# INTRO

-**Distribuzioni**: norm, t, chi2, f (NON F)

-Plot della normale con parte di area colorata e tratteggio su un certo percentile: fine della Exercise\_class1\_PART2.ipynb

-arange:

np.arange(start,stop\_escluso,**step**)

-linspace:

np.linspace(start, stop\_incluso, **n**)

-crea un **np.array classico (flattened)**:

X=np.array([4,5,6,7])

-crea un **np.array colonna**:

*a* = np.array([[0.5], [0.5], [0.5], [0.5]]) ***#column vector***

-**accedi** **velocemente** ad un elemento del df:

value = *data*.iat[1, 2] #eg. Seconda riga, Terza colonna

-ottieni nuove colonne con **formule matematiche** sulla colonna dei dati:

*data* ['exp'] = np.exp(-2 \* *data* ['x']) #e-2x

*data* ['cos'] = np.cos(*data* ['x'])

*data* ['sin'] = np.sin(*data* ['x'])

*data* ['square'] = *data* ['x'] \*\* 2

-**for i in range(n):** fa andare **i da 0 a n-1**

-prendi una colonna di un df e convertila in np.array tramite **.values**:

np\_array\_x= df['x'].**values**

## -crea un **dataframe** vuoto da riempire successivamente (con dei numpy array ad esempio):

#caso base1

*data*=pd.DataFrame() #dataframe vuoto

*numeri*=np.array([1,3,5,56,655,433,65,4,45,423,4376,351])

*data*['numb']=*numeri*

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

#caso base2 dove riempio subito (concettualmente identico al caso base1)

#qui Creo **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)})

#NOTA: quello che metto in generale deve essere o una pandas series o un np.array **flattened**

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

#caso3: qui voglio riempire 5 colonne con CL,UCL E LCL perché avrò gli array di sample mean e sample std da inserire all’interno (qui volevo fare XbarS cc con parametri noti)

*c4*=qda.constants.getc4(*n*)

*data\_XS* = pd.DataFrame(*columns*=['Xbar\_CL', 'Xbar\_UCL', 'Xbar\_LCL','S\_CL','S\_UCL','S\_LCL']) #qui setto prime il nome delle colonne che verranno riempite

*# Compute the CL, UCL and LCL for Xbar and S*

*Xbar\_CL* = *mu*

*Xbar\_UCL* = *mu* + *K*\**std*/np.*sqrt*(*n*)

*Xbar\_LCL* = *mu* - *K*\**std*/np.*sqrt*(*n*)

*S\_CL* = *c4* \* *std*  *# Expected value of s (sample standard deviation)*

*S\_UCL* = *c4* \* *std* + *K* \* np.*sqrt*(1 - *c4*\*\*2) \* *std*

*S\_LCL* = *c4* \* *std* - *K* \* np.*sqrt*(1 - *c4*\*\*2) \* *std*

if *S\_LCL* < 0:

*S\_LCL*=0

for *i* in range(5):

*data\_XS*.*loc*[*i*] = [*Xbar\_CL*, *Xbar\_UCL*, *Xbar\_LCL*, *S\_CL*, *S\_UCL*, *S\_LCL*] #riempio colonne con un ciclo for

*sample\_mean*=np.array([4.0738, 3.9406, 4.0430, 3.9968, 3.8290])

*sample\_std*=np.array([0.1638, 0.2148, 0.1711, 0.1312, 0.1555])

*data\_XS*['sample\_mean']=*sample\_mean* #creo nuove colonne

*data\_XS*['sample\_std']=*sample\_std*

-**droppare colonna** droppa colonna:

*data*.drop(*columns*=['X2'], *inplace*=True)

- **crea pandas dataframe** from **scratch** a partire da un dictionary:

basta sostanzialmente creare un dictionary in cui le **keys sono i nomi delle colonne** e i value sono delle liste e quindi i valori vanno messi tra parantesi quadre

esempio1 (inserendo I dati):

*data\_dictionary* = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],

        'Age': [25, 30, 35, 40],

        'Height': [5.2, 6.0, 5.6, 5.10],

        'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}

*data* = pd.DataFrame(*data\_dictionary*)

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

esempio2 (partendo da np.array tradizionali):

*array1* = np.array([1, 2, 3, 4, 5])

*array2* = np.array([6, 7, 8, 9, 10])

*array3* = np.array([11, 12, 13, 14, 15])

*# Create a DataFrame from the numpy arrays*

*df* = pd.DataFrame({ 'Column1': *array1*, 'Column2': *array2*, 'Column3': *array3* })

-**sottrarre** un dataframe ad un altro:

result\_df = df1.sub(df2)

-**sostituire** con un valore x, il valore max:

*Max\_index= data*['Water content'] == *data*['Water content'].max()

*data*.*loc*[*Max\_index*,'Water content'] = x

## -**sostituire** con un valore x, un valore di cui conosco la posizione nel dataframe (eg. quarta riga)

*data\_corrected*.*loc*[3, 'Water content'] = x *#direct replacement*

#x potrebbe essere np.*nan*, ad esempio se magari devo rimuovere un OOC (se non uso dummyapproach) o un sospetto outlier che farebbe rifiutare H0 allo shapiro

-**droppare** o **eliminare** o **rimuovere** un dato particolare (**outlier**), qui ad esempio si elimina il 30esimo:

*data\_out* = *data*.drop(*index*=29) #nota che avremo un nuovo dataset *data\_out*!!!!!

