Data preprocessing

Hands on python and pandas

Python

Linguaggio interpretato cross-platform

- Disponibile per i principali SO (Linux, Mac, Windows, etc.)
- Un’implementazione di riferimento (CPython) più altre alternative
- Integrabile in altri linguaggi (C, C++, Java, etc.)

Creato alla fine degli anni ’80, divenuto popolare nei 2000

Multi-paradigma

- Imperativo, object-oriented
- Sintassi facilmente estendibile ad altri paradigmi

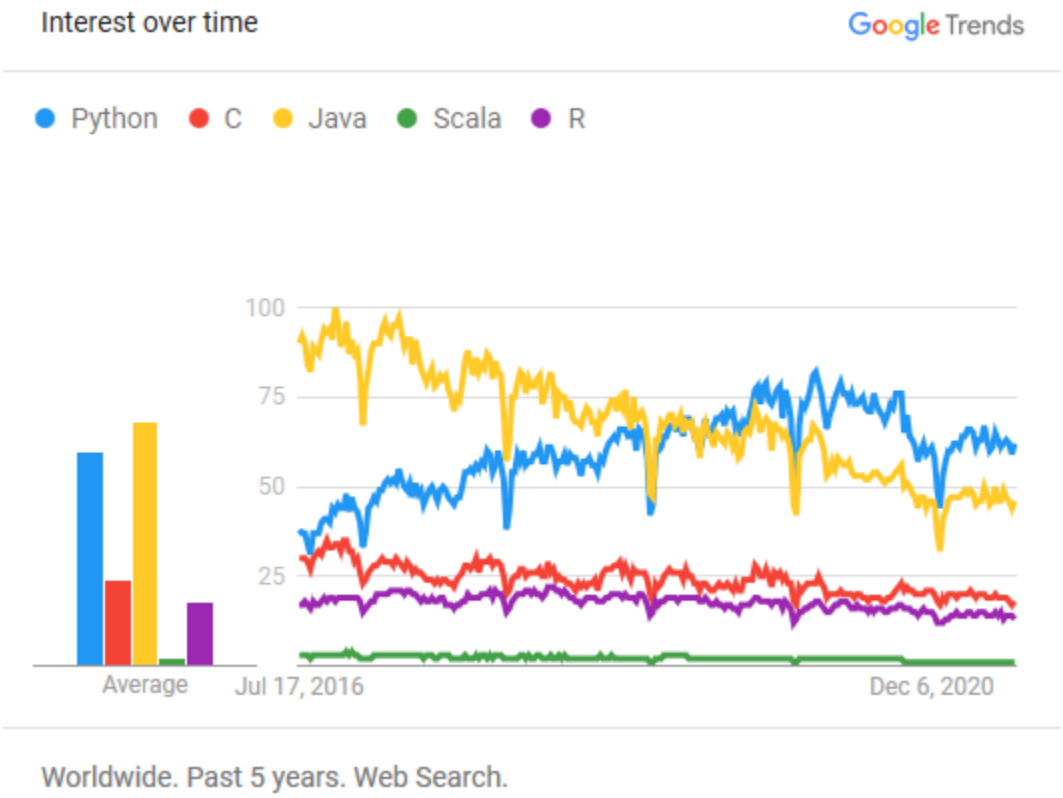
Enfasi sulla facilità di lettura e scrittura del codice

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

Python

Perché Python?

- Facile da imparare
- Usato per molteplici scopi (scripting, data science, etc.)
- Usato per prototipazione e cicli di sviluppo rapidi
- Popolarità in aumento
  - Include una libreria standard
  - Ampia disponibilità di librerie esterne
    - E.g., machine learning, deep learning



Python

Le caratteristiche di Python lo rendono adatto per operazioni di analisi

- Utilizzabile in modo interattivo, script, e programmi completi

Diverse librerie che rendono Python un ambiente completo di analisi dati

- Python è sempre più usato in sostituzione di R e altri software ad-hoc
- E.g., NumPy per la rappresentazione di dati in forma di vettori e matrici
- E.g., Pandas per la manipolazione e trasformazione di dati tabellari
- E.g., Sklearn per l'applicazione di algoritmi di machine learning e data mining
- E.g., Matplotlib per visualizzazione di dati

