Data Science

DAI-101 Spring 2025-26

Dr. Devesh Bhimsaria

Office: F9, Old Building

Department of Biosciences and Bioengineering

Indian Institute of Technology–Roorkee

devesh.bhimsaria@bt.iitr.ac.in

About the Course

- Contact Hours (Hrs. per week): L:3 T:1 P:0
- Total contact hours: 42
- Credits: 4
- Prerequisite: None.
- Taught by: Dr. Devesh Bhimsaria & Dr. Deepak Sharma
- Ran in 5 Batches

Course Outline

- Introduction to Data Science: Latest and greatest in data science
- Python programming
- Data Analysis Foundation: Types of data (data matrix, numeric, categorical datasets), data preparation: data cleaning, data reduction and transformation
- Exploratory Data Analysis and Visualization: Univariate and bivariate analysis, data visualization
- Statistical Analysis: Confidence Intervals, Hypothesis Testing, p-values, Bias and Variance trade-off
- Machine Learning: introduction to supervised and unsupervised methods, model training, overfitting and underfitting, bias and variance, introduction to supervised methods: regression and classification (Linear regression, logistic, decision trees, SVM), Clustering, K-means, PCA
- Deep learning and Big Data: Gradient Descent, Neural nets, Convolutional Neural Networks, Big Data technologies (MapReduce, HDFS)

Evaluation

- Mid-term exam
- End-term exam
- Assignments
- Attendance
- If there is any modification, you'll be informed in advance

Rules and other points

- Maintain class decorum.
- Timely Submission of assignments.
- For any help related to the course or otherwise – a) You can email me, b) ask me during lecture/tutorial.
- If urgent, CR may call or message.

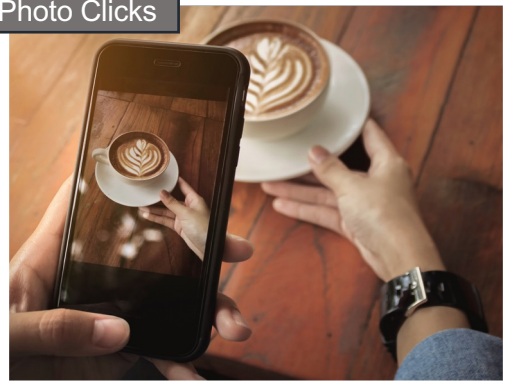
The Age of Data

Our every action generates data

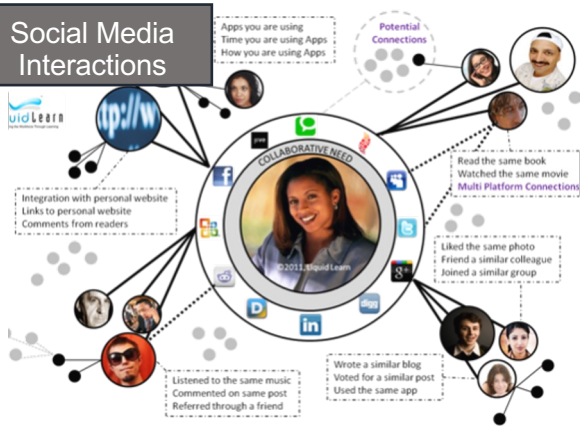
Experiments

Smart Wearables

Photo Clicks



Social Media Interactions



Google Searches

Stock Market Shares

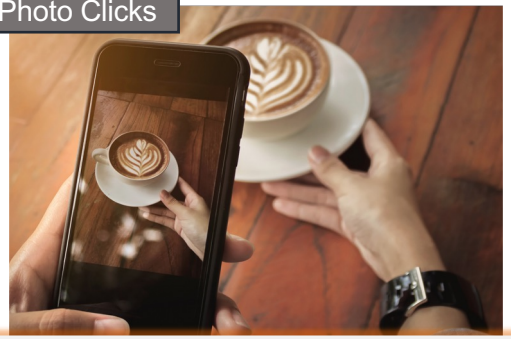
The Age of Data

Our every action generates data

Experiments

Smart Wearables

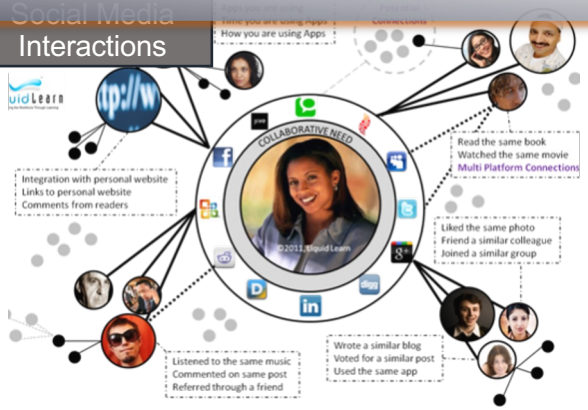
Photo Clicks



328,770,000 TB of data generated per day

How do we make sense of it?

Social Media Interactions



Google Searches

Stock Market Shares

What is Data Science?

- Definition: An interdisciplinary field that uses scientific methods, processes, algorithms, and systems to extract knowledge and insights from structured and unstructured data.
- Key Components:
  - Data Collection
  - Data Cleaning and Preparation
  - Data Analysis and Visualization
  - Machine Learning/AI

Importance

- Real-world applications:
  - Healthcare: Predicting diseases
  - Business: Customer segmentation
  - Finance: Fraud detection
  - Social Media: Recommendation systems
- Industry growth and demand for data professionals

Skills Required for DS analysis

- Technical Skills:
  - Programming: Python, R, SQL
  - Data Manipulation: Pandas, NumPy
  - Visualization: Matplotlib, Seaborn
  - Machine Learning: Scikit-learn, TensorFlow

Challenges in Data Science

- Data Quality Issues: Missing, noisy, or inconsistent data
- Data Privacy & Ethics: Ensuring compliance with regulations
- Model Interpretability: Explaining complex models
- Scalability: Handling large datasets

Thank You

- All my slides/notes excluding third party material are licensed by various authors including myself under https://creativecommons.org/licenses/by-nc/4.0/

Data Science

DAI-101 Spring 2025-26

Dr. Devesh Bhimsaria

Office: F9, Old Building

Department of Biosciences and Bioengineering

Indian Institute of Technology–Roorkee

devesh.bhimsaria@bt.iitr.ac.in

Python review

What is the Python Shell?

- The **Python Shell** is an interactive environment where you can directly execute Python code and see immediate results. It is also known as the **Python interactive interpreter**.

Python Installations

- Conda: Anaconda/Miniconda (Pycharm, etc.)
- Jupyter notebook
- Python IDLE (Integrated Development and Learning Environment)
- VSCode etc.
- Note there is python 2 and python 3 versions

Interactive python

- Terminal- Unix/Linux & Mac OS
  - Type “python” enter- you’ll get prompt with >>>
- Python IDLE
- Jupyter notebook: Web based interactive platform

Running python code

- Terminal- Unix/Linux & Mac OS
  - python filename
  - chmod 0755 filename
  - ./filename
- VS Code and others

Basic Data Types

Boolean type

```
>>> a = True
>>> a
True
>>> type(a)
<type 'bool'>
>>> b = bool(0)
>>> b
False
```

Numerical Types

- Integer – int

```
>>> x = 5
>>> x == 5
True
>>> y = x
```

- Floating point – float

```
>>> a = 0.1
>>> b = 0.2
>>> c = a + b
>>> a
0.1
>>> c
0.30000000000000004 # Binary Floating-Point Representation & Accumulated Rounding Errors. use the decimal module
>>> float(1)
1.0
>>> c == 0.3
False
```

`is` and `is not`

- `is` and `is not` are used to compare

```
>>> y = x
>>> x is y
True
>>> y += 1
>>> x is y
False
>>> x is not y
True
```

Comparison Operators

same values	`==`
not same values	`!=`
same object	`is`
not same object	`is not`
less than	`<`
less than or equal to	`<=`
greater than	`>`
greater than or equal to	`>=`

- All comparison operations yield a Boolean value
- Use `is`/`is not` with `None`, `True`, and `False`
- Can chain inequalities: `1 < x <= 4`

None

- `None` *is a special identity*
- *Like* `NULL` *in* `C/C++`

```
>>> p = None
>>> p is None
True
>>> x is not None
True
>>> print(type(None))
<class 'NoneType'>
```

None

None is treated as False in a boolean context.

The type of None is NoneType.

Like other singleton objects in Python (e.g., True, False), None is immutable. Its value cannot be changed.

```
print(None == False) # Output: False

print(None == 0) # Output: False

print(None == "") # Output: False
```

Python Strings – str

```
>>> s = "abc"
>>> s
'abc'
>>> type(s)
<type 'str'>
>>> str(1)
'1'
>>> len(s)
3
>>> s2 = 'ab'
>>> s2 += 'c'
>>> s2
'abc'
>>> s == s2
True
```

Integers & Strings are Immutable

This means that once they are created, their value cannot be changed. Instead, any operation that seems to modify them actually creates a new object.

Lists, dictionaries, sets, etc. are mutable.

```
>>> a = 'abc'
>>> id(a)
6776912
>>> a += 'd'
>>> id(a)
7775840
>>> i = 5
>>> j = i
>>> i += 5
>>> i is j
False
```

Lists

- Used to create arrays, stacks, FIFOs. Mutable.

```
>>> l = [ 1, 2, 3, 4 ]
>>> id(l)
4352115136
>>> l += [ 5 ]
>>> id(l)
4352115136
>>> l.append( 6 )
>>> l
[1, 2, 3, 4, 5, 6]
>>> l.pop()
6
>>> l.pop(0)
1
>>> l
[2, 3, 4, 5]
>>> l[1:3] # Start from first index to second index-1
[3, 4]
>>> len(l)
4
```

Lists operators

```
x = [42, 't', 1.3]
```

length	`len(x)`	`3`
concatenate	`[1, 2] + x`	`[1, 2, 42, 't', 1.3]`
membership	`42 in x`	`True`
slice	`x[0:2]`	`[42, 't']`
append	`x.append(3)`	`x: [42, 't', 1.3, 3]`
extend	`x += [3, 1]`	`x: [42, 't', 1.3, 3, 1]`
insert	`x.insert(1, 'a')`	`x: [42, 'a', 't', 1.3]`
delete	`del x[1]`	`x: [42, 1.3]`
remove	`x.pop(1)`	`'t'  x: [42, 1.3]`

Tuples

- Immutable lists. Usage e.g. days of a week, months.

```
>>> l = [ 1, 2, 3, 4 ]
>>> t = tuple(l)
>>> t
(1, 2, 3, 4)
>>> l == t
False
>>> l2 = list(t)
>>> l == l2
True
>>> len(l) == len(t)
True
>>> t.append(7)
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    t.append(7)
AttributeError: 'tuple' object has no attribute 'append'
>>> t = (1, 2)
>>> t += (3, 4) # Created new tuple & added to `t`
>>> t+=(3) # Error as 3 is now treated as integer
>>> t+=(3,) # Correct
```

Sets

- *A `set` object is an unordered collection of distinct hashable objects*
- *Mutable*

```
>>> foo = set( [1,2,3] )
>>> 2 in foo
True
>>> 5 in foo
False
>>> foo.add( 2 )
>>> foo
set([1, 2, 3])
>>> foo.add(6)
>>> foo.remove(2)
>>> foo
Set([1, 3, 6])
>>> other = set([4, 5, 6, 7])
>>> foo & other
set([6])
```

Frozen set

- *A `frozenset` is an immutable set*

```
>>> bar = frozenset( foo )
>>> bar.remove(6)
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    bar.remove(6)
AttributeError: 'frozenset' object has no attribute
'remove'
```

Dictionaries

- *A dictionary object maps hashable values to arbitrary objects*

```
>>> d = { 'a':3, 'b':2, 3:'x' }
>>> d['a']
3
>>> d.get( 5 )
>>> d.get(3) # Use 'get()' if you don't know if the key is present
'x'
>>> d['x'] = 666
>>> d
{'a': 3, 3: 'x', 'b': 2, 'x': 666}
>>> 3 in d
True
>>> del d['b'] # removed_value = d.pop('b') # Removes 'b' and
returns its value
>>> d
{'a': 3, 'b': 2, 'x': 666}
>>> d.keys()
['a', 'b', 'x']
>>> d.items()
[('a', 3), ('b', 2), ('x', 666)]
>>> 'y' in d
False
```

Mutable vs. Immutable

- Mutable types allow changes to objects in memory
  - Examples: list, dictionary
- Immutable types do not
  - Examples: int, float, str, bool

```
>>> x = 42
>>> y = x
>>> x += 1
>>> print (x)
43
>>> print (y)  # ??
```

```
>>> x = []
>>> y = x
>>> x += [1]
>>> print (x)
[1]
>>> print (y)    # ??
```

Control Structures

if, elif, else

```
# Input: student's score
score = int(input("Enter the score: "))

# Determine the grade
if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
elif score >= 70:
    grade = "C"
elif score >= 60:
    grade = "D"
else:
    grade = "F"

# Output the result
print(f"The grade is: {grade}")
```

for loop

```
# Loop through numbers 1 to 5
for num in range(1, 6): # outputs less than 2nd input
    print(f"The square of {num} is {num ** 2}")


The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

This uses f-strings (formatted string literals) to construct a dynamic message.

```
print("The square of " + str(num) + " is " + str(num ** 2))
print("The square of {} is {}".format(num, num ** 2))
print("The square of %d is %d" % (num, num ** 2))
```

while loop

```
# Initialize the counter
num = 1

# Loop while the condition is True
while num <= 5:
    print(f"Number: {num}")
    num += 1  # Increment the counter

Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

Sequences and Loops

```
for item in seq:    # list, tuple, str, set
    print (item)
sum = 0
for n in range(1, 11):
    sum += n
print (sum)
```

Dictionaries and Loops

```
for key in dict.keys():
    print ('%s => %s' % (key, dict[key]))
```

```
for pair in dict.items():
    print ('%s => %s' % pair)
```

```
for key, value in dict.items():
    print ('%s => %s' % (key, value))
```

Indentation

Indentation

- In Python, indentation is significant
- Whitespace is used to delimit program blocks
- Blocks are introduced with a colon ":"
- Each line within a basic block must be indented by the same amount

String Methods

String Methods

- String methods are built-in functions in Python that operate on string objects, allowing users to manipulate, format, or query strings easily.
- **Basic String Manipulation**

Method	Description	Example
str.upper()	Converts all characters to uppercase.	"hello".upper() → 'HELLO'
str.lower()	Converts all characters to lowercase.	"HELLO".lower() → 'hello'
str.capitalize()	Capitalizes the first character.	"python".capitalize() → 'Python'
str.title()	Capitalizes the first letter of each word.	"hello world".title() → 'Hello World'
str.strip()	Removes leading/trailing whitespace.	" hello ".strip() → 'hello'
str.lstrip()	Removes leading whitespace.	" hello".lstrip() → 'hello'
str.rstrip()	Removes trailing whitespace.	"hello ".rstrip() → 'hello'
str.replace(old, new)	Replaces all occurrences of a substring.	"banana".replace('a', 'o') → 'bonono'

String Methods

- **Querying a string**

Method	Description	Example
str.startswith(prefix)	Checks if the string starts with the given prefix.	"hello".startswith('he') → True
str.endswith(suffix)	Checks if the string ends with the given suffix.	"hello".endswith('lo') → True
str.find(sub)	Returns the index of the first occurrence of a substring, or -1.	"apple".find('p') → 1
str.index(sub)	Like find() but raises a ValueError if not found.	"apple".index('p') → 1
str.count(sub)	Counts occurrences of a substring.	"banana".count('a') → 3

- **String Splitting and Joining**

Method	Description	Example
str.split()	Splits the string into a list of words (default delimiter: space). From left.	"a,b,c".split(',') → ['a', 'b', 'c']
str.rsplit()	Splits the string from the right into a list. When no maxsplit is specified, both split() and rsplit() produce the same result. Maximum number of splits to perform. If not specified or set to -1.	"a b c".rsplit(maxsplit=1) → ['a b', 'c']
str.join(iterable)	Joins elements of an iterable with the string as a separator.	",".join(['a', 'b', 'c']) → 'a,b,c'

String Vs Integers

- '5’ isn’t equal to 5
- Conversions

```
>>> int('5')
5
>>> float('5')
5.0
>>> str(5)
'5'
>>> int(5.5)
5
>>> float(5)
5.0
```

While vs for

```
password = ""
while password != "secret":
    password = input("Enter password: ")


import random
x = 0
while x < 0.9:
    x = random.random()
    print(x)
```

Strings

- Single quotes and double quotes work same way
- Escape characters : '\n', '\t', etc.

```
>>>a='foo\tbar'
>>> a
'foo\tbar'
>>> print(a)
foo bar
```

Comments & Line Continuations

- Comments are started with “#”
- Long lines can be continued by ending with a backslash “\”
- Multiline Strings- triple quotes

```
>>> a = "Nick " + \
... "LeRoy"
>>>
>>> a
'Nick LeRoy’
>>> multiline1 = '''This is
a multiline
string.'''
>>>  multiline1
'This is\na multiline\nstring.’
>>> print(multiline1)
This is
a multiline
string.
```

```
# This is a comment
a = 5 # This is too
```

Inputs

Feature	raw_input() (Python 2)	input() (Python 2)	input() (Python 3)
Reads as String	Yes	No (evaluates input)	Yes
Evaluates Input	No	Yes	No
Explicit Conversion	Not required for strings	Required for non-strings	Required for non-strings

raw_input removed in Python 3

```
>>> 'Python' > 'C++'
True
>>> 'Python' > 'Z'
False
```

Identity: To get address

```
>>> a = 'abcdef'
>>> b = 'abc'
>>> b += 'def'
>>> a is b
False
>>> id(a), id(b)
(140239763722624, 140239763722864)
```

Operator Precedence Table

s	Operator	Description	Associativity
1 (Highest)	()	Parentheses (grouping)	N/A (evaluated first)
2	**	Exponentiation	Right-to-left
3	+x, -x, ~x	Unary operators: positive, negative, bitwise NOT	Right-to-left
4	*, /, //, %	Multiplication, division, floor division, modulus	Left-to-right
5	+, -	Addition, subtraction	Left-to-right
6	<<, >>	Bitwise shift operators	Left-to-right
7	&	Bitwise AND	Left-to-right
8	^	Bitwise XOR	Left-to-right
9	|	Bitwise OR	Bitwise OR
10	==, !=, >, <, >=, <=, is, is not, in, not in	Comparisons and membership tests	Left-to-right
11	not	Logical NOT	Right-to-left
12	and	Logical AND	Left-to-right
13 (Lowest)	or	Logical OR	Left-to-right

Examples

```
result = 2 ** 3 ** 2  # Equivalent to 2 ** (3 ** 2)
result = -3 + 5  # Unary `-` applied first
result = True or False and False
# `and` is evaluated first, so it becomes True or (False
and False) → True
result = 10 - 5 - 2  # Equivalent to (10 - 5) - 2
```