Qui dovevo poi fare uno shapiro, perciò sta cosa è equivalente a metterci un NaN e usare **la mia** mf.shapiroqq (boxplot non funzia se ho dei NaN però)

-Generare **random samples**  da una determinata distribuzione: C:\Users\alyuk\Desktop\Polimi\2S2A\QDA\lab\_py\_qda\introduction\Exercise\_class1\_PART3.ipynb

## -**esempio creazione di vettori di dati specifici** in base ai valori assunti in determinate colonne a partire dal dataset originali:

*Filtra le righe in cui Step è uguale a 4.*

*Filtra le righe in cui dummy\_OOC è uguale a 0.*

*Seleziona solo la colonna X di queste righe filtrate:*

*datastep4* = *data*[(*data*['Step'] == 4) & (*data*['dummy\_OOC'] == 0)][['X']]

*data\_other\_steps* = *data*[(*data*['Step'] != 4) & (*data*['dummy\_OOC'] == 0)][['X']]

#in questo caso escludo in entrambi l’OOC

-esempio di **dot product (**var linear combination e ottenere gli scores di una pc a mano**)**

*#eg. So che V(a'x)=a'Ra, voglio* **calcolare la varianza di questa combinazione lineare** *a'x*

*a* = np.array([[0.5], [0.5], [0.5], [0.5]]) ***#column vector***

*a\_t*=np.transpose(*a*) *#row vector*

*R\_times\_a*=np.dot(*R*, *a*)

*var\_lin\_comb\_naive*= np.dot(*a\_t*, *R\_times\_a*)

print(float(*var\_lin\_comb\_naive*[0,0])) #accedere all’elemento 1,1 della matrice

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

#eg. lmbda1=V(v1’x)=v1’Rv1, voglio **calcolare a mano il lambda**

*v1* = np.array([[-0.4964], [-0.6674], [-0.4577], [-0.3145]]) *#first eigenvector*

*v1\_t*=np.transpose(*v1*) *#row vector*

*R\_times\_v1*=np.dot(*R*, *v1*)

*lmbda1*= (np.dot(*v1\_t*, *R\_times\_v1*))

print(float(*lmbda1*[0,0]))

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

#altro esempio: voglio ottenere gli **scores della prima pc a mano** avendo

#z1=X\*v1 dove X è il dataset centrato ecc mxp e v1 è il vettore colonna px1 che identifica la prima PC

*mu*=np.array([10,82,1500])#media delle variabili originali

#se mu non fosse np.array perché già creato come pandas series basta usarlo come input per la funzione np.array e lo trasformo easy

*std*=np.array([3.1,4.5,4.9])#std delle variabili originali

*data\_std* = (*data*.*iloc*[:,0:3]-*mu*) / *std* ])#std dei dati originali

*u1*=np.array([[0.65],[-0.19],[0.72]]) #eigenvector della prima pc ***(column vector)***

*lambda1*=1.811

*scores\_1*=np.dot(*data\_std*, *u1*) # mxp x px1 = mx1, **in python** size: (m,1) eg. (50,1)

#ora lo flatteno se no dà problemi (da column np vector a normalissimo np array) per la creazione del pandas dataframe:

*scores\_1*=*(scores\_1)*.flatten() # **in python** size: (m,) eg. (50,)

*#se voglio un df:*

*scores\_df1*=pd.DataFrame({'scores1': *scores\_1*})

# INFERENTIAL STATISTISTICS

Tecnicamente per fare questi test va verificata la normalità sul o sui vettori di dati usati, ma anche l’indipendenza con un runs test (anche se nel capitolo della inferential statistics è stato dato per scontato quando abbiamo a che fare con le time-series tocca checckarla)

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

## SHAPIRO

-**Shapiro test + qq plot**:

mf.shapiroqq (*data*['nomecolonna'],  *alpha*=0.05*, multiobservation*='no')

attenzione:

1)se devo dar dentro il risultato di un processo di differenziazione devo dare **data[‘diff\_order’][order:]** (i NaN nell’ultimo aggiornamento vengono dropatti in automatico quindi forse manco serve, ma va be)

2)se *data* è il un dataset mxn con **n >1** tipico dei CCs NON multivariati (**p=1** quindi), allora non devo dar dentro il data[‘colonna’] ma proprio il dataframe *data* con ***multiobservation* =’yes’**, mentre se è un problema con n=1 devo dar dentro *data* [‘nomecolonna’]

## 1S-Z-TEST

-**z-test** per la media a partire da campioni fittizi (generati da una normale): Exercise\_class2\_PART2.ipynb (cerca “finta”)

-Formula **z statistic** con mu0 che è quella della H0 e x1 è il vettore di sample

*Z\_0* = (np.mean(*x1*) - *mu0*) / (*sigma* / np.sqrt(*n*))

-formula **p\_value** **two-sided** z-test 1 sample:

*pval* = 2 \* ( 1 - stats.norm.cdf(np.abs(*Z\_0*)) )      *#attention: bilateral rejection*

print('p-value = %.3f' % *pval*)

if *pval* < *alpha*:

    print("Reject the null hypothesis.")

else:

    print("Accept the null hypothesis.")

-formula per *double* **o two-sided confidence interval z** equivalente a z-test **1 sample**:

*alpha* = 0.05   *# significance level*

*z\_alpha2* = stats.norm.ppf(1-*alpha*/2)

*CI* = [np.mean(*x1*) - *z\_alpha2* \* *sigma*/np.sqrt(*n*), np.mean(*x1*) + *z\_alpha2* \* *sigma*/np.sqrt(*n*)]

print('Confidence interval: %.3f, %.3f' % (*CI*[0],*CI*[1]))

-fare un **powerplot z-test per 1 sample** (tecnicamente sono presenti tutti e 3 i possibili test grazie a *sided* che può essere no, upper o lower e va anche specificato se il mu1 ipotizzato per il computo della potenza è maggiore o no, tramite true o false):

Chiaramente se voglio posso recuperarci dentro tutte le formule

mf.power\_Z\_single\_plot (*n*, *delta*, *alpha*, *mu0*, *sigma*, *sided*, *mu1\_greater*)

## 1S-T-TEST

-Formula **t statistic** e valore critico t\_alpha per 1 sample *upper* tailed t-test:

*t\_0* = (np.mean(*x1*) - *mu0*) / (np.std(*x1*,*ddof*=1) / np.*sqrt*(*n*))

*t\_alpha* = stats.*t*.ppf(1-*alpha*, *n*-1) *#valore critico per la t*

if *t\_0* > *t\_alpha*: *#controlliamo se la statistica calcolata sta a destra ovviamente*

    print('Reject the null hypothesis at alpha = %.2f' % *alpha*)

else:

    print('Accept the null hypothesis at alpha = %.2f' % *alpha*)

**-**codice per **t-test 1 sample:**

*# Use the built-in function to make the t-test*

*t\_0*, *pval* = stats.ttest\_1samp(*x1*, *mu0*, *alternative*='greater')

