Data preprocessing

Hands on python and pandas

Python

Cross-platform interpreted language

- Available on the main OS (Linux, Mac, Windows, etc.)
- CPython is the reference implementation plus other alternatives
- Can be integrated into other languages (C, C++, Java, etc.)

Created in the late 80s, became popular in the 2000s

Multi-paradigm

- Imperative, object-oriented
- Syntax can be easily extended

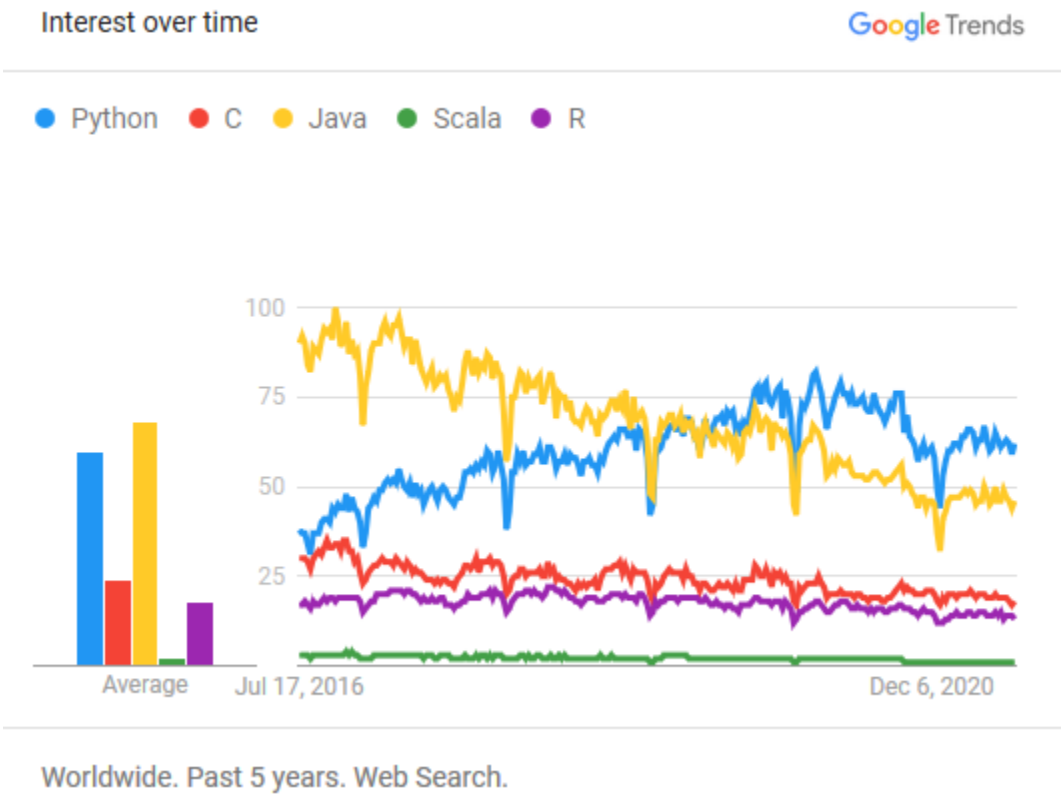
Emphasis on the on ease of reading and writing

- “there should be one—and preferably only one—obvious way to do it”

Python

Why Python?

- Easy to learn
- Used for multiple purposes (scripting, data science, etc.)
- Used for prototyping and rapid development cycles
- Rising popularity
  - Includes a standard library
  - Wide availability of external libraries
  - E.g., machine learning, deep learning



Python

Python features make it suitable for analysis operations

- Interactively usable, scripts, and complete programs

Several libraries that make Python a complete data analysis environment

- Python is increasingly used as a replacement for R and other ad-hoc software
- E.g., NumPy for the representation of data in the form of vectors and matrices
- E.g., Pandas for the manipulation and transformation of tabular data
- E.g., Sklearn for the application of machine learning and data mining algorithms
- E.g., Matplotlib for data visualization

Python

Major versions of Python

- Python 2, end of support in 2020
- Python 3

We will use Python 3

Python

A Python statement is contained by default in a row

- print("Hello, world")

Write several separate line instructions with “;”

- print("Hello"); print("world")

Comments begins with “#” and end at the end of the line

- # this is a comment
- print("Hello, world") # another comment

A statement can continue in the next row

- Explicit: the row ends with “\”
- Implicit: if there are unclosed brackets

```
print("Hello,
  " + "world")
```

Python

In other languages, code blocks (if, for, etc.) are usually delimited by specific symbols (e.g., “{” and “}”)

- Indentation is used for better readability

Python only uses indentation to delimit code blocks

- Each row introducing a block (e.g., if) ends with “:”
- Rows within the same block are indented the same (i.e., same number of spaces)
- An empty block contains the keyword “pass”

```java
// esempio in Java
nums = getNumbers();
for (int x: nums) {
    if (x < 0) {
        System.out.println(x);
    }
}
System.out.println("end");
```

```python
# esempio in Python
nums = get_numbers()
for x in nums:
    if x < 0:
        println(x)
println("end")
```

Python

Everything is an object: numbers, lists, functions, etc.

