Cheatsheet

Libero Biagi

October 11, 2025

Intro

This is a general cheatsheet where I will try to put all the coding, the formulas, the terminology and the possible exercises that might be useful for the exams.

Contents

1 Data Mining

1.1 Theory

1.1.1 Lecture 1

Fernando Becao presents himself and the tutor.
Fernando Becao asks us why we weren't at the party.
We study about supervised and unsupervised learning.
Exam + project.
Exam is 55% of the final grade
Project is 45% and is divided into

- Deliverable 1 -> 30% by November 4th
- Deliverable 2 -> 60% by January 3rd
- Discussion -> 10%

Both exam and project will expire after the year

1.1.2 Lecture 2

Random debate about things
We need to find relevant data to reduce computation and improve the quality of works

Big Data -> we simplify datasets that are too big for normal processing. From big data into mean datasets
Big data characteristics:

- **Volume:** big data are big
- **Velocity:** Big data technology allows databases to process and analyze data while it is being generated
- **Variety:** Big data can be a mixture of structured, unstructured and semi-structured data. To solve this problem Big Data is flexible

Some information about AI, ML and Data Science.

AI: Making things that show human intelligence. Automatize human tasks.

Machine Learning: Approach AI using systems that can find patterns from data and examples. ML systems learn by themselves. We can see them as a sort of subset of AI or a way towards AI

Data Science: the study of where information comes from and how to turn it into a valuable resource.

Data Science vs Data Mining

Data science is a set of principle that guide the extraction of information from data. We try to view problems from a data perspective.

Data mining on the other hand is the extraction of knowledge from data via the use of algorithms.

Find/build attributes is very important. Basically the properties of the things that we are studying have to be selected or preprocessed.

After the construction of the features a ML model can learn how to divide the instances on an hyperplane. This can be used to make predictions. Recall that a model will predict only based on the class that it was trained on.

Is very important to use the relevant and appropriate features. More features can mean more possibility of discriminating between the classes.

Typically we use labeled examples, enough data and clear cut definitions.

If our models try to attribute a label we have supervised learning. Future examples will get a prediction.

To build features we use data warehouse and ETL (extract, transform and load). Recall that we can transform every relational schema into a table (at least with SQL).

Minimum information to identify a costumer:

- Transaction number
- Date and time of transaction
- Item purchased
- Price
- Quantity purchased
- A table that matches product code to its name, subgroup code to name, product group code to group name.
- Product taxonomy to link product code to subgroup code and product subgroup code to product group code.
- Card ID

Consistent behaviors are easier to analyze. To found the level of consistency i can use the standard deviation and I can remove the outliers.

We can also look at relevant variables like:

- Recency
- Frequency
- Monetary value
- Average purchase
- Most frequent store
- Average time between transactions
- Standard deviation of transactional interval
- Costumer stability index
- Relative spend on each product

Canonical tasks in data mining
If we want to classify new data from a decision criterion previously learned we talk about **Supervised learning**

If we want to summarize a data set we talk about **Unsupervised learning**

Supervised learning:

- Classification
- Regression

Unsupervised learning:

- Clustering
- Visualization
- Association

In our datasets the features are the columns while the rows are the instances

Clustering -> We plot the instances on a hyperplane and we try to group them by distance, we can have different plots

Association rules -> based on our data we can use the transactions to find some rule to infer costumer routine. We use confidence, support, lift and so on.

Visualization -> n-D data can be difficult to visualize so we can flatten them into 2-D ones. Examples are histograms, bubbles and goggle boxes. We can also do some dimensionality reduction using PCA or other algorithm.

Data mining process

We can have different methodologies to acquire insights from data.

KDD

1. Data
2. Selection
3. Preprocessing
4. Transformation
5. Data mining
6. Interpretation/Evaluation
7. Knowledge

CRISP-DM

1. Business understanding
2. Data understanding
3. Data preparation
4. Modeling
5. Evaluation
6. Deployment

1.1.3 Lecture 3

We use ML algorithms instead of fixed formulas because the events are too complex for a simple formula.

We don't understand the problems but we try to approximate solution. Black Box ML approach basically.

If the model is not good enough we can either improve the model or gather more data. The second one is quicker. Dumb algorithms with a lot of data will learn better than smart ones with not a lot of them.

Traditional statistics might be described as being characterized by data sets which are small and clean, which are static, which were sampled in an iid manner, which were often collected to answer the particular problem being addressed, and which are solely numeric.

Size of data set is also useful. If we have big datasets even tiny effects exists. They can be useless. Now we should ask if the effect is important or not.

From statistical significance to substantive significance.

Since the datasets are very big we try to compute them in a adaptive or sequential way. Also we might have multiple and interrelated files

We have two main ways to process data

- Incremental

Dimension	Primary data	Secondary data
Definition	Data you collect yourself for a specific, current purpose.	Data collected by others for a different (often past) purpose.
Typical sources	Surveys, experiments, interviews, field measurements, sensors.	Government statistics, research papers, data portals, company databases, syndicated datasets, web-scraped corpora.
Control over design	Full control (sampling, instruments, definitions, timing).	Little/no control; must accept others' design choices.
Fit to your question	High: tailored to your problem and target population.	Varies: often indirect or requires redefinition/derivations.
Cost	Usually higher (money, time, staff, tooling).	Usually lower or free; licensing may apply.
Time to obtain	Longer (planning → collection → cleaning).	Faster (download/access + cleaning/understanding).
Timeliness/recency	Up-to-date by design.	May be outdated; release lags common.
Granularity	Exactly what you need (variables, frequency, detail).	Fixed by source; may lack key variables or be too aggregated.

Table 1: Comparison between primary and secondary data.

- Batch

Other characteristics of problems in data mining are

- Nonstationarity and population drift
- Selection bias
- Spurious relationships

Input variables should be causally related to the output. If we have a small number of observations we will have high correlation.

Confounding variables will correlate both with dependent and independent variables. The confounding factor will estimate incorrectly the relation. A **Spurious relationship** happens when we perceive a relationship between two variables that actually doesn't exist. We ar not accounting for the confounding factor.

Input variables must be causally related to the output to be meaningful.

We have to discriminate between **causality** and **correlation**:

- Correlation -> two things co-occurs, changing one of them will not change the output
- Causality -> a change on the input will cause a change on the output

Correlation is a pattern, causation a consequence

Input Space -> is the input feature vector, the algorithm will look for a solution

Curse of dimensionality -> bad effects can be caused by redundant features, bigger dimensionality means bigger and more sparse input. Clustering can be harder. Generalization is exponentially harder. Feature selection can be an answer.

Input space coverage -> representative training examples will improve our model quality. Test data outside the training input space will be bad for the performance.

Interpolation -> predictions in the range of data that we have
Extrapolation predictions outside the range of data that we have, Time series
Separation -> if we plot our data on a 2-D space we can be able to draw by hand the regions where the data are. ML models will try to do it mathematically. If we can use a linear hyperplane we have **Linearly separable data** if not they are not linearly separable. With separable classes we can get 0 errors. We want to minimize the error (finding the Bayes error). Simple algorithm like perceptron will solve problems with separables classes.