*#nota che il default per alternative è 'different'*

print('Test statistic t\_0 = %.3f' % *t\_0*)

print('p-value = %.3f' % *pval*)

-**lower confidence interval** **t-test** equivalente ad **upper tailed t-test** (se **mu0** è sopra Lower\_limit allora devo accettare **H0** perchè in quel caso è considerato improbabile che mu>mu0, se no rifiuto ossia considero mu>mu0):

*Lower\_limit* = np.mean(*x1*) - *t\_alpha* \* np.std(*x1*,*ddof*=1)/np.*sqrt*(*n*)

print('Confidence interval (lower bound): %.3f' % *(Lower\_limit)*)

-**two-sided confidence interval** **t-test** equivalente al t-test 1 sample (si può usare anche per la differenza nei paired t-test, il dato da prendere deve essere la differenza):

*CL* = 0.98       *# Confidence level*

*alpha* = 1 - *CL*  *# Significance level*

*# Compute the two-sided confidence interval*

*t\_alpha2* = stats.*t*.ppf(1 - *alpha* / 2, *df*)

*CI\_b* = [*data*['Water content'].mean() - *t\_alpha2* \* *data*['Water content'].std() / np.*sqrt*(*n*),

*data*['Water content'].mean() + *t\_alpha2* \* *data*['Water content'].std() / np.*sqrt*(*n*)]

print('Two-sided confidence interval (%.2f): [%.3f, %.3f]' % (*CL*, *CI\_b*[0], *CI\_b*[1]))

#oppure:

*CI*= stats.t.interval(*CL*, df= *df*, loc= *data*['Water content'].mean(), scale= *data*['Water content'].std())

-**potenza** t-test **beta** t-test bilaterale:

*t\_alpha2*=stats.*t*.ppf(1-alpha/2,n-1)

*beta*=stats.*t*.cdf(*t\_alpha2*-delta\_piccolo\*np.*sqrt*(n),n-1)-stats.*t*.cdf(-*t\_alpha2*-delta\_piccolo\*np.*sqrt*(n),n-1)

print(*beta*)

## CHI2-TEST:

-**chi2-test** (testo se la var del mio sample *var\_data* è uguale, maggiore o minore a una certa ***var\_H0***)

[*chi2*,*critical\_values*,*p\_value*]=mf.chi2\_test(*var\_data*,***var\_H0***,*dof*,*direction*= 'greater',*alpha*=0.05) #'greater'vuol dire H1: *var\_data* > *var\_H0*

*#dof generalmente è n-1*

-**two-sided confidence interval** **chi2** **ossia confidence interval** **stdev** (chi2-test):

*CL* = 0.98       *# Confidence level*

*alpha* = 1 - *CL*  *# Significance level*

*# Compute the two-sided CI on the variance*

*chi2\_1* = stats.*chi2*.ppf(*alpha* / 2, *df*)

*chi2\_2* = stats.*chi2*.ppf(1 - *alpha* / 2, *df*)

*CI\_var* = [*df* \* *data*['Water content'].var() / *chi2\_2*,

*df* \* *data*['Water content'].var() / *chi2\_1*]

*CI\_stdev\_d* = np.*sqrt*(*CI\_var*)

print('Two-sided CI on the standard deviation (CL = %.2f): [%.3f, %.3f]' % (*CL*, *CI\_stdev\_d*[0], *CI\_stdev\_d*[1]))

-**Potenza chi2-test bilaterale beta chi2-test bilaterale**

*chi2\_upper\_cv*=stats.*chi2*.ppf(1-alpha/2,n-1)

*chi2\_lower\_cv*=stats.*chi2*.ppf(alpha/2,n-1)

*beta*=stats.*chi2*.cdf(*chi2\_upper\_cv*/lmbda,n-1)-stats.*chi2*.cdf(*chi2\_lower\_cv*/lmbda,n-1)

*#dove lambda=sigma1^2/sigma0^2!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*

*power*=1-*beta*

print(*power*)

## 2S-T-TEST PAIRED

-ricorda che nei **dati** **paired**, **shapiro** lo applichiamo alla differenza! **Se invece le pop sono indipendenti** (2S-T-TEST NOT PAIRED) lo applichiamo a entrambi i samples delle 2 popolazioni e basta (NON sulla differenza)!

-codice per **t-test paired sample** in 1 colpo solo (**sulla** **differenza**), in questo caso posso testare molto facilmente una qualsiasi **d0**, che metto nella *popmean.*

*t0\_stats*, *p\_value\_t0\_stats* = stats.ttest\_1samp(*data*['d'], *popmean* = **0**, *alternative*='greater') #sto testando che la differenzia sia maggiore di **0**

print('t-statistic from stats.ttest\_1samp: %.3f' % *t0\_stats*)

print('p-value from stats.ttest\_1samp: %.3f' % *p\_value\_t0\_stats*)

-codice per **t-test paired sample** in 1 colpo solo (**sui dati originali**):

*t0\_stats\_trel*, *p\_value\_t0\_stats\_trel* = stats.ttest\_rel(*data*['before'], *data*['after'], *alternative*='greater') #in questo caso praticamente **d0=0 per forza!**

print('t-statistic from stats.ttest\_rel: %.3f' % *t0\_stats\_trel*)

print('p-value from stats.ttest\_rel: %.3f' % *p\_value\_t0\_stats\_trel*)

-**lower confidence interval differenza paired sample**, equivalente ad upper tailed t-test, paired sample:

*# Calculate the lower bound of the one-sided confidence interval*

*t\_alpha* = stats.*t*.ppf(1 - *alpha*, *df*) *#valore critico t*

*CI\_lower* = *data*['d'].mean() - *t\_alpha* \* *data*['d'].std() / np.*sqrt*(*n*)

print('Lower bound of the one-sided confidence interval: %.3f' % *CI\_lower*)

## F-TEST

-controllare NaN

*data*.isna().any

-**splittare** dataset o dividere dataset in base alle colonne in due subframe e rimuovere NaN eliminare NaN togliere **NaN** (in questo caso erano presenti solo in ‘supp1’:

*data1* = *data*['supp1']

*data2* = *data*['supp2']

*data1* = *data1*.dropna() #in questo caso mi serviva per un F-test (vedi sotto)

-**doppio boxplot** ossia boxplot di due colonne, due features, due variabili (può essere utile *prima* del test F di comparazione delle varianze)

plt.boxplot([*data1*, *data2*], *labels*=['Supplier 1', 'Supplier 2'])

plt.show()

-**two-sided confidence interval F-test ossia confidence interval varianze** equivalente al bilateral **F-test** per le varianze (post verifica della Normalità) e **p\_value** corrispondente alla F0

*CL* = 0.95       *# Confidence level*

*alpha* = 1 - *CL*  *# Significance level*

*F0* = *data1*.var()/*data2*.var() # F statistic

*(Gli n NON DEVONO ESSERE PER FORZA UGUALI, tra l’altro il test usa F0)*

*df1* = *n1* - 1 *# degrees of freedom for supplier*

*df2* = *n2* - 1 *# degrees of freedom for supplier 2*

*#attenzione ai df che sono invertiti nell'IDC!!!!!!!*

*CI* = [*F0* \* stats.*f*.ppf(*alpha*/2, *df2*, *df1*), *F0* \* stats.*f*.ppf(1-*alpha*/2, *df2*, *df1*)]

print('Confidence interval on the ratio of variances (CL = %.3f): [%.3f, %.3f]' % (*CL*, *CI*[0], *CI*[1]))

*p\_value\_left* = 2 \* stats.*f*.cdf(*F0*, *df1*, *df2*)

*p\_value\_F0=* *2\* min(p\_value\_left, 1 - p\_value\_left)*

print('F stat is: %.3f' % *F0*)

print('p-value for F-test for equal variances: %.3f' % *p\_value\_F0*)

-**two-sided confidence interval stdev due popolazioni** (chi2) per vedere se si intersecano o no (equivalente al bilateral F test):

CI\_sigma\_1 = np.sqrt([(df1 \* data1.var())/stats.chi2.ppf(1-alpha/2, df1), (df1 \* data1.var())/stats.chi2.ppf(alpha/2, df1)])

CI\_sigma\_2 = np.sqrt([(df2 \* data2.var())/stats.chi2.ppf(1-alpha/2, df2), (df2 \* data2.var())/stats.chi2.ppf(alpha/2, df2)])

print('Confidence interval on the standard deviation for supplier 1 (CL = %.3f): [%.3f, %.3f]' % (CL, CI\_sigma\_1[0], CI\_sigma\_1[1]))

print('Confidence interval on the standard deviation for supplier 2 (CL = %.2f): [%.3f, %.3f]' % (CL, CI\_sigma\_2[0], CI\_sigma\_2[1])

-calcolare **beta F-test** **Potenza F-test** fissato un certo ratio da detectare delle due varianze (si vede che dipende solo dalla size dei due samples, da alpha e ovviamente dal ratio):

*ratio* = 1.5 *# sigma1^2/sigma2^2*

*beta* = stats.*f*.cdf(stats.*f*.ppf(1-*alpha*/2, *df1*, *df2*)/*ratio*, *df1*, *df2*) - stats.*f*.cdf(stats.*f*.ppf(*alpha*/2, *df1*, *df2*)/*ratio*, *df1*, *df2*)

print('Power of the test: %.3f' % (1-*beta*))

## 2S-Z-TEST

-**z statistic 2 indipendent sample** **e z-test independent sample** con **p\_value** corrispondente:

standard\_error\_mean\_diff= np.sqrt( *Var1*/n1 + *Var2*/n2 )

Z = (data1.mean() - data2.mean()) / standard\_error\_mean\_diff

p\_value = 2 \* ( stats.norm.cdf (-abs(Z)) )

print('The p-value is: %.3f' % p\_value)

## 2S-T-TEST

-**t statistic 2 indipendent sample** e **t-test** **indipendent sample** con **p\_value** corrispondente (**Varianze uguali**):

*s\_p* = np.sqrt((*data1*.var() \* *df1* + *data2*.var() \* *df2*) / (*df1* + *df2*))

T = (data1.mean() - data2.mean()) / (s\_p \* np.sqrt(1/n1 + 1/n2))

p\_value = 2 \* ( stats.t.cdf (-abs(T), df1 + df2) )

print('The p-value is: %.3f' % p\_value)

-**t statistic 2 indipendent sample** e **t-test** **indipendent sample** con **p\_value** corrispondente (eg. **Varianze diverse**):

s1= *data1*.std()

s2= *data2*.std()

numerator = (s1\*\*2 / n1 + s2\*\*2 / n2)\*\*2

denominator = ((s1\*\*2 / n1)\*\*2 / (n1 - 1)) + ((s2\*\*2 / n2)\*\*2 / (n2 - 1))

*df* = numerator / denominator

standard\_error\_mean\_diff= np.sqrt( (*data1*.var()/n1) + (*data2*.var())/n2 )

T = (data1.mean() - data2.mean()) / standard\_error\_mean\_diff

p\_value = 2 \* ( stats.t.cdf (-abs(T), *df*) )

print('The p-value is: %.3f' % p\_value)

-codice per fare il t**-test 2 indipendent sample** (specifica *equal\_var* come False o True !!!)

