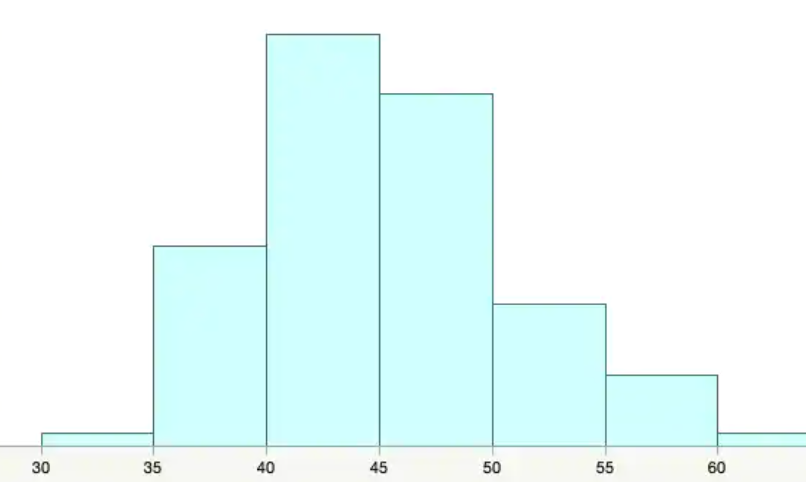
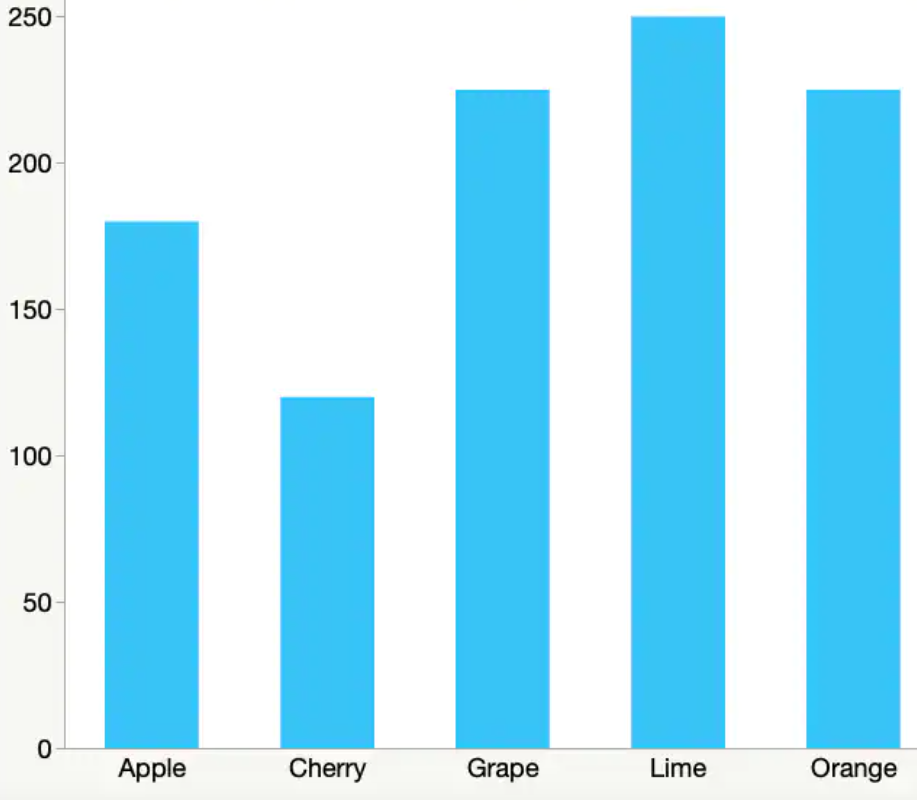
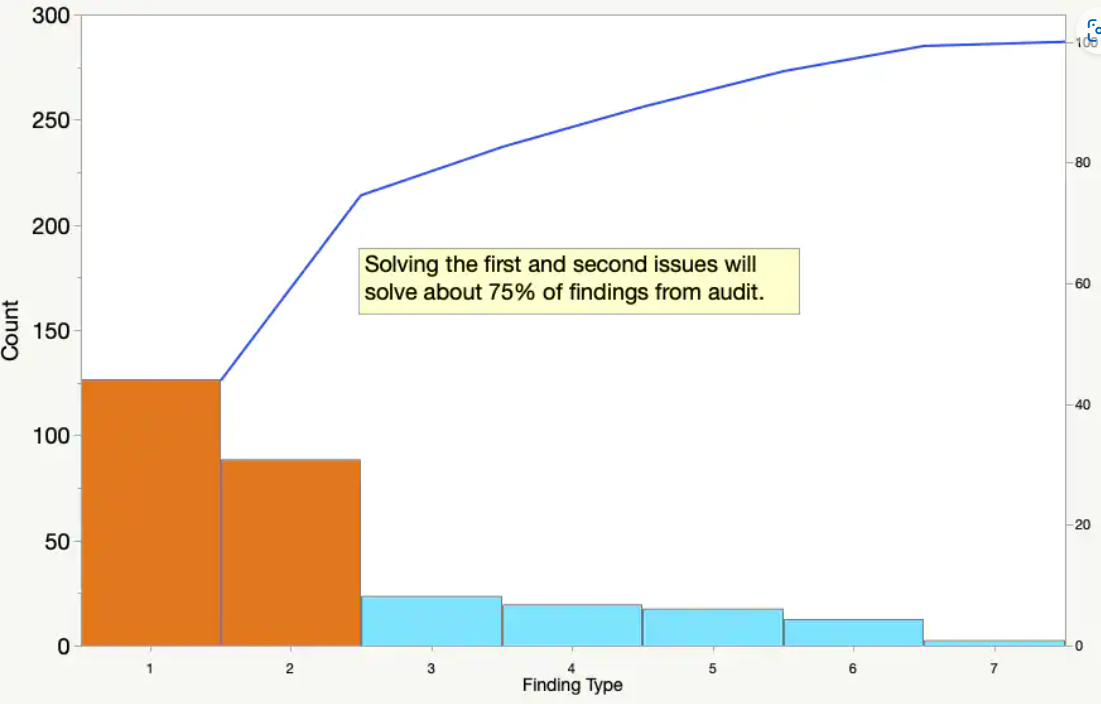
**Appunti Lab SAD – Vecchi Alex**

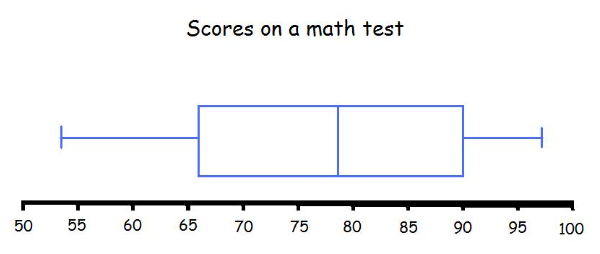
**01- Introduzione**

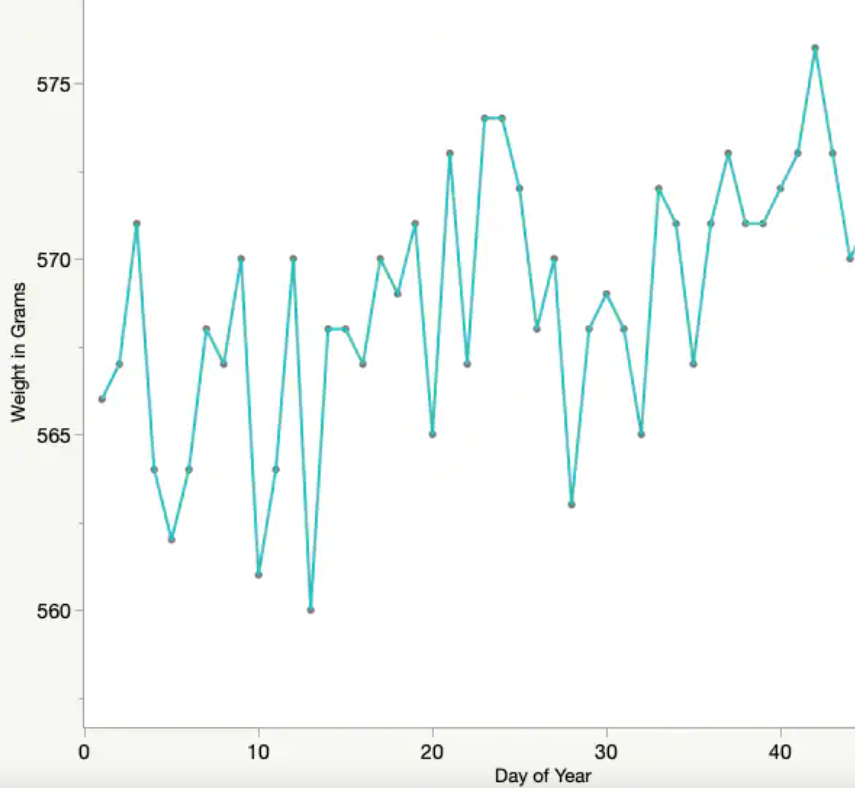
**Tipi di grafici (i più importanti, grafici a torta non spcificati)**

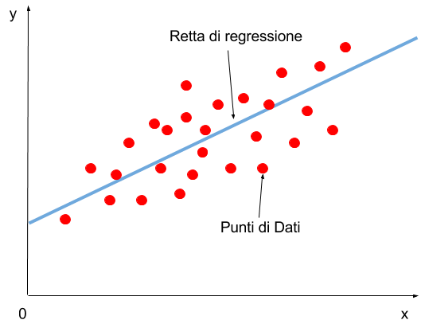
**Istiogrammi:** Mostra la forma o la distribuzione dei dati; è utile per l'identificazione degli outlier  


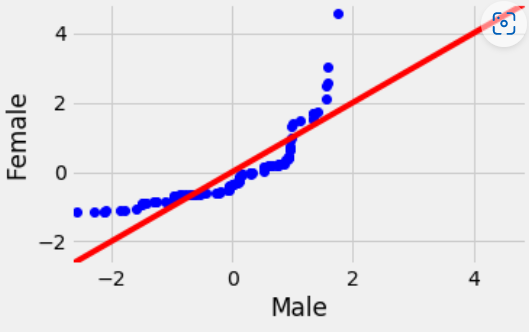
**Grafico a Barre:** Mostra il conteggio della frequenza dei valori di una variabile  


**Diagrammi di Pareto:** Mostra i conteggi di frequenza ordinati di una variabile spesso con unacurva della percentuale cumulativa  


**Box Plot:** Mostra la distribuzione dei dati. Le parti della scatola indicano il 25° percentile, il 50° percentile (percentile mediano) e il 75° percentile. A seconda dei dati, i baffi mostrano il minimo e il massimo, mentre gli outlier compaiono oltre i baffi. Viene usato per trovare gli errori nei dati e studiare una variabile  


**Grafici a linee:** Mostra i cambiamenti nel tempo. I valori sull'asse x devono essere ordinati cronologicamente. I grafici a linea, chiamati anche grafici lineari, sono utili per trovare outlier  


**Grafici dispersione (Scatter Plot):** Mostra un possibile rapporto tra due variabili, identificando gli outlier   


**Grafici QQPlot:** considera due campioni al fine di valutare la validità dell'ipotesi che i campioni stessi seguano una medesima distribuzione utilizzando i quantili, stessa distribuzione=stesso valore di quantile  


**Restituire il tipo di variabile**

type(first\_appearance)

**Operatori**



**Lista**

In python, una lista è una struttura dati eterogenea e ad accesso posizionale e dinamica: pertanto in essa viene memorizzata tipicamente una sequenza di elementi, che possono essere di tipo diverso e a cui è possibile accedere direttamente specificando la corrispondente posizione

Dichiarazione: iron\_man = ['Iron Man','Tony Stark', 'Long Island, New York', 'Marvel Comics']

Una lista può essere acceduta dalla testa e dalla coda e parte sempre da 0 **:** iron\_man[1] o iron\_man[-2]

Si può stampare una sottolista indicando il primo valore da inserire e il primo da escludere: iron\_man[4:6] o iron\_man[-4:-2] **o** names[-5:]per stampare gli ultimi 5 elementi

**Verificare se un elemento è nella lista**

'Thing' in listname

**Eliminare un elemento nella lista specificando la posizione, gli altri vengono shiftati**

del nomelista[0]

**Inserire un elemento nella lista specificando la posizione, gli altri vengono shiftati**

nomelista.insert(4, 'Aquaman')

**Numero di elementi nella lista**

len(nomelista)

**Ordinare una lista**

names.sort()

**Ordinare una lista in modo non crescente**

nomelista.sort(reverse=True)

**Definire una funzione con argomento x e che restituisce una espressione**

lambda x: <espressione> (es: successore = lambda n: n+1; successore(9))

**Ordinare una lista in base alla lunghezza con il parametro key**

nomelista.sort(key=lambda n:len(n))

//trasforma ogni elemento in un valore su cui basare l’ordinamento

**Tupla**

Una Tupla è una lista non modificabile e si dichiara con le parentesi tonde, gli altri operatori (sort, in, len..) rimangono invariati e si accede agli elementi come per le liste

Dichiarazione: rogue = ('Rogue','Anna Marie','Caldecott County, Mississippi','Marvel Comics')

**Eccezioni**

try:

rogue[-2] = 70

except TypeError:

print('Non si possono modificare gli elementi di una tupla')

// il blocco di istruzioni coinvolto è quello che segue la parola chiave try, e le istruzioni dopo except vengono eseguite solo se viene lanciata un eccezione del tipo specificato

**Stringhe**

Sono delle tuple di caratteri, non possono essere modificati ma è possibile fare le altre operazioni

name='ciao' o name=rogue[1]

**Dizionari**

I dizionari servono a memorizzare delle associazioni tra oggetti. È quindi possibile pensare a essi come a insiemi di coppie (chiave, valore), dove una data chiave non occorre più di una volta.

Dichiarazione: rogue = {'name': 'Rogue',

'identity': 'Anna Marie',

'birth\_place': 'Caldecott County, Mississippi',

'publisher': 'Marvel Comics', }

**Lettura da un dizionario di un elemento**

rogue['identity']

**Verificare se una chiave è presente nel dizionario**

key in dictionary

**Ciclo FOR**

for elemento in lista:

elabora(elemento) //indentazione obbligaoria

quindi elemento conterrà il primo elemento di lista durante la prima iterazione, il suo secondo elemento durante la seconda iterazione e così via

**If e Else**

if <condizione>:

<istruzione\_se\_condizione\_vera> //indentazione obbligatoria

else:

<istruzione\_se\_condizione\_falsa>

**Conteggiare frequenze assolute (numero di occorrenze) di un anno da una lista**

years = [1941, 1962, None, None]

counts = {} //crea un dizionario vuoto, ha le graffe

for y in years:

if y in counts:

counts[y] += 1

else:

counts[y] = 1

**Ordinare un dizionario per numero di occorrenze**

pairs = list(counts.items()) //converte il dizionario in una lista di coppie di elementi

sorted(pairs, key=lambda p:p[1], reverse=True) //ordina per il secondo elemnto

**Accedere al secondo elemento del primo elemento di una coppia**

pairs[0][1] //lista di coppie, accedo al secondo elemento della prima coppia

**Funzioni**

La definizione procede con un carattere di due punti e dal corpo della funzione le cui istruzioni devono essere indentate di un livello.

def get\_sorted\_counts(sequence):

counts = {}  
 for x in sequence:

if x in counts:

counts[x] += 1 //incrementa il valore nel dizionario

else:

counts[x] = 1 //inserisce il valore nel dizionario

pairs = counts.items()  
 return sorted(pairs, key=lambda p:p[1], reverse=True)

**Import moduli**

Un modulo è un file che contiene la definizione di una o più funzioni o classi

from collections import defaultdict  
import numpy as np

**Creare un dizionario vuoto con valore predefinito in base al tipo di dato**

defaultdict(<tipo>)

**Trovare l’indice delll’elemento massimo in una lista eliminando i None**

years = [y for y in years if y]  
np.argmax(years)

oppure

years2[np.argmax(years2)]

**Disegnare Grafici**

In generale le funzioni di matplotlib che generano un grafico basato su una serie di punti accettano come argomenti due liste contenenti rispettivamente le ascisse e le ordinate dei punti stessi

**Convertire una lista di coppie in due liste di coordinate per disegnare grafici**

np.array(get\_sorted\_counts(years)).transpose()

//np.array è una struttura dati base che prende in input una lista o una tupla. Transponse restituisce l’array trasposto creando di fatto due liste contenenti ascisse e ordinate

np.array(get\_sorted\_counts(years)[1:]).transpose() //scarto il primo elem

**Assegnare le liste a due variabili**

x, y = np.array(get\_sorted\_counts(years)[1:]).transpose()

**Disegnare un grafico a barre delle frequenze assolute**

%matplotlib inline *//magic line che dice a jupyter di visualizzare direttamente nel notebook*

import matplotlib.pyplot as plt

plt.rc('figure', figsize=(5.0, 2.0)) //misure del grafico  
plt.bar(x, y) //variabili  
plt.show() //visualizzazione

**Leggere dati dai file**

import csv

with open('data/heroes.csv', 'r') as heroes\_file: //apre il file  
heroes\_reader = csv.reader(heroes\_file, delimiter=';', quotechar='"')  
heroes = list(heroes\_reader)[1:] //crea la lista senza titoli del csv

**List comprehension**

[f(e) for e in l]

dove f(e) indica una funzione o un'espressione che dipende dalla variabile muta  e e l è una lista di cui quindi e indica il generico elemento. Questa espressione permette di costruire una nuova lista in cui il primo elemento è il risultato del calcolo di f sul primo elemento di l e così via. È inoltre possibile utilizzare la sintassi [f(e) for e in l if g(e)], che indica che nella creazione della nuova lista bisogna limitarsi a considerare gli elementi e della lista originale che rendono vera l'espressione g(e)

years = [int(h[7]) for h in heroes if h[7]]

assegna a years la lista che contiene l'anno di prima apparizione di ogni supereroe (che infatti occorre in ottava posizione), convertito da stringa a intero, ma senza considerare le stringhe vuote, da cui posso ricavare il grafico delle frequenze assolute

**Rimuovere dati fuori scala non visualizzandoli**

[y for y in years if y > 2020] //visualizza i dati maggiori di 2020

plt.bar(x, y)  
plt.xlim((1950, 2015)) //imposto un limite su x di visualizzazione  
plt.ylim((0, 18.5)) //imposto un limite su y di visualizzazione  
plt.show()

**02 - Pandas**

Libreria per facilitare quanto visto sopra

**Serie in Pandas**

Una delle classi principali implementate in pandas è Series. Le sue istanze rappresentano serie di osservazioni di un certo carattere fatto su un insieme di individui. La cella seguente recupera dalla lista heroes precedentemente creata i nomi dei supereoi e il loro anno di prima apparizione e li utilizza per creare una serie:

years = [int(h[7]) if h[7] else None for h in heroes]  
names = [h[0] for h in heroes]  
first\_appearance = pd.Series(years, index = names)

il primo argomento specificato nel costruttore è una lista (ma sarebbe andata bene anche una tupla) di anni che indicano la prima apparizione di un supereroe e il secondo rappresenta appunto l'indice, che in questo caso è la lista dei corrispondenti nomi. Quando si visualizza una serie, ogni osservazione viene associata al corrispondente elemento usando appunto l'indice

**Accesso ai dati di una serie**

(first\_appearance['Wonder Woman'], first\_appearance.loc['Wonder Woman'])

(first\_appearance[128], first\_appearance.iloc[128])

first\_appearance['Wonder Girl':'Wonder Woman'] //per accedere da WG a WW inclus

first\_appearance[60:63] //ultimo valore escluso

first\_appearance[-5:]

**Stampare i primi n elementi o gli ultimi n elementi**

first\_appearance.head(7)

**Accedere a 3 elementi distinti per posizione specificando una lista**

**first**\_appearance[[1, 42, 709]] //stampa l’elemento 1,42 e 709

**Accedere all’elemento se l’anno è compreso tra 1970 e 1974**

**first**\_appearance [[1970 <= y <1975 for y in first\_appearance]]

first\_appearance[(first\_appearance < 1975) & (first\_appearance >= 1970)]

**Accedere all’elemento se l’anno è maggiore di 2010 (vale anche per il minore)**

**first\_appearance[first\_appearance > 2010]**

**Calcolare le frequenze assolute del valore di una serie e stampare i primi 5**

**first\_appearance.value\_counts().head(5)**

**// i valori None sono esclusi dal coalcolo in automatico**

**Calcolare le frequenze assolute con valore <2090 e ordinandoli per indice crescente**

**first\_app\_freq = first\_appearance[first\_appearance < 2090].value\_counts().sort\_index()**

**Ordinare la serie per i valori**

**first\_app\_freq.sort\_values(ascending=False)**

**Visualizzazione di una serie**

Pandas mette a disposizione l'oggetto plot per visualizzare graficamente i contenuti di una serie, utilizzando matplotlib dietro le quinte; in particolare, il metodo bar visualizza un grafico a barre,  bar considera un punto per ogni elemento della serie. Per ottenere un grafico simile in cui le ascisse siano effettivamente gli anni di prima apparizione è necessario tornare a utilizzare esplicitamente matplotlib, passando al metodo bar rispettivamente l'indice e i valori della serie, che si ottengono rispettivamente utilizzando la proprietà index e invocando il metodo get\_values

plt.bar(first\_app\_freq.index, first\_app\_freq.values)

plt.xlim((1935, 2015))  
plt.ylim(0, 18.5)  
plt.show()

