# PRE:

## Fare cose nei dataset con **n>1** ma p=1 (una sola variabile, più campioni con n osservazioni), il dataset è mxn

-fai un **dot plot** al posto del plot della time series, del runs test ecc (**n>1**, p=1) :

mf.dotplot(*data*) #PRIMA droppa eventuali colonne che non c’entrano nulla se no #interferisce

-fai uno **shapiro (n>1)**

mf.shapiroqq(*data*,0.05,multiobservation='yes') #se abbiamo più osservazioni (n>1) chiaramente dobbiamo mettere ‘yes’

-fai un **istogramma** **stacked** **data** **(n>1)**

mf.histandbox(*data*,'yes')

-calcolare cose utili nei dataset con **n>1**:

*data\_XS*['sample\_mean'] = *data*.mean(*axis*=1) #vector

*Grand\_mean*=*data\_XR*['sample\_mean'].mean() #scalar

*R\_mean* = *data\_XR*['sample\_range'].mean() #scalar (per stimare sigma in Rcc)

*data\_XS*['sample\_std'] = *data*.std(*axis*=1) #vector

*R\_mean* = *data\_XS*['sample\_std']. mean()#scalar (per stimare sigma in Scc)

*data\_XR*['sample\_range'] = *data*.max(*axis*=1) - *data*.min(*axis*=1) #vector

*data*['MR'] = *data*['X'].diff().abs() #vector dove il primo è NaN

-Crea **dataframe media** con una sola colonna che è la media dei samples del dataframe chiamato data:

*data\_mean* = pd.DataFrame({'sample\_mean': *data*.mean(*axis*=1)})

-Se i dati non sono normali (verificato con shapiroqq) dobbiamo fare quello che qui ho sparato in una funzione unica: **stack data** + **boxcox** **transformation** + plot distr + **shapiro** (checcka il risultato!!!) + **unstack** **data** e poi è possibile fare un xbarR control chart sui dati trasformati.

[*data\_BC\_unstack*,*lmbda*]*=*mf.boxcox\_and\_unstack(*data*,*approx\_to\_0\_if\_nec*='False')

*data\_BC\_XR* = qda.ControlCharts.XbarR(*data\_BC\_unstack*) #giusto fare XbarR o XbarS #perchè *data\_BC\_unstack* è nella forma di un dataframe con **n>1**

### phase2

-**estrai** **OOC estrarre OOC**: (se gli OOC sono sia in un chart che nell’altro, eg. Sia Xbar che R, devo farlo due volte mettendo i nomi delle colonne rispettive dei test) per la **phase1**

*OOC\_indx*=mf.extractOOCindx(*df\_CC*,*nomecolonna\_test*)

-**phase2 concat**: se per la **phase2** abbiamo un dataset e posso usare un qda.control\_chart (quindi specificando ***subset\_size***) allora devo **concat**enare e poi fare il CC:

*phase1\_size* = len(*data\_p1*) *#vogliamo tenercela perchè nella funzione dobbiamo specificare il numero*

*#di sample usati per il design del control chart nel subset\_size parameter della funzione*

*data\_all* = pd.concat([*data\_ph1*, *data\_p2*], *ignore\_index*=True)

*data\_XS* = qda.ControlCharts.XbarS(*data\_all*, *sigma* = *stdev*, *subset\_size* = *phase1\_size*)

-fare check OOC **crisi** (esempio da **Trend CC** dove i miei dati di phase 2 devono essere trasformati):

*#Utilizzando tra virgolette il Trend CC diciamo posso fare il controllo anche così:*

*#costruiamo colonne UCL e LCL*

*#recupera differenza tra CL e LCL o CL e UCL che è* ***costante,*** *dall'IMR df risultante dal Trend CC, qui l’ho chiamato span*

*df\_ph2* = pd.DataFrame({'ones': np.ones(len(*data\_norm\_ph2*))}) #regressore cost

*df\_ph2*['t']=*data\_ph2*['t'] #regressore temporale

*span*=1.239216

*#data\_norm\_ph2= stats.boxcox(data\_ph2['th'], lmbda=0)*

*dati\_UCL*=*model*.predict(*df\_ph2*)+*span*

*dati\_LCL*=*model*.predict(*df\_ph2*)-*span*

mf.checkOOC(*data\_norm\_ph2*,*dati\_LCL*,*dati\_UCL*) #funzione che checcka se gli elementi #di *data\_norm\_ph2* sono compresi tra i corrispondenti *dati\_LCL* e tra *dati\_UCL*

## boxcox a mano:

#esempio dove trasformo gli MR

*MR* = pd.DataFrame(*data\_IMR*['MR'])

*MR*['MR'] = *MR*['MR'].transform(**lambda** *x*: (*x*\*\*0.4))

----------------------------------------------------------

#esempio dove trasformo una colonna di dati qualsiasi (n=1):

data['X\_bc']=data['X'].transform(**lambda** *x*: (((*x*\*\*lmbda)-1)/lmbda))

*#oppure:*

***transformed\_X =stats.boxcox(data['X'], lmbda=lambda\_value)***

------------------------------------------------------------------------------------------------------------------

#NOTA se invece ho dataset con n>1, sono in phase2 e devo trasformare tutte le colonne dei miei dati di phase2 *data2* esattamente con la stessa trasformazione fatta per i dati di phase1:

*Data2\_BxC*=*data2*.transform(**lambda** *x*: (((*x*\*\*lmbda)-1)/lmbda))

## Beta ecc

-**Beta Xbar CC** (solo **shift della mean**) che non è altro che la **Potenza di uno z-test bilaterale**:

*beta*=stats.*norm*.cdf(*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(*n*))-stats.*norm*.cdf(-*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(*n*))

*print(beta)*

#nota: per IMR basterebbe mettere n=1

#esempio in cui devo fare due ARL plots (per n=4 e per n=8) e devo farmi stampare i due ARL per delta\_piccolo=1 e per delta\_piccolo=2

*delta\_piccolo*=np.arange(0,2.1,0.1)

*beta4*=stats.*norm*.cdf(*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(4))-stats.*norm*.cdf(-*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(4))

*beta8*=stats.*norm*.cdf(*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(8))-stats.*norm*.cdf(-*k\_alpha*-*delta\_piccolo*\*np.*sqrt*(8))

print(*beta4*)

*ARL4*=1/(1-*beta4*)

print(*beta8*)

*ARL8*=1/(1-*beta8*)

plt.plot(*delta\_piccolo*,*ARL8*, *color*='g')

plt.plot(*delta\_piccolo*,*ARL4*,*color*='r')

plt.grid()

plt.show()

print(*ARL4*[10])#delta\_piccolo=1

print(*ARL4*[20])#delta\_piccolo=2

print(*ARL8*[10])

print(*ARL8*[20])