- Not the same as Java, where int and float are not objects

Objects have attributes and methods accessed via the "." syntax

- object.attribute

The object type determines the available attributes and operations

- Object types are known only at execution time
- Not the same as Java, where object types are known at compile time (i.e., before execution)

The object None (with type NoneType) represents an absence of value

- As null in Java (which is not an object)

Python

Python introduces different object collections

- E.g., lists (["cat", "cat", "dog"]), sets ({"cat", "dog"}), dictionaries ({"key": "value"})
- A collection can contain objects with heterogeneous types (e.g., the list [1, "cat"])
- Collections can be nested (e.g., the list ["cat", ["cat", ["dog"]]])

Collections can be mutable or immutable

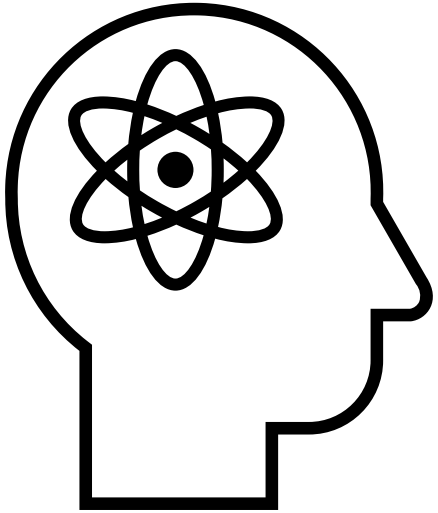
- Mutable: it is possible to add/remove/replaces elements
- Immutable: it is not possible to modify the collection (e.g., numbers, booleans, strings)

Strings (str) are immutable sequences of characters

- A character is a string with length 1, there is not char type

In action!

