Avance_Analyse_bayesienne

February 4, 2026

0.0.1 PROJET - HOUSING

[3]:
```
#les biblioteques
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import statsmodels.formula.api as smf
import statsmodels.stats.anova as smsa
import arviz as az
import pymc as pm
from sklearn.preprocessing import StandardScaler
```

0.1 le jeux de donnée

[4]:
```
Housing=pd.read_csv('housing.csv')
```

[5]:
```
Housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

Nous constatons l'absence de données pour la variable correspondant au nombre total de chambres

dans certaines observations, ce qui nous contraint à supprimer un certain nombre de lignes du jeu de données afin de garantir la qualité de l'analyse.

[6]:
```
Housing.dropna(inplace=True)
```

1 Introduction

Le marché du logement en Californie est connu pour ses prix élevés et ses fortes inégalités selon les régions.

Dans ce projet, l'objectif est de prédire le prix médian des habitations en fonction des caractéristiques propres à chaque zone ou ville : revenus, densité de population, proximité de l'océan, etc.

L'idée, c'est de voir comment ces variables influencent les prix, et d'utiliser des outils de data science pour construire un modèle prédictif. Ce travail permet non seulement d'explorer les dynamiques du marché immobilier californien, mais aussi de mieux comprendre comment certains facteurs peuvent faire grimper (ou baisser) le prix d'un logement.

Nous allons commencer par une analyse du jeu de données puis nous nous poursuivrons par la creation de multiples modele de regression afin d'obtenir une idée des relations entre les variables explicatives pertiennentes et le prix médian des logements. Cette étape servira également à identifier les variables les plus pertinentes grâce à des critères comme la significativité statistique ou le R^2 ajusté. Ensuite, nous approfondirons l'analyse en étudiant individuellement les variables sélectionnées, afin de mieux comprendre leur impact spécifique sur le prix du logement.

[7]:
```
len(Housing) , Housing.columns
```

[7]:
```
(20433,
 Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
        'total_bedrooms', 'population', 'households', 'median_income',
        'median_house_value', 'ocean_proximity'],
       dtype='object'))
```

Le jeu de données contient **20 433 observations** et **10 variables** décrivant les caractéristiques démographiques, économiques et géographiques de différentes zones résidentielles en Californie. **Chaque ligne représente une zone géographique agrégée**, et les colonnes sont les suivantes :

- **longitude :** la longitude de la zone (coordonnée géographique).
- **latitude :** la latitude de la zone.
- **housing_median_age :** l'âge médian des logements dans la zone.
- **total_rooms :** le nombre total de pièces (chambres, salons, etc.) dans l'ensemble des logements de la zone.
- **total_bedrooms :** le nombre total de chambres dans la zone.
- **population :** le nombre total d'habitants.
- **households :** le nombre total de ménages (groupes de personnes partageant un logement).
- **median_income :** le revenu médian des habitants (en dizaines de milliers de dollars).
- **median_house_value :** la valeur médiane des logements dans la zone (en dollars).

- **ocean_proximity :** une variable catégorielle indiquant la proximité de la zone avec l'océan (par exemple : <1H OCEAN, INLAND, ISLAND, NEAR BAY, NEAR OCEAN).

[8]:
```
fig = px.scatter_mapbox(Housing, lat="latitude", lon="longitude",␣
↪color="median_house_value", size="population",
                        color_continuous_scale=px.colors.cyclical.IceFire,␣
↪size_max=15, zoom=10,
                        mapbox_style="carto-positron")
fig.show()
```

```
/var/folders/1c/5b9b_qtd7zs3s40wm509d9w40000gn/T/ipykernel_68154/3518373067.py:1
: DeprecationWarning: *scatter_mapbox* is deprecated! Use *scatter_map* instead.
Learn more at: https://plotly.com/python/mapbox-to-maplibre/
  fig = px.scatter_mapbox(Housing, lat="latitude", lon="longitude",
color="median_house_value", size="population",
```

La carte ci-dessus représente la répartition géographique des zones résidentielles en Californie, avec une coloration en fonction de la valeur médiane des logements (median_house_value).

On observe clairement que les zones côtières, notamment autour de San Francisco, Los Angeles et San Diego, affichent les prix les plus élevés (points allant du rouge au noir selon la palette utilisée).

À l'inverse, les zones plus éloignées de la côte (partie centrale de l'État) présentent des valeurs beaucoup plus faibles, visibles par les points bleus clairs.

La taille des points, proportionnelle à la population, montre également que les grandes agglomérations concentrent à la fois plus d'habitants et des logements plus chers.

Cette carte confirme donc l'intuition selon laquelle la proximité de l'océan, l'urbanisation et la densité de population sont des facteurs clés influençant le prix du logement.

[9]:
```
Housing=Housing.drop(columns=['longitude','latitude'])
```

2 I- Analyse descriptive du jeu de donnee

Avant de pouvoir construire un modèle prédictif, il est essentiel de comprendre la structure et la distribution des variables du jeu de données. Cette étape permet d'identifier les éventuelles anomalies, les corrélations fortes, ou encore les variables nécessitant un traitement particulier.

2.1 A- Statistiques descriptives

[10]:
```
(Housing[Housing['median_house_value'] == Housing['median_house_value'].min()])
```

[10]:

	housing_median_age	total_rooms	total_bedrooms	population	\
2521	16.0	255.0	73.0	85.0	
2799	19.0	619.0	239.0	490.0	
9188	52.0	803.0	267.0	628.0	
19802	36.0	98.0	28.0	18.0	

	households	median_income	median_house_value	ocean_proximity
2521	38.0	1.6607	14999.0	INLAND
2799	164.0	2.1000	14999.0	INLAND
9188	225.0	4.1932	14999.0	INLAND
19802	8.0	0.5360	14999.0	INLAND

La valeur médiane de logement la plus faible dans notre jeu de données est de 14 999 dollars, ce qui nous donne un premier aperçu sur l'impact potentiel des variables explicatives.

Quatre régions présentent exactement cette valeur minimale. En examinant leur position géographique on remarque qu'elles sont situées dans des zones **non côtières**, ce qui laisse penser que la **localisation géographique** joue un rôle déterminant.

Ces quatre régions les moins couteuses partagent donc une caractéristiques principales : **l'absence d'accès à la côte.** Toutefois, aucune autre variable ne semble les relier de manière évidente, ce qui suggère **une forme d'hétérogénéité non expliquée par les seules variables géographiques.**

Cela renforce l'intérêt d'analyser en profondeur l'effet des autres variables socio-économiques dans notre modèle, notamment pour mieux comprendre les écarts extrêmes dans la distribution des prix.

[11]:
```
(Housing[Housing['median_house_value'] == Housing['median_house_value'].max()]).
↪describe()
```

[11]:

	housing_median_age	total_rooms	total_bedrooms	population	\
count	958.000000	958.000000	958.000000	958.000000	
mean	33.817328	2962.031315	501.782881	1113.026096	
std	13.020803	2167.619458	398.408898	815.263027	
min	2.000000	8.000000	1.000000	13.000000	
25%	24.250000	1668.500000	266.000000	637.000000	
50%	34.000000	2514.500000	401.000000	925.000000	
75%	44.000000	3539.750000	609.500000	1346.250000	
max	52.000000	18132.000000	5419.000000	7431.000000	

	households	median_income	median_house_value
count	958.000000	958.000000	958.0
mean	465.686848	7.823533	500001.0
std	361.908488	3.240095	0.0
min	1.000000	0.499900	500001.0
25%	252.000000	5.242650	500001.0
50%	373.500000	7.721350	500001.0
75%	573.500000	10.109325	500001.0
max	4930.000000	15.000100	500001.0

À l'inverse, si l'on s'intéresse aux zones où les logements sont les plus chers, on remarque qu'elles sont quasiment toutes situées le long de la côte californienne, principalement autour des grandes métropoles comme San Francisco, Los Angeles et San Diego.

Ces zones affichent **une valeur médiane plafonnée à 500 001 dollars**, ce qui correspond à la valeur maximale autorisée dans le dataset.

Il est question de 938 zones partageant plusieurs caractéristiques communes : un revenu médian

élevé, une densité de population importante, une forte concentration de ménages, et bien sûr, **une proximité de l'océan** (ocean_proximity étant souvent égal à NEAR OCEAN ou NEAR BAY). Cela correspond au informations que nous avons tiré de la carte plus haut.

Contrairement aux zones les moins chères, qui n'avaient en commun que leur isolement géographique, les zones les plus chères cumulent plusieurs facteurs favorables. Cela confirme l'idée que certains effets peuvent se renforcer mutuellement.

[12]:
```
formula = 'median_house_value ~ C(ocean_proximity)'
model = smf.ols(formula, data=Housing).fit()
print(smsa.anova_lm(model, typ=2))
```

```
                          sum_sq       df             F  PR(>F)
C(ocean_proximity)  6.478688e+13      4.0  1594.710397     0.0
Residual            2.074776e+14  20428.0          NaN     NaN
```

Les résultats de cette ANOVA (p-value = 0.0) indiquent que la proximité à l'océan explique une part substantielle de la variance du prix des logements, ce qui confirme les observations précédentes.

Elle capte des différences structurelles importantes entre les zones côtières et les zones intérieures en Californie.

[13]:
```
Housing.describe()
```

[13]:

	housing_median_age	total_rooms	total_bedrooms	population	\
count	20433.000000	20433.000000	20433.000000	20433.000000	
mean	28.633094	2636.504233	537.870553	1424.946949	
std	12.591805	2185.269567	421.385070	1133.208490	
min	1.000000	2.000000	1.000000	3.000000	
25%	18.000000	1450.000000	296.000000	787.000000	
50%	29.000000	2127.000000	435.000000	1166.000000	
75%	37.000000	3143.000000	647.000000	1722.000000	
max	52.000000	39320.000000	6445.000000	35682.000000	

	households	median_income	median_house_value
count	20433.000000	20433.000000	20433.000000
mean	499.433465	3.871162	206864.413155
std	382.299226	1.899291	115435.667099
min	1.000000	0.499900	14999.000000
25%	280.000000	2.563700	119500.000000
50%	409.000000	3.536500	179700.000000
75%	604.000000	4.744000	264700.000000
max	6082.000000	15.000100	500001.000000

Les variables **population, total_rooms et total_bedrooms** présentent une grande dispersion, comme en témoignent leurs écarts-types particulièrement élevés (respectivement 1133 - 2185 421). Cela reflète une forte hétérogénéité entre les zones résidentielles, certaines étant très denses et urbanisées, tandis que d'autres sont beaucoup plus rurales ou faiblement peuplées.

Cette variabilité importante peut entraîner des effets extrêmes dans la modélisation, surtout dans le choix de nos à priori.

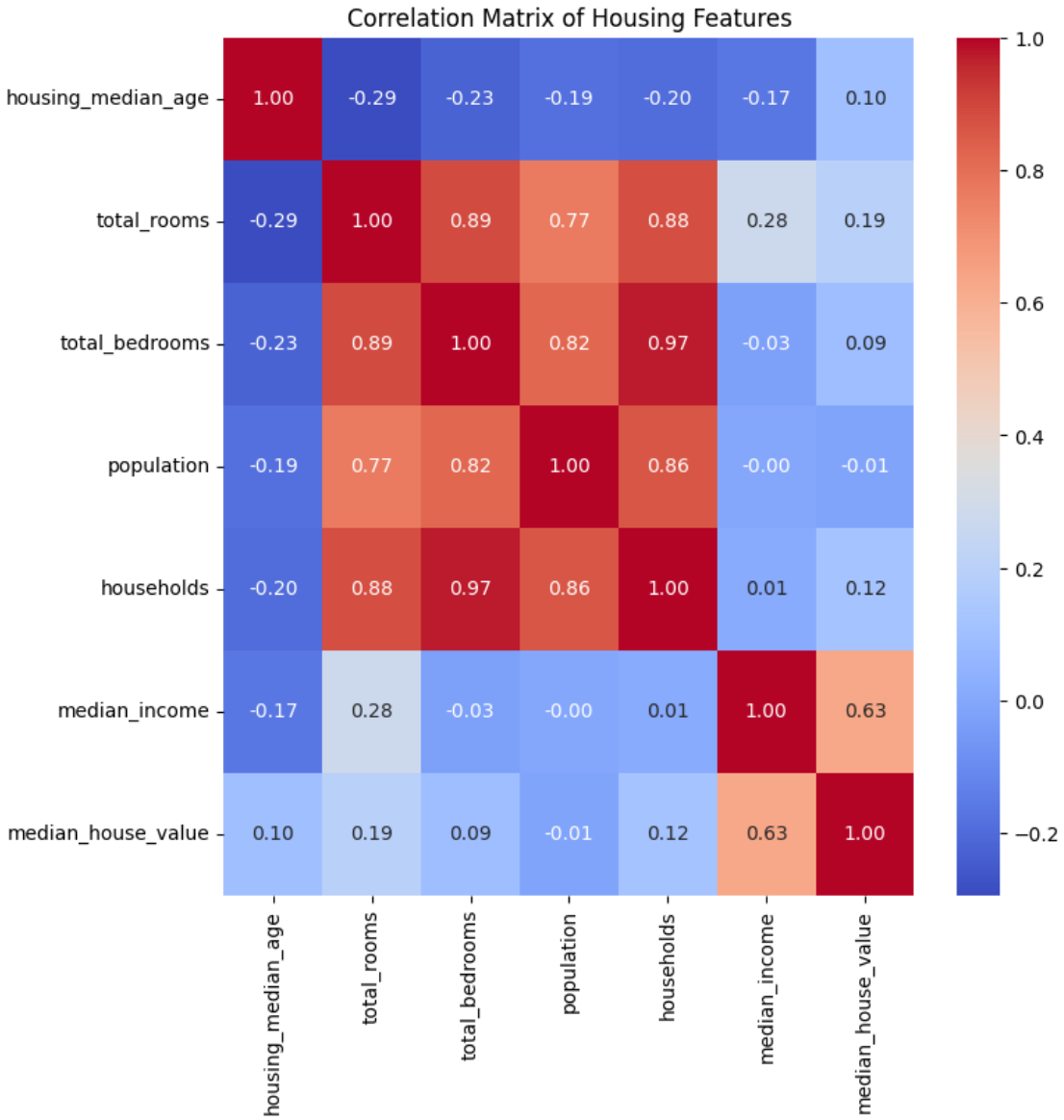
Je fais donc le choix **de retirer les valeurs extremes** pour augmenter la robustesse.

[14]:
```
numerical_features = ['housing_median_age', 'total_rooms', 'total_bedrooms',␣
 ↪'population', 'households', 'median_income', 'median_house_value']
Q1 = Housing[numerical_features].quantile(0.25)
Q3 = Housing[numerical_features].quantile(0.75)
IQR = Q3 - Q1

# Define bounds for outlier removal
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Remove outliers
Housing = Housing[~((Housing[numerical_features] < lower_bound) |␣
 ↪(Housing[numerical_features] > upper_bound)).any(axis=1)]
```

2.2 B- Analyse des corrélations

Il est essentiel d'examiner les relations entre les variables numériques du dataset afin de repérer les éventuelles redondances, les relations linéaires fortes ou encore des risques de multicolinéarité des variables explicatives qui pourraient biaiser l'interprétation des coefficients dans une régression linéaire multiple.

[15]:
```
x=Housing.drop(columns=['ocean_proximity'])
plt.figure(figsize=(8, 8))
sns.heatmap(x.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Housing Features')
plt.show()
```



La matrice de corrélation des variables numériques du jeu de données révèle plusieurs tendances intéressantes :

- total_rooms, total_bedrooms, population et households sont très fortement corrélées entre elles (corrélations > 0.85), ce qui confirme leur redondance structurelle : elles décrivent toutes différentes dimensions de la taille ou de la densité d'une zone. **La corrélation entre total_bedrooms et households atteint 0.97, ce qui est quasiment une colinéarité parfaite.**
- les variables comme **housing_median_age et median_income sont modérément corrélées aux autres.** Cela peut être un atout pour la modélisation, car elles pourraient apporter de l'information nouvelle non captée par les autres variables.

- La variable la plus corrélée avec la valeur médiane des logements (median_house_value), qui est notre variable cible, **est median_income (r = 0.63)**. On imagine que les ménages les plus aisés vivent dans les zones où les prix de l'immobilier sont les plus élevés.
- Les autres variables ont des corrélations beaucoup plus faibles avec la cible ce qui suggère qu'elles ont un rôle explicatif secondaire mais quand même **supposément utile**.

2.3 C- Sélection de variable

L'objectif est de sélectionner les variables les plus pertinentes pour prédire 'median_house_value' sans que la colinéarité des variables explicatives réduisent la significativité des coefficients.

1. `total_rooms, total_bedrooms, households` et `population` sont trop fortement corrélées entre elles. Sachant que ces 4 vrariables traduisent la meme information, je decide de garder **"households"** car elle a correlation significative avec la variabe (r=0,12) et elle est la moins redondente. C'est à dire qu'elle est moins correle aux autres variables explicatves,
2. **median_income** a une corrélation forte avec le prix medians des logements (r = 0.63) donc il s'agit d'une variable pertinente.
3. La variable catégorielle **ocean_proximity** s'est montré très pertinente selon la mesure de l'ANOVA.
4. La variable **housing_median_age** aussi est peu corrélée aux autres, mais liée à la variable cible ce qui fait d'elle une bonne variable explicative.

[16]:
```
Housing=Housing.drop(columns=['total_rooms','total_bedrooms','population'])
Og_Housing=Housing.copy()
```

3 II- Régression linéaire bayésienne

Après avoir sélectionné les variables explicatives les plus pertinentes, nous cherchons désormais à modéliser la relation entre ces variables et la valeur médiane des logements à l'aide d'une régression linéaire.

L'intérêt de cette démarche est, d'une part, de modéliser l'incertitude des paramètres du modèle via des distributions a posteriori.

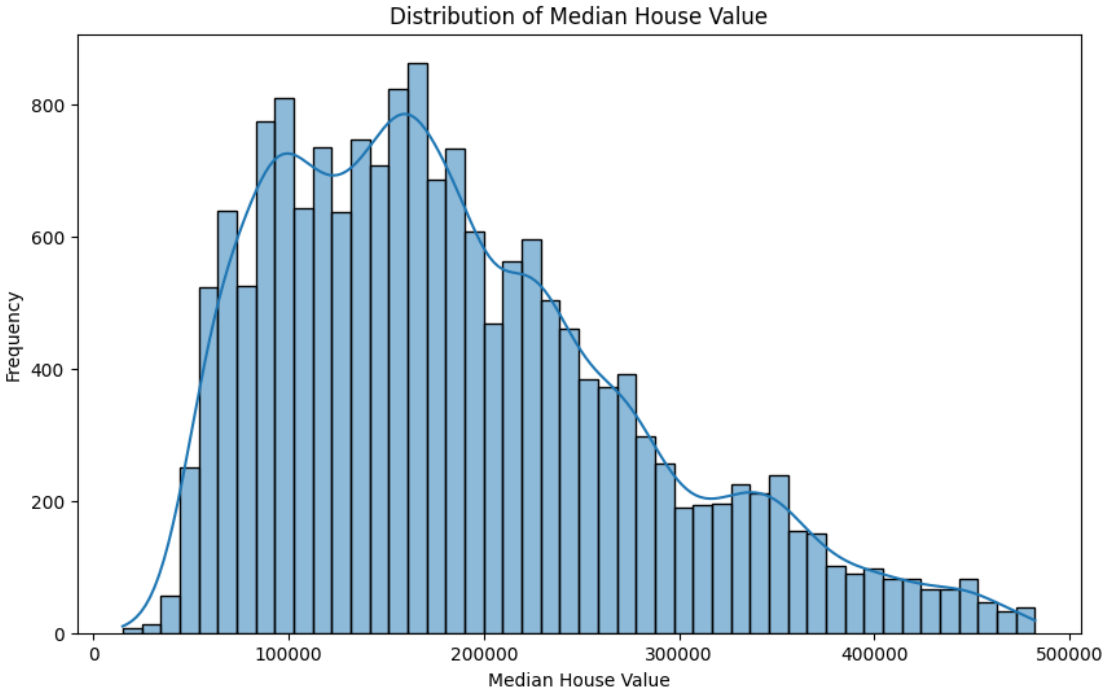
D'autre part, elle offre une flexibilité importante pour intégrer des connaissances a priori ou pour modéliser des effets complexes (non-linéarité, erreurs hétéroscédastiques, etc.).

Pour se faire, nous allons considérer une relation linéaire simple entre la valeur médiane des logements (median_house_value) et un sous-ensemble de variables explicatives sélectionnées précédemment.

Les coefficients associés à chaque variable seront estimés via une inférence bayésienne à l'aide de l'échantillonnage MCMC (Markov Chain Monte Carlo).

[17]:
```
# Distribution of median_house_value
plt.figure(figsize=(10, 6))
sns.histplot(Housing['median_house_value'], kde=True)
plt.title('Distribution of Median House Value')
```