*t0*, *p\_value\_t0* = stats.ttest\_ind(*data1*, *data2*, *equal\_var*=True, *alternative*='two-sided')

print('t-test: t0 = %.3f' % *t0*)

print('p-value for t-test: %.3f' % *p\_value\_t0*)

-**confidence interval sample mean difference** cioè il CI **equivalente al t-test 2 indipendent sample (var uguali)**, al 99% qui:

CL = 0.99      *# Confidence level*

alpha = 1 - CL *# Significance level*

*# Compute the pooled standard deviation*

s\_p = np.sqrt((data1.var() \* df1 + data2.var() \* df2) / (df1 + df2))

*# Compute the CI on the difference of means*

CI = [data1.mean() - data2.mean() - stats.t.ppf(1 - alpha/2, df1 + df2) \* s\_p \* np.sqrt(1/n1 + 1/n2),

        data1.mean() - data2.mean() + stats.t.ppf(1 - alpha/2, df1 + df2) \* s\_p \* np.sqrt(1/n1 + 1/n2)]

print('The confidence interval on the difference of means is: [%.3f, %.3f]' % (CI[0], CI[1]))

-**confidence interval sample mean difference** cioè il CI **equivalente al t-test 2 indipendent sample (var DIVERSE)**, al 99% qui:

CL = 0.99      *# Confidence level*

alpha = 1 - CL *# Significance level*

s1= *data1*.std()

s2= *data2*.std()

numerator = (s1\*\*2 / n1 + s2\*\*2 / n2)\*\*2

denominator = ((s1\*\*2 / n1)\*\*2 / (n1 - 1)) + ((s2\*\*2 / n2)\*\*2 / (n2 - 1))

*df* = numerator / denominator

standard\_error\_mean\_diff= np.sqrt( (*data1*.var()/n1) + (*data2*.var())/n2 )

*# Compute the CI on the difference of means*

CI = [data1.mean() - data2.mean() - stats.t.ppf(1 - alpha/2, *df*) \* standard\_error\_mean\_diff,

        data1.mean() - data2.mean() + stats.t.ppf(1 - alpha/2, *df*) \* standard\_error\_mean\_diff]

print('The confidence interval on the difference of means is: [%.3f, %.3f]' % (CI[0], CI[1]))

-calcolare power o **Potenza t-test 2** **sample indipendenti** (caso varianza uguale):

Talpha2 = stats.*t*.ppf(1 - alpha / 2, df1+df2)

*delta*=np.linspace(0,10,100) *#delta assoluto*

*S1*=data1.var()

*S2*=data2.var()

*SP2* = ((df1)\**S1* +(df2)\**S2*)/(df1+df2) #pooled variance

*standard\_error\_of\_mean\_difference*=np.*sqrt*(*SP2*)\*np.*sqrt*((1/n1)+(1/n2))

*power*= 1 - stats.*t*.cdf(Talpha2 - *delta*/*standard\_error\_of\_mean\_difference*, df1+df2) + stats.*norm*.cdf(-Talpha2 - *delta*/*standard\_error\_of\_mean\_difference*, df1+df2)

plt.plot(*delta*, *power*)

plt.xlabel("delta")

plt.ylabel("power")

plt.grid(True)

plt.show()

#se varianze sono diverse allora basta prendere le formule dal codice appena sopra a questo e ricordarsi che i df adesso sono dati da quella formula complicata (modifcare quindi anche la formula di Talpha2)

# DATA MODELLING:

-**plot time series** singola dal dataset (monovariato),

mf.single\_ts\_plot(*data*['nomecolonnax'])

-plot time series multiplo:

mf.multiple\_ts\_plot(*data*,['x1','x2'])

-conta dei runs, expected value del numero di runs, Confidence interval per il numero di run (assumendo H0 vera, ossia che la distribuzione sia random) equivalente al runs test: C:\Users\alyuk\Desktop\Polimi\2S2A\QDA\lab\_py\_qda\data\_modelling1\ExerciseClass3\_part1.ipynb

-**Runs test** su una colonna del dataset e -plot **ACF** **PACF**:

mf.runs(*data*['nomecolonna'], *alpha*=0.05)

mf.acfpacf(*data*['nomecolonna'])

-**istogramma** **boxplot histogram boxplot**

mf.histandbox(*data*['nomecolonna'],*multiobservation*='no')

#metti *multiobservation*='yes' **solo** se p=1 e **n>1**, in quel caso fa lui lo stack e plotta la distribuzione dei dati stacked in una sola colonna.

Se ho più variabili (ossia p>1) viene fuori una cosa SENZA SENSO!!!In quel caso devo fare gli istogrammi singoli!!!

## Boxcox transformation + shapiro su dati trasformati

Immagine che contiene testo, Carattere, schermata, Elementi grafici

Descrizione generata automaticamente

[*data\_norm*,*lmbda*]=mf.boxcox(*data*, *nomecolonnadati \_dati*,*approx\_to\_0\_if\_nec*='False',*simplify*='False',*add\_cost*=0)

-**Bartlett test** ad un lag specifico (stampa anche risultato del test a parole):

**salta è sempre 0** (anche quando fatto sui residui) **TRANNE** nel caso in cui io sia nel processo di **differenziazione** (per poi verficare se modellizzare o no) e allora metto dentro l’ordine di differenziazione, questo perché la time series differenziata su cui fare il test ha almeno un NaN!

mf.bartlett\_test (*data\_norm*, *lag\_test*, *alpha*=0.05, *salta*=0)

-**LBQ test** (FINO a lag L)

Nota che non c’è un’alpha (intepretazione sta a me)

*# Generally speaking: how many lags?*

*# Rule of thumb: L<=sqrt(n)*

from statsmodels.stats.diagnostic import acorr\_ljungbox

*lbq\_test* = acorr\_ljungbox(*data\_norm*, *lags*=[*L*], *return\_df*=True)

print('LBQ test statistic at lag %d = %f' % (*L*, *lbq\_test*.*loc*[*L*,'lb\_stat']))

print('LBQ test p-value at lag %d = %f' % (*L*, *lbq\_test*.*loc*[*L*,'lb\_pvalue']))

## two-sided prediction interval distributional model next outcome process distributional model (no time series model) all’1-alpha %:

[*pred\_lo*, *pred\_up*] = mf.pred\_interval\_DM (*data\_norm*, *alpha*=0.05)

### -backtransformation (box-cox) + ORIGINAL prediction interval

Immagine che contiene testo, Carattere, schermata, Elementi grafici

Descrizione generata automaticamente

[*pred\_lo\_ORIG*, *pred\_up\_ORIG*] = [(*pred\_lo*\**lmbda*+1)\*\*(1/*lmbda*),(*pred\_up*\**lmbda*+1)\*\*(1/*lmbda*)]

print('Two-sided prediction interval for original data: [%.3f %.3f]' % (*pred\_lo\_ORIG*, *pred\_up\_ORIG*))

-plot **ACF** **PACF**:

mf.acfpacf(*data*['nomecolonna'])

