# Projet Python de master 1 Ingénierie Statistique et Data Science à ISUP.

September 9, 2022


### 0.1 Auteur : Souleymane BA


- `objet`: Implémentation d'un package python qui permet de résoudre un problème de modèles linéaires (régression linéaire et ridge) et de faire la simulation de variables aléatoires.
- nom du package: `mathstats`
- compte Github: https://github.com/Julesba97/package_Regression-Lineaire

[1]:
```
from mathstats.model_lineaire import read_dataset
from mathstats.model_lineaire import LinearModel
from mathstats.model_lineaire import Ridge
import warnings
warnings.filterwarnings("ignore")

dataset = read_dataset('fuel2001.txt')

dataset.head()
```

[1]:

| | Drivers | FuelC | Income | Miles | MPC | Pop | Tax | State |
|---|---|---|---|---|---|---|---|---|
| 0 | 3559897.0 | 2382507.0 | 23471.0 | 94440.0 | 12737.00 | 3451586.0 | 18.0 | Alabama |
| 1 | 472211.0 | 235400.0 | 30064.0 | 13628.0 | 7639.16 | 457728.0 | 8.0 | Alaska |
| 2 | 3550367.0 | 2428430.0 | 25578.0 | 55245.0 | 9411.55 | 3907526.0 | 18.0 | Arizona |
| 3 | 1961883.0 | 1358174.0 | 22257.0 | 98132.0 | 11268.40 | 2072622.0 | 21.7 | Arkansas |
| 4 | 21623793.0 | 14691753.0 | 32275.0 | 168771.0 | 8923.89 | 25599275.0 | 18.0 | California |

[2]:
```
X = dataset[["Income", "Miles", "Tax", "MPC"]]
X["Dlic"] = 1000 * dataset.Drivers / dataset.Pop
y = 1000 * dataset.FuelC / dataset.Pop
```

## 0.2 Régression Linéaire

[3]:
```
model_linear = LinearModel(intercept=True)
model_linear.fit(X, y)
model_linear.summary(X, y)
```

```
                                    Linear Regression Model
**************************************************************************************
**********
Residuals:
                     1Q          Median             3Q          Max
Min
-102.99          -31.59            2.47         25.19        97.05
**************************************************************************************
**********
Coefficients:
          Estimate     Std err          t          P>|t|    BorneInf    BorneSup
Const   103.804295  120.991159  0.857949  3.954671e-01 -139.884407  347.492997
Income   -0.000073    0.001814 -0.040490  9.678818e-01   -0.003728    0.003581
Miles     0.000345    0.000130  2.662172  1.072880e-02    0.000084    0.000606
Tax      -1.943743    1.486963 -1.307190  1.977883e-01   -4.938640    1.051154
MPC       0.031563    0.004403  7.168546  5.734565e-09    0.022695    0.040431
Dlic      0.214783    0.104871  2.048069  4.641499e-02    0.003562    0.426005
**************************************************************************************
**********
Residual Std err: 46.4008  avec  45  degré de liberté
R_square:  0.7551            Adj. R_square : 0.7279
F-statistic : 27.7569  avec 5 et 45 DF p_valeur: 1.0378364834195963e-12
**************************************************************************************
**********
```

[4]:
```
print("Coeff de détermination : ", model_linear.determination_coefficient())
```

```
Coeff de détermination :  0.7551477356878742
```

[5]:
```
print("Coeffs estimés: ", model_linear.get_coeffs())
```

```
Coeffs estimés:  {'Const': 103.80429466814711, 'Income': -7.346554907419174e-05,
'Miles': 0.0003450499285718293, 'Tax': -1.943743344988325, 'MPC':
0.0315629726721714, 'Dlic': 0.21478347845911272}
```