Kind of variables

- Nominal
- Ordinal
- Discrete

- Continuous
- Interval
- Ratio

Metadata -> Information that provides information about data

- Descriptive metadata
- Structural metadata
- Administrative metadata

1.1.4 Lecture 4

Today visualization Strong points of good visualization

- Can show processes
- Show comparisons between numbers
- Can show differences and changes
- Can use geography to show local data
- Can show relationships
- Can show density

What not to do

- Don't clog the graph
- No 3-D
- Don't use unfaithful charts

Guidelines

- Reduce chartjunk
- Increase data-ink ration
- Use similar structures in the same charts

Tufte lie factor -> Measure of distortion in a graph

$$\text{Lie Factor} = \frac{\text{Size of effect in graph}}{\text{Size of effect in data}} \tag{1}$$

As rule of thumb we should have $0.95 < \text{Lie Factor} < 1.05$

Suggestions

- White background
- Don't use colors if not necessary, Charts should be like man suits (Prof, 2025)
- Order the items in a smart way
- Use a scale
- Crisp borders
- No smooth line
- Simple is better
- No 3-D
- Discriminate clearly the clusters with shapes and colors
- Spaghetti chart should highlight interesting patterns
- Clutterplot should highlight interesting points
- Use shadows to highlights interesting points

Charts that can be useful

- Bar charts, vertical and horizontal
- Line charts
- Mix of the previous
- Stacked bar charts
- Scatterplot, also with tendency lines, grids, bubbles
- Pie charts, can mix them with histograms, we can label them, compare with others (same as paired column chart). They are like stacked bar charts. We also have part to whole mini pie charts. We can plot them on a graph.
- Radar plot, easy to compare more of them

Visualization for analysis

- Histogram, can be stacked and combined with scatterplots
- Boxplot, can be combined with scatterplots, histograms

Correlation matrices

- Can be done with distributions, scatter plots and matrices

Parallel coordinate

- Show patterns that can be compared easily

Small Multiples

- Can show many graphs together, easy for comparisons
- Basically we can have a lot of variables in a 2-D graph

Heat maps

- Can show quantities in the plane that we are visualizing

Tree maps

- We can see quantities with sizes and dividing them points based on certain characteristics

Geo-visualization

- We can aggregate data from different geographical point of view, we can use normal maps, bubbles and plot quantities of them. Cartograms are useful for quantities on and densities.

Linked Views

- We can put different vies of the dataset together. We can select different subsets to compare them

1.1.5 Lecture 5

Today data preparation and pre-processing

With data pre-processing and preparation we want to make our data useful for the model. All the datasets are made of signal and noise. We want to maximize the signal and reduce as much as possible the noise.
Real data usually are:

- Incomplete
- Noisy
- Inconsistent

Preparation:

1. Missing values
2. Outliers
3. Discretization and encoding
4. Imbalanced datasets

Pre processing

1. Feature selection →Relevance analysis and redundancy removale
2. Feature engineering
3. Feature scaling and normalization

Treating missing data

Missing values are values that are not available, very common.
How to deal:

- Delete records with missing data, biased
- Delete columns with too many missing data, loose information
- Manually insert them
- Imputing with central tendency metrics, for general and for subsets
- Derive them in an objective way
- Use a predictive model
- Use similarity measures

Usually our go to is to use the quickest and simplest option, after we analyze the performance of the model. If the error is significantly higher on the subset with imputed values than on the one with non imputed we look for other options.

2 Programming

2.1 Theory and code examples

2.1.1 Lecture 1

Printing →A string is returned and shown on the screen

```
    print("Hello World")
```

print() syntax →print(object(s), separator=separator, end=end, file=file, flush=flush)

Errors →They happen when there is a problem in the code

- **Runtime Errors** →Something wrong, the code won't run
- **Semantic Errors** →The output is not what we expected

We have many more of them, to solve we have to **debug the code**.

Using Jupyter notebooks we can use some special commands that are preceded by the % character

- **%timeit** →determines the execution time of a single-line statement. Performs multiple runs to achieve robust results.
- **%%timeit** →same as %timeit but for the entire cell
- **%time**→determines the execution time of a single-line statement. Performs a single run!
- **%%time**→same as %time but for the entire cell
- **%run**→Run the named file inside IPython as a program
- **%history**→displays the command history. Use %history -n to display last n-commands with line numbers.
- **%recall<line_no>**→re-executes command at line_no. You can also specify range of line numbers
- **%who**→Shows list of all variables defined within the current notebook
- **%lsmagic**→Shows a list of magic commands
- **%magic**→Quick and simple list of all available magic functions with detailed descriptions
- **%quickref**→List of common magic commands and their descriptions

And many more

2.1.2 Lecture 2

Semantics →in Python tabs and spaces are used to structure the code. If you use a colon you have to indent the code in the right way. Semicolons instead can be used for multiple statements on the same line

Objects →everything in Python is an object with it's own methods, functions and characteristics

Comments →# denotes comments

To represent information we can use different kind of data types and structures

Variables →a variable can take whatever value we want

```
    a = 2 #variable declaration
    print(a)
>>> 2
    type(a)
>>>int
```

Other possible types are

- float
- string
- complex
- boolean

If we want to save a collection of objects we can use

```
List = [1, 3, "a", 7]
Tuple = (1, "a", 8, 7)
Dictionary = {"a":1, "b":4}
Set = {1, "a", 4, 8}
```

Data Structure	Mutable / Immutable	Ordered / Unordered	Indexed / History
List	Mutable	Ordered	Indexed
Dictionary	Mutable	Unordered	History of addition
Tuple	Immutable	Ordered	Indexed
Set	Mutable	Unordered	No indexing (unique elements)

Table 2: Comparison of Python collection data structures.

Every data structure has it's own methods.
When we are using indexed data structures we have to remember that the first index is 0.

Since we want to work with data and modify them we need a way to do it. Operators can take variables and make computations if those are compatible

Operator	Name	Example
`+`	Addition	`x + y`
`-`	Subtraction	`x - y`
`*`	Multiplication	`x * y`
`/`	Division	`x / y`
`%`	Modulus	`x % y`
`**`	Exponentiation	`x ** y`
`//`	Floor division	`x // y`

Table 3: Python arithmetic operators with names and examples.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Table 4: Python assignment operators with examples.

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Table 5: Python comparison operators.

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverses the result, returns False if the result is True	not(x < 5 and x < 10)

Table 6: Python logical operators.

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Table 7: Python bitwise operators.

Operator	Description
`is`	Returns True if both variables are the same object
`is not`	Returns True if both variables are not the same object

Table 8: Python identity operators.

Flow control →to check that our program is doing what we want we can apply different statements.
if →we define a condition under which a certain action is done if that condition is reached

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Table 9: Python comparison operators with examples for conditional statements.

elif →it means else if, we specify a new condition
else →to be put as last, if neither if or elif are met we use else

```
if name == "Libero":
    print("Hello")
elif name == "Jose":
    print("Forza Milan")
else:
    print("None")
```

Loops →for now we just cared about one time operations, we can do those on multiple times with loops. We have two main loops

- **for loops** →we specify an interval and the action is performed that number of times.
- **while loops** →the operation is made as long as a certain condition is true

```
    for x in range (10):
        print(x)

    a = 0
    while a < 25:
        print(a)
        a +=1
```

Some problems can arise if the condition of the while loop is never met.

Comprehension →for faster performances we can use this particular structure. It works for all the mutable data structure

```
    squares = [x**2 for x in range(10)]
    print(squares)

    even_squares = [x**2 for x in range(10) if x % 2 == 0]
    print(even_squares)


    squares_dict = {x: x**2 for x in range(5)}
    print(squares_dict)

    squares_set = {x**2 for x in range(5)}
    print(squares_set)
```

With this general structure

```
    {key_expression: value_expression for item in iterable if condition}

    {expression for item in iterable if condition}
```

Slices →if we want a subset of the data structure.