-**Beta** **Xbar CC** (**shift della mean** + **change del sigma**), con **lmbda=sigma1/sigma0**

*beta\_XbarCC*=stats.*norm*.cdf(1/*lmbda*\*(*K*-*delta\_piccolo*\*np.*sqrt*(*n*)))-stats.*norm*.cdf(1/*lmbda*\*(-*K*-*delta\_piccolo*\*np.*sqrt*(*n*)))

-**Alpha** e **significatività**(1-alpha) EFFETTIVI **dell’S CC tradizionale**:

*sign*=stats.*chi2*.cdf(((*S\_UCL*\*\*2)\*(*n*-1))/*std*\*\*2,*n*-1)-stats.*chi2*.cdf(((*S\_LCL*\*\*2)\*(*n*-1))/*std*\*\*2,*n*-1)

*alpha*=1-*sign*

*ARL*=1/*alpha*

*ATS=ARL\*T*

print(*alpha*)

print(*ARL*)

*-***Beta** **dell’S CC tradizionale** (sapendo quanto è sigma1 rispetto a sigma0)):

*std1*=2.5\**std #sigma1=2.5sigma0*

beta=stats.*chi2*.cdf(((*S\_UCL*\*\*2)\*(*n*-1))/*std1*\*\*2,*n*-1)-stats.*chi2*.cdf(((*S\_LCL*\*\*2)\*(*n*-1))/*std1*\*\*2,*n*-1)

print(*beta*)

-**Beta** **S** **CC** **PROBABILISTICO#$**(cambio del sigma lineare senza costante addizionale qui…mentre con solo shift della media il beta andrebbe a 1-alpha sostanzialmente, a meno di relazioni particolari tra mu e sigma):

*lmbda=sigma1/sigma0* #ATTENZIONE!!!

*chi2\_cv1*=stats.*chi2*.ppf(1-*alpha*/2,*n*-1)#$

*chi2\_cv2*=stats.*chi2*.ppf(*alpha*/2,*n*-1)#$

*beta\_S\_CC*=stats.*chi2*.cdf(*chi2\_cv1*/*lmbda*\*\*2,*n*-1)-stats.*chi2*.cdf(*chi2\_cv2*/*lmbda*\*\*2,*n*-1)

print(*beta\_S\_CC*)

#è praticamente il beta di un chi2 test con lambda che qui è il rapporto delle deviazioni standard, non delle varianze

## -NOTA per qda\_package:

***(in quelli per gli small shifts possiamo specificare mean e sigma\_xbar) dove il default è la stima, XbarS è l’unico dove posso specificare una sigma (in quel caso non la stima) ma NON posso specificare la media, in più è SEMPRE possibile inserire il k (nei tradizionali, se no abbiamo ad esempio params per quelli small shifts), specificare plotit (non in tutti ) e ed eventualmente il subset\_size (questo in tutti)***

-------------------------------------------

# XbarR CC principalmente:

!!!RICORDA DI **VERIFICARE L’ASSUMPTIONS DI NID!!!**

-per fare un po’ di **cose “a mano” su XbarR CC** controlla QDA\lab\_py\_qda\SPC\_IID\ESE06\_ex1.ipynb

-fai **XbarR CC** (k=3 e in generale assumption di Normalità per le V quindi alpha=0.0027)

*#control chart with the data (ORIGINAL DATAFRAME)*

*data\_XR\_qcc* = qda.ControlCharts.XbarR(*data,*  *K* = *K\_alpha*) #Kdefault=3

il risultato è un df con l’aggiunta di queste colonne:



Nelle **colonne di test** se c’è un entry diversa da NaN significa che quel sample per Xbar o per R è un OOC!

## XbarR CC con mean/sigma note:

*#SE devo calcolare sample\_mean e sample\_R da un dataset con n>1:*

*sample\_mean* = *data*.mean(*axis*=1) *#vettore media dei samples(di ogni riga)*

*sample\_R* = *data*.max(*axis*=1) - *data*.min(*axis*=1)

*#SE devo stimare mu e std:*

*mean\_or\_mu=sample\_mean.mean()*

*std\_or\_simga=sample\_R.mean()/*qda.constants.getd2(n)

*data\_XR*=mf.xbarR(*mean\_or\_mu*,*std\_or\_sigma*,*n*,*alpha*,*sample\_mean*, *sample\_R, subset\_size=None*)

## XbarR CC con R probabilistico:

*#qui alpha è 0.002 quindi alpha/2 è 0.001 e*

*#Dalpha/2 quindi è 0.37 e D\_1-alpha/2 è 5.48*

*data\_XR\_PROB*=mf.xbarR\_prob(*mean*,*std*,*n*,0.0020,*sample\_mean*,*sample\_range*,5.48,0.37)

-fai una **OC curve Xbar Chart** (plotta beta al variare di delta per quanto riguarda il test fatto dall’Xbar CC) e anche **ARL curve**.

*delta* = np.linspace(0, 4, 100) *#100 punti da 0 a 4*

OCcurveXbarChart(*data*,*delta*,*k*=3)

## -Stima **stdev con R** + confidence interval approssimato (visto che la *sigma\_hat* è stata appena stimata) equivalente al test dell’XbarChart:

Nota: la statistica R è molto più stabile rispetto alla statistica s (soprattutto se n è basso, tipo 5 o 6). Quindi in questo specifico caso usiamo **come stima della sigma una quantità che dipende dalla media degli R calcolati**, ma se invece si hanno già tipo 10/15 la s è meglio ma per quello poi si usa di fatto un XbarS control chart

*d2* = qda.constants.getd2(*n*) #preleva una costante dal qda package

*# Make a copy of the data*

*data\_XR* = *data*.copy()

*data\_XR*['sample\_mean'] = *data*.mean(*axis*=1) *#vettore media dei samples(di ogni riga)*

*data\_XR*['sample\_range'] = *data*.max(*axis*=1) - *data*.min(*axis*=1)

*Xbar\_mean* = *data\_XR*['sample\_mean'].mean()

*R\_mean* = *data\_XR*['sample\_range'].mean()

*sigma\_hat* = *R\_mean* / *d2*

print('Sigma\_hat = %.3f' % *sigma\_hat*)

*CI* = stats.*norm*.interval(1-*alpha*, *loc*=*Xbar\_mean*, *scale*=*sigma\_hat*/np.*sqrt*(*n*))

print('CI = (%.3f, %.3f)' % *CI*)

----------------------------------

# IMR principalmente:

!!!RICORDA DI **VERIFICARE L’ASSUMPTIONS DI NID!!!**

-per fare un po’ di **cose “a mano” su IMR CC** controlla SPC\_IID\ESE06\_ex3.ipynb

