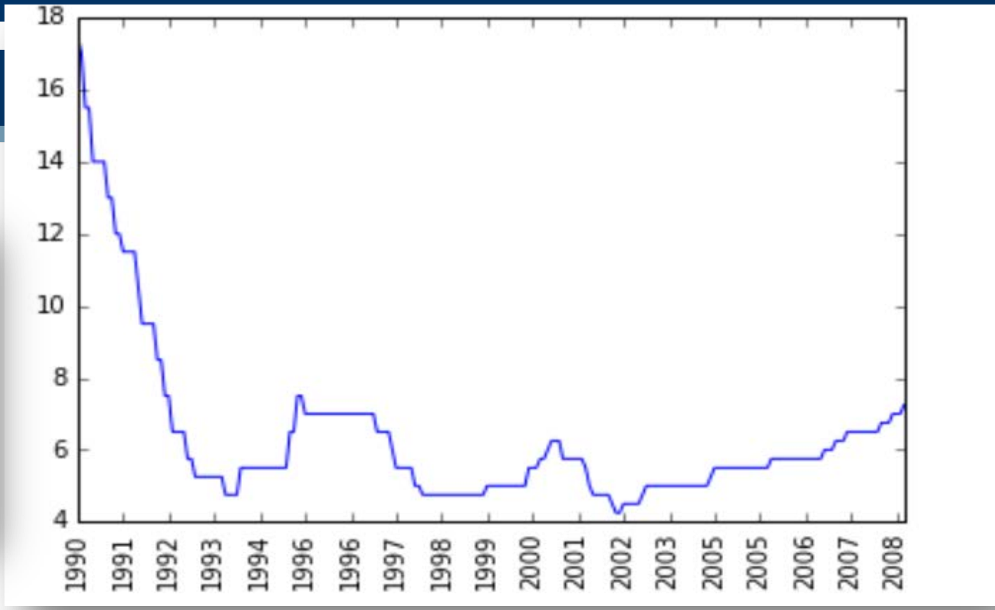
Time series

- Historical information
  - Deduce trends
  - Predict future
- See the example of “monthly interest rate”

Monthly histor

	A
1	1/23/1990,17.50
2	2/15/1990,17.00
3	4/4/1990,15.50
4	8/2/1990,14.00
5	10/15/1990,13.00
6	12/18/1990,12.00

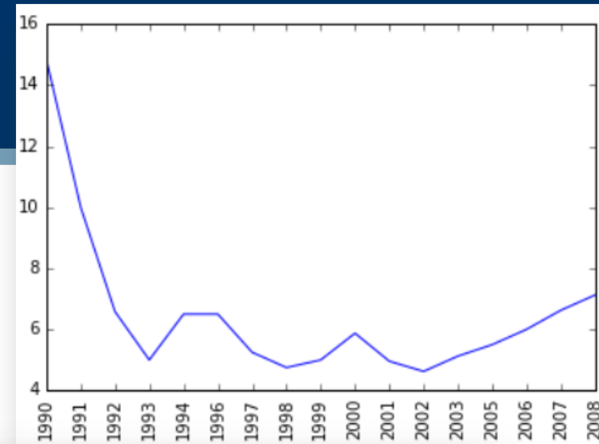



```
hist_rates = [] # historical rates
years = [] # ticks text for year
y_loc = [] # ticks location
prev_idx = 0 # index to previous month

for arow in data:
    rate = float(arow[1]) # interest rate
    date = arow[0].split('/') # data of interest rate change
    month = int(date[0])-1 # map jan-dec to 0-11
    year = int(date[2])-1990 # map 1990-2008 to 0-18
    idx = year*12 + month
    for i in arange(prev_idx, idx+1): # filling earlier months
        hist_rates.append( rate ) # build hist_rates
        if i%12 == 0: # if january build years ticks
            y_loc.append( i )
            years.append( str(1990+year) )
    prev_idx = idx+1
```

- Bas on the dataset, plot a chart of annual average of interest rate, instead of monthly.
  – Calculate the average of each year