Enter the notebook `00-PythonFundamentals`

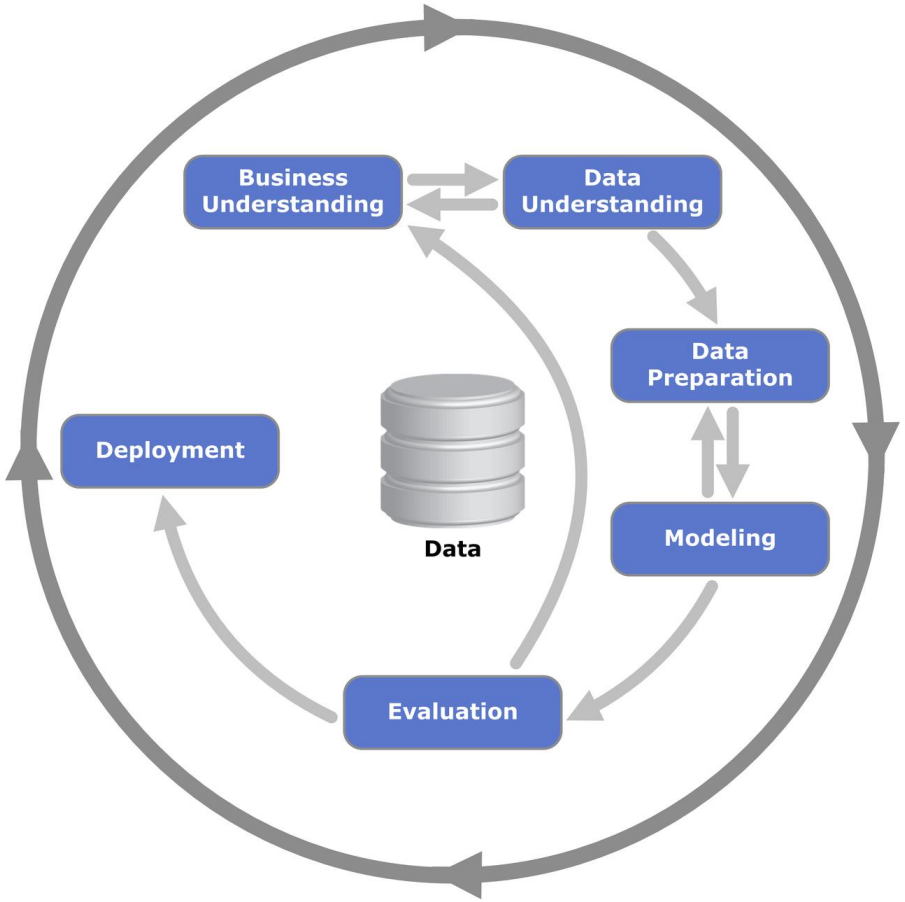


https://github.com/w4bo/2022-bbs-dsaa/blob/master/materials/00-PythonFundamentals.ipynb

Data analysis

Data analysis involves several steps:

- Collection of data from one or more sources (database, web, etc.)
- Understanding of the structure and meaning of data
- Transformation of data into manageable formats for subsequent steps
- Extraction of knowledge from data (statistics, models, patterns, etc.)
- Validation of the extracted knowledge
- Deployment of the extracted knowledge and models



Data preparation

(Relational) data are usually collected in tabular format

- Each row is an observation (or instance)
  - An object of the analysis
  - E.g., a product for market basket analysis
- Each column is an attribute (or feature) characterizing each object
  - All values within a column have the same type (i.e., all values belong to the same attribute domain)
  - E.g., the attributes ID (int), ProductName (str), or Price (float)

"It is imperative to know the attribute properties to carry out meaningful operations and research with them"

- It makes sense to compute the average price but not the average ID
- The attribute type determines which operator can be applied to the attribute
  - E.g., equality, sort, sum, ratio

Data preparation

Different attribute types

- **(Categorical) Nominal:** can distinguish the values (i.e., check equality)
- **(Categorical) Ordinal:** can distinguish and sort the values
- **(Numeric) Interval:** can distinguish and sort the values, and compute their difference
- **(Numeric) Ratio:** can distinguish and sort the values, and compute their difference and ratio

ID	PriceBin	Date of sale	Quantity
xxx	low	05/07/2021	10
xxy	medium	05/07/2021	50
xyz	high	02/06/2021	100

Pandas

pandas (Python) is a solution for the manipulation of relational data

- Two main data types: **Series** (e.g., temporal series) and **DataFrame** (e.g., table)
- Support to SQL-like operations (join/merge, aggregation, etc.)
- Imputation of missing values
- Manipulation of data shape
- By convention, the package pandas is imported as “pd”

```
>>> import pandas as pd
```

Pandas

A series is a sequence of values with the same type

- Each value is associate with a label
- Supported values and label types are the ones from NumPy (float64, int64, etc.)
- In other words, a series is a mono-dimensional vector of elements

The index of a series is the sequence of labels

- Label are usually numeric or string identifiers
  - E.g., the primary key of a database table
- Labels could repeat within the series, but usually do not

Pandas

The constructor of a Series requires value and (optionally) index attributes

- >>> ser = pd.Series([4 , 7 , -5 , 3],
  ... index=["d", "b", "a", "c"])
- If unspecified, the index is a sequence of integers ranging from 0 to N-1

```
>>> ser
d    4
b    7
a   -5
c    3
dtype: int64
```

Pandas

Series support binary operators

- +, -, *, etc.
- Custom functions

Binary operators are applied to the values with the same label

- ... and by the value order
- The result will contain NA for each unmatched label (i.e., a label contained in one series but not the other)

Attribute types

Attribute type can be specified during the creation of a series

Common data types are numeric ones

- np.floatN represents floating numbers (e.g., -3.14)
- np.intN / np.uintN represent integers with/without sign (-42 and 42)
- N is the number of needed bits: 8, 16, 32 o 64

Other data types

- bool: Boolean values
- datetime64, timedelta64: timestamp and time intervals
- object: mainly used for strings

Why is data type important?

Why is data type important?

Microsoft has released an emergency fix for a year 2022 bug that is breaking email delivery on on-premise Microsoft Exchange servers.