*data*['MR'] = *data*['time'].diff().abs() #esempio calcolo MR

-**IMR** CC control chart:

*data\_IMR* = qda.ControlCharts.IMR(*data*, 'X') specifica nome colonna dei dati

data\_IMR è il df originale con queste colonne in aggiunta:



-**rimuovere OOC rimuovi OOC nan ooc nan sostituisci ooc** assumendo di aver **trovato assignable cause** in questi OOC della phase1 posso fare una copy (o farlo direttamente sull’originale, mettere un NaN al posto di quel dato e **rifare CC** (questo è **un esempio con IMR** ma va be):

*# make a copy of the data*

*data\_2* = *data*.copy()

*# replace the OOC point with NaN*

*data\_2*['individual'].*iloc*[*OOC\_idx*] = np.nan

*# Plot the new control chart*

*data\_IMR\_2* = qda.ControlCharts.IMR(*data\_2*, 'GM')

#note: it’s ok since all the statistic function like .mean() ignores the Nan value and m (number of samples) is never used directly in the formulas

## phase2 da array di dati

- **phase2** ***array****:* si crea un nuovo dataframe *new\_data* dove **accodo** o **concateno** quelli nuovi a quelli vecchi del dataset originale

*# Define the array with the new observations*

*new\_obs* = np.array((1.327, 1.594, 0.716, 1.767, 0.915, 2.524, 0.563, 2.053))

*# Add the new observations to the dataset*

*new\_data* = pd.concat([*data*, pd.DataFrame(*new\_obs*, *columns*=['colonnadatiphase1'])], *ignore\_index*=True)#necessario che siano accodate alle phase1 observations

*phase1length=* len(*data*)

*new\_data\_IMR* = qda.ControlCharts.IMR(*new\_data*, 'GM', *subset\_size*=*phase1length*)

#il control chart viene rifatto proprio solo che poi abbiamo anche i dati di phase2 su cui viene applicato il CC (va specificato la dimensione dei dati di phase1)

## IMR half normal

-fai un **IMR**-**half normal**, ossia fai IMR usando half normal distribution degli MRi (**probabilistic**):

ottengo in output sia l’IMR cc tradizionale, che il chart fatto sugli MR trasformati (individuals). Quello che devo “scartare” è l’MR chart del cc tradizionale.

*data\_IMR*=mf.IMRcc\_half\_normal(*data*,nome\_colonna\_dati, *alpha*=0.0027)

## IMR boxcox

**-** fai IMR-boxcox **IMR** CC usando **boxcox teorico (elevazione a 0.4) sugli MRi**(**probabilistic**): (i dettagli li trovo in SPC\_IID\ESE06\_ex5.ipynb), qui stesso discorso del precedente) **Attenzione al risultato dello** **shapirowilk sugli MR trasformati**!!!

*data\_MR\_transf*=mf.IMRccboxcox(*data*, *nome\_colonna\_dati, alpha*=0.0027)

## **☺** **Funzione** per Tutte (“”) le casistiche in cui faccio I cc delle medie o IMR:

*sample\_means*=*data\_XBarR\_on\_original\_data*['sample\_mean'] #oppure ce l’ho già oppure ho un vettore di individuals (n=1), ad esempio gli scores di una PC

*grand\_mean*= *sample\_means*.mean() *Average\_R*=*data\_XBarR\_on\_original\_data*['sample\_range'].mean() *MR\_of\_Xbar\_vector*=*data\_XBarR\_on\_original\_data*['sample\_mean'].diff().abs()

*Average\_MR\_of\_Xbar*=*MR\_of\_Xbar\_vector*.mean()

*d2\_n*= qda.constants.getd2(*n*) #n diverso da 1

*d2\_2*= qda.constants.getd2(2) #qui n in realtà è 1 ma per la questione MR…

*Sigma\_v1*= *Average\_R*/(*d2\_n*\*np.*sqrt*(*n*)) *# (parto da n>1)solo se non ho stima di sigma*

*Sigma\_v2*= *Average\_MR\_of\_Xbar*/*d2\_2* *# (se* ***parto da n=1****) solo se non ho stima di sigma*

*#---usa la funzione*

*k*=3

*mean\_or\_mu*=*grand\_mean*

*sigma\_v*=*Sigma\_v2 #ad esempio*

*df\_CC*=mf.IMR\_my(*sample\_means*,*n*,*k*,*mean\_or\_mu*,*sigma\_****v***,*mean\_to\_est*='yes',*sigma\_to\_est*='yes','yes')#ultimo ‘yes’ sta per “plotta MR chart”

NOTA: questa mia funzione è utile in tutti i casi in cui voglio fare un **IMR control chart e mi vengono date già il sigma e/o la mu**!

Ad esempio potrei usarlo per fare **un IMR cc su una PC** dato il lambda, in quel caso n=1, mean\_or\_mu=0, sigma\_v=sigma\_x perché n=1 e quindi sigmav=np.sqrt(lambda) e non c’è nulla da stimare.

-------------------------

# XbarS CC principalmente:

!!!RICORDA DI **VERIFICARE L’ASSUMPTIONS DI NID!!!**

- per fare un po’ di **cose “a mano” su XbarS CC** controlla SPC\_IID\ESE06\_ex4.ipynb

-**XbarS** CC (nell’esempio è fatto su un dataset che arriva dalla boxcoxunstack):

*data\_BC\_XS* = qda.ControlCharts.XbarS(*data\_BC\_unstack*)

XbarS(*original\_df*, *K* = 3, *sigma* = None, *subset\_size* = None, *plotit* = True)

#in questo caso è possibile specificare la sigma\_x se nota!



## XbarS CC con mean nota

(abbiamo già sample\_means e sample\_stds) vai su:

QDA\Exams ecc\TE\_from\_EE2024\CC\_iid\ese1\_pag2.ipynb

oppure direttamente:

#se devo stimare std allora:

*std\_or\_sigma=mean(sample\_std)/qda.constants.getc4(n)* #OPPURE do’ in input un valore dato dal testo.

#SE mi viene dato un dataset vero e proprio (chiaramente con n>1) e non già medie e #std dei sample allora devo tirar fuori i due vettori di *sample\_mean* e *sample\_std*

perché la funzione vuole quelli!

*data\_XS2*=mf.xbars\_mu\_given(*mu*,*std\_or\_sigma*,*n*,*alpha*,*sample\_mean*,*sample\_std, subset\_size=None*)

## XbarS CC probabilistico

#stesso discorso fatto sopra

*data\_XS\_probabilistic*=mf.XbarS\_prob(*mu\_or\_mean,std\_or\_sigma,n,alpha,sample\_mean, sample\_std, subset\_size=None*)

--------

# ASSUMPTION IID CADE:

## **☺**caso **n=1**, supponiamo di aver iid che cade e di aver già fatto un IMR cc tradizionale e di avere il suo output, ossia abbiamo fatto tipo:

*data\_IMR* = qda.ControlCharts.IMR(*data*, 'X')

poi abbiamo **modellizzato** con successo i dati (residui ok).

