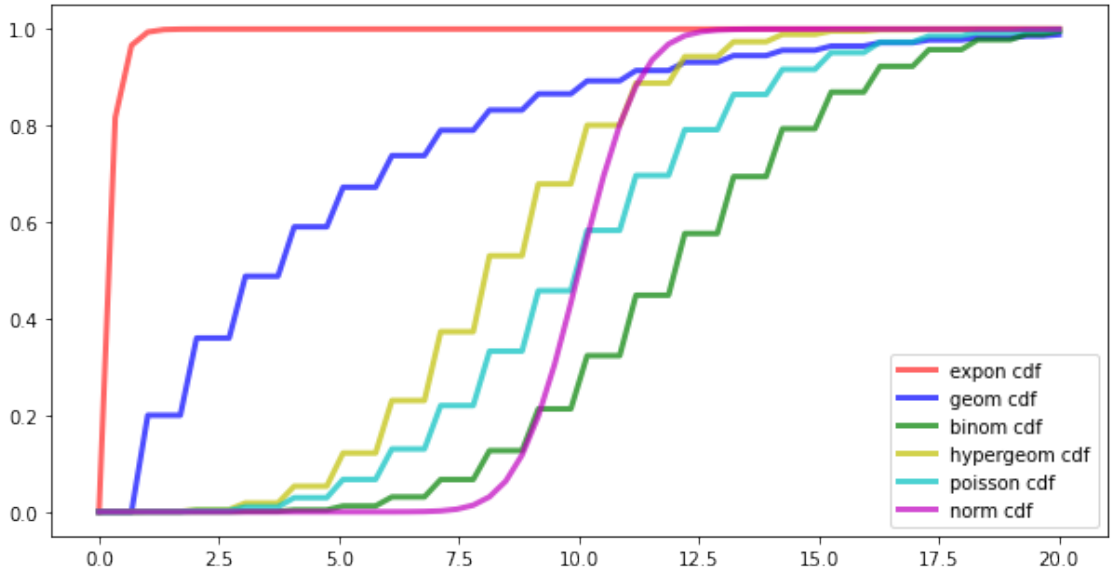
cheat2

June 15, 2023

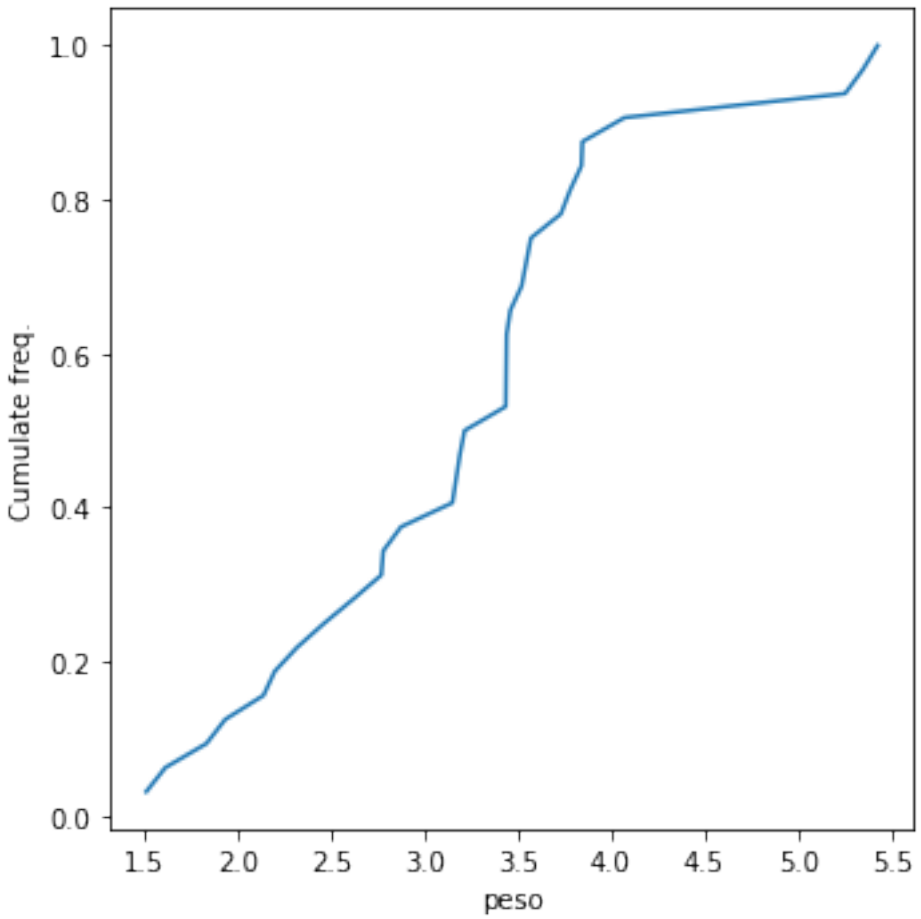
0.1 RECAP

[92]:
```
#Tracciare il grafico della funzione di distribuzione cumulativa == ripartizione
#di una variabile aleatoria esponenziale e di
#una variabile aleatoria geometrica, entrambe di parametro 0.1

#.pdf == probability density function
#.cdf == cumulative distribution function
import numpy as np
from scipy.stats import expon
from scipy.stats import geom
from scipy.stats import hypergeom
from scipy.stats import bernoulli
from scipy.stats import binom
from scipy.stats import poisson
from scipy.stats import norm
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1)
mean, var, skew, kurt = expon.stats(moments='mvsk')