As the year 2022 rolled in and the clock struck midnight, Exchange admins worldwide discovered that their servers were no longer delivering email. After investigating, they found that mail was getting stuck in the queue, and the Windows event log showed one of the following errors.

```
Log Name: Application
Source: FIPFS
Logged: 1/1/2022 1:03:42 AM
Event ID: 5300
Level: Error
Computer: server1.contoso.com
Description: The FIP-FS "Microsoft" Scan Engine failed to load. PID: 23092, Error
Code: 0x80004005. Error Description: Can't convert "2201010001" to long.
```

```
Log Name: Application
Source: FIPFS
Logged: 1/1/2022 11:47:16 AM
Event ID: 1106
Level: Error
Computer: server1.contoso.com
Description: The FIP-FS Scan Process failed initialization. Error: 0x80004005. Err
or Details: Unspecified error.
```

These errors are caused by Microsoft Exchange checking the version of the FIP-FS antivirus scanning engine and attempting to store the date in a signed int32 variable.

A signed integer is a 32-bit datum that encodes an integer in the range

$[-2^{31}, 2^{31}-1] =$

[-2147483648 to 2147483647]

`2201010001 > 2147483647`

https://www.bleepingcomputer.com/news/microsoft/microsoft-releases-emergency-fix-for-exchange-year-2022-bug/

Missing values

Datasets often show missing values

- E.g., they are not applicable (e.g., date of death) or unknown

A series can have missing values, referred to as NA (Not Available)

- Numeric attributes: NA is np.nan (Not a Number)
- nan is never equal, greater, or lower than other values (nor itself)

```
>>> np.nan == np.nan
False
```

- Numeric expressions with nan return nan

```
>>> 2 * np.nan – 1
nan
```

Which problems arise?

Missing values

- isna and notna verify (not) missing values and return a Boolean series
- dropna removes missing values
  - Return a new series by default
  - Modify the same series if inplace=True

```
>>> s = pd.Series(
  [1,2, np.nan, 4],
  index=list("abcd"))
>>> s.isna()
a    False
b    False
c     True
d    False
>>> s.count()
3
>>> s.dropna()
a    1.0
b    2.0
d    4.0
>>> s.dropna(inplace=True)
[...]
```

a	1
b	2
c	NA
d	4

Missing values

fillna replaces NA values

- Return a new series or modify the given one if inplace=True
- All NA are replaced by the same value
  - Usually, the average
- If the attribute method is set to ffill or bfill, each NA is replaced by the first preceding/posterior not NA value (if any)
  - Useful for temporal series

```
(A) s.fillna(s.mean())
(B) s.fillna(method="ffill")
(C) s.fillna(method="bfill")
```

	s	(A)	(B)	(C)
a	1	1	1	1
b	NA	2	1	2
c	NA	2	1	2
d	2	2	2	2
e	3	3	3	3
f	NA	2	3	NA

Aggregation

Series of methods to compute aggregated statistics

- sum, mean, min, max, count
- NA values are ignored by default

```
>>> pd.Series([2, np.nan, 6, 4]).mean()
4.0
```

If skipna=False, NA values invalidate the statistics

```
>>> pd.Series([2, np.nan, 6, 4]).mean(skipna=False)
nan
```

idxmin e idxmax, return the label of the minimum/maximum value

```
>>> pd.Series({"a": 6, "b": 10, "c": 7}).idxmax()
'b'
```

Value distribution

- unique returns an array with distinct values, values are sorted by first appearance
- nunique returns the quantity of unique values

```
>>> x = pd.Series(
  [6, 2, 1, 2, 5, 1, 1])
>>> x.unique()
array([6, 1, 2, 5])
>>> x.nunique()
4
```

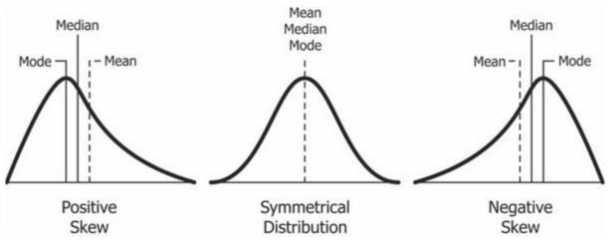
Value distribution

```
>>> x = pd.Series(
  [6, 2, 1, 2, 5, 1, 1])
```

- value_counts returns a new series that associates each value with its number of occurrences, sorted by frequency

```
>>> x.value_counts()
1    3
2    2
6    1
5    1
dtype: int64
```