-quindi ora facciamo un **TREND CC (**Può essere pensato come sostituto dell’I chart nel SCC, ma non dell’MR chart dello SCC**)** (nota che data\_IMR viene modificato di conseguenza nelle colonne che partono con ‘I’, infatti i nuovi I\_CL sono i fitted values, e gli I\_UCL e gli I\_LCL sono i control limits del TREND CC e ovviamente gli I\_TEST sono quelli che varcano i Control limits). Nota: è necessario aver fatto prima un IMR CC tradizionale che dà in output il primo *data\_IMR*

[*data\_IMR*,*df\_res*]=mf. IMR\_TrendCC(*data\_IMR*, *residual*,*fittedvalues*, *datapositive*='no',*alpha*=0.0027)

NOTA: **se ho usato ARIMA package** devo ricordarmi come sempre che avrò dei NaN all’inizio nei residual e nei fittedvalues

### -SCC

(I+MR sui residui), semplicementeusiamo la funzione dell’IMR del qda package su un dataframe che contiene esclusivamente i residui ovviamente**:**

*df\_SCC* = pd.DataFrame({'res': *model*.resid}) #lo chiamo direttamente *df\_SCC*

*df\_SCC* = qda.ControlCharts.IMR(*df\_SCC*, 'res')

**-**assumiamo di aver beccato un OOC con assignable cause nello **SCC** dei residui di un modello e di voler usare il **dummy** **approach**

Allora torno indietro nel codice e creo una colonna nel dataset originale in cui abbiamo un 1 per il sample responsabile dell’OOC e poi RIPETO TUTTO (modellizzazione in cui a sto giro buttiamo dentro anche la dummy appena creata, IMR control chart tradizionale, Trend CC e SCC):

*data*['dummy'] = np.where((*data*['t'] == 8), 1, 0)

#in questo esempio ho messo direttamente il **valore** del sample marcio

*data*.head()

------------------------------

#il modo “esteso” sarebbe:

*OOC\_indx*=mf.extractOOCindx(*df\_res*,'MR\_TEST1')

*values\_to\_check* = *data*['t'].*iloc*[*OOC\_indx*-1] *#il -1 è messo a mano (ispez visiva visto che OOC è risultato nell’MR chart e va capito qual è il sample marcio)*

print(*values\_to\_check*.values)

*data*['dummy'] = *data*['t'].apply(**lambda** *x*: 1 if *x* in *values\_to\_check*.values else 0)

-------------------------------

#o una cosa tipo questa:

*OOC\_indx*=mf.extractOOCindx(*df\_SCC*,'I\_TEST1') #o vedendo a occhio

*OOC\_indx\_corrected*=*OOC\_indx*+1 *#basato su quanto detto*

*data*['observation']=np.linspace(0,len(data)-1,len(data)) #praticamente un vettore di indici che parte proprio da 0 e va fino a len(data)-1

*data*['dummy\_OOC'] = np.where((*data*['observation'] == int(*OOC\_indx\_corrected*)), 1, 0)

- **phase2 usa modello usare modello e SCC**: ipotizziamo qui che arrivi un altro dataset *data\_ph2* e che dobbiamo calcolare i fits su questi dati di phase2 usando lo stesso modello e poi verificare se è IC o OOC stando a quello costruito in phase1: (###**OPPURE se ho usato ARIMA package** ###)

*data\_ph2*=pd.read\_csv('July-1-2.csv')

#ricostruiamo i regressori anche per il nuovo dataset (ed eventualmente **boxcox** su y che qui è *data\_ph2*['te'])

*data\_ph2*['exp']=np.exp(-0.1\**data\_ph2*['t'])

*data\_ph2*['lag1']=*data\_ph2*['te'].shift(1)

#usiamo il modello:

*predictions\_ph2*=*model*.predict(*data\_ph2*.*iloc*[1:,2:])

#*data\_ph2*.*iloc*[1:,2:]) son i miei regressori skippando prima riga per il lag1

*residuals\_ph2*= *data\_ph2*['te'].*iloc*[1:]-*predictions\_ph2* #calcoliamo i residui

#facciamo SCC (abbiamo bisogno di concatenare):

*res\_all*= pd.concat([*model*.resid, *residuals\_ph2*], *ignore\_index*=True)

*df\_SCC2* = pd.DataFrame({'res': *res\_all*})

*df\_SCC2* = qda.ControlCharts.IMR(*df\_SCC2*, 'res',***subset\_size***=len(*model2*.resid),*K*=*k\_alpha*)

--------------------------------------------------------------------------------

#############OPPURE se ho usato ARIMA package##############:

#in questo caso ho precedentemente usato ARIMA package fittando un ARIMA(1,1,0), i #cui residui sono *residuals1,* con differenziazione e poi lag1 diciamo

*old\_data*=*data*['Thickness'].*values* #dati di phase1

*new\_data*=np.array([49,69,78,68,70,77,77,85,73,86]) #dati di phase2 che temporalmente arrivano #esattamente dopo quelli di phase1

*total\_data*=np.concatenate((*old\_data*,*new\_data*)) *#****con due (())***

*total* = pd.DataFrame({'total\_data': *total\_data*})

#usiamo la **formula diretta** dell’ARIMA(1,1,0) che si ricava prendendo il coefficiente del lag1 *a*=-0.5994 e facendo due conti a mano*:*

*total*['fits']=(1+*a*)\**total*['total\_data'].shift(1)-*a*\**total*['total\_data'].shift(2) *#ARIMA(1,1,0)*

*total*['residuals']=*total*['total\_data']-*total*['fits']*#ci sono 2 nan al contrario #dell'unico NaN in uscita* *da ARIMA package*

*#aggiungi residuo che manca(trucchetto che ci può stare visto che i rimanenti sono identici a meno di approssimazioni numeriche):*

*total*['residuals'].*iloc*[1]=*residuals1*.values[0] #è il primo residuo che manca

*resi*=*total*['residuals'].*iloc*[1:]*#skippa il NaN che c’è giustamente*

*df\_SCC* = pd.DataFrame({'res': *resi*})

*df\_SCC* = qda.ControlCharts.IMR(*df\_SCC*, 'res',***subset\_size***=len(*residuals1*))

#nota: in questo caso non badare alla linea che separa i dati di phase1 e phase2 ma guarda se i Control limits sono i medesimi di quelli fatto sui dati di phase1