**Calcolare quanti supereroi sono apparsi dopo il 1960 avendo la serie delle frequenze ordinata**

sum(first\_app\_freq[1960:])

**Calcolare quanti supereroi sono apparsi tra il 1940 e il 1960 avendo la serie delle frequenze ordinata per indice**

sum(first\_app\_freq[1940:1960])

**Calcolare quanti supereroi sono apparsi prima del 1970 avendo la serie delle frequenze ordinata per indice**

sum(first\_app\_freq[:1970])

**Dividere tutti i valori per 100 di una serie**

height = pd.Series([float(h[4]) if h[4] else None for h in heroes], index=names) //creo la serie

(height/100)

**Applicare una funzione a tutti gli elementi della serie**

height.apply(lambda h: (h/100)\*\*2) //uso il metodo apply

**Creare una nuova serie effettuando una operazione tra due serie con stessi indici**

bmi = weight / height.apply(lambda h: (h/100)\*\*2) **Quello che succede quando si esegue un'operazione tra due serie e solo una di essa è definita in corrispondenza di uno specifico valore dell'indice, il risultato conterrà NaN per quel valore**

**Dataframe**

*Un dataframe* è una collezione di serie che hanno lo stesso indice, ed è quindi un insieme di osservazioni di vari *caratteri* per una popolazione di individui. Tra i vari modi che sono disponibili in pandas per creare un *dataframe*, noi faremo riferimento al metodo read\_csv della classe pd.DataFrame, che permette di leggere i contenuti di un file in formato CSV e convertirli automaticamente in un *dataframe*. l'insieme degli indici, dei caratteri e dei valori si ottengono, nell'ordine, alle proprietà index, columns e values

**Creare un dataframe da un csv**

heroes = pd.read\_csv('data/heroes.csv', sep=';', index\_col=0)

**Accedere ad un dataframe leggendo un attributo per tutti gli elementi**

heroes[‘IntestazioneColonna’]

heroes.Nomecolonna

**Accedere ad un dataframe con uno slicing per visualizzare un insieme di righe**

heroes['Agent 13':'Air-Walker']

heroes.iloc[42:46] //restituisce un dataframe

**Accedere ad un dataframe per leggere i dati di un oggetto (restituisce una serie)**

heroes.loc['Professor X'] //restituisce una serie

**Accedere ad un dataframe per leggere i dati di una riga e visualizzare solo alcune colonne**

heroes.loc['Professor X', 'Height':'Weight']

**Accedere ad un dataframe per leggere i dati di più righe e visualizzare solo alcune colonne**

heroes.iloc[[106, 103], [3, 4]]

**Accedere ad un dataframe per leggere esattamente un dato**

heroes.at['Superman', 'Strength']

heroes.iat[500, -1]

**Estrarre dal dataframe determinate colonne di colonne e creare un nuovo dataframe**

features = ['Height', 'Weight', 'Gender', 'First appearance',

'Hair color', 'Eye color', 'Strength', 'Intelligence'] //nomicolonne

X = all\_guys[features] //nuovo dataframe filtrato sulle colonne

**Ordinare un dataframe per un valore**

heroes.sort\_values(by='Weight', ascending=False)

**Ordinare un dataframe per indice**

heroes.sort\_index()

**Selezionare gli eroi per cui l'anno di apparizione esiste e rappresenta un valore non fuori scala**

heroes\_with\_year = heroes[heroes['First appearance'] > 1900]

**Unire due dataframe con gli stessi argomenti**

all\_guys = pd.concat([good\_guys, bad\_guys])

**03 – Dati e frequenze**

Vedremo come esistano tipi differenti di dati, e come in funzione del loro tipo esistano diversi strumenti grafici che li descrivono. Studieremo inoltre in modo più approfondito e diversificato il concetto di *frequenza*. I dati di dividono in qualitativi e quantitativi (discreti o continui)

**Costruire la tabella delle frequenze assolute usando un dataframe**

publisher\_freq = pd.crosstab(index=heroes\_with\_year['Publisher'],

columns=['Abs. frequence'], //nome seconda colonna

colnames=['']) //nome prima colonna

**Costruire la tabella delle frequenze relative usando un dataframe**

publisher\_abs\_freq = pd.crosstab(index=heroes\_with\_year['Publisher'],

columns=['Rel. frequence'],

colnames=[''])   
//creo la tabella con le freq. Assolute con l’intestazione per le relative

publisher\_rel\_freq = publisher\_abs\_freq / publisher\_abs\_freq.sum()   
//divido per n

Oppure usando normalize

**publisher\_rel\_freq = pd.crosstab(index=heroes\_with\_year['Publisher'],**

**columns=['Rel. frequence'],**

**colnames=[''],**

**normalize=True)**

**Arrotondare i valori della tabella delle frequenze relative**

publisher\_rel\_freq.apply(lambda p: 100 \* np.round(p, 3))

**Arrotondare i valori della tabella delle frequenze relative, convertirli in stringhe e visualizzare %**

(publisher\_rel\_freq.apply(lambda p: np.round(100\*p, 2))

.astype(str)

.apply(lambda s: s + '%'))

**Visualizzare il grafico a barre delle frequenze assolute ma senza legenda**

publisher\_freq.plot.bar(legend=False)  
plt.show()

Volendo visualizzare le barre in un ordine differente è sufficiente riordinare il *dataframe* nello stesso già visto per le tabelle delle frequenze, prima di invocare plt.plot

publisher\_order = ['Hanna-Barbera', 'ABC Studios', 'Dark Horse Comics',

'Image Comics', 'Marvel Comics', 'DC Comics',

'George Lucas', 'Rebellion',

'Star Trek', 'Universal Studios']

publisher\_rel\_freq.loc[publisher\_order,:].plot.bar(legend=False)  
plt.show()

**Creare tabella della forza dei soli uomini**

male\_strength\_freq = pd.crosstab(index=heroes.loc[heroes['Gender']=='M','Strength'],

columns='Abs. freq.')

**Contare i valori**

num\_male = sum(male\_strength\_freq['Abs. freq.'])

**Confrontare grafici a linee delle frequenze relative colorandoli diversamente**

male\_strength\_freq = (pd.crosstab(index=heroes.loc[heroes['Gender']=='M',

'Strength'],

columns='Rel. freq.',

normalize=True)

.loc[:, 'Rel. freq.']) //per estrarre la serie

female\_strength\_freq = (pd.crosstab(index=heroes.loc[heroes['Gender']=='F',

'Strength'],

columns='Rel. freq.',

normalize=True)

.loc[:, 'Rel. freq.']) //per estrarre la serie

male\_strength\_freq.plot(marker='o', color='blue', legend=False)  
female\_strength\_freq.plot(marker='o', color='pink', legend=False)  
plt.show()

**Come sopra ma usando un grafico sovrapposto a barre con trasparenza**

male\_strength\_freq.plot.bar(color='blue', alpha=.7)  
female\_strength\_freq.plot.bar(color='pink', alpha=.7)  
plt.show()

**Visualizzare un grafico a torta dei generi (dato qualitativo)**

gender\_freq = pd.crosstab(index=heroes\_with\_year['Gender'],

columns=['Abs. frequence'],

colnames=[''])

gender\_freq.plot.pie(y='Abs. frequence', colors=['pink', 'blue'])  
plt.show()

Oppure

gender\_freq['Abs. frequence'].plot.pie(colors=['pink', 'blue'])

**Visualizzare un grafico a barre degli anni**

first\_app\_freq = heroes\_with\_year['First appearance'].value\_counts()  
plt.bar(first\_app\_freq.index, first\_app\_freq.values)

**Visualizzare un grafico a bastoncini**

first\_app\_freq = heroes\_with\_year['First appearance'].value\_counts()  
plt.vlines(first\_app\_freq.index, 0, first\_app\_freq.values)

oppure per visualizzare un puntino

plt.plot(first\_app\_freq.index, first\_app\_freq.values, 'o')

**Visualizzare un istiogramma per i pesi dei supereri (quantitiativo) visualizzandolo per intervalli di possibili valori osservabili**

heroes['Weight'].hist(bins=50)  
plt.show()

è possibile definire intervalli di ampiezze differenti come ampiezze pari a 20 per i pesi inferiori a 200 kg., pari a 50 per pesi compresi tra 200 e 500 kg., e pari a 100 per i valori rimanenti

heroes['Weight'].hist(bins=np.hstack((np.arange(0, 200, 20),

np.arange(200, 500, 50),

np.arange(500, 1000, 100))))

plt.show()  
Negli istogrammi non ci sono spazi tra le barre, le quali rappresentano il numero dei valori riscontrati all'interno di un range specificato sull'asse orizzontale.  Nei grafici a barre possono esserci degli spazi tra le barre. In questo caso, le barre rappresentano i valori misurati per ogni categoria.

**Qual è il più recente tra gli anni di apparizione di un supereroe? E il minore?**

(heroes\_with\_year['First appearance'].min(), heroes\_with\_year['First appearance'].max())

**Quanti supereroi hanno un anno di apparizione non superiore al 1970? Calcola le frequenze cumulate**

Per calcolare le frequenze cumulate, pandas mette a disposizione il metodo cumsum per gli oggetti di tipo serie e *dataframe*. Quando viene utilizzato sulla serie prodotta da value\_counts è però necessario riordinare le frequenze prodotte rispetto al loro indice e infine si può invocare il metodo. Risulta invece più comodo calcolare generare il *dataframe* corrispondente alla tabella delle frequenze, che risulta già ordinato nel modo corretto, e su questo invocare cumsum

first\_app\_freq\_cumulate = (pd.crosstab(index=heroes\_with\_year['First appearance'],

columns=['Cumulate freq.'],

colnames=[''])

.cumsum())

first\_app\_freq\_cumulate.at[1970.0, 'Cumulate freq.'] //accedo al dato di 1970 dove ho la freq. cumulata

**Quanti hanno un anno di apparizione successivo al 1980?**

first\_app\_freq\_cumulate.iat[-1, 0] - first\_app\_freq\_cumulate.at[1980.0, 'Cumulate freq.']   
//num totale di casi – quelle fino al 1980

**Rappresentare la funzione cumulativa empirica**

import statsmodels.api as sm

ecdf = sm.distributions.ECDF(heroes\_with\_year['First appearance'])  
x = np.arange(1980, 1991)  
y = ecdf(x)  
plt.step(x, y) //grafico a step per funzione costante a tratti  
plt.show()

**Generare un diagramma di Pareto**

Frequenze e frequenze cumulate di una variabile categorica possono essere considerate congiuntamente per generare un *diagramma di Pareto* nel modo seguente: ordinando i dati per frequenza decrescente, su uno stesso sistema di riferimento in cui l'asse delle ascisse fa riferimento ai valori della variabile si sovrappongono il diagramma a barre delle frequenze e una linea spezzata che collega i valori delle frequenze cumulate. In generale, un diagramma di Pareto permette di identificare gli elementi più rilevanti in termini di frequenze all'interno di un insieme di osservazioni, evidenziando simultaneamente il peso di ogni fattore, sia il loro peso cumulativo.

eye\_color = heroes['Eye color']  
eye\_color\_freq = eye\_color.value\_counts(normalize=True) //freq relative  
eye\_color\_freq[eye\_color\_freq>.02].cumsum().plot()

eye\_color\_freq[eye\_color\_freq>.02].plot.bar()  
plt.show()

Se la linea non si estende fino ad 1 è perhè abbiamo considerato dei sottoinsiemi stringenti, per sistemare possiamo fare così, apportiamo alle frequenze una correzione che fa sì che ora la loro somma sia esattamente uno

norm\_eye\_color\_freq = eye\_color\_freq[eye\_color\_freq>.02]/ sum(eye\_color\_freq[eye\_color\_freq>.02])  
norm\_eye\_color\_freq.cumsum().plot()  
norm\_eye\_color\_freq.plot.bar()  
plt.show()

**Implementiamo una funzione my\_pareto che permette di impostare la soglia sulle frequenze e rigeneriamo il grafico, stavolta considerando tutti i dati**

def my\_pareto(data, threshold=0.02, renormalize=False):

freq = data.value\_counts(normalize=True)

freq = freq[freq > threshold]

if renormalize:

freq = freq / sum(freq)

freq.cumsum().plot()

freq.plot.bar() //fine funzione

my\_pareto(heroes['Eye color'], threshold=0.015) //soglia configurabile

**Frequenze congiunte e marginali**

Spesso è utile analizzare un insieme di osservazioni prendendo in considerazione due caratteri al posto di uno, per esempio per vedere se i valori di tali caratteri tendano a essere più o meno collegati tra loro tramite una relazione. Il concetto di frequenza si specializza in questo caso andando a contare il numero di osservazioni in cui i due caratteri considerati assumono due determinati valori, ottenendo la cosiddetta *frequenza congiunta assoluta*. La funzione pd.crosstab può essere utilizzata anche per produrre la tabella delle frequenze congiunte: basta indicare le serie corrispondenti ai caratteri considerati come valori degli argomenti index e columns

int\_gender\_freq = pd.crosstab(index=heroes['Intelligence'], columns=heroes['Gender'])

**Riordinare una tabella delle frequenze secondo l’index**

int\_gender\_freq = int\_gender\_freq.reindex(['low', 'moderate','average', 'good', 'high'])

**Riordinare l’ordine delle colonne di una tabella delle frequenze**

int\_gender\_freq.loc[:,['M', 'F']]

**Visualizzare solo alcune righe della tabella delle frequenze**

int\_gender\_freq.loc['moderate':'good', :]

**Disegnare un grafico a barre per la tabella delle frequenze congiunte**

*raggruppando* le barre che fanno riferimento a uno stesso valore per uno dei caratteri, e *colorandole* in funzione del valore che queste assumono per l'altro carattere in gioco

int\_gender\_freq.plot.bar(color=['pink', 'blue'])

int\_gender\_freq.plot.bar(color=['pink', 'blue'], stacked=True) //per sovrapporre le barre dello stesso attributo

**Raggruppare valori numerici vicini in un range per rappresentarlo nel grafico, ad esempio per pesi e altezze**

pd.crosstab(index=pd.cut(heroes['Weight'],

bins=[30, 50, 80, 100, 200, 500, 1000]),

columns=[heroes['Gender']])

oppure

pd.crosstab(index=pd.cut(heroes['Weight'],

bins=[30, 50, 80, 100, 200, 500, 1000],

right=False), //inverte la chiusura del range(n,m]

columns=[heroes['Gender']])

**Creare una tabella delle frequenze congiunte assolute con anche le frequenze marginali**

è possibile specificare il valore True per l'argomento margins al fine di aggiungere una riga e una colonna che contengono i totali (calcolati rispettivamente sulle singole colonne e sulle singole righe). I valori ivi indicati prendono il nome di *frequenze marginali*, e corrispondono alle frequenze del carattere corrispondente

pd.crosstab(index=heroes['Intelligence'],

columns=heroes['Gender'], margins=True)

**Creare una tabella delle frequenze congiunte relative**

pd.crosstab(index=heroes['Intelligence'],

columns=heroes['Gender'],

margins=True,

normalize='all') //con normalize diventano relative

I parametri di normalize possono essere: 'index' si otterrà una tabella in cui i valori su ogni riga sommano a 1, columns viene generata una tabella in cui tutte le colonne sommano al valore unitario,

**Visualizzare un diagramma di dispersione (scatter plot) dal dataframe per peso e altezza**

I diagrammi di dispersione permettono di valutare visivamente se esistano delle relazioni che legano i due caratteri visualizzati. Per esempio nel grafico precedente si nota come tendenzialmente a un valore alto del peso corrisponda un valore alto per l'altezza e viceversa

heroes[heroes['Gender']=='M'].plot.scatter('Height', 'Weight')

**Visualizzare uno scatter plot per peso-altezza visualizzando una retta**

from sklearn import linear\_model

regr = linear\_model.LinearRegression()

heroes\_with\_data = heroes[heroes['Gender']=='M'].copy().dropna() //fuoriscala li filtro sovrascrivendo heroes\_with\_data

X = heroes\_with\_data.loc[:, ['Height']]  
Y = heroes\_with\_data['Weight']

regr.fit(X, Y)  
heroes[heroes['Gender']=='M'].plot.scatter('Height', 'Weight')  
line, = plt.plot([0, 1000], regr.predict([[0], [1000]]), color='black') //questi numeri si possono modificare per cambiare la pendenza della retta

line.set\_dashes([3, 2])  
line.set\_linewidth(2)  
plt.show()

**Calcolare covarianza**

male\_heroes = heroes\_with\_data[heroes\_with\_data['Gender']=='M']

male\_heroes['Height'].cov(male\_heroes['Weight'])

**Calcolare indice correlazione**

male\_heroes['Height'].corr(male\_heroes['Weight'])

**04 – Indici di dispersione**

Gli oggetti di tipo serie messi a disposizione da pandas permettono di calcolare facilmente i principali indici di dispersione. Gli indici di dispersione di un oggetto di tipo serie si calcolano invocando su quest'ultimo degli appositi metodi: nomedataframe.funzione()

var restiuisce la varianza campionaria

std restituisce la deviazione standard campionaria

describe calcola i principali indici descrittivi di centralità e dispersione (quelli relativi a media, deviazione standard, minimo, primo quartile, mediana, terzo quartile e massimo), unitamente al numero di osservazioni nella serie, che utilizza per popolare un nuovo oggetto di tipo serie

quantile restituisce il quantile corrispondente al livello specificato come argomento (es, .15)

**Visualizzare un BoxPlot dei dati**

Un box plot (o box and whiskers plot, o ancora un diagramma a scatola) è una rappresentazione grafica che riassume le principali caratteristiche di un campione di dati. Tale rappresentazione contiene due componenti principali:

* una scatola, intesa come un rettangolo che evidenzia il primo e il terzo quartile campionario dei dati, che corrispondono alle due basi, e la mediana, indicata tramite un segmento parallelo alle basi stesse;
* due baffi, che si estendono dagli estremi della scatola fino a raggiungere il minimo e il massimo valore osservato.

Si verifica visualmente come questo grafico metta in evidenza: la centralità delle osservazioni, tramite il segmento che individua la mediana campionaria e la loro dispersione, sia in termini di range interquartile (l'altezza della scatola) e di intervallo di variazione dei dati (la distanza tra gli estremi dei baffi).

year.plot.box(whis='range') //whis opzionale, senza evidenzia degli outlier

year.plot.box(vert=False, whis='range') //per visualizzarlo in orizzontale

**Visualizzare un diagramma QQ (quantile quantile)**

Un *diagramma Q-Q* (o *diagramma quantile-quantile*) è una rappresentazione grafica che considera due campioni al fine di valutare la validità dell'ipotesi che i campioni stessi seguano una medesima distribuzione. Questi diagrammi si basano sul fatto che i quantili campionari rappresentano l'approssimazione di quantili teorici che, considerati tutti insieme, individuano univocamente la distribuzione dei dati. Pertanto, se due campioni hanno un'uguale distribuzione, allora estraendo da entrambi il quantile di un livello fissato si dovranno ottenere due numeri molto vicini, stampando ad esempio

(marvel\_sample.quantile(.2), dc\_sample.quantile(.2))

**Creare una lista di lunghezza definita di valori casuali**

marvel\_sample = marvel['Height'].sample(120) //.sample lo fa

**Disegnare il QQPlot visualizzando la bisettrice**

import statsmodels.api as sm

sm.qqplot\_2samples(marvel\_sample, dc\_sample, line='45') //passo i campioni  
plt.show()

non è detto che due campioni seguano necessariamente una medesima distribuzione. In tal caso, i punti ottenuti non si disporranno vicino alla bisettrice, vicevera se invece seguono la stessa distribuzione. Si nota infine che una standardizzazione dei dati permette di confinare il grafico ottenuto in prossimità dell'origine. In tal modo diventa più facile accorgersi di eventuali valori fuori scala.

sm.qqplot\_2samples((marvel\_sample-marvel\_sample.mean())/marvel\_sample.std(),

(dc\_sample-dc\_sample.mean())/dc\_sample.std(),

line='s',

xlabel='DC', ylabel='Marvel')

plt.show() //mean calcola la media aritmetica

**Simmetria e distribuzioni**

Alcuni dei grafici finora visti possono essere utili per mettere in evidenza una proprietà interessante di un campione di dati legata alla simmetria delle corrispondenti frequenze. Quando le frequenze, visualizzate a seconda dei casi tramite un grafico a barre o un istogramma, tendono a distribuirsi in modo simmetrico rispetto al valore della media campionaria si dice che il campione segue una distribuzione approssimativamente simmetrica.  La simmetria è visibile in entrambe le rappresentazioni: l'istogramma è approssimativamente simmetrico rispetto alla sua parte centrale e nel box plot i baffi hanno all'incirca la stessa lunghezza, così come la mediana si posiziona verso il centro della scatola. L'asimmetria in una distribuzione si può invece presentare in due diverse modalità: tende a essere presente una "coda" nella parte destra della distribuzione delle frequenze, evidenziata da valori più bassi delle frequenze nella parte destra dell'istogramma e da un baffo destro sensibilmente più lungo nel box plot, come esemplificato nella coppia di diagrammi qui sotto; in questo caso si parla quindi di distribuzione asimmetrica a destra (utilizzando a volte la terminologia inglese e dicendo che è presente uno skew a destra);  è possibile che la coda della distribuzione sia a sinistra, come nel grafico sottostante, e quindi si parla di asimmetria (o *skew*) a sinistra.