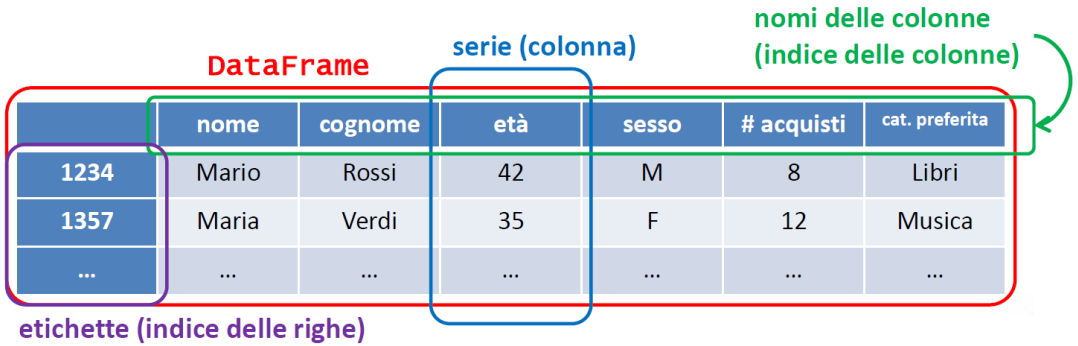
Which problems can cause skewed distributions?



Dataframe

A DataFrame represents relational data

- It can be seen as a set of columns, each column is a series of a given type, all columns share the labels
- Labels are unique row identifiers
- Each series (column) has a unique name, the name is used to access the column values



Dataframe

As in a SQL projection, a column can be selected by name

```
>>> df.year
>>> df["year"]
```

```
>>> df
       year   state  pop
one    2000    Ohio  1.5
two    2001    Ohio  1.7
[...]
six    2003  Nevada  3.2
>>> df["year"]
one      2000
two      2001
[...]
six      2003
Name: year, dtype: int64
```

Dataframe

A DataFrame can be imported from external sources

- pd.read_csv returns DataFrame from a given CSV file
- Data types are automatically inferred (integers, floating numbers, objects, etc.)

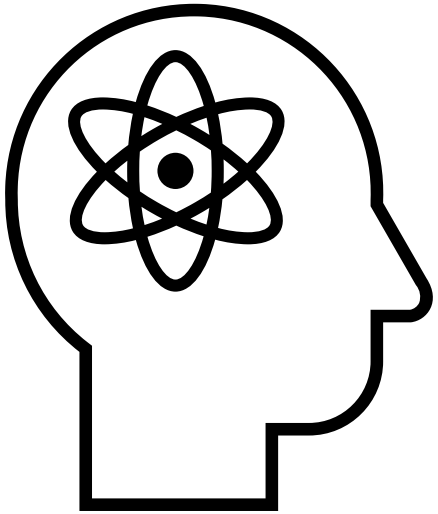
```
>>> data = pd.read_csv("mydata.csv")
```

read_csv has many optional parameters, such as:

- sep: column separator (default: “,”)
- names: column names (default: the first row from the file)
- index_col: column to use as index, indexes can be nested
- dtype: column types
- nrows: maximum number of rows

In action!

Enter the notebook `01-DataPreprocessing`



https://github.com/w4bo/2022-bbs-dsaa/blob/master/materials/01-DataPreprocessing.ipynb

Data preprocessing

Data preprocessing

Discussion time!

Data preprocessing

Data preprocessing plays a key role in a data analytics process [1]

- A broad range of activities; from correcting errors to selecting the most relevant features
- There are no pre-defined rules on the impact of pre-processing transformations
- Data scientists cannot easily foresee the impact of pipeline prototypes and hence require a method to discriminate between them and find the most relevant ones

[1] Joseph Giovanelli, Besim Bilalli, Alberto Abelló: Effective data pre-processing for AutoML. DOLAP 2021: 1-10

Data preprocessing

Data preprocessing avoids "*Garbage in*, garbage out"

- "Garbage in, garbage out" is particularly applicable to data mining and machine learning
- Indeed, data-collection methods are often loosely controlled, resulting in:
  - Out-of-range values (e.g., Income: −100)
  - Impossible data combinations (e.g., Exam mark: 15, Exam result: Passed)
  - Missing values
  - Inconsistent data among multiple sources
  - More?