-Per il **FVC** posso usare:

mf.plotyvsfits(*data*['time'],*model*)

## **☺**caso **n>1** con osservazioni nei sample **IN ORDINE**

-**vettorializza** dataset e mantieni ordine temporale (**da qui si lavora su questi come se avessi una normalissima time series, ossia il caso n=1**, quindi si andrà a modellizzare solitamente\*\*\*):

*data\_stack*=mf.vettorializza(*data*)

-shapiro sugli MR post SCC (necessario recuperare gli MR dal dataframe in output dal **SCC**):

mf.shapiroqq(*df\_SCC*['MR'].*iloc*[1:]) #negli MR c’è **sempre** il primo che è **NaN**

### -IonMRresBoxCox:

Se dopo aver fatto un IMR cc sui residui del modello trovo degli OOC nei residui **solo** nel grafico dell’MR, potrei trasformarli (se non c’è **assignable cause** per quei sample!!!) e vedere cosa cambia (approccio probablistic). Qui è tutto in automatico! Box-cox + shapiro + individual CC sugli MR dei residui trasformati con boxcox (sostituisce l’MR chart dell’I-MR fatto sui residui, mentre l’Individual control chart sui residui rimane affidabile). *Nota che l’input è il dataframe di output dello* **SCC** *fatto sui residui originali*

*df\_MR\_boxcox*=mf.IonMRresBoxCox(*df\_SCC,use04='no',alpha=0.0027,subset\_\_size='False'*) #guarda la colonna I-Test1 per l’OOC

Nota\*\*\*: se non riesco a modellizzare e vedo hugging o stratification può valer la pena fare un I-MR-R control chart partendo dai dati non vettorializzati praticamente.

## **☺**caso **n>1** con osservazioni nei sample **NON ORDINATI**

-è bene sempre **partire con un XBAR-R** control chart ad (**o XBAR-S**) anche perché potremmo al massimo aver fatto un dotplot per verificare la iid assumption, quindi:

*data\_XBarR\_on\_original\_data* = qda.ControlCharts.XbarR(*data*) *#lo plotta comunque in ogni caso:| (solo IMR ha l'opzione figa di non essere plottato)*

*#questo qui sopra fondamentalmente ha bisogno di* ***n>1*** *ovviamente*

-se notiamo dell’**overdispersion**, ad esempio, **nell’Xbar chart** potremmo approcciare la cosa tramite un I-MR-R CC. usiamo quindi l’output del precedente XbarR control chart *data\_XBarR\_on\_original\_data* perché contiene già le sample mean, e poi:

*data\_IMR\_on\_means* = qda.ControlCharts.IMR(*data\_XBarR\_on\_original\_data*, 'sample\_mean') *#se non ho un dataset #parto da una colonna simile (sample means)*

*#questo qui sopra sostanzialmente fa l'IMR chart sulle sample means*

*# ossia tratta le means come individuals!!!*

*#ora ho quindi un* ***I-MR-R*** *control chart, perchè nel primissimo XbarR mi tengo R cc (scarto Xbar cc)*

*#e negli altri due risultanti da dall’IMR guardo entrambi ovviamente.*

*#Quello che cambia tra l'Xbar di XbarR cc e l'I di I-MR fatto sulle sample means sono i control limits*

*#ossia come viene stimata la sigma (formula1 vs formula2, guarda sotto la Nota1@)*

A questo punto abbiamo 4 charts in sostanza: i 2 della XBAR-R CC sui dati originali e i 2 della IMR CC sulle medie. L’unico veramente da scartare è l’Xbar chart, mentre

1) Individual Chart dell’IMR permette di monitorare la **media del processo** trattando le medie del campione come osservazioni individuali;

2) il chart MR consente di monitorare la **variabilità tra** campioni (perché fatto sugli individuals);

3) il chart R dell’Xbar-R CC consente di monitorare la **variabilità all'interno** del campione.

Questo schema di CCs è efficiente in questi casi ed è chiamato **I-MR-R** o **I-MR-S (**se abbiamo fatto XbarS al posto di XbarR**)**

Nota1@: L’I Chart è diverso dall’Xbar chart (che scartiamo) poiché la sigmav è stimata in modo diverso!!! La formula dell’I chart non si richiede l’iid assumption dei datiImmagine che contiene testo, calligrafia, Carattere, linea

Descrizione generata automaticamente

Nota2\*\*\*: In realtà non è detto che riusciamo a togliere l’autocorrelazione col batching, ossia lavorando con le medie dei samples. Se a questo punto con le medie non riuscissimo a levare ad esempio l’autocorrelazione (abbiamo non random pattern nella time series delle sample mean e ad esempio vediamo overdispersion nell’I chart fatto sulle medie) allora a quel punto, se ho capito bene, si pùò prendere una di queste due vie:

A)modellizzare il vettore delle sample means, fare poi IMR sui residui del modello fatto sulle medie.

B)Non modellizzo, faccio un ulteriore gapping, ossia seleziono non tutta la series delle mean, ma una mean ogni 4 ad esempio.

### I-MR-R (di più non si può) o I-MR-S

*data\_XBarR\_on\_original\_data* = qda.ControlCharts.XbarR(*data*) #qui non guardo Xbar ma guardo R

*data\_IMR\_on\_means* = qda.ControlCharts.IMR(*data\_XBarR\_on\_original\_data*, 'sample\_mean') #qui guardo sia l’I chart che l’MR chart

#1)**I chart** permette di monitorare la **media del processo** trattando le medie del campione come osservazioni individuali;

#2) il **chart MR** consente di monitorare la **variabilità tra** campioni;

#3) il **chart R** consente di monitorare la **variabilità all'interno** del campione

----------

# MVCC principalmente (p>1 SEMPRE):

-RICORDA DI **“VERIFICARE” L’ASSUMPTIONS DI NID** (se qualcosa va storto in termini di N usa boxcox sulla quality feature che lo necessita!)

-All’inizio ho i casi con n>1, il terzultimo è flessibile e gli ultimi due con n=1

## ☺CASO in cui abbiamo un vero e proprio dataset di phase1 con n>1:

-estrarre una df (mx**n**) per ogni quality feature se chiaramente viene fornito un unico dataset mxn\*p

*Width* = *data*[['Width1', 'Width2', 'Width3']] #eg. Con n=3

*Corr\_speed* = *data*[['Corr.speed1', 'Corr.speed2', 'Corr.speed3']]

#se le var sono solo queste due allora p=2 chiaramente

- visualizzo OGNUNO di questi df con **dotplot** (ora è come avere dei dataframe con p=1 e n>1) per capire randomness ecc (solite cose)

mf.dotplot(*Width*) #ad esempio

-poi **shapiro per ognuno**: (assumendo poi che la normalità di entrambe corrisponda a multinormalità)

mf.shapiroqq(*Width*,0.05,'yes') #ad esempio, 'yes' tiene conto che deve fare un unstack

-poi fare scatterplot (scatternice non va bene perché richiede un singolo dataset):

plt.scatter(*Width*, *Corr\_speed*) #esempio

plt.xlabel('Width')

plt.ylabel('Corr. speed')

plt.show()

-**calcola sample mean df (***sample\_mean* diventa un df mx**p) e stimare** **grand mean**

*# Create a new dataframe to store the sample mean*

*sample\_mean* = pd.DataFrame()

*#devo calcolare una media per ogni riga (ogni sample) per ottenere il vettore delle medie, ossia si opera la media lungo le colonne*

*sample\_mean*['width'] = *Width*.mean(*axis*=1) #si lavora con i sample mean vectors!!!