Tra le distribuzioni approssimativamente simmetriche, un ruolo particolare spetta alle cosiddette distribuzioni approssimativamente normali, in cui la simmetria è accompagnata da una forma "a campana" del grafico delle frequenze. In questo tipo di distribuzioni i dati si concentrano attorno alla media campionaria secondo la seguente regola empirica:

* approssimativamente il 68% delle osservazioni dista dalla media campionaria non più di una deviazione standard campionaria;
* approssimativamente il 95% delle osservazioni dista dalla media campionaria non più di due deviazioni standard campionarie;
* approssimativamente il 99.7% delle osservazioni dista dalla media campionaria non più di tre deviazioni standard campionarie.

**Verificare la regola empirica**

sample = heroes[(heroes['Height'].between(150, 220))]['Height'] //viaoutlier

def check\_empirical\_rule(n):

within = len(sample[np.abs(sample - sample.mean()) < n\*sample.std()])

return within / len(sample)

pd.DataFrame([check\_empirical\_rule(n) for n in range(1, 4)], columns=['%'],

index=range(1, 4))

// Se il risultato viene vicino a quelle percentuali allora le altezze sono distribuite in modo approssimativamente normale

**05 – Indici di eterogeneità**

Nel caso di variabili qualitative nominali la varianza e gli altri indici da essa derivati non si possono calcolare (infatti non sono calcolabili la media né la mediana né altri valori numerici di riferimento dai quali calcolare le distanze). È comunque necessario avere un indice che misuri la dispersione della distribuzione delle frequenze, detta eterogeneità. In particolare diremo che una variabile si distribuisce in modo eterogeneo se ogni suo valore si presenta con la stessa frequenza.

**Calcolare l’indice di Gini per una serie**

Per calcolare il valore dell'indice di Gini per i dati contenuti in una serie è necessario calcolare le corrispondenti frequenze relative tramite value\_counts; per elevare queste ultime al quadrato risulta utile invocare sulla serie delle frequenze il metodo map e poi sommare i valori ottenuti e sottrarre da 1 il risultato.

def gini(series):

return 1 - (sum(series.value\_counts(normalize=True)

.map(lambda f: f\*\*2)))

**Calcolare l’indice di gini usando la funzione sopra definita per vari valori**

import pandas as pd

heroes = pd.read\_csv('data/heroes.csv', sep=';', index\_col=0)  
publisher = heroes[heroes['Publisher'].isin(['Marvel Comics', 'DC Comics'])]['Publisher']

eye\_color = heroes[pd.notnull(heroes['Eye color'])]['Eye color']  
hair\_color = heroes[pd.notnull(heroes['Hair color'])]['Hair color']  
print(gini(publisher))  
print(gini(eye\_color))  
print(gini(hair\_color))  
print(gini(heroes.index)) /più vicino ad 1 e più è eterogeneo

**Calcolare la variante normalizzata dell’indice di Gini**

def generalized\_gini(series):

freq = series.value\_counts(normalize=True)  
s = len(freq)  
return (1 - (sum(freq.map(lambda f: f\*\*2)))) \* s / (s-1)

print(generalized\_gini(publisher))  
print(generalized\_gini(eye\_color))  
print(generalized\_gini(hair\_color))  
print(generalized\_gini(heroes.index))

**Calcolare l’entropia H per due valori**

def entropy\_2\_val(f):

return 0 if f in (0, 1) else - f \* np.log2(f) - (1-f) \* np.log2(1-f)

x = np.arange(0, 1.01, .01)  
y = list(map(entropy\_2\_val, x))  
plt.plot(x, y)  
plt.ylim((0, 1.1))  
plt.show()

**Calcolare l’entropia H per una serie**

def entropy(series):

return sum((series.value\_counts(normalize=True)

.map(lambda f: -f \* np.log2(f))))

print(entropy(publisher))  
print(entropy(eye\_color))  
print(entropy(hair\_color))  
print(entropy(heroes.index))

**Calcolare l’entropia H normalizzata per una serie**

def entropy(series, normalized=False):

freq = series.value\_counts(normalize=True)

e = sum((freq.map(lambda f: -f \* np.log2(f))))

if normalized:

e /= np.log2(len(freq))

return e

print(entropy(publisher, normalized=True))  
print(entropy(eye\_color, normalized=True))  
print(entropy(hair\_color, normalized=True))  
print(entropy(heroes.index, normalized=True))

**Alberi di decisione**

Gli indici di eterogeneità sono alla base della costruzione di un interessante classificatore chiamato *albero di decisione*. Un albero di decisione assegna *oggetti* a *classi*, dove un oggetto è descritto tramite un'osservazione che consiste, al solito, in un vettore di valori per degli attributi prefissati. Il procedimento di classificazione procede nel modo seguente: si considera la *radice* dell'albero, cioè l'unico nodo a cui non arrivano frecce (disposto di solito in alto nella rappresentazione dell'albero), e che è contrassegnato da una condizione che coinvolge i valori di uno o più attributi per l'oggetto che si vuole classificare. A seconda del valore di questa condizione, si percorre una delle due frecce partenti dalla radice. Se il nodo a cui si arriva è un nodo terminale (una *foglia* da cui non dipartono nuove frecce), in tale nodo è indicata la classe assegnata all'oggetto, altrimenti il nodo riporta un'altra condizione da valutare, iterando il comportamento precedente fino a che non si raggiunge una foglia e quindi si determina una classe per l'oggetto.

**Definire un albero di decisione (ricerca) per supereroi e supercattivi**

La costruzione di un albero di ricerca richiede innanzitutto di individuare un indice di eterogeneità. Facciamo un esempio selezioniamo l'indice di Gini e lavorando solo con due classi: i superori e i supercattivi, che indicheremo rispettivamente con le etichette good\_guy e bad\_guy. Il fatto di avere solo due etichette possibili semplifica notevolmente il calcolo dell'indice: basterà infatti specificare la frequenza relativa 𝑓 di good\_guy, così che la frequenza relativa di bad\_guy sarà 1−𝑓 e l'indice di Gini assumerà il valore 1−𝑓^2−(1−𝑓)^2: potremo quindi riutilizzare la funzione gini\_2\_val precedentemente definita. Attenzione ai valori mancanti

n corrispondenza di ogni osservazione è necessario indicare la relativa etichetta. Per comodità è meglio organizzare queste etichette in un *dataframe* separato che si può costruire sapendo ad esempio di avere 4 supereroi e il resto supercattivi nei rispettivi dataframe divisi (good\_guys e bad\_guys):

Y = pd.concat([pd.DataFrame(['good guy'] \* len(good\_guys),

index=good\_guys.index),

pd.DataFrame(['bad guy'] \* len(bad\_guys),

index=bad\_guys.index)])

Ora abbiamo una suddivisione delle etichette, basta ora decidere una query per il quale smistare gli elementi in modo da avere un ragionevole comportamento, ad esempio scegliendo un valore di streght di 40 come soglia per la divisione possiamo calcolare l’indice di Gini per i valori <= e > di 40:

freq = Y[X['Strength'] <= 40][0].value\_counts(normalize=True)  
freq\_bad = freq['bad guy']  
gini\_left = gini\_2\_val(freq\_bad)  
gini\_left //se viene omogeneità massima

Per combinare insieme i due indici (<= e > di 40) al fine di esprimere in un unico valore l'omogeneità media dei casi suddivisi nei sottogruppi si calcola una loro media, pesata in funzione della numerosità dei sottogruppi stessi.

weight\_left = len(Y[X['Strength'] <= 40]) / len(Y)  
weight\_right = len(Y[X['Strength'] > 40]) / len(Y)

gini\_left \* weight\_left + gini\_right \* weight\_right //gini right è gini quando >40

Tirando le somme, utilizzando la domanda: "la forza è minore o uguale a 40?" come criterio per la radice dell'albero si otterrebbe una suddivisione dei dati a disposizione in due gruppi con un'eterogeneità media pari a circa 0.17. Possiamo ora mantenere fisso l'attributo e considerare valori diversi per la soglia, al fine di trovare il valore che minimizza l'indice di Gini medio (e quindi corrisponde al caso di migliore omogeneità). Attrezziamoci per effettuare questo calcolo indipendentemente dalla scelta dell'attributo, del valore di soglia e dell'indice di eterogeneità

def split\_value(attribute, value, index):

freq = (Y[X[attribute] <= value])[0].value\_counts(normalize=True)  
freq\_bad = freq['bad guy']  
index\_left = index(freq\_bad)  
weight\_left = len(Y[X[attribute] <= value]) / len(Y)  
freq = (Y[X[attribute] > value])[0].value\_counts(normalize=True)  
freq\_bad = freq['bad guy']  
index\_right = index(freq\_bad)  
weight\_right = len(Y[X[attribute] > value]) / len(Y)  
return index\_left \* weight\_left + index\_right \* weight\_right

Ora siamo però in grado di effettuare in modo automatico lo stesso calcolo al variare dei possibili valori per la soglia.

x\_vals = range(10, 90) //definisco un range di soglie per strenght

plt.plot(x\_vals,

list(map(lambda v: split\_value('Strength', v, gini\_2\_val),

x\_vals)))

plt.show()

Per proseguire con la creazione dell'albero di decisione bisognerebbe applicare nuovamente il processo di ottimizzazione al gruppo che non ha omogeneità massima. In realtà esistono già delle implementazioni che si occupano di costruire gli alberi e useremo sklearn. Nonostante il calcolo degli indici di eterogeneità si possa effettuare anche su attributi a valori categorici, questa libreria richiede che i dati siano espressi utilizzando esclusivamente valori numerici. È dunque necessario convertire in numeri tutte le etichette categoriche. Questo procedimento può essere fatto automaticamente utilizzando gli oggetti della classe LabelEncoder, che una volta creati generano automaticamente il mapping tra etichette e valori numerici utilizzando il metodo fit a cui viene passato l'insieme delle osservazioni da convertire. Una volta determinato questo mapping, il metodo transform lo utilizza per convertire la serie corrispondente.

from sklearn.preprocessing import LabelEncoder

hair\_col\_encoder = LabelEncoder()  
hair\_col\_encoder.fit(all\_guys['Hair color'])  
all\_guys['Hair color'] = hair\_col\_encoder.transform(all\_guys['Hair color'])

Stampando all\_guys ora avremo valori numerici nelle varie colone convertite. Usando questo *dataframe* modificato è possibile utilizzare un oggetto della classe DecisionTreeClassifier per costruire l'albero di decisione, passando al metodo fit i *dataframe* che descrivono rispettivamente i supereroi e le loro etichette:

from sklearn import tree

clf = tree.DecisionTreeClassifier()  
clf = clf.fit(all\_guys, Y) //Y dataframe etichettato (vedi sopra)

Una volta costruito, sull'oggetto corrispondente all'albero è possibile invocare il metodo predict per verificare quale etichetta venga associata agli oggetti di partenza:

predictions = clf.predict([X.loc[name] for name in X.index])

**Visualizzare l’albero di decisione in forma grafica**

import graphviz //necessita installazione

graphviz.Source(tree.export\_graphviz(clf, out\_file=None,

class\_names=['bad guy', 'good guy'],

feature\_names=features))

pd.DataFrame(np.array((X.index, Y[0], predictions)))

**Funzione per trasformare un supereroe con valori numerici in alcuni campi**

def filter(obj):

transformed\_obj = obj['Height':'Intelligence']

transformed\_obj['Gender'] = gender\_encoder.transform([transformed\_obj['Gender']])[0]

transformed\_obj['Eye color'] = eye\_col\_encoder.transform([transformed\_obj['Eye color']])[0]

transformed\_obj['Hair color'] = hair\_col\_encoder.transform([transformed\_obj['Hair color']])[0]

transformed\_obj['Intelligence'] = intelligence\_encoder.transform([transformed\_obj['Intelligence']])[0]

return transformed\_obj

**Verificare per un supereroe non usato per definire l’albero a che etichetta viene associata**

clf.predict([filter(heroes.loc['Professor X'])])

In linea di principio gli alberi di decisione si possono costruire utilizzando un qualsiasi indice che valuti l'eterogeneità: la libreria sklearn supporta la scelta di questo criterio attraverso l'argomento opzionale criterion passato al costruttore di DecisionTreeClassifier. Il valore predefinito è 'gini', corrispondente ovviamente all'indice di Gini, ma è anche possibile specificare 'entropy' nel caso in cui si volesse utilizzare l'entropia. Ovviamente l'albero ottenuto usando l'entropia non è necessariamente uguale a quello che si ottiene con l'indice di Gini.

**Indici di concentrazione**

In presenza di variabili che rappresentano beni condivisibili in una popolazione, come per esempio la ricchezza, ci si può chiedere quanto la variabile sia equamente distribuita tra gli individui della popolazione, oppure quanto sia concentrata solo su un numero ridotto di osservazioni. Questo concetto è diverso dalla varianza, che misura la dispersione dei valori intorno a un valore medio. In generale allora ci può interessare valutare un indice di concentrazione, che valga 0 oppure 1 nei casi rispettivamente di concentrazione minima e massima, e che sia negli altri casi sia un valore crescente in funzione della concentrazione

**Analizzare indice di concentrazione della forza**

strength = heroes[pd.notnull(heroes['Strength'])]['Strength'] //filtro per righe valide e poi prendo solo la strenght

Q = strength.sort\_values().cumsum() / sum(strength) // Le quantità relative cumulate si ottengono ordinando i valori, calcolando le corrispondenti somme cumulate e dividendole per la "quantità" totale di forza

import numpy as np  
n = len(strength)  
F = np.arange(1, n+1) / n   
// Le frequenze relative cumulate sono facili da calcolare: il loro vettore ha come componenti i valori che vanno da 1/𝑛 a 1 , incrementandoli ogni volta di 1/𝑛 , dove 𝑛 indica il numero di valori osservati.

**Visualizzare il grafico di concentrazione della forza**

plt.fill\_between(F, F, Q, alpha=0.2)  
plt.plot([0, 1], [0, 1], linestyle='--', linewidth=2, c='gray')//usospezzate  
plt.plot(F, Q, linewidth=2, c='blue')  
plt.show()

**Calcolare l’indice di concentrazione di Gini**

Sottraendo componente per componente il vettore delle quantità cumulate a quello delle frequenze cumulate, sommando i risultati e moltiplicando per 2𝑛−12n−1 si ottiene il valore dell'indice di concentrazione.

2 \* sum(F - Q) / (n-1)

**06 – Trasformazione dei dati**

Può succedere di avere la necessità di trasformare i dati osservati per diverse ragioni: per poterli confrontare con altri riportandoli ad un intervallo predefinito, per poter confrontare la loro distribuzione di frequenza con quella di altri dati, oppure per renderli più facilmente leggibili.

**Copiare un dataset**

year = heroes['First appearance'].copy()

**Traslazione dei dati (y=a\*x+b)**

la media, la mediana e i quantili vengono traslati della stessa quantità 𝑘k (nel medesimo verso);  
il range, la distanza interquartile, la varianza e la deviazione standard dell'insieme traslato 𝑌, la frequenza rimangono invece gli stessi dell'insieme di partenza 𝑋. Ad esempio sottrarre tutti i dati per la loro media, basta applicare l’ operazione al dataset

strength = heroes['Strength'].copy()  
transformed\_strength = strength - strength.mean()

heroes\_copy['Strength']=heroes\_copy['Strength']- heroes\_copy['Strength'].mean()

**Cambiamento di scala (dilatazione o contrazione)**

la media, la mediana e i quantili vengono scalati della stessa quantità 1/h; il range di variazione e la distanza interquartile vengono scalati della stessa quantità 1/h; la varianza viene scalata di una quantità 1/(h^2) mentre la deviazione standard viene scalata di 1/h.  
Come nel caso delle traslazioni, anche per applicare un cambiamento di scala è sufficiente eseguire la corrispondente operazione aritmetica sulla serie coinvolta

transformed\_strength = strength / max(strength)

**Cambiamento di origine e scala**

Se abbiamo un insieme di valori nell'intervallo (𝑎,𝑏) e vogliamo adattarli in modo che appartengano all'intervallo (𝑐,𝑑), la trasformazione da applicare sarà: y= c+ [(d-c)/(b-a)]\*(x-a).  
Immaginiamo che risulti più pratico misurare la forza dei supereroi in una scala che va da −10−10 a 1010. La trasformazione relativa corrisponderà ai valori 𝑐=−10c=−10, 𝑑=10d=10 e rispettivamente al minimo e al massimo dei valori originali per quanto riguarda 𝑎a e 𝑏b.

strength = heroes['Strength'].copy()  
strength = strength[pd.notnull(strength)]  
old\_min = min(strength)   
old\_max = max(strength)  
new\_min = -10.0  
new\_max = 10.0  
transformed\_strength = new\_min + (new\_max - new\_min)/(old\_max - old\_min) \* (strength - old\_min)

Casi particolari:

* vogliamo trasportare i valori in [0,1][0,1]: in questo caso la trasformazione da applicare è y=(x-a)/(b-a)
* nel caso particolare 𝑎=0a=0, la trasformazione precedente consiste nel dividere i dati per il valore massimo 𝑏
* vogliamo trasportare i valori in [0,1][0,1]: in questo caso la trasformazione da applicare è y= 2\*[(x-a)/(b-a)]-1

**Standardizzazione**

La standardizzazione (o normalizzazione) è un caso particolare di cambiamento di origine e scala, e consiste nell'applicare una scala il cui fattore è uguale alla deviazione standard dei valori, per poi traslare verso sinistra rispetto alla media dei valori. Questa trasformazione dovrebbe avere l'effetto di rendere nulla la media campionaria dei dati trasformati.

transformed\_strength = (strength - strength.mean()) / strength.std()

**Trasformazioni logaritmiche**

A volte i valori di una variabile osservata sono molto grandi oppure molto distanziati. In questi casi può essere utile considerare non tanto il valore originale ma, pensando a tale valore come potenza di una data base, ragionare in termini del relativo esponente. Ciò corrisponde ad applicare una trasformazione logaritmica, La scelta della base del logaritmo in questa trasformazione viene di norma fatta a seconda delle situazioni che si affrontano: scelte comunemente utilizzate sono 10, la costante di Napier e≈2.71 oppure 2. La trasformazione applicata è y= log(x)

Nel caso i valori siano molto distanziati tra loro e caratterizzati da una distribuzione di frequenza unimodale fortemente asimmetrica, la trasformazione logaritmica permette di ottenere una distribuzione di frequenza più simmetrica. Questo tipo di trasformazione ha molti altri vantaggi, dovuti al fatto che l'operazione di prodotto (o quoziente) tra due valori viene trasformata nella somma (o nella differenza) dei rispettivi logaritmi.

pd.crosstab(index=np.log10(year),  
 columns=['Abs. freqence'],  
 colnames=['Transformed'])

Anche in questo caso l'iniettività della trasformazione assicura che le frequenze di dati originali e dati trasformati coincidono.

**07 – Analisi della varianza (ANOVA)**

Ipotizziamo di avere a disposizione delle osservazioni di un medesimo attributo divise in 𝐺 gruppi, per esempio perché si tratta del reddito di individui che vivono in diverse città, oppure di un valore ematico di pazienti sottoposti a diversi trattamenti clinici e così via. Formalmente, indichiamo rispettivamente con 𝑛1,…,𝑛𝐺 le numerosità dei vari gruppi, con 𝑛=𝑛1+⋯+𝑛𝐺 il numero totale di osservazioni e, fissato 𝑔∈{1,…,𝐺} e 𝑖∈{1,…,𝑛𝑔} , denotiamo con 𝑥𝑔𝑖 il valore dell' 𝑖 -esima osservazione nel gruppo 𝑔 .