x = np.linspace(0,20,60)

#lambda= 1/5
ax.plot(x, expon.cdf(x, scale =1/5),'r-', lw=3, alpha=0.6, label='expon cdf')
#p= 1/5
ax.plot(x, geom.cdf(x, 1/5),'b-', lw=3, alpha=0.7, label='geom cdf')
#p= 1/5
ax.plot(x, binom.cdf(x, len(x), 1/5),'g-', lw=3, alpha=0.7, label='binom cdf')
#hypergeom.cdf(x, M, n, N)
ax.plot(x, hypergeom.cdf(x,500,len(x),70),'y-', lw=3, alpha=0.7,␣
↪label='hypergeom cdf')
#poisson.cdf(x, mu)
ax.plot(x, poisson.cdf(x,10),'c-', lw=3, alpha=0.7, label='poisson cdf')
#norm.cdf(x, loc=0, scale=1) ho spostato la media a 10 (la metÃă del grafico␣
↪sulla x)
ax.plot(x, norm.cdf(x, 10),'m-', lw=3, alpha=0.7, label='norm cdf')
```

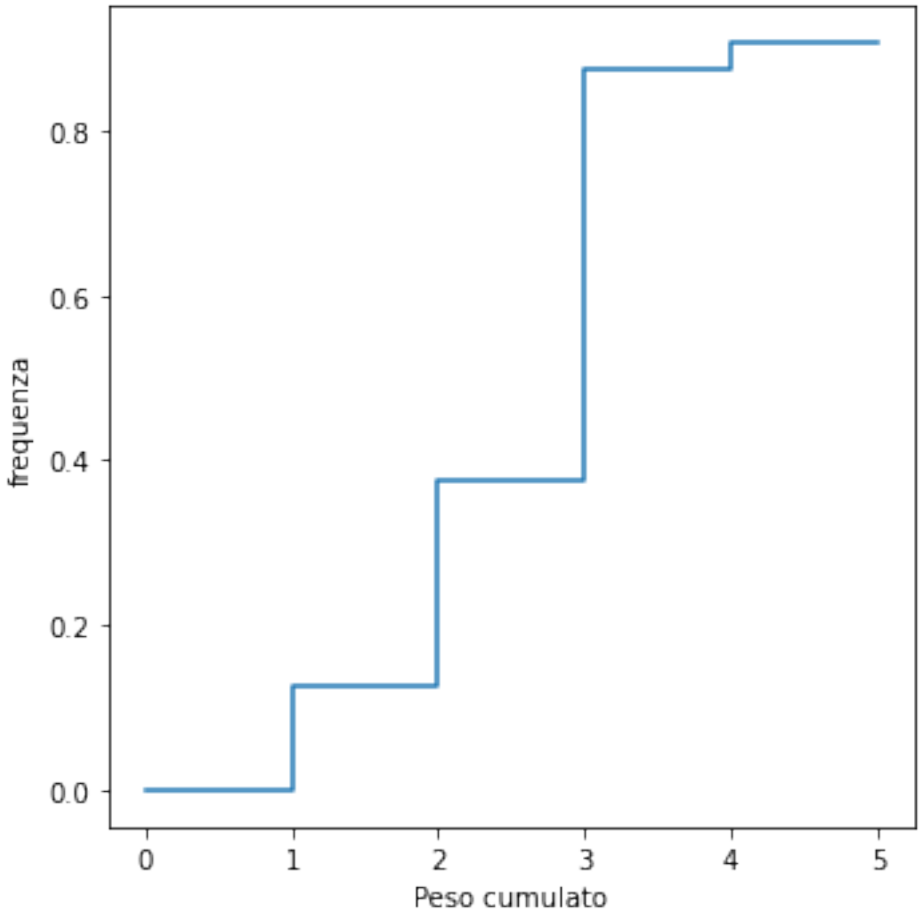
```
plt.legend(loc="lower right")
plt.show()
```



