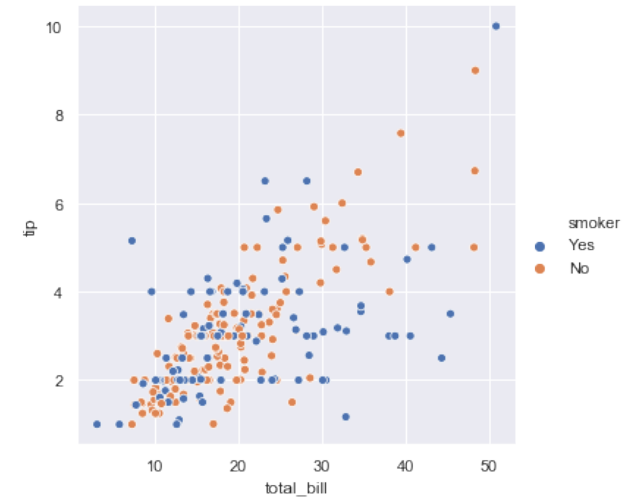
tips = sns.load_dataset("tips")

sns.relplot(x="total_bill", y="tip",
            hue="smoker", data=tips);

sns.relplot(x="total_bill", y="tip",
            hue="smoker", style="time",
            data=tips);
```

show up to three additional variables by modifying the hue, size, and style of the plot elements

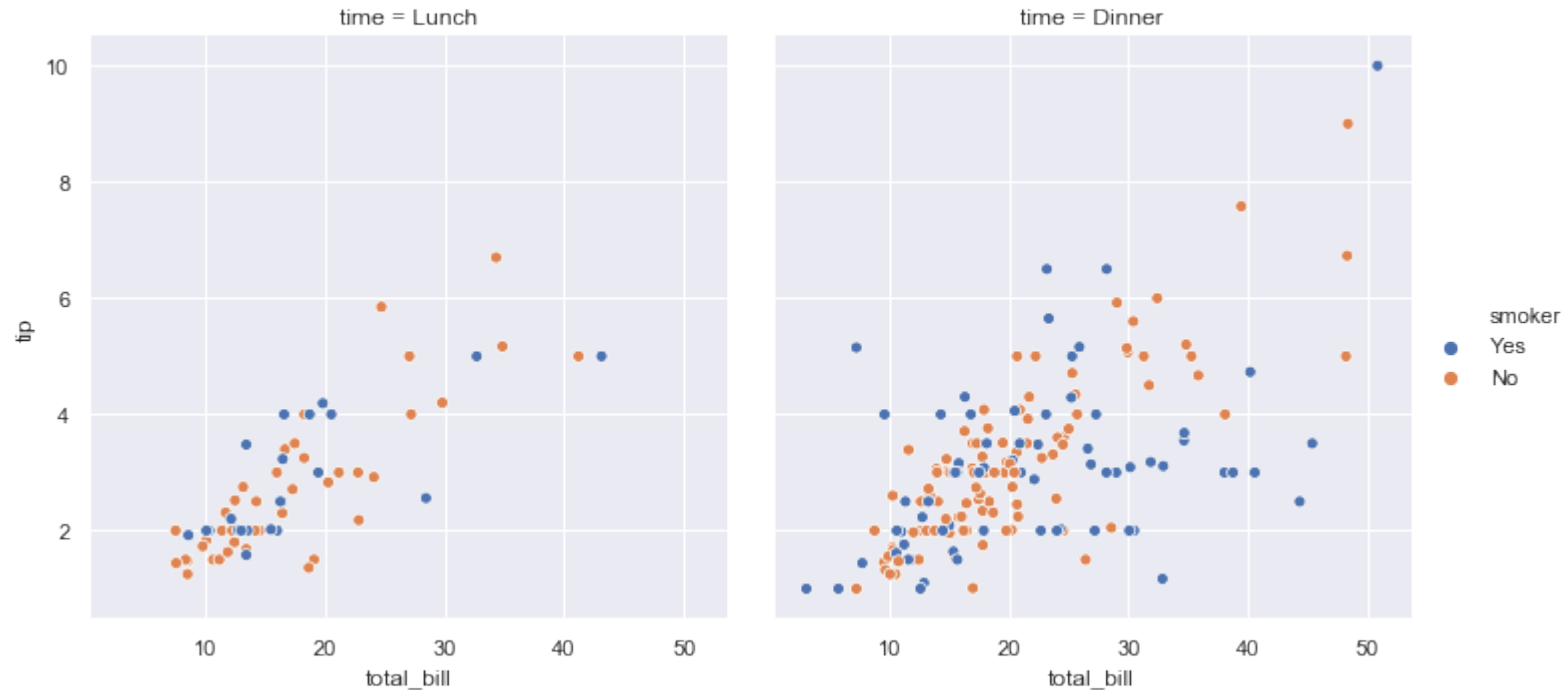
<https://seaborn.pydata.org/tutorial/relational.html>





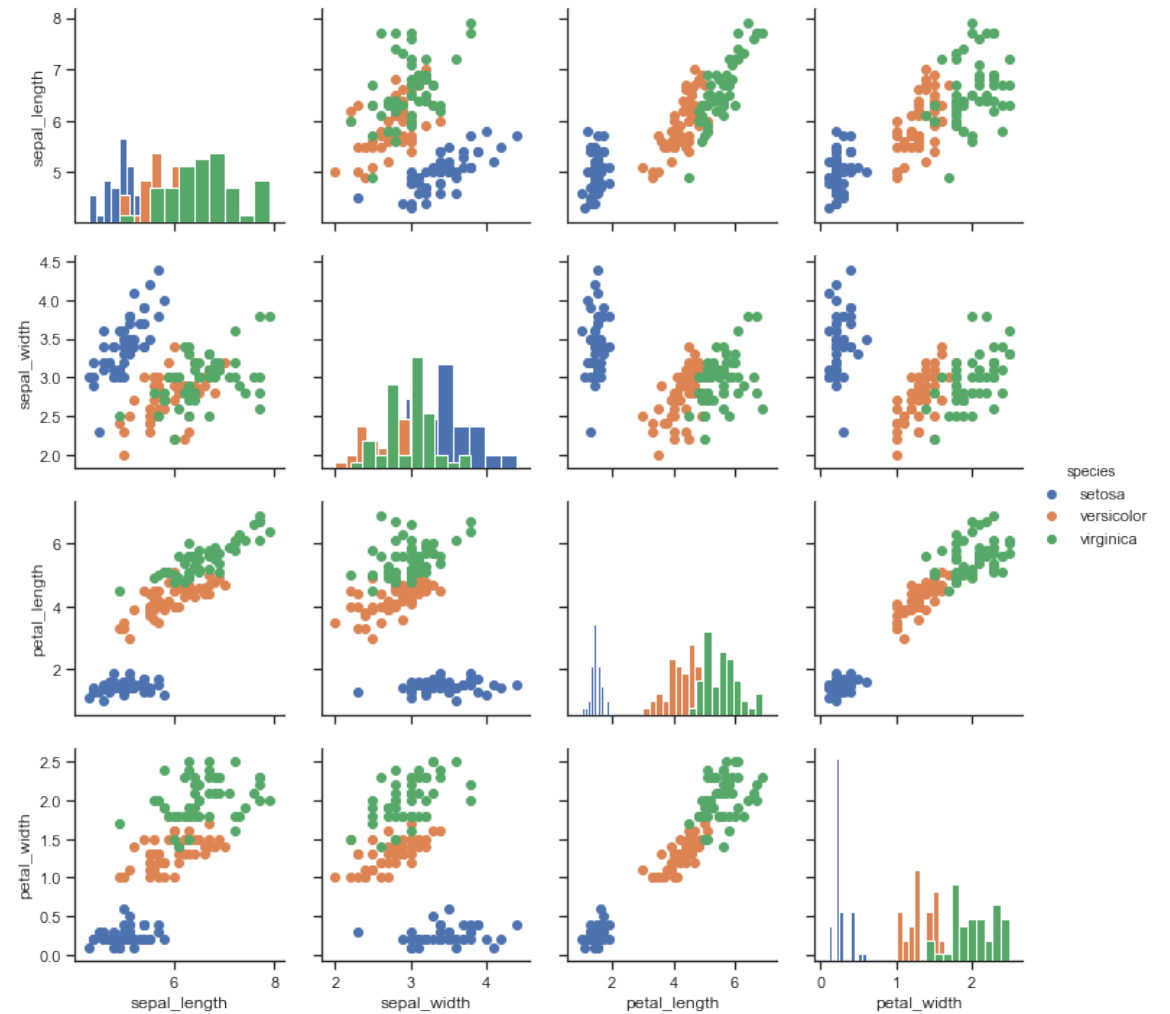
# Show multiple relationships with Facets

```
sns.relplot(x="total_bill", y="tip", hue="smoker", col="time", data=tips);
```