Examples

Operator	Example	Explanation	Output
Parentheses	result = (2 + 3) * 4	Grouping ensures 2 + 3 is evaluated first.	20
Exponentiation	result = 2 ** 3 ** 2	Exponentiation is right-to-left, so 3 ** 2 first, then 2 ** 9.	512
****Unary +, -, ~****	result = -3 + ~2	Unary - makes -3, ~2 is bitwise NOT of 2. ~2 = -(2 + 1) = -3	-3 + (-3) = -6
Multiplication/Division/Floor Division/Modulus	result = 10 % 3 * 2 // 2	10 % 3 → 1, 1 * 2 → 2, then 2 // 2.	1
Addition/Subtraction	result = 5 + 3 - 2	Left-to-right: 5 + 3 = 8, then 8 - 2.	6
Bitwise Shifts	result = 4 << 1 >> 1	4 << 1 → 8, then 8 >> 1.	4
Bitwise AND	result = 5 & 3	5 in binary 101, 3 in binary 011.	1
Bitwise XOR	result = 5 ^ 3	Binary XOR: 101 ^ 011 → 110.	6
Bitwise OR	result = 5 | 3	3`	7
Comparison	result = 5 > 3 == True	5 > 3 → True	True
Membership	result = 'a' in 'abc'	'a' is present in 'abc'.	True
Identity	result = 5 is 5.0	5 and 5.0 have different types.	False
Logical NOT	result = not False	Negates False → True.	True
Logical AND	result = True and False	Both must be True → False.	False
Logical OR	result = True or False	Only one needs to be True → True.	True

break & continue

- To break out of a loop in the middle:
- *break*
- To go back to the top of a loop:
- *continue*

```
i = 0
while True :
    i += 1
    if i == 5 :
        continue
        print (i)
    if i > 10 :
        break
```

range function

- range(n) is a built-in function
- Returns a list of integers
- range(5) will return the list : (0,1,2,3,4)
- range(m,n):
- Returns a list of integers
- range(1,5) will return the list : (1,2,3,4)

```
for i in range(10) :
    if i == 5 :
        continue
    print (i)
```

```
>>> print(range(1,5) )
range(1, 5)
>>> a=range(10)
>>> a
range(0, 10)
>>> list(a)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- This happens because the range() function creates a lazy, iterable object that only generates the numbers on demand, rather than storing them explicitly in memory.

Examples

```
>>> babylon5 = ["Sheridan","G'Kar","G'Kar","Delenn"]
>>> babylon5[1]
"G'Kar"
>>> babylon5[-1]
'Delenn'
>>> babylon5.index("G'Kar")
1
>>> babylon5[2]='Zathras'
>>> babylon5[4]='Zathras'
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Slices

- sequence[start:stop:step]
- start: The index to start the slice (inclusive). Defaults to 0 if omitted.
- stop: The index to end the slice (exclusive). Defaults to the length of the sequence if omitted.
- step: The increment (step) between each index. Defaults to 1 if omitted.

```
lst = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


# Basic slicing
print(lst[2:6])      # Output: [2, 3, 4, 5] (indexes 2 to 5)
print(lst[:4])       # Output: [0, 1, 2, 3] (start defaults to 0)
print(lst[5:])       # Output: [5, 6, 7, 8, 9] (stop defaults to end)


# Negative indexing
print(lst[-5:-2])    # Output: [5, 6, 7]
print(lst[-5:])      # Output: [5, 6, 7, 8, 9]


# Using steps
print(lst[::3])      # Output: [0, 3, 6, 9] (every third element)
print(lst[::-2])     # Output: [9, 7, 5, 3, 1] (reverse with step -2)
```

Functions

Functions

- Allow for code reuse
  - Same block of code can be used from several different places
- Central to procedural and object-oriented programming
- Sometimes required
  - i.e. for callbacks, etc. (In GUIs etc.)
- You've already used a lot of them:
  - int(), len(), float(), etc.

Functions

```
def process_data(data, callback):
    # Perform some processing on data
    processed_data = data.upper()  # Convert data to uppercase
    # Call the callback function with the processed data
    callback(processed_data)

# A callback function to be executed
def display_result(result):
    print(f"Processed Data: {result}")

# Example data
input_data = "hello world"

# Call process_data and pass the callback function
process_data(input_data, display_result)

Processed Data: HELLO WORLD
```

Default Values

- Parameters can have default values
- Parameters can be passed by name

```
#! /usr/bin/env python
"""Example"""
def MyFunc( a, b='xyzzy', c=None ) :
    """Takes an 1, 2 or 3 parameters"""
    s = None
    if c is None :
        s = "a=%d b=%s" % ( a, str(b) )
    else :
        s = "a=%d b=%s c=%s" % ( a, str(b), c )
    if isinstance( b, int ) :
        s += " (b is an int)"
    return s
# Call the function in various ways
print (MyFunc( 1 ))
print (MyFunc( 1, 'foo' ))
print (MyFunc( 1, 5, 'fizbin' ))
print (MyFunc( 1, c='plugh' )) # Note: "c" is specified by name!

a=1 b=xyzzy
a=1 b=foo
a=1 b=5 c=fizbin (b is an int)
a=1 b=xyzzy c=plugh
```

Returning Multiple Values

```
#! /usr/bin/env python
"""Example"""
def MyFunc( num=None ) :
	"""Returns a string, and an int (or None)"""
	sval = input("Enter a string: " )
	if num is not None :
		ival = num
	else :
		try :
			ival = int(input("Enter an integer: ") )
		except ValueError :
			ival = None
	return sval, ival
for v in ( None, 1, 3 ) :
	s, i = MyFunc( v )
	if i is None :
		i = 999
	print "i=%d s=%s" % ( i, s )
```

```
 Enter a string: 123
 Enter an integer: asd
 i=999 s=123
 Enter a string: def
 i=1 s=def
 Enter a string: ags
 i=3 s=ags
```

Help

Built-In Help I

dir(object or type)

- Lists all operations for that object or type
- For now, ignore everything that starts with __
- Use as object.operation(…)

```
dir(str)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getstate__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__', 'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric',
'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'removeprefix', 'removesuffix', 'replace', 'rfind', 'rindex',
'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Built-In Help II

help(something)

- Shows built-in documentation
- Works on objects, types, and their operations

```
help(str.lower)
Help on method_descriptor:

lower(self, /) unbound builtins.str method
    Return a copy of the string converted to lowercase.
help(lower)
Traceback (most recent call last):
  File "<pyshell#67>", line 1, in <module>
    help(lower)
NameError: name 'lower' is not defined
```

Handling Exceptions & other

Handling Exceptions

```
try:
    num = int(input("Enter a number: "))
    result = 10 / num
    print(f"The result is {result}")
#Raised when the input cannot be converted to an integer.
except ValueError:
    print("Invalid input! Please enter a valid integer.")
#Raised when attempting to divide by zero
except ZeroDivisionError:
    print("Division by zero is not allowed.")
# Executes if no exception occurs in the try block
else:
    print("No exceptions occurred. The operation was successful.")
#Executes regardless of whether an exception occurred or not. Often used for cleanup (e.g., closing files or releasing resources).
finally:
    print("Execution completed.")
```

assert

- Used primarily as a convenient way to insert debugging assertions into a program: assert <expression>
- If <expression> evaluates to 0 (zero) or False, an AssertionError exception is raised.

```
def my_func( a, b ) :
    assert isinstance( a, int )
    assert b is not None
    ...
```

Functions: Arguments Are Local Variables

- The function can't modify the caller's passed-in variable

```
def MyFunc( a ) :
    """Paramter a is a local variable, thus it's changes to
    the variable don't affect the caller's copy"""
    a += 10
    print ("MyFunc: a =", a)
    return a
# Call the function in various ways
b = MyFunc( 1 )
print ("main: b =", b)
b = 10
MyFunc( b )
print ("main: b =", b)
```

```
MyFunc: a = 11
main: b = 11
MyFunc: a = 20
main: b = 10
```

Function can access global variable

- The function can't modify the caller's passed-in variable
- Assignments to globals don't work (scooping)
- Use the keyword global to modify
- A function can even have local functions

```
foo = 5
def SomeFunc( ) :
    print ("SomeFunc: foo is", foo)
    foo=10 # Scooping
SomeFunc( )
print ("main: foo is", foo)
```

```
SomeFunc: foo is 5
main: foo is 5
```

File Handling

Read a file

- Read the Entire File

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

- Read File Line by Line

```
with open("example.txt", "r") as file:
    for line in file:
        print(line.strip())  # strip() to remove extra
spaces or newline characters.
```

- Read All Lines into a List

```
with open("example.txt", "r") as file:
    lines = file.readlines()
    print(lines)
```

Read a file with exceptions

```
try:
    # Attempt to open and read the file
    with open("example.txt", "r") as file:
        content = file.read()
        print(content)
except FileNotFoundError:
    # Handle case where the file doesn't exist
    print("Error: The file 'example.txt' was not found.")
except PermissionError:
    # Handle case where the file cannot be accessed
    print("Error: You do not have permission to read the
file.")
except Exception as e:
    # Catch any other unexpected exceptions
    print(f"An unexpected error occurred: {e}")
finally:
    # Optional: Code to execute no matter what happens
    print("File read attempt completed.")
```

Writing in a file

- Writing to a File Using write()

```
with open("example.txt", "w") as file:
    file.write("This is the first line.\n")
    file.write("This is the second line.\n")
```

- Appending to a File Using write()

```
with open("example.txt", "a") as file:
    file.write("This is an appended line.\n")
```

- Writing Multiple Lines Using writelines()

```
lines = ["Line 1\n", "Line 2\n", "Line 3\n"]
with open("example.txt", "w") as file:
    file.writelines(lines)
```

Writing in a file with exceptions

```
try:
    with open("example.txt", "w") as file:
        file.write("This is safe writing.\n")
except Exception as e:
    print(f"An error occurred: {e}")
```

```
try:
    # Attempt to open and write to the file
    with open("example.txt", "w") as file:
        file.write("This is the first line.\n")
        file.write("This is the second line.\n")
    print("File written successfully.")
except FileNotFoundError:
    # Raised if the file path is invalid or inaccessible
    print("Error: The file path does not exist.")
except PermissionError:
    # Raised if the program lacks permission to write to the file
    print("Error: You do not have the necessary permissions to write to
this file.")
except Exception as e:
    # Handles any other exceptions
    print(f"An unexpected error occurred: {e}")
finally:
    # Code in the `finally` block executes no matter what happens
    print("Write operation completed.")
```

Python Modules

Python Modules

- Python has a large collection of standard modules
  - New types, parsers, etc.
  - os, sys, copy, optparse
- There are also a lot of 3rd party modules available
  - NumPy (numerical operations)
  - Beautiful Soup (HTML parser)

Using a module

- To use a module, use the “import” keyword
- import <module>
  - Imports all symbols from the module into it's own namespace
- from <module> import a,b
  - Imports only symbols “a” and “b” from the module into the local namespace
- from <module> import *
  - Import all symbols from the module into the local namespace

Copying a list

```
>>> import copy
>>> a=[1, 2, 3, 4]
>>> b = copy.copy(a)
>>> a is b
False
>>> a.append(5)
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4]
>>> from copy import copy
>>> b = copy(a)
```

lambda

- lambda is a way to create anonymous functions (functions without a name) in a concise manner
- lambda arguments: expression

```
# Lambda function to add two numbers
add = lambda x, y: x + y
print(add(3, 4))  # Output: 7
```

- lambda can be used with map, filter, sorted, conditional, etc.

```
# Lambda function for a conditional operation
greater_than_five = lambda x: "Greater" if x > 5 else
"Lesser"
print(greater_than_five(7))  # Output: Greater
print(greater_than_five(3))  # Output: Lesser
```

map

- map() function applies a given function to all items in an iterable and returns a map object (which is an iterator) containing the results.
- map(function, iterable, ...)

```
numbers = [1, 2, 3, 4, 5]

# Applying a function to square each number
def square(x):
    return x ** 2

result = map(square, numbers)
print(list(result))  # Output: [1, 4, 9, 16, 25]

# Using a lambda function to double each number
result = map(lambda x: x * 2, numbers)
print(list(result))  # Output: [2, 4, 6, 8, 10]
# Adding corresponding elements from two lists
result = map(lambda x, y: x + y, a, b)
print(list(result))  # Output: [5, 7, 9]
```

map

```
a = [1, 2, 3]

# Applying two functions: square and double
def square(x):
    return x ** 2

def double(x):
    return x * 2

result = map(lambda x: double(square(x)), a)
print(list(result))  # Output: [2, 8, 18]
```

Array flattening

- Python does not flatten arrays

```
>>> x = [1,2,3]
>>> [5,x,6,x]
[5, [1, 2, 3], 6, [1, 2, 3]]
```

- To concatenate elements of a list in another list, use the + or +=

```
>>> x=[ 1, 2, 3]
>>> [ 5] + x + [ 6 ] + x
[5, 1, 2, 3, 6, 1, 2, 3]
```

Array concatenation

- join() is used to concatenate elements using delimiter.join(iterable)

```
words = ["Hello", "World", "Python"]
result = " ".join(words) # Joining with a space
print(result)  # Output: "Hello World Python"
```

- join() method does not work if the iterable contains non-string elements. Convert to string first.

```
numbers = [1, 2, 3, 4, 5]
# Convert each number to a string and join with a space
result = " ".join(map(str, numbers))
print(result)  # Output: "1 2 3 4 5"
```

Functions vs Methods

Feature	Function	Method
Definition	Defined using def or lambda	Defined inside a class
Call	Can be called directly (e.g., greet())	Called on an instance or class (e.g., obj.greet())
Binding	Not bound to any object	Always bound to an object or class
First Parameter	No special parameter (can be any argument)	Usually self (for instance) or cls (for class methods)
Access to Object	Does not have access to object attributes	Can access and modify object attributes via self

‘is’ for object identity not value

```
a='abc'
b=a
id(a) is id(b)
False
a is b
True
id(a)
4330969600
id(b)
4330969600
id(a) == id(b)
True

# As id(a) is
value and not
object
```

```
a = 'abcdef'
b='abcdef’ # same memory as a is used
(string and integer specific)
id(a)
4311225744
id(b)
4311225744
a is b # not reliable
True
b='abc'
b+='def’ # New memory created on runtime
b
'abcdef'
a is b
False
```

is

```
a = [1, 2, 3]
b = [1, 2, 3]

a == b   # True  (same value)
a is b   # False (different objects)
```

When to use ‘is’

- To check None: x is None better than x==None
- Check if True is True, etc., for singletons: True, None, and False. They have a single memory.

```
id(True)
4330737016
a=10
id(a==10)
4330737016
```

- Check object identity a=b; a is b

Thank You

- All my slides/notes excluding third party material are licensed by various authors including myself under https://creativecommons.org/licenses/by-nc/4.0/
- LeRoy & De Smet notes

Data Science

DAI-101 Spring 2025-26

Dr. Devesh Bhimsaria

Office: F9, Old Building

Department of Biosciences and Bioengineering

Indian Institute of Technology–Roorkee

devesh.bhimsaria@bt.iitr.ac.in

Python challenges

Loops

```
i = 0
while True :
   i += 1
   if i == 5 :
      continue
      print (i)
   if i > 10 :
      break
```

Loops

```
i = 0
while True :
   if i == 5 :
      continue
      print (i)
   if i > 10 :
      break
   i += 1
```

Single line code challenge

Python challenge

- Python code to print a string with gaps as | 0 to 8
- Output should be following without using loops and writing 0 to 8

0|1|2|3|4|5|6|7|8