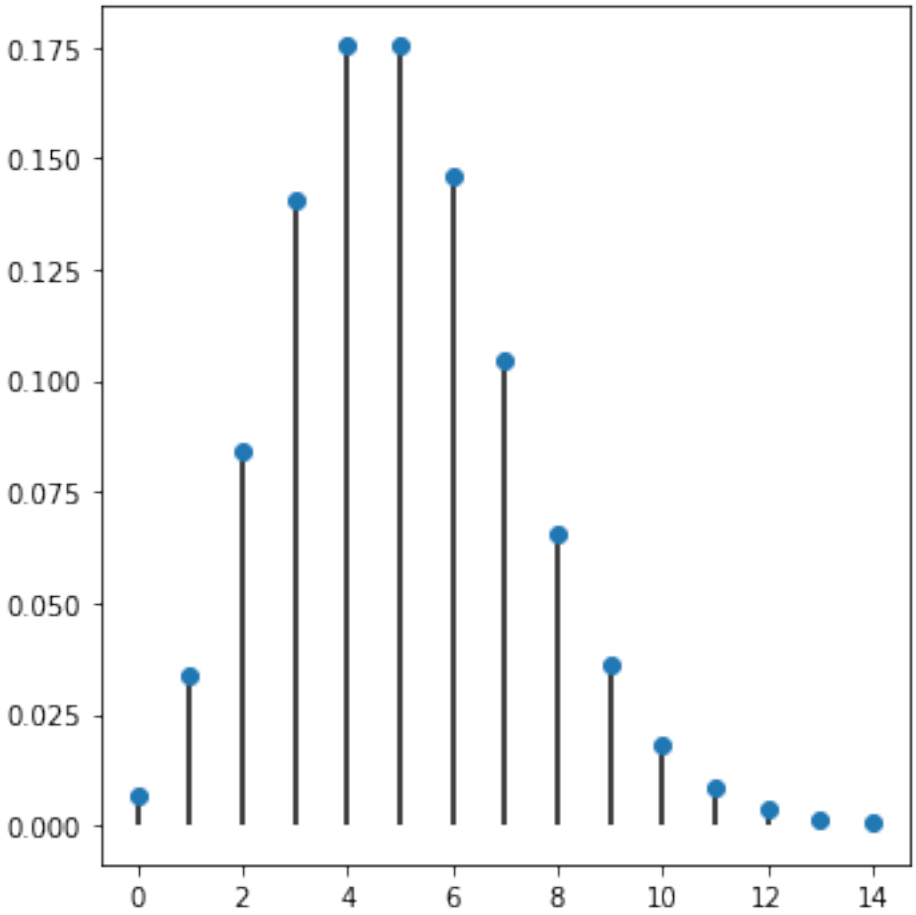
[136]:
```
#Se non conosco la distribuziuone uso:
plt.rc('figure', figsize=(5.0, 5.0))
first_app_relfreq_cumulate = (pd.crosstab(index=cars['peso'],
                              columns=['Cumulate freq.'],
                              colnames=[''],
                              normalize=True).cumsum())
#uso [0:6] o intervallo che voglio zoommare
first_app_relfreq_cumulate[0:6].plot(legend=False)
plt.ylabel("Cumulate freq.")
plt.show()
```



[126]:
```
#OPPURE
#Se voglio rappresentare la funzione cumulativa empirica (stima della
#funzione di ripartizione ma con intervalli e funzione indicatrice),
#posso usare .step al posto di .plot

import statsmodels.api as sm

ecdf = sm.distributions.ECDF(cars.peso)
x = np.arange(0, 6)
y = ecdf(x)
plt.step(x, y)  #USA PLT.STEP E NON PLT.PLOT PER RAPPRESENTARE GLI STEPS
plt.xlabel("Peso cumulato")
plt.ylabel("frequenza")
plt.show()
```



[118]:
```
#plottare la funzione di massa di un campione distribuito come *
l = 5
x = range(15)
X = poisson(l)
plt.vlines(x, 0, X.pmf(x))
plt.plot(x, X.pmf(x), 'o')
plt.show()
print("Se questo fosse un grafico di densitÃă sarebbe una popolazione␣
 ↪distribuita ")
print("approssimativamente normale con coda a destra")
```



Se questo fosse un grafico di densitÃă sarebbe una popolazione distribuita approssimativamente normale con coda a destra

[ ]:

0.2 Robette

[3]:
```
import csv
import pandas as pd

#funzia anche con file txt
cars = pd.read_csv("/home/gorl6/Desktop/STAT/Esami completi/mtcars.
 ↪txt",sep="\t",decimal=",")
```

[4]:
```
#quante osservazioni?
len(cars.index)
```

[4]: 32

[5]:
```
#Se outliers:
# utilizzo la mediana come indice di centralitÃ
# perchÃ¨ piÃ¹ robusto rispetto agli outliers presenti nel carattere.
# di conseguenza utilizzo il range interquartile
# per lo stesso motivo e ben si sposa con la mediana
# (non uso la varianza perchÃ¨ deriva dalla media
# che soffre gli outliers)

#moda mediana e media di un carattere
print(cars.peso.mean())
print(cars.peso.mode(dropna=True).values[0])
print(cars.peso.median())

#varianza, deviazione e range interquartile
print(cars.peso.var())
print(cars.peso.std())
print(cars.peso.quantile(0.75)-cars.peso.quantile(0.25))

#massimo e minimo
print(cars.peso.max())

# Coefficiente di variazione (o deviazione standard relativa)
coeff_var = cars.peso.std() / cars.peso.mean()
print(coeff_var)
```

```
3.21725
3.44
3.325
0.9573789677419354
0.9784574429896966
1.02875
5.4239999999999995
0.30412850819479265
```