```
plt.xlabel('Median House Value')
plt.ylabel('Frequency')
plt.show()
```



La variable median_house_value est une variable continue dont la distribution présente une queue lourde ainsi qu'une asymétrie marquée. Ce comportement indique que les logements très chers sont relativement fréquents.

Nous savons que le choix de la distribution à utiliser comme vraisemblance dépend a fois de la forme empirique de la variable et de la robustesse souhaitée du modèle face aux outliers. Voila pourquoi nous optons pour differents modèles dont ->

- Une distribution skew-normal permet de mieux capturer l'asymétrie observée. Ou,
-

3.1 une distribution de Student est robuste aux valeurs extrêmes

3.2 Pré-traitement

Avant tout, il est nécessaire de faire un prétraitement sur nos données d'observation

Le jeu de données original contient plus de 10 000 observations, ce qui peut représenter une charge computationnelle importante, notamment pour des traitements plus lourds comme l'inférence bayésienne avec PyMC.

Afin d'alléger cette contrainte et d'assurer un traitement plus fluide, un sous-échantillonnage a été réalisé à l'aide de la fonction `train_test_split` de sklearn.

[18]:
```
from sklearn.model_selection import train_test_split

# Échantillonnage stratifié basé sur la colonne 'ocean_proximity'
Housing, _ = train_test_split(Og_Housing, test_size=0.80,␣
 ↪stratify=Og_Housing['ocean_proximity'], random_state=42)
housing2=Housing.copy()
len(Housing),len(Og_Housing)
```

[18]:
```
(3486, 17434)
```

[ ]:
```

```

Dans mon cas, j'ai choisi de conservé **20 % des données ce qui correspond à 3486 lignes.**

L'échantillonnage a été effectué de manière stratifiée selon la variable `ocean_proximity` afin de préserver la représentativité des différentes catégories géographiques présentes dans le jeu initial.

L'objectif étant de pouvoir travailler efficacement avec un échantillon tout en conservant la diversité structurelle des données.

La variable ocean_proximity contient des valeurs catégorielles, ce qui n'est pas directement exploitable dans un modèle de régression. Pour pouvoir l'utiliser, on doit la transformer en variables numériques via un encodage approprié.

[19]:
```
location_dummies = pd.get_dummies(Housing['ocean_proximity'], prefix='loc_',␣
 ↪drop_first=True)

Housing = pd.concat([Housing, location_dummies], axis=1)
house_value=Housing['median_house_value'].values
# Supprimer la colonne d'origine 'location' si plus nécessaire
Housing = Housing.drop('ocean_proximity', axis=1)
Housing2 = Housing.copy()
```

On peut maintenant inclure ocean_proximity dans le modèle de régression linéaire bayésienne via ses dummies.

[20]:
```
Housing.describe(),print('\n'),Og_Housing.describe()
```

[20]:
```
(       housing_median_age   households  median_income  median_house_value
 count         3486.000000  3486.000000    3486.000000         3486.000000
 mean            29.499713   418.240964       3.559972       185848.909639
 std             12.229952   204.165258       1.454313        94609.486077
 min              1.000000     3.000000       0.499900        14999.000000
 25%             19.000000   271.000000       2.446025       110975.000000
 50%             30.000000   390.000000       3.375000       168800.000000
 75%             38.000000   539.000000       4.475900       240900.000000
 max             52.000000  1057.000000       7.981300       479500.000000,
```

```
None,
       housing_median_age    households  median_income  median_house_value
count        17434.000000  17434.000000   17434.000000        17434.000000
mean            29.489216    416.806126       3.575593       187056.188826
std             12.214437    205.412687       1.443540        93742.859159
min              1.000000      2.000000       0.499900        14999.000000
25%             19.000000    271.000000       2.490150       112900.000000
50%             30.000000    387.000000       3.390600       170100.000000
75%             38.000000    540.000000       4.487950       241600.000000
max             52.000000   1090.000000       8.011300       482200.000000)
```

Remarque :

On note que le sous échantillon reste statistiquement représentatif des données observé, ce qui en fait une base solide pour construire et tester des modèles sans surcharger les ressources machine.

Afin d'entraîner efficacement un modèle de régression, il est nécessaire de mettre les variables sur une échelle comparable, surtout lorsqu'elles ont des ordres de grandeur très différents (ex : revenu médian vs âge médian des logements).

Cela permet également d'accélérer la convergence des algorithmes d'inférence, en particulier lors de l'échantillonnage MCMC.

Le code suivant effectue une standardisation (centrage-réduction) sur les colonnes numériques clés :

[21]:
```
# --- Standardisation
cols_to_scale = ['median_income', 'housing_median_age',␣
 ↪'households','median_house_value']

scalers = {}
for col in cols_to_scale:
    scaler = StandardScaler()
    Housing[col] = scaler.fit_transform(Housing[[col]])
    scalers[col] = scaler  # on garde chaque scaler pour pouvoir␣
 ↪inverse_transform plus tard

cols = ['median_income', 'housing_median_age', 'households']
X = Housing[cols + location_dummies.columns.tolist()].values
X = X.astype(float)
y = Housing['median_house_value'].values

name=['median_income', 'housing_median_age', 'households']+location_dummies.
 ↪columns.tolist()
```

- Chaque variable est transformée pour avoir une moyenne de 0 et un écart-type de 1.
- Les objets `StandardScaler` sont stockés dans un dictionnaire (scalers) pour pouvoir effectuer un inverse transform plus tard, notamment pour réinterpréter les résultats dans leur unité d'origine.

3.3 ## A- Modèle linéaire avec distribution normale

Le modèle construit ici est une régression linéaire bayésienne utilisant une vraisemblance normale pour modéliser la variable cible y (la valeur médiane des logements).

- alpha est l'ordonnée à l'origine (l'intercept), supposée suivre une distribution normale centrée réduite (mu=0, sigma=5).
- beta contient les coefficients de régression pour chaque variable explicative. On les suppose également issus d'une loi normale centrée, ce qui permet d'introduire une incertitude a priori symétrique autour de zéro.

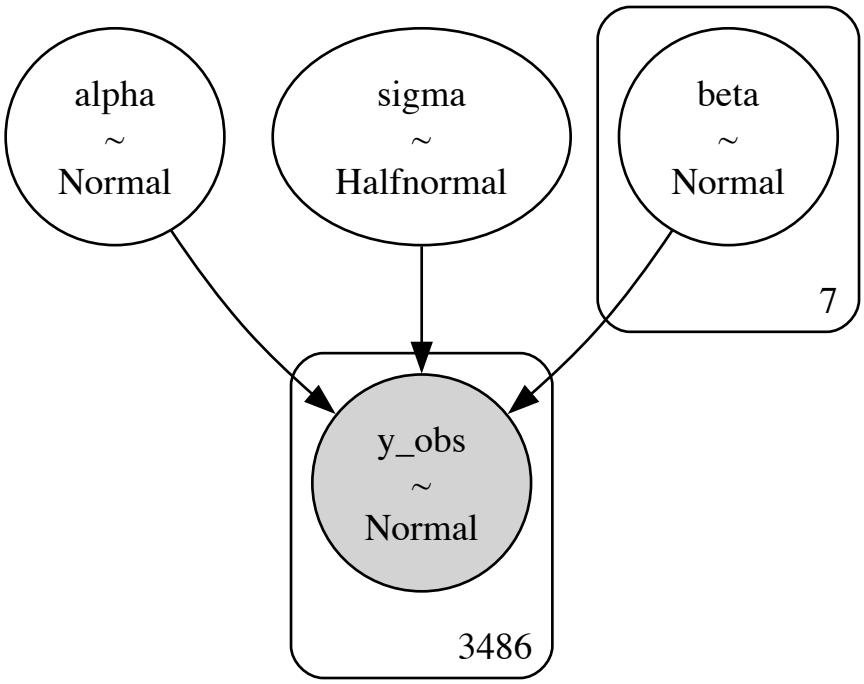
[22]:
```
with pm.Model() as normal_linear_model:
    # Ensure X is of numeric type, for instance by converting to float:

    # Priors pour les coefficients
    alpha = pm.Normal('alpha', mu=0, sigma=5)
    beta = pm.Normal('beta', mu=0, sigma=5, shape=X.shape[1])  # Shape␣
 ↪correspond à la dimension de X (nombre de features)

    # Priors pour l'erreur
    sigma = pm.HalfNormal('sigma', sigma=1)

    # La régression linéaire
    mu = alpha + pm.math.dot(X, beta)

    # La vraisemblance (ou l'observation)
    y_obs = pm.Normal('y_obs', mu=mu, sigma=sigma, observed=y)
```

Ce modèle est intéressant car il permet non seulement d'obtenir des estimations ponctuelles des coefficients, mais aussi des intervalles de crédibilité, ce qui donne une idée de l'incertitude autour de chaque paramètre.

Il est facile à étendre (on pourrait remplacer Normal par SkewNormal ou StudentT pour capter mieux la forme des données).

[23]:
```
pm.model_to_graphviz(normal_linear_model)
```

[23]:



[24]:
```
with normal_linear_model:
    trace1 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta, sigma]
```

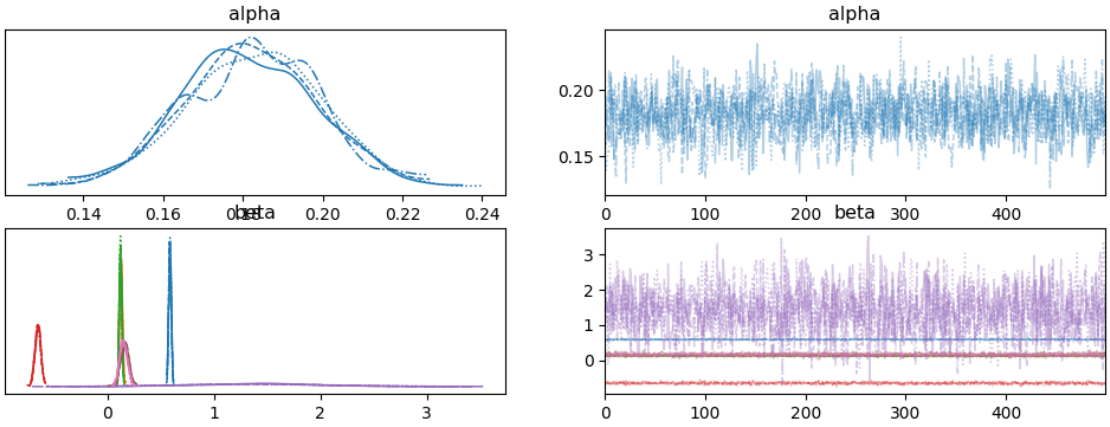
```
Output()
```

```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 2 seconds.
```

[25]:
```
trace1.posterior = trace1.posterior.assign_coords(beta_dim_0=name)
```

[26]:
```
az.plot_trace(trace1, var_names=['alpha', 'beta'])
```

[26]:
```
array([[<Axes: title={'center': 'alpha'}>,
        <Axes: title={'center': 'alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



Ces tracés permettent de visualiser le comportement des chaînes au cours de la simulation et d'évaluer si elles ont convergé vers la distribution cible.

Un mélange adéquat des chaînes et une stationnarité apparente suggèrent une bonne convergence.

3.4 B- Modèle de régression de Poisson

Ici, on modélise la variable cible y (décalée par +3) en supposant qu'elle suit une distribution de Poisson, adaptée à des comptages ou données entières positives.

On ajuste aussi les variables explicatives X et la target y avec un décalage +3 pour éviter les problèmes de log sur des valeurs potentiellement nulles ou négatives.

La Poisson est définie pour k , mais certains algos peuvent buguer avec trop de zéros).

- alpha est l'intercept.
- beta sont les coefficients associés à chaque variable explicative.

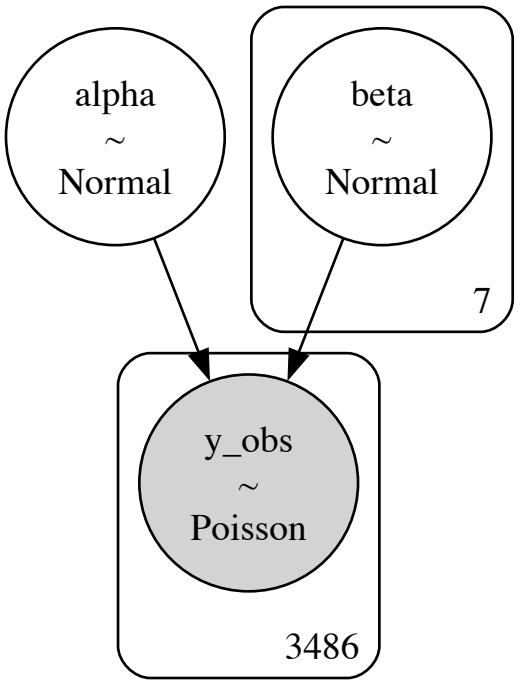
Tous deux sont supposés a priori normaux : pas de connaissance spécifique mais centrés autour de zéro, avec une variance modérée.

On suppose que la variable réponse suit une loi de Poisson paramétrée par mu.

[27]:
```
A=X+3
with pm.Model() as poisson_linear_model:
# Ensure A is of numeric type, for instance by converting to float:
    alpha = pm.Normal('alpha', mu=0, sigma=5)  # Intercept
    beta = pm.Normal('beta', mu=0, sigma=5, shape=A.shape[1])
    eta = alpha + pm.math.dot(A, beta) # Log-lambda
    mu = pm.math.exp(eta)
    # La vraisemblance : y suit une distribution de Poisson avec le paramètre mu
    y_obs = pm.Poisson('y_obs', mu=mu, observed=y+3)
```

[28]:
```
pm.model_to_graphviz(poisson_linear_model)
```

[28]:



[29]:
```
with  poisson_linear_model:
    trace2 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta]
```

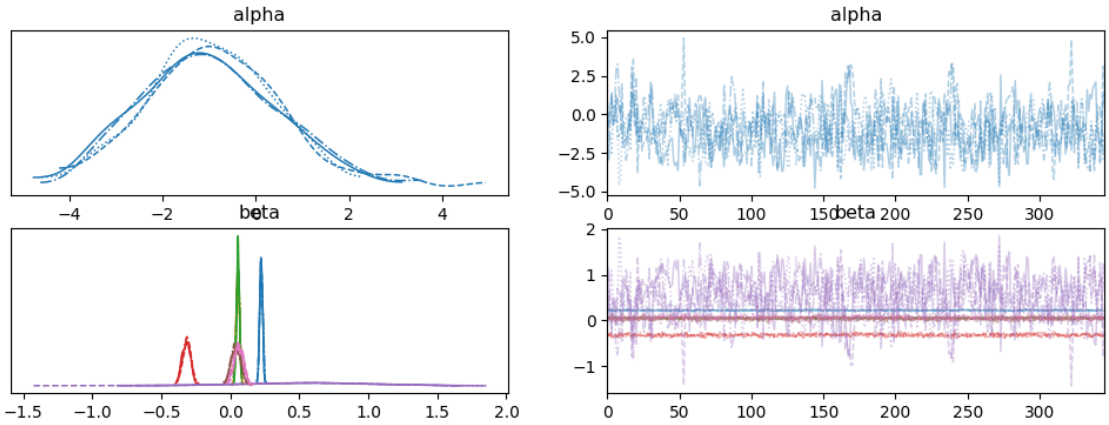
```
Output()
```

```
Sampling 4 chains for 500 tune and 346 draw iterations (2_000 + 1_384 draws
total) took 28 seconds.
Chain 0 reached the maximum tree depth. Increase `max_treedepth`, increase
`target_accept` or reparameterize.
Chain 1 reached the maximum tree depth. Increase `max_treedepth`, increase
`target_accept` or reparameterize.
Chain 3 reached the maximum tree depth. Increase `max_treedepth`, increase
`target_accept` or reparameterize.
```

[31]:
```
trace2.posterior = trace2.posterior.assign_coords(beta_dim_0=name)
```

[32]:
```
az.plot_trace(trace2, var_names=['alpha', 'beta'])
```

[32]:
```
array([[<Axes: title={'center': 'alpha'}>,
        <Axes: title={'center': 'alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



3.5 C- Modèle de régression linéaire avec distribution SkewNormal

On cherche ici à capturer une asymétrie dans la distribution de la variable cible y (median_house_value).

Contrairement au modèle gaussien classique (symétrique), on suppose que les erreurs suivent une loi normale asymétrique, dite SkewNormal, avec un paramètre qui contrôle la direction et l'intensité de l'asymétrie.

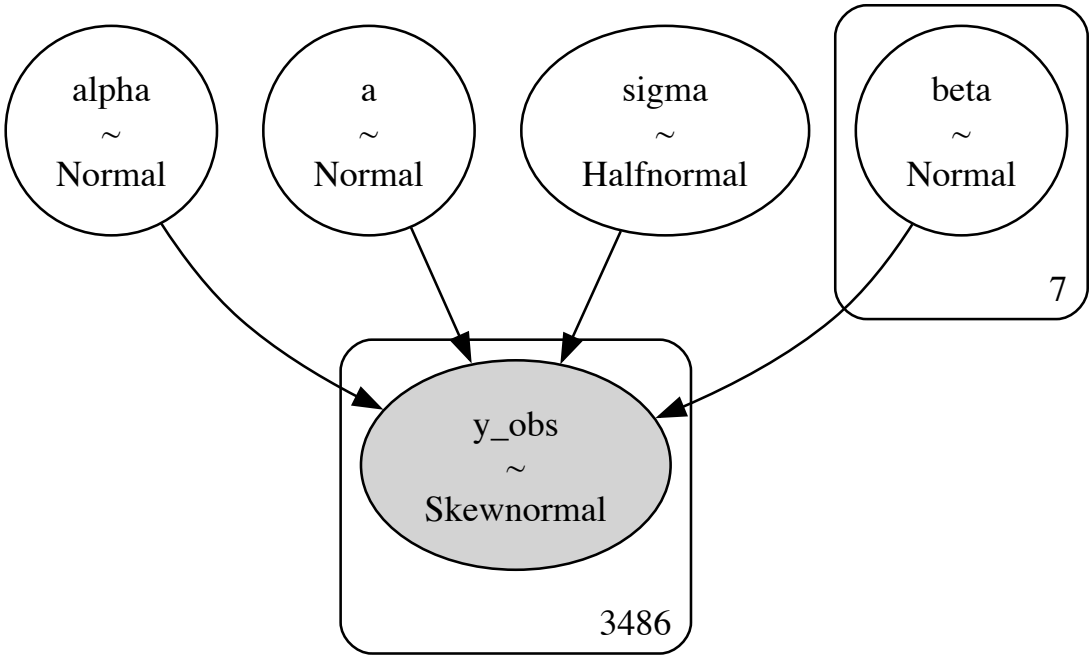
On desire capturer une distribution déséquilibrée (skewed), qui reflète mieux la réalité du marché immobilier californien, où les prix peuvent présenter une longue queue à droite (logements très chers mais rares).

[33]:
```
with pm.Model() as skewnormal_linear_model:
    # Ensure X is of numeric type, for instance by converting to float:
    X = X.astype(float)  # Convert X to float dtype

    alpha = pm.Normal('alpha', mu=0, sigma=5)
    beta = pm.Normal('beta', mu=0, sigma=5, shape=X.shape[1])
    a= pm.Normal("a", mu=0, sigma=5)

    sigma = pm.HalfNormal('sigma', sigma=1)

    # La régression linéaire
    mu = alpha + pm.math.dot(X, beta)

    # La vraisemblance
    y_obs = pm.SkewNormal('y_obs', mu=mu, sigma=sigma, alpha=a, observed=y)
```

- alpha : l'intercept de la régression.
- beta : les coefficients associés à chaque variable explicative.
- sigma : l'écart-type des résidus.
- a : le paramètre d'asymétrie de la SkewNormal
  Si a > 0, la queue droite est plus lourde (positive skew).
  Si a < 0, la queue gauche est plus lourde (negative skew).

[34]:
```
pm.model_to_graphviz(skewnormal_linear_model)
```

[34]:



[35]:
```
with skewnormal_linear_model:
    trace3 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta, a, sigma]
```

```
Output()
```

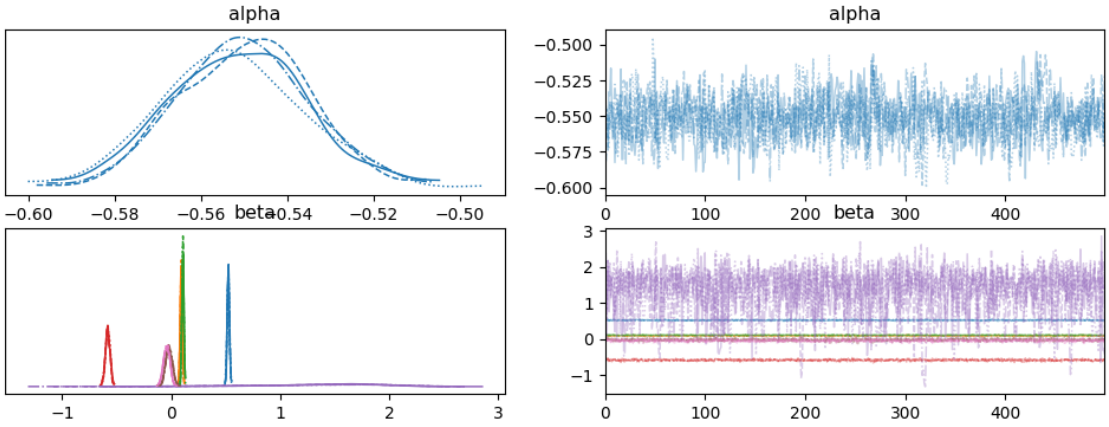
```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 8 seconds.
```

Pourquoi ce modèle est une bonne idée ici ? Le modèle SkewNormal permet une meilleure flexibilité tout en restant relativement simple et rapide à inférer. Il donne une meilleure robustesse face aux outliers ou à la non-normalité des résidus.

[36]:
```
trace3.posterior = trace3.posterior.assign_coords(beta_dim_0=name)
```

[37]:
```
az.plot_trace(trace3, var_names=['alpha', 'beta'])
```

[37]:
```
array([[<Axes: title={'center': 'alpha'}>,
        <Axes: title={'center': 'alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



3.6 D- Modèle bayésien Bimodal

Ce type de modèle est utile lorsque la variable cible (y) semble être générée à partir de deux distributions différentes.