-gapping e plot: (eg. Gap\_size=6 o gap\_size=len(data)/10

gap\_data=mf.gapping(*data*, *gap\_size*)

-batching e plot:

batch\_data= mf.batching(*data*, *column*, *batch\_size*)

-per fare uno scatter plot: (può essere usato anche tra una continua e una categorica come i batches)

-mf. scatter\_plot\_nice\_h(*data*, *nomecolonna1*, *nomecolonna2*, *highlight\_index*=None)

-**shapiro** **residui** +plots sui residui (verificare normalità, eventuali trend ecc) sui residui del modello:

Nota: sembra che *salta* vada messo diverso da 0 SOLO quando uso qda.ARIMA package, ad esempio se fitto un ARIMA(0,1,1) salta va messo a 1 a causa dell’operazione di differenziazione. Nota che dentro va il *model*!

mf.shapirplusresplots(*model, salta*=0) #p\_value viene stampato ma senza dirti risultato del test perché non c’è alpha

-per testare la **randomness** **residui** abbiamo gli altri strumenti come sempre (eg. Runs test) e devo accedere ai residui come *model.resid*

## Stuff modelling

-aggiungere lag1 al dataset\*

*data*['lag1'] = *data*['Ex4'].shift(1)

-aggiungere **annata** al dataset (che è la variabile\_t, classico regressore):

*data*['year'] = np.arange(1900, 1998)

-costruire una matrice\_di\_regressori a partire dai nomi delle colonne del dataset:

*x* = *data*[['Dummy\_Batch1', 'Dummy\_Batch2', 'Dummy\_Batch3', 'Dummy\_Batch4']] *#pandas dataframe 100x4*

*x* = sm.add\_constant(*x*) *#pandas dataframe 100x5*

-processo di **differenziazione** dalla A alla Z (restituisce la **diff**, mi fa il **plot** **della diff** da cui capisco se ho stazionarietà, mi fa il **runstest** mi plotta **acf e pacf** cosi capisco se devo procedere con modellizzazione eg. AR(1)) TRANNE EVENTUALI BARTLETT TEST SUCCESSIVI su specifici lag. Bisogna dare il dataset, il nome della collonna a cui sottrarre la collonna2 (solitamente lag1) e l’ordine che sto valutando della differenza. Ricorda che se mi fermo con la modellizzazione (ossia la diff viene random, senza lag che rompono le scatole) dovrò fare uno shapiro su Data[‘diff\_order’][order:],se no sui residui del modello.

*data*['diff\_order']=mf.diffprocess (*data*,*nomecolonna1*,*nomecolonna2*,*order*)

Esempio: Data[‘diff\_2’]=mf.diffprocess (*mydataframe*,*’diff1’*,*lag2*,*2*)

-***esempio*** di costruzione di matrice di regressione con creazione di una **dummy specifici giorni**, creazione di una variabile ‘Day’ (la variabile t) e poi **fitting con OLS, fitta modello**

*data*['dummy'] = np.zeros(len(*data*))

*data*['dummy'][17:22] = 1

*#create day of week variable*

*data*['Day'] = np.arange(1, len(*data*)+1)

import statsmodels.api as sm

import qda

*x* = *data*[['Day', 'dummy']] #se solo un regressore togli le doppie quadre e metti singole

*x* = sm.add\_constant(*x*)

*y* = *data*['Temp']

*model* = sm.OLS(*y*, *x*).fit()

qda.summary(*model*)

-plotta **y vs fits** (in y va una cosa tipo data[‘exe1’]):

plotyvsfits(*y*,*model*)

-extract **residual standard error**/deviation del modello varianza residui stdev residui deviazione standard residui MSE. In più è possibile estrarre un certo parametro e lo standard error associato volendo

*residual\_standard\_error* = np.*sqrt*(*model*.mse\_resid) #NOTA CHE HO GIA MESSO la #np.*sqrt*

*beta0*=model.params[0]

*std\_error\_beta0*=model.bse[0]

Per le prossime cose può aver senso vedere: C:\Users\alyuk\Desktop\Polimi\2S2A\QDA\lab\_py\_qda\data\_modelling1\_additional\ESE3\_es4\_v2.ipynb

## Confidence interval parametro modello

-metodo1 (mf): stampa il **Confidence interval** **parametro** **modello** associato ad un certo regressore equivalente ad un t-test di **significatività** per quel parametro (specifica regressore associato con il nome della colonna che è stata messa nella **matrice x** per fittare il modello). Stampa anche la **stima puntuale** e lo standard error del parametro:

mf.ci\_param\_model(*data*,*model*, *associated\_regressor*,*number\_of\_parameters\_of\_the\_model*, *alpha*=0.05)

-metodo2 (in questo caso il **regressore associato è 'lag1'**) :

*CI\_beta1* = *model*.conf\_int(*alpha*=0.05).loc['lag1']

print('The confidence interval for beta1 is [%.3f, %.3f]' % (*CI\_beta1*[0], *CI\_beta1*[1]))

## confidence interval e prediction interval

-calcola la **prediction** **modello** per il next process outcome e i relativi **confidence interval** **media modello** e **prediction interval modello**. Se ho più di un regressore metto in ordine tutto quello che mi serve nella parantesi di get\_prediction:

*last\_lag* = *data*[‘quality\_feature’].*iloc*[-1] #in questo caso -1 perché regressore era lag1 e cosi prendo effettivamente l’ultimo campione presente, se no altri due modi per prelevarlo:

#last\_lag= *data*['Ex4'].*iloc*[34] #checko l’indice e faccio a mano (n=35)

#last\_lag= *data*['Ex4'].*iloc*[*n*-1] #dove n=len(data)

*prediction\_df* = *model*.get\_prediction([1,*last\_lag*]).summary\_frame(*alpha*=0.05)

print(*prediction\_df*)

*#predizione puntuale in realtà (prima colonna aka mean)*

*#3a e 4a lower e upper bound del CI*

*#5a e 6a Lower e upper bound del PI*

----------------esempio con ARIMA(1,1,0) fatto a mano senza package (causa dummy)