THE UNIVERSITY OF MELBOURNE

	A
1	1/23/1990,17.50
2	2/15/1990,17.00
3	4/4/1990,15.50
4	8/2/1990,14.00
5	10/15/1990,13.00
6	12/18/1990,12.00

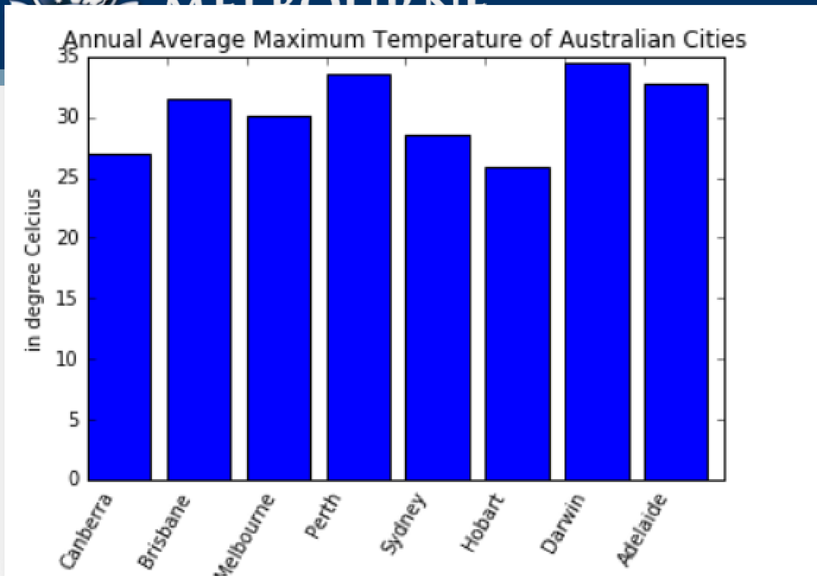



```
1990: [17.5, 17.0, 15.5, 14.0, 13.0, 12.0], 1991: [11.5, 10.5, 9.5, 8.5],
```

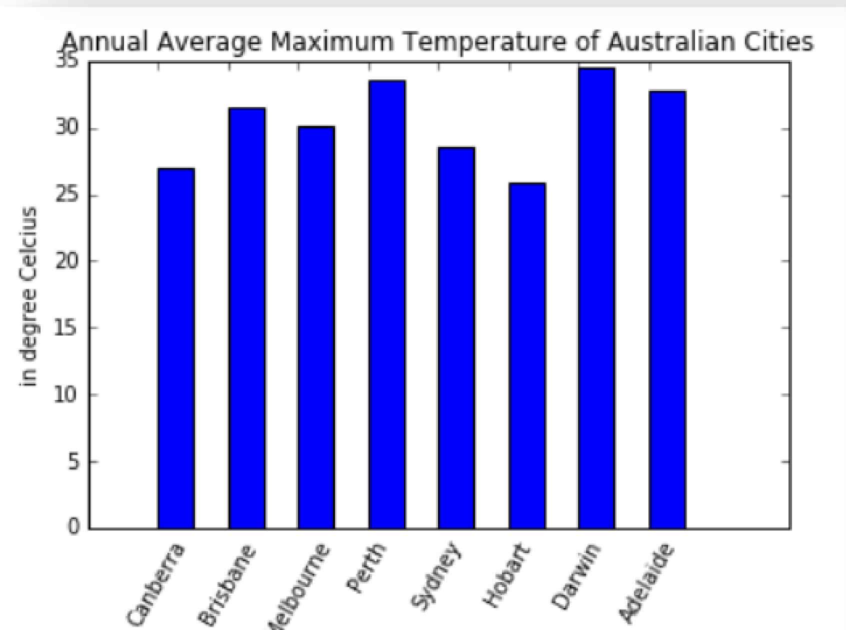
```
# data from Jan 1990 to Mar 2008
source = open("data/au_interest_rates.csv","r")
data = list(csv.reader(source))
source.close()

hist_rates = defaultdict(list) # historical rates
years = [] # ticks text for year
y_loc = [] # ticks location
prev_idx = 0 # index to previous month

for row in data:
    rate = float(row[1]) # interest rate
    date = row[0].split('/') # data of interest rate change
    year = int(date[2])
    hist_rates[year].append(rate)
#print hist_rates
print hist_rates.items()
for year, values in hist_rates.items():  #match the year and iterate the value
    hist_rates[year] = sum(values)/len(values)

plt.xticks(arange(len(hist_rates)),hist_rates.keys(),rotation='vertical' )
plt.plot(hist_rates.values())
```

