Regresie liniară
Cursul 7

Programare şi modelare probabilistă - anul III

Facultatea de Informatică, UAIC

e-mail: adrian.zalinescu@uaic.ro

web: https://sites.google.com/view/fiicoursepmp/home

27 Noiembrie 2023

Plan

1. Regresia liniară robustă
2. Regresia liniară ierarhică
3. Probleme legate de modelul liniar
   - Variabile ascunse
   - Mascarea efectului variabilelor
4. Interacţiuni între variabile
5. Dispersie variabilă

Regresia liniară robustă

Utilizarea ipotezei că datele folosesc o distribuţie Gaussiană este rezonabilă în multe cazuri.

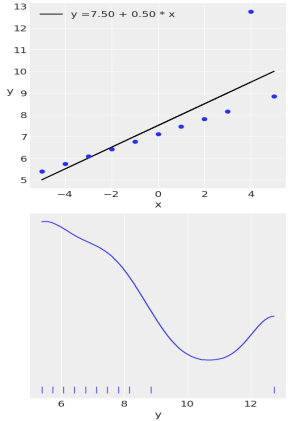
- Acest lucru nu înseamnă neapărat că datele sunt Gausseine (au fost generate conform unei distribuţii normale), ci că acest lucru este o aproximare rezonabilă pentru o problemă dată.
- După cum am văzut deja în Cursul 5, presupunerea că datele sunt Gaussiene eşuează în prezenţa *valorilor aberante* (*outliers*).
- În acest caz, o distribuţie t a lui Student este mai eficientă pentru luarea în calcul a valorilor aberante, pentru a obţine o inferenţă mai robustă.

Exemplu: al treilea set de date din cvartetul lui Anscombe – `https://en.wikipedia.org/wiki/Anscombe%27s_quartet`.
Vom centra mai întâi datele:

```
ans = pd.read_csv('./data/anscombe.csv')
x_3 = ans[ans.group == 'III']['x'].values
y_3 = ans[ans.group == 'III']['y'].values
x_3 = x_3 - x_3.mean()with model:
    data_plus_one = data_generator + 1
```

Apoi, vedem cum arată acest set de date:

```
_, ax = plt.subplots(2, 1, figsize=(5, 10))
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]
ax[0].plot(x_3, (alpha_c + beta_c * x_3), 'k',
           label=f'y ={alpha_c:.2f} + {beta_c:.2f} * x')
ax[0].plot(x_3, y_3, 'C0o')
ax[0].set_xlabel('x')
ax[0].set_ylabel('y', rotation=0)
ax[0].legend(loc=0)
az.plot_kde(y_3, ax=ax[1], rug=True)
ax[1].set_xlabel('y')
ax[1].set_yticks([])
plt.tight_layout()
```



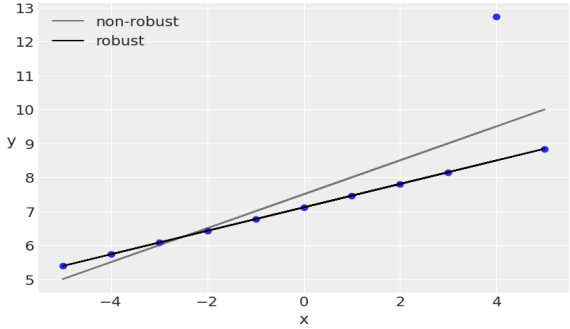
Vom rescrie acum modelul, folosind o distribuţie Student $t(\nu, \mu, \sigma)$ în loc de distribuţia normală $\mathcal{N}(\mu, \sigma)$, unde ν este *parametrul de normalitate*.

```
with pm.Model() as model_t:
    α = pm.Normal('α', mu=y_3.mean(), sigma=1)
    β = pm.Normal('β', mu=0, sigma=1)
    ϵ = pm.HalfNormal('ϵ', 5)
    ν_ = pm.Exponential('ν_', 1/29)
    ν = pm.Deterministic('ν', ν_ + 1)

    y_pred = pm.StudentT('y_pred', mu=α + β * x_3,
                         sigma=ϵ, nu=ν, observed=y_3)

    idata_t = pm.sample(2000, return_inferencedata=True)
```