Se si è interessati a valutare l'ipotesi che i valori delle medie nei vari gruppi siano sensibilmente differenti, per esempio perché si vuole dimostrare che il reddito non sia troppo diverso in un gruppo di città, oppure per dimostrare l'efficacia di un dato trattamento medico, è possibile applicare un metodo chiamato ANOVA (ANalysis Of VAriance). L'idea alla base di questo metodo è che se non vi sono sostanziali differenze tra i gruppi considerati, allora calcolare la varianza all'interno di un gruppo qualsiasi non dovrebbe portare a un risultato molto dissimile da quello ottenuto effettuando il calcolo su tutti i dati a disposizione.  
Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Possiamo utilizzare questa uguaglianza per validare o confutare l'ipotesi che le medie nei gruppi siano diverse: se la varianza totale e la varianza entro i gruppi assumono valori non troppo diversi tra loro (e dunque la varianza tra i gruppi risulta trascurabile), allora si confuta l'ipotesi; se al contrario la varianza tra i gruppi assume un valore elevato si può convalidare l'ipotesi

**Calcolare SST della forza di supereroi Marvel e supereroi DC**

import pandas as pd

heroes = pd.read\_csv('data/heroes.csv', sep=';', index\_col=0)  
marvel\_strength = heroes[(heroes['Publisher'] == 'Marvel Comics') &

(pd.notnull(heroes['Strength']))]['Strength']  
dc\_strength = heroes[(heroes['Publisher'] == 'DC Comics') & \

(pd.notnull(heroes['Strength']))]['Strength']

La cella precedente mostra come effettuare un'operazione di selezione complessa su di un \_dataframe\_, basandosi sull'operatore `&` di congiungzione logica. Analogamente, l'operatore `|` permette di calcolare disgiunzioni logiche.

all\_strength = pd.concat([marvel\_strength, dc\_strength])

SST = sum((all\_strength - all\_strength.mean())\*\*2)

**Calcolare SSW della forza di supereroi Marvel e supereroi DC**

sum\_within = sum((marvel\_strength - marvel\_strength.mean())\*\*2) + \ sum((dc\_strength - dc\_strength.mean())\*\*2)

**Calcolare SSB della forza di supereroi Marvel e supereroi DC**

sum\_between = len(marvel\_strength) \* (marvel\_strength.mean() - all\_strength.mean())\*\*2 + \ len(dc\_strength) \* (dc\_strength.mean() - all\_strength.mean())\*\*2

**Verificare che SST=SSW+SSB**

sum\_total - sum\_within - sum\_between //dovrebbe essere vicino lo zero

n = len(all\_strength)

total\_var = SST / (n-1) //varianza totale

within\_var = sum\_within / (n-2) //varianza entro i gruppi, ne ho 2

within\_var \* (n-2) / (n-1) //non so perchè questa moltiplicazione, se la tolgo viene simile

Se i valori sono molto vicini possiamo dire che i valori medi della forza siano sostanzialmente uguali, se nell’ordine delle decine è ok

**Definire una funzione che gestisca più di due gruppi e restituisca la varianza totale e la varianza entro i gruppi**

import numpy as np

def anova(groups):

all\_elements = pd.concat(groups)

sum\_total = sum((all\_elements - all\_elements.mean())\*\*2)

sum\_within = sum([sum((g - g.mean())\*\*2) for g in groups])

sum\_between = sum([len(g) \* (g.mean()-all\_elements.mean())\*\*2

for g in groups])

assert(np.abs(sum\_total - sum\_within - sum\_between) < 10\*\*-5)

n = len(all\_elements)

total\_var = sum\_total / (n-1)

within\_var = sum\_within / (n-len(groups))

return (total\_var, within\_var\*(n-len(groups))/(n-1))

Ad esempio confrontando i pesi

male\_weight = heroes[(heroes['Publisher'] == 'DC Comics') & \

(heroes['Gender'] == 'M') & \

(pd.notnull(heroes['Weight']))]['Weight']

female\_weight = heroes[(heroes['Publisher'] == 'DC Comics') & \

(heroes['Gender'] == 'F') & \

(pd.notnull(heroes['Weight']))]['Weight']

anova([male\_weight, female\_weight])

**08 – Analisi dei classificatori**

Immaginiamo di avere a disposizione un classificatore binario, costruito cioè per discriminare tra due classi che convenzionalmente indicheremo come positiva e negativa. Possiamo valutare la bontà di questo classificatore calcolando il numero di casi (o la corrispondente frazione) che vengono classificati in modo errato. Notiamo però che ci sono due possibili modi di sbagliare la classificazione:

* un esempio positivo viene classificato come negativo, dando luogo a un cosiddetto falso negativo;
* un esempio negativo viene classificato come positivo, e in questo caso si parla di falso positivo.

Tenuto anche conto del fatto che tipicamente è molto difficile riuscire a ottenere un buon classificatore in termine sia di falsi positivi, sia di falsi negativi, un modo efficace di valutare entrambi questi tipi di errore consiste nel disegnare la matrice di confusione (o tabella di confusione) che mostra il numero di falsi positivi e di falsi negativi unitamente al numero di casi correttamente classificati, a loro volta divisi in veri positivi e veri negativi.  
A partire dalla matrice di confusione è possibile derivare due indici che valutano separatamente la capacità del classificatore a lavorare correttamente con gli oggetti positivi e con quelli negativi: La sensibilità (VP/TP) e la specificità (VN/TN).  
Una volta calcolati i valori per questi due indici, è possibile valutare il classificatore in funzione della posizione assunta dal punto di coordinate (1−specificità,sensibilità) sul piano cartesiano

**Classificatori a soglia**

Un classificatore a soglia effettua il procedimento di classificazione di un generico oggetto calcolando una quantità e verificando poi che quest'ultima sia superiore a una soglia prefissata. La quantità varierà ovviamente in funzione dell'oggetto considerato, mentre la soglia resterà uguale. Per fissare la soglia possiamo usare gli indici di sensibilità e specificità: indicato con 𝜃 un generico valore per la soglia e individuato un intervallo [𝜃min,𝜃max], per ogni 𝜃∈𝐷 è poi possibile calcolare la sensibilità e la specificità del classificatore e disegnare sul piano cartesiano il punto corrispondente. Il risultato è una traiettoria che prende il nome di curva ROC che ha sempre l’origine e il punto (1,1) come estremi.  
Il valore di 𝜃 può quindi essere scelto in modo da trovare un giusto compromesso tra sensibilità e specificità. Il grafico della curva ROC viene inoltre utilizzato per valutare la bontà del classificatore indipendentemente da uno specifico valore della soglia. Ciò viene fatto misurando l'area compresa tra l'asse delle ascisse e la curva stessa chiamata AUC che più si avvicina ad 1 più il classificatore è ideale.  
I classificatori probabilistici sono casispeciali di classificatori a soglia che associano a un oggetto una stima della probabilità che questo appartenga alle varie classi

**Disegnare una ROC in funzione delle etichette che individuano le classi degli oggetti e delle probabilità emesse dal classificatore**

from sklearn import metrics

y = [1, 0, 1, 1, 0] //etichette corrette  
prob = [.7, .4, .8, .7, .3] //probabilità  
fpr, tpr, \_ = metrics.roc\_curve(y, prob) //primo array contenente le ascisse e il seocnod contentente le ordiante dei punti della ROC

plt.plot(fpr, tpr)  
plt.plot([0, 1], [0, 1], dashes=[3, 3], color='gray')  
plt.show()

**Calcolare la AUC sotto la ROC**

metrics.auc(fpr,tpr)

**Calcolare la ROC per l’editore dei supereroi**

import pandas as pd

//Focalizziamo poi l'attenzione su alcune delle colonne, trasformando i valori categorici in valori numerici.

heroes = pd.read\_csv('data/heroes.csv', sep=';', index\_col=0).dropna()  
heroes = heroes[heroes['Publisher'].isin(['Marvel Comics', 'DC Comics'])]  
features = ['Height', 'Weight', 'Gender', 'First appearance', 'Hair color', 'Eye color', 'Strength', 'Intelligence']

from sklearn.preprocessing import LabelEncoder

gender\_encoder = LabelEncoder()  
gender\_encoder.fit(heroes['Gender'])  
eye\_col\_encoder = LabelEncoder()  
eye\_col\_encoder.fit(heroes['Eye color'])  
hair\_col\_encoder = LabelEncoder()  
hair\_col\_encoder.fit(heroes['Hair color'])  
intelligence\_encoder = LabelEncoder()

\_ = intelligence\_encoder.fit(heroes['Intelligence'])

heroes['Gender'] = gender\_encoder.transform(heroes['Gender'])  
heroes['Eye color'] = eye\_col\_encoder.transform(heroes['Eye color'])  
heroes['Hair color'] = hair\_col\_encoder.transform(heroes['Hair color'])  
heroes['Intelligence'] = intelligence\_encoder.transform(heroes['Intelligence'])

Nell'idea di costruire un classificatore che associa i supereroi a una tra le due case editrici, estraiamo le colonne per ottenere l'insieme degli oggetti e parimenti costruiamo la serie delle etichette corrispondenti, i cui valori vanno trasformati da categorici a numerici

X = heroes[features]  
Y = heroes['Publisher']

publisher\_encoder = LabelEncoder()  
publisher\_encoder.fit(Y)  
Y = publisher\_encoder.transform(Y)

Il passo successivo consiste nel dividere gli oggetti a disposizione e le corrispondenti etichette in due gruppi: uno per costruire il classificatore e uno per valutarne l'efficacia in termini di curva ROC. Usando infatti dei dati nuovi per valuare un classificatore si cattura meglio la sua capacità di funzionare

num\_train = 90

X\_train = X[:num\_train]  
X\_test = X[num\_train:]  
Y\_train = Y[:num\_train]  
Y\_test = Y[num\_train:]

Utilizzando la classe DecisionTreeClassifier vista in precedenza è quindi possibile ottenere l'albero di decisione che ci interessa. Il risultato ottenuto può sia assegnare un oggetto a una specifica classe (cosa che abbiamo finora fatto invocando il metodo predict), sia stimare la probabilità che questo oggetto appartenga alle due classi considerate: ciò si ottiene invocando il metodo predict\_proba che restituirà in questo caso una lista con due valori che corrispondono rispettivamente alla probabilità di essere editi da Marvel o da DC. Considerando solo il primo di questi due valori si può facilmente procedere come prima e tracciare la corrispondente curva ROC.

from sklearn import tree

clf = tree.DecisionTreeClassifier()  
clf = clf.fit(X\_train, Y\_train)  
preds = clf.predict\_proba(X\_test)[:,1]  
fpr, tpr, \_ = metrics.roc\_curve(Y\_test, preds)  
plt.plot(fpr, tpr)  
plt.plot([0, 1], [0, 1], dashes=[3, 3], color='gray')  
plt.show()

auc = metrics.auc(fpr,tpr) //calcolo AUC  
print(auc)

pd.DataFrame(metrics.confusion\_matrix(Y\_test, preds)) //stampo matrice di confusione

**09 – Calcolo combinatorio**

**Generare tutte le possibili disposizioni con ripetizione degli elementi in una lista in 𝑘 posti**

l = range(3)  
disp\_rep = [(l1, l2) for l1 in l for l2 in l]

oppure

import itertools  
list(itertools.product(range(3), repeat=2)) //lista valori e num posti

**Generare tutte le possibili disposizioni senza ripetizione degli elementi in una lista in 𝑘 posti**

l = range(3)  
disp\_rep = [(l1, l2) for l1 in l for l2 in l]  
disp\_norep = [d for d in disp\_rep if len(d)==len(set(d))]

oppure

import itertools  
list(itertools.permutations(range(3), 2)) //lista e num posti

**Generare permutazioni**

l = range(3)  
disp\_rep = [(l1, l2, l3) for l1 in l for l2 in l for l3 in l]  
perm = [d for d in disp\_rep if len(d)==len(set(d))]

oppure

import itertools  
list(itertools.permutations(range(3), 3)) //lista e numero di posti

**Generare combinazioni**

l = range(4)  
disp\_rep = [(l1, l2) for l1 in l for l2 in l]  
comb = [frozenset(d) for d in disp\_rep if len(d)==len(set(d))]  
list(map(set, set(comb)))

oppure

import itertools  
list(itertools.combinations(range(4), 2)) //lista e num posti

**10 – Naive Bayes**

Una semplice applicazione del teorema di Bayes permette di costruire un particolare classificatore che prende il nome di classificatore naive Bayes.  
Un modo semplice per stimare le probabilità indicate è quello di osservare con che frequenza gli eventi associati si verificano nel dataset a nostra disposizione. In particolare,

* la probabilità P(𝑁|𝑀) può essere approssimata calcolando la frequenza relativa con cui un supereroe Marvel del dataset ha gli occhi neri;
* la probabilità P(𝑀) può essere approssimata calcolando la frequenza relativa con cui un supereroe del dataset è edito dalla Marvel;
* la probabilità P(𝑁) può essere approssimata calcolando la frequenza relativa con cui un supereroe del dataset ha gli occhi neri.

la libreria sklearn mette a disposizione delle classi che permettono di ottenere direttamente un classificatore a partire dai dati iniziali. Per esemplificarne il funzionamento, estraiamo dal nostro dataset alcune colonne ed eliminiamo tutte le righe in cui occorre almeno un valore mancante. è necessario utilizzare solamente dati numerici

features = ['Height', 'Weight', 'Gender', 'First appearance',

'Hair color', 'Eye color', 'Strength', 'Intelligence']

heroes\_cleaned = heroes.dropna()

X = heroes\_cleaned[features].copy()

Procediamo poi a convertire i valori degli attributi rimanenti nel modo che segue: dividiamo innanzitutto l'intervallo di variazione dei corrispondenti valori in quattro intervalli, utilizzando i quattro quartili come estremi; associamo successivamente un diverso valore numerico a ogni intervallo ottenuto e trasformiamo ogni osservazione nel numero che corrisponde all'intervallo individuato. Questa operazione viene effettuata automaticamente dalla funzione pd.qcut, a cui passiamo le osservazioni, il numero di intervalli da considerare e i valori numerici da associare a questi ultimi.

from sklearn.preprocessing import LabelEncoder

n = 4

X['Height'] = pd.qcut(X['Height'], n, labels=range(n))

X['Weight'] = pd.qcut(X['Weight'], n, labels=range(n))

gender\_encoder = LabelEncoder()

gender\_encoder.fit(X['Gender'])

X['Gender'] = gender\_encoder.transform(X['Gender'])

X['First appearance'] = pd.qcut(X['First appearance'],

n,

labels=range(n))

hair\_encoder = LabelEncoder()

hair\_encoder.fit(X['Hair color'])

X['Hair color'] = hair\_encoder.transform(X['Hair color'])

eye\_encoder = LabelEncoder()

eye\_encoder.fit(X['Eye color'])

X['Eye color'] = eye\_encoder.transform(X['Eye color'])

X['Strength'] = pd.qcut(X['Height'], n, labels=range(n))

X['Intelligence'] = pd.qcut(X['Height'], n, labels=range(n))

L'ultimo ingrediente mancante è l'insieme dei valori per le classi. Per semplificarci la vita, ritorniamo alla classificazione binaria inizialmente considerata: anche in questo caso è necessario lavorare con valori numerici, quindi costruiamo una serie i cui valori sono 1 in corrispondenza dei supereroi Marvel e 0 per tutti gli altri casi.

Y = (heroes\_cleaned['Publisher']=='Marvel Comics').apply(lambda x: 1 if x else 0)

In realtà sklearn implementa diverse varianti del classificatore naive Bayes, ognuna delle quali stima in modo diverso le probabilità P(𝑋𝑖=𝑥𝑖|𝑌=𝑦𝑘). Consideriamo per esempio la classe GaussianNB, che ipotizza che tali probabilità si possano calcolare utilizzando una distribuzione gaussiana (uno dei modelli teorici che vedremo tra qualche lezione). La costruzione del classificatore viene fatta usando lo stesso procedimento visto per gli alberi di decisione: si crea un oggetto della classe GaussianNB, e poi si invoca su di esso il metodo fit specificando come argomenti le osservazioni e le corrispondenti etichette. Dopo sarà sufficiente invocare il metodo predict passando una lista di oggetti da classificare. Ricordando che per avere una valutazione imparziale di un classificatore è necessario utilizzare dati non utilizzati per costruire il classificatore stesso, per semplicità limitiamoci a verificare che cosa succeda con le osservazioni in X.

from sklearn.naive\_bayes import GaussianNB  
model = GaussianNB()  
model.fit(X, Y)  
model.predict(X)

A partire dai vettori ottenuti è facile calcolare separatamente il numero di veri positivi, falsi positivi, falsi negativi e veri negativi. È anche possibile costruire un dataframe che contenga la corrispondente matrice di confusione.

tp = sum(model.predict(X[Y==1]))  
fn = len(model.predict(X[Y==1])) – tp  
fp = sum(model.predict(X[Y==0]))  
tn = len(model.predict(X[Y==0])) – fp  
pd.DataFrame([[tp, fn],

[fp, tn]],

index=['Etichetta positiva', 'Etichetta negativa'],

columns=['Classif. positiva', 'Classif. negativa'])

Volendo riassumere in un unico indice la bontà del risultato ottenuto è possibile calcolare la percentuale di dati corretti.

(tp + tn) / len(X)

**11 – Distribuzione Geometrica**

La distribuzione geometrica descrive il numero di insuccessi necessari affinché si verifichi il primo successo in una successione di esperimenti bernoulliani indipendenti e identicamente distribuiti. Questa distribuzione, che ha quindi come supporto l'insieme dei numeri naturali (zero incluso), è quindi completamente descritta specificando il parametro 𝑝 del corrispondente esperimento bernoulliano con 𝑝∈(0,1]

**Generare un numero casuale tra 0 e 1**

import random as rnd  
rnd.random()

**Generare una serie geometrica specificando p e la lunghezza dell’array risultato**

from scipy.stats import geom

g = geom(.3) //serie geometrica di parametro 0.3  
g.rvs(10) //10 elementi

Tenuto conto dell'indipendenza tra le ripetizioni dell'esperimento bernoulliano, richiedere che il primo successo sia avvenuto esattamente dopo 𝑖 insuccessi equivale a calcolare la probabilità di 𝑖 insuccessi successivi, seguiti da un successo. Pertanto, la distribuzione geometrica ha la seguente funzione di massa di probabilità:

𝑓𝑋(𝑖;𝑝)=𝑝(1−𝑝)^𝑖 \*Iℕ∪{0}(𝑖).

Nella cella seguente viene definita una funzione geom\_pdf che accetta come argomenti rispettivamente un valore reale 𝑥 e un possibile parametro 𝑝 e restituisce il valore della funzione di probabilità geometrica. La funzione è costruito in modo da associare massa nulla ai punti al di fuori del dominio, verificando anche in questo caso che il parametro specificato assuma un valore valido

def geom\_pdf(x, p):

assert p > 0 and p <= 1, '{} is not a valid parameter ' \

'for the geometric distribution.'.format(p)

return p \* (1 - p)\*\*x if x > 0 and x==int(x) else 0

È quindi possibile visualizzare i valori di massa di probabilità, fissando per esempio 𝑝=12p=12 e usando un grafico a bastoncini (così da evidenziare che le masse di probabilità sono associate a singoli valori). Ovviamente la visualizzazione dovrà coinvolgere un sottoinsieme delle possibili specificazioni, essendo il dominio della distribuzione infinito

%matplotlib inline

from ipywidgets import \*

import matplotlib.pyplot as plt

import numpy as np

def gr\_geom\_pdf(p):  
 x = np.arange(0, 10, 1)  
 plt.vlines(x, [0]\*len(x), list(map(lambda \_: geom\_pdf(\_, p), x)))  
 plt.plot(x, list(map(lambda \_: geom\_pdf(\_, p), x)), 'o')  
 plt.ylim(ymax=1, ymin=0)  
 plt.xlim(xmax=11, xmin = -1)  
 plt.show()

interact(gr\_geom\_pdf, p=(0.1, 1, 0.1))   
plt.show()

**Generare il grafico dell’andamento del valore atteso in funzione del parametro p**

x = np.arange(0.01, 1.01, 0.01)  
y = list(map(lambda p: (1-p)/p, x))

plt.plot(x, y)  
plt.ylim(0, 50)  
plt.show()

Il risultato è coerente con il significato di 𝑝p: più il suo valore si avvicina a 11, più è probabile che si ottenga un successo nel primo esperimento, così che il numero atteso di insuccessi tende a zero. Parimenti, al diminuire di 𝑝p diventa meno probabile ottenere un successo e quindi il numero atteso di insuccessi aumenta. In particolare, al tendere di 𝑝p a zero l'esperimento bernoulliano alla base della distribuzione geometrica non avrà mai successo e quindi il numero atteso di insuccessi tenderà a infinito.

**Generare il grafico dell’andamento della varianza in funzione del parametro p**

x = np.arange(0.01, 1.01, 0.01)  
y = list(map(lambda p: (1-p)/p\*\*2, x))

plt.plot(x, y)  
plt.ylim(0, 50)  
plt.show()

Anche in questo caso il grafico ottenuto è coerente con il signigicato probabilistico della distribuzione: al diminuire di 𝑝p la massa di probabilità si concentra su un insieme sempre più grande di valori, dunque la varianza aumenta. Quando 𝑝=1p=1 il numero di insuccessi è sempre nullo e quindi la variabile aleatoria degenera in una costante che per definizione ha varianza nulla.