```
print('|'.join((map(str,list(range(9))))))
```

Reverse a String

- Print the reverse of the string "Python" in one line of code.
- print("Python"[::-1])

Print Even Numbers from a List

- Print all even numbers from the list [1, 2, 3, 4, 5, 6, 7, 8, 9] in one line
- print([x for x in range(1,10) if x % 2 == 0])
- [expression for item in iterable if condition]
- Above creates a list of expression when condition is satisfied

Flatten a Nested List

- Flatten the list [[1, 2], [3, 4], [5, 6]] in one line.
- print([x for sublist in [[1, 2], [3, 4], [5, 6]] for x in sublist])

Count Vowels in a String

- Count the number of vowels in the string "Hello World" in one line.
- print(sum(1 for char in "Hello World" if char.lower() in "aeiou"))

Remove duplicates from a list

- Remove duplicates from the list [1, 2, 2, 3, 4, 4, 5] in one line.
- print(list(set([1, 2, 2, 3, 4, 4, 5])))

Generate Squares of Numbers

- Generate a list of squares of numbers from 1 to 10 in one line.
- print([x**2 for x in range(1, 11)])

Swap variables

- Swap the values of a and b in one line.
- a, b = b, a

Check Prime Number

- Write a one-liner to check if x is a prime number.
- x=29
- all() function returns True if all items in an iterable are true
- print(all(x % i != 0 for i in range(2, int(x**0.5) + 1)))

Create a Dictionary from Two Lists

- Create a dictionary from the lists ['a', 'b', 'c'] and [1, 2, 3] in one line.
- The zip() function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc.
- print(dict(zip(['a', 'b', 'c'], [1, 2, 3])))

Sort a List of Tuples by Second Element

- Sort the list of tuples [(1, 3), (4, 1), (2, 2)] by their second element in one line.
- sorted([(1, 3), (4, 1), (2, 2)], key=lambda x: x[1])

Thank You

- All my slides/notes excluding third party material are licensed by various authors including myself under https://creativecommons.org/licenses/by-nc/4.0/

Data Science

DAI-101 Spring 2025-26

Dr. Devesh Bhimsaria

Office: F9, Old Building

Department of Biosciences and Bioengineering

Indian Institute of Technology–Roorkee

devesh.bhimsaria@bt.iitr.ac.in

Data Analysis Foundation

Data Matrix

- Tabular representation:
  - Rows: Individual observations.
  - Columns: Variables/features.

Sample ID	Age	Income	Gender
1	25	50000	Male
2	30	60000	Female
3	22	45000	Male

Types of Data

Numeric Data

Types:

Continuous Data: Any number within a range (e.g., height).

Discrete Data: Distinct whole numbers (e.g., student count).

Age- actually continuous, we can discretize it.

Sample ID	Height (cm)	Age
1	165.5	25
2	170.2	30
3	160	22

Types of Data

Types:

Nominal: No inherent order (e.g., gender).

Ordinal: Ordered categories (e.g., education level).

Sample ID	Gender	Education Level
1	Male	Bachelor
2	Female	Master
3	Male	High School

Numeric vs Categorical Data

Feature	Numeric Data	Categorical Data
Representation	Numbers	Labels or categories
Operations	Arithmetic operations allowed	No arithmetic operations
Visualization	Histograms, scatter plots	Bar charts, pie charts

Mixed Datasets

- Real-world datasets often combine numeric and categorical variables.

Sample ID	Age	Income	Gender
1	25	50000	Male
2	30	60000	Female
3	22	45000	Male

Choosing the Right Analysis Techniques

Data Type	Common Techniques/Methods
Numeric	Descriptive statistics, regression analysis, time series
Categorical	Frequency analysis, chi-square test, clustering
Mixed	Data preprocessing (encoding), decision trees, random forests

Data Matrix

Data can often be represented or abstracted as an $n \times d$ *data matrix*, with n rows and d columns, given as

$$\boldsymbol{D} = \left(\begin{array}{c|cccc} & X_1 & X_2 & \cdots & X_d \\ \hline \boldsymbol{x}_1 & x_{11} & x_{12} & \cdots & x_{1d} \\ \boldsymbol{x}_2 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_n & x_{n1} & x_{n2} & \cdots & x_{nd} \end{array}\right)$$

- **Rows**: Also called *instances*, *examples*, *records*, *transactions*, *objects*, *points*, *feature-vectors*, etc. Given as a d-tuple

$$\boldsymbol{x}_i = (x_{i1}, x_{i2}, \ldots, x_{id})$$

- **Columns**: Also called *attributes*, *properties*, *features*, *dimensions*, *variables*, *fields*, etc. Given as an n-tuple

$$X_j = (x_{1j}, x_{2j}, \ldots, x_{nj})$$

Iris Dataset Extract

	Sepal length X_1	Sepal width X_2	Petal length X_3	Petal width X_4	Class X_5
$\boldsymbol{x}_1$	5.9	3.0	4.2	1.5	Iris-versicolor
$\boldsymbol{x}_2$	6.9	3.1	4.9	1.5	Iris-versicolor
$\boldsymbol{x}_3$	6.6	2.9	4.6	1.3	Iris-versicolor
$\boldsymbol{x}_4$	4.6	3.2	1.4	0.2	Iris-setosa
$\boldsymbol{x}_5$	6.0	2.2	4.0	1.0	Iris-versicolor
$\boldsymbol{x}_6$	4.7	3.2	1.3	0.2	Iris-setosa
$\boldsymbol{x}_7$	6.5	3.0	5.8	2.2	Iris-virginica
$\boldsymbol{x}_8$	5.8	2.7	5.1	1.9	Iris-virginica
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\boldsymbol{x}_{149}$	7.7	3.8	6.7	2.2	Iris-virginica
$\boldsymbol{x}_{150}$	5.1	3.4	1.5	0.2	Iris-setosa

Data: Algebraic and Geometric View

For numeric data matrix $\boldsymbol{D}$, each row or point is a d-dimensional column vector:

$$\boldsymbol{x}_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{id} \end{pmatrix} = \begin{pmatrix} x_{i1} & x_{i2} & \cdots & x_{id} \end{pmatrix}^T \in \mathbb{R}^d$$

whereas each column or attribute is a n-dimensional column vector:

$$X_j = \begin{pmatrix} x_{1j} & x_{2j} & \cdots & x_{nj} \end{pmatrix}^T \in \mathbb{R}^n$$

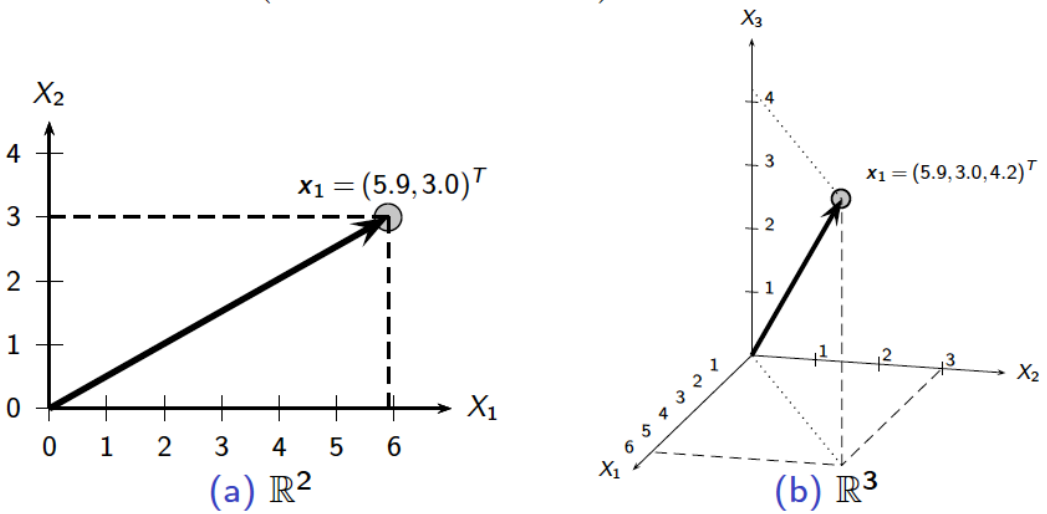
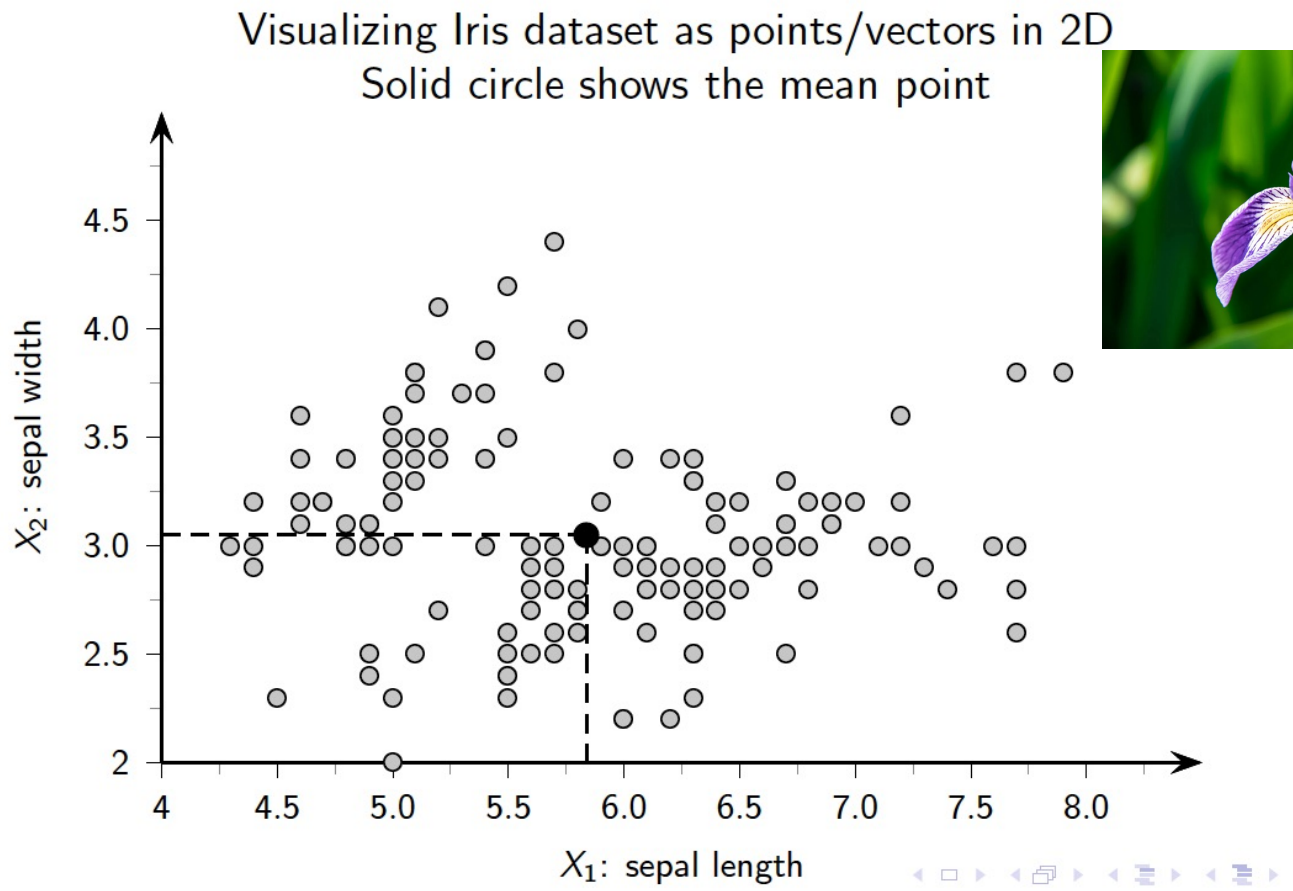


Figure: Projections of $\boldsymbol{x}_1 = (5.9, 3.0, 4.2, 1.5)^T$ in 2D and 3D

Scatterplot: 2D Iris Dataset
sepal length versus sepal width.



Numeric Data Matrix

If all attributes are numeric, then the data matrix $\boldsymbol{D}$ is an $n \times d$ matrix, or equivalently a set of n row vectors $\boldsymbol{x}_i^T \in \mathbb{R}^d$ or a set of d column vectors $X_j \in \mathbb{R}^n$

$$\boldsymbol{D} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} = \begin{pmatrix} — \boldsymbol{x}_1^T — \\ — \boldsymbol{x}_2^T — \\ \vdots \\ — \boldsymbol{x}_n^T — \end{pmatrix} = \begin{pmatrix} | & | & & | \\ X_1 & X_2 & \cdots & X_d \\ | & | & & | \end{pmatrix}$$

The *mean* of the data matrix $\boldsymbol{D}$ is the average of all the points: $mean(\boldsymbol{D}) = \boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$

The *centered data matrix* is obtained by subtracting the mean from all the points:

$$\boldsymbol{Z} = \boldsymbol{D} - 1 \cdot \boldsymbol{\mu}^T = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_n^T \end{pmatrix} - \begin{pmatrix} \boldsymbol{\mu}^T \\ \boldsymbol{\mu}^T \\ \vdots \\ \boldsymbol{\mu}^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{x}_1^T - \boldsymbol{\mu}^T \\ \boldsymbol{x}_2^T - \boldsymbol{\mu}^T \\ \vdots \\ \boldsymbol{x}_n^T - \boldsymbol{\mu}^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{z}_1^T \\ \boldsymbol{z}_2^T \\ \vdots \\ \boldsymbol{z}_n^T \end{pmatrix} \quad (1)$$

where $\boldsymbol{z}_i = \boldsymbol{x}_i - \boldsymbol{\mu}$ is a centered point, and $1 \in \mathbb{R}^n$ is the vector of ones.

In **BOLD** means vector/matrix

Norm, Distance and Angle

Given two points $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^m$, their *dot product* is defined as the scalar

$$\begin{aligned}\boldsymbol{a}^T \boldsymbol{b} &= a_1 b_1 + a_2 b_2 + \cdots + a_m b_m \\ &= \sum_{i=1}^{m} a_i b_i\end{aligned}$$

The *Euclidean norm* or *length* of a vector $\boldsymbol{a}$ is defined as

$$\|\boldsymbol{a}\| = \sqrt{\boldsymbol{a}^T \boldsymbol{a}} = \sqrt{\sum_{i=1}^{m} a_i^2}$$

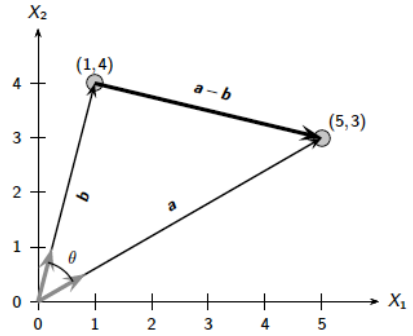
The *unit vector* in the direction of $\boldsymbol{a}$ is $\boldsymbol{u} = \frac{\boldsymbol{a}}{\|\boldsymbol{a}\|}$ with $\|\boldsymbol{a}\| = 1$.

Distance between $\boldsymbol{a}$ and $\boldsymbol{b}$ is given as

$$\|\boldsymbol{a} - \boldsymbol{b}\| = \sqrt{\sum_{i=1}^{m} (a_i - b_i)^2}$$

Angle between $\boldsymbol{a}$ and $\boldsymbol{b}$ is given as

$$\cos\theta = \frac{\boldsymbol{a}^T \boldsymbol{b}}{\|\boldsymbol{a}\| \, \|\boldsymbol{b}\|} = \left(\frac{\boldsymbol{a}}{\|\boldsymbol{a}\|}\right)^T \left(\frac{\boldsymbol{b}}{\|\boldsymbol{b}\|}\right)$$



Data Preparation

Introduction to Data Preparation

- Data preparation is a crucial step in data science.
- It ensures data quality, consistency, and efficiency for analysis.
- Main steps: Data Cleaning, Data Reduction, Data Transformation

Data Cleaning

Data Cleaning

- Noise and outliers
- Missing values
- Duplicate data

A mistake or a millionaire?

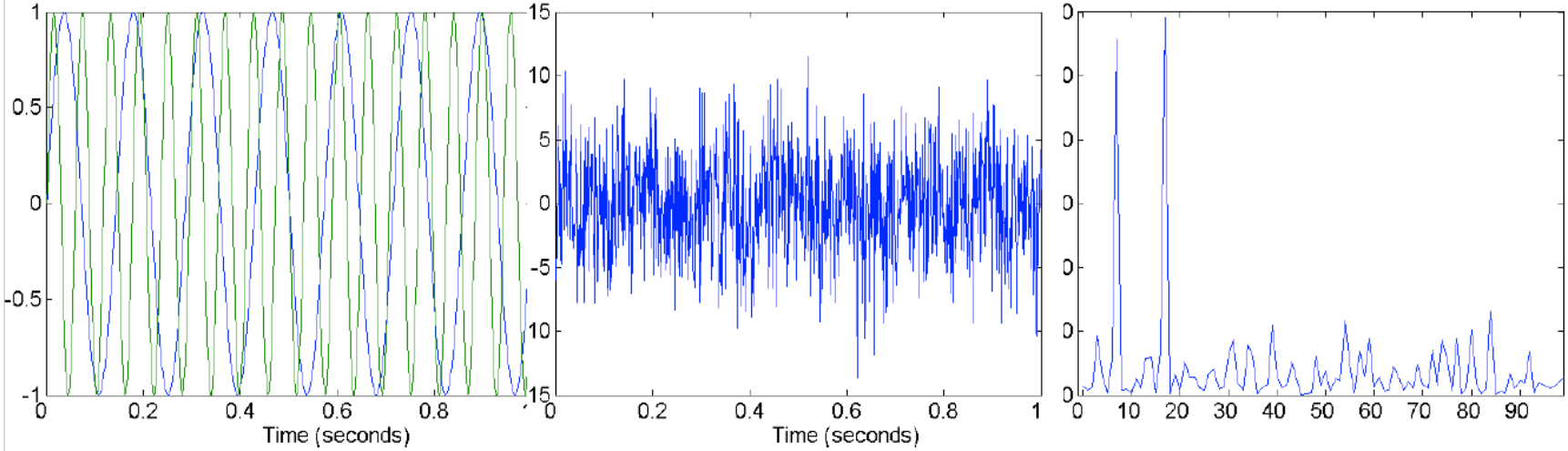
Missing values

Inconsistent duplicate entries

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	10000K	Yes
6	No	NULL	60K	No
7	Yes	Divorced	220K	NULL
8	No	Single	85K	Yes
9	No	Married	90K	No
9	No	Single	90K	No

Noise in Data

- Noise refers to modification of original values
  - Examples: distortion of a person's voice when talking on



Two Sine Waves

Two Sine Waves + Noise

Frequency Plot (FFT)

Noise in Data

- Smooth by fitting a regression function
- Combined computer and human inspection
- Fourier Transform & filter out noise frequency

What is Data Cleaning?

- Data cleaning is the process of identifying and correcting errors in datasets.
- Ensures data quality, consistency, and accuracy.
- Common issues: missing values, duplicates, inconsistencies, and outliers.
- Improves model performance and reliability.

Handling Missing Values

Methods to handle missing data:

- **Deletion**: Remove rows/columns with missing values (only if data loss is minimal).
- **Imputation**: the process of replacing missing data in a dataset with estimated values.
  - Mean, median, or mode replacement.
  - Predictive models (KNN, regression, deep learning).
  - Using domain knowledge to fill in the missing values appropriately.

Handling Missing Values

- Original Data:

Name | Age | Salary

Alice | 25 | 50000

Bob | | 60000

Eve | 30 |

- After Imputation (Mean Age: 27.5, Mean Salary: 55000):

Bob | 27.5 | 60000

Eve | 30 | 55000

Removing Duplicates

- Duplicate records can arise from multiple data sources or errors in data entry. Example: Same person with multiple email addresses.
- Methods to remove duplicates:
  - Identify duplicates using unique identifiers or record matching.
  - Remove exact duplicates using `drop_duplicates()` in Pandas.
  - Merge similar records where applicable instead of outright deletion.

Removing Duplicates

- Original Data:

ID | Name | Age | Salary
1 | Alice | 25 | 50000
2 | Bob | 30 | 60000
2 | Bob | 30 | 60000
3 | Eve | 35 | 70000

- After Removing Duplicates:

ID | Name | Age | Salary
1 | Alice | 25 | 50000
2 | Bob | 30 | 60000
3 | Eve | 35 | 70000

Fixing Inconsistent Data

Data inconsistency occurs due to format mismatches, spelling errors, or conflicting values.

- Steps to fix inconsistencies:
  - Standardizing formats (e.g., date formats: YYYY-MM-DD).
  - Correcting typos (e.g., 'NY' vs. 'New York').
  - Unifying categorical data (e.g., 'Male' vs. 'M').
  - Automating data validation using scripts.

Fixing Inconsistent Data

- Before Fixing:

ID | Name | Date | Gender
1 | Alice | 01-02-2024 | Female
2 | Bob | 2024/02/01 | M
3 | Eve | 02-01-2024 | Female

- After Standardization:

ID | Name | Date | Gender
1 | Alice | 2024-02-01 | Female
2 | Bob | 2024-02-01 | Male
3 | Eve | 2024-02-01 | Female

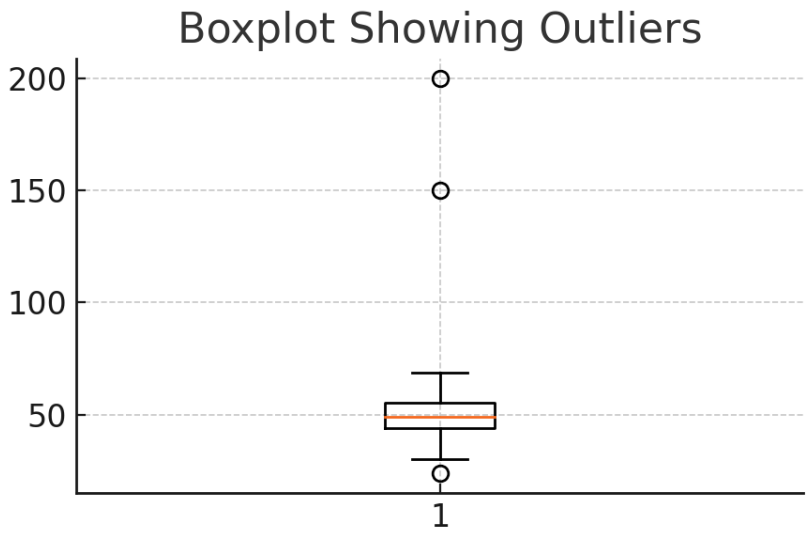
Handling Outliers

Outliers are extreme values that deviate significantly from other observations.

- Detection Methods:
  - Statistical methods (Z-score, IQR method).
  - Visualization techniques (box plots, scatter plots).
- Handling Strategies:
  - Remove (if clearly an error).
  - Transform (log transformation, winsorization: a statistical technique that reduces the impact of outliers in data by replacing extreme values with less extreme values).
  - Use robust models that are less sensitive to outliers.

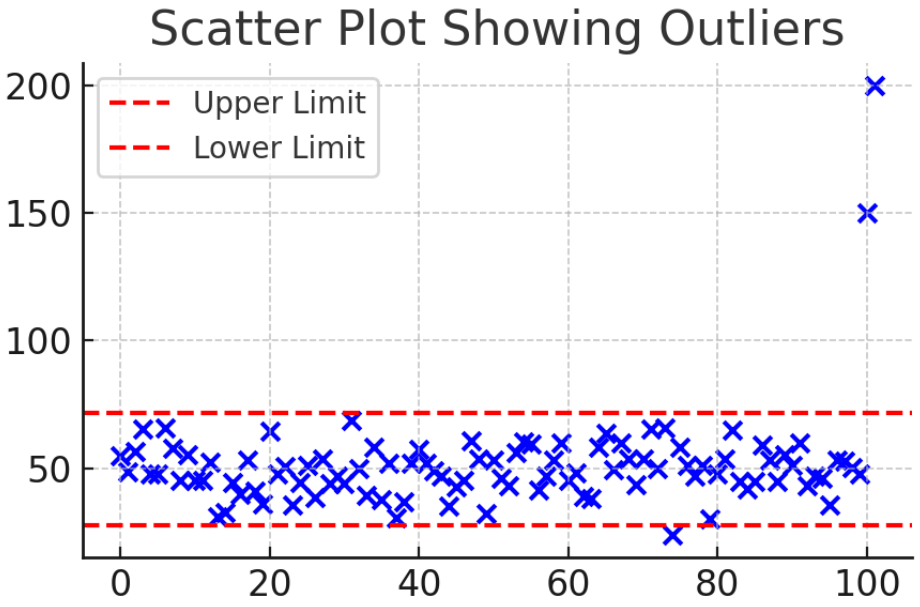
Handling Outliers

- The boxplot shows extreme values (outliers) beyond the whiskers.
- These values can be handled using methods like trimming, transformation, or replacing with median.



Handling Outliers

- The scatter plot highlights outliers that are far from the main cluster.
- These can be detected using statistical methods like Z-score or IQR (Interquartile range).



Automating Data Cleaning

Use libraries like Pandas, NumPy, and OpenRefine for efficient data cleaning.

- Automated tools and techniques:
  - Pandas (`fillna()`, `drop_duplicates()`, `replace()` for cleaning data).
  - Data pipelines in Python (e.g., Scikit-learn).
  - SQL-based data cleaning (using queries for filtering and standardization).
  - Ensures reproducibility and efficiency in large-scale datasets.
- But most of the time, it is data- dependent. So you have to do it on your own.

Automating Data Cleaning