[40]:
```
#tipi di caratteri: scalare, ordinale, categorico
#valori possibili per un categorico:
cars.marce.unique()
```

[40]: array([4, 3, 5])

[79]:
```
#frequenze realtive con bins
marce_fre_con= pd.crosstab(index=pd.cut(cars.peso,
                         bins=[1, 2, 3, 4, 5, 6]),
                          columns=["Frequenze relative"],
                            normalize=True  #if relative needed
```

```
                            #margins=True
                           )
marce_fre_con
```

[79]:
```
col_0   Frequenze relative
peso
(1, 2]             0.12500
(2, 3]             0.25000
(3, 4]             0.50000
(4, 5]             0.03125
(5, 6]             0.09375
```

[8]:
```
#accesso ai dati nella tabella
marce_fre_con.loc[1:,:][5].values[0]
#.loc trasforma in dataframe
#[5] screma per la colonna di nome 5
#.values toglie l'indice e restituisce un array
#[0] restituisce la testa
```

[8]:
```
5
```

[9]:
```
#applicare funzione per ottenere percentuali
import numpy as np
percent= (pd.crosstab(index=cars['trasmissione'],
                      columns=cars['marce'],
                      normalize=True,
                      margins=True
                      )
.apply(lambda p: np.round(100*p, 2))
.astype(str)
.apply(lambda s: s + '%'))
percent
```

[9]:
```
marce              3      4       5     All
trasmissione
0              46.88%  12.5%    0.0%  59.38%
1                0.0%  25.0%  15.62%  40.62%
All            46.88%  37.5%  15.62%  100.0%
```

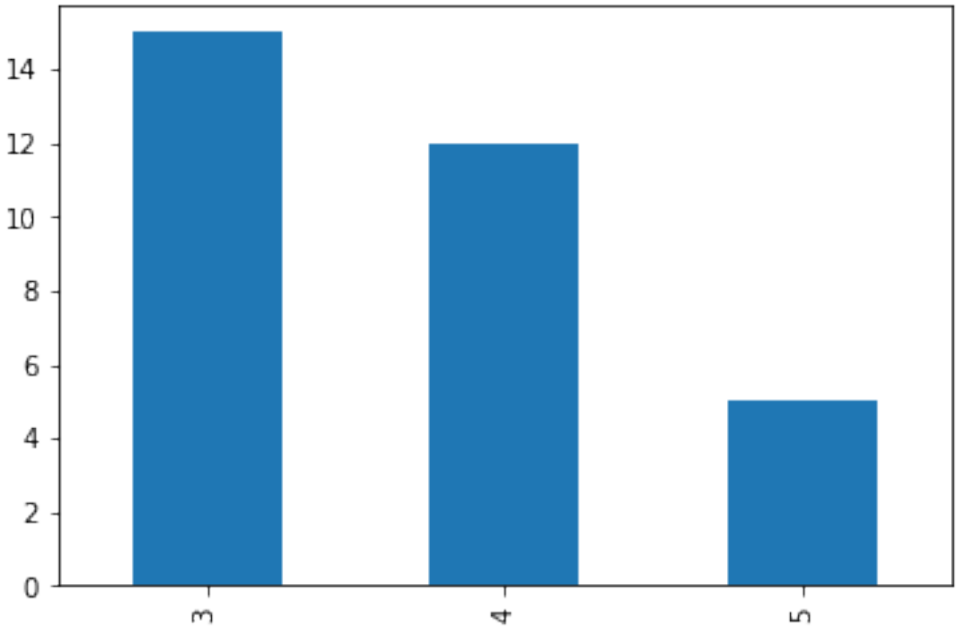
[10]:
```
#domanda tricky: tra i modelli che hanno tramissione manuale,
#quale percentuale ha 4 marce?
#Non posso vederlo dalla tabella perchÃ© i modelli che hanno
#4 marce sono in totale 13, di cui hanno 4 marce solo 8,
#ovvero 8/13= 0.61, 61%
#posso produrre un'altra tabella:
manuals= cars[cars['trasmissione']>0].trasmissione
percent2= (pd.crosstab(index=manuals,
```

```
                    columns=cars['marce'],
                    normalize=True,
                    margins=True
                    )
.apply(lambda p: np.round(100*p, 2))
.astype(str)
.apply(lambda s: s + '%'))
percent2
```