General structure

```
    list[start:stop:step]
```

Negative indexes mean to start from the end

typecast →we can transform a variable with a certain structure into another with another structure

```
    # Typecasting examples in one snippet

# String to int
x_str = "10"
x_int = int(x_str)
print(x_int, type(x_int))  # 10 <class 'int'>

# String to float
y_str = "3.14"
y_float = float(y_str)
print(y_float, type(y_float))  # 3.14 <class 'float'>

# Int to string
z_int = 100
z_str = str(z_int)
print(z_str, type(z_str))  # '100' <class 'str'>

# Tuple to list
tup = (1, 2, 3)
lst = list(tup)
print(lst, type(lst))  # [1, 2, 3] <class 'list'>

# List to tuple
lst2 = [4, 5, 6]
tup2 = tuple(lst2)
print(tup2, type(tup2))  # (4, 5, 6) <class 'tuple'>

# Int to float
num = 7
num_float = float(num)
print(num_float, type(num_float))  # 7.0 <class 'float'>

# Float to int
num2 = 9.8
num2_int = int(num2)  # truncates decimal part
print(num2_int, type(num2_int))  # 9 <class 'int'>
```

2.1.3 Lecture 3

Functions →like a mathematical function a python one will take some inputs and will apply the same operations to get a certain result. We define them to avoid writing the same code over and over.

```
    def function(input): #to define
        input operation
        return output

    function(other_input) #to call
```

A function can also work with the elements of a list or other collections of elements using a for loop.

If we want to have a flexible number of inputs we can use *args and **kwargs

- ***args** →the function will take a list as input and will do the computations on each element of that list
- ****kwargs** →same as args but with a list of keys and values, the function will return a dictionary

```
    def multiply(*args):
        output = 1
        for n in args:
            output *= n
        return output

    print(multiply(2, 3, 4))
```

```
    def my_function(**kwargs):
        for key, value in kwargs.items():
            print(key, value)

    my_function(name="Alice", age=25, city="Paris")

    ###Output
name Alice
age 25
city Paris
```

Link for a video if it's still confusing

You can add infos about your function with the ___doc___

```
    def greets(*args):
        """This function says Hello"""
        for name in args:
            print("Hello", name)
    greet.__doc__
```

If we want a fast singular use function we can use **lambda functions**.

```
    lambda arguments: expression

    add = lambda x, y : x + y
    add(2, 3)
```

Map →we want to transform certain values into others in a fast way

```
    list(map(lambda x : x * 2, [1, 2, 3, 4]))
```

Filter →if we want to select a subset of our data

```
    from random import sample

    X = sample(range(-25, 25), 25)
    f = filter(lambda x: x > 0, X)
```

When we want to call a certain function multiple times we can use two main ways:

- **Iteration** →We define a number of times and the function will do its job for that number.
- **Recursion** →The function will call itself until it reaches a base case. It's important to define base, edge and general cases. Useful when we work with trees or graphs

```
    def factorial(n):
    if n == 1:
        return 1    #base case
    elif n == 0:
        return 1    #edge case
    else:
        return n * factorial (n-1) #general case
```

We should differentiate between:

- **Functions** →defined by def or lambda, they can be applied to almost everything if well defined.
- **Methods** →associated with certain objects
- **Attributes** →variables associated to certain objects

Remember kids, if you want to code something probably someone has already done it. So why bother? We can import the work done by other people

```
    import module as mod
    from module import method1, method2
```

sys.path will give us all the directories that our interpreter is watching
Useful modules can be:

- csv
- datetime
- io
- json
- math
- os
- random
- sqlite3
- xml
- zipfile
- zlib

Namespace A namespace is a collection of names that reference objects.
- **Built-in Names**: predefined names available in every Python interpreter. Examples: `list`, `dict`, `map`, `tuple`.

```
import builtins
print(dir(builtins))   # list built-in names
```

- **Global Names**: user-defined names created in the main program body (variables, functions, classes).

```
x = 42   # global variable
def foo():
    return x
print(globals())   # list global names
```

- **Local Names**: names defined inside a function, valid only inside it.

```
def bar():
    y = 10  # local variable
    print(locals())   # list local names
bar()
# print(y)  -> Error: y does not exist in the global scope
```

Scope Scope defines where a variable can be accessed.
- **Global Scope**: variable accessible throughout the whole program. - **Local Scope**: variable accessible only inside the function where it was declared.

```
animal = "dog"  # global variable

def test_scope():
    # local variable with the same name
    animal = "cat"
    print("Local:", animal)

test_scope()
print("Global:", animal)
```

Output:

```
Local: cat
Global: dog
```

Using the keyword global It allows modifying a global variable inside a function.

```
count = 0

def increment():
    global count
    count += 1

increment()
print(count)  # 1
```

2.1.4 Lecture 4

Today we start with Pandas, fav library for data.

```
    import pandas a pd #always import as pd
```

Series →a Pandas series is 1-D object that can hold any data type. Is made of 2 arrays, one for the index and the other one with the actual data.

```
    obj = pd.Series([11, 28,72, , 5, 8])

    obj.array #information on the array
    obj.index #information on the indices
```

The left column is the index one, always. We can use custom index lists

```
    fruits = ['apples', 'oranges', 'cherries', 'pears']
    quantities = [20, 33, 52, 10]
    S = pd.Series(quantities, index=fruits)
    #the index list is fruits

    S2 = pd.Series([17, 13, 31, 32], index=fruits)
    print(S + S2) #will sum the elements on the same position
    sum(S) #will tell us the total of the nubers in the array

    fruits2 = ['raspberries', 'oranges', 'cherries', 'pears']
    S2 = pd.Series([17, 13, 31, 32], index=fruits2)
    print(S + S2) #the operations are aligned by index, elements that don't appear in
        both lists will be NaN, like a Joint in relational algebra

```

```
print(S["apples]) #we can access like a dictionary, output will be 20
S["apples"] = 25 #to modify the element

print(S + 2) #to add 2 to every element
print(np.sin(S)) #to apply the sin function

S.apply(lambda x: x if x > 25 else -1 * x) #apply will make a function to all the
    elements of the series, Map() will work element wise

print(S[S > 25]) #to filter using booleans

"apples" in S #to see if a key exists

cities = {"London": 8615246,
"Sesto San Giovanni": 79121,
"Montevideo": 1405798}
city_series = pd.Series(cities) #from dictionary to series

print(cities.isnull()) #to see which element is NaN
print(cities.notnull()) #to see which element is not Nan
print(cities.fillna(0)) #to change Nan to 0.0 (floats)
print(cities.fillna(0).astype(int)) #to change Nan to 0 (integers)

obj.name = "Libero" #to give our series a name
obj.index.name = "diocane" #to give the index comlumn a name
obj.index #we see a list of the index column
```

Characteristics of Index object

- Immutable
- Behaves like a fixed size set (we can check if a certain index is in the column with the in method)
- Can contain duplicate labels

We have several useful methods for the index objects.