```
import pandas as pd

# Handling Missing Values
df['Age'].fillna(df['Age'].mean(), inplace=True)

# Removing Duplicates Inplace=True to modify original df
df.drop_duplicates(inplace=True)

# Standardizing Data Formats
df['Date'] = pd.to_datetime(df['Date'], format='%Y-%m-
%d')

# Handling Outliers
df = df[df['Salary'] < df['Salary'].quantile(0.95)]
```

Conclusion & Best Practices

- Clean data is the foundation of accurate and reliable analysis.
- Always handle missing values, duplicates, and inconsistencies carefully.
- Use statistical and visualization methods to detect and address outliers.
- Automate repetitive cleaning tasks using Python or SQL.
- Regularly validate and document data cleaning steps.

Data Reduction: Dimensionality Reduction

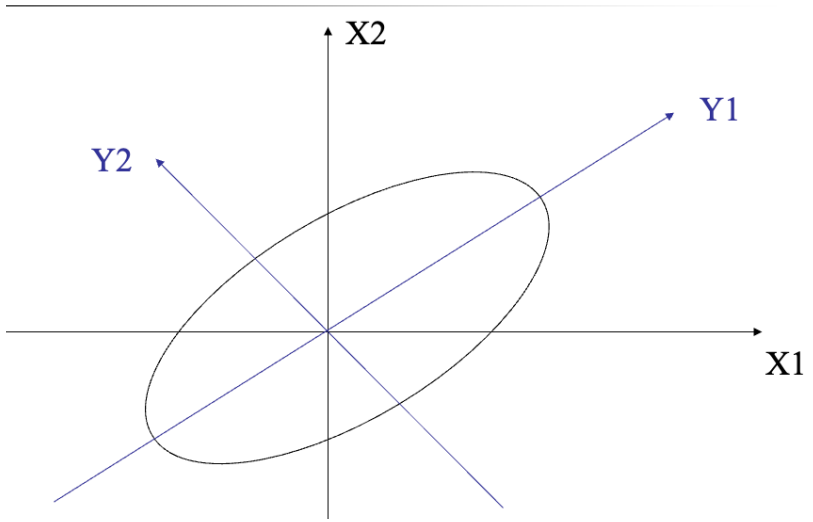
- **Curse of dimensionality**
  - When dimensionality increases, data becomes increasingly sparse
  - Density and distance between points, which is critical to clustering, outlier analysis, becomes less meaningful
  - The possible combinations of subspaces will grow exponentially
- **Dimensionality reduction**
  - Avoid the curse of dimensionality
  - Help eliminate irrelevant features and reduce noise
  - Reduce time and space required in data mining
  - Allow easier visualization

Data Reduction

- **Data reduction**: Obtain a reduced representation of the data set that is much smaller in volume but yet produces the same (or almost the same) analytical results
- Why data reduction? — A database/data warehouse may store terabytes of data. Complex data analysis may take a very long time to run on the complete data set.
- Data reduction strategies
  - Dimensionality reduction, e.g., remove unimportant attributes
    - Wavelet transforms
    - Principal Components Analysis (PCA)
    - t-SNE
    - Autoencoders
    - Feature subset selection, feature creation
  - Numerosity reduction (simply Data volume Reduction)
    - Regression and Log-Linear Models
    - Histograms, clustering, sampling
    - Data cube aggregation
  - Feature Selection (Filter, Wrapper, Embedded Methods)
  - Sampling (Random, Stratified, Systematic Sampling)
  - Data compression

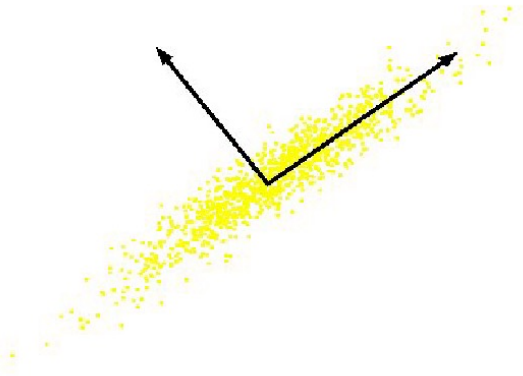
Principal Component Analysis (PCA)

- Find a projection that captures the largest amount of variation in data
- The original data are projected onto a much smaller space, resulting in dimensionality reduction. We find the eigenvectors of the covariance matrix, and these eigenvectors define the new space



Principal Component Analysis (PCA)

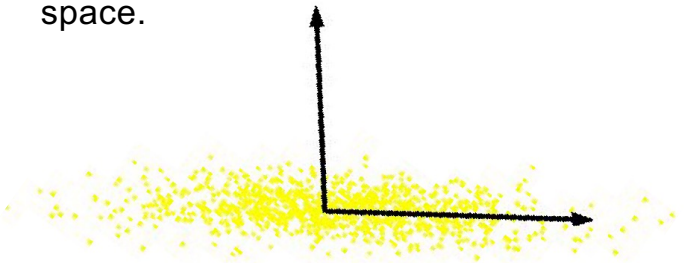
What is PCA: technique for extracting variance structure from high dimensional datasets.



- PCA is an orthogonal projection or transformation of the data into a (possibly lower dimensional) subspace so that the variance of the projected data is maximized.

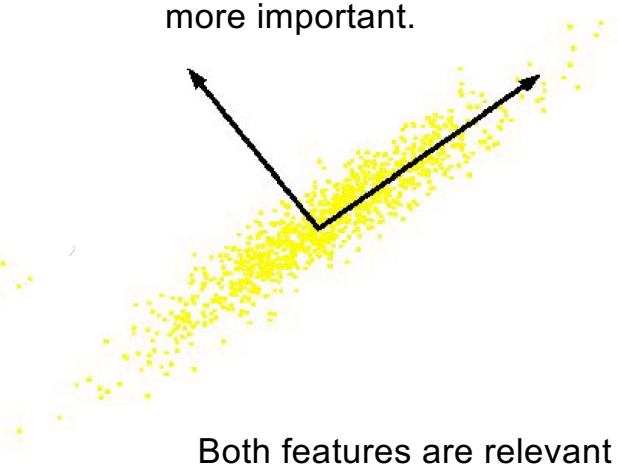
Principal Component Analysis (PCA)

Intrinsically lower dimensional than the dimension of the ambient space.



Only one relevant feature

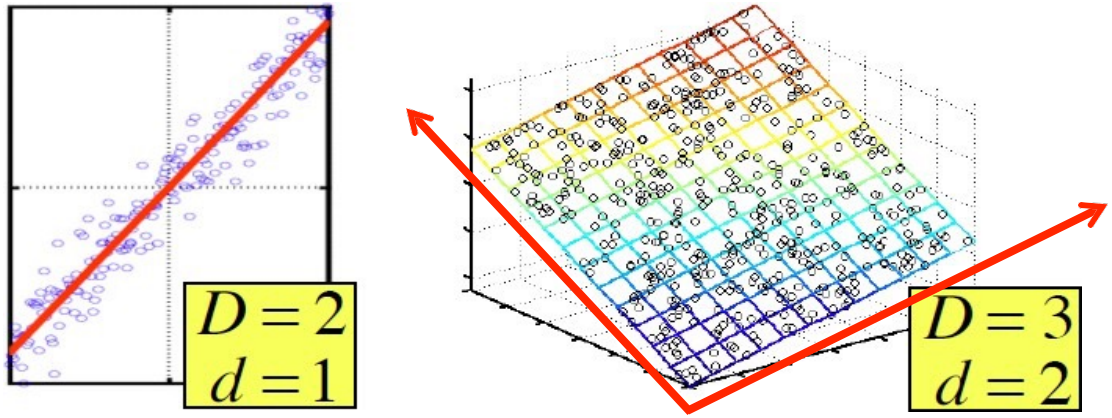
If we rotate data, again only one coordinate is more important.



Both features are relevant

Question: Can we transform the features so that we only need to preserve one latent (hidden) feature?

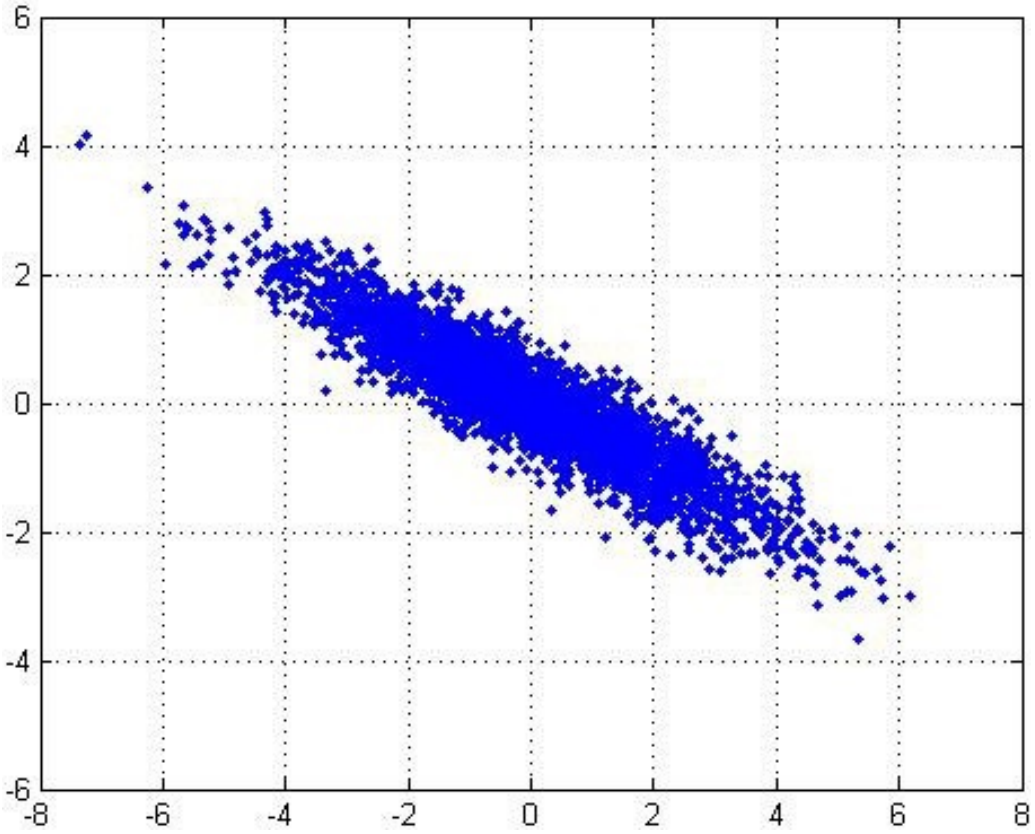
Principal Component Analysis (PCA)



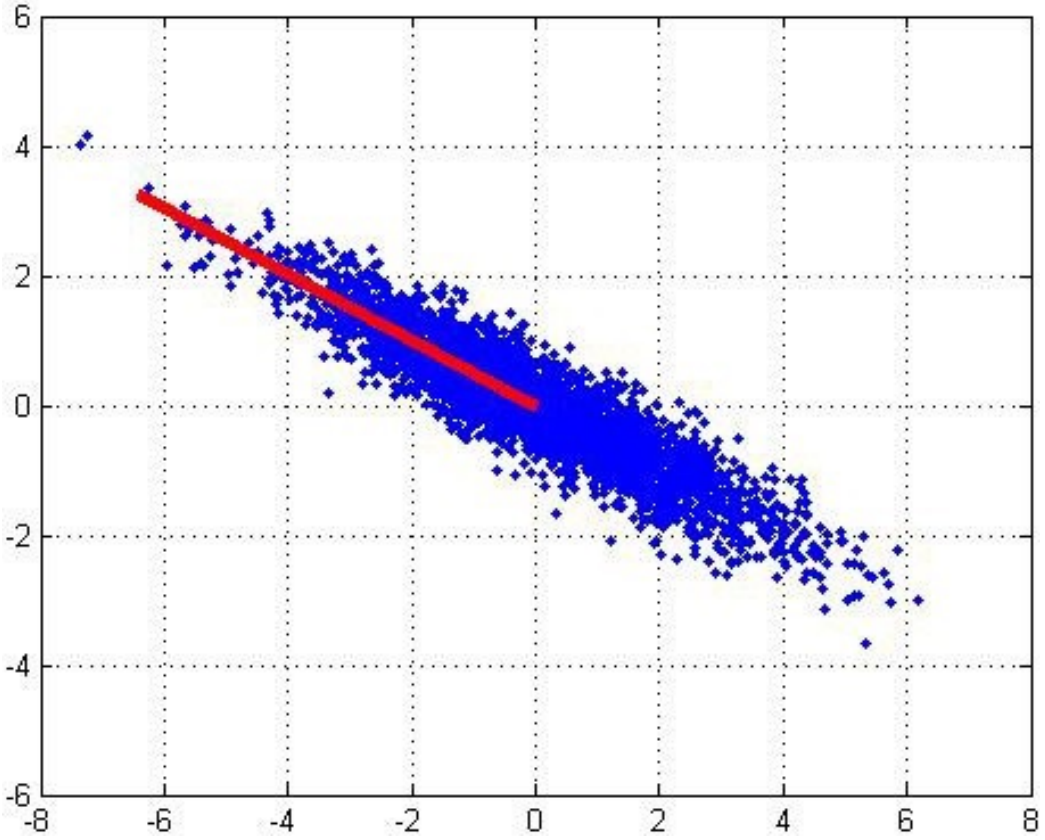
In case where data lies on or near a low d-dimensional linear subspace, axes of this subspace are an effective representation of the data.

Identifying the axes is known as Principal Components Analysis, and can be obtained by using classic matrix computation tools (Eigen or Singular Value Decomposition).

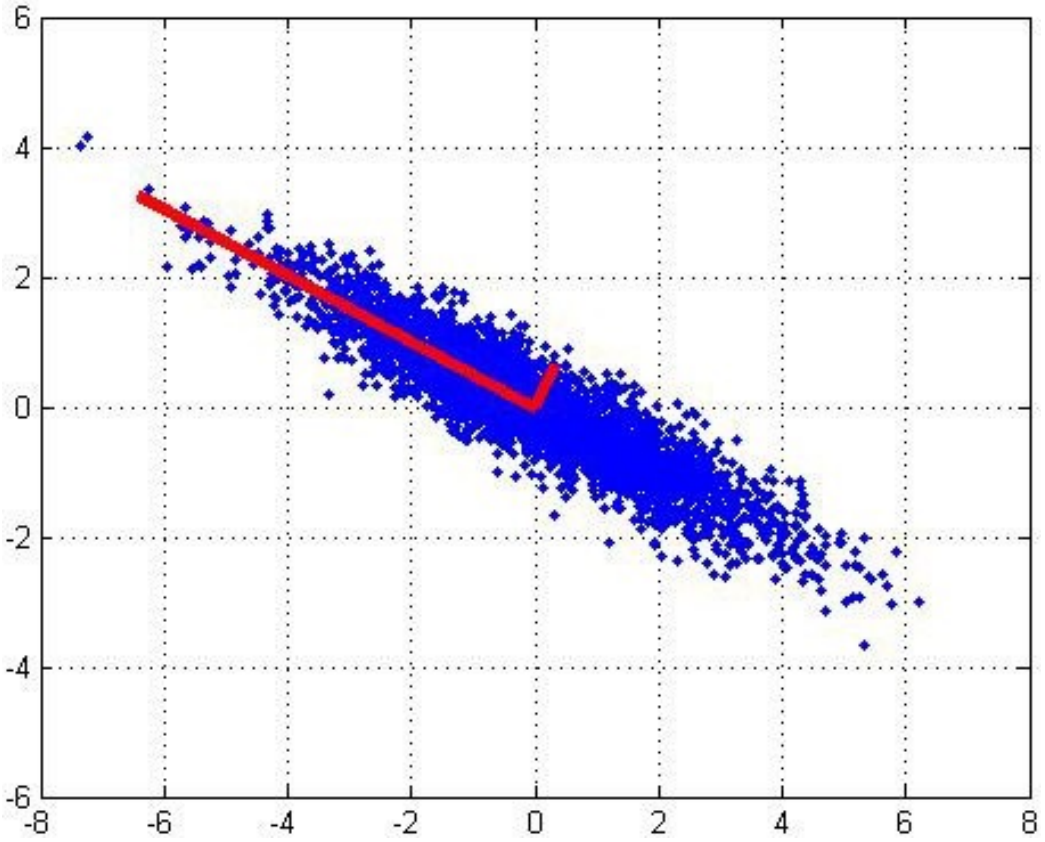
2D Gaussian dataset



1st PCA axis



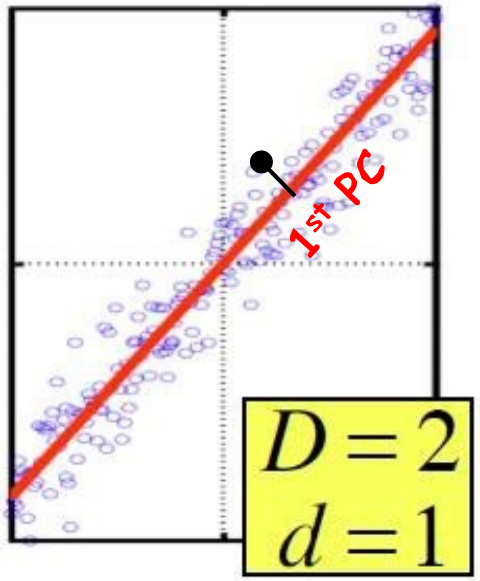
2nd PCA axis



Principal Component Analysis (PCA)

Principal Components (PC) are orthogonal directions that capture most of the variance in the data.

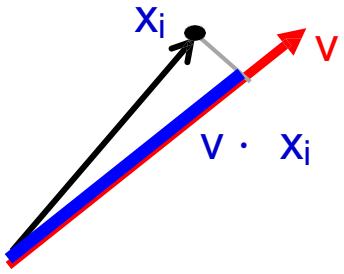
- First PC – direction of greatest variability in data.
- Projection of data points along first PC discriminates data most along any one direction



Quick reminder:

||v||=1, Point x_i (D-dimensional vector)

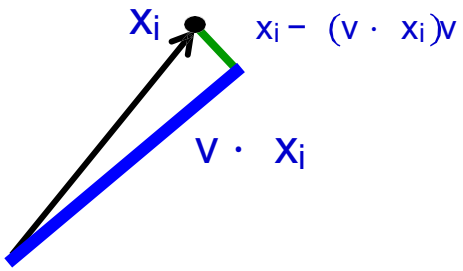
Projection of x_i onto v is $v \cdot x_i$



Principal Component Analysis (PCA)

Principal Components (PC) are orthogonal directions that capture most of the variance in the data.

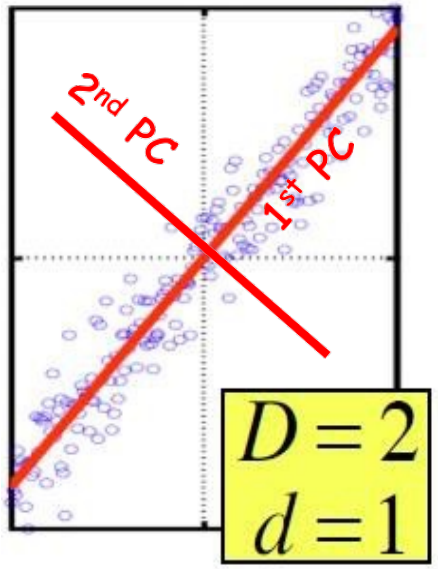
- 1st PC – direction of greatest variability in data.



- 2nd PC – Next orthogonal (uncorrelated) direction of greatest variability

  (remove all variability in first direction, then find next direction of greatest variability)

- And so on …

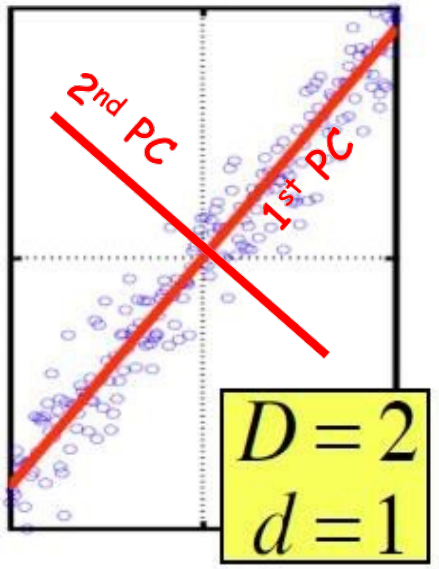


Principal Component Analysis (PCA)

Let $X = [x_1, x_2, \ldots, x_n]$ (columns are the datapoints)

$XX^T v = \lambda v$, so v (the first PC) is the eigenvector of sample correlation/covariance matrix $X X^T$

Sample variance of projection $v^T X X^T v = \lambda v^T v = \lambda$



Thus, the eigenvalue λ denotes the amount of variability captured along that dimension (aka amount of energy along that dimension).

Eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots$

- The 1st PC v_1 is the the eigenvector of the sample covariance matrix $X X^T$ associated with the largest eigenvalue
- The 2nd PC v_2 is the the eigenvector of the sample covariance matrix $X X^T$ associated with the second largest eigenvalue
- And so on …

Principal Component Analysis (PCA)

- Geometrically: centering followed by **rotation.**
- **Key computation**: eigendecomposition of XX^T (closely related to SVD of X).

Original representation

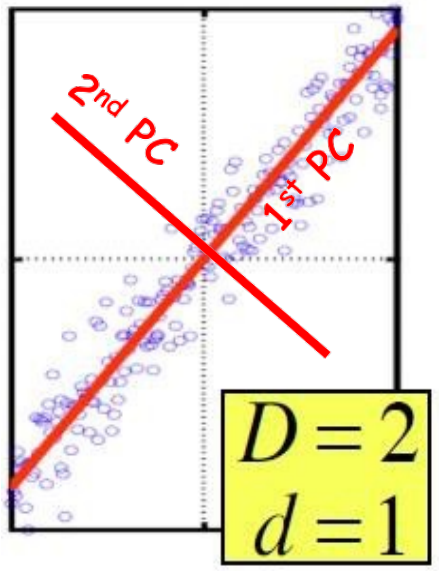
Data point $x_i = (x_i^1, \ldots, x_i^D)$

D-dimensional vector

Transformed representation

projection $(v_1 \cdot x_i, \ldots, v_d \cdot x_i)$

d-dimensional vector



Two Interpretations

E.g., for the first component.

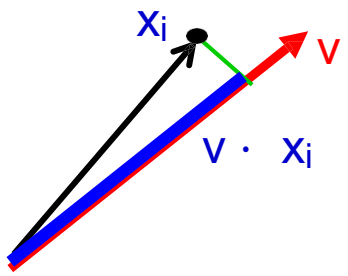
Maximum Variance Direction: 1st PC a vector v such that projection on to this vector capture maximum variance in the data (out of all possible one dimensional projections)

Minimum Reconstruction Error: 1st PC a vector v such that projection on to this vector yields minimum MSE reconstruction

blue^2 + green^2 = black^2

black^2 is fixed (it's just the data)

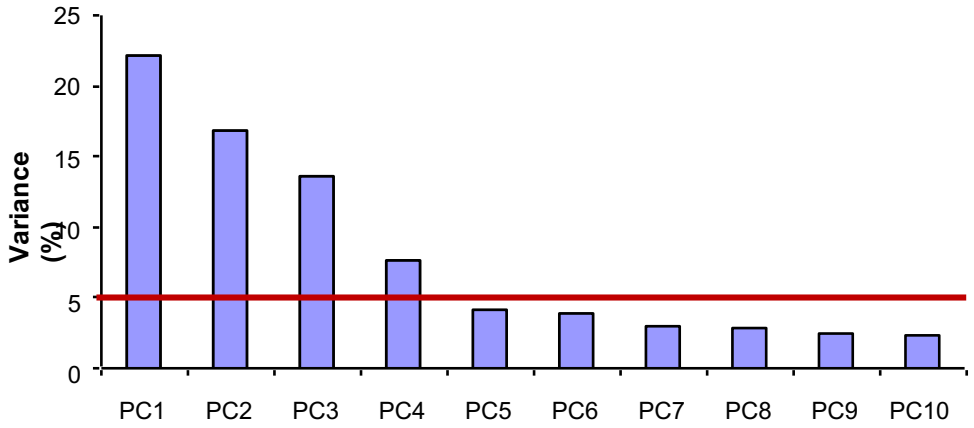
So, maximizing blue^2 is equivalent to minimizing green^2



Principal Component Analysis (PCA)