Python

Sono diffuse due diverse versioni major di Python

Su Python 2 si basa molto software tutt'ora in uso

- L'ultima versione minor prevista è la 2.7, rilasciata nel 2010
- Il termine del supporto è previsto nel 2020

Python 3 introduce novità incompatibili con Python 2

- Molte librerie di uso comune sono state (ri)scritte per funzionare con entrambe le versioni

Useremo Python 3

Python

Un’istruzione Python è contenuta di default in una riga

- print("Hello, world")

Si possono però scrivere più istruzioni in riga separate con “;”

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

I commenti sono introdotti da “#” e finiscono a fine riga

- # Questo è un commento
- print("Hello, world") # altro commento

Si può far continuare un’istruzione in una riga successiva

- esplicitamente se la riga termina in “\”
- implicitamente se ci sono parentesi non chiuse (più comune)

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

Python

In altri linguaggi i blocchi di codice (if, for, etc.) sono delimitati da simboli specifici (spesso “{” e “}”)

- L’indentazione è usata convenzionalmente per migliore leggibilità

Python usa l’indentazione come sintassi per i blocchi

- Ogni riga che introduce un blocco (e.g., if) termina in “:”
- Le righe a pari livello sono indentate con pari numero di spazi
- Per indicare un blocco vuoto si usa la parola chiave “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

In Python ogni cosa è un oggetto: numeri, liste, funzioni, …

- Al contrario di Java, dove non sono oggetti valori int, float,…

Ogni oggetto ha attributi e metodi, accessibili tramite la tipica sintassi

- oggetto.attributo

Il tipo di un oggetto ne determina gli attributi esistenti e le operazioni che si possono compiere su di esso

- I tipi degli oggetti sono noti solo durante l'esecuzione
- Al contrario di Java, dove i tipi degli oggetti sono verificati prima dell'esecuzione

Oggetto predefinito None (di tipo NoneType) indica assenza di valore

- Simile a null in Java (che però non è un oggetto)

Python

Python definisce diversi tipi di collezioni di oggetti

- E.g., liste, insiemi, dizionari
- Una collezione può contenere oggetti di tipi eterogenei
- Le collezioni possono essere innestate

Le collezioni si possono distinguere in mutabili e immutabili

- Solo nelle collezioni mutabili è possibile aggiungere, rimuovere e sostituire elementi
- Gli oggetti visti finora (numeri, booleani, stringhe) sono immutabili

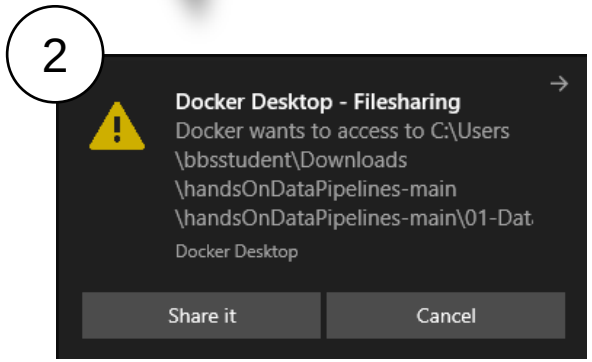
Le stringhe (str) sono trattabili come sequenze immutabili di caratteri

- Un carattere è una stringa lunga 1, non c'è un tipo di dato apposito

Python fornisce funzionalità comuni per accedere a collezioni

In action!

- Enter the folder `01-DataPreprocessing`
- Double click on `run.bat`
- Open the browser
- Copy and paste the link to the notebook
- Enter the notebook `00-PythonFundamentals`



Data manipulation

Un processo di analisi di dati prevede diversi passaggi:

- raccolta dei dati da una o più sorgenti (database, servizi web, etc.)
- comprensione della struttura e del significato dei dati
- trasformazione e pulizia dei dati in una forma utile alle fasi successive
- estrazione di conoscenza dai dati (statistiche, modelli predittivi, etc.)
- validazione e interpretazione della conoscenza estratta
- deployment della conoscenza in applicazioni