# Pairs plot

```
iris = sns.load_dataset("iris")
g = sns.PairGrid(iris, hue="species")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend();
```



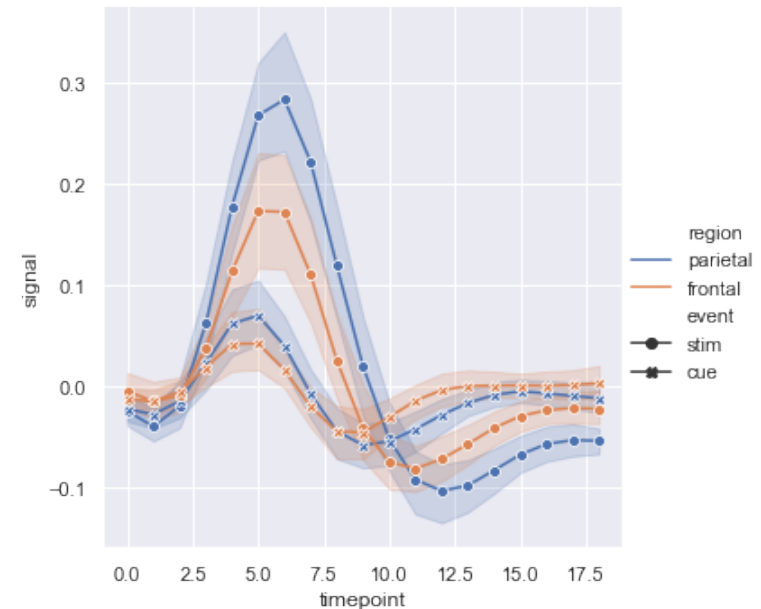
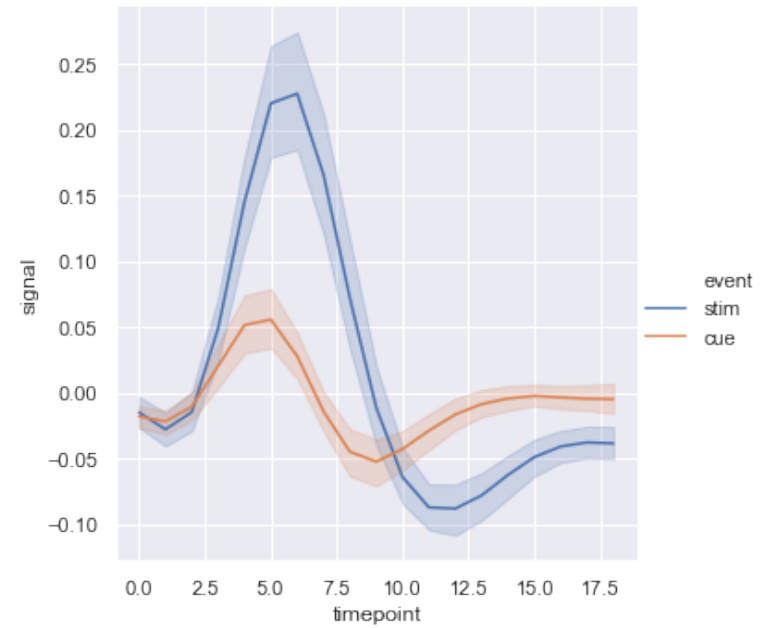
# Lineplot example

```
fmri = sns.load_dataset("fmri")
```

```
sns.relplot(x="timepoint", y="signal",  
hue="event", kind="line", data=fmri)
```

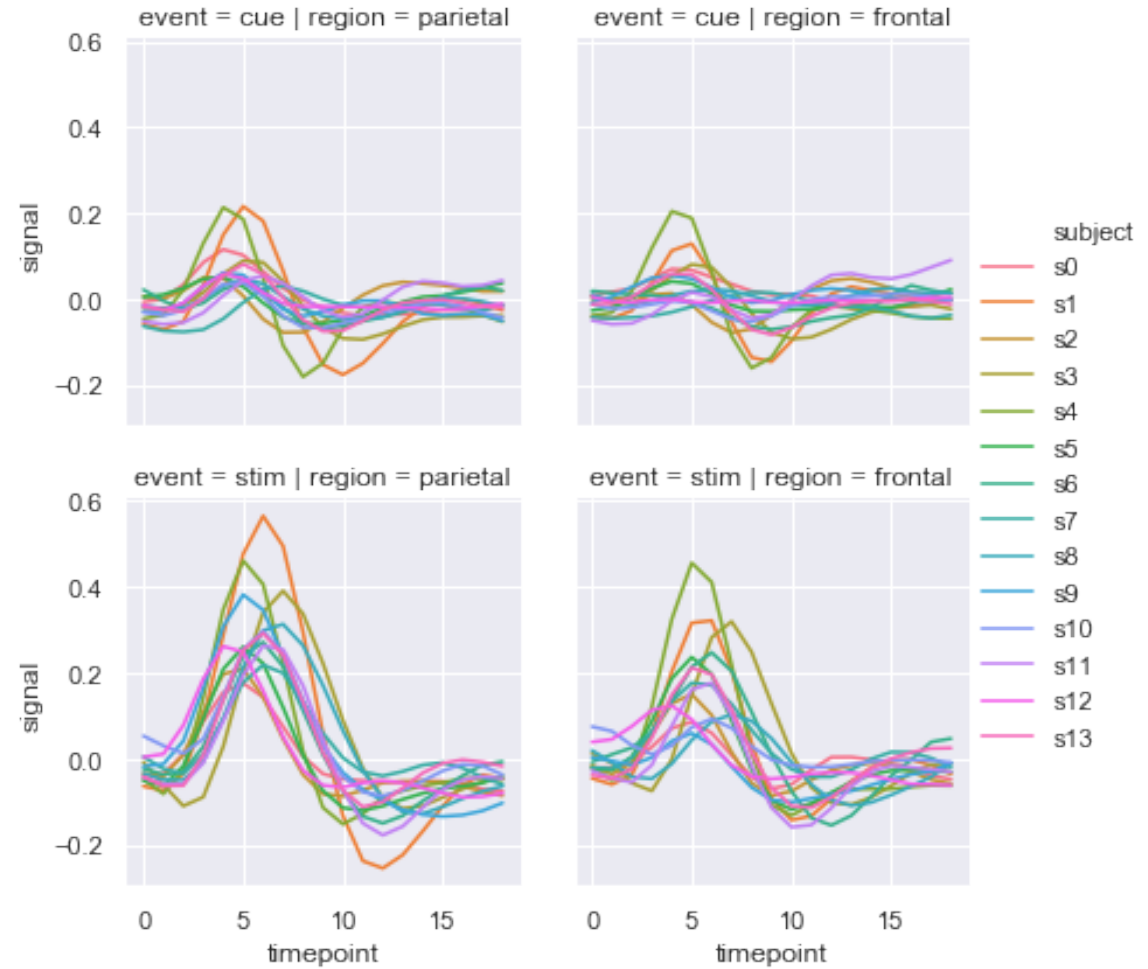
```
sns.relplot(x="timepoint", y="signal",  
hue="region", style="event", dashes=False,  
markers=True, kind="line", data=fmri);
```

Same API as scatterplot – hue, size and style