Only keep data projections onto principal components with **large** eigenvalues

Can *ignore* the components of smaller significance.



Might lose some info, but if eigenvalues are small, do not lose much

Principal Component Analysis (Steps)

- Given *N* data vectors from *n*-dimensions, find $k \leq n$ orthogonal vectors (*principal components*) that can be best used to represent data
  - Normalize input data: Each attribute falls within the same range
  - Compute *k* orthonormal (unit) vectors, i.e., *principal components*
  - Each input data (vector) is a linear combination of the *k* principal component vectors
  - The principal components are sorted in order of decreasing "significance" or strength
  - Since the components are sorted, the size of the data can be reduced by eliminating the *weak components*, i.e., those with low variance (i.e., using the strongest principal components, it is possible to reconstruct a good approximation of the original data)
- Works for numeric data only
- PCA does not look at axis-aligned variance. It looks at variance along arbitrary directions in feature space.

Attribute Subset Selection

- Another way to reduce dimensionality of data
- Redundant attributes
  - Duplicate much or all of the information contained in one or more other attributes
  - E.g., purchase price of a product and the amount of sales tax paid
- Irrelevant attributes
  - Contain no information that is useful for the data mining task at hand
  - E.g., students' ID is often irrelevant to the task of predicting students' GPA

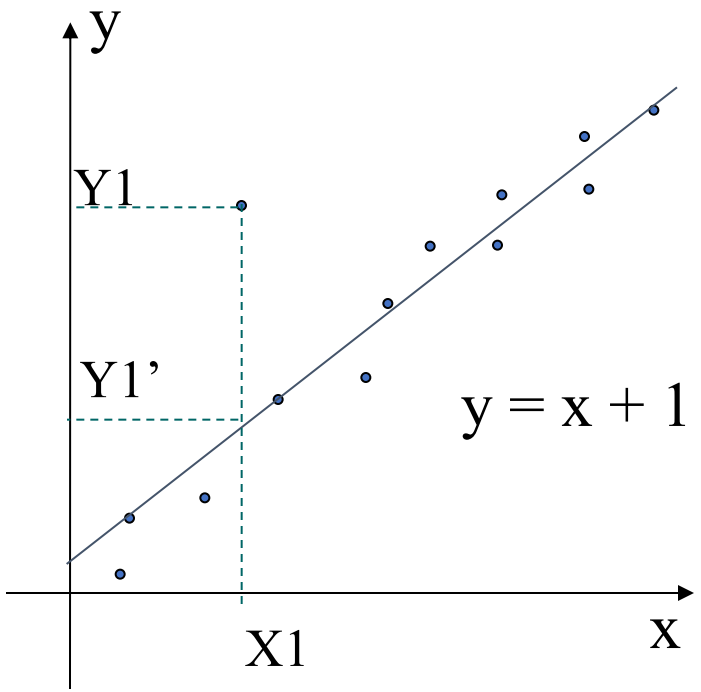
Data Reduction 2: Numerosity Reduction

- Reduce data volume by choosing alternative, *smaller forms* of data representation
- **Parametric methods** (e.g., regression)
  - Assume the data fits some model, estimate model parameters, store only the parameters, and discard the data (except possible outliers)
  - Ex.: Log-linear models—obtain value at a point in *m*-D space as the product on appropriate marginal subspaces
- **Non-parametric** methods
  - Do not assume models
  - Major families: histograms, clustering, sampling, …

Parametric Data Reduction: Regression and Log-Linear Models

- **Linear regression**
  - Data modeled to fit a straight line
  - Often uses the least-square method to fit the line
  - Example: Let's say weight is roughly dependent on height $w = n * h + m + noise$. So we can just use height and remove weight from data.
- **Multiple regression**
  - Allows a response variable Y to be modeled as a linear function of multidimensional feature vector
- **Log-linear model for categorial data**
  - Approximates discrete multidimensional probability distributions
  - Instead of storing full joint probability tables (which grow exponentially), log-linear models approximate them using fewer parameters.

Regression Analysis



- Regression analysis: A collective name for techniques for the modeling and analysis of numerical data consisting of values of a ***dependent variable*** (also called ***response variable*** or *measurement*) and of one or more *independent variables* (aka. ***explanatory variables*** or ***predictors***)
- The parameters are estimated so as to give a "**best fit**" of the data
- Most commonly the best fit is evaluated by using the ***least squares method***, but other criteria have also been used
- Used for prediction (including forecasting of time-series data), inference, hypothesis testing, and modeling of causal relationships

Regress Analysis and Log-Linear Models

- Linear regression: $Y = w X + b$
  - Two regression coefficients, *w* and *b,* specify the line and are to be estimated by using the data at hand
  - Using the least squares criterion to the known values of $Y_1, Y_2, \ldots, X_1, X_2, \ldots$.
- Multiple regression: $Y = b_0 + b_1 X_1 + b_2 X_2$
  - Many nonlinear functions can be transformed into the above
- Log-linear models:
  - Approximate discrete multidimensional probability distributions
  - Estimate the probability of each point (tuple) in a multi-dimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations
  - Useful for dimensionality reduction and data smoothing

Log-Linear Models

- Reduces the complexity of **categorical datasets** by capturing key interactions.
- Approximation: Models the joint distribution of categorical variables without explicitly storing all frequencies in the contingency table.

Log-Linear Models

- Example:

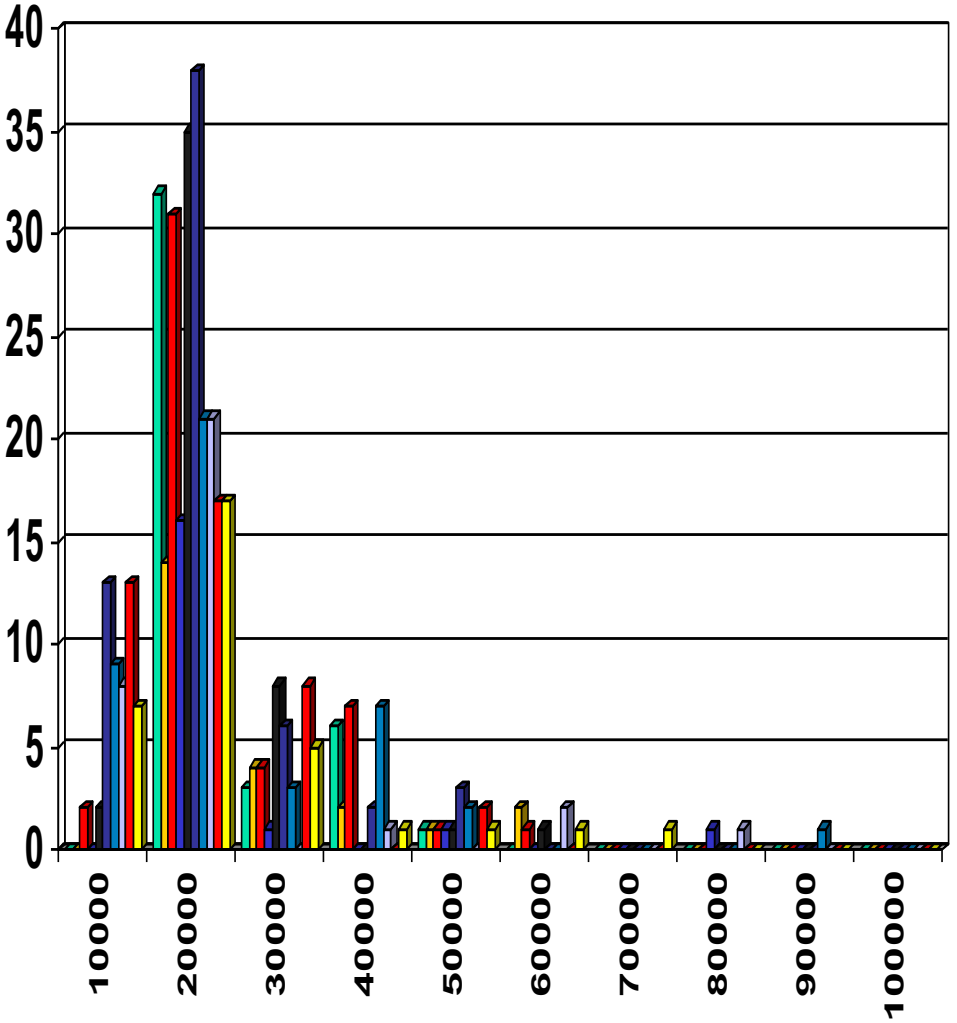
Gender	Age	Group	Purchased Item Frequency
Male	Young	Electronics	50
Male	Old	Electronics	30
Female	Young	Clothing	70
Female	Old	Clothing	40

$$\log(Frequency) = \mu + \lambda(Gender) + \lambda(Age\ Group) + \lambda(Purchased\ Item) + \lambda(Gender, Age\ Group)$$

- Basically, it is relating the categorial data to the log of frequency using lamda as a function.

Histogram Analysis

- Divide data into buckets and store average (sum) for each bucket
- Partitioning rules:
  - Equal-width: equal bucket range
  - Equal-frequency (or equal-depth)



Clustering

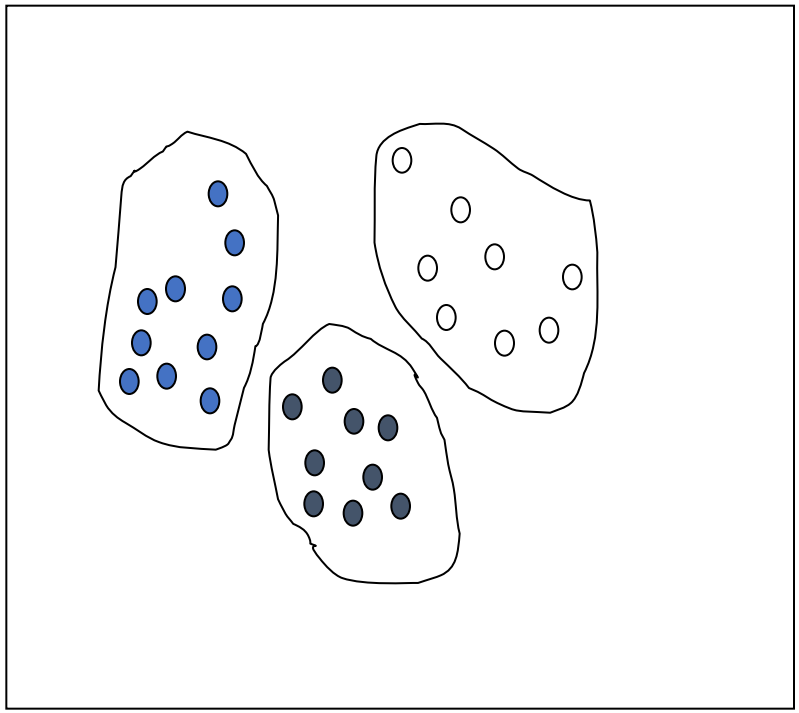
- Partition data set into clusters based on similarity, and store cluster representation (e.g., centroid and diameter) only
- Can be very effective if data is clustered but not if data is "smeared"
- Can have hierarchical clustering and be stored in multi-dimensional index tree structures
- There are many choices of clustering definitions and clustering algorithms

Sampling

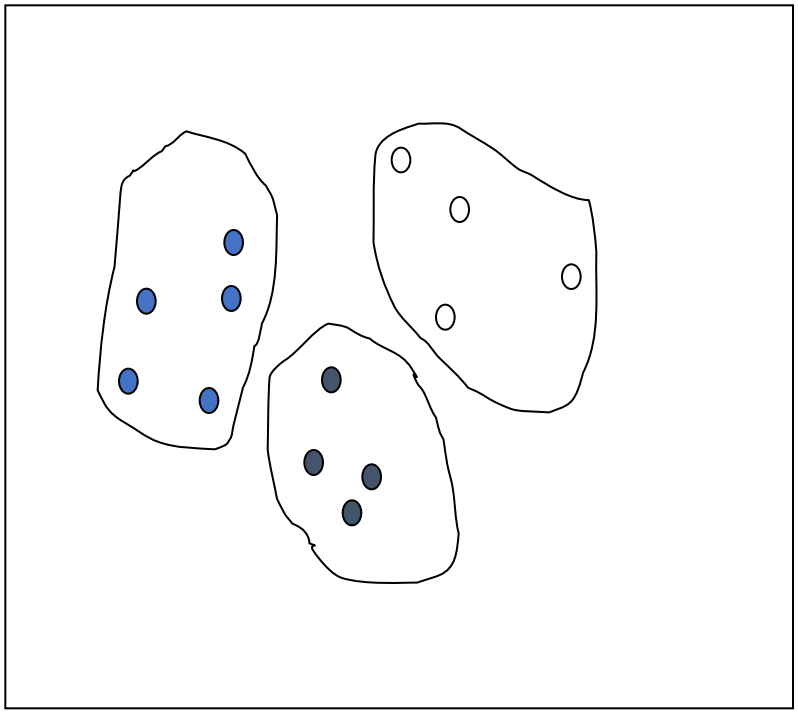
- Sampling: obtaining a small sample *s* to represent the whole data set *N*
- Allow a mining algorithm to run in complexity that is potentially sub-linear to the size of the data
- Key principle: Choose a representative subset of the data
  - Simple random sampling may have very poor performance in the presence of skew
  - Develop adaptive sampling methods, e.g., stratified sampling (probability sampling method)

Sampling: Cluster or Stratified Sampling

Raw Data



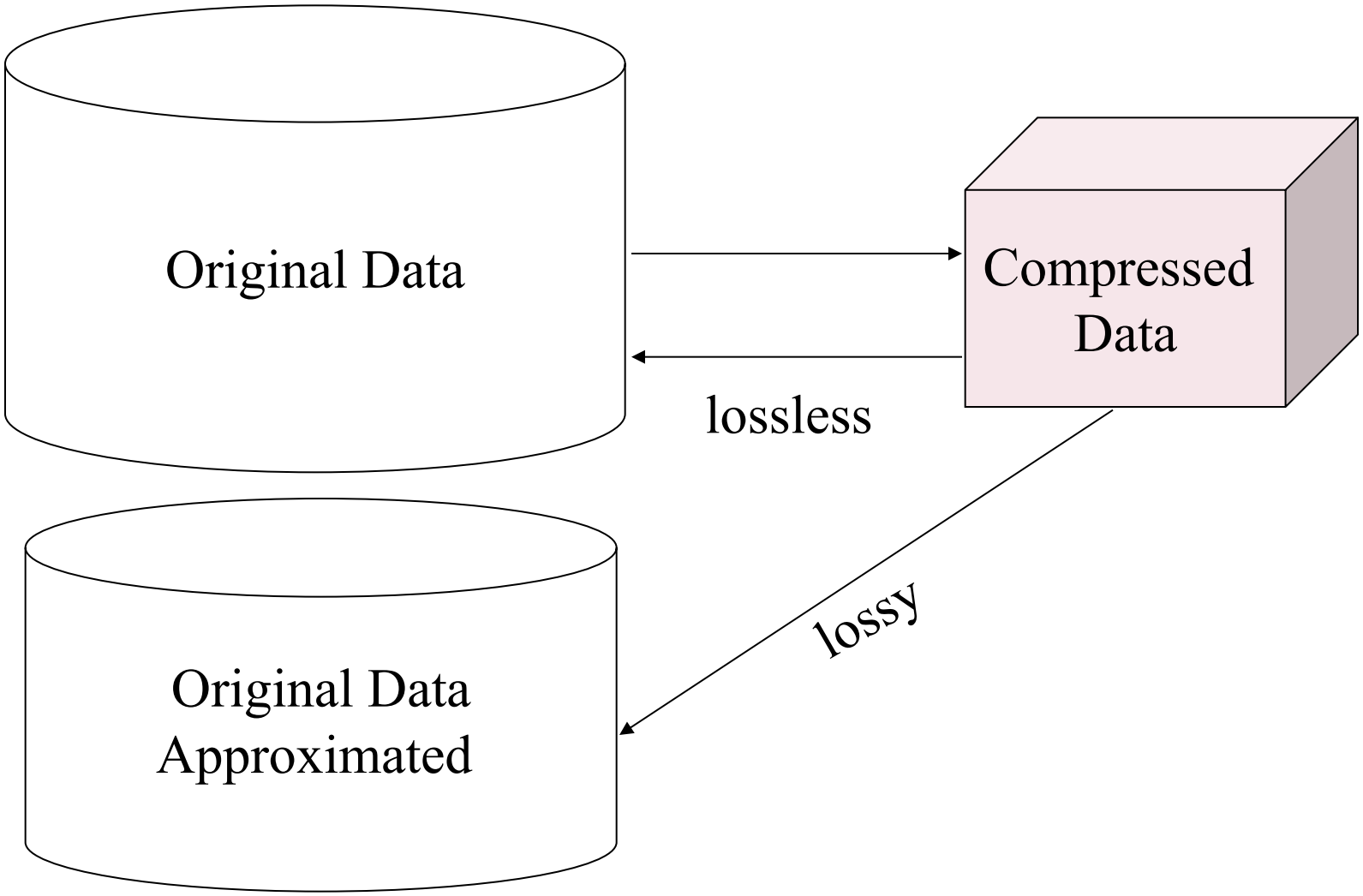
Cluster/Stratified Sample



Data Reduction 3: Data Compression

- String compression
  - There are extensive theories and well-tuned algorithms
  - Typically lossless
- Audio/video compression
  - Typically lossy compression, with progressive refinement
  - Sometimes small fragments of signal can be reconstructed without reconstructing the whole
- Time sequence is not audio
  - Typically short and vary slowly with time