**Accoppiare la visualizzazione della funzione di massa di probabilità a un'indicazione del valore atteso e della dispersione della distribuzione**

%matplotlib inline

from ipywidgets import \*  
import matplotlib.pyplot as plt  
import matplotlib.patches as patches  
import numpy as np

def geom\_pdf(x, p):

assert p > 0 and p <= 1, '{} is not a valid parameter ' \

'for the geometric distribution.'.format(p)

return p \* (1 - p)\*\*x if x==int(x) else 0

def gr\_geom\_pdf(p):

x = np.arange(0, 10, 1)   
avg = (1 - p) / p  
stdev = (1-p)\*\*0.5/p  
plt.gca().add\_patch(patches.Rectangle((avg-stdev, 0.95), 2\*stdev, 0.05, edgecolor='None', facecolor='green'))

plt.plot([avg, avg], [0.9, 1], color='green')  
plt.vlines(x, [0]\*len(x), list(map(lambda \_: geom\_pdf(\_, p), x)))  
plt.plot(x, list(map(lambda \_: geom\_pdf(\_, p), x)), 'o')  
plt.ylim(ymax=1, ymin=0)  
plt.xlim(xmax=11, xmin = -1)  
plt.show()

interact(gr\_geom\_pdf, p=(0.1, 1, 0.1))   
plt.show()

**Visualizzare il grafico della funzione di ripartizione**

Immagine che contiene testo

Descrizione generata automaticamente

def geom\_cdf(x, p):

assert p > 0 and p <= 1, '{} is not a valid parameter ' \

'for the geometric distribution.'.format(p)

return 1 - (1-p)\*\*(int(x) + 1) if x >= 0 else 0

Il grafico della funzione di ripartizione è visualizzato qui di seguito. Anche in questo caso la visualizzazione, fatta scegliendo p=0.5, è modificabile nella versione interattiva degli appunti.

def gr\_geom\_cdf(p):

x = np.arange(-2, 10, .1)  
y = list(map(lambda \_: geom\_cdf(\_, p), x))  
plt.step(x, y)

plt.ylim(ymax=1, ymin=-.1)  
plt.xlim(xmax=10, xmin = -2)  
plt.show()

interact(gr\_geom\_cdf, p=(0.1, 1, 0.1))  
plt.show()

**11 – Distribuzione Geometrica**

**Disegnare un grafico del funz. Di massa di probabilità di una V.A. di Poisson con paramtero 𝜆**

import scipy.stats as st

l = 5 //parametro **𝜆**  
x = range(15) //prime 15 specificazioni

X = st.poisson(l) //instanzio un oggetto della distribuzione  
x=x = range(15) //  
X.pmf(x) //calcolo della funzione di massa di probabilità

Il calcolo della funzione di massa di probabilità viene effettuato invocando il metodo pmf, a cui passare una specificazione, oppure una lista o un array di specificazioni: la funzione restituirà un valore nel primo caso e un array nei casi rimanenti.

plt.vlines(x, 0, X.pmf(x))  
plt.plot(x, X.pmf(x), 'o')  
plt.show()

**Si calcoli la probabilità che 𝑋 V.A. di Poisson con parametro 5 assuma valori maggiori di 6**

Sfruttiamo il fatto che P(𝑋>6)=1−P(𝑋≤6)

1-sum([X.pmf(i) for i in range(7)]) //pmf=probability mass function

Oppure

1-X.cdf(6) //cdf=cumulative distribution function

Si determini il più piccolo valore di 𝑋, chiamiamolo 𝑥0.8, tale che 𝑃(𝑋≤𝑥0.8)≥0.8

Siccome 𝑃(𝑋≤𝑥)≥0.8 equivale a 𝐹𝑋(𝑥)≥0.8, un modo operativo per trovare il valore di 𝑥0.8 è quello di calcolare la funzione di ripartizione di 𝑋 per valori crescenti delle specificazioni, fermandosi la prima volta che si supera 0.8. Per calcolare agevolmente 𝐹𝑋FX si può invocare il metodo cdf

x = 0

while X.cdf(x) < 0.8:

x +=1

print(x) //stamperà il valore della specificazione di X che supera 0.8

**Si calcoli il quantile 0.8 di X**

X.ppf(0.8) //dovrebbe essere lo stesso valore sopra trovato

Il package non mette a disposizione metodi per il calcolo esplicito di percentili, quartili o decili, che però si possono ottenere facilmente in termini del corrispondente quantile: per esempio il 34-esimo percentile coincide con il quantile 0.34, il sesto decile coincide con il quantile 0.6 e così via

**Contare il numero di casi di un dataframe**

import pandas as pd

acqua = pd.read\_csv('data/ComposizioneAcqua.csv')  
len(acqua)

**Quanti litri sono stati analizzati in totale se ogni misurazione è basta su 5 litri?**

len(acqua) \* 5

**Quante sorgenti sono state analizzate? Vedi il campo “NomeSorgente”**

acqua['NomeSorgente'].unique() //estraggo in maniera univoca il nome

pd.DataFrame(acqua['NomeSorgente'].unique()) //visualizzo come dataframe

len(acqua['NomeSorgente'].unique()) //numero di sorgenti

**Le diverse sorgenti sono rappresentate in modo uniforme nel dataset?**

Il modo in cui le diverse sorgenti sono rappresentate nel *dataset* viene calcolato in termini della rispettiva eterogeneità. A sua volta, l'eterogeneità si può calcolare usando diversi indici. Richiamiamo qui di seguito l'implementazione del calcolo dell'indice di Gini nella sua versione originale (la funzione gini) e in quella normalizzata

def gini(series):

return 1 - sum(series.value\_counts(normalize=True)

.map(lambda f: f\*\*2))

def normalized\_gini(series):

s = num\_values(series) //funz(da definire)che conta il numero di valori

return s \* gini(series) / (s-1)

Se il valore è prossimo all'unità, denotando dunque un'elevata eterogeneità che corrisponde a un alto livello di uniformità per le sorgenti.

**Si calcoli la tabella delle frequenze delle particelle di oro su 5 litri di acqua e si visualizzi un grafico**

gold\_rel\_freq = pd.crosstab(index=acqua['Oro'],

columns=['Freqenza'],

normalize=True,

colnames=[''])

plt.vlines(gold\_rel\_freq.index, 0, gold\_rel\_freq.values)  
plt.plot(gold\_rel\_freq.index, gold\_rel\_freq.values, 'o')  
plt.show() //grafico a bastoncini migliore

**La distribuzione delle frequenze osservata è compatibile con un modello di Poisson?**

Se si ordinano le frequenze rispetto a valori crescenti delle specificazioni, in un modello di Poisson si possono ottenere due andamenti: uno strettamente decrescente e uno in cui i valori crescono fino a un massimo per poi decrescere, e in ogni caso allontanandosi dal massimo si ha una diminuzione di carattere non lineare. Guardando il grafico generato nel punto precedente si vede come le frequenze studiate ricadano nel secondo dei casi possibili, per cui a una prima analisi i dati osservati non sono incompatibili con un modello di Poisson.

acqua['Oro'].mean() //media

acqua['Oro'].var() //varianza

Nel modello di Poisson media e varianza coincidono. Se sono diversi dobbiamo confutare l'ipotesi di applicabilità di tale modello per l'attributo *Oro*.

**Si stimi il numero atteso di particelle di oro riscontrate in 5 litri di acqua**

Al punto precedente abbiamo già stimato il valore atteso del numero di particelle di oro calcolando la media campionaria sulla serie corrispondente, ottenendo un risultato approssimativamente uguale a 1.72

**Sia 𝑋 la variabile casuale che conta il numero di particelle di oro riscontrate in 5 litri di acqua. Scrivere lo stimatore utilizzato al punto precedente, specificare la numerosità del campione a cui è applicato e dire se è uno stimatore non distorto**

Immagine che contiene testo

Descrizione generata automaticamente

pd.isnull(acqua['Oro']).sum() //se zero allora non ci sono valori mancanti

pd.isnull(acqua['Oro']).any() //se False allora non ci sono valori mancanti

Infine, sappiamo che la media campionaria è sempre uno stimatore non distorto per il valore atteso della popolazione.

**Si tracci un grafico opportuno per visualizzare la distribuzione della durezza dell'acqua**

A differenza dell'attributo Oro, i valori per DurezzaAcqua sono numeri reali, e quindi una descrizione grafica ragionevole richiede la generazione di un istogramma

acqua['DurezzaAcqua'].plot.hist()  
plt.show()

**La distribuzione delle frequenze osservata è compatibile con un modello normale?**

Dal grafico precedente si vede come l'istogramma abbia una forma bimodale, cioè con due massimi locali. Il modello normale è invece caratterizzato da una forma a campana. Pertanto i dati osservati non sono compatibili con una distribuzione normale. Possiamo approfondire l'analisi visualizzando il diagramma Q-Q per l'attributo e sovrapponendolo a una linea che indica la curva attesa in caso di normalità dei dati. Per fare ciò possiamo utilizzare la funzione qqplot

import statsmodels.api as sm

sm.qqplot(acqua['DurezzaAcqua'], fit=True, line='45')  
plt.show()

Nel codice utilizzato, l'argomento fit=True indica di standardizzare i dati in modo che la curva attesa sia sovrapposta alla bisettrice del primo e del terzo quadrante, mentre line='45' specifica di visualizzare quest'ultima bisettrice. Dal grafico ottenuto si ottiene ulteriore conferma del fatto che il modello normale non sia compatibile con i dati rilevati per la durezza dell'acqua

**L'ufficio analisi chimiche ipotizza che ci sia una relazione tra la quantità di oro riscontrata e la durezza dell'acqua. Si produca un grafico e si calcoli un indice numerico per convincere il titolare della ditta che l'ipotesi è accettabile**

Per valutare un'eventuale relazione tra i due attributi tracciamo il diagramma di dispersione corrispondente

acqua.plot.scatter('Oro', 'DurezzaAcqua')  
plt.show()

In effetti il grafico evidenzia una relazione di tipo inverso con un andamento di carattere lineare. Il coefficiente di correlazione lineare ci permette di quantificare la forza di questa relazione

acqua['Oro'].corr(acqua['DurezzaAcqua'])

Se il coefficiente di correlazione sia sufficientemente vicino a -1 conferma che tra i due attributi sussista una relazione lineare di tipo inverso e di carattere abbastanza forte, viceversa altrimenti

**Ai chimici viene il sospetto che non tutte le sorgenti siano caratterizzate dalla stessa durezza media dell'acqua, e in particolare che nel dataset ci siano due gruppi distinti dal punto di vista della durezza dell'acqua. Si valuti se questa ipotesi è condivisibile**

L'ipotesi che vi siano due gruppi distinti è corroborata dall'istogramma della durezza dell'acqua, che è interpretabile come la sovrapposizione di due campane

**Si calcoli la durezza media dell'acqua per ogni sorgente rappresentata nel dataset**

acqua.groupby('NomeSorgente').mean()

**Selezionate e memorizzate nella variabile sorgente\_5 gli attributi Oro e DurezzaAcqua soltanto della sorgente 5; selezionate e memorizzate nella variabile altre\_sorgenti i medesimi attributi per tutte le altre sorgenti**

sorgente\_5 = acqua[acqua['NomeSorgente']=='Sorgente5']  
altre\_sorgenti = acqua[acqua['NomeSorgente']!='Sorgente5']

**Si stimi il numero atteso di particelle di oro riscontrate in 5 litri di acqua nel caso della sorgente 5 e per tutte le altre sorgenti**

num\_particelle\_5 = sorgente\_5['Oro'].mean()  
altre\_sorgenti['Oro'].mean()

**La distribuzione delle frequenze osservate per la sorgente 5 è compatibile con un modello di Poisson?**

s5\_rel\_freq = sorgente\_5['Oro'].value\_counts(normalize=True, sort=False)  
plt.vlines(s5\_rel\_freq.index, [0]\*len(s5\_rel\_freq), s5\_rel\_freq.values)  
plt.plot(s5\_rel\_freq.index, s5\_rel\_freq.values, 'o')  
plt.show()

Se l’andamento è unimodale e il valore atteso e la varianza sono molto simili allora si può dire che è compatibile

(sorgente\_5['Oro'].mean(), sorgente\_5['Oro'].var()) //deve essere molto vicina

**Durante l'attività di filtraggio dalla sorgente 5, una particella di oro ha inceppato il dispositivo di filtraggio dell'acqua. Si calcoli la probabilità di non trovare altre particelle di oro nei prossimi 10 litri di acqua, dopo aver sbloccato il dispositivo**

Siccome il numero 𝑋 di particelle d'oro per cinque litri di acqua provenienti dalla sorgente 5 è descrivibile tramite un modello di Poisson, la quantità di litri di acqua da analizzare tra due scoperte successive di una particella è descrivibile tramite una legge esponenziale di parametro uguale al valore atteso di 𝑋 diviso per 5. Noi non conosciamo il valore atteso di 𝑋 , ma lo abbiamo già stimato e a partire da tale stima possiamo ottenerne una per il parametro della legge esponenziale che ci interessa

l\_5 = sorgente\_5['Oro'].mean()/5

Ora, indicata con 𝑇T la variabile aleatoria che corrisponde alla quantità di litri d'acqua da analizzare prima di trovare la prossima particella di oro, la probabilità di non trovare particelle nei prossimi dieci litri sarà pari a

P(𝑇>10)=1−P(𝑇≤10)=1−𝐹𝑇(10)

Per calcolare questa probabilità possiamo utilizzare il package scipy.stats, che mette a disposizione una classe expon per le distribuzioni esponenziali. Per motivi che esulano da questa esercitazione, la sua istanziazione richiede però di passare come valore per l'argomento scale l'inverso del parametro considerato.

T\_5 = st.expon(scale=1/l\_5)  
1 - T\_5.cdf(10) //valore cercato

**Dal punto in cui il dispositivo applicato alla sorgente 5 ha individuato l'ultima particella di oro sono stati filtrati altri 5 litri di acqua senza trovare altro oro. Qual è la probabilità di non trovare oro ancora per i prossimi 10 litri di acqua?**

La proprietà di assenza di memoria della distribuzione esponenziale ci permette di dire che la probabilità che non si trovino particelle nei dieci litri che seguono l'ultima scoperta è uguale alla probabilità che i primi dieci litri analizzati quando è partito il macchinario non abbiano portato a trovare alcuna particella. Tale probabilità è quindi quella calcolata al punto precedente

**Si tracci il grafico della funzione di ripartizione della variabile casuale 𝑌=Y= "quantità di acqua da analizzare (espressa in litri) prima di incontrare la prossima particella di oro" nel caso della sorgente 5 e per tutte le altre sorgenti**

Per quanto visto ai punti precedenti, la variabile aleatoria 𝑌Y segue una legge esponenziale descritta dall'oggetto memorizzato in T\_5

import numpy as np

t = np.arange(0, 10, .1)

plt.plot(t, T\_5.cdf(t))

plt.show()

Per quanto riguarda le altre sorgenti, una volta verificata la validità dell'assunzione legata ai modelli di Poisson ed esponenziale (attività lasciata come esercizio) basta procedere in modo analogo a quanto visto sopra

l\_altre = altre\_sorgenti['Oro'].mean()/5

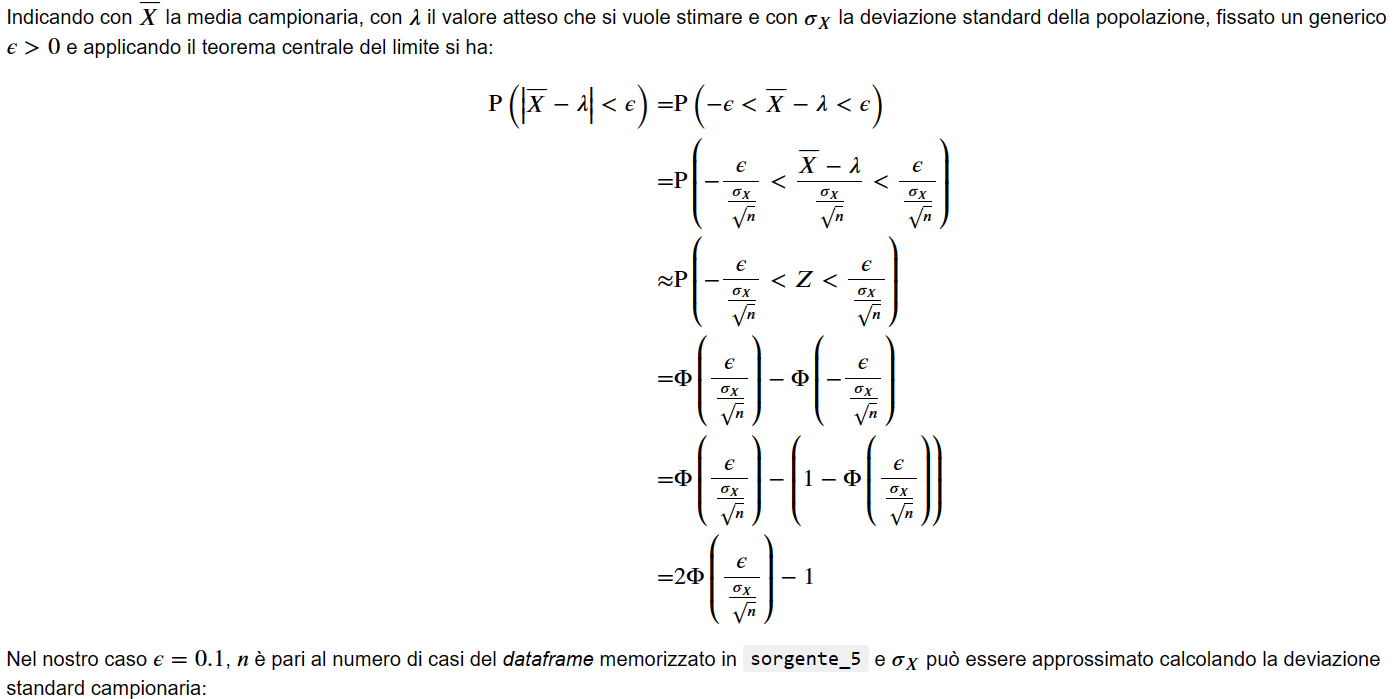
T\_altre = st.expon(scale=1/l\_altre)

t = np.arange(0, 20, .1)

plt.plot(t, T\_altre.cdf(t))

plt.show()

**Secondo il titolare dell'azienda il valore atteso di particelle di oro riscontrate è abbastanza elevato da poter pensare di estrarre l'oro dall'acqua per venderlo. Prima di iniziare questa nuova attività è bene soppesare l'errore compiuto nella stima di tale parametro. Calcolare la probabilità che, per la sorgente 5, l'errore compiuto nella stima del valore atteso di particelle di oro riscontrate in 5 litri di acqua sia al più di 0.1 particelle, in eccesso o in difetto**



import scipy.stats as st

sigma\_x = sorgente\_5['Oro'].std()

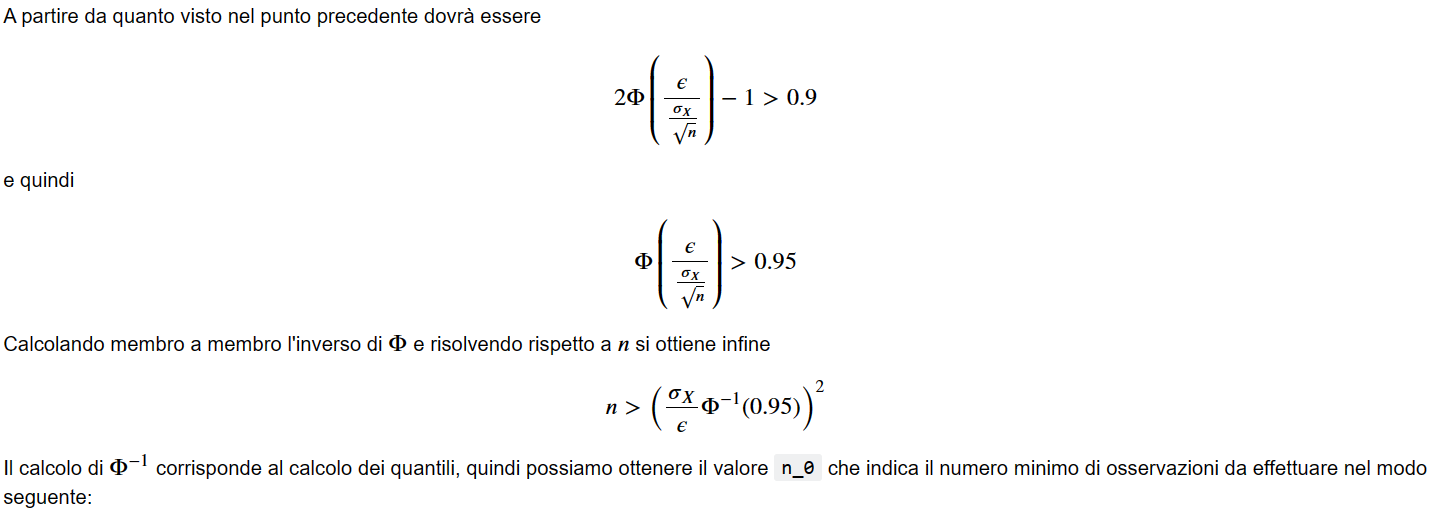
eps = 0.1

n = len(sorgente\_5)

Z = st.norm()

2 \* Z.cdf(eps \* n\*\*0.5 / sigma\_x) – 1 //valore finale di probabilità

**Quanti litri di acqua si dovrebbero ancora analizzare affinché la probabilità che, per la sorgente 5, l'errore compiuto nella stima del valore atteso di particelle di oro riscontrate in 5 litri di acqua sia al più di 0.1 particelle, in eccesso o in difetto sia almeno uguale a 0.9?**



n\_0 = math.ceil((sigma\_x / eps \* Z.ppf(0.95))\*\*2)

n\_0 - len(sorgente\_5) //numero di osservazioni necessarie

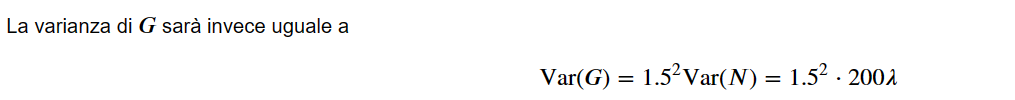
5 \* (n\_0 - len(sorgente\_5)) //numero litri di acqua

**Per semplicità ipotizziamo che una particella estratta dalla sorgente 5 possa essere venduta a 1.5 euro e che il dispositivo per l'estrazione dell'oro abbia un costo fisso di 1000 euro. Stimare il guadagno atteso nel caso in cui si voglia estrarre l'oro da 1000 litri di acqua e la varianza del guadagno**

Immagine che contiene testo

Descrizione generata automaticamente

100 \* (3 \* num\_particelle\_5 - 10) //guadagno atteso di questo problema



var\_g = 1.5\*\*2 \* 200 \* num\_particelle\_5 //varianza di G di questo problema

var\_g\*\*0.5 //deviazione standard

**Generare due grafici affianco**

plt.figure(figsize=(10, 3))

plt.subplot(1, 2, 1) //crea un subplot

strength.plot.hist(bins=20)

plt.xlabel('Original')

plt.subplot(1, 2, 2)

transformed\_strength.plot.hist(bins=20)

plt.xlabel('Transformed')

plt.show()

Unire due serie:

import pandas as pd

serie1 = pd.Series({'A': 1, 'B': 2, 'C': 3})

serie2 = pd.Series({'B': 3, 'C': 4, 'D': 5})

fused\_series = pd.merge(serie1, serie2, on=None, how='left')

Nell'esempio sopra, le due serie serie1 e serie2 vengono fuse sulla base delle loro etichette utilizzando l'opzione how='left'. Questo significa che i valori presenti solo in serie1 verranno mantenuti e i valori presenti solo in serie2 verranno trascurati.