[10]:
```
marce              4       5     All
trasmissione
1              61.54%  38.46%  100.0%
All            61.54%  38.46%  100.0%
```

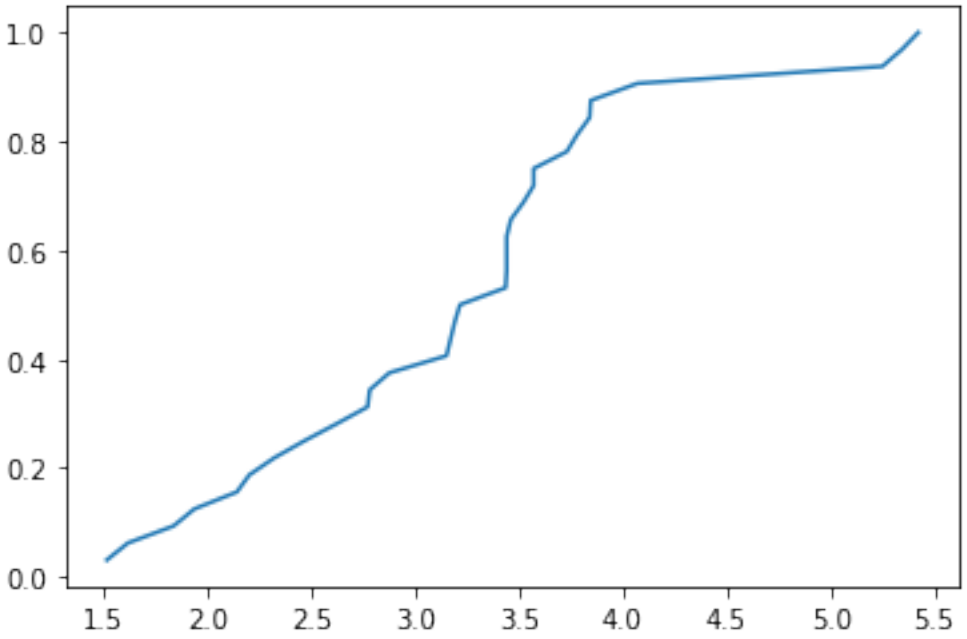
0.3 Single Character Analisys

[11]:
```
#opportuno grafico per i contenuti di un carattere
import matplotlib.pyplot as plt
cars.marce.value_counts(sort=False).plot.bar()
plt.show()
```

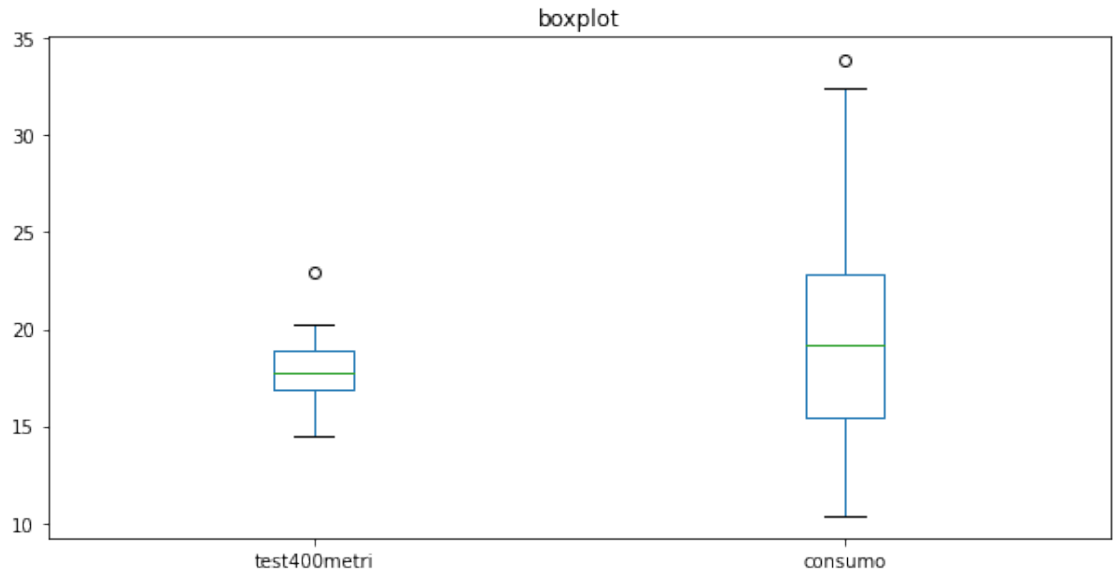


[12]:
```
#Visualizzate il grafico della funzione cumulativa empirica per lâĂŹattributo
from statsmodels.distributions.empirical_distribution import ECDF
dist = ECDF(cars.peso.dropna())
```

```
plt.plot(dist.x, dist.y)
plt.show()
```

