000_Intro

December 5, 2024

Oreum Industries Reference Project, 2024Q1

---

1 000_Intro.ipynb

1.0.1 Oreum Reference - Copula Regression `oreum_copula`

Demonstrate Bayesian Copula Regression Modelling using Bayesian inference and a Bayesian workflow, specifically using the `pymc` & `arviz` ecosystem.

This **Intro** can also be used for verbal presentation and discussion purposes, ideally followed by a deeper technical walkthrough of the project in a long-form style. Because this project is a reference, it contains huge amounts of detail which is not worthwhile to summarise too much.

The interested reader should refer to the project notebooks where we evaluate the behaviour and performance of the models throughout the workflows, including several state-of-the-art methods unavailable to conventional max-likelihood / machine-learning models.

1.1 ***What is Copula Regression?***

We seek to create *principled* models that provide explanatory inference and predictions of Marginal distributions M that are jointly coupled by a Latent Copula C, using quantified uncertainty to support real-world decision-making.

Motivation:

- A classic use-case for this model architecture in the 2-dimensional setting is insurance claims frequency and severity
- The `frequency` of claims and the `severity` of each claim each have marginal distributions and a natural covariance Σ between marginals M_0, M_1
- The joint product `frequency * severity = Loss Cost` i.e. the dollar value of insurable losses
- If we use a naive model that doesn't account for the covariance between `frequency` and `severity`, then the model predictions for `Loss Cost` can be hugely wrong

Demonstration:

- In this notebook:

  - We create a small synthetic dataset of observations of two marginals M_0, M_1 which have covariance Σ, and also (because we can) a version of the marginals M_{0x}, M_{1x} without covariance
  - We compare the resulting values of the joint product $y = M_0 * M_1$ vs $y = M_{0x} * M_{1x}$ and see that impact of ignoring the covariance is substantial.
- In the rest of the reference guide:
  - We create a series of principled copula models using advanced architectures and Bayesian inference to fit to the data and estimate the covariance on M_0, M_1
  - The first model is naive and ignores the covariance, the final model is very sophisticated and estimates the covariance
  - We demonstrate a substantial 32 percentage-point improvement in model accuracy when using the copula

General project approach

The emphasis in this project is to build a variety of models of increasing sophistication and demonstrate their usage. We strike a balance between building up concepts & methods vs practical application & worked examples in a `pmyc`-based Bayesian workflow.

We don't focus on specific analysis of the dataset, nor try to infer too much. The dataset is simply a good substrate on which to learn and demonstrate the variety of model architectures used herein.

We evaluate the behaviour and performance of the models throughout the workflows, including several state-of-the-art methods unavailable to conventional max-likelihood / machine-learning models

This series of Notebooks covers

- `000_Intro.ipynb`: Orientiation and fundamental concepts
- `100_ModelA0.ipynb`: Core (naive) architecture: Create priors, marginal likelihoods, but no copula
- `101_ModelA1.ipynb`: Partial architecture (extends ModelA0): Include Gaussian copula (w/ Jacobian adjustment), and several technical innovations to let `pymc` work with the transformations
- `102_ModelA2.ipynb`: Full architecture (extends ModelA1): Include Jacobian Adjustment on transformed observations

In this Notebook

We dive straight into **Orientation** and **Fundamental General Abstractions** with a simple real-world observational censored dataset, and then go on to demonstrate the theory and usage of an increasing sophistication of models.