Pentru ν, distribuţia a priori utilizată este $1 + \text{Exp}$ de medie 30, şi nu Exp, pentru a evita valori ale lui ν prea apropiate de 0.

```
beta_c, alpha_c = stats.linregress(x_3, y_3)[:2]
plt.plot(x_3, (alpha_c + beta_c * x_3), 'k', label='non-robust', alpha=0.5)
plt.plot(x_3, y_3, 'C0o')
alpha_m = idata_t.posterior['α'].mean().item()
beta_m = idata_t.posterior['β'].mean().item()
plt.plot(x_3, alpha_m + beta_m * x_3, c='k', label='robust')
plt.xlabel('x')
plt.ylabel('y', rotation=0)
plt.legend(loc=2)
```

O distribuţie Student acordă mai puţină importanţă valorilor aberante, depărtate de grosul datelor:



Rezumatul distribuţiei a posteriori:

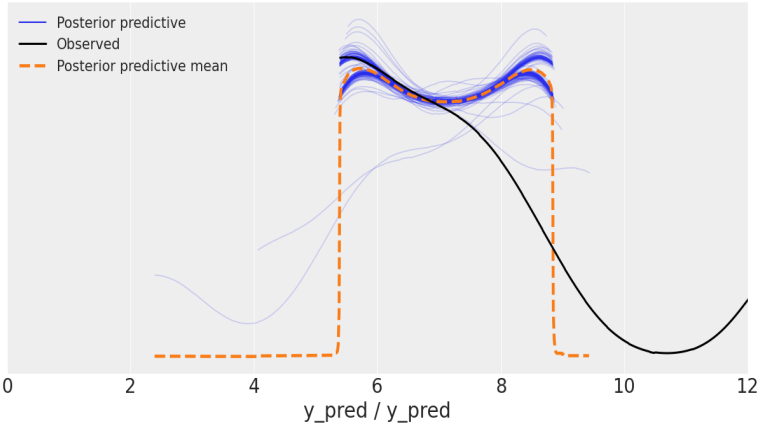
```
az.summary(idata_t, var_names=varnames)
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
α	7.114	0.001	7.111	7.116	0.000	0.000	4721.0	4205.0	1.0
β	0.345	0.000	0.345	0.346	0.000	0.000	4554.0	3904.0	1.0
ϵ	0.003	0.002	0.001	0.006	0.000	0.000	1268.0	514.0	1.0
ν	1.211	0.201	1.000	1.570	0.003	0.002	3413.0	1591.0	1.0

Observăm că parametrii α, β şi ε sunt foarte precişi (deviaţii standard foarte mici), deoarece datele erau, cu excepţia valorii aberante, perfect aliniate.

Distribuţia predictivă a posteriori:

```
pm.sample_posterior_predictive(idata_t, model=model_t, random_seed=2, extend_inferencedata=True)
ax = az.plot_ppc(idata_t, num_pp_samples=200, figsize=(12, 6), mean=True)
plt.xlim(0, 12)
```



Regresia liniară ierarhică

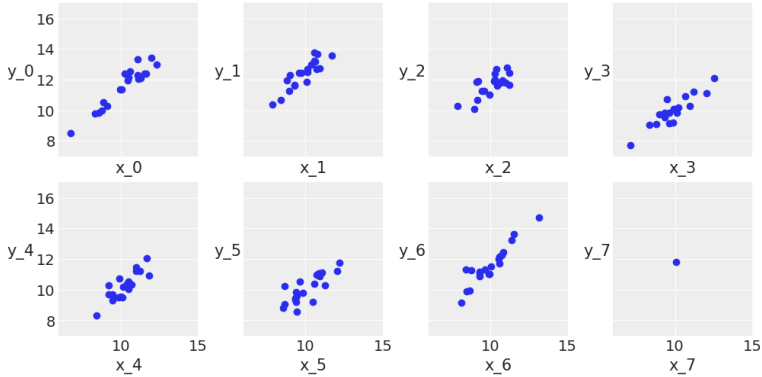
Vom aplica conceptul de modele ierarhice pentru regresie liniară.

- Acesta permite modelului de a considera inferenţe la nivel de grup şi estimări la nivel mai înalt.
- După cum am văzut în Cursul 5, acest lucru se realizează folosind *distribuţii hiper a priori*.

Vom crea 8 grupuri, incluzând un grup cu o singură intrare:

```
N = 20
M = 8
idx = np.repeat(range(M-1), N)
idx = np.append(idx, 7)
np.random.seed(314)

alpha_real = np.random.normal(2.5, 0.5, size=M)
beta_real = np.random.beta(6, 1, size=M)
eps_real = np.random.normal(0, 0.5, size=len(idx))

y_m = np.zeros(len(idx))
x_m = np.random.normal(10, 1, len(idx))
y_m = alpha_real[idx] + beta_real[idx] * x_m + eps_real
```

```
_, ax = plt.subplots(2, 4, figsize=(10, 5), sharex=True, sharey=True)
ax = np.ravel(ax)
j, k = 0, N
for i in range(M):
    ax[i].scatter(x_m[j:k], y_m[j:k])
    ax[i].set_xlabel(f'x_{i}')
    ax[i].set_ylabel(f'y_{i}', rotation=0, labelpad=15)
    ax[i].set_xlim(6, 15)
    ax[i].set_ylim(7, 17)
    j += N
    k += N
```



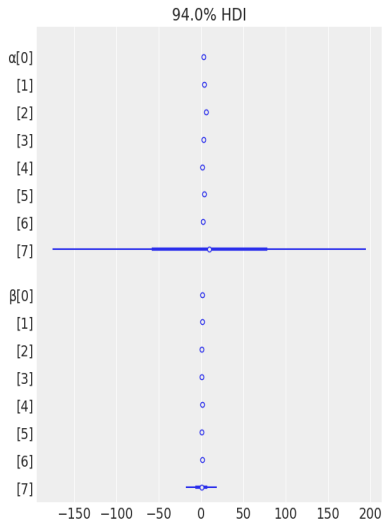
Mai întâi, centrăm datele: `x_centered = x_m - x_m.mean()`

Apoi, considerăm un model robust, ne-ierarhic:

```
with pm.Model() as unpooled_model:
    α_tmp = pm.Normal('α_tmp', mu=0, sigma=10, shape=M)
    β = pm.Normal('β', mu=0, sigma=10, shape=M)
    ϵ = pm.HalfCauchy('ϵ', 5)
    ν = pm.Exponential('ν', 1/30)

    y_pred = pm.StudentT('y_pred', mu=α_tmp[idx] + β[idx] * x_centered,
                         sigma=ϵ, nu=ν, observed=y_m)

    α = pm.Deterministic('α', α_tmp - β * x_m.mean())