If we put multiple series one after the other we get a matrix called **DataFrame** which can be considered like an Excel spreadsheet. Like Excel a pandas dataframe has both row and column index.

```
pd.concatenate([row1, row2, row3]) #to connect the rows if they share the same
    index
pd.concatenate([row1, row2, row3], axis=1) #they become columns, default is 0

df.colum.values #retrieve the columns names
df = pd.Dataframe(df, index= a_list) #to rename the index column with the names
    of a list
df = pd.Dataframe(df,
columns=("name2", "name3", "name1") #to rearrange the columns order, same as df.
    reindex(["name3", "name1", "name2"])

df.rename(columns={
"name1" = "newname1",
"name2" = "newname"
}, inplace=True) #to rename columns, inplace False will return a copy of the
    dataset

df = pd.Dataframe(df, columns=["name1", "name2"], index=df["other_column"]) #to
    put other_column as new index

df.loc("label", "other_label") #select all the rows with that label
df.loc(df.condition > number) #for conditions

df.sum() #sum on all the columns
df["column1"].sum() #sum on a single column
df["column1"].cumsum() #cumulative sum

```

```
    #to add columns we put the new column name in the column list and we impute it
        with df["new_col"] = df["old_col].cumsum() for example. Default values for
        new columns are NaN

    df["attribute"] #access a column same as df.attribute()

    df["attribute"] = number #will fill the column with that number, for unique
        values use a list of the same size

    df.T #to get the transpose

    df = pd.read_csv("path/to/csv/file")

    df.head(n) #to see the first n lines of the dataframe

    df.to_csv("path/where/you/want/to/save/the/file")

    df.loc[] #access with label based approach, KeyError if it can't find the value
    df.iloc[] #access with index, IndexError if the requested index is out of bounds,
         except for slice indexers

    df["b": "m"] #slices can also work by labels, we will take also the middle values
```

How to select subsets

- Fetching like a dictionary, select the index that you want
- Slice by index
- Slice by labels
- Fetch multiple entries
- Condition based selection

You should use .loc[] and by index

iloc vs loc

- iloc = by label
- loc = by index
- duloc = lord Farquad city

```
    df.loc["label"] #select a row by index label
    df.loc["label", ["col1", "col3"]] #row and column selection
    df.loc[:"label", ["col1", "col3"]] #as before but with slicing

    df.iloc[[2, 1]] #fetch rows with that position
    df.iloc[[1, 2], [3, 0, 1]] select certain rows and columns
    df.iloc[:, :3] #with slices
```

2.1.5 Lecture 5

To see statistical characteristics of our dataset we can do

```
    df.describe() #the arguments are percentiles, include and exclude
```

To see how many null values and the object in the column

```
    df.info() #the arguments are verbose, buf, max_cols, memory_usage, null_counts
```

To see the number of unique values

```
    df["name_col"].value_counts() #the arguments are normalize, sort, bins, dropna
```

To group or divide kind of data, basically binning

```
    df["col1"] = df["col1"].map(lambda x: operation on x)
    df.groupby("col1") ["col2"].value_counts() #with parameters by, group_keys, dropn
        , as_index
```

If we want to make a multi dimensional visualization of group by we can use pivot

```


pd.pivot(index="col1", columns="col2", values="col3") #with arguments data, values,
    index, columns, aggfunc
```

To aggregate df using a relational algebra join

```
    df3 = pd.merge(df1, df2) #with arguments right, how, on, left_on, right_on
```

To make queries (SQL style)

```
    df.head() #to see first 5 rows
    df.query("price < 150").head() #with argument inplace.

    #we can apply and, or and so on, basically SQL queries inside a string
```

For 1-Hot encoding

```
    pd.get_dummies(df["key"], prefix="key", dtype=int) #to join with another df that
        has those columns we use
    df = df[["data"]].join(dummies)
```

Now example of a full work with Pandas, not gonna write it.

3 Machine Learning

3.1 Theory

3.1.1 Lecture 1

Machine Learning differs from traditional programming, we want to model a function to predict data or patterns. Machine Learning has many applications in multiple fields. Since we are using data is important to notice that those can have biases like people. Some biases related to people can be confirmation, falalcy of centrality or survivorship.

ML as said has many applications, some can be:

1. Image and object recognition
2. Videogames
3. Sound generation and recognition
4. Art and style imitation
5. Predictions

Many algorithms exist for those tasks but all of them are made from three main components:

1. **Representation** →Space of models
2. **Evaluation** →How to assess which model is better for a certain task
3. **Optimization** →How to improve the model

Optimizing a model means finding the best parameters and hyperparameters, we can do it via:

- Combinatorial optimization
- Convex optimization
- Constrained optimization

Different tasks require different kind of learning:

- **Supervised/Inductive** →We have the labels of the training data
- **Unsupervised** →We don't have the labels in the training data
- **Semi-supervised** →Mix of the previous two
- **Reinforcement** →We want to maximize a reward function after performing some actions

More specifically we will study the first one

Supervised →We have many examples of data, we want to find the function that describes best their distribution. We can do classification, regression or estimating probabilities. The core idea is that we want to fit a curve that discriminates between classes or that fit at best a scatterplot. Since we are estimating we will have for sure a certain degree of errors, that is called the **bias**. Bias can also be positive if is used in a smart way, if we have knowledge about something we can use it to avoid useless steps.

As we know in the 1-D case a math function is on the form of

$$y = f(x) \tag{2}$$

And if we open $f(x)$

$$f(x) = ax + b \tag{3}$$

This is the most basic linear case. We want to estimate the parameters a and b to fit the data in the best way. The core idea is to reduce as much uncertainty as possible.

3.1.2 Lecture 2

Machine learning implies that the machine will learn something, there are different approaches for that. They can be applied to different tasks for better performances.

1. **Symbolism** →Fill gaps in existing knowledge
2. **Connectionism** →Emulate the brain
3. **Evolutionary Computation** →Simulate evolution
4. **Statistical Learning** →Reduce uncertainty
5. **Analogy Modelling** →Check for similarities between old and new

Tribe	**Origins**	**Master Algorithm**
Symbolists	Logic, Philosophy	Inverse Deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary Biology	Genetic Programming
Bayesians	Statistics, Probability	Inference
Analogizers	Psychology	Support Vector Machines (SVM)

Symbolists:

- Logic programming
- Expert systems
- Decision trees
- Functional programming

Connectionists:

- Perceptron
- MLP
- DNN
- Hopfield networks
- Boltzmann networks

Evolutionary:

- Genetic algorithms
- Genetic programming
- Ant Colony optimization
- Particle Swarm optimization

Bayesians:

- Bayesian classifiers
- Bayesian Belief networks
- Markov models

Analogizers:

- KNN
- Self-organization
- SVM

If we cross together the tribes and the components seen in the previous class we can see that:

- **Representation**
  - Probabilistic Logic
  - Each rule has a weight
- **Evaluation**
  - Posterior probability
  - User defined objective function
- **Optimization**
  - Genetic programming
  - Backpropagation

3.1.3 Lecture 3

The most important part of a machine learning model is the dataset, sometimes the data aren't good enough on their own so we must account for that. Some ways are

- **Feature extraction and engineering** →extract something from simpler features
- **Feature transformation** →modify the features to make them more useful for the model
- **Feature selection** →use the most relevant features

In this course we will see mainly the third point. When we do **Feature selection** we aim to reduce the number of inputs variables to a subset that can be equally useful. Simpler models are easier to interpret, the training time is reduced, generalization is enhanced and we remove redundant variables.

Feature selection is done via 3 main methods:

- **Filter methods** →Statistical methods to evaluate the importance of the features, like correlation
- **Wrapper methods** →We train a model with subsets of features until we find the best one
- **Embedded methods** →During the training of our model the importance of the features is assessed.

Filter methods:

- Pearson correlation coefficient
- Spearman rank coefficient
- ANOVA correlation coefficient
- Kendall rank coefficient
- Chi-Squared test
- Mutual information

Pearson Correlation coefficient, it measures linear correlation:

$$\rho_{X,Y} = \frac{\text{cov(X,Y)}}{\sigma_X \sigma_Y}$$

For m samples

$$r = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2}\ \sqrt{\sum_{i=1}^{m}(y_i - \bar{y})^2}}$$

Spearman rank correlation coefficient, it asses how well the relationship between two variables can be described by a monotonic function

$$r_s = \rho_{rg_x, rg_y} = \frac{cov(rg_x, rg_y}{\sigma_{rg_x}\sigma_{rg_y}}$$

We shouldn't always discard variables with a small score. It depends by case to case

We can also use **Mutual information**, this can help finding non-linear dependencies but is harder to estimate than pure correlation.