- Dimensionality and numerosity reduction may also be considered as forms of data compression

Data Compression



Data Transformation

- A function that maps the entire set of values of a given attribute to a new set of replacement values s.t. each old value can be identified with one of the new values
- Methods
  - Smoothing: Remove noise from data
  - Attribute/feature construction
    - New attributes constructed from the given ones
  - Aggregation: Summarization, data cube construction
  - Normalization: Scaled to fall within a smaller, specified range
    - min-max normalization
    - z-score normalization
    - normalization by decimal scaling
  - Binning & Discretization (Equal-width, Equal-frequency), Concept hierarchy climbing
  - Encoding Categorical Variables (One-Hot, Label Encoding)

Normalization

- **Min-max normalization**: to [new_min$_A$, new_max$_A$]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to $\frac{73{,}600 - 12{,}000}{98{,}000 - 12{,}000}(1.0 - 0) + 0 = 0.716$

- **Z-score normalization** (μ: mean, σ: standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Ex. Let μ = 54,000, σ = 16,000. Then $\frac{73{,}600 - 54{,}000}{16{,}000} = 1.225$

- **Normalization by decimal scaling**

$$v' = \frac{v}{10^j} \quad \text{Where } j \text{ is the smallest integer such that Max}(|v'|) < 1$$

Discretization

- Three types of attributes
  - Nominal—values from an unordered set, e.g., color, profession
  - Ordinal—values from an ordered set, e.g., military or academic rank
  - Numeric—real numbers, e.g., integer or real numbers
- Discretization: Divide the range of a continuous attribute into intervals
  - Interval labels can then be used to replace actual data values
  - Reduce data size by discretization
  - Supervised vs. unsupervised
  - Split (top-down) vs. merge (bottom-up)
  - Discretization can be performed recursively on an attribute
  - Prepare for further analysis, e.g., classification

Data Discretization Methods

- Typical methods: All the methods can be applied recursively
  - Binning
    - Top-down split, unsupervised
  - Histogram analysis
    - Top-down split, unsupervised
  - Clustering analysis (unsupervised, top-down split or bottom-up merge)
  - Decision-tree analysis (supervised, top-down split)
  - Correlation (e.g., χ^2) analysis (unsupervised, bottom-up merge)

66

Simple Discretization: Binning

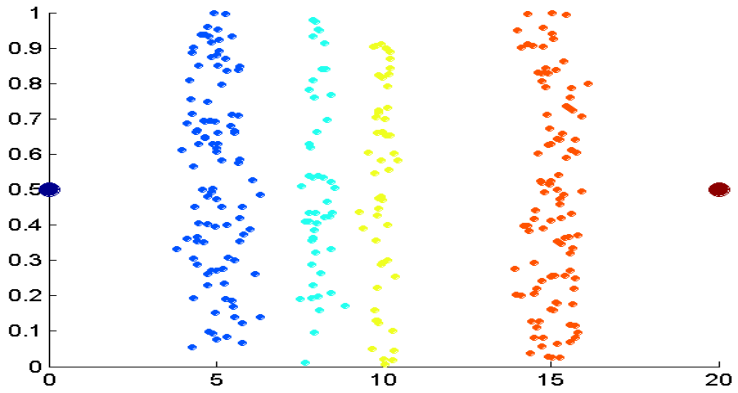
- Equal-width (distance) partitioning
  - Divides the range into *N* intervals of equal size: uniform grid
  - if *A* and *B* are the lowest and highest values of the attribute, the width of intervals will be: $W = (B - A)/N$.
  - The most straightforward, but outliers may dominate presentation
  - Skewed data is not handled well
- Equal-depth (frequency) partitioning
  - Divides the range into *N* intervals, each containing approximately same number of samples
  - Good data scaling
  - Managing categorical attributes can be tricky

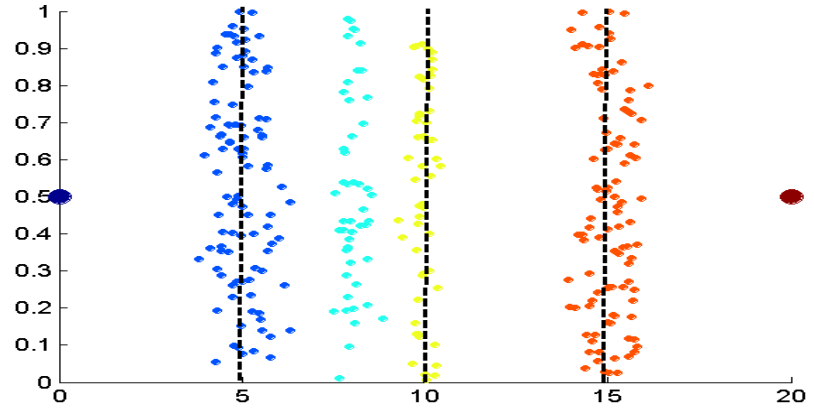
Binning Methods for Data Smoothing

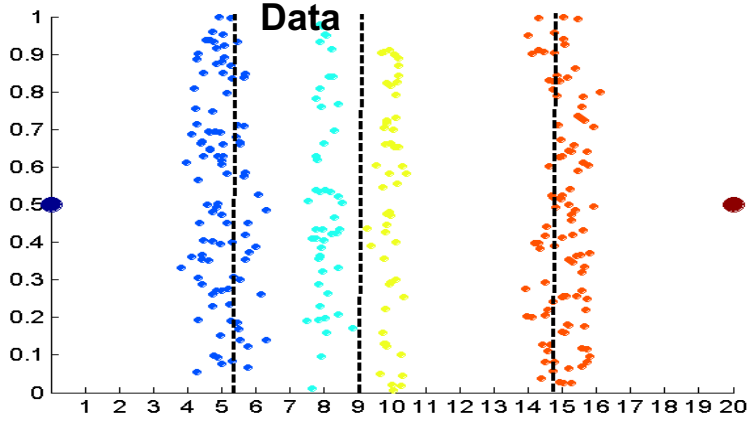
- Sorted data for price (in dollars): 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34

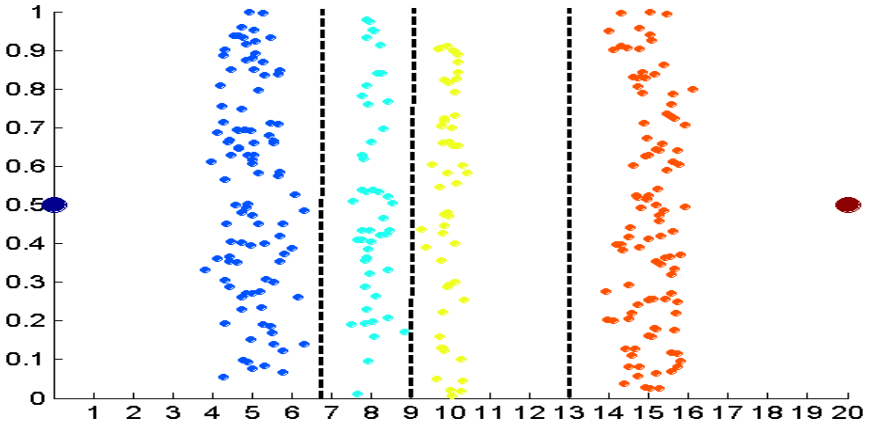
* Partition into equal-frequency (**equi-depth**) bins:
  - Bin 1: 4, 8, 9, 15
  - Bin 2: 21, 21, 24, 25
  - Bin 3: 26, 28, 29, 34
* Smoothing by **bin means**:
  - Bin 1: 9, 9, 9, 9
  - Bin 2: 23, 23, 23, 23
  - Bin 3: 29, 29, 29, 29
* Smoothing by **bin boundaries**:
  - Bin 1: 4, 4, 4, 15
  - Bin 2: 21, 21, 25, 25
  - Bin 3: 26, 26, 26, 34

Discretization Without Using Class Labels (Binning vs. Clustering)









Equal frequency (binning)

K-means clustering leads to better results

69

Discretization by Classification & Correlation Analysis

- Classification (e.g., decision tree analysis)
  - Supervised: Given class labels, e.g., cancerous vs. benign
  - Using *entropy* to determine split point (discretization point)
  - Top-down, recursive split
- Correlation analysis (e.g., Chi-merge: χ^2-based discretization)
  - Supervised: use class information
  - Bottom-up merge: find the best neighboring intervals (those having similar distributions of classes, i.e., low χ^2 values) to merge
  - Merge performed recursively, until a predefined stopping condition

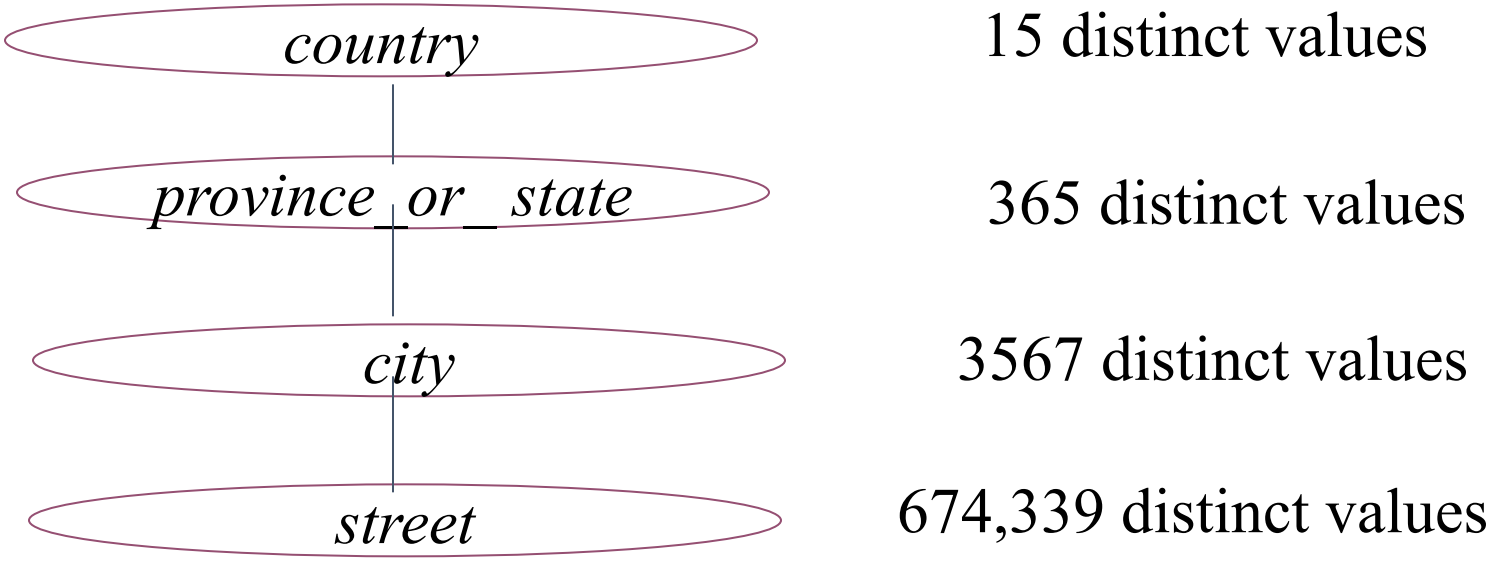
70

Concept Hierarchy Generation

- **Concept hierarchy** organizes concepts (i.e., attribute values) hierarchically and is usually associated with each dimension in a data warehouse
- Concept hierarchies facilitate drilling and rolling in data warehouses to view data in multiple granularity
- Concept hierarchy formation: Recursively reduce the data by collecting and replacing low level concepts (such as numeric values for *age*) by higher level concepts (such as *youth, adult*, or *senior*)
- Concept hierarchies can be explicitly specified by domain experts and/or data warehouse designers
- Concept hierarchy can be automatically formed for both numeric and nominal data. For numeric data, use discretization methods shown.

Automatic Concept Hierarchy Generation

- Some hierarchies can be automatically generated based on the analysis of the number of distinct values per attribute in the data set
  - The attribute with the most distinct values is placed at the lowest level of the hierarchy
  - Exceptions, e.g., weekday, month, quarter, year



Conclusion & Best Practices

- Always clean, reduce, and transform data before analysis.
- Use domain knowledge for effective feature selection.
- Validate transformed data to ensure consistency and accuracy.
- Automate repetitive tasks using scripts or pipelines.

Thank You

- All my slides/notes excluding third party material are licensed by various authors including myself under https://creativecommons.org/licenses/by-nc/4.0/
- Book- Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Machine Learning: Fundamental Concepts and Algorithms, 2nd Edition, Cambridge University Press, March 2020. ISBN: 978-1108473989
- Dr. Dhaval Patel
- Jiawei Han, Micheline Kamber, and Jian Pei Notes
- https://dataminingbook.info/

Bias-variance Trade-off

Generalization

Training set (labels known)

Test set (labels unknown)

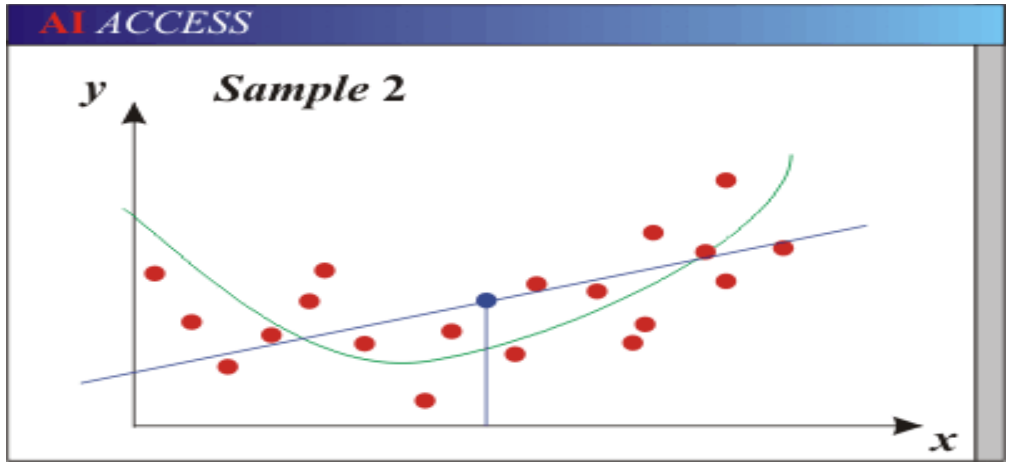
- How well does a learned model generalize from the data it was trained on to a new test set?

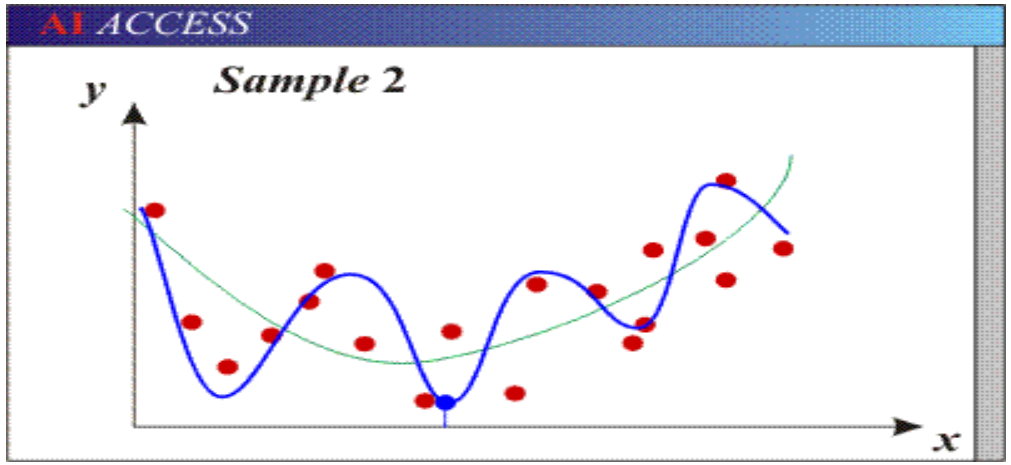
Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance:** how much models estimated from different training sets differ from each other