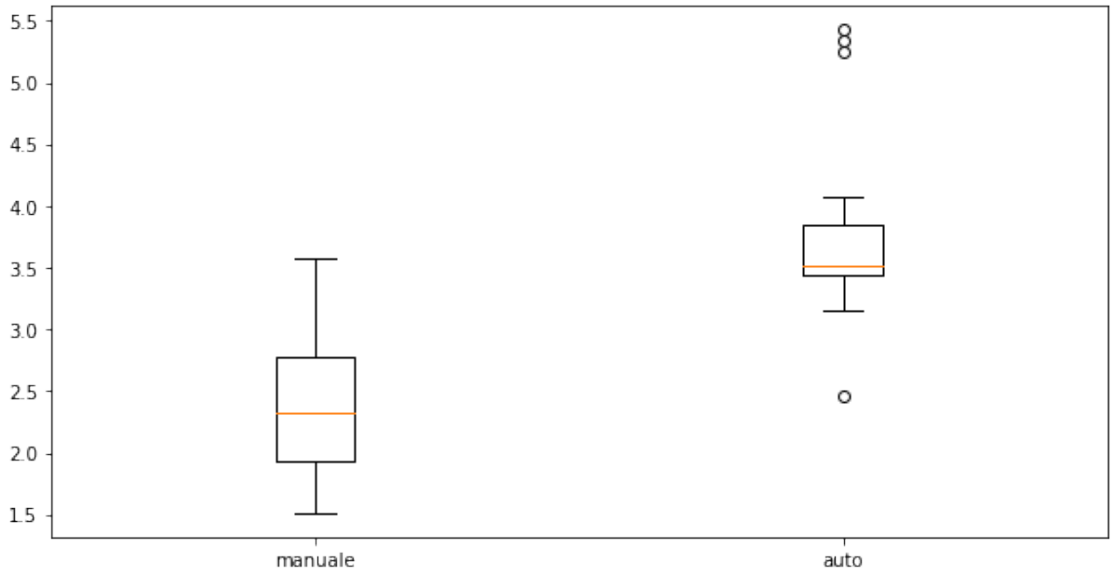


[81]:
```
#boxplot: per evidenziare outliers o per confrontare diversi attributi
plt.rcParams["figure.figsize"] = [8.50, 4.50]
plt.rcParams["figure.autolayout"] = True
cars[['test400metri', 'consumo']].plot(kind='box', title='boxplot')
plt.show()
```



