```python
import numpy as np
import pandas as pd
import pymc as pm
import arviz as az


def get_VAR_arr(data: np.array, n_lags: int) -> np.array:
    return np.concatenate([data[i:len(data) - (n_lags - i)] for i in range(1, n_lags + 1)], axis=1)


def get_samp(max_dim, size=100):
    return np.random.randint(0, max_dim, min(size, max_dim))


def get_gp_smoothing(y: np.array):
    """
    Uses Gaussian process to implement spline regression smoothing
    :param y:
    :return:
    """
    X = np.linspace(0, 2, len(y))[:, None]

    with pm.Model() as gp_mod:
        ell = pm.Gamma("ell", alpha=2, beta=1)
        eta = pm.HalfNormal("eta", sigma=5)

        cov = eta ** 2 * pm.gp.cov.ExpQuad(1, ell)
        gp = pm.gp.Latent(cov_func=cov)

        f = gp.prior("f", X=X)

        sigma = pm.HalfNormal("sigma", sigma=2.0)
        nu = 1 + pm.Gamma(
            "nu", alpha=2, beta=0.1
        )  # add one because student t is undefined for degrees of freedom less than one
        obs = pm.Deterministic('obs', nu)
        y_ = pm.StudentT("y", mu=f, lam=1.0 / sigma, nu=nu, observed=y)

        prior = pm.sample_prior_predictive()
        trace = pm.sample(1000, nuts_sampler="numpyro", tune=1000, chains=2)
        post = pm.sample_posterior_predictive(trace)

    return gp_mod, prior, trace, post


def get_pymc_mod_table(idata, lst_params=None, n_round: int = 2, seperator: str = "\n"):
    """
    Prints PYMC model summary table
    :param idata:
    :param lst_params:
    :param n_round:
    :param seperator:
    :return:
    """
    assert 'log_likelihood' in list(idata.keys()), "pls add idata_kwargs = {'log_likelihood': True}) to pm.sample"

    dict_params, dict_information = {}, {}
```

```
    n_chains = len(idata.sample_stats.chain)
    n_draws = len(idata.sample_stats.draw)
    post = idata.posterior

    if lst_params is None:
        lst_params = list(idata.posterior.data_vars.keys())

    for param in lst_params:

        shape_param = post[param].values.shape[2:]
        if len(shape_param) == 0:
            shape_param = (1,)
        assert len(shape_param) == 1, "param dimension >1 not supported"

        arr_param_est = post[param].values.reshape(*shape_param, n_chains * n_draws)

        for idx in range(*shape_param):
            _dict = {}
            _dict['mean'] = arr_param_est.mean(axis=1)[idx]
            _dict['std'] = arr_param_est.std(axis=1)[idx]
            _dict['confu'] = np.percentile(arr_param_est, 97.5, axis=1)[idx]
            _dict['confl'] = np.percentile(arr_param_est, 2.5, axis=1)[idx]

            dict_params[f'{param}_{idx}'] = _dict

    dict_information['N chains'] = n_chains
    dict_information['N draws'] = n_draws
    dict_information['N'] = len(idata.observed_data.likelihood)
    dict_information['LOO'] = az.loo(idata).p_loo
    dict_information['WAIC'] = az.waic(idata).p_waic
    dict_information['MCMC divergence'] = idata.sample_stats.diverging.sum().values

    df_params = pd.DataFrame(dict_params).T

    dict_information = {key: dict_information for key in df_params.columns}
    df_info = pd.DataFrame(dict_information)

    df = pd.concat(
        [
            pd.DataFrame(dict_params).T,
            pd.DataFrame(dict_information),

        ],
        axis=0,
    ).astype(float)

    df['print'] = df['mean'].round(n_round).astype(str) + seperator + "[" + df['confl'].round(n_round).astype(str) \
                + "," + df['confu'].round(n_round).astype(str) + "]"

    return df
```