*#faccio la stessa cosa per l'altra variabile:*

*sample\_mean*['corr\_speed'] = *Corr\_speed*.mean(*axis*=1)

*# Calculate the grand mean*

*Xbarbar* = *sample\_mean*.mean() #

print(*Xbarbar*)

-**stimare** **sample cov matrix o stimare** **sample covariance matrix: (**Per i dettagli vedi multivariate\_cc\ESE8\_ex1.ipynb**)**

*data\_stack* = pd.DataFrame()

#qui ho 2 variabili, ‘sample’ è solo l’indice che creo

*data\_stack*[['sample', 'width']] = *Width*.transpose().melt()

*data\_stack*['corr\_speed'] = *Corr\_speed*.transpose().melt()['value']

#eg. Se ce ne fosse un’altra:

# *data\_stack*['glide'] = *glide*.transpose().melt()['value']

*#quello che otteniamo è sostanzialmente un dataframe composto da covariance matrices che tecnicamente è un multi-indexed dataframe*

*cov\_matrix* = *data\_stack*.groupby('sample').cov()

*#print(type(cov\_matrix))*

*cov\_matrix*.head(8)

*S* = *cov\_matrix*.groupby(*level*=1).mean()

*cols* = *S*.*columns*.tolist()

*S* = *S*.reindex(*columns*=*cols*, *index*=*cols*)

print(*S*)

-fai un Hotelling CC per la phase1 (calcolo dell’UCL e della T2 per ogni sample):

[*UCL*,*T2*]=mf.HotCC\_phase1(*p*,*m*,*n*,*alpha*, *sample\_mean*,*Xbarbar*,*S,quanti\_sample\_levati=0*)

*quanti\_sample\_levati* Serve per eventuali OOC trovati in precedenza

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

-calcola la **statistica d** per la diagnosi degli OOC:

*d*=mf.calculate\_d(*sample\_mean*, *mu*, *Sigma*)

#mu è una pandas series

#Sigma è un pandas dataframe

#esempio:

*sample\_mean* = {'x1': [2, 1, 1],

        'x2': [0, 1, -1],

        'x3': [0, -1, 0],

       }

*sample\_mean*=pd.DataFrame(*sample\_mean*)

*Sigma* = pd.DataFrame([[1, 0.9,0.9],

                [0.9,1,0.9],

                [0.9,0.9,1]],

*columns*=['x1', 'x2','x3'],

*index*=['x1', 'x2','x3'])

*mu* = pd.Series({'x1': 0, 'x2': 0, 'x3': 0})

*d*=mf.calculate\_d(*sample\_mean*, *mu*, *Sigma*)

-se vengono degli OOC (non avrò in output gli indici degli OOC dal codice di prima, **devo guardare**), se **richiesto (assignable cause is found)**, torna indietro e togli quei sample tipo cosi direttamente dal df *sample\_mean*:

*sample\_mean*.*iloc*[OOC\_idx,1:] = np.nan #salto la Colonna 0 perché è una sorta di indice del campione in questo caso

e **rifai (ri-runna** mf.HotCC\_phase1 **modificando m e metti** *quanti\_sample\_levati***)**. è importante anche perché devo **ristimare** correttamente *Xbarbar e S!*

-se ho anche dei **dati di phase2** allora supponendo che debba dividere il dataset in 2 (phase1 e phase2) data per prima cosa avrei dovuto dividere e fare tutto il lavoro descritto nei punti precedenti su *Data\_first*:

*Data\_first* = df.iloc[:c]

*Data\_second* = df.iloc[-l:]

Poi mi calcolo il dataframe *sample\_mean2* allo stesso modo della phase1 poiché mi servono ovviamente per calcolare le T2 di ognuno dei miei nuovi sample (**SENZA** **ricalcolare di nuovo *Xbarbar e S****)* e a questo punto:

[*UCL*,*T2*]= mf.HotCC\_phase2(*p*,*m*, *m\_phase2*,*n*,*alpha*,*sample\_mean2*,*Xbarbar*,*S*)

#m:numero di sample effettivi usati nell’ultimo “tentativo” di phase1

#m\_phase2: numero di sample da controllare in phase2

Attenzione che **l’UCL** qui è **diverso** solo perché c’è c2 che moltiplica e non c1, quindi è corretto che vengano diversi i control limits!!!

## ☺caso in cui C’E’ GIA’ STATA una phase1, abbiamo le **#stime** e i sample da controllare in phase2 (qui può essere considerata davvero phase2 visto che abbiamo le stime, la phase1 è già avvenuta) (con n>1)

-Importiamo i dati (da notare che i parametri *Xbarbar e S* che ci danno è come se arrivassero da una phase1 dove tutto è IC e quindi quei parametri sono approvati praticamente):

*n* = 25          *# (sample size)*

*p* = 2           *# number of random variables*

*alpha* = 0.05    *# significance level*

*#attenzione!!! sample\_means QUI è* m\_phase2x**p**

*sample\_means* = pd.DataFrame({'xbar1': [58, 60], 'xbar2': [32, 33]}, *index*=['sample1', 'sample2']) #sono i 2 sample (perché ho solo 2 elementi nelle []) che dovremo controllare (phase2), e giusto per confondersi ho anche p=2 cioè ho solo 2 features e quindi xbar1 e xbar2.

*#immaginiamo che ci sia stata una phase1 su m samples e questi sono i risultati:*

*Xbarbar* = pd.Series({'xbar1': 55, 'xbar2': 30})

*S* = pd.DataFrame([[200, 130],

                [130, 120]],

*columns*=['xbar1', 'xbar2'],

*index*=['xbar1', 'xbar2'])

-da qui potrei usare mf.HotCC\_phase2(*p*,*m*,*m\_phase2*,*n*,*alpha*,*sample\_mean2*,*Xbarbar*,*S*) qualora mi sia detto l’m della phase1 oppure mf.chi2cc(*p*,*m*,*n*,*alpha*,*sample\_means*,*mu*,*SIGMA*) se uso l’approssimazione asintotica.