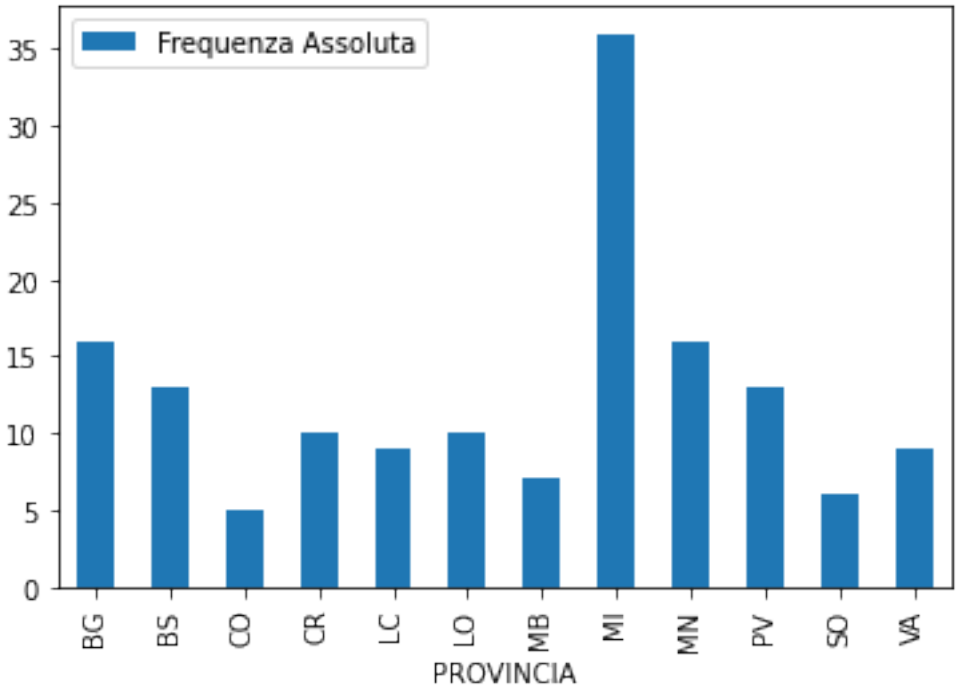
[83]:
```
#se stesso carattere con maschera
manuale = cars[cars['trasmissione'] == 1]['peso']
auto = cars[cars['trasmissione'] == 0]['peso']
plt.boxplot([manuale,auto], labels=['manuale','auto'])
plt.show()
```

```
/home/gorl6/.local/lib/python3.6/site-packages/numpy/core/_asarray.py:83:
VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```



0.4 Confronto tra 2 caratteri

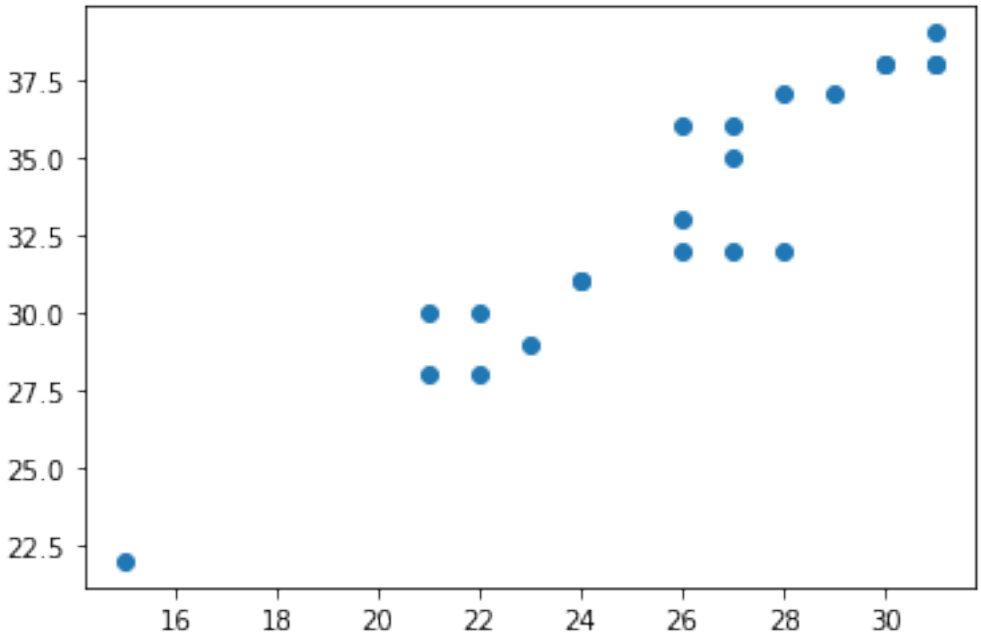
[13]:
```
#Come sono distribuite le centraline nelle diverse
#province della Lombardia? Rispondere con un grafico
centr = pd.read_csv('/home/gorl6/Desktop/STAT/Esami completi/DATI-AMBIENTE.
 ↪txt',sep=";",decimal=",",na_values=' ')
fprov = pd.crosstab(index=centr['PROVINCIA'],colnames=[''],columns="Frequenza␣
 ↪Assoluta")
fprov.plot.bar()
plt.show()
```



[14]:
```
#qual ÃÍ la provincia piÃź rappresentata e la meno rappresentata?

sorted_lomb= centr.PROVINCIA.value_counts().sort_values()
(sorted_lomb.head(1), sorted_lomb.tail(1))
```

[14]:
```
(CO    5
 Name: PROVINCIA, dtype: int64,
 MI    36
 Name: PROVINCIA, dtype: int64)
```

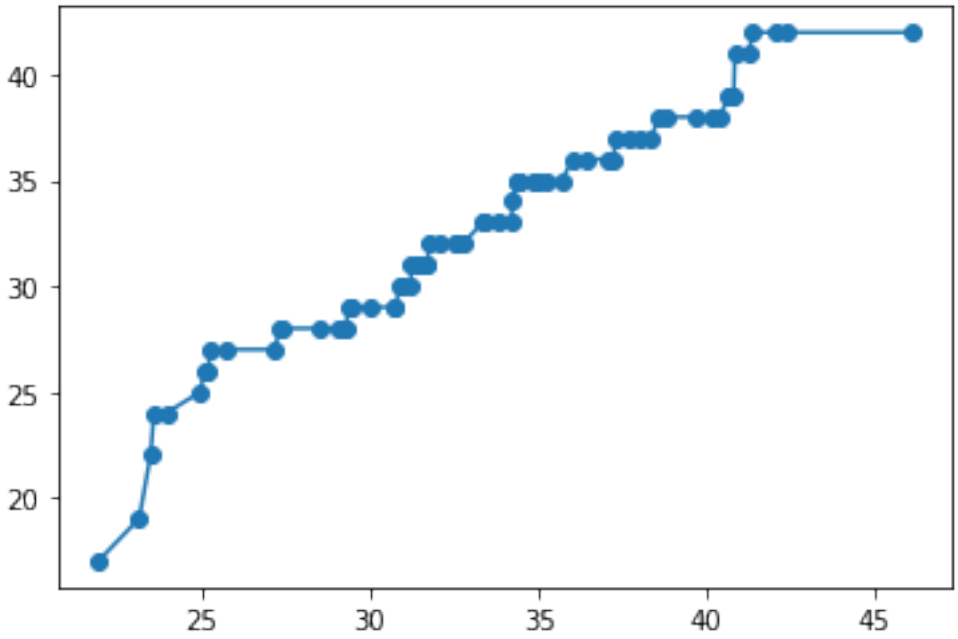
[15]:
```
#i due caratteri sono correlati? diagramma di dispersione