RTS HERE

- See the average maximum temperature in major Australian cities in 2008
- Using bar chart to plot



A
city/month,Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec
Melbourne,41.2,35.5,37.4,29.3,23.9,16.8,18.2,25.7,22.3,33.5,36.9,41.1
Brisbane,31.3,40.2,37.9,29,30,26.7,26.7,28.8,31.2,34.1,31.1,31.2
Darwin,34,34,33.2,34.5,34.8,33.9,32,34.3,36.1,35.4,37,35.5
Perth,41.9,41.5,42.4,36,26.9,24.5,23.8,24.3,27.6,30.7,39.8,44.2
Adelaide,42.1,38.1,39.7,33.5,26.3,16.5,21.4,30.4,30.2,34.9,37.1,42.2
Canberra,35.8,29.6,35.1,26.5,22.4,15.3,15.7,21.9,22.1,30.8,33.4,35
Hobart,35.5,34.1,30.7,26,20.9,15.1,17.5,21.7,20.9,24.2,30.1,33.4
Sydney,30.6,29,35.1,27.1,28.6,20.7,23.4,27.7,28.6,34.8,26.4,30.2




```
import matplotlib
import matplotlib.pyplot as plt
import csv

data = list(csv.reader(open('data/max_temp.csv')))
cities = {}

for row in data[1:]:
    cities[row[0]] = sum([float(col) for col in row[1:]]) / 12

city_num = len(cities)
plt.bar(arange(city_num), cities.values())
plt.xticks(arange(city_num), cities.keys(), rotation=60)
plt.ylabel("in degree Celcius")
plt.grid(False)
plt.title("Annual Average Maximum Temperature of Australian Cities")
```

Tools of Analysis

INFO20002: Foundations of Informatics Week 8

Outcomes

- 1. Recap previous knowledge (Web application, Data analysis and HTML/CSS)

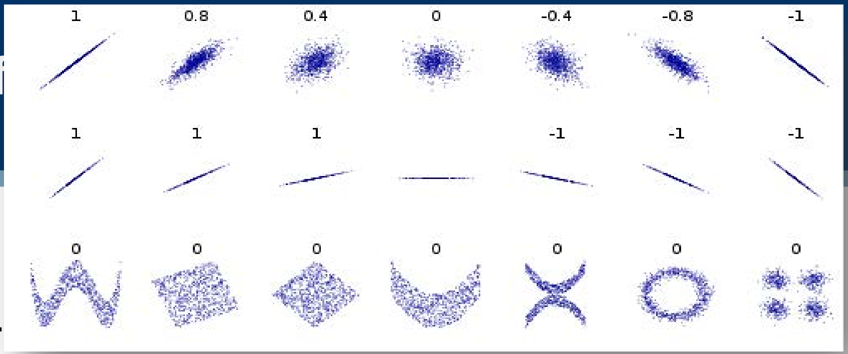
- 2. Highlight some regularly-used analytical techniques
  – Pearson
  – Transform numerical into categorical
  – Classifier (op)
- 3. Phase-3 project
  – Deliverables: (week 10)
    - The final team members
    - Final choice of domain and data-sets

Spearman's Correlation (Rank Correlation)

- Calculates similarities between ranks
- Applied in non-numerical values

Team	*rank_i*	*rank_j*	*d*	d^2
Melbourne	1	6	-5	25
Monash	2	5	-3	9
Sydney	3	4	-1	1
New South Wales	4	3	1	1
Adelaide	5	2	3	9
Perth	6	1	5	25
n	6			
$\sum d_i^2$	70			
$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$	-1.000			

Correlation coeff



- Pearson's correlation coefficient
  - The degree of relationship between two continuous variables
  - Pearson correlation [-1,1]
    - =-1/1 –maximum correlation in positive or negative direction
      - As A increases, so does B
      - As A increases, B decreases
    - = 0 – no correlation
    - (-1,0) ∪ (0,1) – correlated to different extents
      - "the closer to zero, the less evidence to show the relationship"

Part – A Linear correlation coefficient

- Explore correlation across various attributes within the body fat data set. Build a 15 x 15 HTML table that display the Pearson's correlation coefficients for all possible pairs of attributes.
  - Since Pearson's *r* can range from -1 to +1, colour the cells of your table with shades from red to blue (See the example output).
  - [-1,1] color mapping