**12 – Correzione Esami**

**Prova di Giugno 2022**

**Disegnare a mano il grafico della funzione di ripartizione (distribuzione) cumulativa di una variabile aleatoria E esponenziale di parametro sconosciuto**

Ogni variabile aleatoria esponenziale ha il grafico fatto allo stesso modo, nullo prima di 0 e poi cresce esponenzialmente con 1 come asintoto orizzontale, in funzione del valore di lambda raggiunge più o meno velocemente l’asintoto. Sugli assi mettiamo x sulle ascisse per indicare una specificazione e sull’asse delle ordinate possiamo mettere F   
Immagine che contiene testo

Descrizione generata automaticamente

In Phyton:  
param\_lambda=0.1 //parametro della esponenziale

E=st.expon(scale=1/param\_lambda) //creo una esponenziale con 1/parametro per questioni tecniche

x=np.arange(-10,E.ppf(0.99),0.1) //tot di valori tra -10 e il 99 quantile

y=E.cdf(x) //funzione di ripartizione di E dati i valori x

plt.plot(x,y)

**Disegnare a mano il grafico della funzione di ripartizione (distribuzione) cumulativa di una variabile aleatoria normale N con valore atteso uguale a quello di E (sopra) e varianza sconosciuta**

Il grafico di una distribuzione normale è un grafico con ad S, parte da 0 e poi cresce e poi va ad 1 seguendo l’asintoto, avendo valore atteso uguale ad E bisogna cercare di mantenere la localizzazione della distribuzione in modo coerente rispetto alla curva di E dato che il valore atteso dovrebbe essere uguale. L’area al di sopra della funzione di ripartizione nelle esponenziali è uguale esattamente al valore atteso e se considero una variabile normale con densità nulla per ogni valore negativo di x il grafico può essere rappresentato come segue, compensando l’area della normale per avere circa la stessa area

Immagine che contiene lavagnabianca

Descrizione generata automaticamente//siccome valore atteso uguale la prima parte è sotto quella di E e poi va sopra

In Phyton:

N=st.norm(1/param\_lambda, 3) //dev.stand a piacere e stesso valore atteso

x=np.arange(-10,E.ppf(0.99),0.1) //valori ancora nello stesso range

y\_n=N.cdf(x) //funzione di ripartizione della normale

plt.plot(x,y\_n)

**Sia**  **la media campionaria di un campione casuale X1,X2..Xn estratto da una popolazione X distribuita come N dell’esercizio precedente. Che cosa ha senso stimare utilizzando la media campionaria come statistica?**

Ha sempre senso utilizzare la media campionaria per stimare il valore atteso della popolazione o della VA siccome è non distorto e consistente in media quadratica

**Esprimete, in un opportuno sottoinsieme dei parametri** 

La media campionaria ha come valore atteso il valore atteso della popolazione distribuita con modello normale con media 1/lambda. L’opportuno sottoinsieme dei parametri è il sottoinsieme che conteneva solo lambda con valore atteso 1/lambda

**Esprimete, in un opportuno sottoinsieme dei parametri** 

La varianza della media campionaria è uguale alla varianza della popolazione diviso n e avendo sigma^2 quindi la risposta è sigmaì^2/n

**Sia epsilon>0 un valore prefissato, determinare un valore minimo per n affinché con probabilità non inferiore a 0.9 la stima del punto 1 disti al più epsilon del valore sconosciuto. Da quali quantità dipende questa soglia?**

Determiniamo la taglia minima del campione (numerosità del campione). La stima del punto 1 era la stima che noi facciamo usando la media campionaria per stimare il valore atteso della popolazione (1/lamda) e che l’errore sia al più epsilon.

Siamo lavorando con un campione estratto da una normale la cui media campionaria segue una distribuzione normale; quindi, il teorema centrale del limite si può non applicare, e so che il risultato della standardizzazione segue un modello normale con media 0 e varianza 1 quindi è una variabile aleatoria standard e posso utilizzare la proprietà di simmetria della densità normale standard

Immagine che contiene testo

Descrizione generata automaticamente  
quindi si impone che sia >= 0.9 e si risolve rispetto ad n:  
 ovvero Immagine che contiene testo

Descrizione generata automaticamente

**Importare un dataset con separatore di colonne ; e con i numeri reali usano il carattere . come separatore dei decimali**

Il separatore dei decimali è quello di default, usando decimal=’.’ Si potrebbe specificare

pd.read\_csv('nomefile.csv', sep=';')

**Definire ogni colonna se è un attributo nominale, ordinale o scalare**

Un attributo nominale sono stringhe o etichette. Un attributo scalare è un attributo su cui ha poco senso considerare un solo attributo ma ha senso accorparli e fare operazioni matematiche (es: temperatura massima ecc..), gli ordinali sono ad esempio il giorno in cui è stata fatta la misurazione

**Indicare quante zone sono indicate nel dataset (zone geografiche) e dire se sono rappresentate in modo uniforme tramite un indice**

len(Precipitazioni[‘zona’].unique()) //numero delle zone

Per valutare l’uniformità con un indice posso usare l’indice di Gini per valutare l’eterogeneità

nz= len(Precipitazioni[‘zona’].unique

(1-sum(Precipitazioni[‘zona’].valuecounts(normalize=True)\*\*2))\*nz/(nz-1)

Se vicino ad 1 è uniforme

**Descrivere in modo opportuno l’attributo umidità utilizzando un grafico motivando la scelta**

L’umidità è di tipo scalare e possiamo usare un boxplot o un istogramma, l’ultimo ha senso perché posso raggruppare i valori vicini

precipitazioni[‘umidità’].plot.hist()

plot.show()

**L’umidità ha dei valori mancanti? Quanti?**

precipitazioni[‘umidità’].isnull().sum() //somma dei true

**L’umidità ha dei valori outlier? Quanti?**

precipitazioni[‘umidità’].plot.box() //provo a vedere se ho dei pallini oltre le stanghette, se non ne ho non ho valori fuoriscala

plot.show()

**Esiste una relazione tra temperatura minima e umidità?**

Mi serve visualizzare un diagramma di dispersione

precipitazioni.plot.scatter(‘temp\_min’,’umidita’)

plt.show()

precipitazioni[‘temp\_min’].corr(precipitazioni[‘umidita]) //indice correlaz, se vicino ad 1 ho una relazione forte diretta

**Utilizziamo ora il campione centro\_autunno e presupponiamo che i valori dell’attributo pioggia siano assimilabili a un campione aleatorio estratto da una popolazione descritta da una variabile aleatoria X**

**Fornite un valore numerico che rappresenti una stima del valore atteso del valore X indicando quale è la dimensione del campione, quale stimatore è stato utilizzato e specificando se questo goda di particolari proprietà teoriche**

Posso usare la media campionaria che è uno stimatore non distorto per il valore atteso della popolazione ed è consistente in media quadratica

centro\_autunno[‘pioggia’].mean()

Per la numerosità del campione mi server verificare se tutto il campione è usato

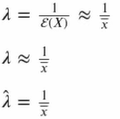
centro\_autunno[‘pioggia’].isnull().any() //se ritorna false non ci sono NA

len(centro\_autunno[‘pioggia’]) //lunghezza del campione

**I metereologi vogliono modellare la quantità dell’acqua piovuta nel centro storico ma sono indecisi se usare una distribuzione esponenziale o una normale. Ci concentriamo su una ipotesi alla volta.**

**Fornite una stime numerica del parametro lambda della ipotetica distribuzione esponenziale indicando anche quale stimatore è stato utilizzato ed elencando le proprietà interessanti, è possibile confermare o confutare l’ipotesi che i valori di temperatura osservati rappresentino un campione estratto dalla distribuzione esponenziale con parametro stimato?**

Il valore atteso è l’inverso del parametro e posso ottenere che lambda è uguale ad 1/valore atteso(X). Quindi 1/valoreatteso(X) è circa 1\media campionaria

quindi possiamo usare come stimatore l’inversa della media campionaria

1/centro\_autunno[‘pioggia’].mean()

Lo stimatore non ha particolari proprietà dato che per misurare se è non distorto dovrei calcolare il valore atteso dell’inverso della media campionaria (non visto a lezione)

Per confermare o confutare posso provare a tracciare la funzione di ripartizione e vederne la forma o tracciare l’istogramma e verificare che abbia la forma della densità di una esponenziale che è strettamente decrescente, se non lo è non ci siamo.

**Rifare lo stesso del punto sopra con una normale**

Utilizzando un QQplot possiamo verificare l’ipotesi

qqplot(centro\_autunno[‘pioggia’], fit=True, line=’45’)

se i punti sono abbastanza allineati con la bisettrice non possiamo confutare l’ipotesi che sia una normale, se devo scegliere tra una esponenziale e una normale scelgo una normale

**Sulla base dell’esercizio precedente i meteorologi volgono la loro attenzione su una variabile aleatoria Y che misura le precipitazioni totali nel centro nell’arco di un mese**

**Che modello andrebbe usato per Y?**

Sarà la somma dei millimetri di acqua caduti in ogni giorno del mese ipotetico, ipotizziamo 30 giorni. Quindi Y non è altro che  con le Xi le variabili che indicano i millimetri che piovono in un giorno, dato che le Xi seguono un modello normale Y seguirà un modello normale aggiungendo l’ipotesi di indipendenza. Y avrà valore atteso 30\*il valore atteso delle X e deviazione standard

**I vigili del fuoco hanno informato il consiglio comunale che possono gestire emergenze fino a 120mm di precipitazioni totali in un mese. Quale è la probabilità che le precipitazioni superino questa soglia?**

Sarà necessario stimare il valore atteso di Y e quello della deviazione standard, siccome abbiamo stimato quello di X otteniamo:  
mu\_y=30\* centro\_autunno[‘pioggia’].mean()

std\_y=(30\*\*0.5)\* centro\_autunno[‘pioggia’].std()

Y=st.norm(mu\_y,std\_y)

Ora posso calcolare :  
1-Y.cdf(120)

**È possibile asserire che con probabilità maggiore di 0.9 la stima del valore atteso del valore atteso X disti in valore assoluto meno di mezzo millimetro dal valore corretto?**

rain\_mean=centro\_autunno[‘pioggia’].mean() //stima valore atteso

Immagine che contiene testo

Descrizione generata automaticamente

dove sigma^2 è la varianza della popolazione

2.7\*centro\_autunno[‘pioggia’].var()/0.5\*\*2 //meno di mezzo mm

Se nel nostro campione ci sono almeno gli stessi casi siamo tranquilli

**La risposta alla domanda precedente si basa su qualche tipo di approssimazione?**

Stiamo viaggiando sull’ipotesi che la popolazione sia normale, arrivo a dimostrare la formula senza applicare il teorema al centrale del limite, però ho dovuto comunque stimare il valore di sigma^2 e anche il valore 2.7 è un valore approssimato

**Prova di Febbraio 2022**

**Sia X una variabile casuale che segue una legge binomiale di parametri n e p [0,1]. Quali valori può assumere X?**

Una distribuzione binomiale può assumere tutti i valori da 0 ad n dato che misura i successi

**Per n fissato che valori può assumere il valore atteso di X**

Nella distribuzione binomiale il valore atteso è uguale ad n\*p

**Che valori può assumere la sua varianza?**

La varianza nel modello binomiale è definita come n\*p(1-p)

**Per n fissato, tracciate su un riferimento cartesiano il grafico della funzione di massa di probabilità della binomiale al variare di p che succede?**

Una binomiale ha funzione di massa di probabilità a forma di campana dove quando p=0.5 abbiamo la moda al centro, quando la moda è >0.5 il grafico si sposta verso destra e quando <0.5 si sposta a sinistra

**Per n fissato, tracciate su due sistemi di riferimento cartesiano i grafici del valore atteso e della varianza al variare di p , evidenziando tutte le informazioni che ritenete rilevanti**

Il valore atteso per la binomiale è n\*p, invece la varianza è uguale ad n\*p\*(1-p)

def valatteso(x):

return 7\*x

def var(x):

return 7\*x\*(1-x)

p=np.arange(0,1,0.1)

plt.plot(p,valatteso(p))

plt.plot(p,var(p)) //separati, prima uno poi l’altro

Immagine che contiene testo

Descrizione generata automaticamente

La media campionaria di X1,..Xm è con i da 1 ad m. dunque per linearità il valore atteso della media è dato che ogni Xi è distribuita come X e ipotizziamo l’indipendenza il valore atteso è lo stesso di X. Quindi il valore atteso di E( )= E(X)=n\*p

La varianza della media si calcola come la varianza di x fratto la numerosità del campione: VAR(X)/m

Il massimo della varianza di una variabile bernoulliana avviene quando p=1/2, dunque la VAR(X) ha valore massimo quando VAR(X)=n\*1/2\*(1-1/2))=n/4

Calcolando ora VAR() viene che il valore massimo è VAR(X)/m che porta a n/4m quindi è sempre minore o uguale al suo massimo

Il prossimo punto si esegue per chebyshev: Immagine che contiene shoji

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Per stimare p posso stimare il valore atteso con la media campionaria e con la formula inversa calcolare p, ovvero E(X)=n\*p 🡪 p=E(X)/n. Il valore atteso della statistica non è altro che il valore atteso della popolazione perché non distorto per il valore atteso e ottenendo E()=n\*p che è diverso da p e quindi è distorto in ogni caso.

Il BIAS viene calcolato come E()-tau(t)= E()-p=n\*p-p

Uno stimatore non distorto è Tx=/m dato che E(/n)=(1/n)\*E()= (1/n)\*n\*p=p quindi il valore atteso della mia statistica stima esattamente quello che voglio stimare

E(X)=n\*p il che significa p=E(X)/n da cui ricaviamo 1-p=1-E(X)/n

Per stimare 1-p basta usare la stessa statistica di prima ma con un -1 davanti Ty=1- (/m) in modo che pe rle proprietà del valore atteso otttengo E(Ty)=1-E(Tx)=1-(1/n)\*n\*p=1-p quindi è non distorto, oltretutto lo astatistica era non distorta per p e lo è anche per 1-p

**Qual è la percentuale dei casi nel dataset che contengono almeno un valore mancante?**len(cars.dropna()) / len(cars) //non sono convinto

**Tracciate il boxplot del carattere consumo e determinate qual è o quali sono i modelli di auto che possono essere considerati degli outlier rispetto a questo attributo. Basandovi poi esclusivamente sull'analisi qualitativa del grafico prodotto, fornite delle approssimazioni del primo, secondo e terzo quartile dell'attributo consumo**

cars['consumo'].plot.box()

plt.show() //ommetendo l'argomento whis

// dal grafico vediamo che c'e un unico outlier, il minimo

minimoconsumo = cars['consumo'].min() cars[cars['consumo'] == minimoconsumo].modello

guardando il box plot, il primo e il terzo quartile corrispondo alle due basi, la mediana la invece è la riga verde, si possono calcolare

**Verificate che la percentuale di outlier nell'attributo consumo equivale alla percentuale di casi del dataset che hanno almeno un valore mancante. Questo fatto è da ascriversi a pura casualità oppure esiste un nesso tra i due concetti?** Correlazione?

**Tracciate un grafico, diverso dal boxplot, che secondo voi ben rappresenta la distribuzione dell'attributo consumo. Giustificate la vostra scelta**

cars['consumo'].hist() plt.show()

siccome i dati sono decimali ha senso raggrupparli e usare un istiogramma

**I dati evidenziano una relazione tra il consumo e la cilindrata di un'auto? In caso affermativo, di che tipo è tale relazione? Motivate la vostra risposta anche sulla base di un'opportuna visualizzazione grafica e sul calcolo di un indice numerico.**

Per analizzare la presenza di una relazione posso usare uno scatter plot che mostra se esiste una relazione tra i dati

cars.plot.scatter('consumo','cilindrata')

plt.show()

e vediamo che c’è una sorta di relazione lineare diretta, possiamo quindi averne conferma calcolando il coefficiente di correlazione

cars['consumo'].corr(cars['cilindrata'])

Dato che è abbastanza vicino ad 1, confermiamo che tra i due attrivuti c’è una relazione di tipo lienare diretto

**I dati a disposizione permettono di valutare l'ipotesi che i valori dell'attributo *consumo* siano compatibili con un modello normale?**

La legge normale e' caratterizzato da una forma campana (la sua funzione di densita' e unimodale, cioe' con un massimo locale). Guardando il grafico delle frequenze osservate per l'attributo consumo (istogramma alpunto 4), possiamo dire che i dati osservati per quel attributo non sono compatibili con una distribuzione normale.

Usando il diagramma Q-Q per l'attributo 'consumo' possiamo approfondire questa analisi, la linea indica la curva attesa in caso di normalita dei dati. Dal grafico otteniamo ulteriore conferma del fatto che la legge normale non sia compatibile con i dati rilevati per l'attributo consumo

cars['consumo'].describe()

**import** statsmodels.api **as** smsm.qqplot(cars['consumo'], fit **= True**, line**=**'45')plt.show()

Immagine che contiene testo

Descrizione generata automaticamente

La modellazione che possiamo usare è una binomiale dove p è il parametro del successo che in questo caso è il fallimento dei freni, una geometrica richiederebbe che i fallimenti siano consecutivi.