-calcola la **T2** statistics in questo caso per 1 solo sample:

*S\_inv* = np.linalg.inv(*S*)

*T2\_1* = *n* \* (*sample\_means*.*loc*['sample1'] - *Xbarbar*).transpose().dot(*S\_inv*).dot(*sample\_means*.*loc*['sample1'] - *Xbarbar*) #va fatto anche #per 'sample2'

print("T2\_1 = %.3f" % *T2\_1*)

-**UCL** **asintotico** (nel mio caso era una phase2 a tutti gli effetti perché i miei parametri dati erano stime che possiamo immaginare essere fatte su un dataframe di phase1):

*UCL* = stats.chi2.ppf(1-*alpha*, *p*) #ovviamente non contano il numero dei sample da cui è stata fatta l’ipotetica phase1 se utilizziamo l’UCL asintotico con la chi2

print("UCL = %.4f" % *UCL*)

-**UCL** **effettivo** (phase2) assumendo un certo m da cui sono stati stimati i parametri:

*m* = 20          *# number of samples*

*c2* = (*p*\*(*n*-1)\*(*m*+1))/(*m*\*(*n*-1)-(*p*-1))

*UCL\_new* = *c2*\*stats.f.ppf(1-*alpha*, *p*, (*m*\*(*n*-1)-(*p*-1)))

print("New UCL = %.4f" % *UCL\_new*)

## **☺**caso in cui NON c’è una vera phase1, abbiamo i **parametri** **veri** e i sample da controllare (n>1 o n=1)

-si procede come nel caso precedente assumendo ovviamente *mu* e *SIGMA* siano una pandas series e un pandas dataframe come nel caso precedente (rispetto a caso precedente al posto di *Xbarbar* ho *mu* eal posto di *S* ho *SIGMA*) usando però la distribuzione chi2 per il test dei sample da controllare(Che è come abbiamo fatto per il caso asintotico):

*S\_inv* = np.linalg.inv(*SIGMA*)

#Esempio di calcolo della chi2 stat per un solo sample

*chi2\_1* = *n* \* (*sample\_means*.*loc*['sample1'] - *mu*).transpose().dot(*S\_inv*).dot(*sample\_means*.*loc*['sample1'] - *mu*)

**UCL chi2: (visto che i parametri veri sono noti si usa la chi2 distribution)**

*UCL* = stats.chi2.ppf(1 - *alpha*, *df* = *p*)

-Se ho un set di dati di phase2 da controllare nel dataframe data\* allora:

[*UCL*,*data\_CC*]=mf.chi2cc(*p*,*m*,*n*,*alpha*,*sample\_means*,*mu*,*SIGMA*)

#*m*: il numero di dati da controllare.

\*nota che *sample\_means* può essere

1)la matrice di sample means mx**p** calcolata dal dataframe originale come visto sopra per il CASO in cui abbiamo un vero e proprio dataset con n >1 (primo caso)

2)una parte (phase2) del df originale mx**p** data qualora n=1.

## ☺caso in cui NON c’è una vera phase1, abbiamo i parametri veri e vogliamo fare univariate CC con Bonferroni (con n>1)

-importiamo quello che ci viene dato:

*# Define the parameters*

*ARL* = 500

*p* = 3           *# number of random variables*

*n* = 25          *# number of replicates (sample size)*

*mu\_x* = 10

*var\_x* = 1

Immagine che contiene testo, calligrafia, Carattere, tipografia

Descrizione generata automaticamente#useremo queste formule per gli LCL e gli UCL degli Univariate CC

*mu\_y* = 12

*var\_y* = 1.5

*mu\_z* = 13

*var\_z* = 2

*# Calculate the Z*

*alpha\_FAM* = 1 / *ARL* *#alpha overall 1/500*

*alpha* = *alpha\_FAM* / *p* *#singoli alpha 1/1500*

*Zalpha2* = norm.ppf(1-*alpha*/2)

print("alpha\_FAM = %f " % *alpha\_FAM*)

print("alpha = %f " % *alpha*)

print("Z alpha/2 = %.3f " % *Zalpha2*)

-calcoliamo I Control limits degli Univariate CC:

*UCL\_x* = *mu\_x* + *Zalpha2*\*np.sqrt(*var\_x*)/np.sqrt(*n*)

*CL\_x* = *mu\_x*

*LCL\_x* = *mu\_x* - *Zalpha2*\*np.sqrt(*var\_x*)/np.sqrt(*n*)

#idem per le altre 2 variabili usando le *var\_y* e *var\_z*

*# Print a table with the control limits*

*results* = pd.DataFrame({'UCL': [*UCL\_x*, *UCL\_y*, *UCL\_z*],

                        'CL': [*CL\_x*, *CL\_y*, *CL\_z*],

                        'LCL': [*LCL\_x*, *LCL\_y*, *LCL\_z*]}, *index* = ['x', 'y', 'z'])

print(*results*)

## ☺caso in cui NON c’è una vera phase1, abbiamo i parametri veri e vogliamo fare univariate CC con **Bonferroni** (con n>1):

Vedi C:\Users\alyuk\Desktop\Polimi\2S2A\QDA\Exams ecc\TE\_from\_CQPP\08\_07\_2022\ese2.ipynb

## ☺caso in cui NON c’è una vera phase1, abbiamo i **parametri veri** ma dobbiamo calcolarci prima qualcosina(con n=1) e abbiamo un dataset da controllare (come se fosse phase2 ma va be, quando ci sono i parametri la distinzione conta poco)

-importiamo quello che abbiamo e dalla corr e le std calcoliamo la cov

*std\_1* = 1.483

*std\_2* = 1.435

*std\_3*= 1.530

*corr12* = 0.799

*corr13*=0.705

*corr23*= 0.683

*#correlazione e std sono date, dobbiamo costruire la sigma da*

*# Calculate the variance/covariance matrix*

*covariance12* = *corr12* \* *std\_1* \* *std\_2*

*covariance23*= *corr23* \* *std\_3* \* *std\_2*

*covariance13*= *corr13* \* *std\_3* \* *std\_1*

*var\_1* = *std\_1*\*\*2

*var\_2* = *std\_2*\*\*2

*var\_3*=*std\_3*\*\*2

*# define the covariance matrix as a pandas dataframe*

*SIGMA* = pd.DataFrame([[*var\_1*, *covariance12*,*covariance13*],

                [*covariance12*, *var\_2*,*covariance23*],

                [*covariance13*,*covariance23*,*var\_3*]],

*columns*=['x1', 'x2','x3'],

*index*=['x1', 'x2','x3'])