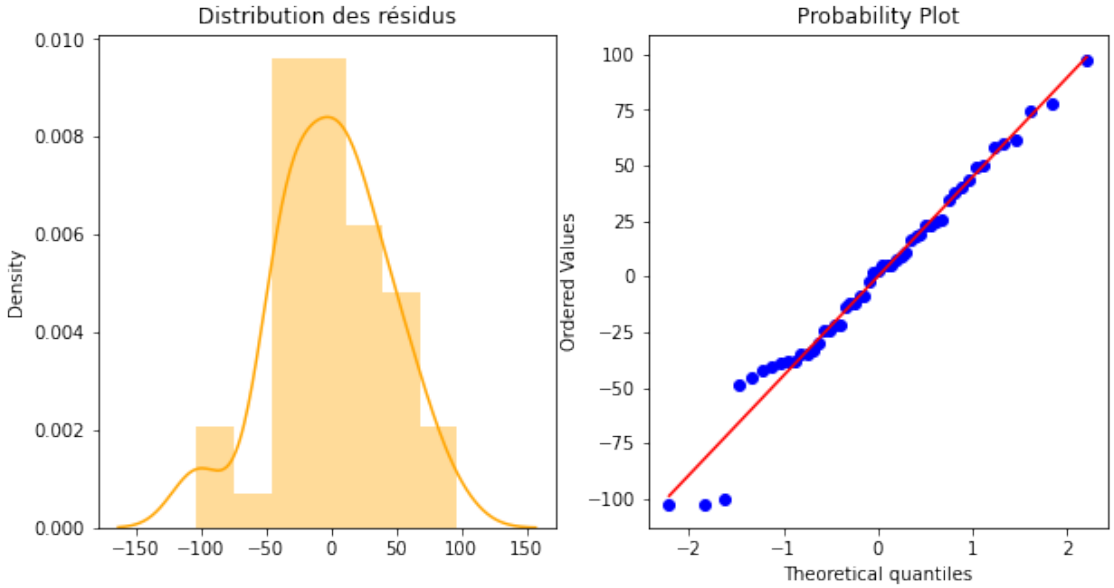
## 0.3 prédiction

[6]:
```
X_test = X[:10]
model_linear.predict(X_test)
```

[6]:
```
array([[723.22010114],
       [553.44207419],
       [578.2082485 ],
```

```
       [652.82230309],
       [587.7730524 ],
       [607.67456015],
       [558.87684266],
       [601.10395617],
       [420.02770311],
       [631.41980645]])
```

[7]:
```
model_linear.graphe_residus()
```




## 0.4 Régression Ridge

[8]:
```
model_ridge = Ridge(intercept=True, lambada=0.5)

model_ridge.fit(X, y)
model_ridge.summary(X, y)
```

```
                                    Ridge Model
**************************************************************************************
**********
Residuals:
                     1Q          Median             3Q          Max
Min
-112.07          -29.77             1.8         26.71        92.78
**************************************************************************************
**********
Coefficients:
         Estimate     Std err         t         P>|t|     BorneInf     BorneSup
```

```
Const   23.588315  121.580639  0.194014  8.470383e-01 -221.287661  268.464291
Income   0.000678    0.001823  0.371813  7.117773e-01   -0.002994    0.004350
Miles    0.000373    0.000130  2.864769  6.320678e-03    0.000111    0.000635
Tax     -1.547902    1.494208 -1.035935  3.057675e-01   -4.557390    1.461587
MPC      0.032506    0.004424  7.346946  3.125213e-09    0.023595    0.041417
Dlic     0.257540    0.105382  2.443863  1.851527e-02    0.045289    0.469790
********************************************************************************
**********
Residual Std err: 46.6269  avec  45  degré de liberté
R_square:  0.7528            Adj. R_square : 0.7253
F-statistic : 27.4013  avec 5 et 45 DF p_valeur: 1.2858603071208563e-12
********************************************************************************
**********
```

[9]:
```
print("Coeff de détermination : ", model_ridge.determination_coefficient())
```

```
Coeff de détermination :  0.7527560375742374
```