- **Underfitting:** model is too “simple” to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting:** model is too “complex” and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

Slide credit: L. Lazebnik

Bias-Variance Trade-off



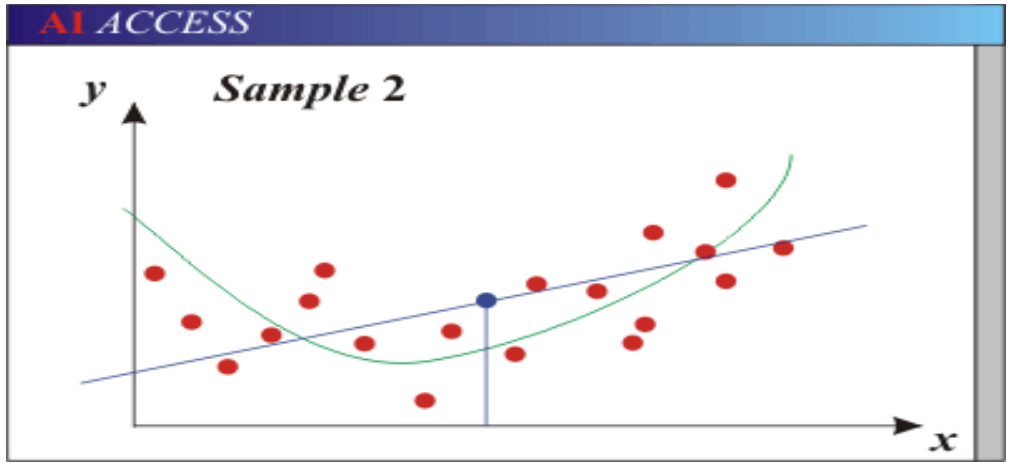


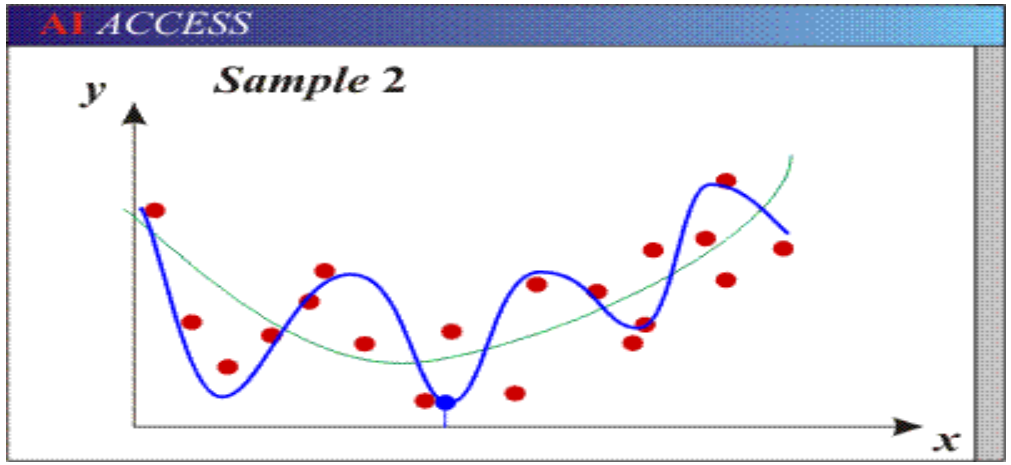
Models with too few parameters are inaccurate because of a large bias (not enough flexibility).

Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

Slide credit: D. Hoiem

Bias-Variance Trade-off





- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).
- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

Slide credit: D. Hoiem

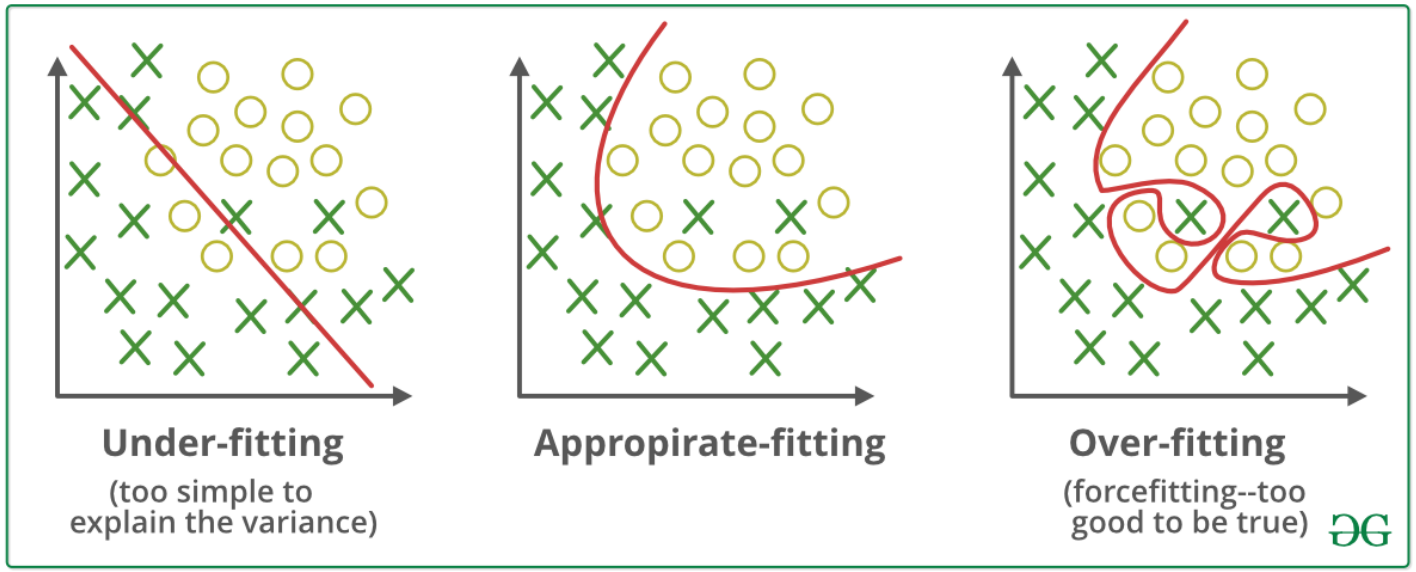
Bias-Variance Trade-off

$$E(MSE) = noise^2 + bias^2 + variance$$

Unavoidable error

Error due to incorrect assumptions

Error due to variance of training samples

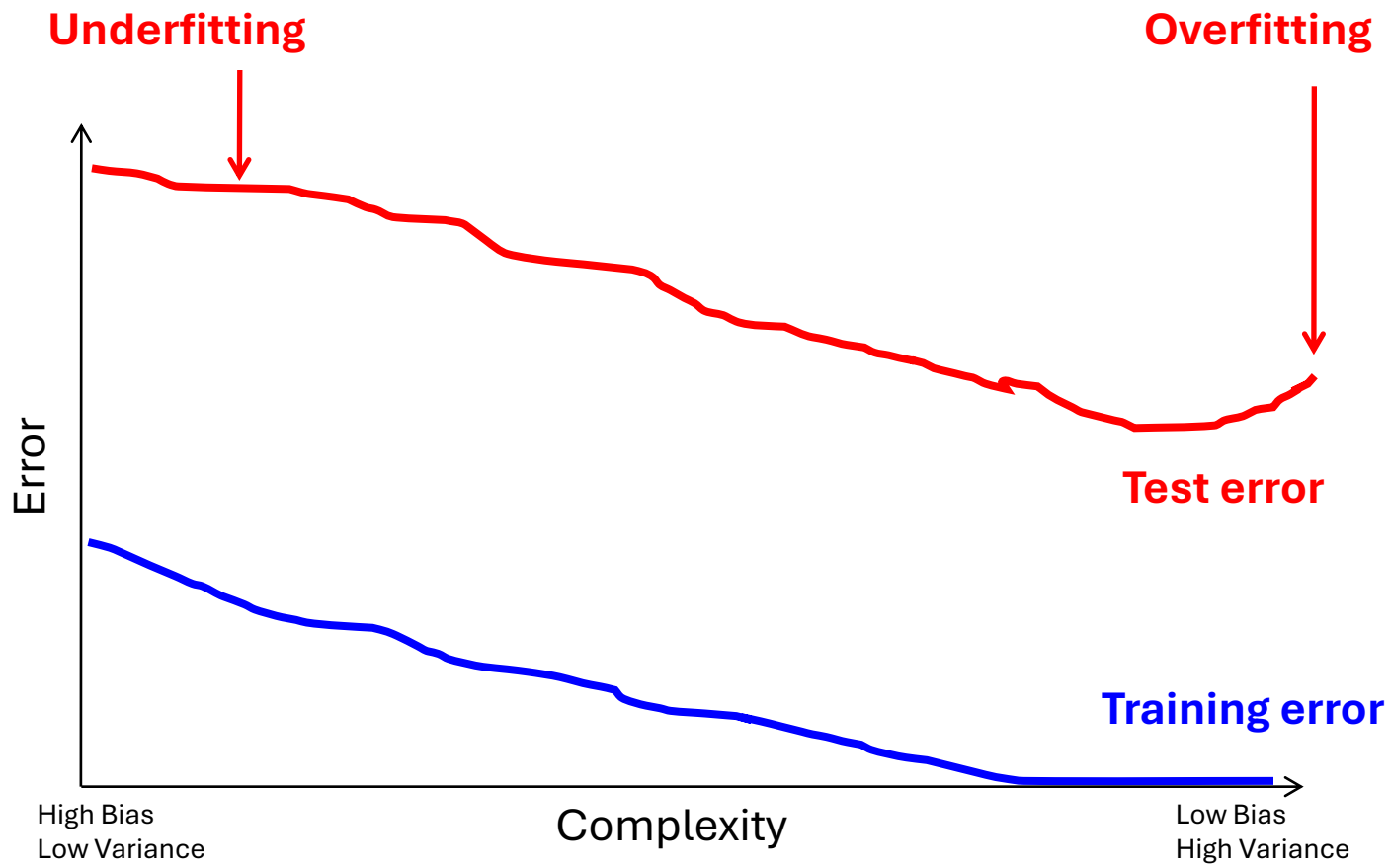


See the following for explanations of bias-variance (also Bishop's "Neural Networks" book):

•http://www.inf.ed.ac.uk/teaching/courses/mlsc/Notes/Lecture4/BiasVariance.pdf

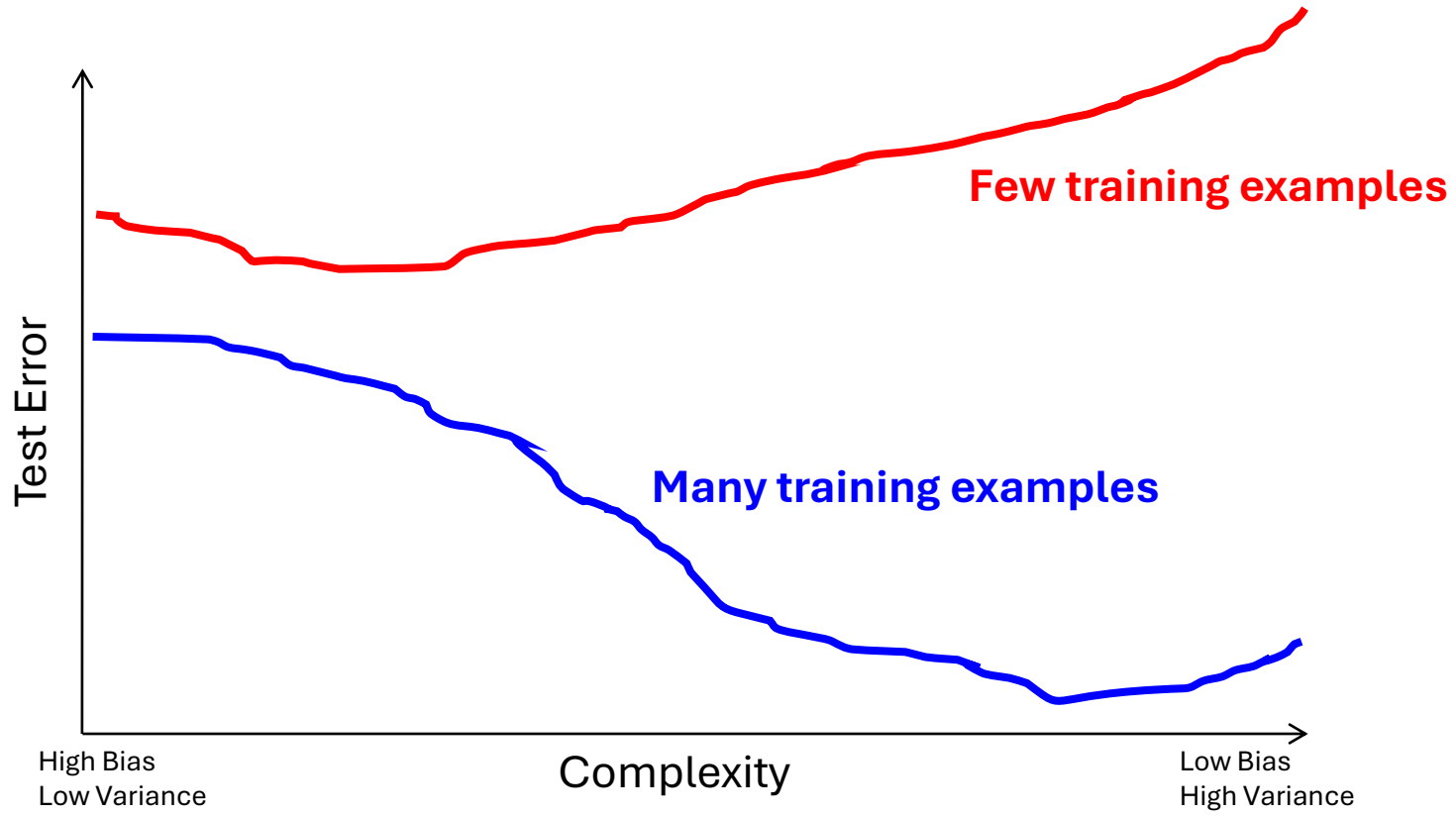
Image credit: geeksforgeeks.com
Slide credit: D. Hoiem

Bias-variance tradeoff



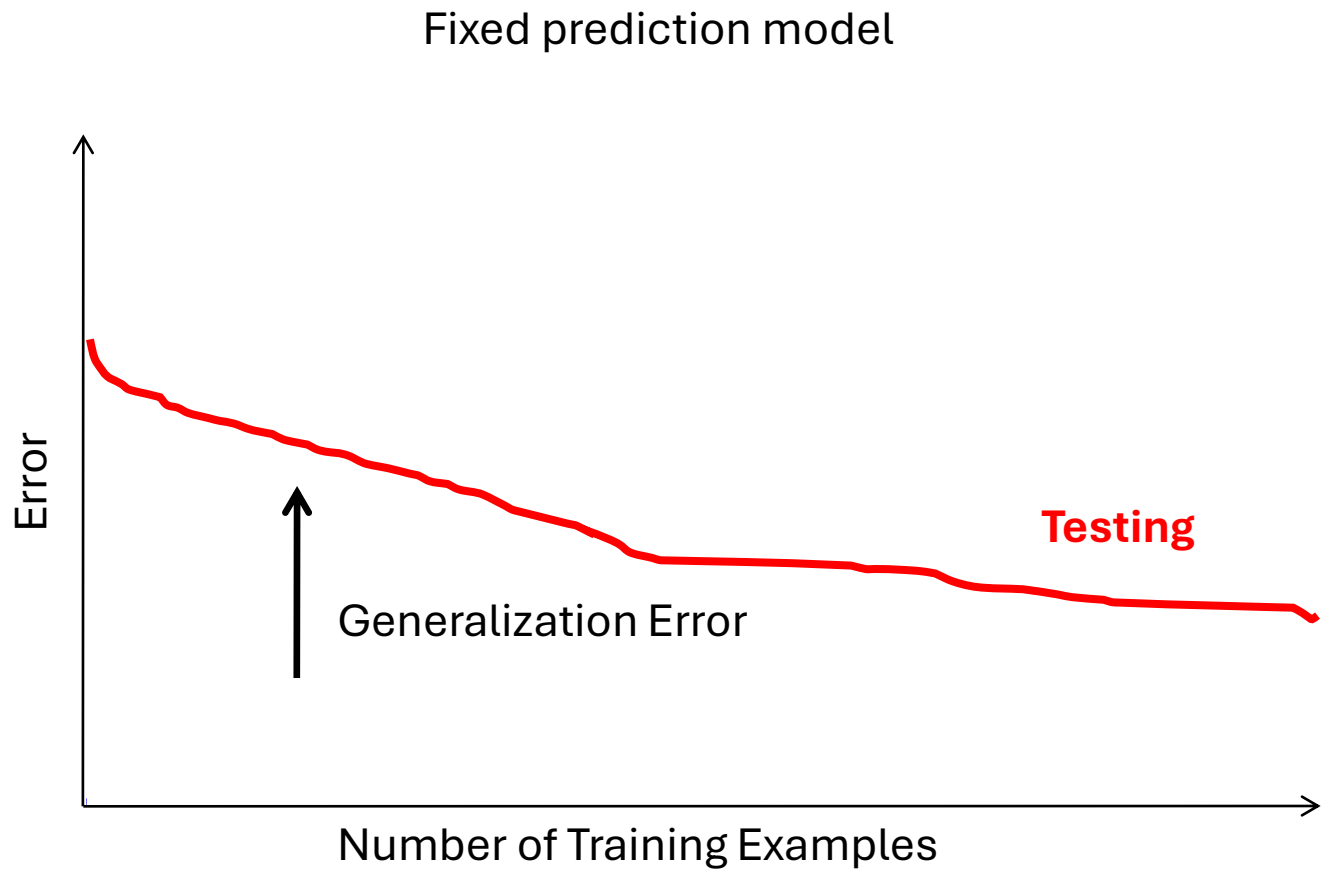
Slide credit: D. Hoiem

Bias-variance tradeoff



Slide credit: D. Hoiem

Effect of Training Size



Slide credit: D. Hoiem

The perfect classification algorithm

- Objective function: encodes the right loss for the problem
- Parameterization: makes assumptions that fit the problem
- Regularization: right level of regularization for amount of training data
- Training algorithm: can find parameters that maximize objective on training set
- Inference algorithm: can solve for objective function in evaluation

Slide credit: D. Hoiem

Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize

- Three kinds of error
  - Inherent: unavoidable
  - Bias: due to over-simplifications
  - Variance: due to inability to perfectly estimate parameters from limited data

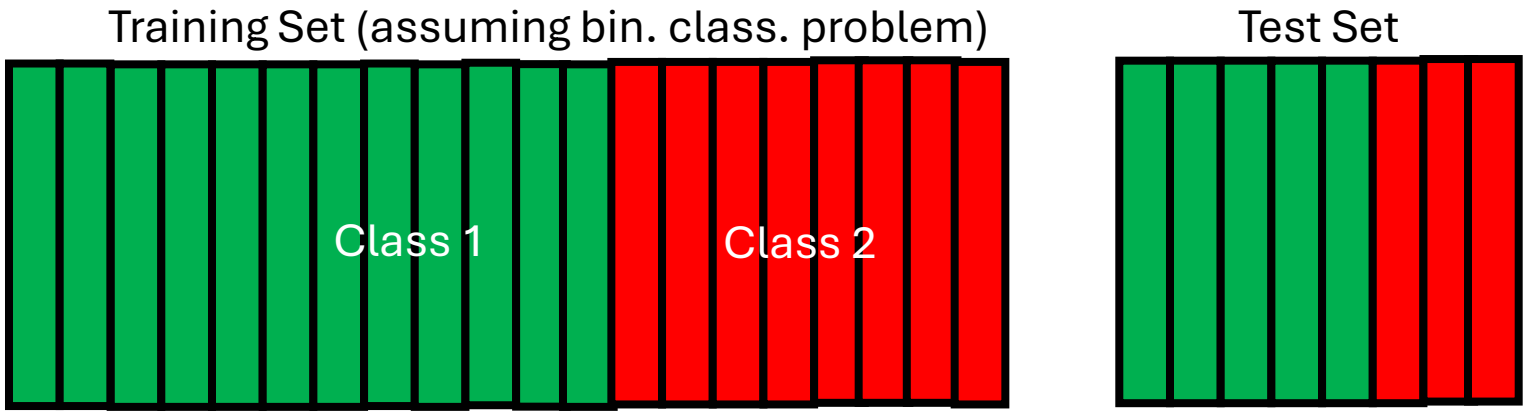
Slide credit: D. Hoiem

How to reduce variance?

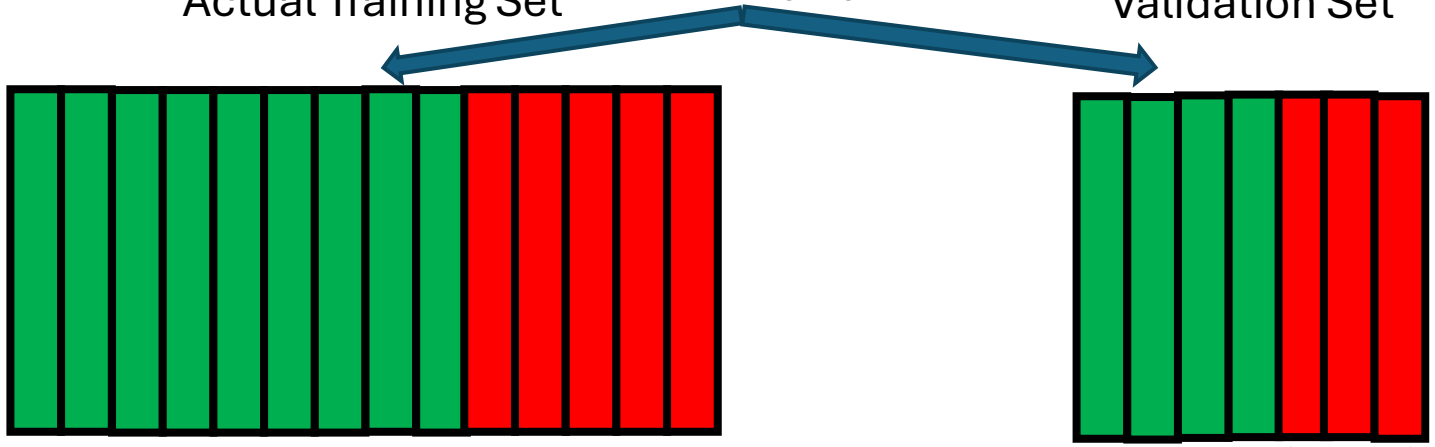
- Choose a simpler classifier
- Cross-validate the parameters
- Get more training data

Cross-Validation

No peeking while building the model



Note: Not just h.p. selection; we can also use CV to pick the best ML model from a set of different ML models (e.g., say we have to pick between two models we may have trained - LwP and nearest neighbors. Can use CV to choose the better one.



Randomly split the original training data into actual training set and validation set. Using the actual training set, train several times, each time using a different value of the hyperparam. Pick the hyperparam value that gives best accuracy on the validation set

What if the random split is unlucky (i.e., validation data is not like test data)?

If you fear an unlucky split, try multiple splits. Pick the hyperparam value that gives the best average CV accuracy across all such splits. If you are using N splits, this is called N–fold cross validation

Parametric and Non-parametric

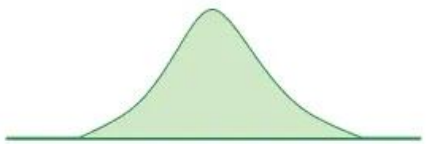
- Parametric and non-parametric tests are named based on whether they require assumptions about the underlying population parameters (like mean and variance). **Parametric tests** rely on specific distributional assumptions (usually normal distribution) and specific parameters, while **non-parametric tests** make no such assumptions, acting as distribution-free methods

Parametric vs. Non-Parametric Tests

Parametric Tests

Assumptions :

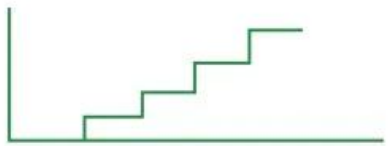
- Normality
- Homogeneity of variance
- Independence



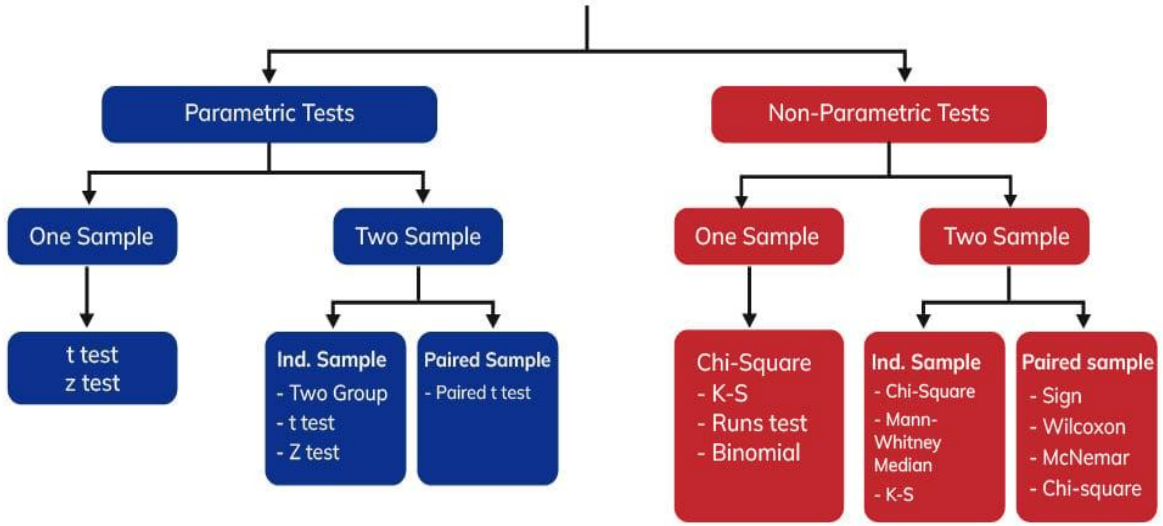
Non-Parametric Tests

When to Use :

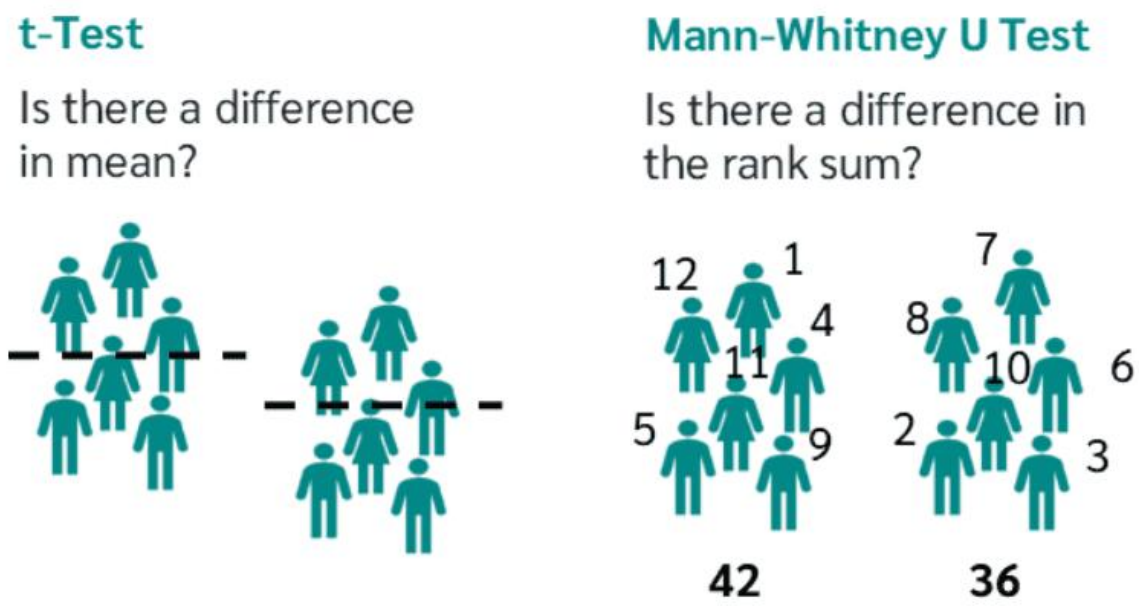
- Data is not normally distributed
- Small sample size
- Ordinal/nominal data
- Presence of outliers



Parametric & Non-Parametric Test



Mann-Whitney U Test



- **Null hypothesis:** There is no difference (in terms of central tendency) between the two groups in the population.
- **Alternative hypothesis:** There is a difference (with respect to the central tendency) between the two groups in the population.

Gender	Reaction time	Rang
female	34	2
female	36	4
female	41	7
female	43	9
female	44	10
female	37	5
male	45	11
male	33	1
male	35	3
male	39	6
male	42	8

Calculation of the rank sums

$T_1 = 2 + 4 + 7 + 9 + 10 + 5 = 37$

$T_2 = 11 + 1 + 3 + 6 + 8 = 29$

Female

Number of cases $n_1 = 6$ Rank sum $T_1 = 37$

$$U_1 = n_1 \cdot n_2 + \frac{n_1 \cdot (n_1 + 1)}{2} - T_1$$
$$= 6 \cdot 5 + \frac{6 \cdot (6 + 1)}{2} - 37$$
$$= 14$$

Male

Number of cases $n_2 = 5$ Rank sum $T_2 = 29$

$$U_2 = n_1 \cdot n_2 + \frac{n_2 \cdot (n_2 + 1)}{2} - T_2$$
$$= 6 \cdot 5 + \frac{5 \cdot (5 + 1)}{2} - 29$$
$$= 16$$

U-Wert

$$U = min(U_1, U_2) = min(14, 16) = 14$$

Expected value of U

$$\mu_U = \frac{n_1 \cdot n_2}{2} = \frac{6 \cdot 5}{2} = 15$$

Standard error of U

$$\sigma_U = \sqrt{\frac{n_1 \cdot n_2 \cdot (n_1 + n_2 + 1)}{12}} = \sqrt{\frac{6 \cdot 5 \cdot (6 + 5 + 1)}{12}} = 5.4$$

z-value

$$z = \frac{U - \mu_U}{\sigma_U} = \frac{14 - 15}{5{,}4772} = -0.1825$$

P-value = 0.855 (two tail) , can not reject the null hypothesis

t-test

Some more materials on T-test

Steps for Significance Testing

1. Set alpha (p level).
2. State hypotheses, Null and Alternative.
3. Calculate the test statistic (sample value).
4. Find the critical value of the statistic.
5. State the decision rule.
6. State the conclusion.

t-test

- t –test is about means: distribution and evaluation for group distribution
- Withdrawn form the normal distribution
- The shape of distribution depend on sample size and, the sum of all distributions is a normal distribution
- t- distribution is based on sample size and vary according to the degrees of freedom

What is the t -test

- *t* test is a useful technique for comparing mean values of two sets of numbers.
- The comparison will provide you with a statistic for evaluating whether the difference between two means is statistically significant.
- *t* test is named after its inventor, William Gosset, who published under the pseudonym of student.
- *t* test can be used either :
  1. to compare two independent groups (independent-samples *t* test)
  2. to compare observations from two measurement occasions for the same group (paired-samples *t* test).

What is the t -test

- The null hypothesis states that any difference between the two means is a result to difference in distribution.
