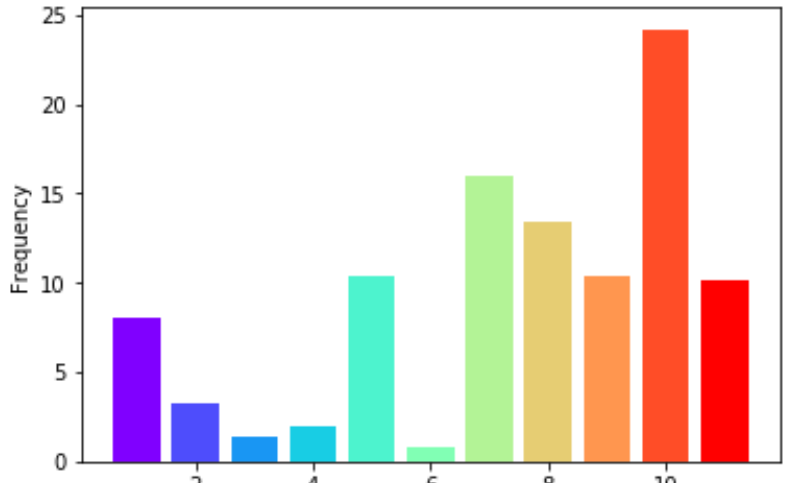
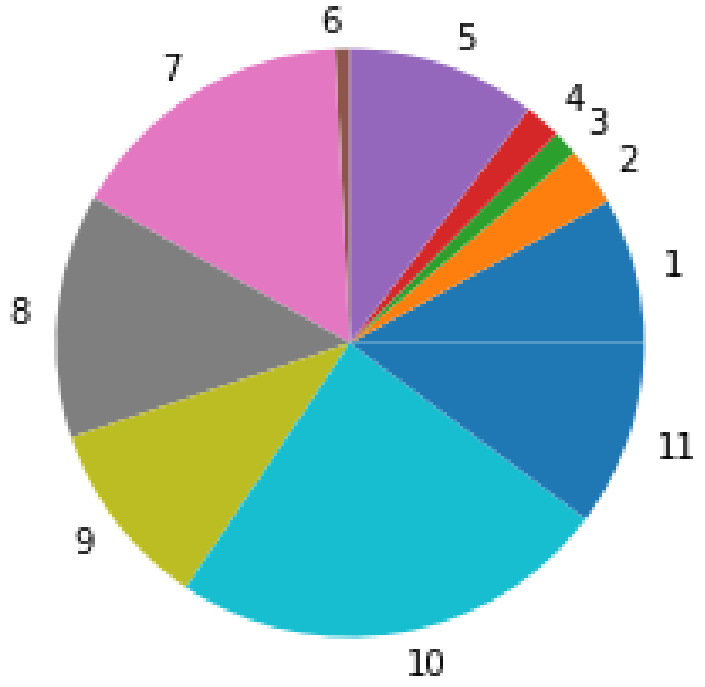
COMP1730/COMP6730
Programming for Scientists

Data science

Data analysis



* Representing tables
* Reading data files
* Working with data: selecting, visualising
* Interpretation



Working example

Table shows how often each model fits best to each test data set. We want to answer: Which model is the best?

Model	test1	test2	test3	test4	test5	test6	test7	test8
1	40	571	353	9	95	41	1428	350
2	16	200	108	2	495	434	88	0
3	7	352	216	9	1201	1897	9	0
4	10	187	202	280	704	215	47	0
5	52	616	204	2	47	17	122	5
6	4	147	146	0	3646	536	0	0
7	80	914	373	4	45	2	161	60
8	67	406	778	1	9	2	3	30
9	52	635	303	1	5	0	5	860
10	121	712	595	0	19	0	1	53
11	51	1914	449	0	29	18	4	50

Data files

* Many data file formats (e.g., excel, csv, json, binary). We'll use the following csv file.

```
Model,test1,test2,test3,test4,test5,test6,test7,test8
1,40,571,353,9,95,41,1428,350
2,16,200,108,2,495,434,88,0
3,7,352,216,9,1201,1897,9,0
4,10,187,202,280,704,215,47,0
5,52,616,204,2,47,17,122,5
6,4,147,146,0,3646,536,0,0
7,80,914,373,4,45,2,161,60
8,67,406,778,1,9,2,3,30
9,52,635,303,1,5,0,5,860
10,121,712,595,0,19,0,1,53
11,51,1914,449,0,29,18,4,50
```

Which data type can we use to represent tables?