[10]:
```
print("Coeffs estimés: ", model_ridge.get_coeffs())
```

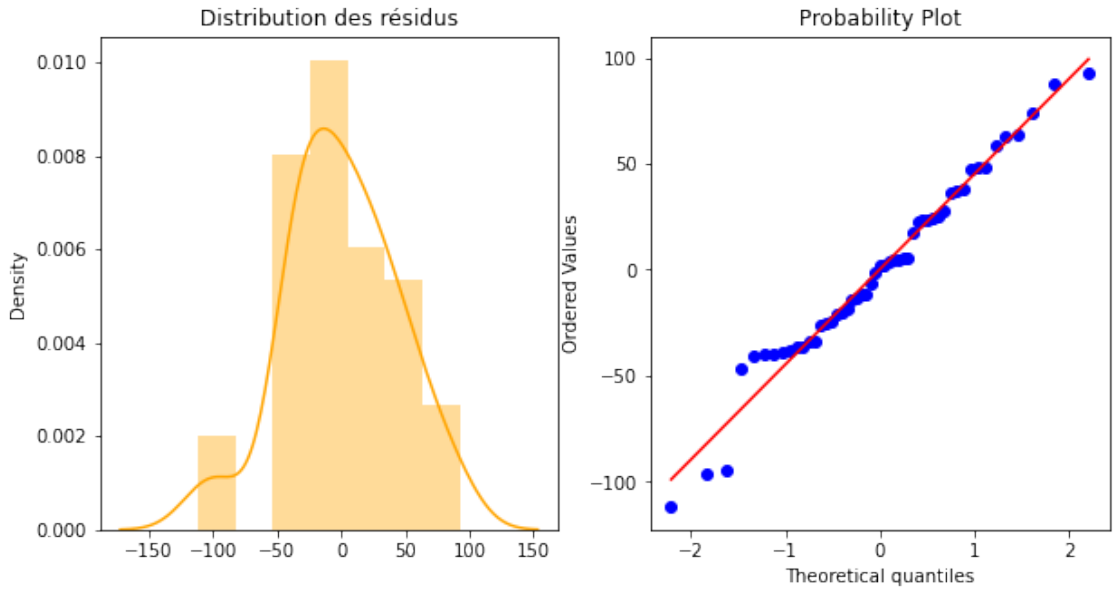
```
Coeffs estimés:  {'Const': 23.58831513522843, 'Income': 0.0006779132267759142,
'Miles': 0.0003731179683630864, 'Tax': -1.547901800684706, 'MPC':
0.03250606911470074, 'Dlic': 0.257539600420733}
```

### 0.5 Prédiction

[11]:
```
X_test = X[:5]
model_ridge.predict(X_test)
```

[11]:
```
array([[726.52565766],
       [550.67822546],
       [573.61089198],
       [651.7727703 ],
       [588.20236958]])
```

[12]:
```
model_ridge.graphe_residus()
```




## 0.6 Simulation de variables aléatoires

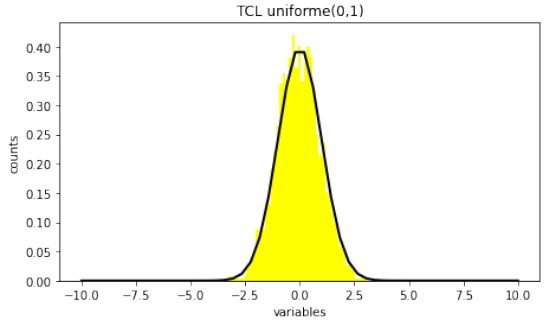
[13]:
```
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
from mathstats.random_simulation import centrales_limites
from mathstats.random_simulation import transform_sampling
```

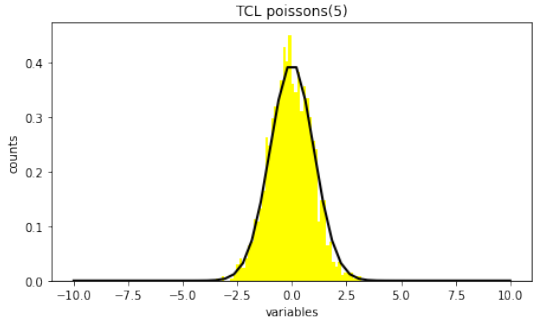
### 0.6.1 Méthode du théorème centrale limite