*alpha* = 0.0027 *# significance level*

*n* = 1          *# sample size*

*m* = len(*data*)  *# number of samples*

*p* = 3          *# number of variables*

-gestiamo il dataset da controllare:

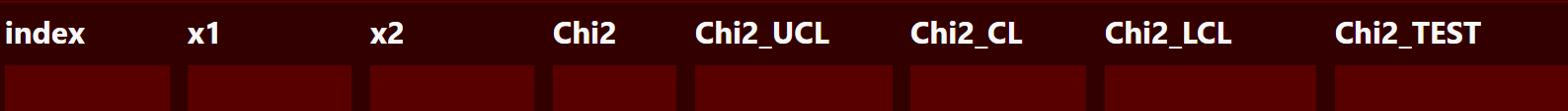
*data* = pd.read\_csv('ESE8\_ex4.csv')

*data* = *data*.drop('Sample', *axis*=1) *#non ce ne facciamo nulla della colonna Sample*

-fai un **chi2** **CC** per questo caso con n=1 usando la **chi2 statistics:**

[*UCL*,*data\_CC*]=mf.chi2cc\_n\_1(*p*,*m*,*alpha*,*data*,*mu*,*SIGMA*) #nota che data è

Con *data\_CC* che è un df con le seguenti colonne:



Nota:il fatto che ho i parametri mi fa usare la chi2 al posto della hotelling T2!

## ☺CASO in cui abbiamo un dataset di phase1 (stimiamo i parametri) con n=1

**-stima** **sample mean vector:**

*Xbar* = *data*.mean() #sample mean vector

**-stima** **sample cov matrix o stima** **sample covariance matrix:**

*V* = *data*.diff().dropna() *#matrice (m-1)x(p)*

*# Calculate the short range estimator S2*

*S2* = 1/2 \* *V*.transpose().dot(*V*) / (*m*-1) #Short range estimator

*# Display the short range estimator*

print("The short range estimator is: \n", *S2*)

*S1* = *data2*.cov() #il long range estimator è la semplice cov matrix 😊

print("The long range estimator is: \n", *S1*)

-fare un **T2 CC** per la phase1:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

[*UCL*,*data\_CC*]=mf.T2cc\_n\_1(*p*,*m*,*alpha*,*data*,*Xbar*,*S2*)

-In phase2 bisogna usare un’altra formula per i control limits (ma la stessa per la T2): (nota che non vengono stampati anche quelli di phase1)

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

*new\_data*['X1\_plus']=*data*['X1']+1e-3

*new\_data*['X1\_norm']=*new\_data*['X1\_plus'].transform(**lambda** *x*: ((*x*\*\**lmbda*)))

*data\_to\_use\_2*=pd.DataFrame({'X1': *new\_data*['X1\_norm']})

*data\_to\_use\_2*['X2']=*new\_data*['X2'] #dati di phase2 (qui trasformati)

*m*=30 #numero dati di phase1

*m\_ph2*=10 #numero dati di phase2

*alpha*=0.02 #alpha overall (ovvio che sia overall perché è un MVC)

*p*=2 #numero di quality feature

[*UCL\_ph2*,*data\_CC\_ph2*]=mf.T2cc\_n\_1\_ph2(*p*,*m*,*m\_ph2*,*alpha*,*data\_to\_use\_2*,*Xbar*,*S2*)

# SMALL SHIFTS CCs

## ☺caso in cui vengono usati in presenza di violazione di iid assumption (qui tecnicamente ci dicono n=5, MA NON ci vengono forniti gli xij, ci vengono fornite le sample mean, gli xbar\_i)

Nota bene: *data* non contiene gli xij, ma le sample mean (è un df costituito dal sample mean vector)

-esempio **CUSUM**:

*xbarbar* = *data*['EXE4'].mean() #grand\_mean

*sigma\_xbar* = *sigma*/np.*sqrt*(*n*) #ho già la sigma e mi serve *sigma\_xbar*

*df\_CUSUM*=mf.cusum(*data*,'EXE4',*xbarbar*,*sigma\_xbar*,*h*=4,*k*=0.5)

in realtà qda.package ce l’ha:

*sigma\_xbar* = *sigma*/np.*sqrt*(*n*) #ho già la sigma

*df\_CUSUM* = qda.ControlCharts.CUSUM(*data*, 'EXE4', *params*=(*h*,*k*), *mean* = 10.75, *sigma\_xbar* = *sigma\_xbar*) #esempio in cui ho sia la mean che la sigma

-esempio **EWMA**:

*xbarbar* = *data*['EXE4'].mean()

*sigma\_xbar* = *sigma*/np.*sqrt*(*n*) #ho già la sigma

*lambda\_* = 0.2 #più vado verso 1 e più ho Schewart, più vado verso 0 e più ho CUSUM

*df\_EWMA*=mf.EWMA(*data*,'EXE4',*xbarbar*,*sigma\_xbar*,*lambda\_*)

in realtà qda.package ce l’ha:

*sigma\_xbar* = *sigma*/np.*sqrt*(*n*) #ho già la sigma

*lambda\_* = 0.2

*df\_EWMA* = qda.ControlCharts.EWMA(*data*, 'EXE4', *lambda\_*, 10.75, *sigma\_xbar*)

## **☺****caso in cui i dati rispettano la Nid (abbiamo n=5 quindi** verifica qualitativa della indipendence visto che non sappiamo l’ordine con mf.dotplot e della Normalità mf.shapiroqq specificando yes in multiobs**), ci vengono date mu e sigma e lo voglio usare veramente per detectare small shifts!**

Immagine che contiene testo, Carattere, linea, calligrafia

Descrizione generata automaticamenteCp è generalmente < 1 per i processi in cui usiamo questi CCs

Nota: **fare un XbarR o un XbarS** ci porta a dire “ok è tutto IC” ma non sto ragionando in termini di small shifts! (UCL e LCL sono calcolati sulla base della variabilità del processo, stando a questo magari un processo potrebbe essere pensato come IC quando gli shifts sono molto piccoli chiaramente perché la potenza per quegli shifts è molto bassa, ma in questo caso siamo interessanti a detectarli!)

Tuttavia estraiamo velocemente le medie (da cui poi possiamo fare quello che abbiamo visto sopra):

*data\_mean* = pd.DataFrame(*data\_XS*['sample\_mean']) #sample mean vector

poi possiamo fare quello che abbiamo visto sopra poiché *data* era il sample mean vector praticamente

NOTA:

-Potrebbero darmi un vero e proprio data\_set anche con n=1 (e ovviamente p=1), **in quel caso come** *data* **uso gli individuals**!

-Potremmo usarli anche sui **residui**, basta creare un pandas dataframe dei residui e siamo apposto (le funzioni mie e del qda package necessitano di un pandas dataframe)