Data manipulation

I dati (relazionali) sono comunemente reperiti o convertiti in forma tabulare

- Ogni riga rappresenta un'osservazione o istanza
  - Uno degli oggetti su cui si sta compiendo l'analisi
  - E.g., un prodotto
- Ogni colonna è una variabile, attributo o feature che caratterizza ciascun oggetto
  - Tutti i valori di una colonna sono dello stesso tipo
  - E.g., nome del prodotto e prezzo

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

- Un prodotto è descritto da ID e prezzo, ma non ha senso calcolare l'ID medio dei prodotti
- Il tipo dell'attributo ci dice quali operatori ha senso applicare ai valori che esso assume
  - E.g., diversità, ordinamento, addittività, moltiplicatività

Data manipulation

Diversi tipi di attributo

- **(Categorico) Nominale:** possiamo solo distinguere i valori
- **(Categorico) Ordinale:** posso distinguere e ordinare i valori
- **(Numerico) Di intervallo:** posso distinguere, ordinare i valori e calcolare differenza
- **(Numerico) Di rapporto:** posso distinguere, ordinare i valori e calcolare differenza e rapporto

Prodotto	Fascia prezzo	Data vendita	Quantità
xxx	bassa	05/07/2021	10
xxy	media	05/07/2021	50
xyz	alta	02/06/2021	100

Pandas

pandas (Python) è una soluzione per la manipolazione di dati tabulari

- Introduce nuovi tipi di dati: **Series** (e.g., serie temporale) e **DataFrame** (e.g., tabelle)
- Supporto operazioni SQL-like (join/merge, aggregazione, etc.)
- Trattamento dati mancanti, riorganizzazione della loro forma (shape)
- Per convenzione il package pandas si importa con nome “pd”

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

Pandas

Una serie (Series) è una sequenza di valori dello stesso tipo

- Ad ogni valore è associata un'etichetta
- I tipi supportati, sia per i valori che per le etichette, sono quelli di NumPy (float64, int64, …)
- In pratica un vettore a una dimensione con un'etichetta associata ad ogni elemento

L'indice di una serie (index) è la sequenza delle etichette associate ai valori

- Le etichette sono spesso identificatori di tipo numerico o stringa
  - E.g., la chiave primaria di una tabella in un database
- Le etichette in un indice possono non essere univoche, ma nell'uso pratico spesso lo sono

Pandas

Il costruttore di Series accetta i valori della serie e come attributo index opzionale le etichette corrispondenti

- >>> ser = pd.Series([4 , 7 , -5 , 3],
  ... index=["d", "b", "a", "c"])
- Se non specificato, l'indice è la sequenza di interi da 0 a N-1, così che ogni elemento sia etichettato dalla posizione (come in liste e array)

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

Pandas

Le serie supportano anche operazioni binarie tra esse

- Con operatori +, -, *, etc.
- Con funzioni universali

L’operazione è applicata per elementi con pari etichetta

- Non viene considerate l’ordine dei valori
- Per ogni etichetta presente solo in un operando si avrà un valore NA nel risultato

Tipi di attributo

Quando si crea una serie è possibile specificare il tipo di dato

I tipi di dati utilizzati più comunemente sono quelli numerici

- I tipi np.floatN memorizzano numeri a virgola mobile
- I tipi np.intN / np.uintN memorizzano numeri interi con/senza segno
- N è il numero di bit usati, pari a 8, 16, 32 o 64

Altri tipi di dato includono

- bool: valori booleani
- datetime64, timedelta64: timestamp e intervalli di tempo
- object: generici oggetti Python, usato principalmente per stringhe

Perché è importante selezionare il giusto tipo di dato?

Valori mancanti

Nella pratica, dataset ha spesso dei valori mancanti

- E.g., perché non esistono o non sono stati forniti

Una serie può avere valori mancanti, detti NA (Not Available)

- In caso di numeri reali, NA è rappresentato internamente dal valore nan (Not a Number)
- Come in altri linguaggi, il valore nan non risulta mai uguale, maggiore o minore di altri numeri

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

- Qualsiasi espressione numerica con nan ha risultato nan

```
>>> 2 * np.nan – 1
nan
```

Quali problemi possono sorgere?