Mutual information (continuous and discrete), for non linear and non monotonic relationships

$$I(X_i; y) = \int \int p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)\, p(y)}\, dx\, dy$$

$$I(X_i; Y) = \sum_{x_i} \sum_{y} P(X = x_i, Y = y) \log \frac{P(X = x_i, Y = y)}{P(X = x_i)\, P(Y = y)}$$

The problem is that mutual information can be inconvenient for feature ranking. To solve the associated problems we can use the **Maximal information coefficient** that transforms the previous and binds it in [0;1]. We obtain the percent of a variable Y that can be explained by a variable x

$$\text{MIC}(X, Y) = \max\{\frac{I(x, y)}{\log_2 \min\{n_x, n_y\}}\}$$

Wrapper methods
We consider certain subsets of our features, at each iteration we try to find the best combination of those. We can vary which and how many features to consider. The selection process involves evaluate the subsets using a predictive model and using the subset with the best performance.

The search process can use different strategies:

- Methodical
- Stochastic
- Heuristics

One of the most used is RFE

1. The model is fitted on all the predictors
2. Each predictor is ranked using it's importance to the model
3. We choose the n top ranked predictors
4. We find the best number n with the best n features

RFE can encounter problems if it overfit on uninformative features.

Our training set is used for selecting predictors, fitting the model and evaluating the performance.
To improve RFE we can add **Cross-validation**

Embedded methods
This last kind of methods consists in regularizing the model while it's been created. The process consists in adding constraints into the optimization and adding a bias toward lower complexity, so reducing the number of coefficients

We have two main regularization algorithms:

1. Ridge, or L2
2. Lasso or L1

Ridge Regression tries to minimize the sum of squared residuals and the slope of the line. This can cause many lower coefficients and can solve for parameters where we do not enough data samples.

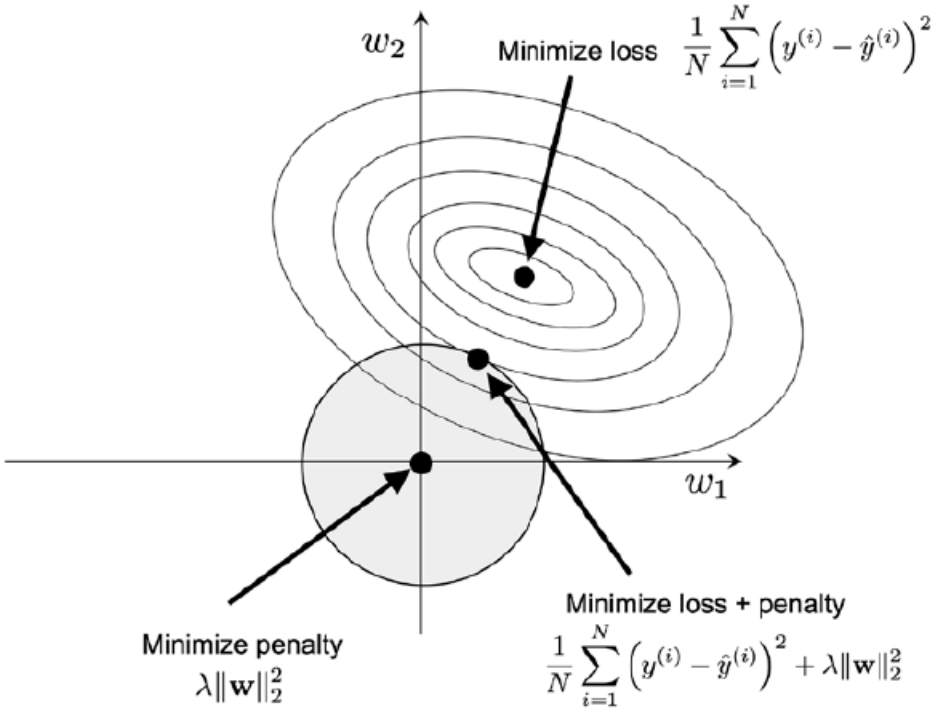


Figure 4.6: Applying L2 regularization to the loss function

Lasso Regression on the other hand tries to minimize the magnitudes. It can lead to many zero values in our coefficients set

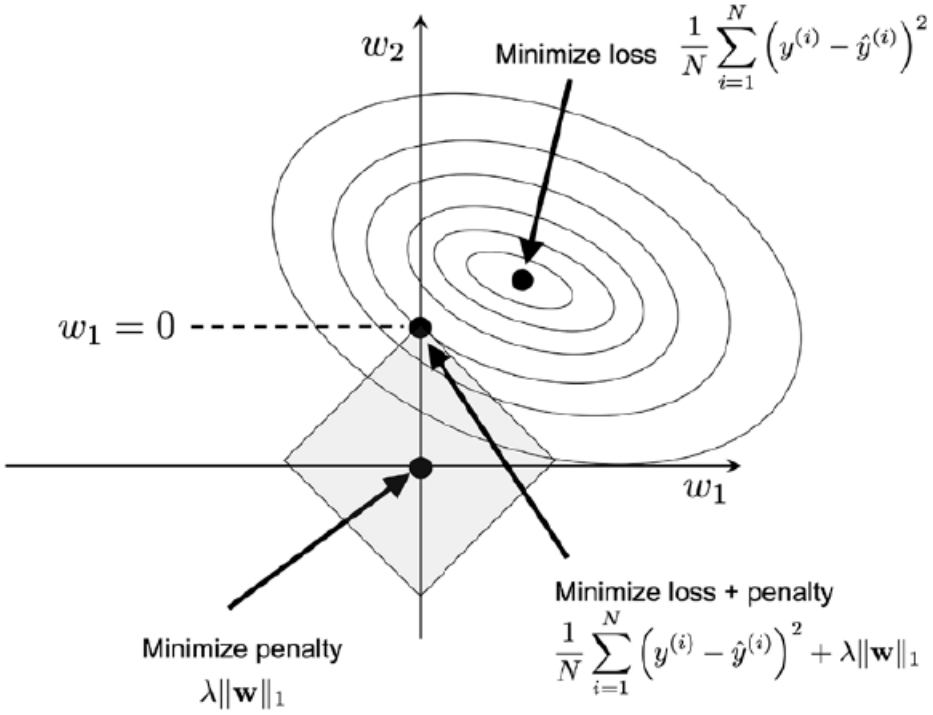


Figure 4.7: Applying L1 regularization to the loss function

3.1.4 Lecture 4

Today we look at our first algorithms, we start with **Linear Regression**.

We want to find a relationship between independent variables and the dependent one. The objective is to make predictions based on the past observations.

As we know one kind of task for ML models is regression, we want to predict a continuous values. One way is using Linear Regression.

The steps are:

1. Use least-squares to fit a line on the data
2. Calculate the R^2
3. Calculate the p-value for R^2

A regression model can be simple or multiple

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \varepsilon \tag{4}$$

Where:

- Y →Model/output
- β_0 →Intercept
- β_n →Parameters
- X_n →Inputs
- ε →Residuals

For the simple model we only use the first 2 β and the first X

We want to minimize the squared sum of the residuals, this will give us a good model. Since this is a minimization problem on both the parameters we differentiate them getting the following formulas (notes that those are estimations)

$$\hat{\beta_1} = \frac{\Sigma(X_i - \bar{X})(Y_i - \bar{Y})}{\Sigma(X_i - \bar{X})^2} \tag{5}$$

$$\hat{\beta_0} = \hat{Y} - \hat{\beta_1}\bar{X} \tag{6}$$

As said these are estimations, so we want to evaluate how good they are.