Data preprocessing

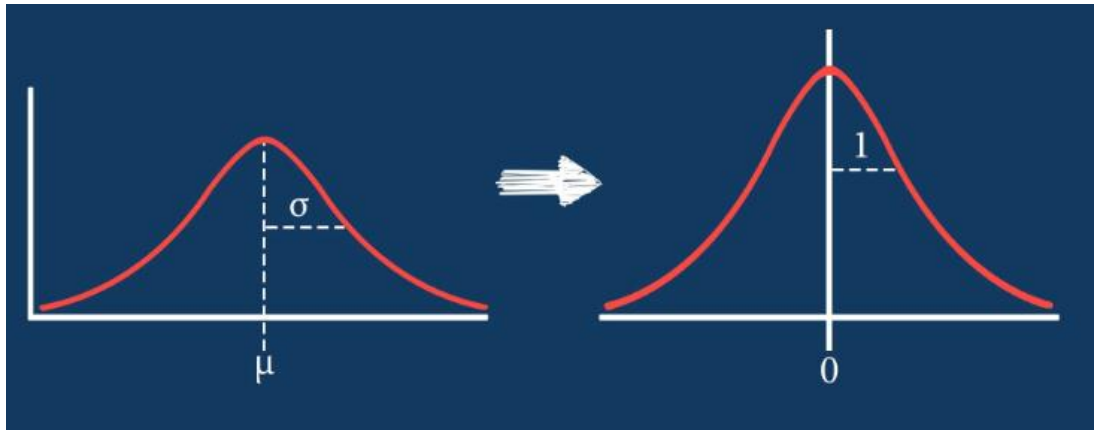
Which transformations can we apply?

- Encoding: transforming categorical attributes into continuous ones
- Discretization: transforming continuous attributes into categorical ones
- Normalization: normalizing continuous attributes such that their values fall in the same range
- Imputation: imputing missing values
- Rebalancing: adjusting the class distribution of a dataset (i.e., the ratio between the different classes/categories represented)
- Feature Engineering: defining the set of relevant attributes (variables, predictors) to be used in model construction

Data preprocessing

Things are even more complex when applying sequences of transformations

- E.g., normalization should be applied before rebalancing since rebalancing (e.g., by resampling) alters average and standard deviations
- E.g., applying feature engineering before/after rebalancing produces different results which depends on the dataset and the algorithm



More an art than a science

- At least for now

Data preprocessing

First, collect and understand the data

- Data integration: data are usually spread across multiple (even inconsistent) documents/files
  - We keep things simple: you have already downloaded integrated *.csv files
- Visualization helps the process of understanding the data

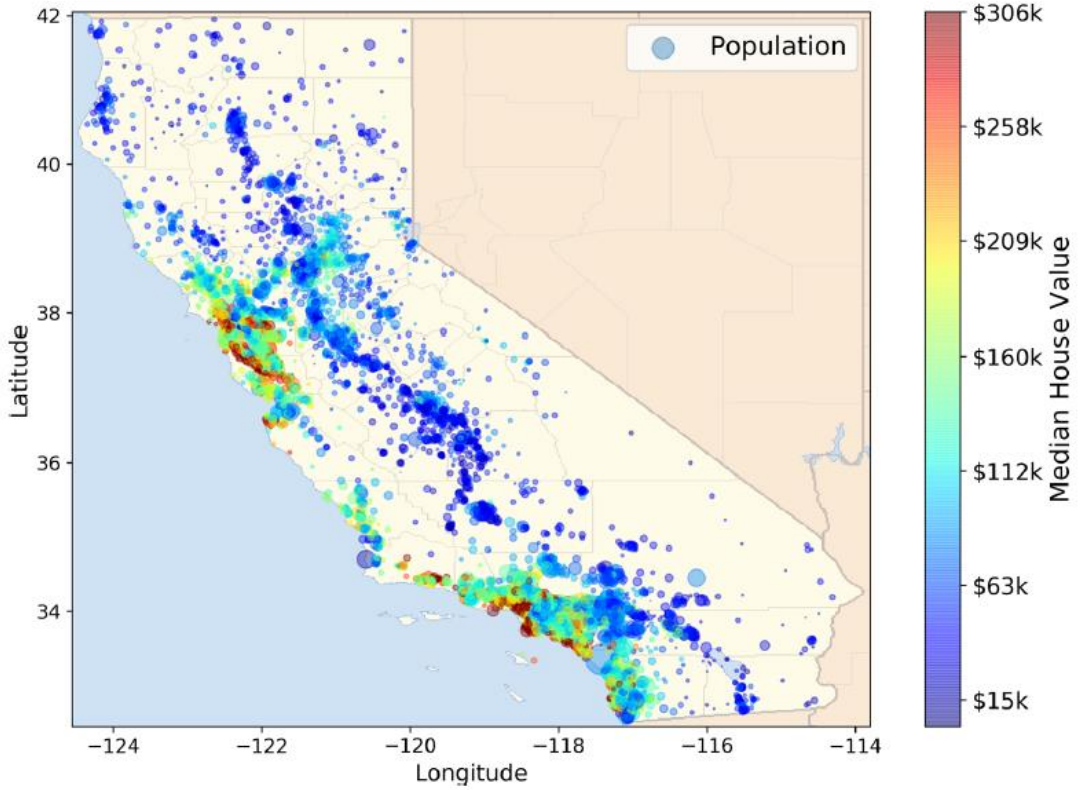
Then, ask (yourself) some questions:

- Which attributes (i.e., columns) are contained in the dataset?
- Which is the distribution of each attribute?
- Which is the range of each attribute?
- How do we treat missing data?