    idata_up = pm.sample(2000, target_accept=0.9, return_inferencedata=True)
```

După cum vedem mai jos, estimările pentru parametrii α_7 şi β_7 sunt foarte împrăştiate faţă de celelalte (α_i, β_i, $0 \leq i \leq 6$):

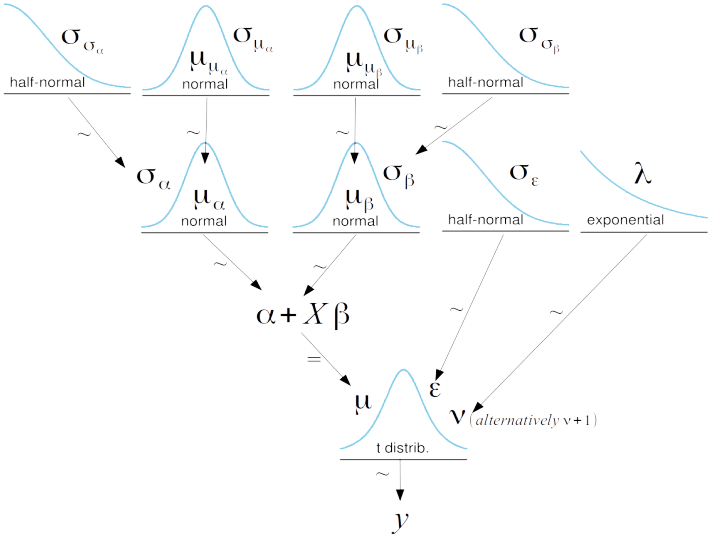
```
az.plot_forest(idata_up, var_names=['α', 'β'], combined=True)
```



- Acest lucru se întâmplă deoarece nu are sens să potrivim (fit) o dreaptă numai după un punct.
- Avem nevoie de cel puţin două puncte pentru acest lucru.
- O primă posibilitate ar fi să dăm mai multe informaţii legate de α şi β, adică să introducem distribuţii tari (informative) a priori pentru acestea.
- O altă posibilitate este de a introduce un model ierarhic, ceea ce permite ca informaţia să fie partajată între grupuri, fapt ce conduce la fenomenul de "shrinkage" ale valorilor plauzibile ale parametrilor estimaţi.
- Acest lucru este foarte folositor în cazul în care avem grupuri sărace în date (cazul de mai sus este extrem: un grup cu o singură intrare).

Descrierea modelului:

cu o *diagramă Krusche*:

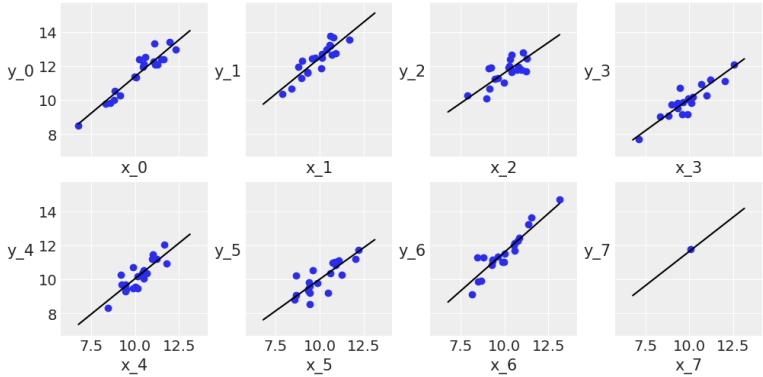


Modelul ierarhic:

```
with pm.Model() as hierarchical_model:
    # hyper-priors
    α_μ_tmp = pm.Normal('α_μ_tmp', mu=0, sigma=10)
    α_σ_tmp = pm.HalfNormal('α_σ_tmp', 10)
    β_μ = pm.Normal('β_μ', mu=0, sigma=10)
    β_σ = pm.HalfNormal('β_σ', sigma=10)

    # priors
    α_tmp = pm.Normal('α_tmp', mu=α_μ_tmp, sigma=α_σ_tmp, shape=M)
    β = pm.Normal('β', mu=β_μ, sigma=β_σ, shape=M)
    ϵ = pm.HalfCauchy('ϵ', 5)
    ν = pm.Exponential('ν', 1/30)

    y_pred = pm.StudentT('y_pred', mu=α_tmp[idx] + β[idx] * x_centered,
                         sd=ϵ, nu=ν, observed=y_m)

    α = pm.Deterministic('α', α_tmp - β * x_m.mean())
    α_μ = pm.Deterministic('α_μ', α_μ_tmp - β_μ * x_m.mean())
    α_σ = pm.Deterministic('α_sd', α_σ_tmp - β_μ * x_m.mean())

    idata_hm = pm.sample(2000, target_accept=0.99, return_inferencedata=True)
```

Vizualizarea rezultatelor:

```
_, ax = plt.subplots(2, 4, figsize=(10, 5), sharex=True, sharey=True)
ax = np.ravel(ax)
j, k = 0, N
x_range = np.linspace(x_m.min(), x_m.max(), 10)

posterior_hm = idata_hm.posterior.stack(samples={"chain", "draw"})

for i in range(M):
    ax[i].scatter(x_m[j:k], y_m[j:k])
    ax[i].set_xlabel(f'x_{i}')
    ax[i].set_ylabel(f'y_{i}', labelpad=17, rotation=0)
    alpha_m = posterior_hm["α"].sel({"α_dim_0":i}).mean().item()
    beta_m = posterior_hm["β"].sel({"β_dim_0":i}).mean().item()
    ax[i].plot(x_range, alpha_m + beta_m * x_range, c='k',
               label=f'y = {alpha_m:.2f} + {beta_m:.2f} * x')
    plt.xlim(x_m.min()-1, x_m.max()+1)
    plt.ylim(y_m.min()-1, y_m.max()+1)
    j += N
    k += N
```


y_0
y_1
y_2
y_3
y_4
y_5
y_6
y_7
x_0
x_1
x_2
x_3
x_4
x_5
x_6
x_7
8
10
12
14
7.5
10.0
12.5

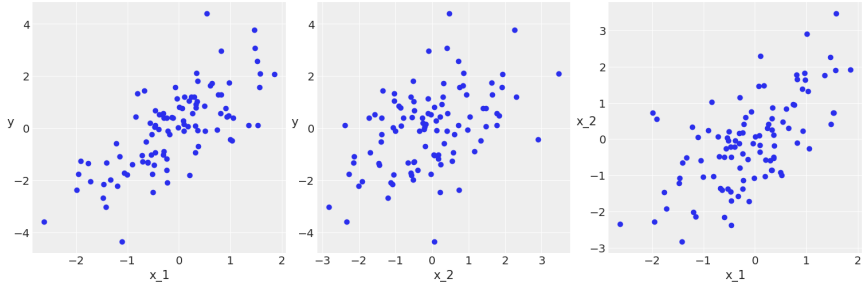
Variabile ascunse

Să ne imaginăm următoarea situaţie:

- Fie variabila prezicătoare x şi variabila prezisă y;
- să presupunem că avem o altă variabilă z, ce influenţează direct atât pe x, cât şi pe y.
- De exemplu,
  - z ar putea fi "revoluţia industrială";
  - x, numărul de mori de vânt;
  - y, concentraţia de CO_2 din atmosferă.
- Dacă îl omitem pe z din analiză, am putea deduce o relaţie liniară între x şi y, ba chiar am putea prezice y în funcţie de x.
- Astfel, am putea omite mecanismul care leagă aceste variante, punând încălzirea globală pe seama reducerii numărului de mori de vânt.
- O astfel de variabilă z se numeşte *variabilă ascunsă* (sau *latentă*).

Următorul cod simulează o variabilă ascunsă x_1, ce influenţează variabilele x_2 şi y:

```
np.random.seed(42)
N = 100
x_1 = np.random.normal(size=N)
x_2 = x_1 + np.random.normal(size=N, scale=1)
#x_2 = x_1 + np.random.normal(size=N, scale=0.01)
y = x_1 + np.random.normal(size=N)
X = np.vstack((x_1, x_2)).T

scatter_plot(X, y)
```



Vom construi trei modele:

- primul model, de regresie multiplă, cu variabilele de input x_1 şi x_2:

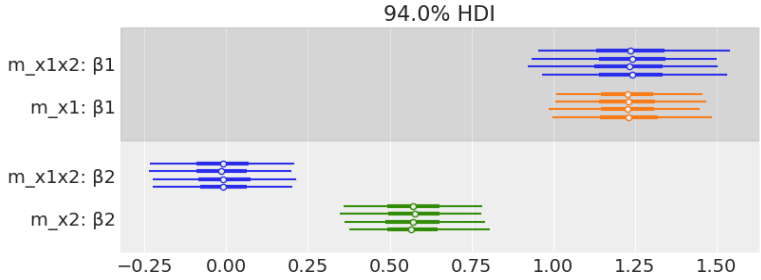
```
with pm.Model() as m_x1x2:
    α = pm.Normal('α', mu=0, sigma=10)
    β1 = pm.Normal('β1', mu=0, sigma=10)
    β2 = pm.Normal('β2', mu=0, sigma=10)
    ϵ = pm.HalfCauchy('ϵ', 5)
    μ = α + β1 * X[:, 0] + β2 * X[:, 1]
    y_pred = pm.Normal('y_pred', mu=μ, sigma=ϵ, observed=y)
    idata_x1x2 = pm.sample(2000, return_inferencedata=True)
```

- celelalte modele, de regresie simplă, cu x_1, respectiv x_2:

```
with pm.Model() as m_x1:
    α = pm.Normal('α', mu=0, sigma=10)
    β1 = pm.Normal('β1', mu=0, sigma=10)
    ϵ = pm.HalfCauchy('ϵ', 5)
    μ = α + β1 * X[:, 0]
    y_pred = pm.Normal('y_pred', mu=μ, sigma=ϵ, observed=y)
    idata_x1 = pm.sample(2000, return_inferencedata=True)
with pm.Model() as m_x2:
    α = pm.Normal('α', mu=0, sigma=10)
    β2 = pm.Normal('β2', mu=0, sigma=10)
    ϵ = pm.HalfCauchy('ϵ', 5)
    μ = α + β2 * X[:, 1]
    y_pred = pm.Normal('y_pred', mu=μ, sigma=ϵ, observed=y)
    idata_x2 = pm.sample(2000, return_inferencedata=True)
```

Să aruncăm o privire asupra parametrilor β din aceste modele:

```
az.plot_forest([idata_x1x2, idata_x1, idata_x2],
               model_names=['m_x1x2', 'm_x1', 'm_x2'],
               var_names=['β1', 'β2'],
               combined=False, colors='cycle', figsize=(8, 3))
```

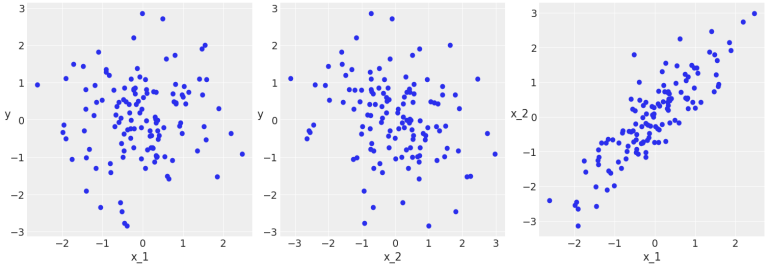


- După cum vedem, β_2 din modelul `m_x1x2` este în jurul lui 0, ceea ce indică o contribuţie aproape neglijabilă a variabilei x_2 pentru a explica variabila y.
- În schimb, dacă variabila x_1 este ascunsă, β_2 devine semnificativ mai mare (peste 0.5).
- Astfel, informaţia în x_2 este *redundantă* dacă ştim x_1.

Mascarea efectului variabilelor

Vom crea un set de date ce exemplifică acest fenomen:

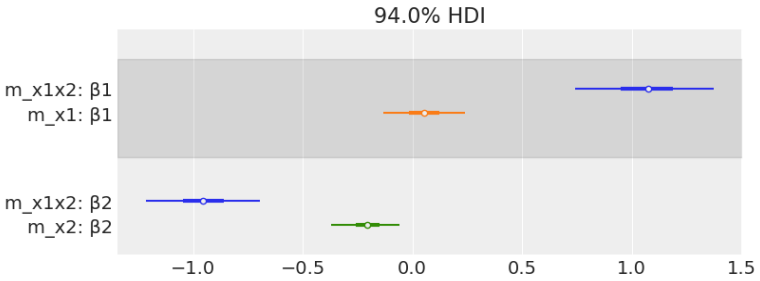
```
np.random.seed(42)
N = 126
r = 0.8
x_1 = np.random.normal(size=N)
x_2 = np.random.normal(x_1, scale=(1 - r ** 2) ** 0.5)
y = np.random.normal(x_1 - x_2)
X = np.vstack((x_1, x_2)).T
scatter_plot(X, y)
```



Astfel, x_1 şi x_2 sunt pozitiv corelate, iar x_1 şi x_2 sunt corelate cu y, dar în sens invers.

Construind cele trei modele, ca mai înainte, obţinem următoarele rezultate:

```
az.plot_forest([idata_x1x2, idata_x1, idata_x2],
               model_names=['m_x1x2', 'm_x1', 'm_x2'],
               var_names=['β1', 'β2'],
               combined=True, colors='cycle', figsize=(8, 3))
```



- Observăm că valorile lui β_1 şi β_2 din modelul `m_x1x2` sunt în jurul lui 1, respectiv -1, ceea ce este de aşteptat.
- În schimb, în modelele simple, aceste valori sunt mai apropiate de 0, ceea ce indică un efect mai slab: are loc o anulare parţială a efectelor:
- dacă y şi x_1 cresc, atunci x_2 descreşte, chiar dacă x_1 şi x_2 sunt pozitiv corelate.

Interacţiuni între variabile

Într-un model de regresie multiplă, se presupune (implicit) că o schimbare în x_i rezultă într-o schimbare constantă în y, dacă păstrăm fixe valorile restului variabilelor independente.

- Dar acest lucru nu este întotdeauna adevărat.
- Ce se întâmplă dacă schimbările în x_i afectează y, dar rata acestora este modulată de schimbările în x_j?
- Exemplu privind interacţiunea între medicamente:
  - creşterea dozei medicamentului A rezultă în efecte pozitive asupra pacientului;
  - dar acest lucru este valabil în absenţa (sau doze mici ale) medicamentului B, în timp ce efectul lui A este negativ (sau letal) pentru doze mărite ale lui B.

Dacă dorim să captăm aceste efecte, o opţiune este să includem termeni non-aditivi în modelul nostru.

- O opţiune standard, dar nu unica, este să multiplicăm variabilele, de exemplu:

$$\mu = \alpha + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2.$$

- Acest termen non-aditiv este cunoscut în statistică drept *interacţiune*.
- Putem interpreta acest termen ca un model liniar. De exemplu, putem scrie expresia de mai sus ca

$$\mu = \alpha + (\underbrace{\beta_1 + \beta_3 x_2}_{\text{panta lui } x_1}) x_1 + \beta_2 x_2$$

sau ca

$$\mu = \alpha + \beta_1 x_1 + (\underbrace{\beta_2 + \beta_3 x_1}_{\text{panta lui } x_2}) x_2.$$

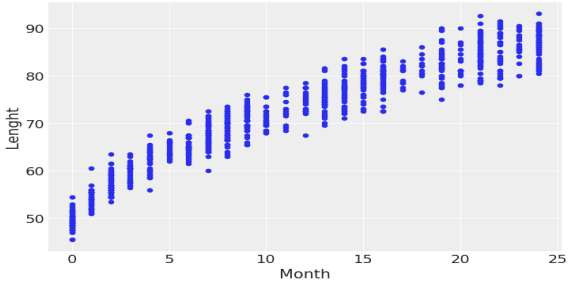
Dispersie variabilă

Până acum am folosit un tipar liniar (sau polinomial) pentru a modela **media** unei variabile:

$$\mathbb{E}y = \mu = \alpha + \beta x.$$

Putem face acelaşi lucru şi pentru dispersia (ori deviaţia standard) a variabilei dependente, atunci când nu mai putem presupune că aceasta este constantă.
În acest caz, vom considera dispersia ca o funcţie (liniară) de variabila independentă.
Exemplu: OMS colectează date despre copiii din întreaga lume şi concepe standarde de creştere pentru aceştia. Un exemplu de asemenea date este oferit de înălţimea copiilor de până la o anumită vârstă în funcţie de vârsta lor (în luni):