*#qui che devo prendere l'ultimo di c\_diff, non l'ultimo c\_diff\_lag1!!!*

*last\_lag* = *c\_diff*.*iloc*[-1] #-1 perchè regressore è c\_diff\_lag1

*last\_dummy*=0

*prediction\_df* = *model*.get\_prediction([*last\_lag*,*last\_dummy*]).summary\_frame(*alpha*=0.05)

print(*prediction\_df*)

*#da sommare però visto che calcolo la diff c'è l'ultimo....?*

*#visto che delta\_Xt=Xt-Xt-1 avrei detto che avrei dovuto sommare l'ultimo lag1*

*#ma sono interessato a t=51*

*#delta\_X51=X51-X50 quindi X51=delta\_X51+X50* *e quello che ho stimato con le istruzioni sopra è delta\_X51, ora devo sommare X50 che chiamo tosum*

*tosum*=*c*.*iloc*[-1] #dove *c*=data[‘concentration’]

print(*tosum*)

*pred\_low*=-2.831+10.95

*pred\_up*=8.65+10.95

print(*pred\_low*)

print(*pred\_up*)

*--------------------------------------------------------------------------------*

#posso anche ACCEDERE AI CAMPI DEL summary\_frame()!

**summary\_frame** *= prediction.summary\_frame()*

***mean*** *= summary\_frame['mean']* **mean\_se** *= summary\_frame['mean\_se']* **obs\_ci\_lower** *= summary\_frame['obs\_ci\_lower']* **obs\_ci\_upper** *= summary\_frame['obs\_ci\_upper']* **mean\_ci\_lower** *= summary\_frame['mean\_ci\_lower']* **mean\_ci\_upper** *= summary\_frame['mean\_ci\_upper']*

*print("Predicted mean:", mean.values) print("Standard error of the mean prediction:", mean\_se.values) print("Confidence interval of the mean prediction:", (mean\_ci\_lower.values, mean\_ci\_upper.values)) print("Prediction interval for an individual point:", (obs\_ci\_lower.values, obs\_ci\_upper.values))*

## stepwise

-**stepwise** **regression**: qui mettiamo ‘both’ perché vogliamo forward+backward (**NON** METTERE LA COLONNA DI 1 AD X)

import qda

*stepwise* = qda.StepwiseRegression(*add\_constant* = True, *direction* = 'both', *alpha\_to\_enter* = 0.15, *alpha\_to\_remove* = 0.15)

*# Fit the model*

*model* = *stepwise*.fit(*y*, *X*)#*X* QUI NON DEVE AVERE LA COLONNA DI 1 perché c’è già

#*add\_constant* nelle opzioni di qda.StepwiseRegression

*results* = *model*.*model\_fit* *#regression result object*

qda.summary(*results*) #vediamo il modello finale e i p\_value

-**estrai fits stepwise, estrai residui stepwise, estrai parametri stepwise** dopo la stepwise regression:

*results* = *model*.*model\_fit* *#regression result object*

*residuals* = *results*.resid

*fits* = *results*.fittedvalues

*params* = *results*.params

## -prediction e prediction interval POST stepwise

(buona norma farsi un minidataframe di una riga con le colonne ordinate come si vuole e i dati esatti che si vuole per la prediction, in questo caso per l’anno 1998):

*mini\_data\_predict* = pd.DataFrame({'const': [1], 'Ex5\_lag1': [*data*['Ex5'].*iat*[-1]], 'year': [1998], 'Ex5\_lag4': [*data*['Ex5'].*iat*[-4]]})

*results* = *model*.*model\_fit* *#regression result object*



*prediction\_summary* = *results*.get\_prediction(*mini\_data\_predict*).summary\_frame(*alpha*=0.05)

print(*prediction\_summary*)

## Arima easy

-**ARIMA easy**: dal summary possiamo vedere gli LBQ test fatto sui residui del modello(p\_value grandi vuol dire che va bene)! Estrai residui modello, estrai fits modello con le dovute accortezze (quando si usa ARIMA package non vengono rimossi automaticamente i NaN all’inizio\*)

*y* = *data*['EXE2'] #variabile y, quella su cui voglio fittare

*p*=0

*d*=1

*q*=1

*model* = qda.ARIMA(*y*, *order*=(p,d,q), *add\_constant* = True) #eg. IMA(1,1)

qda.ARIMAsummary(*model*)

*residuals* = *model*.resid[np.max((*p*,*d*,*q*)):] #\*ATTENZIONE

*fits* = *model*.fittedvalues[np.max((*p*,*d*,*q*)):] #\*ATTENZIONE

## Batch exercise

-Crea **Batch\_variable** e **day variable**: Nell’esempio len(data)=100 e per data[‘Batch’] dobbiamo far si che si ripeta una situa tipo 12341234 ecc fino alla fine mentre per i giorni dobbiamo avere 111122223333 ecc Per la prima si usa np.tile (to tile significa impilare, impiliamo quindi un vettore da 1 a 4 per 25 volte) e per la seconda si usa np.repeat (creiamo il nostro vettore da a 1 25 e ripetiamo gli elementi dell’aray 4 volte

*# Create Batch variable*

*data*['Batch'] = np.tile(np.arange(1, 5), int(len(*data*)/4)) *#impila array ([1 2 3 4])* int(len(*data*)/4 *volte (ripeti* ***array*** *tot volte)*

#nota che in questo caso sarà inutile come variabile *data*['Day'] perché di fatto fittiamo un modello costante con la (o le) dummy associate ai batch (la variabile #*data*['Batch'] invece ci serve per capire le differenze tra i batch e quindi per capire che dummy variabile costruire se non vogliamo farle per tutte)

*data*['Day'] = np.repeat(np.arange(1, (len(*data*)/4)+1), 4) *#ripeti gli elementi the dell’array* np.arange(1, (len(*data*)/4)+1) ognuno 4 volte (step sottinteso a 1)

#(ripeti **ogni elemento dell’array** tot volte)

-plot **multiplo** **time series** **Batches**: (necessario per capire le differenze tra i batches, un'altra possibilità è usare gli scatter plot)

mf.scatter\_plot\_nice\_h(*data*,'Batch','Diameter')#Diameter è la quality feature