# Facets

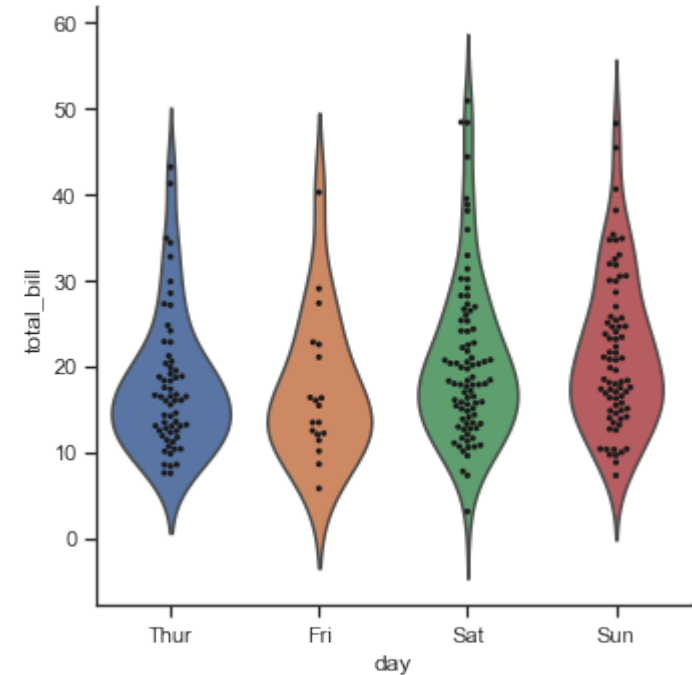
```
sns.relplot(x="timepoint",  
            y="signal",  
            hue="subject",  
            col="region",  
            row="event",  
            height=3,  
            kind="line",  
            estimator=None,  
            data=fmri);
```



# Plotting categorical data

- Combined violin plot and swarm plot

```
g = sns.catplot(x="day", y="total_bill",  
               kind="violin", inner=None, data=tips)  
sns.swarmplot(x="day", y="total_bill",  
             color="k", size=3, data=tips, ax=g.ax)
```



# Data Science Summary

- Pandas provides powerful tools for manipulating matrix data
  - Reorganising / tidying
  - Subsetting / filtering
  - Sorting and merging
  - Grouping & summarising
- Producing meaningful plots requires data to be in the correct format
- High level plotting packages can produce pretty plots quickly
  - Seaborn, pandas
  - ggplot/plotnine
- If you are familiar with Matplotlib you can customise every detail

# Pandas & Plotting Exercises

- Load read count dataset from week 1
- Rename columns and remove unwanted columns
- Filter out genes with no counts
- Plot scatter plot of correlation of replicates (pairs plot)
- Tidy data
- Plot histogram and density plot of read counts across all sample
- Plot a violin plot of read counts per sample
- Identify gene with highest expression in each sample
- Normalise read counts to total reads per sample total reads
- Log transform normalised read counts
- Create new dataframe of 100 genes with highest average variance across conditions
- Plot the of expression of the top 5 most variable genes across all samples (line plot)
- Plot a heatmap for top 100 most variable genes across all samples
- Optional:
  - Add TAD information to dataframe (<http://chromosome.sdsc.edu/mouse/hi-c/download.html>)
  - Calculate average expression for each TAD