Representing tables

* Lists are 1-dimensional, but a list can contain values of any type, including lists.
* A table can be stored as a list of lists, by row, for example:

```
data[i]     # i:th row
data[i][j]  # j:th column of i:th row
```

* Indexing (and slicing) are *operators*
* Indexing (and slicing) associate to the left:

```
data[i][j] == (data[i])[j]
```

Reading data files

* Use a python module that helps with reading the file format:

```
import csv
with open("filename.csv") as csvfile:
    reader = csv.reader(csvfile)
    next(reader) # skip the header
    data = [ row for row in reader ]
```

* More about (reading and writing) files later in the course.

List comprehension

* A *list comprehension* creates a list by evaluating an expression for each value in an iterable collection (e.g., a sequence) using syntax:

```
[ expression for item in a_sequence ]
```

* Example: selecting columns of the table

```
first_col = [ row[0] for row in data ]
last_two_cols = [ row[-2:] for row in data ]
```

* Equivalent to:

```
first_col = []
for row in data:
    first_col.append(row[0])
```

Conditional list comprehension

* Syntax:

```
[ expression for item in a_sequence if boolean_expression ]
```

* Example: select rows where column-1 is $>$ 10

```
sel_rows = [ row for row in data if int(row[1]) > 10 ]
```

* Equivalent to:

```
sel_rows = []
for row in data:
    if int(row[1]) > 10:
        sel_rows.append(row)
```

Sorting

* `sorted(seq)` returns a list with values in `seq` sorted in default order (<).
  - We can sort the rows in a table.
  - Reminder: comparison of sequences is lexicographic.
* `sorted(seq, key=`*`fun`*`)` sorts value `x` by *`fun`*`(x)`.

```
def new_order(row):
    return -row[-1]  # decreasing on last col

sd = sorted(data, key=new_order)
```

Descriptive statistics

* `min(seq)`;
* `max(seq)`;
* mean (`sum(seq) / len(seq)`);
* variance.
* No built-in function for median.

```
def median(seq):
    if len(seq) % 2 == 1:
        return sorted(seq)[len(seq) // 2]
    else:
        return sum(sorted(seq)[(len(seq)//2-1):(len(seq)//2+1)])/2
```

Visualisation

* The purpose of visualisation is to see or show information – not drawing pretty pictures!
* Different kinds of plots show different things:
  - barplot
  - pie-chart
  - histogram or cumulative distribution
  - scatterplot
  - line and area plot
* Use one that best makes the point!
* Choose your dimensions carefully.
* Label axes, lines, etc.

Matplotlib

* Matplotlib is a Python 2D plotting library, which produces publication quality figures.
* "*Matplotlib makes easy things easy and hard things possible*".
* Documentation: `matplotlib.org`