- Remember, both samples drawn randomly form the same population.
- Comparing the chance of having difference is one group due to difference in distribution.
- ***Assuming that both distributions came from the same population, both distribution has to be equal***.

What is the t -test

- Then, what we intend:

“To find the difference due to chance”

- Logically, The larger the difference in means, the more likely to find a significant *t* test.
- But, recall:

1. Variability

More variability = less overlap = larger difference

2. Sample size

Larger sample size = less variability (pop) = larger difference

Types

1. The ***independent-sample t test*** is used to compare two groups' scores on the same variable. For example, it could be used to compare the salaries of dentists and physicians to evaluate whether there is a difference in their salaries.

2. The ***paired-sample t test*** is used to compare the means of two variables within a single group. For example, it could be used to see if there is a statistically significant difference between starting salaries and current salaries among the general physicians in an organization.

Assumption

1. Dependent variable should be continuous (I/R)
2. The groups should be randomly drawn from normally distributed and independent populations

e.g. Male X Female

Dentist X Physician

Manager X Staff

NO OVER LAP

Assumption

3. the independent variable is categorical with two levels
4. Distribution for the **two independent** variables is normal
5. Equal variance (homogeneity of variance)
6. large variation = less likely to have sig t test = accepting null hypothesis (fail to reject) = Type II error = a threat to power

Sending an innocent to jail for no significant reason

Independent Samples *t*-test

- Used when we have two independent samples, e.g., treatment and control groups.
- Formula is:

$$t_{\bar{X}_1 - \bar{X}_2} = \frac{\bar{X}_1 - \bar{X}_2}{SE_{diff}}$$

- Terms in the numerator are the sample means.
- Term in the denominator is the standard error of the difference between means.

Independent samples *t*-test

The formula for the standard error of the difference in means:

$$SE_{diff} = \sqrt{\frac{SD_1^2}{N_1} + \frac{SD_2^2}{N_2}}$$

Suppose we study the effect of caffeine on a motor test where the task is to keep a the mouse centered on a moving dot. Everyone gets a drink; half get caffeine, half get placebo; nobody knows who got what.

Independent Sample Data (Data are time off task)

Experimental (Caff)	Control (No Caffeine)
12	21
14	18
10	14
8	20
16	11
5	19
3	8
9	12
11	13
	15
$N_1=9$, $M_1=9.778$, $SD_1=4.1164$	$N_2=10$, $M_2=15.1$, $SD_2=4.2805$

Independent Sample Steps(1)

1. Set alpha. Alpha = .05
2. State Hypotheses.

   Null is H_0: $\mu_1 = \mu_2$.

   Alternative is H_1: $\mu_1 \neq \mu_2$.

Independent Sample Steps(2)

3. Calculate test statistic:

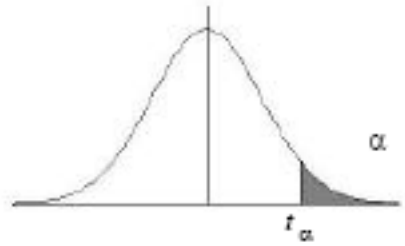
$$t = \frac{\overline{X}_1 - \overline{X}_2}{SE_{diff}} = \frac{9.778 - 15.1}{1.93} = \frac{-5.322}{1.93} = -2.758$$

$$SE_{diff} = \sqrt{\frac{SD_1^2}{N_1} + \frac{SD_2^2}{N_2}} = \sqrt{\frac{(4.1164)^2}{9} + \frac{(4.2805)^2}{10}} = 1.93$$

Independent Sample Steps (3)

4. Determine the critical value. Alpha is .05, 2 tails, and df = N1+N2-2 or 10+9-2 = 17. The value is 2.11.
5. State decision rule. If |-2.758| > 2.11, then reject the null.
6. Conclusion: Reject the null. the population means are different. Caffeine has an effect on the motor pursuit task.

Table 4: Percentage Points of the *t* distribution



	α					
df	0.250	0.100	0.050	0.025	0.010	0.005
1	1.000	3.078	6.314	12.706	31.821	63.657
2	0.816	1.886	2.920	4.303	6.965	9.925
3	0.765	1.638	2.353	3.182	4.541	5.841
4	0.741	1.533	2.132	2.776	3.747	4.604
5	0.727	1.476	2.015	2.571	3.365	4.032
6	0.718	1.440	1.943	2.447	3.143	3.707
7	0.711	1.415	1.895	2.365	2.998	3.499
8	0.706	1.397	1.860	2.306	2.896	3.355
9	0.703	1.383	1.833	2.262	2.821	3.250
10	0.700	1.372	1.812	2.228	2.764	3.169
11	0.697	1.363	1.796	2.201	2.718	3.106
			⋮			
29	0.683	1.311	1.699	2.045	2.462	2.756
30	0.683	1.310	1.697	2.042	2.457	2.750
40	0.681	1.303	1.684	2.021	2.423	2.704
60	0.679	1.296	1.671	2.000	2.390	2.660
120	0.677	1.289	1.658	1.980	2.358	2.617
∞	0.674	1.282	1.645	1.960	2.326	2.576

Dependent Samples t-tests

Dependent Samples *t*-test

- Used when we have dependent samples – matched, paired or tied somehow
  - Repeated measures
  - Brother & sister, husband & wife
  - Left hand, right hand, etc.
- Useful to control individual differences. Can result in more powerful test than independent samples *t*-test.

Dependent Samples *t*

Formulas:

$$t_{\bar{X}_D} = \frac{\overline{D}}{SE_{diff}}$$

t is the difference in means over a standard error.

$$SE_{diff} = \frac{SD_D}{\sqrt{n_{pairs}}}$$

The standard error is found by finding the difference between each pair of observations. The standard deviation of these difference is SD_D. Divide SD_D by sqrt (number of pairs) to get SE_{diff}.

Another way to write the formula

$$t_{\bar{X}_D} = \frac{\bar{D}}{SD_D / \sqrt{n_{pairs}}}$$

Dependent Samples *t* example

Person	Painfree (time in sec)	Placebo	Difference
1	60	55	5
2	35	20	15
3	70	60	10
4	50	45	5
5	60	60	0
M	55	48	7
SD	13.23	16.81	5.70

Dependent Samples *t* Example (2)

1. Set alpha = .05
2. Null hypothesis: H_0: $\mu_1 = \mu_2$.
   Alternative is H_1: $\mu_1 \neq \mu_2$.
3. Calculate the test statistic:

$$SE_{diff} = \frac{SD}{\sqrt{n_{pairs}}} = \frac{5.70}{\sqrt{5}} = 2.55$$

$$t = \frac{\overline{D}}{SE_{diff}} = \frac{55-48}{2.55} = \frac{7}{2.55} = 2.75$$

Dependent Samples t Example (3)

4. Determine the critical value of t.
   Alpha =.05, tails=2
   df = N(pairs)-1 =5-1=4.
   Critical value is 2.776
5. Decision rule: is absolute value of sample value larger than critical value?
6. Conclusion. Not (quite) significant. Painfree does not have an effect.

Dependent or Paired t-Test: Output

Paired Samples Statistics

		Mean	N	Std. Deviation	Std. Error Mean
Pair 1	Pre	4.7000	10	2.11082	.66750
	Post	6.2000	10	2.85968	.90431

Paired Samples Correlations

		N	Correlation	Sig.
Pair 1	Pre & Post	10	.968	.000

Paired Samples Test

		Paired Differences					t	df	Sig. (2-tailed)
		Mean	Std. Deviation	Std. Error Mean	95% Confidence Interval of the Difference Lower	Upper			
Pair 1	Pre - Post	-1.50000	.97183	.30732	-2.19520	-.80480	-4.881	9	.001

Is there a difference between pre & post?

$t(9) = -4.881, p = .001$

Yes, 4.7 is significantly different from 6.2