Usually we use R^2, it explains how much variation in the output can be explained differently from using just the mean.

$$R^2 = \frac{SS(mean) - SS(fit)}{SS(mean)} \tag{7}$$

Where

- SS(mean) or SST →$\Sigma(y_i - \bar{y})^2$
- SS(fit) or SSE = →$\Sigma(y_i - \hat{y_i})^2$

We can also do

$$R^2 = \frac{\text{Var(mean) - Var(fit)}}{\text{Var(mean)}} \tag{8}$$

Where

- Var(mean) = $\frac{\Sigma(y_i - \bar{y})^2}{n} = \frac{SS(mean)}{n}$
- Var(fit) = $\frac{\Sigma(y_i - \hat{y}_i)^2}{n} = \frac{SS(fit)}{n}$

Since bigger models with more features will explain more variability we can use the adjusted R^2, it will penalize models with a lot of independent variables

$$\text{Adjusted } R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1} \tag{9}$$

Where

- R^2 →Current R^2
- N →Samples
- p →Number of independent variables

After that we can check the p-value for each independent variable to check it's statistical relevance, low p-value means high probability that the variable is useful.

Another model that we have to look at is **Logistic Regression**, despite the name is a classifier and not a regressor. We want to make a binary classification (1 or 0) based on our data. Differently from the Linear regression loss function (SSE) here we use MLE (Maximum Likelihood Estimation) to maximize the probability of being right.

Usually our decision is like this

$$\begin{cases} 1 & \text{if True} \\ 0 & \text{if False} \end{cases} \tag{10}$$

In such cases we can't easily (or we straight up can't) linearly separate those examples, the we need a non linear discriminator with an output between 0 and 1. We want to model the probabilities of being of either class. The function used is a Sigmoid.

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}} \tag{11}$$

As said we use MLE to estimate the parameters of the function. Basically we maximise the probability of observing the sample.

Since we want to get a probability between 0 and 1 we have to follow some passages.

- We start from the probability $p = P(Y = 1 \mid X)$.
- We convert it into **odds**: $\frac{p}{1-p}$, which range from 0 to $+\infty$.
- We take the **log** of the odds (logit): $\log\left(\frac{p}{1-p}\right)$, which can take any real value $(-\infty, +\infty)$.
- We model the log-odds with a **linear function of the inputs**:

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

- We estimate the parameters $\beta_0, \beta_1, \ldots, \beta_k$ using **Maximum Likelihood**: each observed y_i contributes a probability

  $$p_i^{y_i}(1 - p_i)^{1-y_i}$$

  and we choose the parameters that maximize the likelihood of all observations.
- Once the parameters are learned, we plug the linear combination back and convert it to a probability:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k)}}$$

- This final expression is the **sigmoid (logistic) function**, which always returns a value between 0 and 1.

To estimate the parameters and how to interpret them:

- Parameters are chosen to maximize the likelihood of observing the sample.
- Unlike linear regression, there are no closed-form formulas.
- We maximize the **log-likelihood** numerically.
- ML estimates are obtained iteratively until the log-likelihood stabilizes.

- Predicted log-odds:
$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \cdots + \hat{\beta}_k x_{ki}$$
- Estimated probability:
$$\hat{P}(Y_i = 1|X_i) = \frac{e^{\hat{y}_i}}{1 + e^{\hat{y}_i}}$$
- Sign of $\hat{\beta}_k$ indicates direction of curve:
  - $\hat{\beta}_k > 0$: probability increases with x_k
  - $\hat{\beta}_k < 0$: probability decreases with x_k
- Rate of change increases with magnitude of $\hat{\beta}_k$
- Example: $\hat{malign}_i = -7.7836 + 0.6683 \cdot thickness + 0.5540 \cdot size + 0.6807 \cdot shape$
  - thickness=5, size=2, shape=2 $\Rightarrow \hat{P} \approx 0.12$
  - thickness=10, size=8, shape=9 $\Rightarrow \hat{P} \approx 0.999$

3.1.5 Lecture 5

Today we see model selection with Carina Albuquerque (breaking bad mentioned).

We want to find the best model from a set of possible ones. Either comparing different models or different hyperparameters and the same model.

We want to reduce overfitting, emphasize understanding vs memorizing the data.

The Model selection follows these steps

1. Split in Train (training the parameters, 70%), Validation (best model and parameters, 15%), Test (test the performance, 15%)
2. Choose the metric
3. Choose the eval method
4. Run the models
5. Compare metrics
6. Pick the model

To avoid overfitting we should minimize the Validation error and the Training error. When both are minimized we find the best complexity for our model. The opposite of overfitting is called underfitting.

If the dataset is imbalanced we apply stratification. Or use SMOTE.

Use k-fold CrossValidation to get the more realistic performance and improve training, usually 10 folds. We can also have Leave-One-out cross val.

Usually the best model is the one with better performance. We have a trade off between simpler models and their accuracy. Interpretable models are usually the simple ones. This is also true for time needed for the training.

Test set and hold out →Big data Cross-val →Middle data
Model Assessment is a way to quantify the strength of the model.
The usual metrics are

-

3.2 Brief recap of models and other useful things

3.2.1 Lecture 1

- **Training example:** A pair $(x, f(x))$.
- **Target function:** The true function $f(x)$ that we want to estimate.
- **Hypothesis:** A proposed function h, believed to be similar to f; the output of the learning algorithm.
- **Concept:** Boolean function that describes the instances of the dataset.
  - $f(x) = 1 \Rightarrow$ positive example
  - $f(x) = 0 \Rightarrow$ negative example
- **Classifier:** A discrete-valued function produced by the learning algorithm. The possible values of $f \in \{1, 2, \ldots, K\}$ are the *classes* or *class labels*. (In most algorithms the classifier actually returns a real-valued function that must be interpreted).
- **Hypothesis space:** The space of all hypotheses that can, in principle, be produced by the learning algorithm.
- **Version space:** The subset of the hypothesis space that has not yet been ruled out by the training examples.

3.2.2 Lecture 2

3.2.3 Lecture 3

3.2.4 Lecture 4

3.3 Things seen at the practical

4 Statistic for data science

4.1 Theory

4.1.1 Lecture 1

Statistics is the approach of answering questions using data. We aim to find rules that are true for a certain population basing our findings on a smaller subset of the mentioned population (sample)

Unit → Basic objrcts on which the data are collected
Variable → Characteristics of units that can take different values

We can have different types of variables

Categorical

- Nominal
- Ordinal

Quantitative

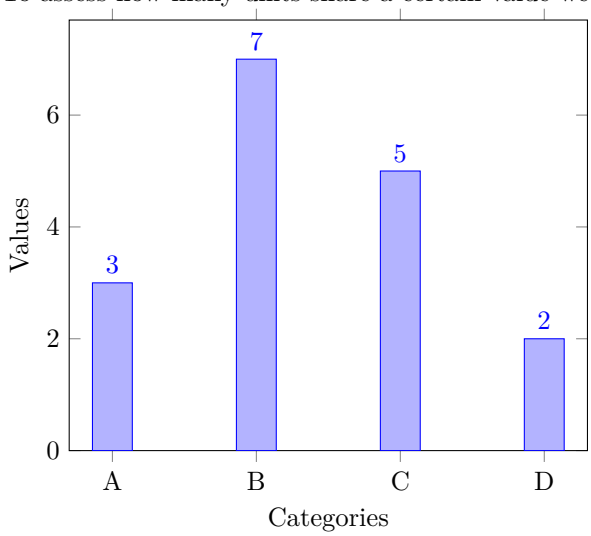
- Discrete
- Continuous

Our questions involve large **populations**. Since we can't measure all of the units of them we usually take a subset called **sample**. Values that concern the sample are called **statistics** while the ones that concern the whole population are called **parameters**.