plt.scatter(centr.PM2_5, centr.PM10)
plt.show()

centr.PM2_5.corr(centr.PM10)
# il grafico mostra una dipendenza lineare tra i due caratteri
# il valore di correlazione molto vicino a 1 conferma la relazione
```



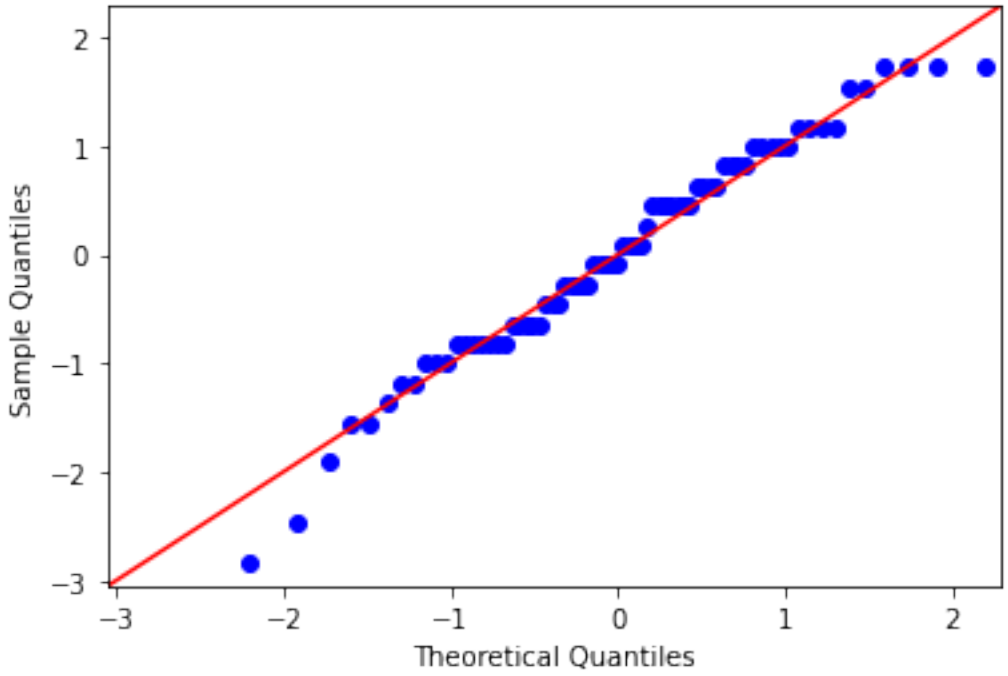
[15]: 0.9486649555820706

[62]:
```
#Se i dati lo permettono (se ad esempio sono giÃă ordinati)
#posso plottare uno scatter unendo i puntini per evidenziare una
#tendenza.

#Posso produrre campione casuale da una popolazione normale con gli stessi
#parametri di media e deviazione di un carattere e poi plottare un
#diagramma di dispersione per controllare se un carattere ha o meno una␣
 ↪distribuzione normale

attrib= centr.PM10.sort_values().dropna()

m= centr.PM10.mean()
ds= centr.PM10.std()
X = norm(m,ds)

x= X.rvs(71)
y = attrib

plt.scatter(x,y)
plt.plot(x,y) #collegare i puntini!!!!
plt.show()
```

None



[42]:
```
#La distribuzione ÃÍ compatibile con una normale?
#Stesso esercizio, metodo diverso: qui ÃÍ centrato tutto in 0
import statsmodels.api as sm

sm.qqplot(centr.PM10.dropna(), fit=True, line='45')
plt.show()
```



[64]:
```
#gruppi , per capire se ci sono caratterizzazioni notevoli
cars.groupby('marce').mean()
```

[64]:

marce	consumo	cilindrata	peso	test400metri	trasmissione
3	16.100667	176.133333	3.892600	17.692	0.000000
4	24.533333	89.500000	2.616667	18.965	0.666667
5	21.380000	195.600000	2.632600	15.640	1.000000

[72]:
```
#frequenze congiunte
marce_fre_con= pd.crosstab(index=cars.trasmissione,
                           columns=cars.marce
                             #normalize=True  #if relative needed
                             #margins=True
                           )
marce_fre_con
```

[72]:

marce	3	4	5
trasmissione			
0	15	4	0
1	0	8	5

0.5 Indici

[18]:
```
def gini(series):
    return 1 - sum(series.value_counts(normalize=True)
                           .map(lambda f: f**2))

def normalized_gini(series):
    s = len(series.unique())
    return s * gini(series)/(s-1)

normalized_gini(centr.PROVINCIA.dropna())
```

[18]:
```
0.9639757575757577
```

[ ]:
```

```