Using `matplotlib`

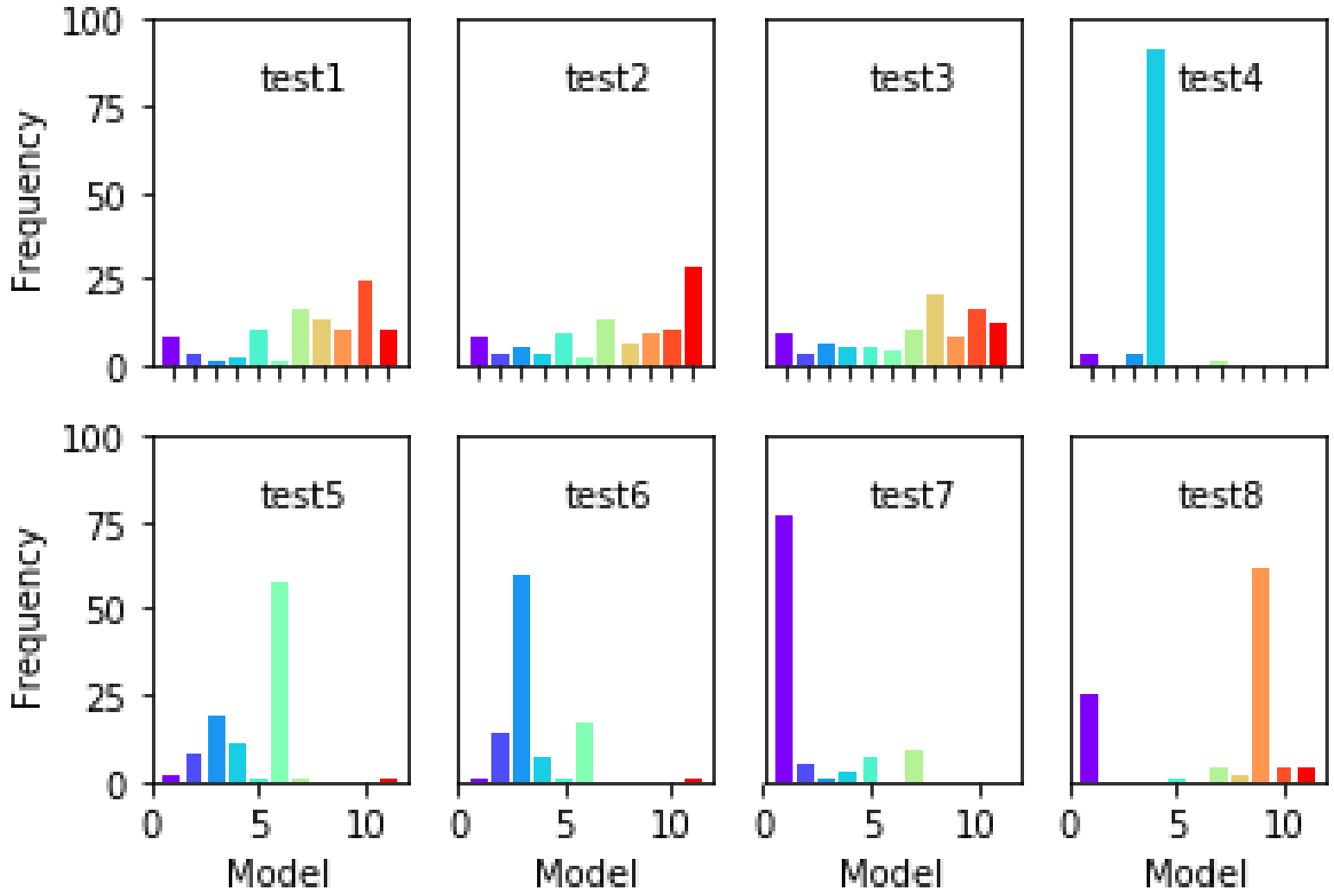
```
import matplotlib.pyplot as plot
first_col = [ int(row[0]) for row in data ]
second_col = [ int(row[1]) for row in data ]

# draw a bar plot
plot.bar(first_col, second_col)
plot.xlabel("Model")
plot.ylabel("Best frequency")
plot.show()

# draw a pie-chart
plot.pie(second_col, labels = first_col, autopct='%1.1f%%')
plot.show()
```