On s'attaques à un modèle mixte, où il y a deux distributions avec une probabilité différente d'appartenance à chacune.

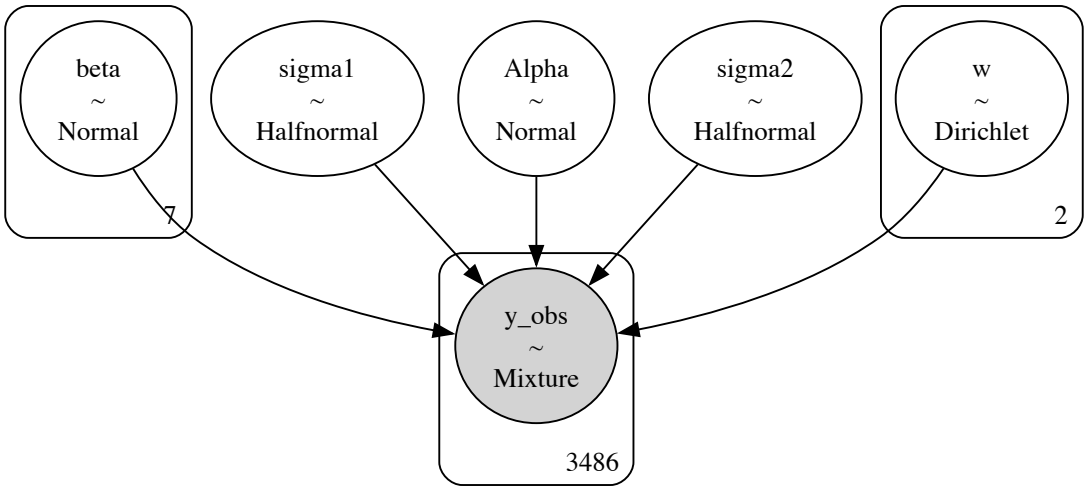
Cela peut refléter des comportements différents dans les prix des logements selon les régions, les quartiers, etc.

[38]:
```
# Définition du modèle PyMC Bimodal
with pm.Model() as model_linear_bimodal:
     #Priors sur les coefficients (normaux centrés)
    coefs = pm.Normal("beta", mu=0, sigma=1, shape=X.shape[1])
    intercept = pm.Normal("Alpha", mu=0, sigma=1)


    # Moyennes et écarts-types des deux composants
    # Modèle linéaire
    mu1 = intercept + pm.math.dot(X, coefs)
    mu2 = intercept + pm.math.dot(X, coefs) -5
    sigma1 = pm.HalfNormal("sigma1", sigma=1)
    sigma2 = pm.HalfNormal("sigma2", sigma=1)
```

```
    w = pm.Dirichlet("w", a=np.ones(2), shape=2)
    # Lien entre les deux distributions
    y_obs = pm.Mixture("y_obs", w=w,
                       comp_dists=[
                           pm.Normal.dist(mu=mu1, sigma=sigma1),
                           pm.Normal.dist(mu=mu2, sigma=sigma2)
                       ],
                       observed=y)
```

[39]:
```
pm.model_to_graphviz(model_linear_bimodal)
```

[39]:



- coefs sont les coefficients de régression pour chaque feature, avec des prioris centrés et une variance de 1 (modéré).
- intercept est l'ordonnée à l'origine, supposée aussi suivre une distribution normale centrée.
- mu1 et mu2 sont les moyennes des deux distributions normales qui forment le modèle bimodal.

  mu1 est le modèle de base.

  mu2 est décalé par -5, ce qui signifie qu'il correspond à un groupe différent dans le modèle, avec une moyenne plus faible.
- sigma1 et sigma2 sont les écarts-types de chaque composant, modélisés par des distributions HalfNormal (ce qui les contraint à être positifs).
- w représente les poids des deux composants dans la mixture. Il suit une distribution Dirichlet, ce qui donne les probabilités d'appartenance des données à chaque groupe.

  Si le poids de w[0] est plus grand que celui de w[1], alors le premier groupe (avec mu1) est plus représenté dans les données, et vice-versa.

[40]:
```
with model_linear_bimodal:
    trace4 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, Alpha, sigma1, sigma2, w]
```

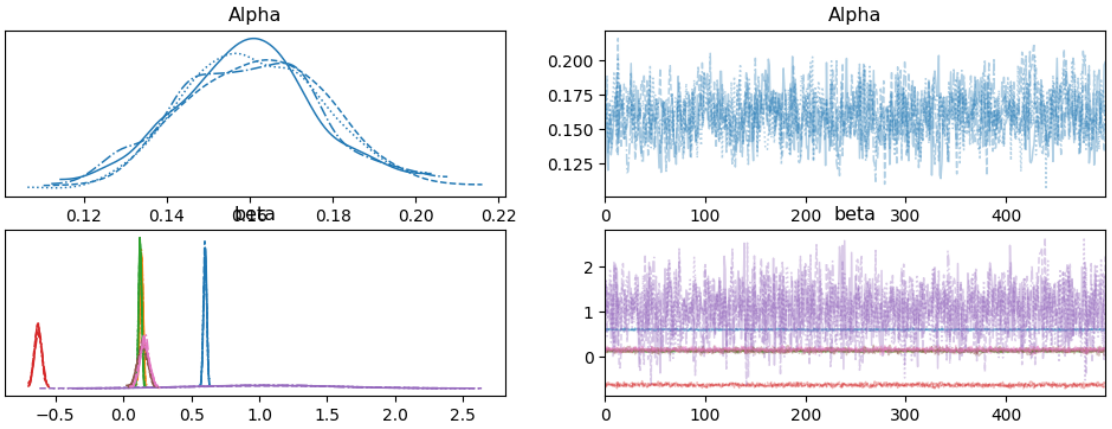
```
Output()
```

```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 12 seconds.
```

[41]:
```
trace4.posterior = trace4.posterior.assign_coords(beta_dim_0=name)
```

[42]:
```
az.plot_trace(trace4, var_names=['Alpha', 'beta'])
```

[42]:
```
array([[<Axes: title={'center': 'Alpha'}>,
        <Axes: title={'center': 'Alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



3.7 E- Modèle linéaire de régression avec distribution Student-t

Ce modèle suppose que la distribution des prix de maison median suit une loi de Student-t, ce qui peut est utile car la variable presente une que plus lourdes que celles d'une loi normale classique.

Cela permet de mieux gérer les outliers ou des distributions non symétriques, ce qui est particulièrement pertinent dans des cas comme celui des prix immobiliers où des valeurs extrêmes peuvent être présentes. La distribution Student-t est plus robuste aux outliers que la normale et il n'est pas aussi sensible aux valeurs extrêmes.

La distribution Student-t permet aussi une meilleure modélisation de la variance dans les données,

ce qui peut être crucial si les erreurs ne suivent pas une distribution normale, mais présentent des variations plus importantes.

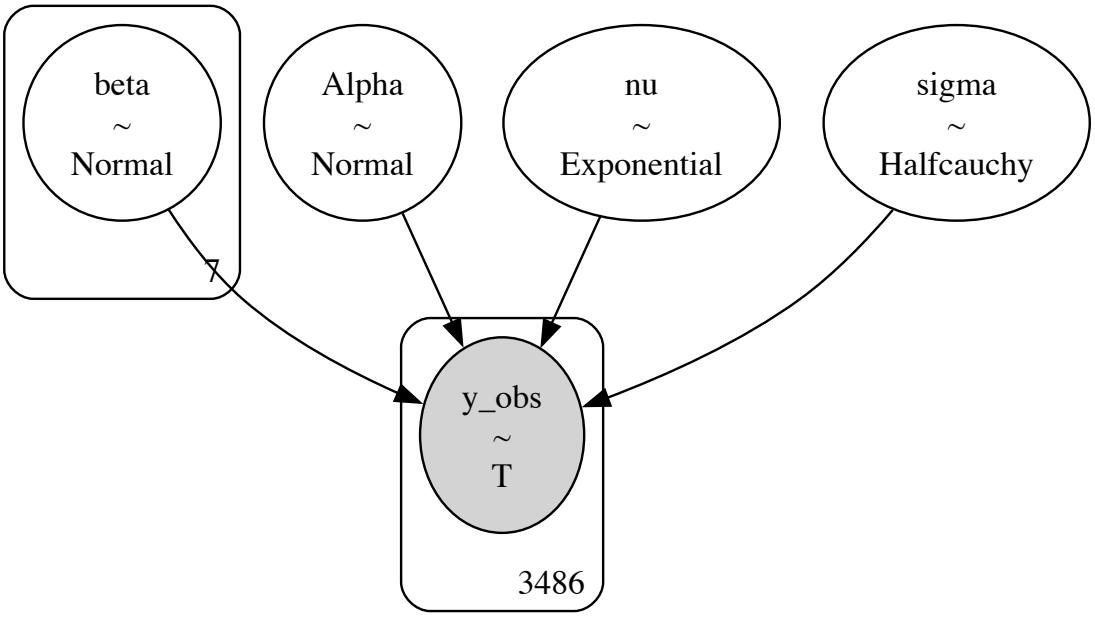
[43]:
```
with pm.Model() as model_student_t:
    # Priors pour les coefficients de la régression
    coefs = pm.Normal("beta", mu=0, sigma=1, shape=X.shape[1])
    intercept = pm.Normal("Alpha", mu=0, sigma=1)

    # Prior sur la variance de l'erreur (lié à la distribution Student-t)
    nu = pm.Exponential("nu", 1)  # Degrés de liberté pour Student-t
    sigma = pm.HalfCauchy("sigma", beta=1)  # Écart-type de l'erreur

    # Modèle de régression linéaire
    mu = intercept + pm.math.dot(X, coefs)

    # Likelihood avec distribution Student-t pour les erreurs
    y_obs = pm.StudentT("y_obs", mu=mu, sigma=sigma, nu=nu, observed=y)
```

- coefs sont les coefficients de régression pour chaque variable explicative. Ils sont supposés suivre une distribution normale centrée autour de zéro avec une variance de 1 à priori.
- intercept représente l'ordonnée à l'origine, supposée aussi suivre une distribution normale centrée avec une variance de 1.
- nu est le paramètre des degrés de liberté pour la distribution Student-t. Cela régit l'épaisseur de la queue de la distribution.

  Plus nu est petit, plus la distribution est lourde et capable de gérer des outliers extrêmes.
- sigma est l'écart-type des erreurs, modelé comme une demi-cauchy pour garantir qu'il soit positif.

[44]:
```
pm.model_to_graphviz(model_student_t)
```

[44]:



y_obs, la variable observée, suit une distribution Student-t avec les paramètres calculés précédemment : - mu : la moyenne de la distribution (prédicteur linéaire), - sigma : l'écart-type des erreurs, - nu : les degrés de liberté (déterminant l'épaisseur de la queue).

[45]:
```
with model_student_t:
    trace5 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, Alpha, nu, sigma]
```

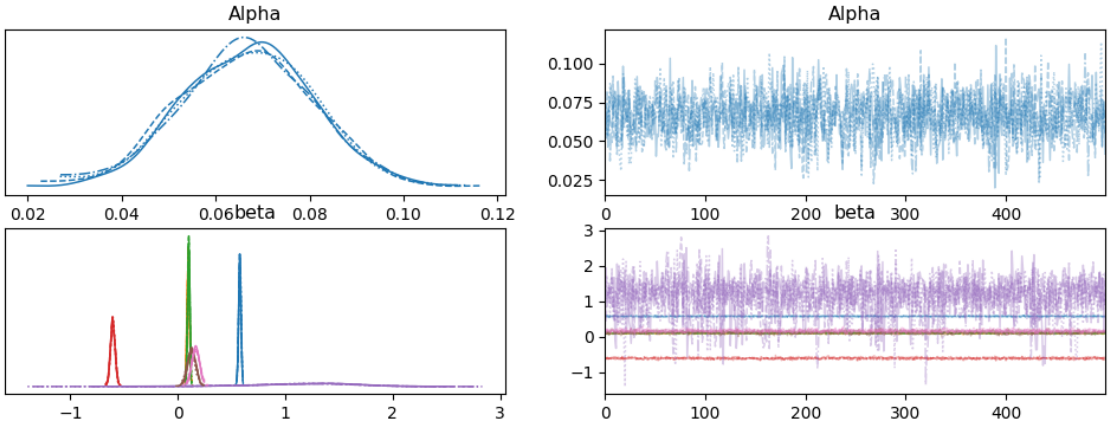
```
Output()
```

```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 3 seconds.
```

[46]:
```
trace5.posterior = trace5.posterior.assign_coords(beta_dim_0=name)
```

[47]:
```
az.plot_trace(trace5, var_names=['Alpha', 'beta'])
```

[47]:
```
array([[<Axes: title={'center': 'Alpha'}>,
        <Axes: title={'center': 'Alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



3.8 F- Modèle de régression avec distribution Skewed Student-t

Le modèle **Skewed Student-t** est une extension de la distribution **Student-t** permettant de modéliser des données avec une **asymétrie** (ou "skewness").

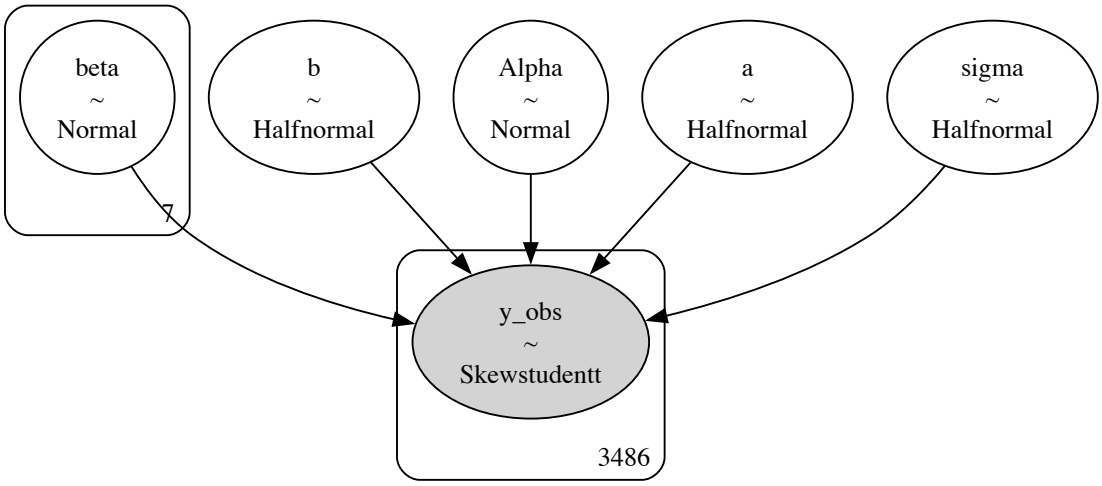
Ce modèle est particulièrement utile lorsque les erreurs dans le modèle de régression ne sont pas seulement **plus lourdes** que celles d'une loi normale, mais également **dérivées** dans une certaine direction (asymétrie).

Ce modèle est adapté si la distribution des erreurs présente une **asymétrie** (par exemple, des prix de logements où certaines valeurs sont beaucoup plus grandes que la majorité). En utilisant une distribution **Skewed Student-t**, tu peux mieux capturer cette **asymétrie** tout en restant robuste aux **outliers**.

[48]:
```
with pm.Model() as model_skewt:

    # Priors
    beta = pm.Normal("beta", mu=0, sigma=1, shape=X.shape[1])
    alpha0 = pm.Normal("Alpha", mu=0, sigma=1)

    mu = alpha0 + pm.math.dot(X, beta)
    sigma = pm.HalfNormal("sigma", sigma=1)
    a = pm.HalfNormal("a", sigma=10)  # Paramètre d'asymétrie
    b = pm.HalfNormal("b", sigma=10)  # Paramètre d'asymétrie
    # Likelihood
    y_obs = pm.SkewStudentT("y_obs", mu=mu, sigma=sigma, a=a, b=b, observed=y)
```

[49]:
```
pm.model_to_graphviz(model_skewt)
```

[49]:



- **beta** : Les **coefficients de régression** pour chaque variable explicative sont supposés suivre une distribution **normale** avec une moyenne de 0 et une variance de 1. Cela permet de ne pas imposer de biais initial et de laisser le modèle apprendre les relations.
- **Alpha** : L'**intercept** suit également une **distribution normale** centrée autour de 0, avec une variance de 1.
- **sigma** : L'**écart-type des erreurs** suit une distribution **HalfNormal** avec une échelle de 1. Cela assure que les erreurs sont toujours positives.
- **a et b** : Les **paramètres d'asymétrie** (skewness) sont modélisés par des **distributions HalfNormal** avec une échelle de 10. Ces paramètres permettent de capturer une **asymétrie positive ou négative** dans la distribution des erreurs. Le paramètre **a** contrôle la direction de l'asymétrie, tandis que **b** régule l'intensité.

[50]:
```
with model_skewt:
    trace6 = pm.sample(500, tune=500, target_accept=0.95,␣
 ↪idata_kwargs={'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, Alpha, sigma, a, b]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 8 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates
problems during sampling. See https://arxiv.org/abs/1903.08008 for details
```

[51]:
```
trace6.posterior = trace6.posterior.assign_coords(beta_dim_0=name)
```

3.9 G- Modèle Bayésien de Régression avec une Mixture de Skewed Student-t

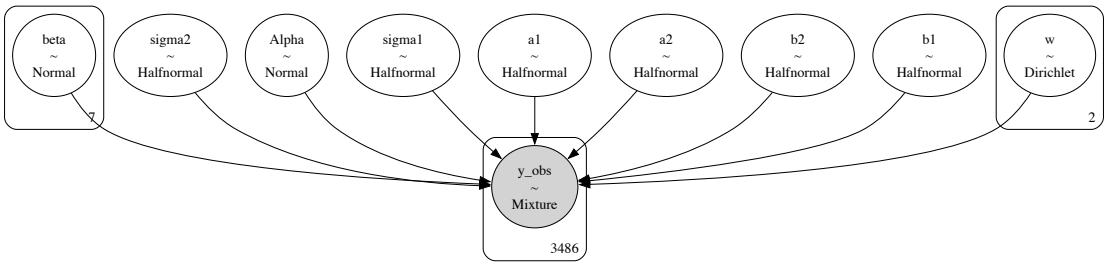
Ce modèle est une extension d'un modèle de régression linéaire avec une **mixture de distributions Skewed Student-t**. L'idée est de modéliser une **distribution bimodale** tout en tenant compte de l'**asymétrie** de ces erreurs.

- **`beta`** : Les **coefficients de régression** sont supposés suivre une **distribution normale** centrée autour de zéro avec une variance de 1. Ces coefficients représentent l'impact des variables explicatives sur la variable cible `y`.
- **`intercept`** : L'**ordonnée à l'origine** (intercept) suit également une distribution **normale** centrée autour de zéro.
- **`sigma1` et `sigma2`** : Les **écarts-types des erreurs** pour chaque composante de la mixture suivent des distributions **HalfNormal** avec une échelle de 1. Cela garantit que l'écart-type des erreurs soit positif et permet de modéliser la variance de manière flexible.
- **`a1`, `b1`, `a2`, `b2`** : Les **paramètres d'asymétrie** (skewness) sont modélisés par des **distributions HalfNormal** avec une échelle de 10. Ces paramètres contrôlent l'asymétrie de chaque composante de la mixture, capturant des comportements non symétriques dans les erreurs.