1. Per rappresentare ‘etstfreni’ dato che sono dati numerici non molto alti posso calcolare le frequenza assolute e fare un grafico a bastoncini:

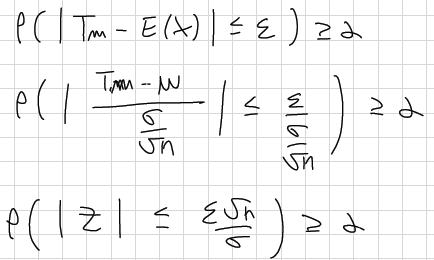
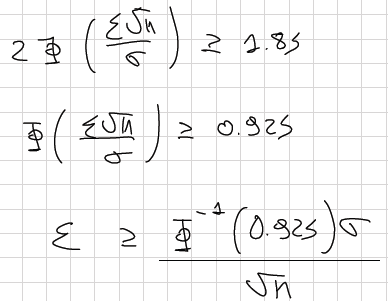
cars['testfreni']testfreni\_freq\_assol **=** pd.crosstab(index **=** cars['testfreni'], columns**=**['Frequenza assoluta'], colnames**=**[''])

plt.vlines(testfreni\_freq\_assol.index, 0, testfreni\_freq\_assol.values)plt.plot(testfreni\_freq\_assol.index, testfreni\_freq\_assol.values, 'o')plt.show()

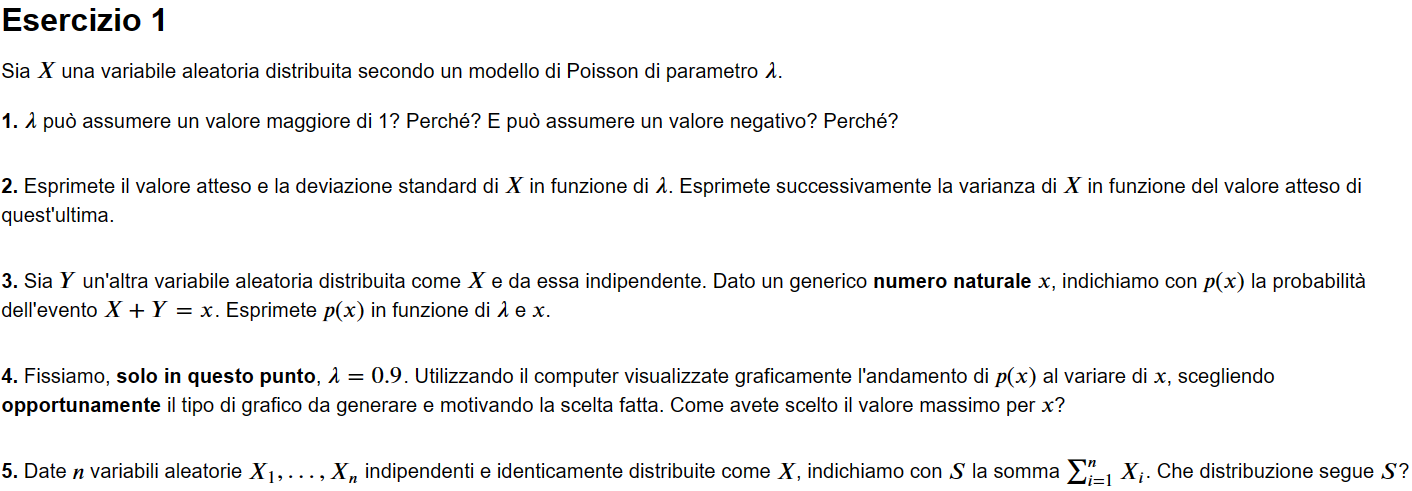
1. Come detto una binomiale potrebbe andare bene
2. La media campionaria è sempre uno stimatore corretto per il valore atteso:

cars['testfreni'].mean()

1. Non so da veder
2. **import** scipy.stats **as** stsigma\_  
   x **=** cars['testfreni'].std()  
   n **=** len(cars['testfreni'].dropna())  
   Z **=** st.norm()  
   alpha **=** 0.85  
   eps **=** (sigma\_x **/** n**\*\***0.5) **\*** Z.ppf(0.925) //inverso di sigma si calcola come il calcolo dei quantili .pff(valore)  
   eps

 🡪 

**Prova di Luglio 2022**



1. Il parametro lambda di una distribuzione di Poisson è sempre >0. Il parametro di una variabile di Poisson deve essere sempre maggiore di zero perché rappresenta il numero atteso di eventi in un intervallo di tempo o in una regione di spazio. Se il parametro fosse negativo, non avrebbe senso parlare di un numero atteso di eventi. Inoltre, la distribuzione di Poisson è definita solo per parametri positivi.
2. Per definizione il valore attedo E(X) di una V.A. di poisson è E(X)= λ, la varianza è definita come VAR(X)= λ e quindi la deviazione standard non è altro che la radice della varianza quindi la radice di λ
3. Dato che la distribuzione di poisson è riproducibile possiamo scrivere X+Y=Z che segue un modello di Poisson (λ1+ λ2), anche se identicamente distribuite unapoissoniana sprime un numero di eventi che si verificano in un intervallo e i parametri potrebbero essere differenti. Quindi per calcolare la probabilità semplicemente scriviamo la funzione di massa di probabilità con λ= λ1+ λ2
4. l = 0.9

Z = st.poisson(2\*l) //ipotizziamo che X e Y abbiano stesso lambda

x = np.arange(0, (Z.ppf(0.999)+1), 1)

plt.vlines(x, 0, Z.pmf(x)) // grafico a bastoncini perché lavoriamo con parametri naturali

plt.plot(x, Z.pmf(x), 'o')

scelgo il valore massimo utilizzando la funzione di ripartizione inversa con un parametro prossimo all'1, in questo modo sono sicuro di includere nel grafico

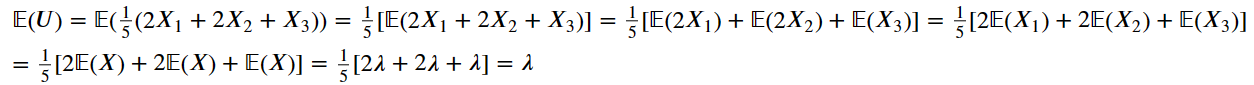
# tutti i punti per cui la funzione di massa di probabilità assume valori significativi (non prossimi allo zero)

1. plt.show() La variabile di Poisson possiede la proprietà di riproducibilità, una somma di variabili aleatorie di Poissoin è sempre una variabile di poisson con parametro uguale alla somma dei parametri

Immagine che contiene tavolo

Descrizione generata automaticamente

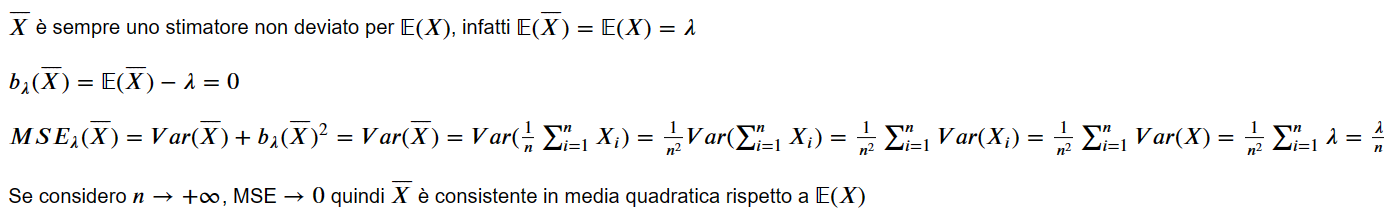
1. Poiché X1, X2 e X3 sono campioni estratti da una popolazione modellata da una variabile aleatoria di Poisson con parametro λ, la loro varianza è uguale al loro valore atteso, ovvero λ.

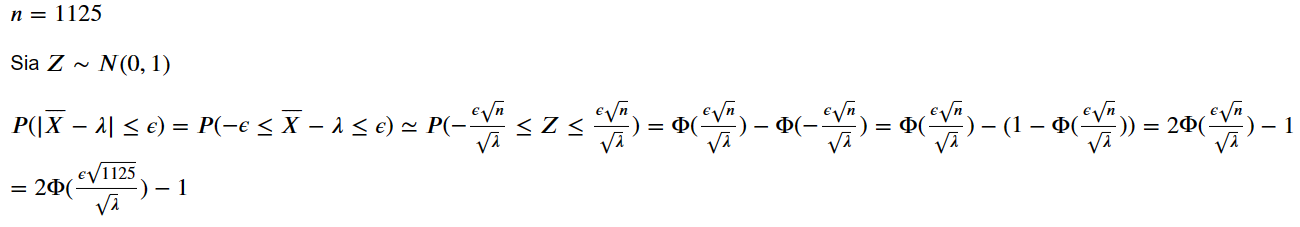
 

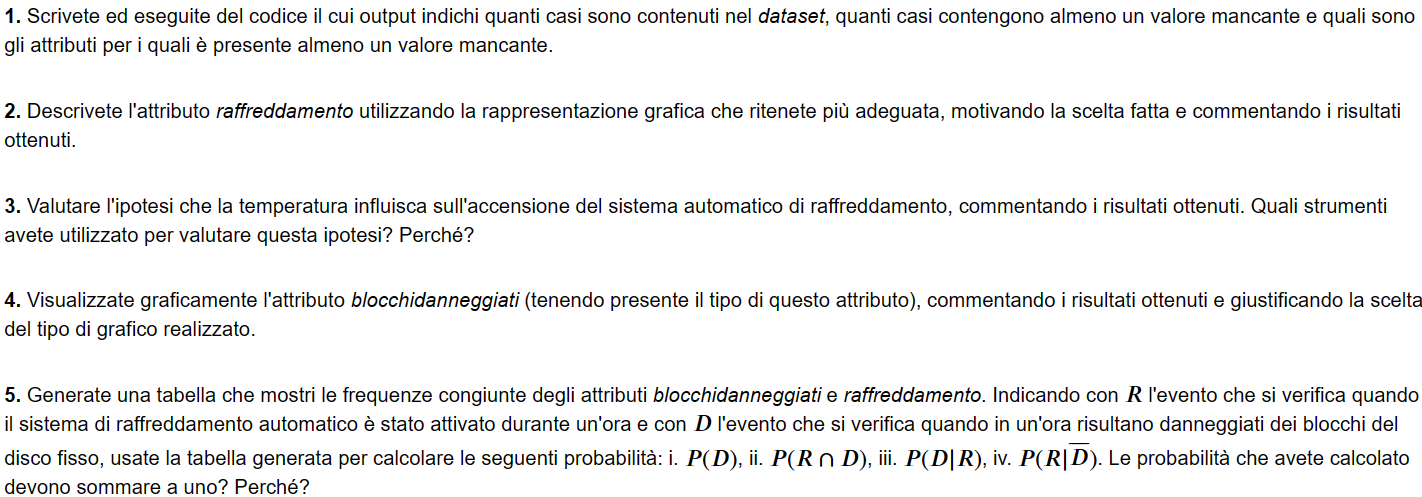
Sono entrambi non distorti, quindi mi basta calcolare la varianza per vere se consistenti quadraticamente. VAR(T1)=3λ e VAR(T2)=9/25 λ non dipendendo da n non posso calcolare il limite e quindi no.

1. Immagine che contiene testo

   Descrizione generata automaticamente
2. Per il teorema centrale del limite ogni variabile aleatoria può essere approssimata ad una normale  
   Immagine che contiene testo

   Descrizione generata automaticamente
3. La media campionaria è sempre uno stimatore corretto e non deviato per il calcolo del valore atteso
4. Nella distribuzione di poisson VAR(X)=E(X) quindi quanto sopra è identico
5. Immagine che contiene testo

   Descrizione generata automaticamente
6. 



1. len(data)  
   data.isnull.any()

sum(pd.isnull(data['raffreddamento']))

1. Potremmo usare grafico a torta dato che è un valore binario ma anche un grafico a barre andava bene

raff\_freq=data['raffreddamento'].value\_counts(normalize=True).sort\_index()

plt.pie(raff\_freq, labels=['senza raffreddamento', 'con raffreddamento'])

plt.show()

data["raffreddamento"].value\_counts().plot.bar()

plt.show()

1. Potremmo dividere il dataset tra chi ha il riscaldamento acceso e chi lo ha spento andando a misurare la media di temperatura

data[data['raffreddamento'] == 0]['temperatura'].mean(), data[data['raffreddamento'] == 1]['temperatura'].mean()

oppure calcolare la correlazione

data["temperatura"].corr(data["raffreddamento"])

1. blocchi\_freq = data['blocchidanneggiati'].value\_counts(normalize=True).sort\_index() # più si considerano numeri elevati di blocchi danneggiati, più le frequenze diminuiscono.

plt.vlines(blocchi\_freq.index, 0, blocchi\_freq.values) # grafico a bastoncini perchè consideriamo valori da 0 a 6 interi quindi ....

plt.plot(blocchi\_freq.index, blocchi\_freq.values, 'o')

plt.show()

1. pd.crosstab(index=data["blocchidanneggiati"], columns=data["raffreddamento"], colnames=[""], margins=True)

pD = (434+200+94+37+8+3) / (1722) # P(D)

pRD = (253+34+5) / (1722) # P(R intersecato D)

pR = 1122/1722

pDdatoR = pRD / pR

pNotD = 1 - (pD) # P(R dato !D)

pRnotD = ((830/1722) / (pNotD))

print("P(R | not(D)): ", pRnotD)

# No non devono valere 1 perchè non sono eventi mutualmente esclusivi e indipendenti

Immagine che contiene testo

Descrizione generata automaticamente

1. raffreddamento\_si = data[data['raffreddamento'] == 1]['blocchidanneggiati']
2. decili=np.arange(0,1.1,0.1) //definisco i valori dei decili, se fossero percentili andrebbero da 0 a 100 con passo 1

raffreddamento\_si.quantile(decili)

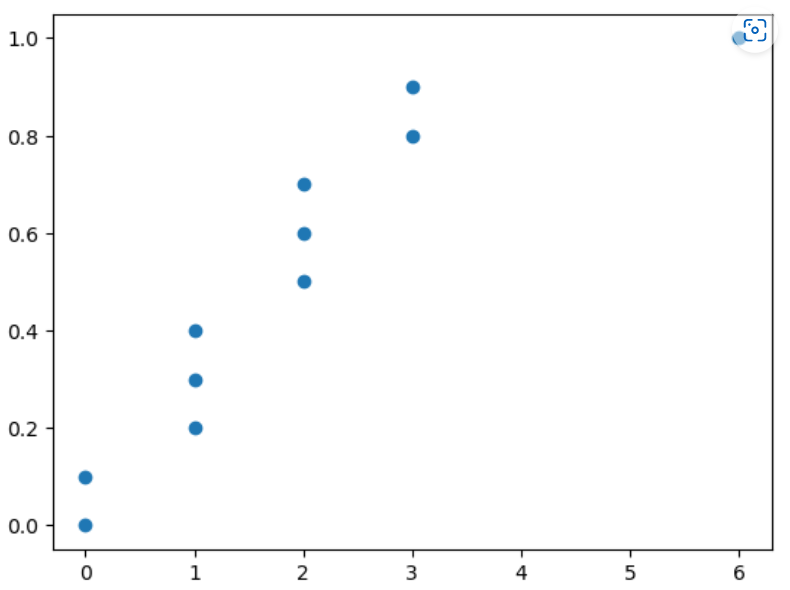
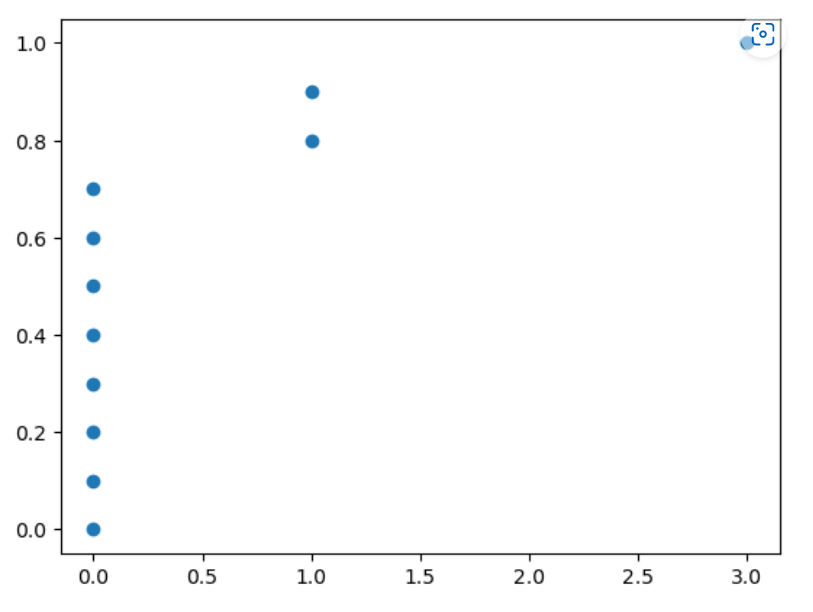
raffreddamento\_no.quantile(decili)

plt.plot(raffreddamento\_si.quantile(decili),decili,'o')#sulla y decili

plt.plot(raffreddamento\_no.quantile(decili),decili,'o')#sulla y decili

Dal grafico si evince che il numero di blocchi danneggiati dipenda dall'accensione del raffredamento, quando questo è spento aumenta il numero di blocchi danneggiati. Confermato anche dalla somma di questi ultimi:  
sum(raffreddamento\_no[raffreddamento\_no!=0].value\_counts())

sum(raffreddamento\_si[raffreddamento\_si!=0].value\_counts()) o dalla media

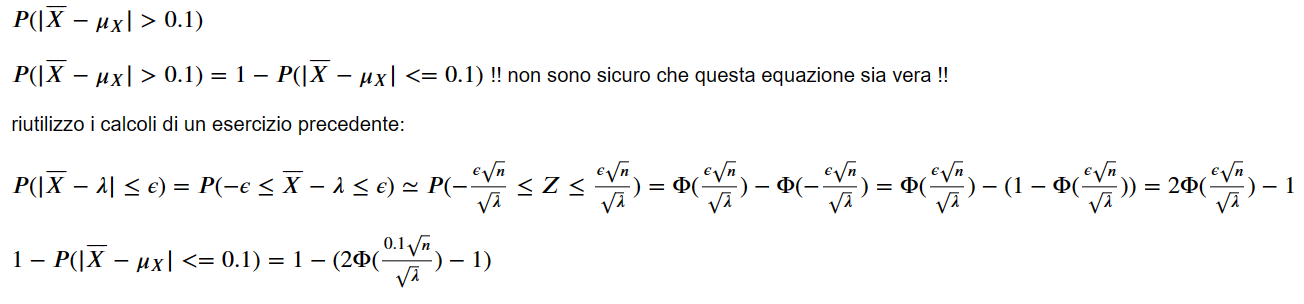
 il primo fa vedere che i punti sono di più sullo zero ed è il grafico con il raffr. acceso

1. La media campionaria è sempre uno stimatore non distorto per il valore atteso: raffreddamento\_si.mean()
2. Posso innanzitutto verificare che varianza e media siano uguali in modo da non escludere che sia esponenziale e poi provo a tracciare il QQ plot, per il parametro delle esponenziali posso usare la media campionaria del mio campione per capire se la distribuzione si comporta in quel modo:  
   raffreddamento\_si.mean(),raffreddamento\_si.var()  
   sm.qqplot(raffreddamento\_si,dist=st.poisson(raffreddamento\_si.mean()), line='45')

plt.show() //per raffreddamento si i punti si allineano quasi sulla bisettrice quindi possiamo non escludere l’ipotesi

e lo stessso per

1. Dato che abbiamo verificato che la distribuzione è assimilabile ad una distribuzione di poisson, Per il parametro λ posso usare sempre la media campionaria ed è sempre non distorto, e sempre consistente in media quadratica.  
   Per la varianza posso usare sempre l a media quadratica che per la poisson è sempre uguale al valore atteso e quindi sempre non deviato e consistente, invece per la deviazione standard è semplicemente la radice della varianza



Z = st.norm()

n = len(raffreddamento\_si)\*\*.5

l = raffreddamento\_si.mean()\*\*.5

1 - (2\*Z.cdf((0.1 \* n) / l) - 1)

1. La probabilità che il riscaldamento sia acceso la potremmo calcolare come la probabilità dei casi favorevoli fratto i casi possibili

sum(data['raffreddamento'] == 1)/len(data.dropna())

1. l\_si = raffreddamento\_si.mean()

l\_no = raffreddamento\_no.mean()

X\_SI = st.poisson(l\_si)

X\_NO = st.poisson(l\_no)

1. X\_SI.cdf(4) # P(X\_SI > 4) = 1 - P(X\_SI <= 4)
2. X\_NO.cdf(4) # P(X\_NO > 4) = 1 - P(X\_NO <= 4)
3. p = ((1 -X\_SI.cdf(4))\*(sum(data['raffreddamento']==1)/len(data.dropna()))) + ((1 - X\_NO.cdf(4)) \* (sum(data['raffreddamento'] == 0)/len(data.dropna())))
4. # se supponiamo che l'allarme possa scattare una sola volta al giorno allora questo ha senso

R = st.binom(365, p)

p2 = 1 - R.cdf(7)

B = st.binom(7500, p2)

B.mean()

**Prova di Gennaio 2022**

**1)Siano X e Y due variabili aleatorie indipendenti e distribuite secondo un modello esponenziale rispettivamente di valore attes mux e muy. Esprimete il parametro, la varianza e la deviazione standard di X in funzione di mux**

Siccome X e Y sono varibaili esponenziali con parametro λx e λy rispettivamente possiamo calcolare λx come λx=1/E(X)=1/mux, VAR(X)=1/ (λx)^2=mux^2 e la deviazione standard è la radice della varianza. Lo stesso vale per Y.

**Sia Z=X-Y. Quali valori può assuemre tale V.A?**

Siccome le variabili esponenziali sono in R+ la differenza di due valori positivi può dare origine a valori negativi quindi il dominio è in R

**Z segue una distribuzione che abbiamo studiato?**

No, non segue una distribuzione che abbiamo visto, la differenza tra varibili aleatorie che abbiamo visto era solo sulla normale. Inoltre la differenza di due esponenziali non è detto che rimanga una esponenziale dato che si riferiscono alla probabilità di osservare un numero atteso di eventi.

**Esprimete il valore atteso e la varianza di Z in funzione di mux e muy**

E(Z)=E(X)-E(Y) per linearità e ottengo E(Z)=1/ λx-1/ λy e la varianza VAR(Z)=VAR(X)-VAR(Y)=VAR(X)+VAR(-Y)=VAR(X)+VAR(Y)=mux^2+muy^2.

**Indicando con E(Z)=muz il e dato un generico ε>0, sia P(|Z-muz|> ε), usando chebyshev troviamo una maggiorazione che dipenda da muz e muy**

Per il teorema di chebyshev P(|Z-muz|> ε) è sempre <= VAR(Z)/ (ε)^2 che è uguale a (mux)^2+(muy^2)/ (ε)^2

**2) Per n numero naturale fissato, siano rispettivamente X1,…Xn e Y1…Yn due campioni aleatori tra loro indipendenti ed estratti da due popolazioni, rispettivamente distribuite come varibaili X e Y del precedente esercizio.**

**Lo stimatore S=(X1+Y1)/2 è uno stimatore non deviat per muz=mux-muy? E invece T=(X1+Y1+X2+Y2)/2 ?**

Sono entrambi distorti perché: E((X1+Y1)/2)=1/2\*(E(X1)+E(Y1)=(mux+muy)/2

E((X1+Y1+X2+Y2)/2)=1/2\*(E(X1)+E(Y1)+E(X2)+E(Y2))=(2mux+2muy)/2= mux+muy

**Concentriamoci ora sulla variabile aleatoria**  **indicano rispettivamente le media campionarie dei due campioni che stiamo considerando**

**Questa nuova variabile aleatoria è distribuita secondo uno dei modelli che abbiamo studiato? Esiste un modello che approssima in modo ragionevole al distribuzione?**

No, non sono definite con uno dei modelli che abbiamo studiato. Per TCL ogni distribuzione per n abbastanza grande può essere approssimata ad una normale di parametri, mediaX avrà parametri di valore atteso uguale a quello della distribuzione originale quindi mux e deviazione standard uguale alla deviazione standard della distribuzione originale/ e cosi anche per mediaY, la loro differenza saà come per X-Y. Ovvero con valore atteso uguale alla differenza dei valori attesi e come deviazione la somma della varianze sotto radice

Perche VAR(mediaX)=VAR(X)/n e la deviazione standard è la radice di questo abbiamo che

 è distribuita secondo un modello normale che ha valore atteso mux-muy e sigma=rad((mux^2+muy^2)/n).