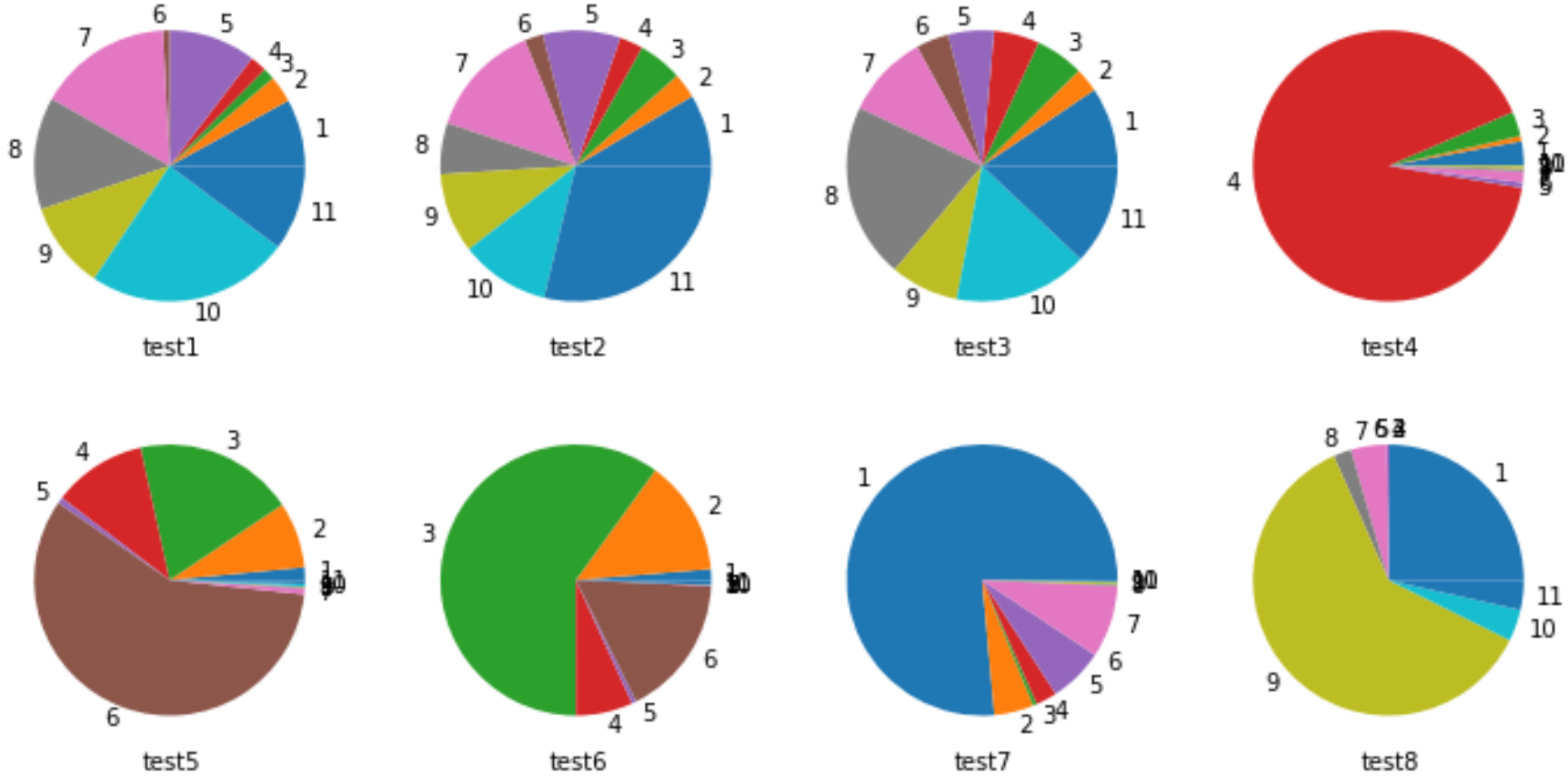
Interpretation

What is this telling us?



Interpretation

What is this telling us?



Interpretation

* To answer question which model is best, Barplots are better than pie-chart in visualising the “goodness of fit” of the models.
* There is no absolute answer: some model is better than other depending on the test sets. None of the models is always the best.
* Some test statistics is needed to measure if a model is “significantly” better than others for a given data set.

Advanced modules

NumPy and SciPy

* The NumPy and SciPy libraries are not part of the python standard library, but often considered essential for scientific / engineering applications.
* The NumPy and SciPy libraries provide
  - an *n*-dimensional array data type (`ndarray`);
  - fast math operations on arrays/matrices;
  - linear algebra, Fourier transform, random number generation, signal processing, optimisation, and statistics functions;
  - plotting (via `matplotlib`).
* Documentation: `numpy.org` and `scipy.org`.

NumPy Arrays

* `numpy.ndarray` is sequence type, and can also represent *n*-dimensional arrays.
  - `len(A)` is the size of the first dimension.
  - Indexing an *n*-d array returns an $(n-1)$-d array.
  - `A.shape` is a sequence of the size in each dimension.
* All values in an array must be of the same type.
* Element-wise operators, functions on arrays.
* Read/write functions for some file formats.

Generalised indexing

* If `A` is a 2-d array,
  - `A[i,j]` is element at `i`, `j` (like `A[i][j]`).
  - `A[i,:]` is row `i` (same as `A[i]`).
  - `A[:,j]` is column `j`.
  - `:` can be `start:end`.
* If `L` is an array of `bool` of the same size as `A`, `A[L]` returns an array with the elemnts of `A` where `L` is `True` (does not preserve shape).
* If `I` is an array of integers, `A[I]` returns an array with the elemnts of `A` at indices `I` (does not preserve shape).

Pandas

* Library for (tabular) data analysis.
  - Special types for 1-d (`Series`) and 2-d (`DataFrame`) data.
  - General indexing, selection, alignment, grouping, aggregation.
* Documentation: `pandas.pydata.org`
* *Beware:* Pandas data types do not behave as you expect.

Take home message

* Python is powerful in data analysis.
* Think carefully about visualisation: How can people quickly interpret the results?
* We have only scratched the surface of Matplotlib. Extensive documentation: `https://matplotlib.org` or just **google it**!