[52]:
```
with pm.Model() as skew_mixture_model:
    # Coefficients linéaires
    beta = pm.Normal("beta", mu=0, sigma=1, shape=X.shape[1])
    intercept = pm.Normal("Alpha", mu=0, sigma=1)

    # Moyennes des deux composantes (mêmes coefs mais tu pourrais en avoir␣
 ↪différents)
    mu1 = intercept + pm.math.dot(X, beta)
    sigma1 = pm.HalfNormal("sigma1", sigma=1)
    a1 = pm.HalfNormal("a1", sigma=10)  # Paramètre d'asymétrie
    b1 = pm.HalfNormal("b1", sigma=10)  # Paramètre d'asymétrie

    mu2 = intercept + pm.math.dot(X, beta)-5
    sigma2 = pm.HalfNormal("sigma2", sigma=1)
    a2 = pm.HalfNormal("a2", sigma=10)  # Paramètre d'asymétrie
    b2 = pm.HalfNormal("b2", sigma=10)  # Paramètre d'asymétrie

    # Composantes
    comp_1 = pm.SkewStudentT.dist(mu=mu1, sigma=sigma1, a=a1,b=b1)
    comp_2 = pm.SkewStudentT.dist(mu=mu2, sigma=sigma2, a=a2,b=b2)

    # Mixture

     # Poids de la mixture
    w = pm.Dirichlet("w", a=np.ones(2))  # pour bimodale


    y_obs = pm.Mixture("y_obs", w=w, comp_dists=[comp_1, comp_2], observed=y)
```

[53]:
```
pm.model_to_graphviz(skew_mixture_model)
```

[53]:



En utilisant une mixture de **Skewed Student-t**, j'espere capturer des comportements complexes tout en restant robuste aux valeurs extrêmes et en permettant une meilleure flexibilité que les modèles classiques de régression linéaire.

[54]:
```
with skew_mixture_model:
    trace7 = pm.sample(500, tune=500, target_accept=0.99,␣
 ↪idata_kwargs={'log_likelihood': True})
```

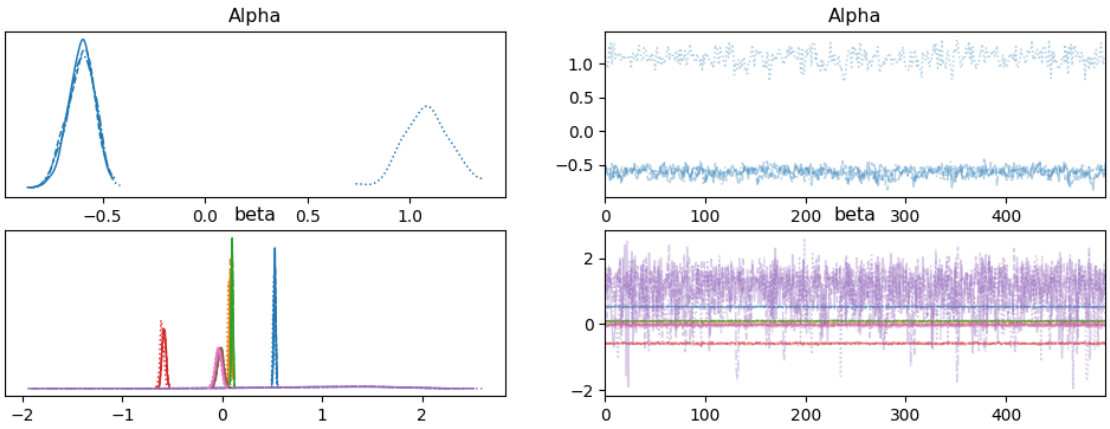
```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, Alpha, sigma1, a1, b1, sigma2, a2, b2, w]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 500 draw iterations (2_000 + 2_000 draws
total) took 73 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates
problems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A
higher number is needed for reliable rhat and ess computation. See
https://arxiv.org/abs/1903.08008 for details
```

[55]:
```
az.plot_trace(trace7, var_names=['Alpha', 'beta'])
```

[55]:
```
array([[<Axes: title={'center': 'Alpha'}>,
        <Axes: title={'center': 'Alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



Comparaison –

Afin de déterminer quel modèle est le plus approprié pour prédire la variable cible `y` (par exemple, la valeur médiane des logements), nous avons comparé plusieurs modèles de régression en utilisant la fonction `pm.compare()`. Les modèles inclus dans la comparaison sont :

1. **Modèle Linéaire Normal** (`normal_linear_model`)
2. **Modèle Linéaire de Poisson** (`poisson_linear_model`)
3. **Modèle Linéaire avec Distribution SkewNormal** (`skewnormal_linear_model`)
4. **Modèle Linéaire Bimodal** (`model_linear_bimodal`)
5. **Modèle avec Distribution Student-t** (`model_student_t`)
6. **Modèle avec Distribution Skew-t** (`model_skewt`)
7. **Modèle de Mixture Skewed Student-t** (`skew_mixture_model`)

On effectue une comparaison entre les différents modèles en utilisant des critères elpd_loo (l'approximation de la log-vraisemblance pour Leave-One-Out Cross Validation) et p_loo (le nombre de paramètres utilisés pour l'ajustement).

[56]:
```
comparaison=pm.compare({
    "normal_linear_model":trace1,
    "poisson_linear_model":trace2,
    "skewnormal_linear_model":trace3,
   " model_linear_bimodal":trace4,
    "model_student_t":trace5,
    "model_skewt":trace6,
    "skew_mixture_model":trace7
})
comparaison
```

```
/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/stats/stats.py:782: UserWarning:

Estimated shape parameter of Pareto distribution is greater than 0.68 for one or
more samples. You should consider using a more robust model, this is because
```

```
importance sampling is less likely to work well if the marginal posterior and
LOO posterior are very different. This is more likely to happen with a non-
robust model and highly influential observations.

/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/stats/stats.py:782: UserWarning:

Estimated shape parameter of Pareto distribution is greater than 0.70 for one or
more samples. You should consider using a more robust model, this is because
importance sampling is less likely to work well if the marginal posterior and
LOO posterior are very different. This is more likely to happen with a non-
robust model and highly influential observations.

/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/stats/stats.py:782: UserWarning:

Estimated shape parameter of Pareto distribution is greater than 0.70 for one or
more samples. You should consider using a more robust model, this is because
importance sampling is less likely to work well if the marginal posterior and
LOO posterior are very different. This is more likely to happen with a non-
robust model and highly influential observations.
```

[56]:

	rank	elpd_loo	p_loo	elpd_diff	\
model_skewt	0	-3048.904539	12.147048	0.000000	
skew_mixture_model	1	-3059.098393	38.852016	10.193854	
skewnormal_linear_model	2	-3120.046230	12.015493	71.141691	
model_student_t	3	-3267.861231	12.249334	218.956692	
model_linear_bimodal	4	-3429.949116	12.825652	381.044577	
normal_linear_model	5	-3449.688006	11.274282	400.783467	
poisson_linear_model	6	-5119.862399	2.094433	2070.957860	

	weight	se	dse	warning	scale
model_skewt	9.177956e-01	55.799643	0.000000	False	log
skew_mixture_model	0.000000e+00	54.932938	2.788837	False	log
skewnormal_linear_model	0.000000e+00	54.489721	14.455584	True	log
model_student_t	0.000000e+00	59.103108	19.485006	False	log
model_linear_bimodal	0.000000e+00	61.563555	32.143730	True	log
normal_linear_model	8.220442e-02	61.659541	35.812305	False	log
poisson_linear_model	8.524811e-13	16.405031	44.884490	True	log

Les messages d'avertissement , indiquant que le paramètre de forme de la distribution Pareto est supérieur à 0.67 pour un ou plusieurs échantillons.

Cela indique qu'un modèle non robuste est plus susceptible de ne pas bien généraliser, et il pourrait être nécessaire d'ajuster ou de reformuler le modèle.

Les modeles presentant cee warning ne sont donc pas les plus adaptés dans ce cas. il s'agit de skewnormal_linear_model, model_linear_bimodal et poisson_linear_model

Meilleur modèle selon elpd_loo : Le modèle model_skewt (avec un score elpd_loo de -3049.14)

La probabilité pondérée que ce modèle soit le meilleur (weight) est de 91.79%. Il n'y a pas d'avertissement (False), donc le modèle ne présente pas de problèmes de robustesse ou d'instabilité importants.

4 III- Resulat du meilleur modele - Regression Skew student

Dans cette section, nous effectuons des **prédictions postérieures** à partir du modèle `skewt` que nous avons ajusté à l'aide de PyMC.

L'objectif est de générer des prédictions basées sur les distributions postérieures des paramètres du modèle.

Nous commençons par échantillonner les valeurs postérieures prédites pour la variable `y_obs` à partir de la chaîne de Markov générée par notre modèle

4.1 A- Prédictions Postérieures avec le Modèle Skew-t

[57]:
```
with model_skewt:
  ppc=pm.sample_posterior_predictive(trace6,var_names=["y_obs"])
```

```
Sampling: [y_obs]
```

```
Output()
```

Une fois l'échantillonnage effectué, nous extrayons les valeurs des prédictions.

Les prédictions postérieures ont été générées sur des données standardisées, donc nous appliquons une inverse de la standardisation pour revenir aux échelles originales des données. Nous utilisons ici un objet scaler_y qui a été précédemment créé pour la standardisation de median_house_value.

Cela nous permet d'obtenir les prédictions dans les mêmes unités que les données originales.

Une fois les prédictions dé-standardisées, nous reformons la structure des données pour la rendre compatible avec le format d'origine (chains, draws, obs) et Nous mettons à jour les valeurs observées dans l'objet ppc avec les valeurs dé-standardisées :

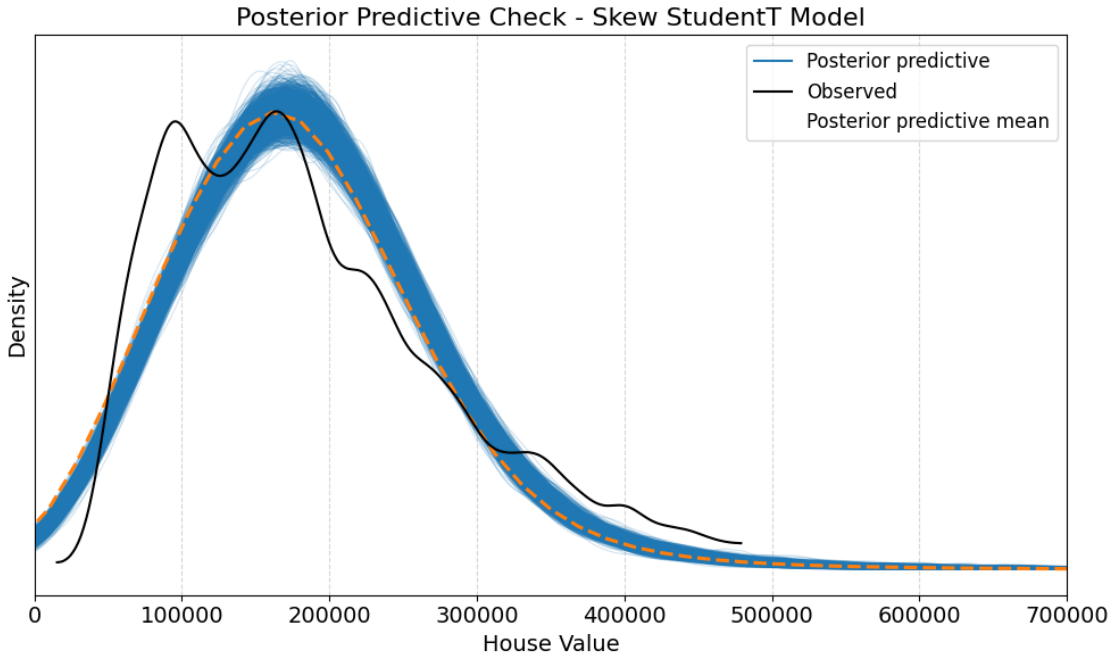
[58]:
```
# Récupération des valeurs postérieures prédites
y_ppc = ppc.posterior_predictive['y_obs'].values
# Flatten chaînes et tirages en une seule dimension
y_ppc_flat = y_ppc.reshape(-1, y_ppc.shape[-1])
# Inverse la standardisation
scaler_y = scalers["median_house_value"]
y_ppc_unscaled = scaler_y.inverse_transform(y_ppc_flat)
y_ppc_unscaled = y_ppc_unscaled.reshape(y_ppc.shape)


ppc.posterior_predictive['y_obs'].values = y_ppc_unscaled
```

De manière similaire aux étapes précédentes, nous récupérons également les valeurs observées (y_obs) depuis ppc et les transformons en les dé-standardisant pour avoir les prédictions sur l'échelle originale.

[59]:
```
# Récupération des valeurs postérieures prédites
x_ppc = ppc.observed_data['y_obs'].values  # shape: (chains, draws, n_obs)

# Flatten chaînes et tirages en une seule dimension
x_ppc_flat = x_ppc.reshape(-1, y_ppc.shape[-1])  # shape: (samples, n_obs)


x_ppc_unscaled = scaler_y.inverse_transform(x_ppc_flat)  # shape: (samples,␣
 ↪n_obs)

# Reforme dans le format original (chains, draws, obs)
x_ppc_unscaled = x_ppc_unscaled.reshape(x_ppc.shape)


ppc.observed_data['y_obs'].values= x_ppc_unscaled
```

4.2 B- Visualisation du Posterior Predictive Check (PPC)

[60]:
```
fig, ax = plt.subplots(figsize=(10, 6))

az.plot_ppc(ppc, ax=ax, kind='kde', data_pairs={"y_obs": "y_obs"},
            mean=True)

# Customisation
ax.set_title("Posterior Predictive Check - Skew StudentT Model", fontsize=16)
ax.set_xlabel("House Value", fontsize=14)
ax.set_ylabel("Density", fontsize=14)
ax.legend(["Posterior predictive", "Observed", "Posterior predictive mean"],␣
 ↪fontsize=12)
ax.grid(True, linestyle="--", alpha=0.5)
ax.set_xlim(0, 700000)
plt.tight_layout()
plt.show()
```



Ce graphique de vérification prédictive a posteriori suggère que le modèle Skew StudentT utilisé est capable de générer des données qui présentent des caractéristiques similaires aux données observées.

Il semble y avoir un certain degré de chevauchement entre la distribution observée (noire) et les distributions prédictives a posteriori (bleues). La distribution observée se situe globalement dans la plage des prédictions du modèle, ce qui est un signe positif.

4.3 C- Evaluation des performance

[61]:
```
print(pd.Series(house_value).describe())
```

```
count      3486.000000
mean     185848.909639
std       94609.486077
min       14999.000000
25%      110975.000000
50%      168800.000000
75%      240900.000000
max      479500.000000
dtype: float64
```

[62]:
```
# Accédez à la clé correcte dans ppc
predicted = ppc.posterior_predictive["y_obs"]

# Calcul de statistiques descriptives sur les prédictions
predicted_mean = np.mean(predicted )
predicted_std = np.std(predicted )
```

```
hdi = az.hdi(predicted.values.flatten(), hdi_prob=0.94).round(2)


print(f"Moyenne prédictive : {predicted_mean:.2f}")
print(f"Écart-type prédictif : {predicted_std:.2f}")
print(f"Intervalle de crédibilité à 94 % : {hdi}")
```

```
Moyenne prédictive : 186530.18
Écart-type prédictif : 94252.46
Intervalle de crédibilité à 94 % : [ 24315.18 348943.11]
```

La moyenne des données observées (185848.91) est très proche de la moyenne prédictive du modèle (186581.96).

De même l'écart-type des données observées (94609.49) est également très similaire à l'écart-type prédictif (94230.15).

Cela indique que le modèle capture bien la tendance centrale de la variable house_value et que la dispersion des prédictions du modèle autour de sa moyenne est comparable à la dispersion observée.

Cependant,

Le minimum observé (14999.00) est inférieur à la borne inférieure de l'intervalle de crédibilité (22874.93). Et le maximum observé (479500.00) est supérieur à la borne supérieure de l'intervalle de crédibilité (347759.91).

Cela suggére que le modèle a une probabilité plus faible de prédire les valeurs les plus basses que observées ou des valeurs les plus élevées que observées.

[63]:
```
from sklearn.metrics import mean_squared_error
y_pred_mean = np.mean(predicted, axis=0)
y_pred_mean = y_pred_mean.T
y_pred_mean = np.mean(y_pred_mean, axis=1)
rmse = np.sqrt(mean_squared_error(house_value, y_pred_mean))

print(f"RMSE : {rmse:.3f}")
```

```
RMSE : 62703.826
```

le RMSE est inférieur à l'écart-type des données observées ce qui est positif. Cela suggère que les erreurs de prédiction du modèle sont, en moyenne, plus petites que la dispersion naturelle des valeurs de la variable cible.

En d'autres termes, **le modèle explique une partie de la variance des données.**

Cette information se précise avec le calcul du R_carré.

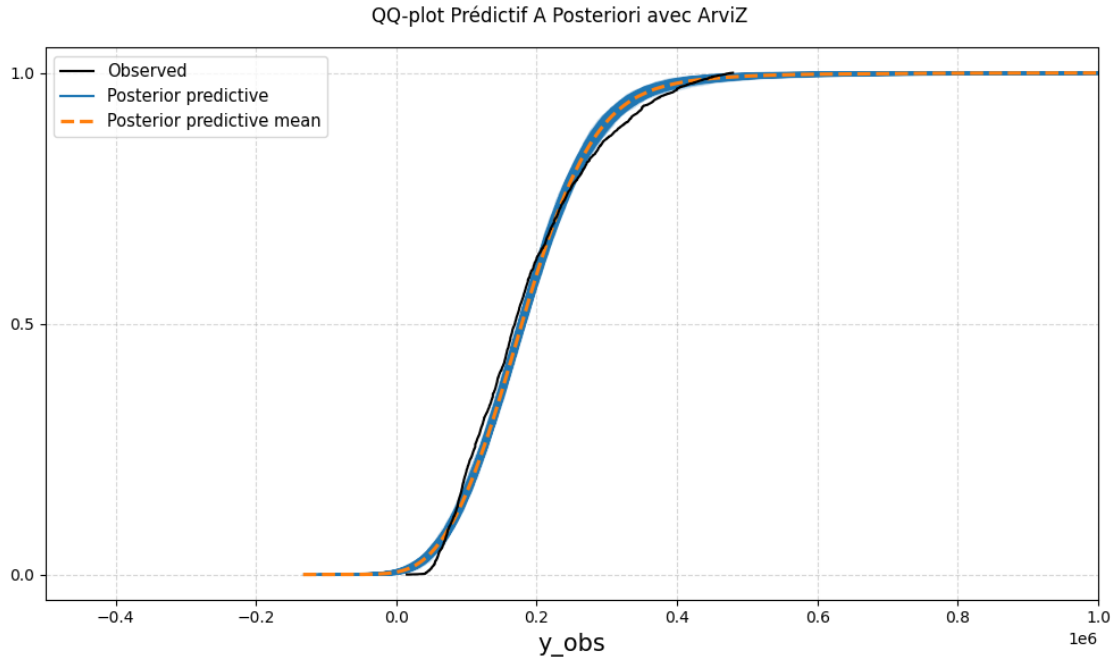
[64]:
```
# Calculer le R² bayésien global
r2,r2_std = az.r2_score(house_value, az.extract(ppc,␣
 ↪group="posterior_predictive", var_names="y_obs").values.T)
print(f"Bayesian R² global: {r2:.3f}")
print(f"Bayesian R² écart-type: {r2_std:.3f}")
```

```
Bayesian R² global: 0.498
Bayesian R² écart-type: 0.007
```

Une valeur de 0.498 du Bayesian R^2 signifie que le modèle explique environ 49.8% de la variance observée dans les prix des maisons.

De plus l'estimation de cette variance expliquée est relativement précise (faible écart-type du R^2).

[65]:
```
fig, ax = plt.subplots(figsize=(10, 6))