```
data = pd.read_csv('./data/babies.csv')
data.plot.scatter('Month', 'Lenght')
```



Comparativ cu modelele precedente, vom mai introduce 3 elemente suplimentare:

- ε este acum o funcţie liniară de x: $\varepsilon = \gamma + \delta x$;
- astfel γ şi δ sunt analoage lui α, respectiv β;
- modelul liniar pentru μ (media) este o funcţie de $\sqrt{x}$: $\mu = \alpha + \beta\sqrt{x}$;
- acesta este doar o modalitate de a potrivi modelul pe datele de mai sus.

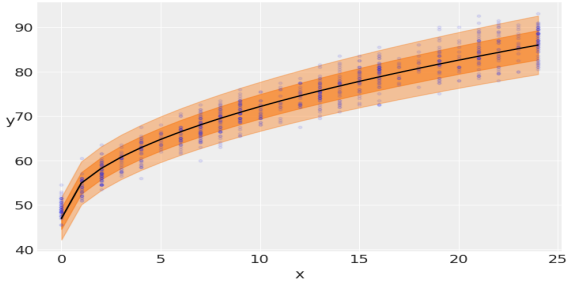
- definim o variabilă partajată, `x_shared`. O vom folosi pentru a schimba valorile lui x fără a recalibra modelul.

Modelul este definit astfel:

```
with pm.Model() as model_vv:
    α = pm.Normal('α', sigma=10)
    β = pm.Normal('β', sigma=10)
    γ = pm.HalfNormal('γ', sigma=10)
    δ = pm.HalfNormal('δ', sigma=10)

    x_shared = pm.MutableData("x_shared", data.Month.values.astype(float))

    μ = pm.Deterministic('μ', α + β * x_shared**0.5)
    ϵ = pm.Deterministic('ϵ', γ + δ * x_shared)

    y_pred = pm.Normal('y_pred', mu=μ, sigma=ϵ, observed=data.Lenght)

    idata_vv = pm.sample(1000, tune=1000, return_inferencedata=True)
```

Vizualizarea distribuţiei a posteriori:

```
plt.plot(data.Month, data.Lenght, 'C0.', alpha=0.1)
μ_m = idata_vv.posterior['μ'].mean(("chain", "draw")).values
ϵ_m = idata_vv.posterior['ϵ'].mean(("chain", "draw")).values
plt.plot(data.Month, μ_m, c='k')
plt.fill_between(data.Month,
                 μ_m + 1 * ϵ_m, μ_m - 1 * ϵ_m,
                 alpha=0.6, color='C1')
plt.fill_between(data.Month,
                 μ_m + 2 * ϵ_m, μ_m -2 * ϵ_m,
                 alpha=0.4, color='C1')
plt.xlabel('x')
plt.ylabel('y', rotation=0)
```



- Să presupunem acum că avem un exemplu concret de copil de două săptămâni (0.5 luni) şi dorim să aflăm cum se situează el în această analiză.
- Putem face acest lucru cu ajutorul funcţiei `sample_posterior_predictive` din `PyMC3`, cerând modelului distribuţia variabilei $y =$`length` pentru $x = 0.5$ luni.
- Problema este că acestă funcţie returnează predicţii ale lui y pentru valori observate ale lui x, iar valoarea 0.5 nu a fost observată.
- O posibilitate este de a defini o variabilă partajată (ca parte a modelului) şi apoi a actualiza valoarea aceste variabile partajate chiar înainte de a genera valori din distribuţia predictivă a posteriori:

```
pm.set_data({"x_shared":[0.5]}, model=model_vv)
ppc = pm.sample_posterior_predictive(idata_vv, model=model_vv)
y_ppc = ppc.posterior_predictive['y_pred'].stack(sample=("chain", "draw")).val
```

Putem acum vizualiza distribuţia lui $y \mid x = 0.5$ şi, adiţional, cum se plasează o valoare de referinţă (o înălţime) relativ la aceasta:

```
ref = 53
grid, pdf = az.kde(y_ppc)
plt.plot(grid, pdf)
percentile = int(np.mean(y_ppc <= ref) * 100)
plt.fill_between(grid[grid < ref],
                 pdf[grid < ref],
                 label=f'percentile = {percentile:2d}')
plt.xlabel('length')
plt.yticks([])
plt.legend()
```