Valori mancanti

Alcuni metodi

- isna e notna verificano quali elementi (non) sono mancanti e restituiscono una serie booleana
- dropna rimuove i valori mancanti dalla serie
  - di default, viene creata una copia della serie
  - indicando inplace=True viene invece modificata la serie stessa

```
>>> 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

Valori mancanti

Il metodo fillna permette di rimpiazzare i valori NA

- anche qui viene creata una copia a meno che non si specifichi inplace=True
- Indicando un valore, tutti gli NA sono sostituiti con esso
  - è comune usare la media
- Usando invece il parametro method pari a ffill o bfill ogni NA è sostituito col valore non NA prima o dopo (se esiste!)
  - utile per serie temporali

(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

Funzioni aggregate

Le serie offrono metodi per calcolare statistiche aggregate sui valori con nomi e funzionamento pari a quelle degli ndarray

- sum (somma), mean (media), min (minimo), max (massimo), count (conta)
- Di default, eventuali valori mancanti vengono ignorati

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

Specificando skipna=False invece gli NA invalidano il calcolo

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

Rispetto a NumPy sono aggiunti i metodi idxmin e idxmax, che restituiscono l'etichetta del valore minimo o massimo

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

Distribuzione dei valori

Alcuni metodi

- unique restituisce un vettore con tutti i valori distinti in una serie, ordinati sulla prima apparizione
- nunique ne restituisce direttamente la quantità

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

Distribuzione dei valori

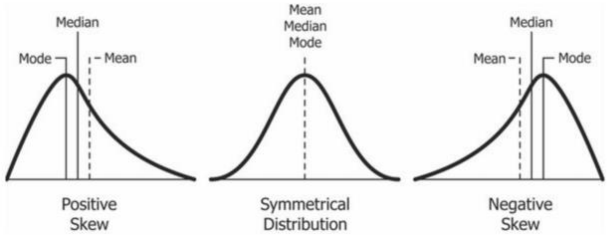
```
>>> x = pd.Series(
  [6, 2, 1, 2, 5, 1, 1])
```

Alcuni metodi

- value_counts restituisce una nuova serie che associa ad ogni valore distinto il suo numero di occorrenze, partendo dal più frequente

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

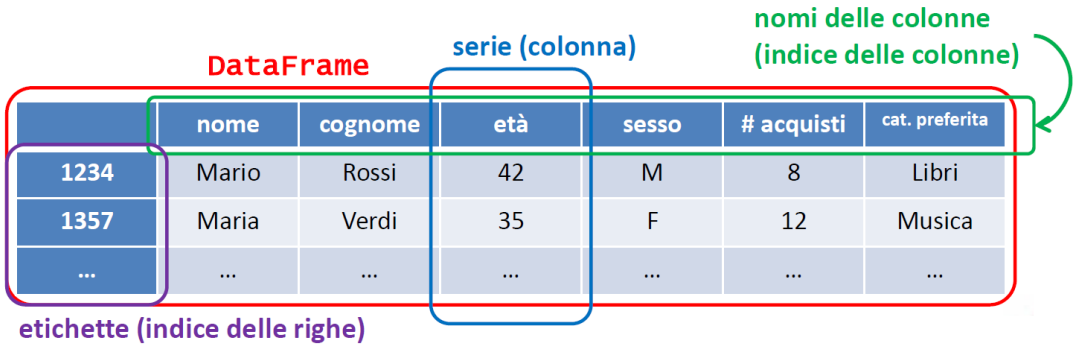
Quali problemi può causare una distribuzione “skewed”?



Dataframe

Un DataFrame rappresenta dati in forma relazionale

- Può essere visto come una sequenza di colonne rappresentate da serie di diverso tipo con etichette condivise
- Le etichette sono di solito identificatori univoci delle righe
- Ogni serie (colonna) ha un nome, utilizzabile come chiave per accedere ad essa



Dataframe