We want to estimate in a precise way those parameters basing the computation on the sample.
One of the problems regarding the samples is that those can be biased. To avoid that we want to get a sample that is representative of the population. One way is the use of **simple random sampling method**.
Describing and visualizing data
We have different ways to describe our data, they depend on multiple factors.
Categorical variables can be described by a proportion

$$p = \frac{\text{Number in the category}}{\text{Total number}}$$

To assess how many units share a certain value we can use a **Bar chart**



One important concept in statistics is central tendency, which refers to finding the center of our data. It also helps us understand how the other values are distributed around that center. Quantitative variables can be summarized using measures of central tendency.

Mean → Sum of the data values divided by the number of the values. For the mean of the sample we use $\bar{x}$ and for the one of the population μ. The formula is

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Median → The center of sample after it was ordered from smallest to largest. Better than mean if we have outliers.

Standard deviation → It describes how the units are distributed from the mean. We can square it to get the variance. For the sample we call it s, for the population we use σ.

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

Remember that both $\bar{x}$ and s (or s^2) are relative to the sample. They are just an estimation that might be correct if we have a big enough sample.

Percentile → Is a measure that tells us the percentage of the observations that fall at or below it. At 75% we will find on the left the 75% of the observations and on the right the 25%.

Quartiles → Basically the same as the percentiles but now we are dividing the sample in 4 parts. Q_1 is the first quartile, we will find on the left 25% of the observations and the rest on the right. Q_2 is the median.

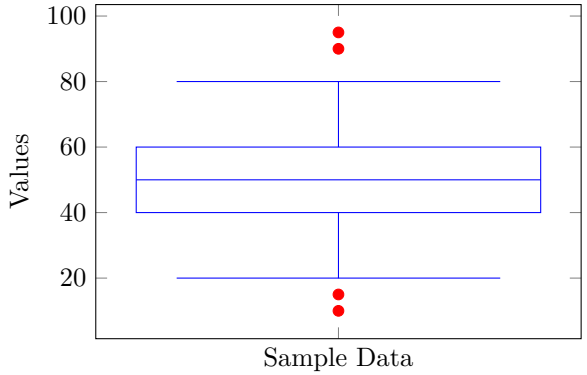
$$Q_1 = 25\text{th percentile}, \quad Q_2 = \text{median (50th percentile)}, \quad Q_3 = 75\text{th percentile}$$

$$\text{Interquartile Range (IQR)} = Q_3 - Q_1$$

$$\text{Lower fence} = Q_1 - 1.5 \cdot \text{IQR}, \quad \text{Upper fence} = Q_3 + 1.5 \cdot \text{IQR}$$

$$\text{Outliers} = \text{values outside [Lower fence, Upper fence]}$$

We can visualize those statistics using a **Boxplot**

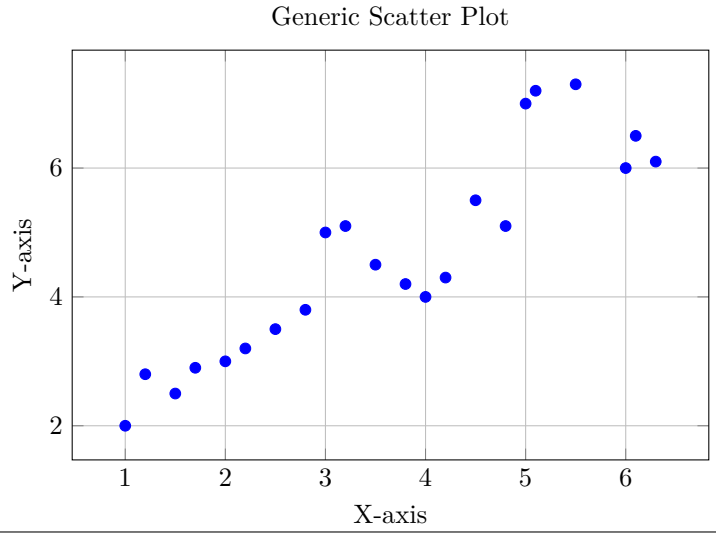


Correlation → Describes the strength and direction between two quantitative variables. For the sample we use r, for the population we use ρ.

Correlation has many properties:

1. ρ is bounded in [-1, 1]
2. $\rho > 0$ means positive association, $\rho < 0$ negative, $\rho = 0$ no relationship
3. The bigger $|\rho|$ the stronger the correlation
4. The sign tells the direction
5. ρ is unit free
6. ρ(X, Y) = ρ(Y, X)

To visualize correlation we use a **Scatterplot**



4.1.2 Lecture 2

Since random variables can take certain values with certain probabilities we would like to describe them somehow. The collection of those probability is called probability distribution. This can describe the probability of each event happening. Recall that the sum of the probabilities is 1.

CDF -> the **cumulative distribution function** gives me the probability that a random variable is less than or equal to a given value.

$$F(x) = P(X \leq x) \quad \text{for all } x \tag{12}$$

PMF/PDF -> the **probability mass function** or **probability density function** (for discrete or continuous case respectively) will give us the probability of a certain event happening

$$f(x) = P(X = x), \quad \text{for all } x \tag{13}$$

$$F(x) = \int_{-\infty}^{x} f(t)\,dt, \quad \text{for all } x \tag{14}$$

For the continuous case we calculate the integral between the 2 values that we care

Discrete distributions
If we can count the sample space we say that our random variable is discrete
Recall

$$Var[X] = E[X^2] - E[X]^2 \tag{15}$$

Bernoulli -> the probability that a certain event will happen

$$X \sim Ber(p) \tag{16}$$

$$P(X = 1) = p = 1 - P(X = 0) = 1 - q \tag{17}$$

With PMF

$$f(x) = p^x(1-p)^x \quad \text{for x} \in \{0, 1\} \tag{18}$$

Where

- p -> positive probability
- q -> negative probability, 1-p basically

And with

- E[X] = p
- Var[X] = pq = p(1-p)

Binomial -> probability of getting a certain number of getting n independent Bernoulli outcome

$$X \sim Bin(n, p) \tag{19}$$

$$P(X = x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{20}$$

Where

- n -> number of trials
- p -> positive probability
- x -> successes that we want to calculate

Recall that

$$\binom{n}{x} = \frac{n!}{x!(n-x)!} \tag{21}$$

And with

- E[X] = np
- Var[X] = np(1-p)

Poisson -> probability of getting a certain number of events during a fixed interval. We need to know the probability of the event happening.

$$X \sim Poi(\lambda) \tag{22}$$

$$f(x) = P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!} \tag{23}$$

Where

- e -> Euler number
- x -> number of successes
- λ -> known mean rate

With

- E[X] = Var[X] = λ

4.1.3 Lecture 3

Continuous distributions

Normal distribution -> If we want to describe a series of random events that have a tendency of being near a certain mean.

$$X \sim N(\mu, \sigma^2 \tag{24}$$

$$\phi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} \tag{25}$$

- E[X] = μ
- Var[X] = σ^2

It has a bell shape and if the parameter are $\mu = 0$ and $\sigma^2 = 1$ we say that is a standard normal distribution known as Z distribution. σ^2 will tell us how thin or wide is the distribution, so how variable it is, and so how unpredictable.