Part – A Linear correlation coefficient

- Pearson’s correlation coefficient
  - The degree of relationship between two continuous variables (numerical)
  - Using python library
    pearsonr (array a, array b)

    >>> from scipy.stats.stats import pearsonr

    >>> pearsonr([1,2,4],[0.6,0.777,0.91]) [0]

    - Returns (**pearson’s coefficient**, 2-tailed p-value)

P-value – measure the significance of study

In this case, “hypothesis” is two sets are unrelated

P-value is the possibility of H establish - [0,1]

The p-value roughly indicates the **probability (at least)** of an **uncorrelated** system producing datasets that have a **Pearson correlation r**

For instance if there are 100 pairs of data whose r = 0.2 then the p-value is 0.01.

- That means there is a 1% chance to have the unrelated data sets with a r (r=0.2)
- 0.05 or 0.01 is typically used as the significance level
  - > 0.05 the relation between A and B is insignificant; Accept the H
  - < 0.05 the relation between A and B is significant; Reject the H

- http://www.statsdirect.com/help/default.htm#basics/p_values.htm

HTML generator

- CSV processing
- HTTP string concatenation – table and color
- Web application
- Calculate and map the Pearson values
    - `from scipy.stats.stats import pearsonr`
    - `>>> pearsonr([1,2,4],[0.6,0.777,0.91])[0]`

"Recipe" – hello.py

```
#import the Flask class
from flask import Flask
# create an instance of "Flask"
app = Flask(__name__, static_folder='.', static_url_path='')
# route() decorator to tell app what URL can trigger functions.
@app.route("/")
#define a function which is to display the message on the browser
def root():
  return 'Hello World!'
# using run() to specify how to run the app
if __name__ == "__main__":
  app.run (debug=True, host='127.0.0.1', port=80)
```

THE UNIVERSITY OF MELBOURNE

One more cell…

	density	siri
density	1.000	−0.977
siri	−0.977	1.000
age	−0.282	0.291

```
def make_pearsons():
  html = '''<html><body>
  <style>

  td {
    font-family: 'Lucida Sans Unicode', sans-serif;
    padding: 5px;
    text-align: right;
  }

  </style>
  <table><tr><td></td>'''
  #this cell is invisible but it is necessary
```

Color values - RGB

RGB(0-225, 0-225, 0-225)
HSL(0-360, 0-1,0-1)
The specification, “red and blue”, means we need to **map** the Pearson coefficient in RGB

Blue (0,0,225) --- r = 1 #0000FF
Red (225,0,0) ----r = -1 #FF0000
X ([0-225], 0, [0-225]) --- [-1,1] ---# [FF-00]00[00-FF]

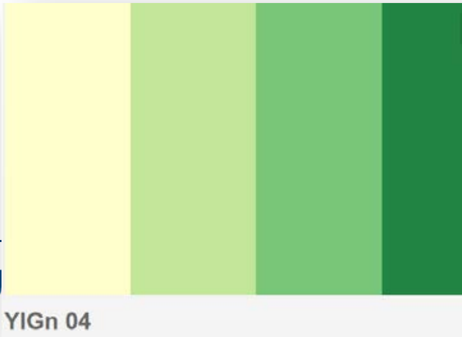
```
def color(pear):
    # 1 blue 0000ff
    # -1 red ff0000
    '''blue: #0000FF -> rgb(0,0,255) red: #FF0000 -> rgb(255,0,0)'''

    n = (pear + 1)/2 # map it to the range 0 - 1
    red = '%02x' % int((1-n)*255)   # red will control the "Red"
    blue = '%02x' % int(n*255)      # blue will control the "Blue"
    return '#' + red + '00' + blue # return the unique color value to each pearson
''' format specifier - print the int in a hexadecimal value with at least two digits'''
```

0-0.5: red decrease
0.5-1: blue increase

Part-B：Transform numerical data points to categories

- Produce an HTML representation (not web application) that split the data into 4 groups of data. (output sample)
  - Try to open/use online dataset - **COLORBREWER 2.0**

```
>>> import json
>>> import urllib
>>> palette = json.load(urllib.urlopen('http://colorbrewer2.org/export/colorbrewer.json'))
>>> palette['YlGn']['4']
```