# Plot PPC
az.plot_ppc(ppc, kind="cumulative", ax=ax)

# Customisation
plt.suptitle("QQ-plot Prédictif A Posteriori avec ArviZ")
ax.grid(True, linestyle="--", alpha=0.5)
ax.set_xlim(-500000, 1000000)
plt.tight_layout()
plt.show()
```



En observant ce QQ-plot prédictif a posteriori, on peut affirmer que :

1. La ligne noire (quantiles observés) suit globalement la tendance des lignes bleu-vert (quantiles prédictifs) et de la ligne orange (quantiles moyens). Cela suggère que **le modèle est capable de capturer la distribution générale des données observées**.
2. La ligne orange en pointillés suit de près la ligne noire, ce qui indique que la prédiction moyenne des quantiles du modèle est assez proche des quantiles observés. on note donc une

Adéquation de la moyenne prédictive

3. La dispersion des lignes bleu-vert donne une indicant l'incertitude, ici, semble relativement faible: **le modèle est assez confiant** dans ses prédictions de la forme de la distribution.

4.4 D- Analyse des coefficients

Nous allons à prêsent analyser les coefficients obtenus par les différents modèles afin de comprendre leur influence sur la variable cible (le prix médian des maisons).

L'analyse des coefficients est cruciale dans les modèles linéaires, car **elle permet de mesurer l'impact de chaque variable explicative sur la prédiction du modèle.**

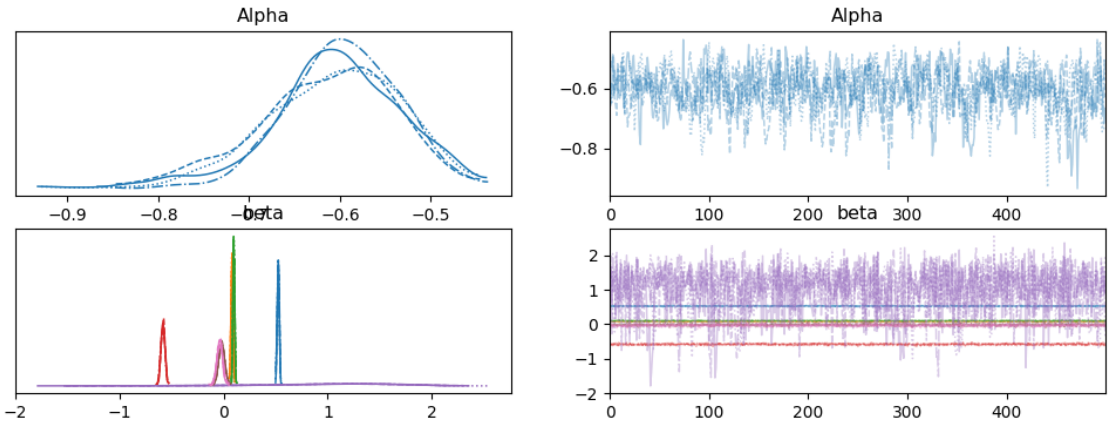
En utilisant nos modèles bayésiens, nous avons estimé ces coefficients sous forme de distributions. Cela nous permet de non seulement obtenir une **valeur estimée** pour chaque coefficient, mais aussi leur **significativité** grâce aux intervalles de crédibilité postérieurs.

Nous regardons ceux du meilleur modèle

Pour rappel, Un coefficient positif signifie qu'une augmentation de la variable associée entraîne une augmentation du prix des maisons, et un coefficient négatif signifie qu'une augmentation de cette variable entraîne une diminution du prix des maisons.

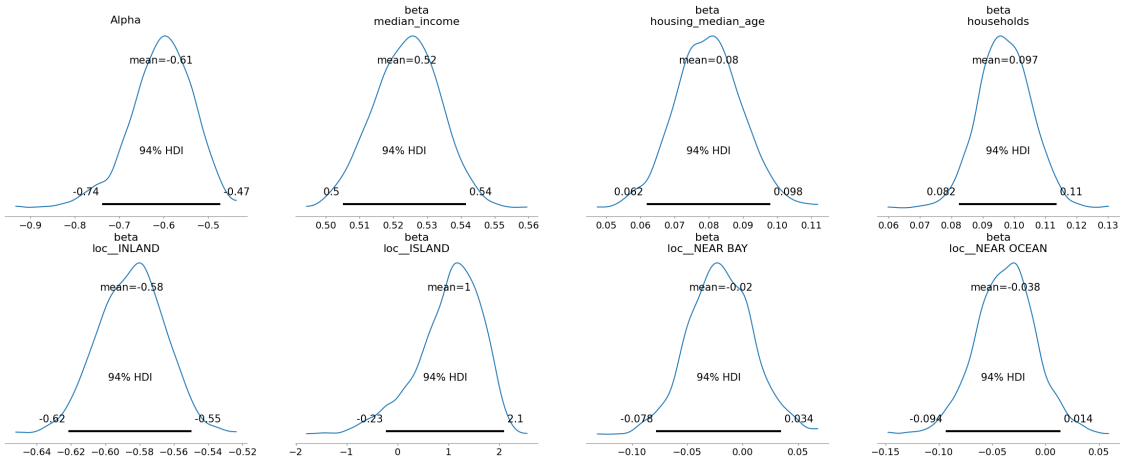
[66]:
```
az.plot_trace(trace6, var_names=['Alpha', 'beta'])
```

[66]:
```
array([[<Axes: title={'center': 'Alpha'}>,
        <Axes: title={'center': 'Alpha'}>],
       [<Axes: title={'center': 'beta'}>,
        <Axes: title={'center': 'beta'}>]], dtype=object)
```



Nous rappelons, le modèle a convergé correctement. Cette information sera comfirmet par le r_hat dans le summary.

[67]:
```
az.plot_posterior(trace6, var_names=['Alpha', 'beta'])
plt.show()
```



On remarque :

- La plupart des distributions a posteriori sont unimodales, ce qui facilite l'interprétation.
- L'intervalle de crédibilité plus large pour `beta_oc__ISLAND` indique une plus grande incertitude quant à l'effet de cette variable. Le fait qu'il contienne zéro suggère que l'effet de "ISLAND" pourrait ne pas être significativement différent de zéro.
- En regardant les signes des moyennes a posteriori, on peut interpréter la direction de l'effet de chaque variable sur la variable cible.

[68]:
```
az.summary(trace6,var_names=['beta'], round_to=2)
```

[68]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
beta[median_income]	0.52	0.01	0.50	0.54	0.00	0.00	1535.48	1256.17	1.01
beta[housing_median_age]	0.08	0.01	0.06	0.10	0.00	0.00	1427.65	1472.66	1.00
beta[households]	0.10	0.01	0.08	0.11	0.00	0.00	1797.68	1487.19	1.00
beta[loc__INLAND]	-0.58	0.02	-0.62	-0.55	0.00	0.00	1404.49	1303.28	1.00
beta[loc__ISLAND]	1.01	0.65	-0.23	2.09	0.02	0.02	1745.60	929.55	1.00
beta[loc__NEAR BAY]	-0.02	0.03	-0.08	0.03	0.00	0.00	2223.86	1507.97	1.00
beta[loc__NEAR OCEAN]	-0.04	0.03	-0.09	0.01	0.00	0.00	1557.02	1343.55	1.00

- **beta[median_income]** : 0.52 (avec un intervalle de crédibilité entre 0.49 et 0.54).
  -> le revenu médian des ménages est fortement lié à la valeur médiane des maisons. L'intervalle de crédibilité est étroit.

- **beta[housing_median_age]** : 0.06 (avec un intervalle de crédibilité entre 0.04 et 0.09).
  -> l'âge moyen des maisons a un effet positif sur la valeur des maisons.
- **beta[households]** : 0.10 (avec un intervalle de crédibilité entre 0.08 et 0.13).
  -> un nombre plus élevé de ménages est associé à des valeurs immobilières plus élevées.
- **beta[loc___INLAND]** : -0.57 (avec un intervalle de crédibilité entre -0.62 et -0.52).
  -> les zones intérieures (non côtières) ont des prix des maisons significativement plus bas par rapport à d'autres régions côtières.
- **beta[loc___ISLAND]** : 1.07 (avec un intervalle de crédibilité entre -0.04 et 2.12).
  -> les îles peuvent avoir une prime cependant car l'intervalle de crédibilité inclut 0 on peut dire que le coefficient n'est pas significativement different de 0.
- **beta[loc___NEAR BAY]** : 0.02 (avec un intervalle de crédibilité entre -0.06 et 0.10).
  -> la proximité de la baie a un effet positif relativement faible sur le prix des maisons, avec une grande incertitude.
- **beta[loc___NEAR OCEAN]** : -0.07 (avec un intervalle de crédibilité entre -0.15 et -0.00).
  -> la proximité de l'océan pourrait légèrement diminuer les prix des maisons mais encore une fois la credibilité du coefficient est faible car l'intervalle inclut 0.

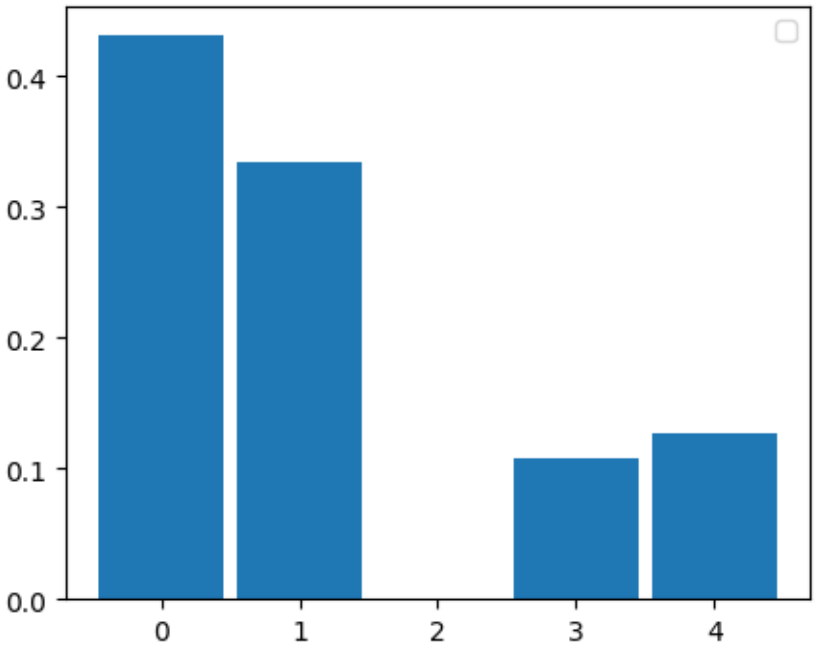
5 IV- Analyse Bayésienne

5.1 A- Ocean proximity

ocean_proximity est une variable catégorielle (genre '<1H OCEAN', 'INLAND', 'ISLAND', etc.), donc on parle ici d'un modèle de classification probabiliste.

[69]:
```
Ocean_pro= Og_Housing['ocean_proximity'].values
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
Ocean_pro = encoder.fit_transform(Ocean_pro)


plt.figure(figsize=(5, 4))
az.plot_dist(Ocean_pro, label="")
plt.show()

# Print the mapping:
for i, category in enumerate(encoder.classes_):
    print(f"Numeric Value {i}: Category '{category}'")
```

```
/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/plots/backends/matplotlib/distplot.py:176: UserWarning:

No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```




```
Numeric Value 0: Category '<1H OCEAN'
Numeric Value 1: Category 'INLAND'
Numeric Value 2: Category 'ISLAND'
Numeric Value 3: Category 'NEAR BAY'
Numeric Value 4: Category 'NEAR OCEAN'
```

[70]:
```
ocean_proximity_counts = Og_Housing['ocean_proximity'].
  ↪value_counts(normalize=True) * 100
ocean_proximity_counts
```

[70]:
```
ocean_proximity
<1H OCEAN     43.151314
INLAND        33.365837
NEAR OCEAN    12.641964
NEAR BAY      10.812206
ISLAND         0.028680
Name: proportion, dtype: float64
```

5.1.1 Approche avec un modèle de multinoulli

On veut modéliser **la distribution marginale** de ocean_proximity donc pas conditionnée aux autres variables avec un modèle **de multinoulli** avec un **Dirichlet comme prior** sur les proportions.

[71]:
```
labels, ocean_codes = pd.factorize(Og_Housing['ocean_proximity'])
```

```
with pm.Model() as ocean_dist_model:
    # Prior de Dirichlet sur les proportions de chaque catégorie
    probs = pm.Dirichlet("probs", a=np.ones(len(ocean_codes)))

    # Likelihood : Multinomial pour les catégories
    obs = pm.Categorical("obs", p=probs, observed=labels)

    trace_ocean1 = pm.sample(1000, tune=500, target_accept=0.95, idata_kwargs=␣
 ↪{'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [probs]
```

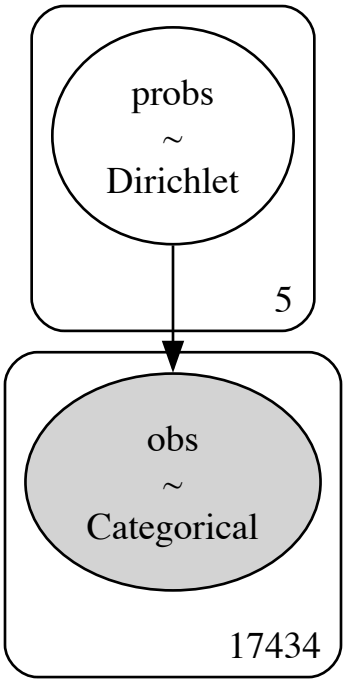
```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 6 seconds.
The effective sample size per chain is smaller than 100 for some parameters.  A
higher number is needed for reliable rhat and ess computation. See
https://arxiv.org/abs/1903.08008 for details
```

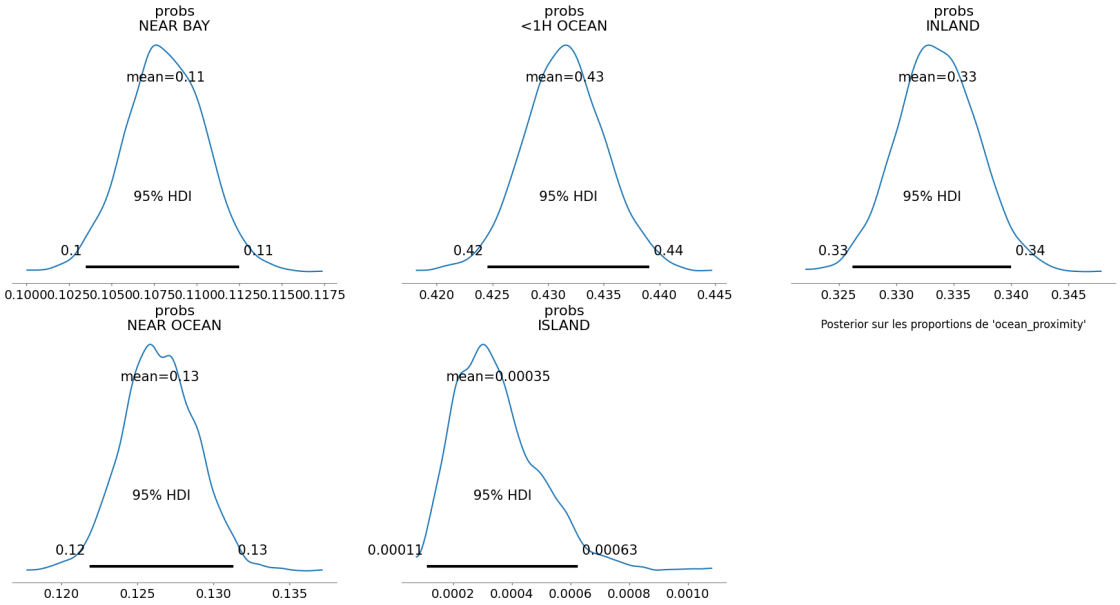
[72]:
```
trace_ocean1.posterior = trace_ocean1.posterior.
 ↪assign_coords(probs_dim_0=ocean_codes)
```

[73]:
```
pm.model_to_graphviz(ocean_dist_model)
```

[73]:

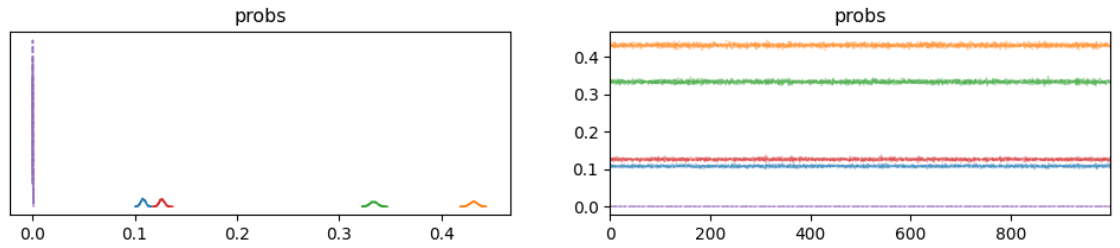


[74]:
```
az.plot_posterior(trace_ocean1, var_names=["probs"], hdi_prob=0.95)
plt.xticks(ticks=np.arange(len(ocean_codes)), labels=ocean_codes, rotation=45)
plt.title("Posterior sur les proportions de 'ocean_proximity'")
plt.show()
```



[75]:
```
az.plot_trace(trace_ocean1, var_names=["probs"])
```

[75]:
```
array([[<Axes: title={'center': 'probs'}>,
        <Axes: title={'center': 'probs'}>]], dtype=object)
```



on constate bien une convergence. Les différentes chaînes semblent osciller autour de niveaux relativement constants et montrent un certain degré de mélange.

Les chaînes semblent s'approcher de la stationnarité : les fluctuations autour de leurs moyennes respectives sont plus ou moins aléatoires et bornées, sans tendance claire à la hausse ou à la baisse

sur le long terme.

[76]:
```
az.summary(trace_ocean1, round_to=2)
```

[76]:
```
                    mean   sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
probs[NEAR BAY]     0.11  0.0    0.10     0.11        0.0      0.0   4170.19
probs[<1H OCEAN]    0.43  0.0    0.42     0.44        0.0      0.0   4970.64
probs[INLAND]       0.33  0.0    0.33     0.34        0.0      0.0   5118.58
probs[NEAR OCEAN]   0.13  0.0    0.12     0.13        0.0      0.0   4270.81
probs[ISLAND]       0.00  0.0    0.00     0.00        0.0      0.0    338.10

                    ess_tail  r_hat
probs[NEAR BAY]      2769.43   1.00
probs[<1H OCEAN]     3103.13   1.00
probs[INLAND]        3257.77   1.00
probs[NEAR OCEAN]    2951.40   1.00
probs[ISLAND]         437.75   1.01
```

[77]:
```
ocean_proximity_counts
```

[77]:
```
ocean_proximity
<1H OCEAN     43.151314
INLAND        33.365837
NEAR OCEAN    12.641964
NEAR BAY      10.812206
ISLAND         0.028680
Name: proportion, dtype: float64
```

les R_hat confirme le constat visuel.

Les résultats montrent des valeurs de r-hat proches de 1 pour toutes les variables, ce qui indique une bonne convergence des chaînes de Markov. De plus les efficacité d'échantillonnage (ess_bulk et ess_tail) sont élevées, ce qui suggère un bon échantillonnage et une estimation fiable des paramètres.

Le modèle estime que la catégorie <1H OCEAN a la probabilité la plus élevée (43%), suivie de INLAND (33%), NEAR OCEAN (13%), et NEAR BAY (11%). La catégorie ISLAND a une probabilité pratiquement nulle.

Les proportions observées dans les données réelles confirment que la catégorie <1H OCEAN est prédominante, suivie par INLAND, avec des proportions de NEAR OCEAN et NEAR BAY plus faibles. La catégorie ISLAND représente une très petite proportion des observations.

[78]:
```
with ocean_dist_model:
    ppc1 = pm.sample_posterior_predictive(trace_ocean1)
```

```
Sampling: [obs]
```

```
Output()
```

5.1.2 Approche avec un modèle Multinomial

(ou une Dirichlet-Multinomial)

L'objectif principal est de caractériser la distribution catégorielle (ocean_proximity), de manière probabiliste. Au lieu de juste compter les fréquences comme en stats classiques. Il s'agit d'un modèle naturel pour des données catégorielles agrégée : nous avons **n observations** réparties en **k classes** et la Multinomial modélise exactement cela.