-plot multiplo quality feature divisa per batch, plot batch

plt.plot(*data*['Diameter'][*data*['Batch'] == 1], 'o:b', *label* = 'Batch 1') *#'o:b' cerchi blu*

plt.plot(*data*['Diameter'][*data*['Batch'] == 2], 's:r', *label* = 'Batch 2')

plt.plot(*data*['Diameter'][*data*['Batch'] == 3], 'D:g', *label* = 'Batch 3')

plt.plot(*data*['Diameter'][*data*['Batch'] == 4], '^:m', *label* = 'Batch 4')

plt.xlabel('Index')

plt.ylabel('Diameter')

plt.legend()

plt.title('Time series plot of Diameter')

plt.grid()

plt.show()

-**Crea dummy\_variable associata ai batch**

#esempio in cui voglio etichettare solo in corrispondenza del batch3

*data*['Dummy'] = np.tile(np.array([0, 0, 1, 0]), int(len(*data*)/4))

#oppure se voglio creare una dummy per ogni batch usa istruzione di questo tipo

*data*['Dummy\_Batch3'] = np.where(*data*['Batch']==3, 1, 0)

#oppure un altro esempio in cui devo mettere l’1 su molte righe:

*data*['Dummy\_batch1218'] = np.where(*data*['Day'].isin([10,11, 12, 13, 14, 15, 16, 17]), 1, 0)

## usare modello (no arima package)

**Usare modello** già fittato su altri dati: (se INVECE hai usato ARIMA package o serve nello specifico per i CC vedi file pt2)

*predictions\_ph2*=*model2*.predict(*data\_ph2*.*iloc*[1:,2:])

#l’input dell’istruzione sopra deve necessariamente essere la matrice dei NUOVI dati da buttar dentro nella formula pre-fatta del modello (qui parto da 1: perché il primo dato è un NaN). Quindi bisogna costruirsela sui dati di phase2 con tanto di eventuale boxcox, inserimento del lag1 ecc, tutto quello che serve

*residuals\_ph2*=*data\_ph2*['X'].*iloc*[1:]-*predictions\_ph2*

# PCA:

NOTA: la **normalità** non è richiesta nella PCA, ma un control chart lo richiederebbe! Non solo, richiederebbe iid tecnicamente (difficile che si vada a modellizzare post pca, generalmente si fa un IMR o un MVC se tengo più di una PC). E’ vero che le variabili vengono trasformate poi dalla PCA stessa, ma partire dove c’è assumption di NID per le variabili originali è buona cosa se poi devo fare CCs! In ogni caso poi **se si fa un CC andrà fatto il check delle assumptions (NID) sugli score che ci teniamo (terremo solo un certo numero di PC)!**

-fai **scatter plot** **variabili**:

mf.scatter\_b\_vars(*data*)

-calcola la **cov matrix e la corr matrix**:

*cov\_matrix* = *data*.cov()

print(*cov\_matrix*)

*corr\_matrix* = *data*.corr()

print(*corr\_matrix*)

#oppure per la corr matrix:

*data\_std* = (*data* - *data*.mean()) / *data*.std() #data.mean() e data.std() sono o pandas series oppure np.array piatti

#e poi calcoli cov matrix di *data\_std se serve*

-Codice per **passare da cov matrix (ad esempio se la cov matrix è data) a corr matrix**:

*std\_devs* = np.*sqrt*(np.diag(*S*))

*# Create the correlation matrix*

*correlation\_matrix* = *S*.copy()

for *i* in range(len(*S*)):

    for *j* in range(len(*S*)):

*correlation\_matrix*.*iloc*[*i*, *j*] = *S*.*iloc*[*i*, *j*] / (*std\_devs*[*i*] \* *std\_devs*[*j*])

print(*correlation\_matrix*)

-fai **PCA(**centrazione del dato compresa già**)** stampa tutto il necessario, fai anche gli **scores**, ritorna il **pca object** e lo **scores\_df**. Specifico in standardize se voglio farlo sulla correlation matrix (‘yes’). In ***sample\_to\_use*** se non ho distinzione tra dati **phase1** e **phase2** metto n=len(data), se no metto il **numero di sample su cui “stimo” gli eigenvectors** da cui poi invece **vengono calcolati TUTTI gli n scores** usando il modello fittato su un numero di dati pari a sample\_to\_use.

[pca,scores\_df]=mf.p\_pca(*data*, *sample\_to\_use*, *standardize*='no')

*#nota: gli autovettori sono messi qui sulle righe!!!! (le colonne rappresentano quindi le features originali)*

*#NOTA: può capitare che il dataset che mi viene dato abbia n>1 negli esercizi dei CC, al posto di data si metterà il df sample\_mean mxp*

-**Plot** **loadings** (ogni immagine è un loading vector):

mf.plotloadings(*pca*)

-Plot **eigenvalues** ossia **scree** **plot** e la **cumulative explained variance**:

mf.screeplotandcumexplvar(*pca*)

-Plot **scores scatterplot** along the first two PCs:

mf.scatter\_plot\_nice(*scores\_df*,'z1','z2')

-**Ricostruisci** o **reconstruct** I dati usando le prime k PCs a partire dalla pca:

*reconstructed\_data*=mf.reconstruct\_from\_pca\_obj(*data*,*pca*,*scores\_df*,*k*,*from\_standardize*='yes') #si sottintende che mean e stdev sono stimati dai dati

-Plot **scores scatterplot ph1-ph2** along the first two PCs **distinguishing among phase1data and phase2data**:

*# Create a scatterplot with the first 40 values of the two scores*

plt.scatter(*scores\_df*['z1'][:40], *scores\_df*['z2'][:40], *label*='First 40 values')

*# Add the rest of the values to the scatterplot with a different color*

plt.scatter(*scores\_df*['z1'][40:], *scores\_df*['z2'][40:], *color*='r', *label*='All the rest')

plt.xlabel('z1')

plt.ylabel('z2')

plt.title('Scatterplot of z1 vs z2')

plt.legend()

plt.show()

## SE ho mu e SIGMA (var-cov matrix) **date**:

C:\Users\alyuk\Desktop\Polimi\2S2A\**QDA\Exams ecc\TE\_from\_CQPP\17\_06\_2022\ese2.ipynb**