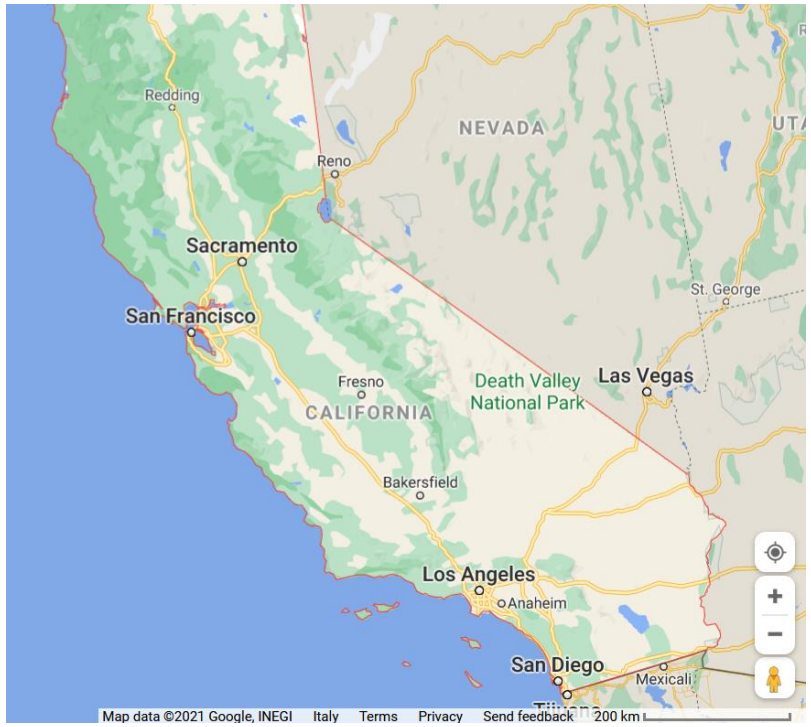
Integrated analytics lab

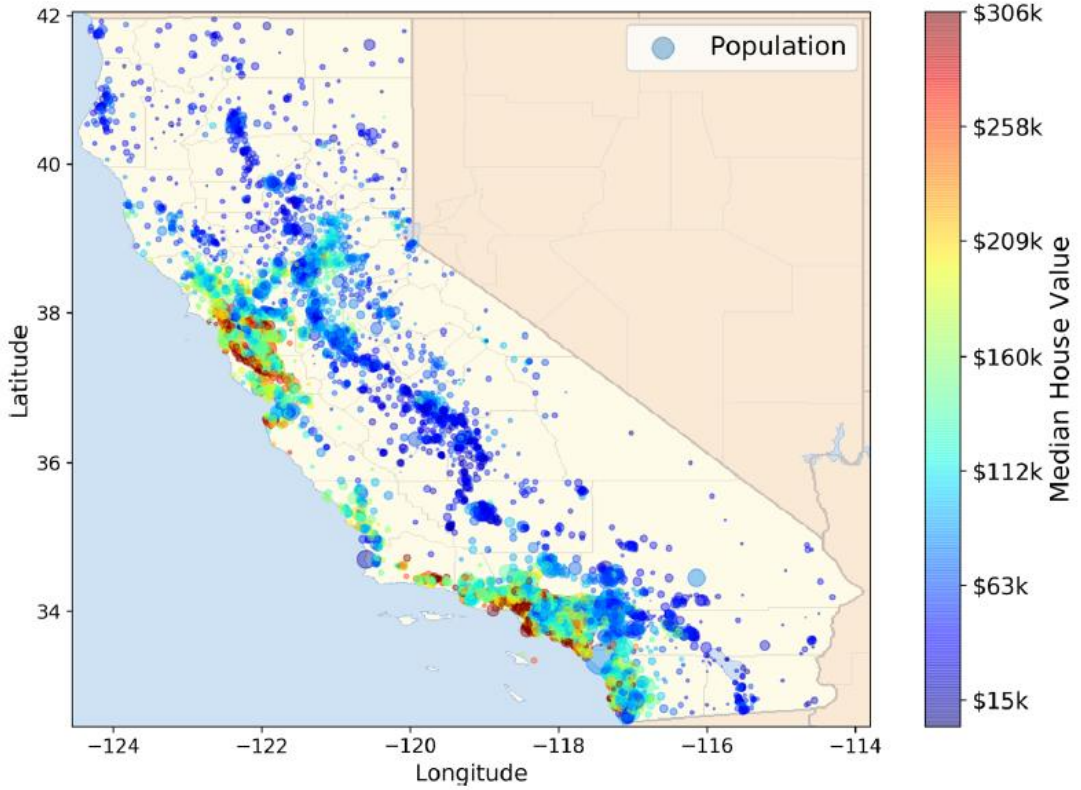
This checklist can help you while building your projects