[79]:
```
one_hot_ocean = pd.get_dummies(Og_Housing["ocean_proximity"], drop_first=False)␣
↪# keep it as a DataFrame
n_classes = one_hot_ocean.shape[1]
counts = one_hot_ocean.sum(axis=0).values  # access the values of the sum

with pm.Model() as multinoulli_model:
    probs = pm.Dirichlet("probs", a=np.ones(n_classes))  # prior non-informatif
    obs = pm.Multinomial("obs", n=counts.sum(), p=probs, observed=counts)
    trace_ocean2 = pm.sample(1000, tune=500, target_accept=0.95, idata_kwargs=␣
↪{'log_likelihood': True})
```

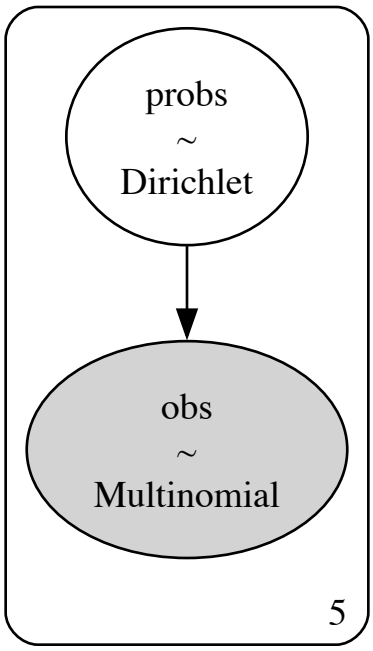
```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [probs]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 1 seconds.
```

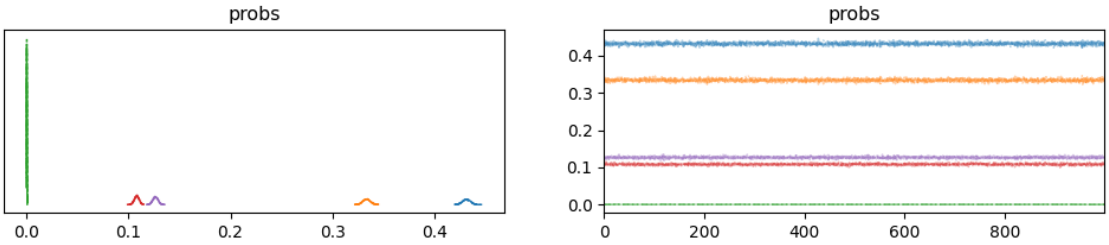
[80]:
```
pm.model_to_graphviz(multinoulli_model)
```

[80]:



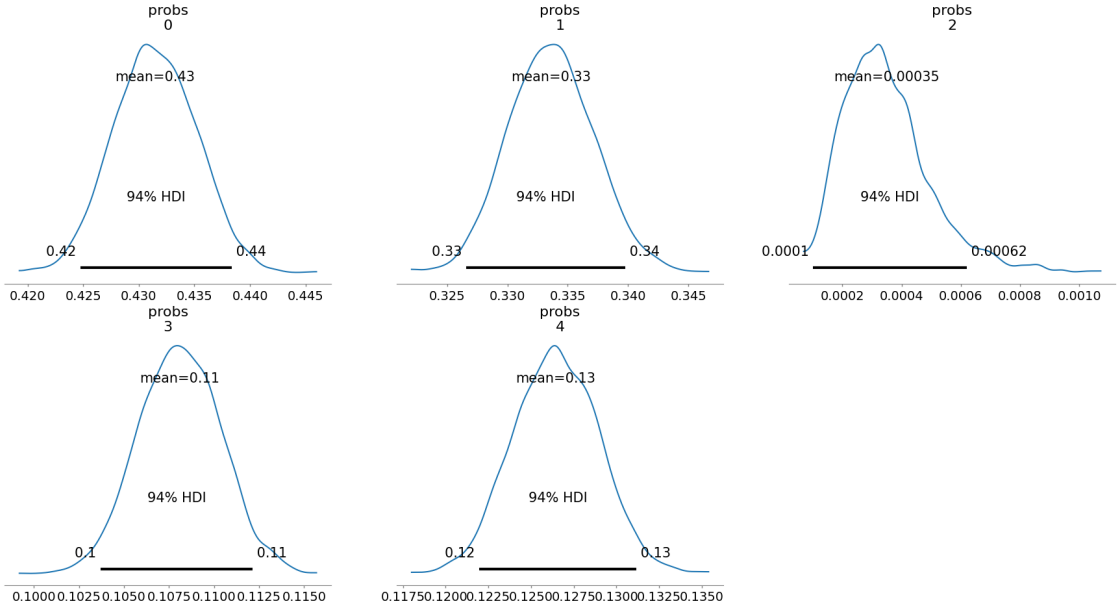
[81]:
```
az.plot_trace(trace_ocean2, var_names=["probs"])
```

[81]:
```
array([[<Axes: title={'center': 'probs'}>,
        <Axes: title={'center': 'probs'}>]], dtype=object)
```



[82]:
```
az.plot_posterior(trace_ocean2, var_names=["probs"])
```

[82]:
```
array([[<Axes: title={'center': 'probs\n0'}>,
        <Axes: title={'center': 'probs\n1'}>,
        <Axes: title={'center': 'probs\n2'}>],
       [<Axes: title={'center': 'probs\n3'}>,
        <Axes: title={'center': 'probs\n4'}>, <Axes: >]], dtype=object)
```



[83]:
```
observed_props = counts / counts.sum()
posterior_means = trace_ocean2.posterior['probs'].mean(dim=("chain", "draw")).
 ↪values

for cat, obs, post in zip(one_hot_ocean.columns, observed_props,␣
 ↪posterior_means):
    print(f"{cat}: Observed={obs:.3f}, Posterior Mean={post:.3f}")
```

```
<1H OCEAN: Observed=0.432, Posterior Mean=0.431
INLAND: Observed=0.334, Posterior Mean=0.334
ISLAND: Observed=0.000, Posterior Mean=0.000
NEAR BAY: Observed=0.108, Posterior Mean=0.108
NEAR OCEAN: Observed=0.126, Posterior Mean=0.126
```

5.1.3 Conclusion

Les deux modèles, Multinoulli et Multinomial, ont permis de modéliser efficacement la distribution de la variable catégorielle ocean_proximity dans notre jeu de données.

Le modèle Multinoulli, en utilisant la distribution multinomiale, a montré que les classes de ocean_proximity peuvent être modélisées en fonction des priors Dirichlet, ce qui nous a donné un aperçu de la probabilité d'appartenance de chaque observation à une classe donnée.

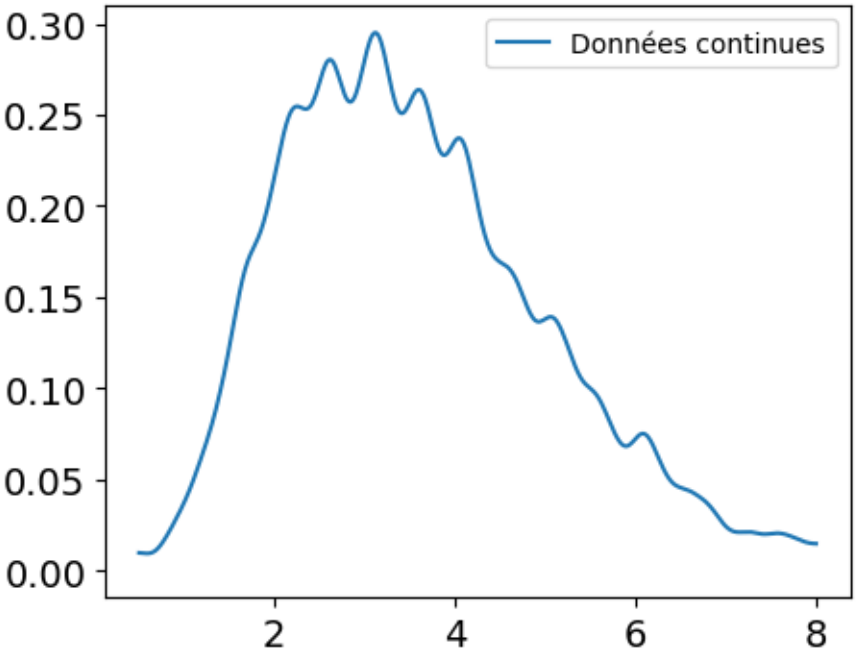
D'un autre côté, le modèle Multinomial a permis de capturer de manière plus nuancée les relations entre les variables explicatives et les catégories de ocean_proximity, en offrant des prévisions plus robustes pour la classification des données.

5.2 B- Median Income

On a affaire à une distribution continue qui présente de l'asymetrie de manière flagrante.

[84]:
```
Income= Og_Housing['median_income'].values
plt.figure(figsize=(5, 4))
az.plot_dist(Income, label="Données continues")
```

[84]: <Axes: >



5.2.1 Création des modèles

1. Modèle Skew Normal

Le modèle Skew Normal permet de modéliser des distributions avec une asymétrie, c'est-à-dire des distributions où la courbe de probabilité n'est pas symétrique.

2. Modèle Student-t

Le modèle Student-t est souvent utilisé lorsque les données présentent des queues plus lourdes que celles d'une distribution normale. Cela permet de mieux capturer les extrêmes.

3. Modèle Skew Student-t

Le modèle Skew Student-t combine les propriétés de la distribution Student-t et d'une distribution asymétrique. Il est utile lorsque les données ont à la fois des queues lourdes et une asymétrie.

[85]:
```
with pm.Model() as skew_normal_model:
    # Paramètres pour la distribution Skew Normal
    mu = pm.Normal("mu", mu=0, sigma=5)  # Moyenne de la distribution
    sigma = pm.HalfNormal("sigma", sigma=5)  # Écart-type
    a = pm.Normal("a", mu=0, sigma=2)  # Paramètre d'asymétrie

    # Distribution Skew Normal pour median_income
    income_dist = pm.SkewNormal("income", mu=mu, sigma=sigma, alpha=a,␣
 ↪observed=Income)

    # Échantillonnage
    trace_skew_normal = pm.sample(1000, tune=500, target_accept=0.95,␣
 ↪idata_kwargs= {'log_likelihood': True})
```

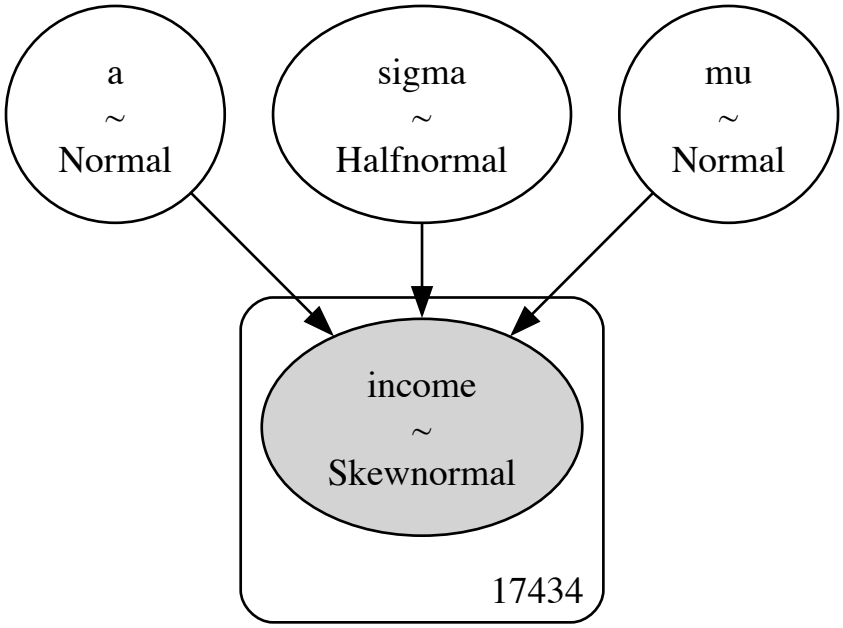
```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, a]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 12 seconds.
```

[86]:
```
pm.model_to_graphviz(skew_normal_model)
```

[86]:



[87]:
```
with pm.Model() as student_t_model:
    # Paramètres pour la distribution Student-t
    mu = pm.Normal("mu", mu=0, sigma=10)  # Moyenne de la distribution
    sigma = pm.HalfNormal("sigma", sigma=1)  # Écart-type
    nu = pm.Exponential("nu", 1)  # Degrés de liberté

    # Distribution Student-t pour median_income
    income_dist = pm.StudentT("income", mu=mu, sigma=sigma, nu=nu,␣
 ↪observed=Income)

    # Échantillonnage
    trace_student_t = pm.sample(1000, tune=500, target_accept=0.95,␣
 ↪idata_kwargs= {'log_likelihood': True})
```

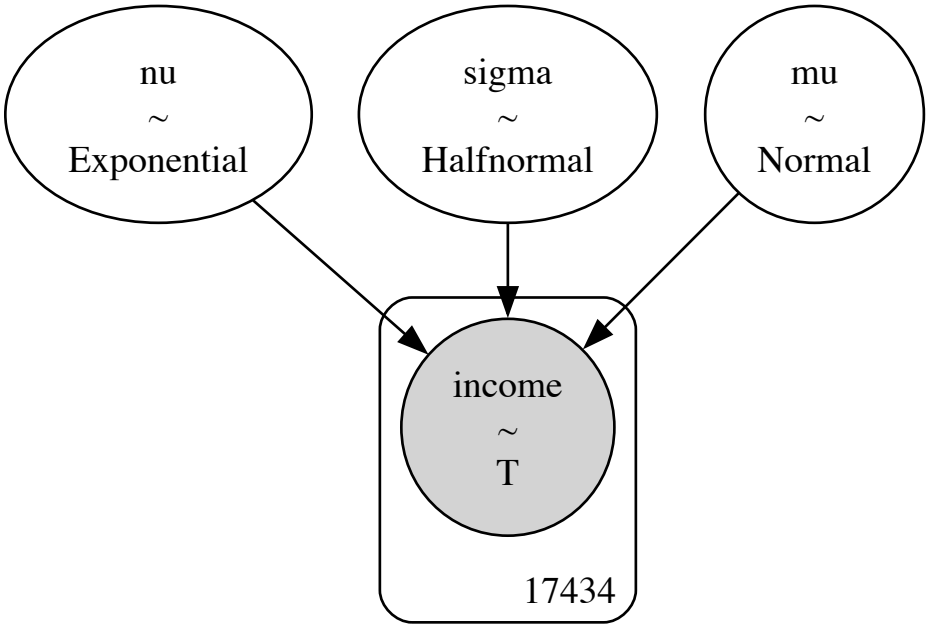
```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, nu]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 3 seconds.
```

[88]:
```
pm.model_to_graphviz(student_t_model)
```

[88]:



[89]:
```
with pm.Model() as skew_student_t_model:
    # Paramètres pour la distribution Skew Student-t
    mu = pm.Normal("mu", mu=0, sigma=10)  # Moyenne de la distribution
    sigma = pm.HalfNormal("sigma", sigma=1)  # Écart-type
    a = pm.HalfNormal("a", sigma=10)  # Paramètre d'asymétrie
    b = pm.HalfNormal("b", sigma=10)  # Paramètre d'asymétrie

    # Distribution Skew Student-t pour median_income
    income_dist = pm.SkewStudentT("income", mu=mu, sigma=sigma, a=a, b=b,␣
↪observed=Income)

    # Échantillonnage
    trace_skew_student_t = pm.sample(1000, tune=500, target_accept=0.95,␣
↪idata_kwargs= {'log_likelihood': True})
```

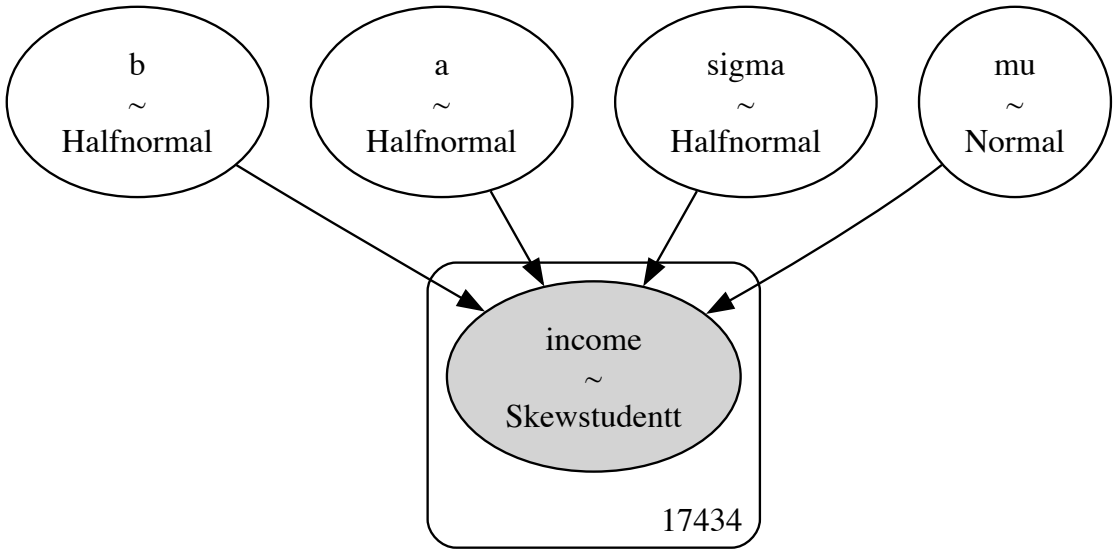
```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, a, b]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 73 seconds.
```

[90]:
```
pm.model_to_graphviz(skew_student_t_model)
```

[90]:



5.2.2 Comparaison

[91]:
```
comp=pm.compare({
    "model_skew_normal":trace_skew_normal,
    "model_student_t":trace_student_t,
    "model_skewt":trace_skew_student_t
})
comp
```

[91]:

	rank	elpd_loo	p_loo	elpd_diff	weight	\
model_skew_normal	0	-30409.081887	2.815129	0.000000	1.000000e+00	
model_skewt	1	-30537.067822	2.398490	127.985935	2.179832e-12	
model_student_t	2	-31162.532764	2.052535	753.450877	0.000000e+00	

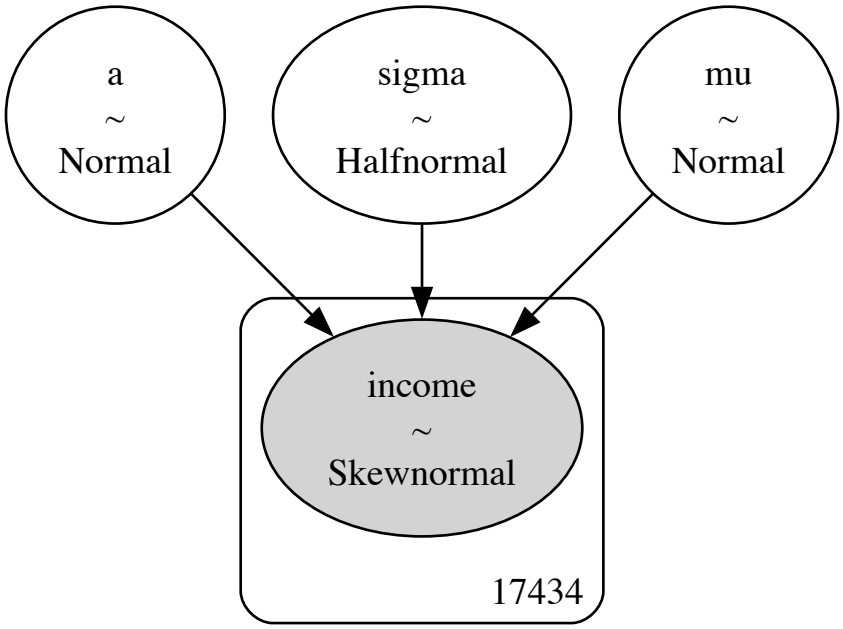
	se	dse	warning	scale
model_skew_normal	86.955988	0.000000	False	log
model_skewt	89.290857	10.105023	False	log
model_student_t	91.839846	33.864256	False	log

model__skew__normal est le meilleur modèle parmi ceux testés, avec le plus bas elpd_loo (meilleur ajustement) et une évaluation robuste des performances.

5.2.3 Résultat du meilleur modele - model__skew__normal

[92]:
```
pm.model_to_graphviz(skew_normal_model)
```

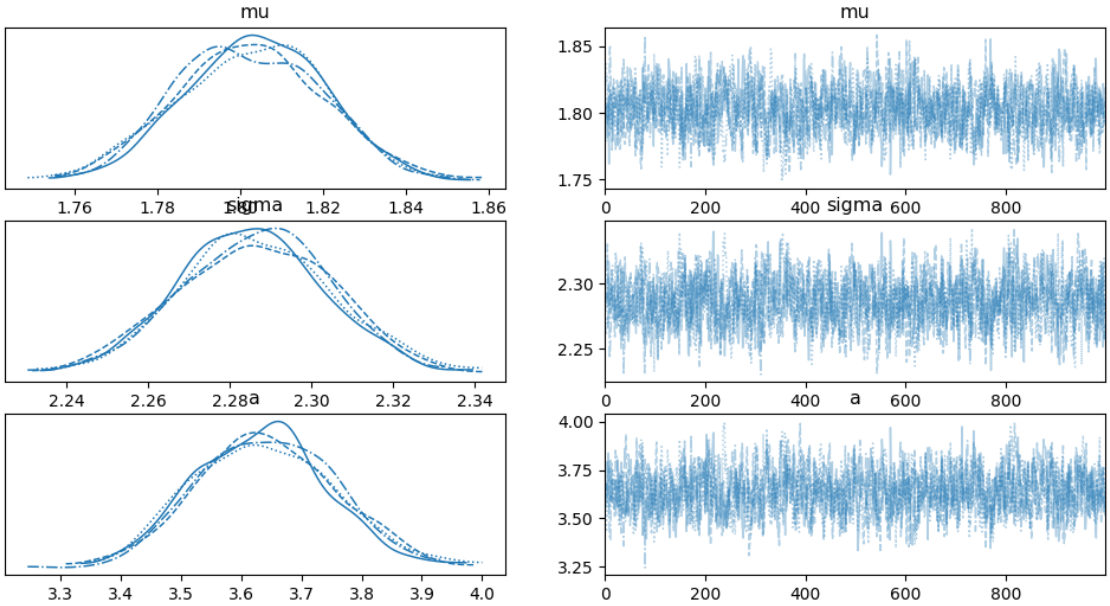
[92]:



[93]:
```
az.plot_trace(trace_skew_normal, var_names=["mu", "sigma", "a"])
az.summary(trace_skew_normal, round_to=2)
```

[93]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
mu	1.80	0.02	1.77	1.83	0.0	0.0	956.02	1537.58	1.0
a	3.64	0.11	3.42	3.84	0.0	0.0	978.34	1384.74	1.0
sigma	2.29	0.02	2.25	2.32	0.0	0.0	1072.05	1334.06	1.0



Les statistiques de convergence $\hat{R}$ (R-hat) pour les trois paramètres sont égales à ~1, ce qui indique une excellente convergence des chaînes MCMC. Les différentes chaînes sont parvenues à une stationnarité.