**È uno stimatore corretto per muz? consistente?**

Si perché il suo valore atteso è esattamente la definizione di muz. Dato che è non distorto basta calcolare la varianza di entrambi e sottrarla, ovvero sommarla e otteniamo il limite di VAR(X)/n + VAR(Y)/n che tende a zero quindi è consistente in media quadratica. Con VAR(X) e VAR(Y) uguali a quelle del campione originale che è mux^2

**È un buon stimatore per VAR(Z)?**

No perché la varianza di Z è stat calcolata come mux^2 + muy^2 che è diversa da mux^2 /n + muy^2/n

**Esprimete in funzione di n, mux, muy il più piccolo valore ε per il quale si può garantire con probabilità 0.9 l’errore che si compie usando**  **per stimare muz sia minore o uguale a ε**

**Prova di Febbraio 2020**

**Sia X una variabile casuale che segue una legge bernoulliana di parametro p.**

**Quali valori può assumere X?**

Una variabile aleatoria bernoulliana può assumere solo due valori X=1 che indica il successo e X=0 che indica il fallimento

**Quali valori può assumere il parametro p?**

p può assumere i valori compresi tra 0 e 1 estremi inclusi

**Quale è il valore atteso di X? E la sua varianza?**

E(X)=p

VAR(X)= p\*(1-p)

**Grafico del valore atteso al variare di p e di VAR(X)**

Dato che il valore atteso di X è p sarà una retta diagonale (bisettrice), invece la varianza è una parabola tra 0 ed 1 con il max ¼ (?)

**Quali valori può assumere il valore atteso e la varianza?**

Il valore atteso essendo uguale a p può assumere valori tra 0 ed 1, la varianza tra 0 e ½

**Sia**  **media campionaria di un campione di una popolazione bernoulliana di parametro p.  
Esprimete il valore atteso e la varianza della media campionaria in funzione di p**

Il valore atteso della media è il valore atteso di X e quindi uguale a p. La varianza della media è la varianza del campione originale7numerosità del campione:  
EImmagine che contiene testo

Descrizione generata automaticamente Immagine che contiene testo

Descrizione generata automaticamente

**Verificare che** Immagine che contiene testo, orologio, calibro

Descrizione generata automaticamente

Nel punto massimo VAR(X) è uguale a ¼ dato che p=1/2 quindi VAR(X)/n non è altro che 1/n\*1/4 al suo massimo quindi il valore sarà sempre minore o uguale a 1/4n

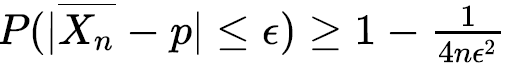
**Controllare che per ogni la seguente disuguaglianza è valida:** 

Immagine che contiene testo, tavolo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente **è uno stimatore di p non distorto?**

Calcolando il valore atteso della mia statistica devo ottenere esattamente quello che voglio stimare, e per la linearità del valore atteso ottengo:  
 //quel 2 è un n

**Esprimere 1-p in funzione di E(X)**

Siccome E(X)=p allora 1-p=1-E(X), dato che P(X=0)=1-p e E(X)=p ottengo 1-p=1-E(X)

**Determinare uno stimatore S del parametro 1-p e verificare sche sia non distorto**

Per stimare p la media campionaria è un buon stimatore, quindi per stimare 1-p possiamo usare 1-mediacampionaria

È non distorto, infatti:   
 //E di una costante è la costante

**Caricare i dati dove il carattere |t separa le colonne e i numeri decimali separati da ,**

car=pd.read\_csv("mtcars.txt",sep="\t",decimal=",")

**Calcolare la numerosità del campione, tracciare il boxplot del carattere della cilindrata e trovare gli outlier rispetto alla cilindrata**

len(car.dropna()) //dropna per togliere i null

plt.boxplot(car['cilindrata']) //visualizzo gli outlier

rslt\_df = car[(car['cilindrata'] <300)] //tagli gli outlier

plt.boxplot(rslt\_df['cilindrata']) //outlier spariti

rslt\_df = car[(car['cilindrata'] >300)] //prendo gli otulier

rslt\_df.modello //stampo il risultato

**Calcolare i quartili della cilindrata e la distanza interquartile**

**13 – Operazioni tra VA (da prendere con le pinze, generati con ChatGPT)**

**Modello Bernoulliano**

**Somma:** Se X e Y sono due variabili aleatorie Bernoulli indipendenti con probabilità di successo p\_x e p\_y, allora la loro somma Z = X + Y è una variabile aleatoria binomiale con parametri n = 2 e p = p\_x + p\_y - p\_x \* p\_y. Il valore atteso di Z è dato da E(Z) = E(X + Y) = E(X) + E(Y) = 2p\_x + 2p\_y - 2p\_x \* p\_y e la varianza è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = 2p\_x + 2p\_y - 2p\_x \* p\_y.

**Differenza:** La sottrazione di due variabili aleatorie Bernoulli non è necessariamente una variabile aleatoria Bernoulli. La distribuzione di Z = X - Y dipende dalla correlazione tra X e Y. In generale, il valore atteso e la varianza di Z non sono semplicemente determinati da p\_x e p\_y.

**Moltiplicazione:** Se X e Y sono due variabili aleatorie Bernoulli indipendenti con probabilità di successo p\_x e p\_y, allora il loro prodotto Z = X \* Y è ancora una variabile aleatoria Bernoulli con probabilità di successo p\_z = p\_x \* p\_y. Il valore atteso di Z è dato da E(Z) = E(X \* Y) = p\_x \* p\_y e la varianza è data da Var(Z) = Var(X \* Y) = p\_x \* p\_y \* (1 - p\_x) \* (1 - p\_y)

**Divisione:** La divisione di due variabili aleatorie Bernoulli non è necessariamente una variabile aleatoria Bernoulli. La distribuzione di Z = X / Y non può essere definita quando Y = 0. Inoltre, anche quando Y ≠ 0, la distribuzione di Z dipende dalla correlazione tra X e Y. Il valore atteso e la varianza di Z non sono semplicemente determinati da p\_x e p\_y

**Modello binomiale**

**Somma:** Se X e Y sono due variabili aleatorie binomiali indipendenti con parametri n\_x e p\_x e n\_y e p\_y, allora la loro somma Z = X + Y è una variabile aleatoria binomiale con parametri n = n\_x + n\_y e p = p\_x + p\_y - p\_x \* p\_y. Il valore atteso di Z è dato da E(Z) = E(X + Y) = E(X) + E(Y) = n\_x \* p\_x + n\_y \* p\_y e la varianza è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = n\_x \* p\_x \* (1 - p\_x) + n\_y \* p\_y \* (1 - p\_y).

**Differenza**: La differenza di due variabili aleatorie binomiali non è necessariamente una variabile aleatoria binomiale. La distribuzione di Z = X - Y dipende dalla correlazione tra X e Y e potrebbe non essere definita se X < Y. In generale, il valore atteso e la varianza di Z non sono semplicemente determinati da n\_x, p\_x, n\_y e p\_y

**Moltiplicazione:** La moltiplicazione di due variabili aleatorie binomiali non è una variabile aleatoria binomiale. La distribuzione di Z = X \* Y è più complessa di una distribuzione binomiale e dipende da n\_x, p\_x, n\_y e p\_y. Il valore atteso e la varianza di Z non sono semplicemente determinati da n\_x, p\_x, n\_y e p\_y

**Divisione:** La divisione di due variabili aleatorie binomiali non è una variabile aleatoria binomiale. La distribuzione di Z = X / Y non può essere definita quando Y = 0. Inoltre, anche quando Y ≠ 0, la distribuzione di Z dipende dalla correlazione tra X e Y e potrebbe essere complessa. Il valore atteso e la varianza di Z non sono semplicemente determinati da n\_x, p\_x, n\_y e p\_y.

**Modello Uniforme discreto**

**Somma:** Se X e Y sono due variabili aleatorie uniformi discrete indipendenti con range [a, b] e [c, d], rispettivamente, allora la loro somma Z = X + Y è una variabile aleatoria discreta con probabilità p(z) = p\_x \* p\_y, dove p\_x è la probabilità che X = z - Y e p\_y è la probabilità che Y = z - X. Il valore atteso di Z è dato da E(Z) = E(X + Y) = E(X) + E(Y) = (a + b + c + d) / 2 e la varianza è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = ((b - a + 1)^2 + (d - c + 1)^2 - 2) / 12

**Differenza:** Se X e Y sono due variabili aleatorie uniformi discrete indipendenti con range [a, b] e [c, d], rispettivamente, allora la loro differenza Z = X - Y è una variabile aleatoria discreta con probabilità p(z) = p\_x \* p\_y, dove p\_x è la probabilità che X = z + Y e p\_y è la probabilità che Y = z + X. Il valore atteso di Z è dato da E(Z) = E(X - Y) = E(X) - E(Y) = (b - a + 1 + d - c + 1) / 2 e la varianza è data da Var(Z) = Var(X - Y) = Var(X) + Var(Y) = ((b - a + 1)^2 + (d - c + 1)^2 - 2) / 12.

**Moltiplicazione:** La moltiplicazione di due variabili aleatorie uniformi discrete non è una variabile aleatoria uniforme discreta. La distribuzione di Z = X \* Y dipende dalle correlazioni tra X e Y e potrebbe essere complessa. Il valore atteso e la varianza di Z non sono semplicemente determinati da a, b, c e d.

**Divisione:** La divisione di due variabili aleatorie uniformi discrete non è generalmente una variabile aleatoria uniforme discreta. La distribuzione di Z = X / Y dipende dalle correlazioni tra X e Y e potrebbe essere complessa o non esistere (ad esempio se Y può assumere il valore 0). Il valore atteso e la varianza di Z non sono semplicemente determinati da a, b, c e d.

**Modello geometrico**

**Somma:** Se X e Y sono due variabili aleatorie geometriche indipendenti con parametri p\_x e p\_y, allora la loro somma Z = X + Y è una variabile aleatoria negativa binomiale con parametro p = p\_x \* p\_y. Il valore atteso di Z è dato da E(Z) = E(X + Y) = E(X) + E(Y) = (1 / p\_x) + (1 / p\_y) e la varianza è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = (1 - p\_x) / (p\_x^2) + (1 - p\_y) / (p\_y^2).

**Differenza:** La differenza di due variabili aleatorie geometriche non è generalmente una variabile aleatoria geometrica. La distribuzione di Z = X - Y dipende dalle correlazioni tra X e Y e potrebbe essere complessa. Il valore atteso e la varianza di Z non sono semplicemente determinati da p\_x e p\_y

**Moltiplicazione:** La moltiplicazione di due variabili aleatorie geometriche non è una variabile aleatoria geometrica. La distribuzione di Z = X \* Y dipende dalle correlazioni tra X e Y e potrebbe essere complessa. Il valore atteso e la varianza di Z non sono semplicemente determinati da p\_x e p\_y.

**Divisione:** La divisione di due variabili aleatorie geometriche non è una variabile aleatoria geometrica. La distribuzione di Z = X / Y dipende dalle correlazioni tra X e Y e potrebbe essere complessa o non esistere (ad esempio se Y può assumere il valore 0). Il valore atteso e la varianza di Z non sono semplicemente determinati da p\_x e p\_y.

**Modello di Poisson**

**Somma:** Se X e Y sono due variabili aleatorie di Poisson indipendenti con parametri λ\_x e λ\_y, allora la loro somma Z = X + Y è anche una variabile aleatoria di Poisson con parametro λ = λ\_x + λ\_y. Il valore atteso di Z è dato da E(Z) = E(X + Y) = E(X) + E(Y) = λ\_x + λ\_y e la varianza è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = λ\_x + λ\_y.

**Differenza:** La differenza di due variabili aleatorie di Poisson può non essere una variabile aleatoria di Poisson, a meno che λ\_x ≥ λ\_y. In questo caso, Z = X - Y è anche una variabile aleatoria di Poisson con parametro λ = λ\_x - λ\_y. Il valore atteso di Z è dato da E(Z) = E(X - Y) = E(X) - E(Y) = λ\_x - λ\_y e la varianza è data da Var(Z) = Var(X - Y) = Var(X) + Var(Y) = λ\_x + λ\_y.

**Moltiplicazione:** La moltiplicazione di due variabili aleatorie di Poisson non è una variabile aleatoria di Poisson. Il valore atteso e la varianza di Z = X \* Y, dove X e Y sono variabili aleatorie di Poisson con parametri λ\_x e λ\_y, rispettivamente, possono essere calcolati utilizzando le proprietà delle variabili aleatorie. Il valore atteso è dato da E(Z) = E(X \* Y) = E(X) \* E(Y) = λ\_x \* λ\_y e la varianza è data da Var(Z) = Var(X \* Y) = E(X \* Y)^2 - [E(X \* Y)]^2 = λ\_x \* λ\_y \* (λ\_x + λ\_y).

**Divisione:** La divisione di due variabili aleatorie di Poisson non è una variabile aleatoria di Poisson. Il valore atteso e la varianza di Z = X / Y, dove X e Y sono variabili aleatorie di Poisson con parametri λ\_x e λ\_y, rispettivamente, non possono essere calcolati utilizzando le proprietà delle variabili aleatorie, poiché la divisione non è un'operazione definita per le variabili aleatorie di Poisson. Tuttavia, si potrebbero calcolare stime approssimative utilizzando altre tecniche statistiche.

**Modello ipergeometrico**

**Somma:** La somma di due variabili aleatorie ipergeometriche non è una variabile aleatoria ipergeometrica. Il valore atteso e la varianza di Z = X + Y, dove X e Y sono variabili aleatorie ipergeometriche con parametri N, K e n, rispettivamente, possono essere calcolati utilizzando le proprietà delle variabili aleatorie.

Il valore atteso di Z è dato da:E(Z) = E(X) + E(Y) = n \* K / N + n \* K / N = 2 \* n \* K / N

La varianza di Z è data da:Var(Z) = Var(X) + Var(Y) = n \* K \* (N - K) / (N^2 \* (N - 1)) + n \* K \* (N - K) / (N^2 \* (N - 1)) = 2 \* n \* K \* (N - K) / (N^2 \* (N - 1))

Si noti che l'ipergeometrica non è associativa e quindi la somma e la differenza non sono distribuzioni ipergeometriche.

**Differenza**: La differenza di due variabili aleatorie ipergeometriche non è una variabile aleatoria ipergeometrica. Il valore atteso e la varianza di Z = X - Y, dove X e Y sono variabili aleatorie ipergeometriche con parametri N, K e n, rispettivamente, possono essere calcolati utilizzando le proprietà delle variabili aleatorie.

Il valore atteso di Z è dato da:E(Z) = E(X) - E(Y) = n \* K / N - n \* K / N = 0

La varianza di Z è data da:Var(Z) = Var(X) + Var(Y) = n \* K \* (N - K) / (N^2 \* (N - 1)) + n \* K \* (N - K) / (N^2 \* (N - 1)) = 2 \* n \* K \* (N - K) / (N^2 \* (N - 1))

**Moltiplicazione:** La moltiplicazione di due variabili aleatorie ipergeometriche non è una variabile aleatoria ipergeometrica. Il valore atteso e la varianza di Z = X \* Y, dove X e Y sono variabili aleatorie ipergeometriche con parametri N, K e n, rispettivamente, non possono essere facilmente calcolati in quanto la distribuzione di Z potrebbe non essere ben definita o avere proprietà statistiche complesse. In generale, la moltiplicazione di due variabili aleatorie non conduce a una distribuzione nota o facilmente analizzabile.

**Divisione:** La divisione di due variabili aleatorie ipergeometriche non è una variabile aleatoria ipergeometrica. Il valore atteso e la varianza di Z = X / Y, dove X e Y sono variabili aleatorie ipergeometriche con parametri N, K e n, rispettivamente, non possono essere facilmente calcolati in quanto la distribuzione di Z potrebbe non essere ben definita o avere proprietà statistiche complesse. In generale, la divisione di due variabili aleatorie non conduce a una distribuzione nota o facilmente analizzabile.

**Modello Uniforme continuo**

**Somma:** Se X e Y sono due variabili aleatorie uniformi continue con medesimo supporto e medesima funzione di densità di probabilità, allora la somma Z = X + Y è anche una variabile aleatoria uniforme continua con supporto esteso di [a + b, c + d], dove [a, b] e [c, d] sono i supporti di X e Y rispettivamente. Il valore atteso di Z è dato da E(Z) = E(X) + E(Y) e la varianza di Z è data da Var(Z) = Var(X) + Var(Y).

**Differenza:** Se X e Y sono due variabili aleatorie uniformi continue con medesimo supporto e medesima funzione di densità di probabilità, allora la differenza Z = X - Y è anche una variabile aleatoria uniforme continua con supporto esteso di [a - d, b - c], dove [a, b] e [c, d] sono i supporti di X e Y rispettivamente. Il valore atteso di Z è dato da E(Z) = E(X) - E(Y) e la varianza di Z è data da Var(Z) = Var(X) + Var(Y).

**Moltiplicazione:** La moltiplicazione di due variabili uniformi continue non produce in generale una distribuzione uniforme continua.

**Divisione:** La divisione di due variabili uniformi continue non produce in generale una distribuzione uniforme continua.

**Modello esponenziale**

**Somma:** La somma di due variabili aleatorie esponenziali indipendenti è ancora una variabile aleatoria esponenziale con rate λ\_z = λ\_x + λ\_y e La varianza di Z = X + Y è data da Var(Z) = Var(X + Y) = Var(X) + Var(Y) = 1 / λ\_x^2 + 1 / λ\_y^2.

**Differenza:** La differenza di due variabili aleatorie esponenziali non è necessariamente una variabile aleatoria esponenziale.

Se X e Y sono due variabili aleatorie esponenziali indipendenti con rate λ\_x e λ\_y, allora la loro differenza Z = X - Y ha una distribuzione di Erlang con due parametri e una rate λ\_z = λ\_x + λ\_y. Il valore atteso di Z è E(Z) = 1 / λ\_z e la varianza è Var(Z) = 1 / λ\_z^2. l valore atteso di Z è dato da E(Z) = E(X - Y) = E(X) - E(Y) = 1 / λ\_x - 1 / λ\_y. La varianza di Z è data da Var(Z) = Var(X - Y) = Var(X) + Var(Y) = 1 / λ\_x^2 + 1 / λ\_y^2.

**Moltiplicazione:** Il prodotto di due variabili aleatorie esponenziali non è necessariamente una variabile aleatoria esponenziale.

**Divisione:** La divisione di due variabili aleatorie esponenziali non è necessariamente una variabile aleatoria esponenziale

**Modello normale**

**Somma:** La somma di due variabili normali è una variabile normale con lo stesso valore atteso di somma delle variabili originali e varianza pari alla somma delle varianze delle variabili originali.

**Differenza:** La differenza di due variabili normali è ancora una variabile normale con valore atteso pari alla differenza dei valori attesi delle variabili originali e varianza pari alla somma delle varianze delle variabili originali.

**Moltiplicazione:** La moltiplicazione di due variabili normali non è generalmente una variabile normale. Tuttavia, se le due variabili sono indipendenti e hanno medie zero e stesse varianze, allora la loro moltiplicazione segue una distribuzione di tipo Gauss-Lévy.

**Divisione:** Se X e Y sono due variabili normali indipendenti con valori attesi E(X) e E(Y), e devianze standard σ\_X e σ\_Y, la divisione Z = X / Y non sarà in generale una variabile normale.

Il valore atteso di Z non può essere calcolato direttamente dai valori attesi di X e Y. Può essere approssimato utilizzando tecniche come la trasformazione delta method, ma questo dipende dalla precisione richiesta e dalla distribuzione di Y.

La varianza di Z non può essere calcolata direttamente dalle devianze standard di X e Y, ma può essere approssimata utilizzando tecniche simili.

**Modello normale Standard**

**Somma**: Se X e Y sono due variabili normali standard indipendenti, allora Z = X + Y è anche una variabile normale con valore atteso E(Z) = E(X) + E(Y) = 0 + 0 = 0 e varianza Var(Z) = Var(X) + Var(Y) = 1 + 1 = 2. La distribuzione di Z sarà quindi una distribuzione normale con media 0 e varianza 2.

**Differenza:** Se X e Y sono due variabili normali standard indipendenti, la loro differenza Z=X-Y è ancora una variabile normale standard con valore atteso E(Z) = E(X) - E(Y) = 0 - 0 = 0 e varianza Var(Z) = Var(X) + Var(Y) = 1 + 1 = 2.

**Moltiplicazione:** Se X e Y sono due variabili normali standard indipendenti, allora il prodotto Z=X\*Y non sarà una variabile normale standard. Questo perché il prodotto di due variabili normali non sarà in generale una variabile normale. Tuttavia, Z avrà una distribuzione che può essere descritta come la distribuzione di Rice.

**Divisione:** Se X e Y sono due variabili normali standard indipendenti, la loro divisione Z=X/Y ha una distribuzione t studentiana con un numero infinito di gradi di libertà. La sua funzione densità di probabilità è data da:

f(z) = (1 + z^2)^(-1/2) / (π^(1/2))

Il valore atteso di Z è undefined, mentre la varianza di Z è infinita.