Una colonna di DataFrame può essere estratta in forma di serie usando il suo nome come indice

- Se il nome è un identificatore Python valido non usato da pandas, si può accedere alla colonna come se fosse un attributo

  >>> 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

pandas fornisce varie funzioni per caricare DataFrame da sorgenti esterne

- Tra queste read_csv consente di creare un DataFrame caricando i dati da un file CSV
- Va passato un oggetto file da cui leggere, oppure direttamente il nome di un file da aprire
- I dati letti sono convertiti automaticamente nei tipi appropriati (numeri interi, reali, etc.)

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

Dataframe

Il metodo read_csv ha molti parametri opzionali, ad es.:

- sep: separatore di colonna da usare (default “,”)
- names: nomi delle colonne (di default letti dalla prima riga)
- index_col: numero della colonna da usare come indice, passando una lista di numeri si ottiene un indice a più livelli
- dtype: tipo di dati delle colonne
  - Con dtype possiamo definire tipi di dati efficienti da usare
- nrows: massimo numero di righe da leggere
  - Con nrows si possono importare poche righe per verificare preventivamente i tipi di dati da usare

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., Sex: Male, Pregnant: Yes)
  - Missing values
  - Inconsistent data among multiple sources
  - More?

Data preprocessing

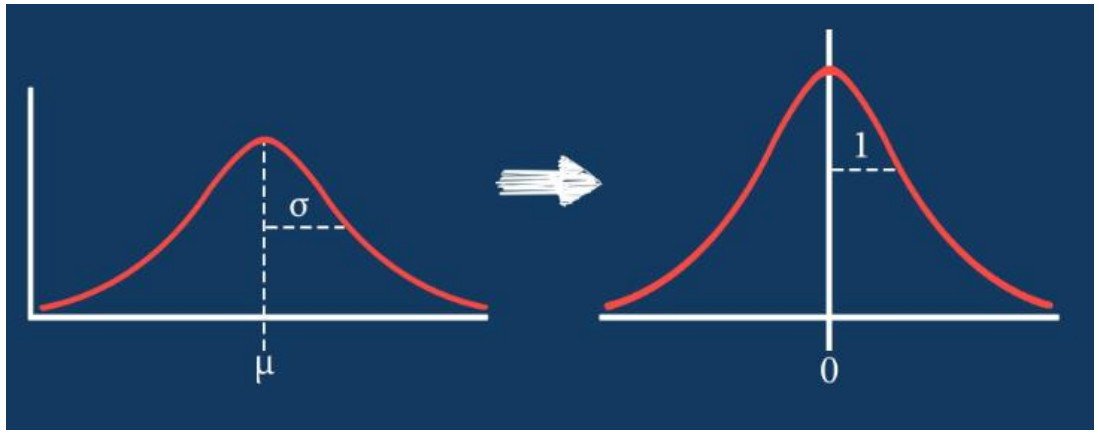
Which transformations can we apply?

- 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, get 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

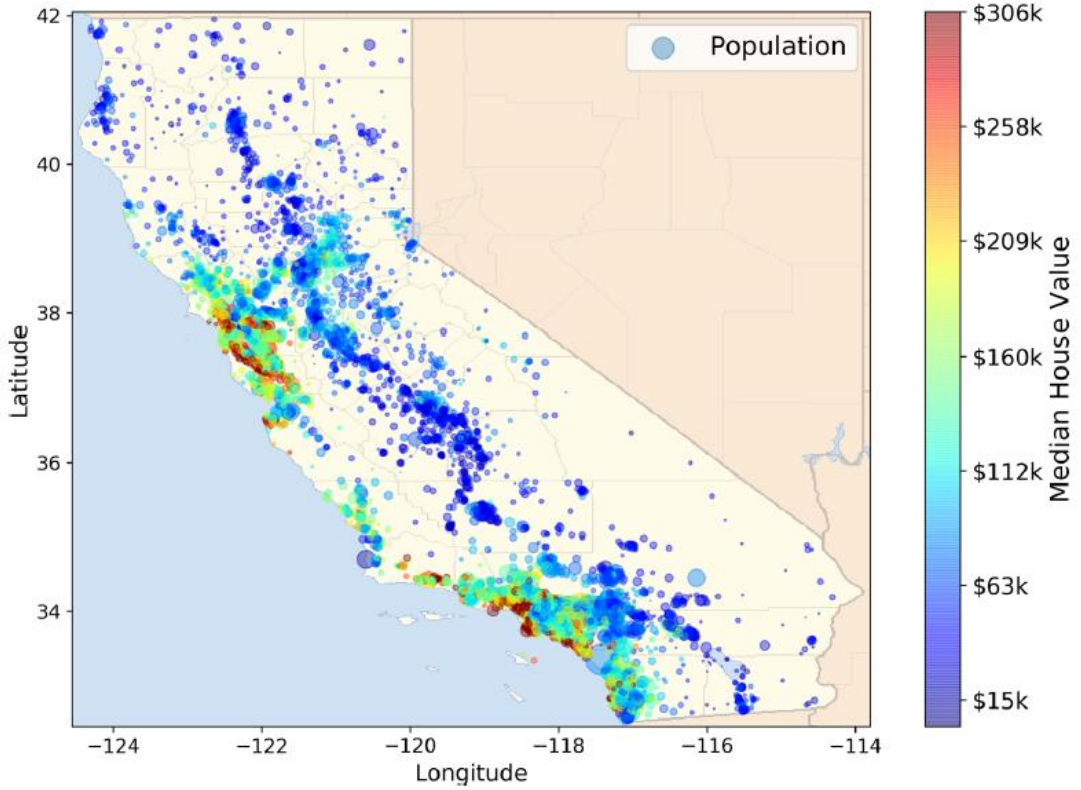
Then, ask (yourself) some questions:

- 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?

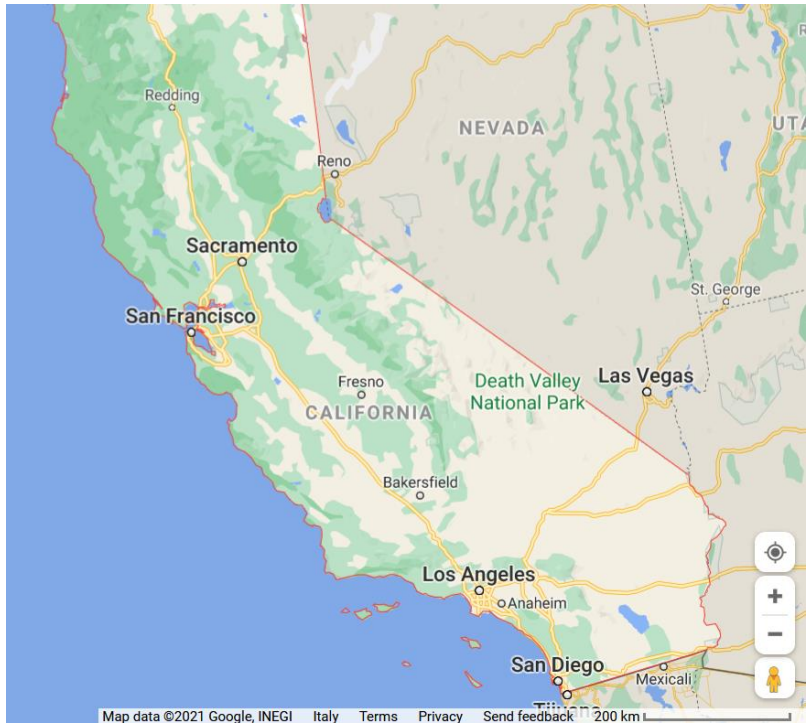
Integrated analytics lab

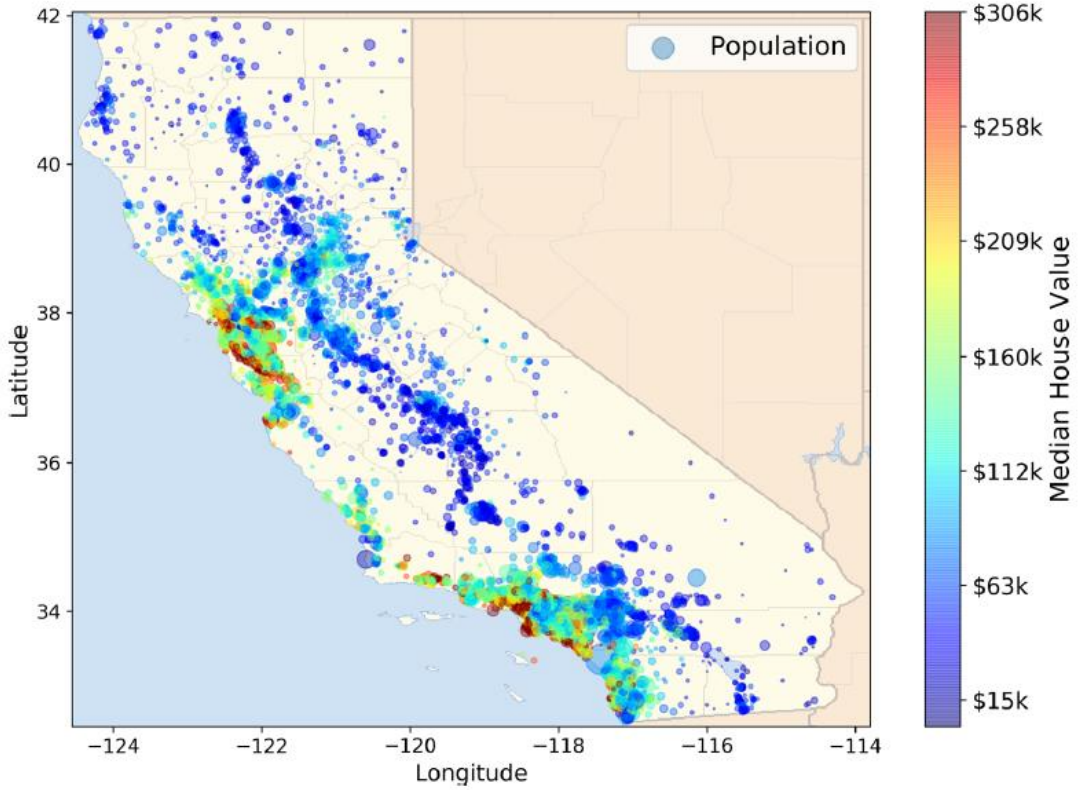
This checklist can help you while building your projects

- 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

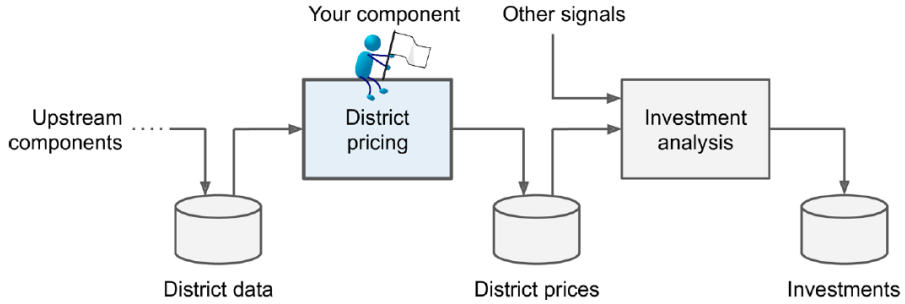




Integrated analytics lab

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."*



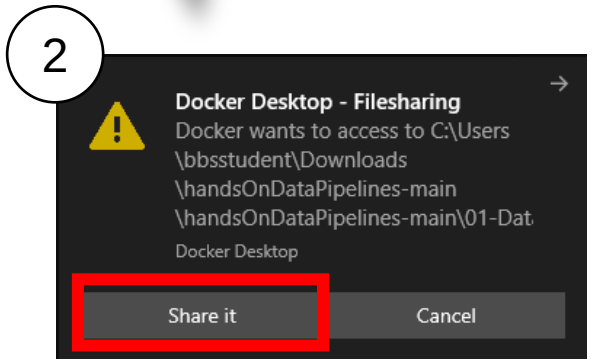
Integrated analytics lab

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 folder `01-DataPreprocessing`
- Double click on `run.bat`
- Open the browser
- Copy and paste the link to the notebook
- Enter the notebook `01-DataPreprocessing`



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