- The diverging palettes are
  BrBG PiYG PRGn PuOr RdBu RdGy RdYlBu RdYlGn YlGn Spectral

```
var colorbrewer = {YlGn: {
3: ["#f7fcb9","#addd8e","#31a354"],
4: ["#ffffcc","#c2e699","#78c679","#238443"],
5: ["#ffffcc","#c2e699","#78c679","#31a354","#006837"],
6: ["#ffffcc","#d9f0a3","#addd8e","#78c679","#31a354","#006837"],
7: ["#ffffcc","#d9f0a3","#addd8e","#78c679","#41ab5d","#238443","#005a32"],
8: ["#ffffe5","#f7fcb9","#d9f0a3","#addd8e","#78c679","#41ab5d","#238443","#0
9: ["#ffffe5","#f7fcb9","#d9f0a3","#addd8e","#78c679","#41ab5d","#238443","#0
```

Part B – Transform numerical data points to categories

- There are two methods for your reference:
  - Equal intervals – data (from the minimum to maximum) is split into 4 equal intervals
    - Min and max
    - Other 3 bounds ?
  - Quartile intervals– use the "5 numbers" as the boundaries of 4 classes
    - Min-q1; q1-mean; mean-q3; q3-max

```python
def equal_interval(xs, k):
  max_x = max(xs)
  print max_x #99
  min_x = min(xs)
  print min_x #-100
  dx = (max_x - min_x)/k
  print dx # 49.75
  bounds = reduce(lambda b, i: b + [b[i] + dx], range(k), [min_x])  # 
  '''it is always used with reduce, map and filter'''
  '''[0,1,2,3]'''
  '''[min]+[min+dx]  --->[min,min+dx]
  [min,min+dx]+[min+dx+dx]----->[min,min+dx,min+2dx]
  [min,min+dx,min+2dx]+[min+dx+dx+dx]---> [min,min+dx,min+2dx,min+3dx]
  ...
  min, min+dx,min+2dx,min+3dx, max'''
  print bounds
  bounds[-1] = max_x
  return bounds
```

lambda - anonymous function - no need for function name

```python
>>> foo = [2, 18, 9, 22, 17, 24, 8, 12, 27]
>>>
>>> print filter(lambda x: x % 3 == 0, foo)
[18, 9, 24, 12, 27]
>>>
>>> print map(lambda x: x * 2 + 10, foo)
[14, 46, 28, 54, 44, 58, 26, 34, 64]
>>>
>>> print reduce(lambda x, y: x + y, foo)
139
```

```python
def get_class(bounds, x):
  for i in range(0, len(bounds) - 1): #[0,1,2,3]
    if x <= bounds[i + 1]:  # it has to be greater than the min so
     return i;  # [0--group0--1--group1--2--group2--3--group3--4]
  return -1
```

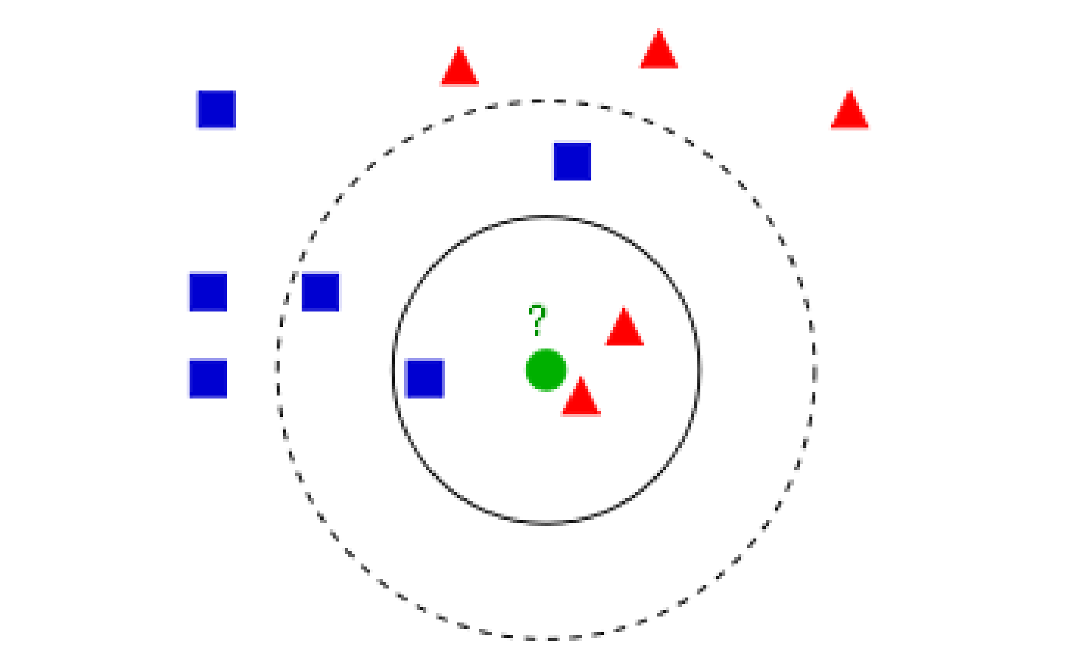
```python
def classify(bounds, xs):
  tally = defaultdict(int)
  groups = defaultdict(list)
  cs = []
  for x in xs:
    c = get_class(bounds, x)
    tally[c] += # counter of
    groups[c] += [x] # get li
  return tally, groups
```