- Frame the problem and look at the big picture
- *"We'll use the California Housing Prices. Our task is to use California census data to forecast housing prices given the population, median income, and median housing price for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people)"*



Integrated analytics lab

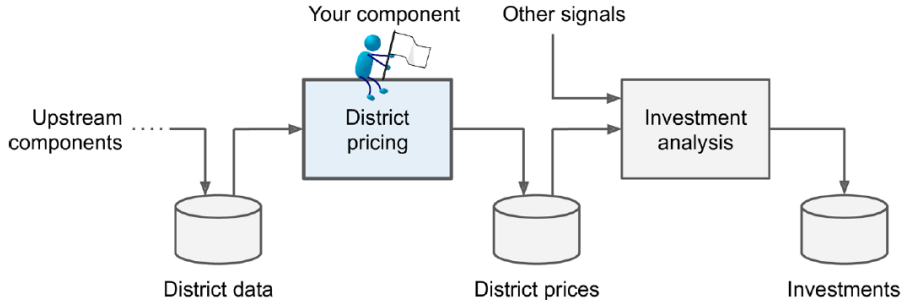




Integrated analytics lab

This checklist can help you while building your projects

- Frame the problem and look at the big picture
  - Knowing the objective is important because it will determine how you frame the problem, which algorithms you will select, which performance measure you will use to evaluate your model, and how much effort you will spend tweaking it.
  - *"Your boss answers that your model's output (a prediction of a district's median housing price) will be fed to another Machine Learning system, along with many other signals. This downstream system will determine whether it is worth investing in a given area or not. Getting this right is critical, as it directly affects revenue."*



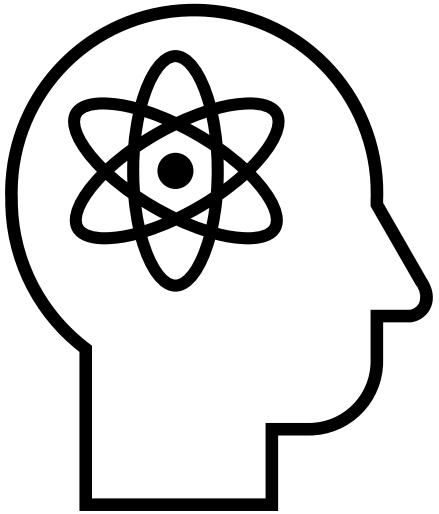
Integrated analytics lab

This checklist can help you while building your projects

- Frame the problem and look at the big picture
  - ✔ Define the objective in business terms
  - ✘ How should performance be measured? (postponed for later)
- Get the data
  - ✔ List the data you need and how much you need
    - In typical environments your data would be available in a relational database (or some other common data store) and/or spread across multiple tables/documents/files
    - In this project, however, things are much simpler
- Explore the data to gain insights
  - ✔ Create an environment to keep track of your data exploration
    - You have been provided with notebook environments
  - ✔ Study each attribute and its characteristics
    - Let's do this!

In action!

Enter the notebook `01-DataPreprocessing`



https://github.com/w4bo/2022-bbs-dsaa/blob/master/materials/01-DataPreprocessing.ipynb

Integrated analytics lab

This checklist can help you while building your projects

- Frame the problem and look at the big picture
  - ✔ Define the objective in business terms
  - ✔ How should performance be measured?
- Get the data
  - ✔ List the data you need and how much you need
- Explore the data to gain insights
  - ✔ Create an environment to keep track of your data exploration
  - ✔ Study each attribute and its characteristics
- Prepare the data
  - ✔ Fix or remove outliers (optional)
  - ✔ Fill in missing values (e.g., with zero, mean, median…) or drop their rows (or columns)
  - ✔ Feature selection (optional): drop the attributes that provide no useful information for the task
  - ✔ Feature engineering, where appropriate: discretize continuous features