[94]:
```
with skew_normal_model:
    ppc = pm.sample_posterior_predictive(trace_skew_normal,␣
 ↪var_names=["income"])
```

```
Sampling: [income]
```

```
Output()
```

[95]:
```
pd.Series(Income).describe()
```

[95]:
```
count    17434.000000
mean         3.575593
std          1.443540
min          0.499900
25%          2.490150
50%          3.390600
75%          4.487950
max          8.011300
dtype: float64
```

[96]:
```
predicted = ppc.posterior_predictive["income"]

# Calcul de statistiques descriptives sur les prédictions
predicted_mean = np.mean(predicted)
predicted_std = np.std(predicted)
hdi = az.hdi(predicted.values.flatten(), hdi_prob=0.94).round(2)



print(f"Moyenne prédictive : {predicted_mean:.2f}")
print(f"Écart-type prédictif : {predicted_std:.2f}")
print(f"Intervalle de crédibilité à 94 % : {hdi}")
```

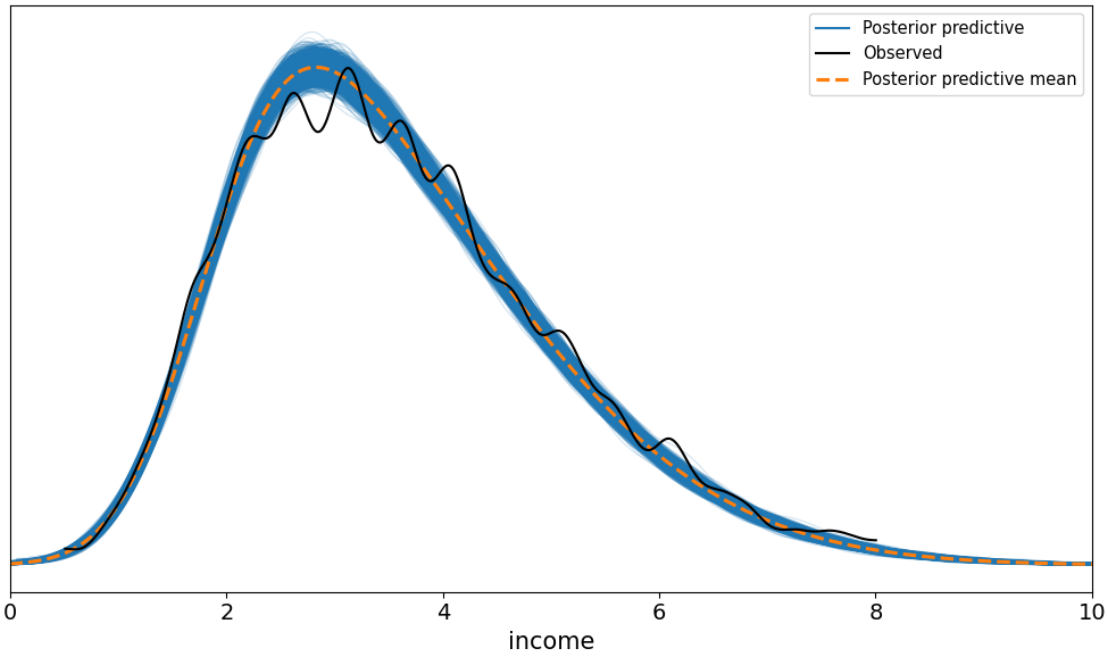
```
Moyenne prédictive : 3.56
Écart-type prédictif : 1.46
Intervalle de crédibilité à 94 % : [1.13 6.38]
```

la moyenne des données observées (3,57) est très proche de la moyenne prédictive du modèle (3,56).

De même l'écart-type des données (1,44) est également très similaire à l'écart-type prédictif (1,46).

L'intervalle de crédibilité à 94% indique que la majorité des valeurs de la variable d'intérêt se trouvent dans la plage de 1.13 à 6.38. Or, les valeurs observées en dehors de cet intervalle suggèrent que des observations extrêmes peuvent être présentes dans vos données. Ce qui n'est pas idéal.

[97]:
```
fig, ax = plt.subplots(figsize=(10, 6))
az.plot_ppc(ppc, ax=ax, kind='kde')
ax.set_xlim(0, 10)
plt.tight_layout()
plt.show()
```



Visuellement, la distribution prédictive a posteriori semble globalement bien capturer la forme générale et la dispersion de la distribution observée.

Ainsi la vérification suggère que le modele est bien capable de générer des données qui ressemblent qualitativement aux données income observées.

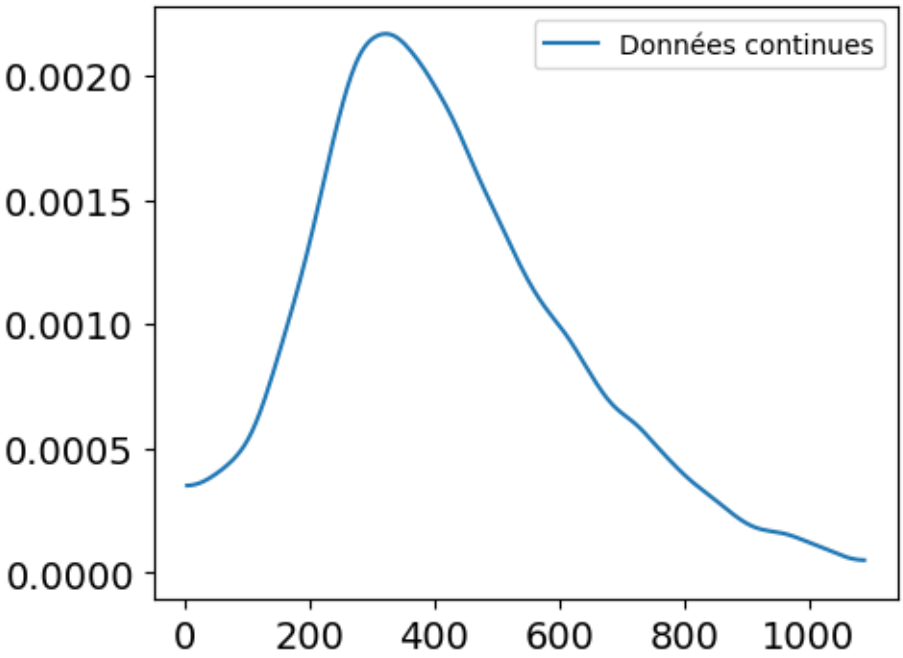
5.3 C- Households

Il s'agit d'une variable continue avec une distribution unimodale (avec un pic principal) centrée autour de 350, avec une asymétrie positive. Il y a une concentration importante des données autour de cette valeur, mais aussi une certaine dispersion vers des valeurs plus élevées.

Nous allons donc tester plusieurs variation d'un modele skew-normal

[98]:
```
Households=Og_Housing['households'].values
plt.figure(figsize=(5, 4))
az.plot_dist(Households, label="Données continues")
```

[98]: <Axes: >



5.3.1 Création des modèles

[99]:
```
with pm.Model() as skew_normal1:
  mu = pm.Normal("mu", mu=0, sigma=5)
  sigma = pm.HalfNormal("sigma", sigma=5)
  a = pm.Normal("a", mu=0, sigma=2)
  households_dist = pm.SkewNormal("households", mu=mu, sigma=sigma, alpha=a,␣
 ↪observed=Households)
  trace_skew_normal1 = pm.sample(1000, tune=500, target_accept=0.95,␣
 ↪idata_kwargs= {'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, a]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 18 seconds.
```

[100]:
```
with pm.Model() as skew_normal2:
  mu = pm.Normal("mu", mu=0, sigma=1)
  sigma = pm.HalfNormal("sigma", sigma=1)
  a = pm.Normal("a", mu=0, sigma=2)
```

```
  households_dist = pm.SkewNormal("households", mu=mu, sigma=sigma, alpha=a,␣
 ↪observed=Households)
  trace_skew_normal2 = pm.sample(1000, tune=500, target_accept=0.95,␣
 ↪idata_kwargs= {'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, a]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 12 seconds.
```

[101]:
```
with pm.Model() as skew_normal3:
  mu = pm.Normal("mu", mu=300, sigma=50)
  sigma = pm.HalfNormal("sigma", sigma=100)
  a = pm.Normal("a", mu=0, sigma=5)
  households_dist = pm.SkewNormal("households", mu=mu, sigma=sigma, alpha=a,␣
 ↪observed=Households)
  trace_skew_normal3 = pm.sample  (1000, tune=500, target_accept=0.95,␣
 ↪idata_kwargs= {'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [mu, sigma, a]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 13 seconds.
```

5.3.2 Comparaison

[102]:
```
comp1=pm.compare({
    "model_skew_normal1":trace_skew_normal1,
    "model_skew_normal2":trace_skew_normal2,
    "model_skew_normal3":trace_skew_normal3
})
comp1
```

[102]:

	rank	elpd_loo	p_loo	elpd_diff	weight	\
model_skew_normal3	0	-116922.019937	2.537647	0.000000	1.000000e+00	
model_skew_normal1	1	-117057.582542	2.534595	135.562605	0.000000e+00	
model_skew_normal2	2	-127456.956192	2.981303	10534.936255	1.238620e-14	

	se	dse	warning	scale

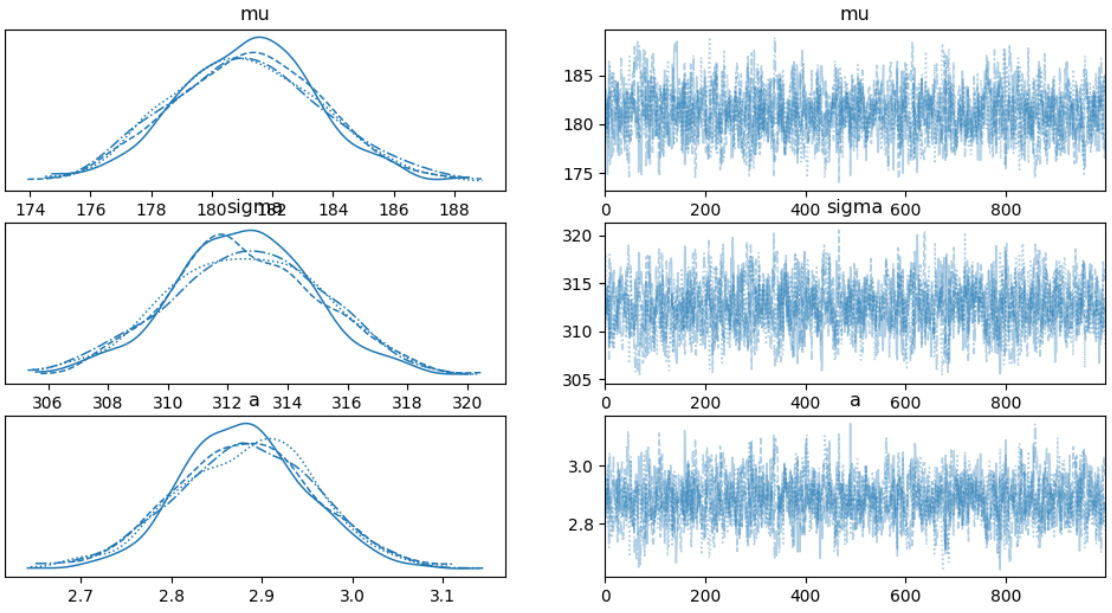
```
model_skew_normal3   92.169978    0.000000    False   log
model_skew_normal1  105.938012   16.910295    False   log
model_skew_normal2  253.590539  186.789613    False   log
```

On remarque que dans ce cas, plus les priors sont informatifs, mieux se comporte le modèle.

5.3.3 Résultat du meilleur modele - skew_normal3

[103]:
```
az.plot_trace(trace_skew_normal3, var_names=["mu", "sigma", "a"])
az.summary(trace_skew_normal3, round_to=2)
```

[103]:
```
          mean    sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  ess_tail  \
mu      181.17  2.49  176.31   185.64       0.08     0.05   1037.46   1210.69
a         2.88  0.08    2.75     3.04       0.00     0.00   1047.34   1137.29
sigma   312.57  2.53  307.76   317.34       0.08     0.05   1076.99   1290.99

        r_hat
mu        1.0
a         1.0
sigma     1.0
```



Les chaînes semblent s'approcher de la stationnarité : les fluctuations autour de leurs moyennes respectives sont plus ou moins aléatoires et bornées, sans tendance claire à la hausse ou à la baisse sur le long terme.

Les r_hat sont à 1

[104]:
```
with skew_normal3:
    ppc = pm.sample_posterior_predictive(trace_skew_normal3,␣
 ↪var_names=["households"])
```

```
Sampling: [households]
```

```
Output()
```

[105]:
```
pd.Series(Households).describe()
```

[105]:
```
count    17434.000000
mean       416.806126
std        205.412687
min          2.000000
25%        271.000000
50%        387.000000
75%        540.000000
max       1090.000000
dtype: float64
```

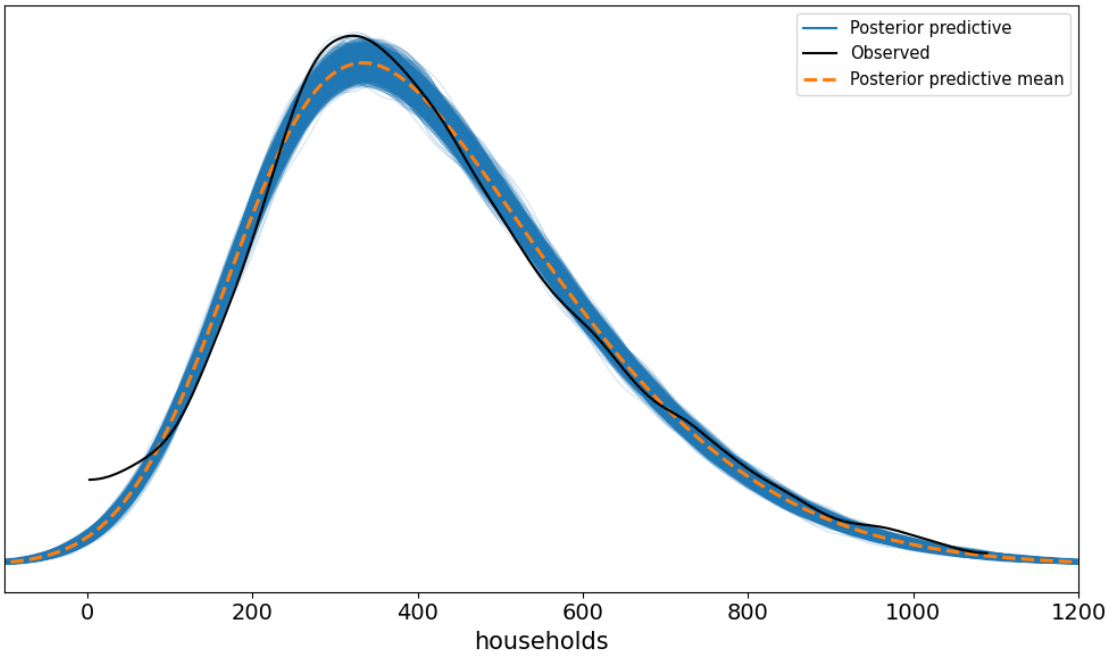
[106]:
```
pp= ppc.posterior_predictive["households"]

predicted_mean = np.mean(pp)
predicted_std = np.std(pp)
hdi = az.hdi(pp.values.flatten(), hdi_prob=0.94).round(2)

print(f"Moyenne prédictive : {predicted_mean:.2f}")
print(f"Écart-type prédictif : {predicted_std:.2f}")
print(f"Intervalle de crédibilité à 94 % : {hdi}")
```

```
Moyenne prédictive : 416.78
Écart-type prédictif : 205.39
Intervalle de crédibilité à 94 % : [ 63.77 813.73]
```

Les résultats statistiques sont plutôt satisfaisants, notamment les moyennes et les écarts-types prédictifs qui sont proches des valeurs observées.

En ce qui concerne l'intervalle de crédibilité, on remarque qu'un grand nombre des observations sont bien prises en compte par le modèle. Cependant, comme on peut s'y attendre d'un modèle normal, ce dernier sous-estime les valeurs extrêmes n'incluant donc pas le min et le max observé dans notre jeux de donnée.

[107]:
```
fig, ax = plt.subplots(figsize=(10, 6))
az.plot_ppc(ppc, ax=ax, kind='kde')
ax.set_xlim(-100, 1200)
plt.tight_layout()
plt.show()
```



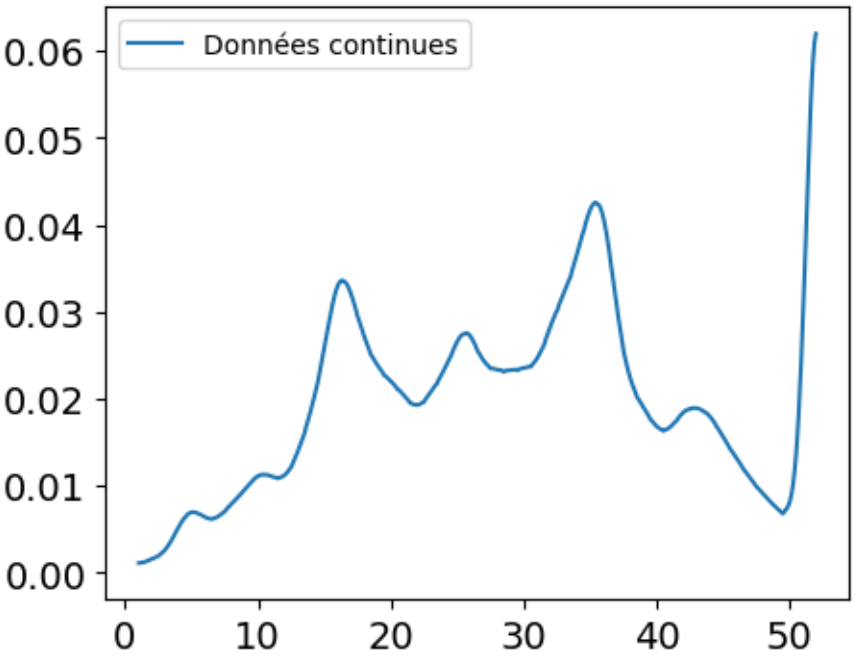
Nous remarquons également une bonne adhérence visuelle entre les distributions prédites et les observations réelles, ce qui suggère que le modèle capture correctement la tendance générale des données, bien que certaines valeurs extrêmes ne soient pas aussi bien ajustées.

La moyenne prédictive suit de maniere satisfaisante la tendance générale de la distribution.

5.4 D- Median Age

[108]:
```
Age= Og_Housing['housing_median_age'].values
plt.figure(figsize=(5, 4))
az.plot_dist(Age, label="Données continues")
```

[108]: <Axes: >

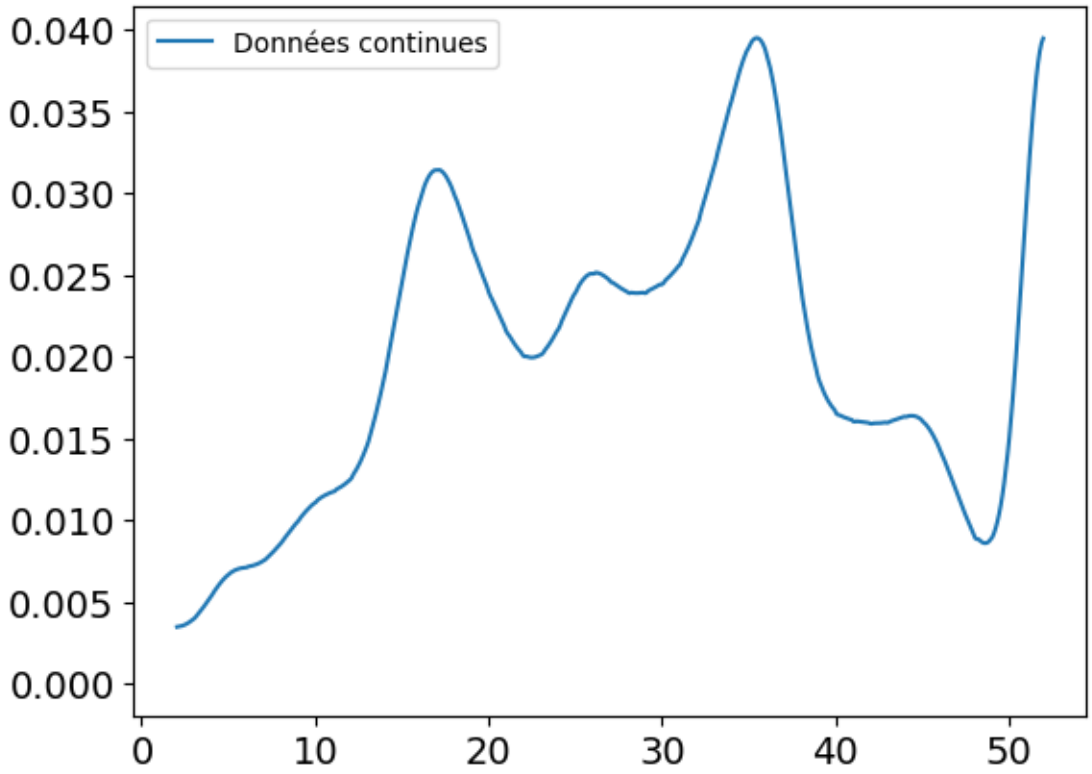


Pour modéliser la distribution de l'âge médian des maisons avec PyMC, en se basant sur la forme de la distribution, un modele multimodale pourrait être efficace.