Kurtosis -> Measure of how heavy the tails are, how frequent are the outliers

Chi squared -> If we have n independent X variables that are distributed with the Z distribution if we square and sum them we say that this quantity follows a Chi-squared distribution

$$V = X_1^2 + X_2^2 + ... + X_n^2 \sim \chi^2_{(n)}, n > 0 \tag{26}$$

With

- E[V] = n
- Var[V] = 2n

The n parameter is called **degrees of freedom**. The shape is asymmetric

Student's t distribution → If we have a random variable Z distributed as a standard normal distribution and another independent variable V following a Chi-squared distribution with n degrees of freedom, then the following ratio defines a random variable that follows a t-distribution:

$$T = \frac{Z}{\sqrt{V/n}} \sim t_{(n)}, \quad n > 0 \tag{27}$$

With

- $E[T] = 0 \quad$ (for $n > 1$)
- $Var[T] = \dfrac{n}{n-2} \quad$ (for $n > 2$)

The parameter n is called **degrees of freedom**. The distribution is symmetric around zero and has heavier tails than the normal distribution. As $n \to \infty$, the t-distribution approaches the standard normal distribution.

Snedecor's F distribution → If we have two independent random variables U and V such that $U \sim \chi^2_{(n_1)}$ and $V \sim \chi^2_{(n_2)}$, then the following ratio defines a random variable that follows an F-distribution:

$$F = \frac{(U/n_1)}{(V/n_2)} \sim F_{(n_1,n_2)}, \quad n_1, n_2 > 0 \tag{28}$$

With

- $E[F] = \dfrac{n_2}{n_2 - 2} \quad$ (for $n_2 > 2$)
- $Var[F] = \dfrac{2n_2^2(n_1 + n_2 - 2)}{n_1(n_2 - 2)^2(n_2 - 4)} \quad$ (for $n_2 > 4$)

The parameters n_1 and n_2 are called **degrees of freedom** of the numerator and denominator, respectively. The distribution is asymmetric and always non-negative. It is mainly used to compare variances, especially in the context of ANOVA (Analysis of Variance) and regression models.

To summarize

Distribution	**PDF**	**CDF**	**Quantile**
Normal	dnorm	pnorm	qnorm
Chi-Squared	dchisq	pchisq	qchisq
Student-t	dt	pt	qt
F	df	pf	qf

4.1.4 Lecture 4

Today point estimation, hypothesis testing and confidence interval, from a sample we try to generate a good models

Suppose that we have a population distributed with certain parameters. Like μ as mean or p as proportion. Since we can't check the full population (usually), we start from a sample and we try to see if the estimate can be good enough.

We have two main ways

- Method of moments
- MLE

To start we need to define the **Parameter space**, the range of possible values of the parameter θ inside the parameter space Ω. Let $X_1, X_2, ..., X_n$ be n random variables that can take a value $x_1, x_2, ..., x_n$. The parameter space is bounded.

To estimate the θ we use a point estimator. For example the mean is a point estimator of the μ.

$$\bar{X} = \frac{1}{n}\Sigma X_i \tag{29}$$

We define the statistic T$(X_1, ..., X_n)$ as point estimator to get the parameter estimation. If T$(x_1, ..., x_n)$ we get the estimate.

The first way is the **Method of moments**, we equate sample moments with theoretical moments.

- E(X^k) is the kth theoretical moment of the distribution, k=1,2,3....
- $m_k = \frac{1}{n}\Sigma X_i^k$

We equate the first k sample moments to the corresponding k population moments until we have as many equations as se have parameters, then we solve the resultant system.

The second way is the so called **Maximum Likelihoodestimator**. We start from the assumption that in our sample the values follow a distribution based on an unknown parameter θ. We want to find a good point estimator and estimate for θ using the data from our sample. The idea is that we want to find the parameter that maximise the probability of getting those values. If $X_1, ..., X_2$ are identically independent distributed from a population with PDF f(x $\mid$ $\theta_1, ..., \theta_k$) the likelihood function is defined by

$$L(\theta|x) = L(\theta_1, ..., \theta_k|x_1, ..., x_k) = \Pi f(x_i|\theta_1, ..., \theta_k) \tag{30}$$

After we take the log of the likelihood to obtain the log-likelihood function. This is because the logarithm is a strictly increasing function so maximising the log of the likelihood is like maximising the likelihood itself. The candidates are the values that put the derivative with respectively to θ_i of the log-likelihood at 0. Also is easier to differentiate a log and working with sums

$$logL(\theta|x) = \Sigma log(f(x_i|\theta_1, ..., \theta_k)) \tag{31}$$

$$\frac{d}{d\theta_i} logL(\theta|\mathrm{X}) = 0 \tag{32}$$

We want to assess how good are our estimates. To do it we want to look at the sampling distribution that we receive. Our estimators have some properties.

- **Unbiasedness** →An estimator T is unbiased iff E(T) = θ for every θ in Ω. Usually the population mean is an ubiased estimator.
- **Efficiency** →We have two unbiased estimators T and T' for the same parameter θ. T is more efficient than T' iff

$$Var(T) \leq Var(T') \tag{33}$$

T is the most efficient if the equation is true for any other unbiased estimator T'. This is called absolute efficiency, before it was only relative efficiency. To know that we have absolute efficiency we use the Frechet-Cramer-Rao inequality.

Let $(X_1, X_2, \ldots, X_n)$ be a sample from a population with pdf $f(x \mid \theta)$ and let $T = T(X_1, X_2, \ldots, X_n)$ be an unbiased estimator of θ. Then

$$\operatorname{Var}(T) \geq \frac{1}{nI(\theta)}$$

where

$$I(\theta) = \mathrm{E}\left[\left(\frac{\partial \ln f(X \mid \theta)}{\partial \theta}\right)^2\right] = -\mathrm{E}\left[\frac{\partial^2 \ln f(X \mid \theta)}{\partial \theta^2}\right]$$

and $I(\theta)$ is the Fisher information.

This inequality allows us to find a lower limit to compare our variance with.

If we want to compare relative efficiency, we can use the mean squared error (MSE).

Let $T = T(X_1, \ldots, X_n)$ be an unbiased estimator of the parameter θ. The mean squared error is defined as

$$\operatorname{MSE}(T) = \mathrm{E}[(T - \theta)^2]. \tag{34}$$

The estimator with the lower MSE is considered more efficient.

- **Consistency** →Final property, it tells us that if the sample increases the precision of the estimator will also do that. An estimator T_n is said to be consistent in quadratic mean iff

$$lim_{n \to +\infty} \mathrm{E}[(T - \theta)^2] = 0 \tag{35}$$

4.2 Exercises

- We have a N∼(102, 8) find the area less than 120

```
pnorm(120, mean=102, sd=8, lower.tail=TRUE)
```

- Same as above but we want the area more than 120

```
pnorm(120, mean=102, sd=8, lower.tail=FALSE)
```

- Proportion of a Z distribution where the z score is between 0 and 1.75

```
pnorm(1.75, mean=0, sd=1, lower.tail=TRUE) -
pnorm(0, mean=0, sd=1, lower.tail=TRUE)
```

- Proportion of a Z distribution more extreme than ±2

```
pnorm(2, mean=0, sd=1, lower.tail=False)*2
```

- Which z-scores separate middle 90% and the outer 10%

```
qnorm(0.95, mean=0, sd=1, lower.tail=TRUE)
```

- We have a N∼(102, 8), which value separates the top 10%

```
qnorm(0.90, mean=102, sd=8, lower.tail=TRUE)
```

- Find the 95th percentile of a chi-squared with 7 df

```
qchisq(0.95, df=7)
```

- Find 2.5th and 97.5 percentiles of a Student distribution with 5 df

```
qt(0.95, df=5)
```

```
or
```

```
qt(c(0.25, 0.975), df=5)
```

- Finde 95th percetile of an F distribution with 12, 5 df

```
qt(0.95, df1=12, df2=5)
```

4.3 Code