Classifier

Foundation of Informatics - Week 8

Classifier

- A classifier is a model that contains some **parameters**
- Build the model to predict which category the next data-point is going to fall into…
  - You have a "training data set" for a classifier to learn from (i.e. use the data to find the parameters).



KNN - have to read the specification

>>> from sklearn import neighbours

\# sklearn.neighbors provides functionality for unsupervised and supervised neighbors-based learning methods.

>>> knn=neighbors.KNeighborsClassifier()

>>>knn.fit (features, classes)

\# features - some health indicators

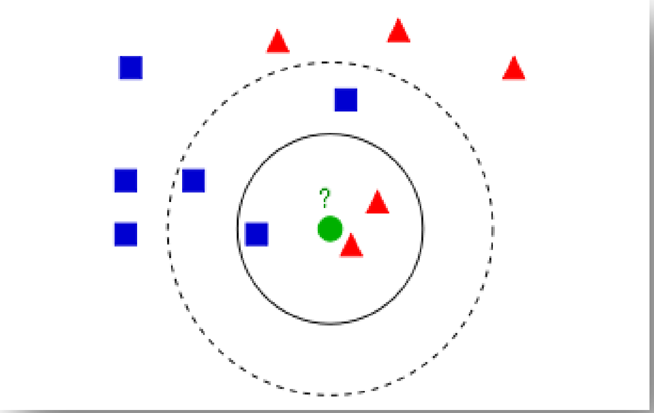
\# classes – binary-valued classes (0 and 1)

\# Fit the model using X as training data and y as target

>>>knn.predict(features)

\#find out the classes of a testing dataset

\# (in this example, we reuse the training data as the testing da



>>>knn.score(features,classes)

\# get the mean accuracy by validation

Another case:

knn.fit (x_training, y_training) …

knn.score(x_test,y_test) …

K nearest neighbors

- Distance computation – **euclidean distance**

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

- Find k nearest neighbors
- Make classification – according to the major neighbors

- Reference: http://people.revoledu.com/kardi/tutorial/KNN/index.html

confusion matrix

- Assess the quality of a model using a confusion matrix (contingency table):

Actual / Predicted	**Positive (1)**	**Negative (0)**
True (1)	TP	TN
False (0)	FP	FN

- TP = true positive
- TN = true negative
- FN = false negative
- FP = false positive

Disease example

- Imagine a study evaluating a new test that screens people for a disease.
  - The test outcome can be positive (classifying the person as having the disease) or negative (classifying the person as not having the disease). (Positive or Negative)

  - The test results for each subject may or may not match the subject's actual status. (True or False)

    - True positive: Sick people identified having disease
    - True negative: Healthy people identified having no disease

    - False positive: Healthy people identified having disease (Type I error)
    - False negative: Sick people identified having no disease (Type II error)

Sensitivity and Specificity

- Sensitivity (**true positive rate**)
- $$\text{Sensitivity} = \frac{TP}{TP + FN}$$

  the test's ability to correctly detect patients who do have the condition. 0.652

- Specificity **(true negative rate)**

  $$\text{Specificity} = \frac{TN}{FP + TN}$$

  the test's ability to correctly detect patients without a condition. 0.884
- This test with high specificity is useful for ruling out disease.

THE UNIVERSITY OF
MELBOURNE