Comme nous allons travailler sur des modèles complexes, j'ai opté pour un sous-échantillonnage de la variable Age afin de faciliter l'entraînement du modèle tout en conservant une représentation adéquate de la distribution des âges.

[109]:
```
from sklearn.model_selection import train_test_split
Age,_ = train_test_split(Age, test_size=0.90, random_state=42)
az.plot_dist(Age, label="Données continues")
```

[109]: <Axes: >



5.4.1 Création des modèles

J'opte pour différente forme de mixture

[110]:
```
#Trimodal

with pm.Model() as Trimodal:
    weights = pm.Dirichlet("weights", a=np.ones(3))
    mu = pm.Normal("mu", mu=np.linspace(Age.min(), Age.max(),3), sigma=10,␣
 ↪shape=3)
    sd = pm.HalfNormal("sd", sigma=5, shape=3)

    category = pm.Categorical("category", p=weights, observed=np.
 ↪zeros_like(Age, dtype=int))

    likelihood = pm.Normal("age", mu=mu[category], sigma=sd[category],␣
 ↪observed=Age)

    trace1 = pm.sample(1000, tune=500, target_accept=0.9, idata_kwargs=␣
 ↪{'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [weights, mu, sd]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 3 seconds.
```

[111]:
```
#Quadrimodal

with pm.Model() as Quadrimodal:
    weights = pm.Dirichlet("weights", a=np.ones(4))
    mu = pm.Normal("mu", mu=np.linspace(Age.min(), Age.max(),4), sigma=10,␣
 ↪shape=4)
    sd = pm.HalfNormal("sd", sigma=5, shape=4)

    category = pm.Categorical("category", p=weights, observed=np.
 ↪zeros_like(Age, dtype=int))

    likelihood = pm.Normal("age", mu=mu[category], sigma=sd[category],␣
 ↪observed=Age)

    trace2 = pm.sample(1000, tune=500, target_accept=0.9, idata_kwargs=␣
 ↪{'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [weights, mu, sd]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 3 seconds.
```

[112]:
```
with pm.Model() as mixture_skew_normal:
    weights = pm.Dirichlet("weights", a=np.ones(3))
    means = pm.Normal("means", mu=10, sigma=5, shape=3) # Priorité informative
    sd = pm.HalfNormal("sd", sigma=5, shape=3)
    a= pm.HalfNormal('a', sigma=5, shape=3) # Priorité faiblement informative

    components = pm.SkewNormal.dist(mu=means, sigma=sd, alpha=a)
    median_house_age_mixed = pm.Mixture("age", w=weights,␣
 ↪comp_dists=components, observed=Age)

    trace3 = pm.sample(1000, tune=500, target_accept=0.9, idata_kwargs=␣
 ↪{'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [weights, means, sd, a]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 124 seconds.
There were 61 divergences after tuning. Increase `target_accept` or
reparameterize.
The rhat statistic is larger than 1.01 for some parameters. This indicates
problems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A
higher number is needed for reliable rhat and ess computation. See
https://arxiv.org/abs/1903.08008 for details
```

[113]:
```
with pm.Model() as mixture_skew_student:
    weights = pm.Dirichlet("weights", a=np.ones(3))
    means = pm.Normal("means", mu=10, sigma=5, shape=3) # Priorité informative
    sd = pm.HalfNormal("sd", sigma=5, shape=3)
    a= pm.HalfNormal('a', sigma=5, shape=3)
    b= pm.HalfNormal('b', sigma=5, shape=3)# Priorité faiblement informative

    components = pm.SkewStudentT.dist(mu=means, sigma=sd, a=a, b=b)
    median_house_age_mixed = pm.Mixture("age", w=weights,␣
 ↪comp_dists=components, observed=Age)

    trace4 = pm.sample(1000, tune=500, target_accept=0.9, idata_kwargs=␣
 ↪{'log_likelihood': True})
```

```
Initializing NUTS using jitter+adapt_diag…
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [weights, means, sd, a, b]
```

```
Output()
```

```
Sampling 4 chains for 500 tune and 1_000 draw iterations (2_000 + 4_000 draws
total) took 42 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates
problems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A
higher number is needed for reliable rhat and ess computation. See
https://arxiv.org/abs/1903.08008 for details
```

5.4.2 Comparaison

[114]:
```
com=az.compare({
    "Trimodal_normal":trace1,
    "Quadrimodal_normal":trace2,
    "Trimodal_skewnormal":trace3,
    "mixture_skew_student":trace4
},var_name="age")
com
```

```
/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/stats/stats.py:1042: RuntimeWarning:

overflow encountered in exp

/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/numpy/_core/_methods.py:51: RuntimeWarning:

overflow encountered in reduce

/Users/yerimasika/miniconda3/envs/spark-env/lib/python3.11/site-
packages/arviz/stats/stats.py:782: UserWarning:

Estimated shape parameter of Pareto distribution is greater than 0.70 for one or
more samples. You should consider using a more robust model, this is because
importance sampling is less likely to work well if the marginal posterior and
LOO posterior are very different. This is more likely to happen with a non-
robust model and highly influential observations.
```

[114]:

	rank	elpd_loo	p_loo	elpd_diff	weight	\
mixture_skew_student	0	-6792.211861	8.398003	0.000000	1.000000e+00	
Trimodal_skewnormal	1	-6804.990660	281.418735	12.778799	2.197409e-12	
Trimodal_normal	2	-6842.914662	1.548318	50.702801	0.000000e+00	
Quadrimodal_normal	3	-6842.946709	1.570535	50.734848	0.000000e+00	

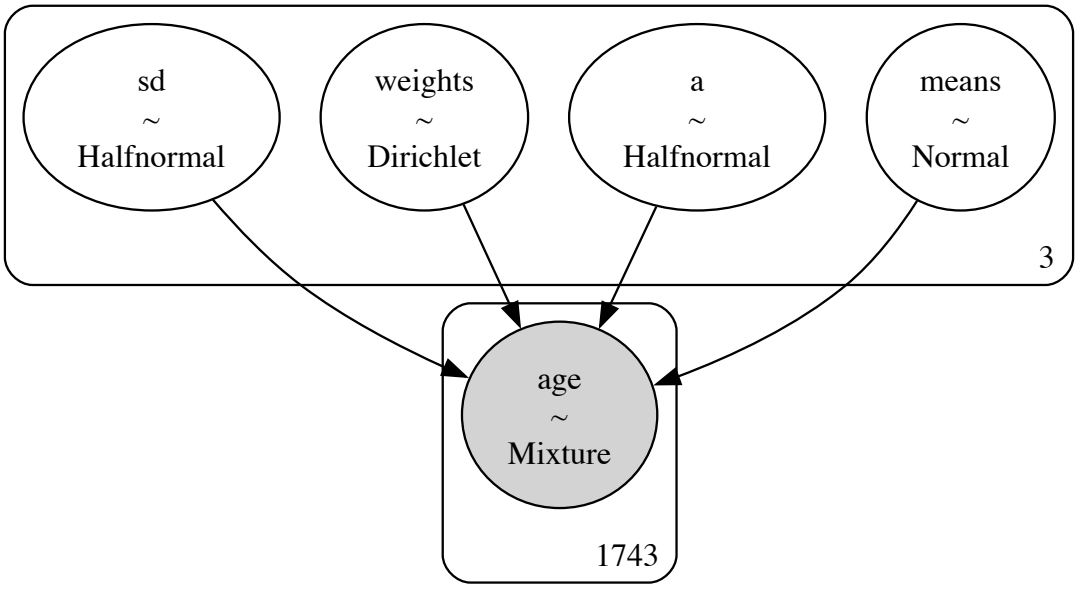
	se	dse	warning	scale
mixture_skew_student	22.368100	0.000000	False	log
Trimodal_skewnormal	21.267173	2.833741	True	log
Trimodal_normal	23.003000	9.566693	False	log
Quadrimodal_normal	23.005755	9.572268	False	log

Le modèle Trimodal_skewnormal est celui qui a la meilleure performance parmi les trois, avec la valeur elpd_loo la plus élevée (la moins négative) et un poids de 1. Ce modèle a une meilleure capacité de prédiction par rapport aux autres.

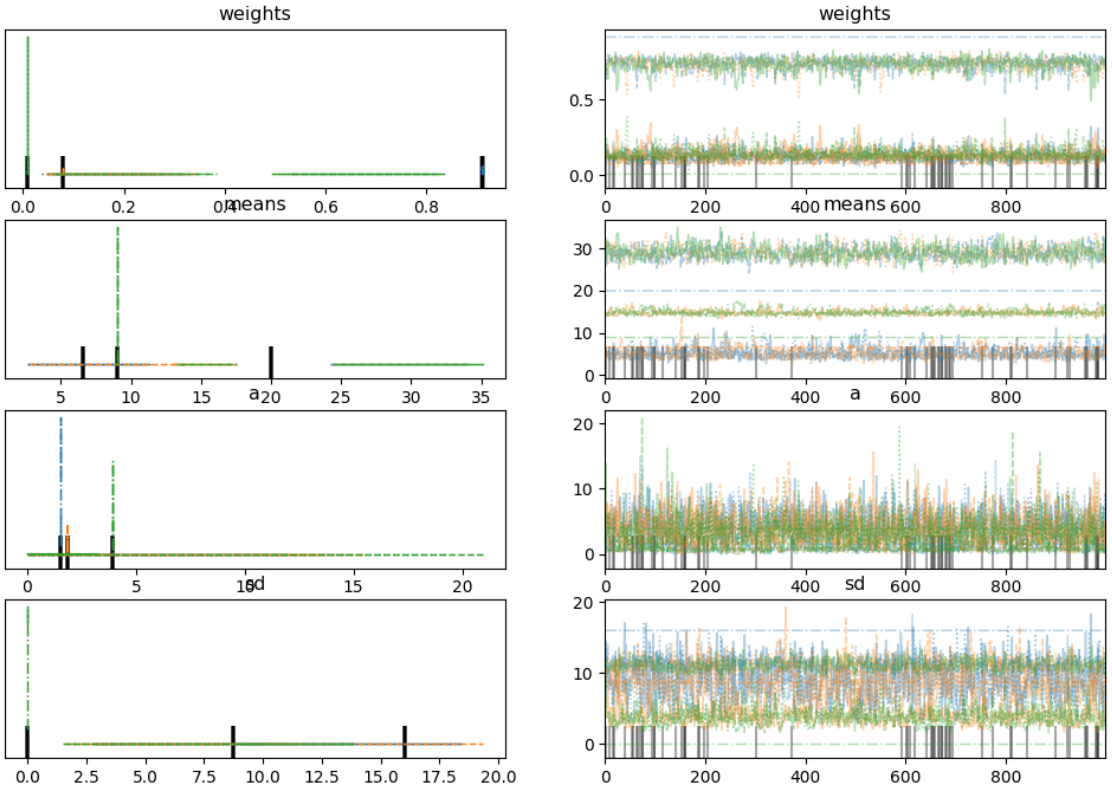
5.4.3 Résultat du meilleur modèle - Trimodal_skewnormal

[115]:
```
pm.model_to_graphviz(mixture_skew_normal)
```

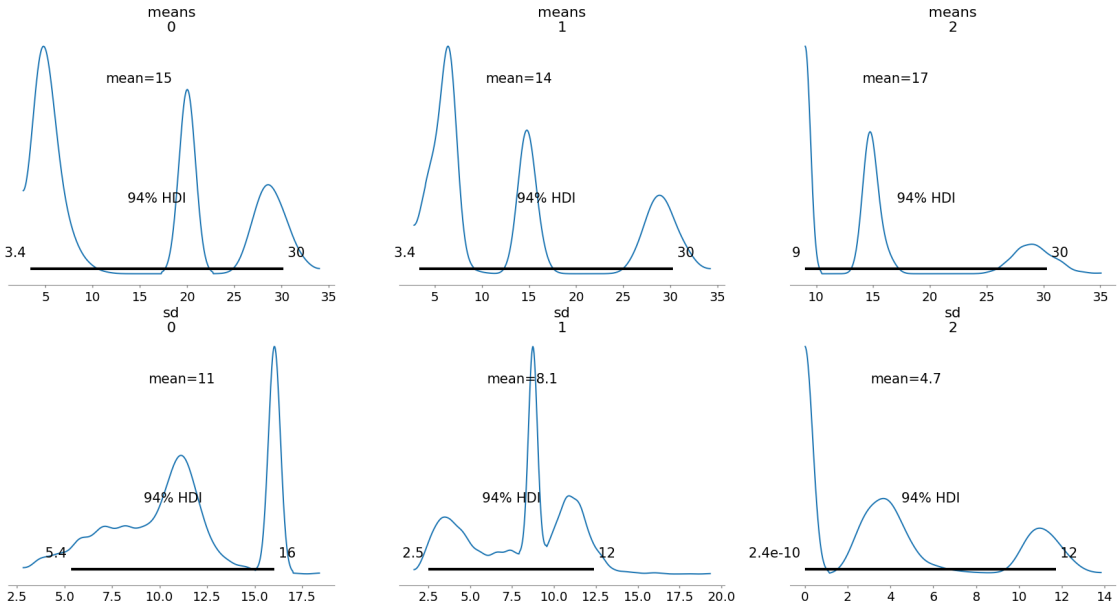
[115]:



[116]:
```
az.plot_trace(trace3, var_names=["weights", "means",'a', "sd"])
plt.show()
```



[117]:
```
az.plot_posterior(trace3, var_names=["means", "sd"])
plt.show()
az.summary(trace3, var_names=["means", "sd"],round_to=2)
```



[117]:
```
            mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  \
means[0]   14.85  10.17    3.37    30.19       5.03     1.53      5.22
means[1]   13.92   9.58    3.38    30.28       4.74     2.25      4.79
means[2]   16.98   7.48    9.00    30.30       3.70     2.00      4.92
sd[0]      11.15   3.61    5.35    16.02       1.51     0.57      6.42
sd[1]       8.11   3.05    2.51    12.37       1.32     0.55      6.01
sd[2]       4.71   4.15    0.00    11.73       2.02     1.00      4.88

           ess_tail  r_hat
means[0]      33.37   2.16
means[1]      29.22   2.63
means[2]      29.79   2.41
sd[0]         54.19   1.67
sd[1]         35.99   2.00
sd[2]         24.13   2.42
```

[118]:
```
# Vérification prédictive a posteriori
with mixture_skew_normal:
    ppc = pm.sample_posterior_predictive(trace3)
```

```
Sampling: [age]
```

```
Output()
```

[119]:
```
pd.Series(Age).describe()
```

[119]:
```
count    1743.000000
mean       29.209983
std        12.259375
min         2.000000
25%        19.000000
50%        30.000000
75%        37.000000
max        52.000000
dtype: float64
```

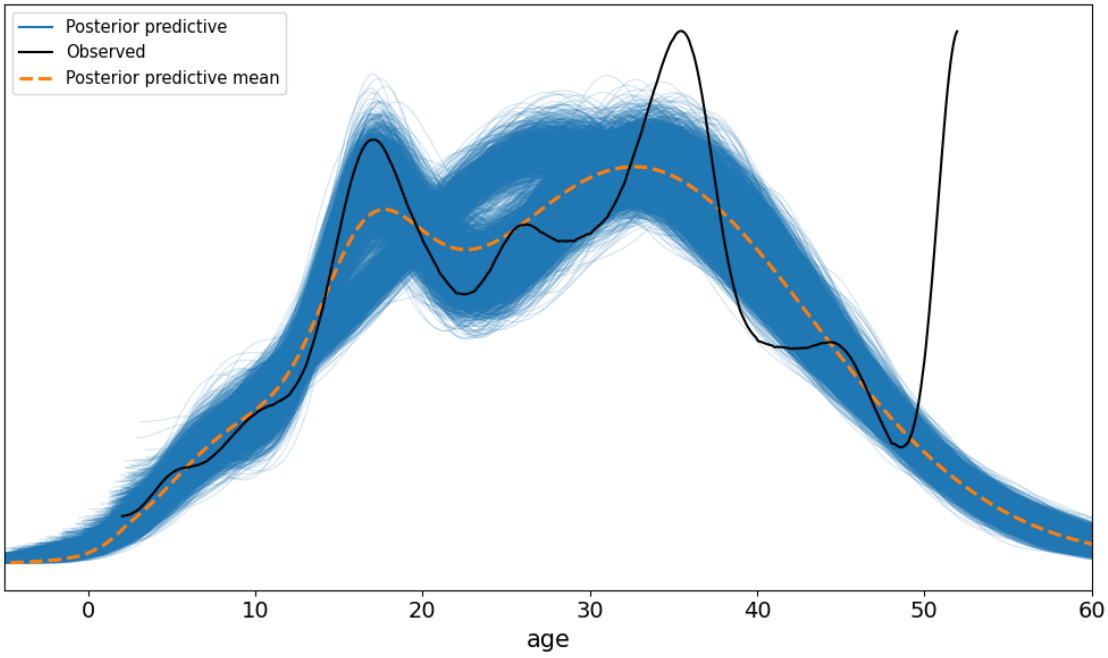
[120]:
```
pp= ppc.posterior_predictive["age"]

predicted_mean = np.mean(pp)
predicted_std = np.std(pp)
hdi = az.hdi(pp.values.flatten(), hdi_prob=0.94).round(2)

print(f"Moyenne prédictive : {predicted_mean:.2f}")
print(f"Écart-type prédictif : {predicted_std:.2f}")
```

```
print(f"Intervalle de crédibilité à 94 % : {hdi}")
```

```
Moyenne prédictive : 29.14
Écart-type prédictif : 12.36
Intervalle de crédibilité à 94 % : [ 5.98 50.56]
```

Les moyennes prédictives similaires à celles observées (Moyenne : 29.15, Écart-type : 12.26)

[121]:
```
fig, ax = plt.subplots(figsize=(10, 6))
az.plot_ppc(ppc, ax=ax, kind='kde')
ax.set_xlim(-5, 60)
plt.tight_layout()
plt.show()
```



Visuellement, l'ajustement du modèle, bien que non parfait, demeure globalement très satisfaisant. On observe une bonne concordance entre les courbes de densité prédictives postérieures et la distribution empirique, notamment dans les zones centrales de la distribution.

La moyenne prédictive suit de manière la tendance générale de la distribution.

Certaines zones présentent des écarts notables, en particulier aux extrémités de la distribution (au-delà de 50 ans, par exemple), où le modèle tend à lisser davantage que la courbe observée. Cela peut s'expliquer par la nature de la distribution normale, qui représente mal des comportements extrêmes.

6 Conclusion

Ce travail a permis d'explorer l'application de méthodes bayésiennes avancées à la modélisation de données complexes issues du jeu de données California Housing.

Dans un premier temps, la régression linéaire bayésienne sur median_house_value a mis en évidence l'effet significatif de plusieurs variables explicatives, tout en fournissant des intervalles de crédibilité permettant une interprétation plus nuancée que les simples intervalles de confiance fréquentistes. Les distributions postérieures des coefficients, ainsi que les prédictions issues du posterior predictive check, ont montré une bonne capacité du modèle à capturer la tendance centrale et la dispersion des données observées.

Après avoir identifié les variables les plus pertinentes, nous avons proposé une manière de les modéliser.

La modélisation des variables catégorielles, comme ocean_proximity, à l'aide de modèles multinouilliens et multinomiaux bayésiens, a révélé une forte concordance entre les proportions postérieures estimées et les proportions observées. Les diagnostics LOO et WAIC, bien que légèrement affectés par le caractère agrégé des données, ont confirmé la qualité des ajustements.

Pour les variables continues (income, age), l'ajustement de distributions asymétriques (Skew Normal, Student, Skew Student) et de mélanges de lois a permis de capturer plus fidèlement les formes complexes des distributions empiriques, notamment les queues épaisses et la multimodalité. Les comparaisons par critères d'information (elpd_loo, WAIC) ont systématiquement favorisé les modèles les plus flexibles.

En somme, ce travail illustre l'efficacité des méthodes bayésiennes pour modéliser, prédire et interpréter des données complexes. Il constitue une base solide pour des analyses futures incluant des structures plus riches, notamment spatio-temporelles ou multi-niveaux.