[14]:
```
tcl_vars2 = centrales_limites(np.random.uniform(0, 1, 100000),
                              size_n=100, nsim=2000, mux=0.5, sigma=np.sqrt(1/12))
tcl_vars1 = centrales_limites(np.random.poisson(5, 100000),
                              size_n=100, nsim=2000, mux =5, sigma=np.sqrt(5))
abscisse = np.linspace(-10, 10)
```

[15]:
```
plt.figure(figsize=(16, 4))
plt.subplot(1, 2, 1)
plt.hist(tcl_vars2, bins=50, color="yellow", density=True)
plt.plot(abscisse, stats.norm.pdf(abscisse, loc=0, scale=1), color="black",␣
 ↪lw=2)
plt.title("TCL uniforme(0,1)")
plt.xlabel("variables")
plt.ylabel("counts")
plt.subplot(1, 2, 2)
plt.hist(tcl_vars1, bins=50, color="yellow", density=True)
```

```
plt.plot(abscisse, stats.norm.pdf(abscisse, loc=0, scale=1), color="black",␣
 ↪lw=2)
plt.title("TCL poissons(5)")
plt.xlabel("variables")
plt.ylabel("counts")
plt.show()
```

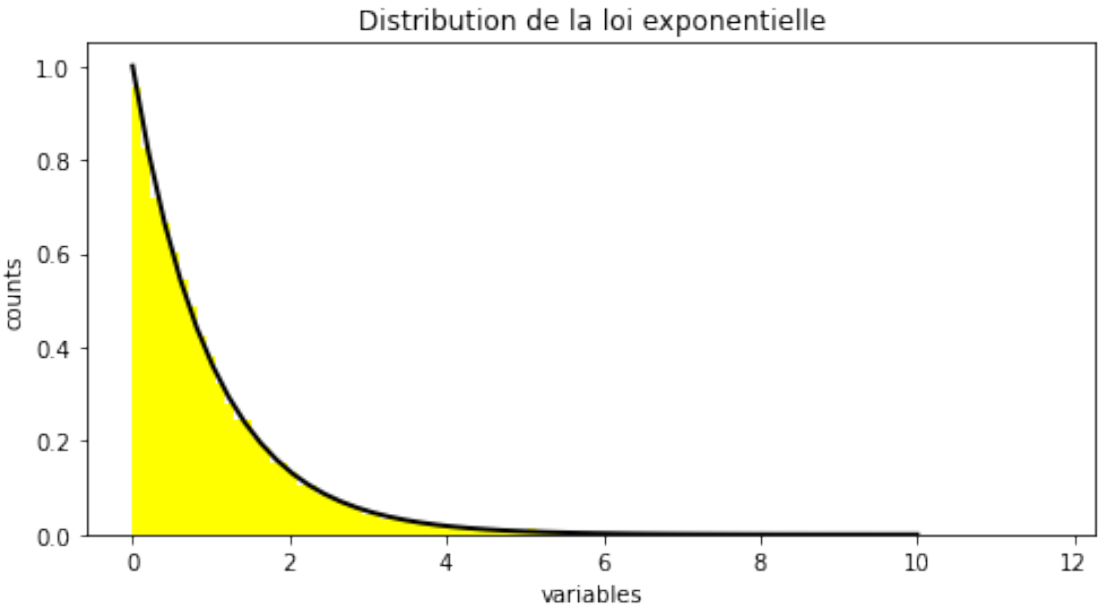



### 0.6.2 Méthode des quantiles

[16]:
```
qantiexp = lambda u, lambada: -np.log(1-u)/lambada
var_alea = transform_sampling(qantiexp, 1, 10000)
abscisse1 = np.linspace(0, 10)
```

[17]:
```
plt.figure(figsize=(8, 4))
plt.hist(var_alea, bins=100, color="yellow", density=True)
plt.plot(abscisse1, stats.expon.pdf(abscisse1, loc=0, scale=1), color="black",␣
 ↪lw=2)
plt.title("Distribution de la loi exponentielle")
plt.xlabel("variables")
plt.ylabel("counts")
plt.show()
```


Distribution de la loi exponentielle
counts
variables
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12