1.2 Contents

2 Setup

2.0.1 Imports

```
import sys
from pathlib import Path

import numpy as np
import pandas as pd
from oreum_core import eda
from pyprojroot.here import here

# prepend local project src files
module_path = here('src').resolve(strict=True)
if str(module_path) not in sys.path:
    sys.path.insert(0, str(module_path))    # sys.path.append(str(module_path))

# autoreload local modules to allow local dev
%load_ext autoreload
%autoreload 2
from engine import logger
from synthetic.create_copula import CopulaBuilder

import warnings  # noqa
warnings.simplefilter(action='ignore', category=FutureWarning)  # noqa
warnings.simplefilter(action='ignore', category=UserWarning)  # noqa
import seaborn as sns
```

2.0.2 Notebook config

```
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

log = logger.get_logger('000_Intro', notebook=True)
_ = logger.get_logger('oreum_core', notebook=True)
```

2.0.3 Local Functions and Global Vars

```
RSD = 42
RNG = np.random.default_rng(seed=RSD)
```

2.0.4 Data Connections

```
figio = eda.FigureIO(here(Path('plots')).resolve(strict=True))
```

---

3 Preamble: Why Bayes?

3.1 We gain massive advantage by using a Bayesian Framework

We specifically use **Bayesian Inference** rather than Frequentist Max-Likelihood methods for many reasons, including:

	Bayesian Inference	Frequentist Max-Likelihood
General formulation $\rightarrow$ **Desirable Trait** $\downarrow$	*Bayes' Rule* $\underbrace{P(\hat{\mathcal{H}}\Vert D)}_{\text{posterior}} = \frac{\overbrace{P(D\Vert\mathcal{H})}^{\text{likelihood}} \cdot \overbrace{P(\mathcal{H})}^{\text{prior}}}{\underbrace{P(D)}_{\text{evidence}}}$	*MLE* $\hat{\mathcal{H}}^{\text{MLE}} \propto \arg\max_{\mathcal{H}} P(D\Vert\mathcal{H})$
***Principled* model structure represents hypothesis about the data-generating process**	**Very strong** Can build bespoke arbitrary and hierarchical structures of parameters to map to the real-world data-generating process.	**Weak** Can only state structure under strict limited assumptions of model statistical validity.
Model parameters and their initial values represent domain expert knowledge	**Very strong** Marginal prior distributions represent real-world probability of parameter values before any data is seen.	**Very weak** No concept of priors. Lack of joint probability distribution can lead to discontinuities in parameter values.
Robust parameter fitting process	**Strong** Estimate full joint posterior probability mass distribution for parameters - more stable and representative of the expectation for the parameter values. Sampling can be a computationally expensive process.	**Weak** Estimate single-point max-aposterioi-likelihood (density) of parameters - this can be far outside the probability mass and so is prone to overfitting and only correct in the limit of infinite data. But optimization method can be computationally cheap.
Fitted parameters have meaningful summary statistics for inference	**Very strong** Full marginal probability distributions can be interpreted exactly as probabilities.	**Weak** Point estimates only have meaningful summary statistics under strict limited assumptions of model statistical validity.

continues …

... continued

Desirable Trait	Bayesian Inference	Frequentist Max-Likelihood
Robust model evaluation process	**Strong** Use entire dataset, evaluate via Leave-One-Out Cross Validation (best theoretically possible).	**Weak** Cross-validation rarely seen in practice, even if used, rarely better than 5-fold CV. Simplistic method can be computationally cheap.
Predictions made with quantified variance	**Very strong** Predictions made using full posterior probability distributions, so predictions have full empirical probability distributions.	**Weak** Predictions using point estimates can be bootstrapped, but predictions only have interpretation under strict limited assumptions of model validity.
Handle imbalanced, high cardinality & hierarchical factor features	**Very strong** Can introduce partial-pooling to automatically balance factors through hierarchical priors.	**Weak** Difficult to introduce partial-pooling (aka mixed random effects) without affecting strict limited assumptions of model validity.
Handle skewed / multimodal / extreme value target variable	**Very strong** Represent the model likelihood as any arbitrary probability distribution, including mixture (compound) functions e.g. a zero-inflated Weibull.	**Weak** Represent model likelihood with a usually very limited set of distributions. Very difficult to create mixture compound functions.
Handle small datasets	**Very strong** Bayesian concept assumes that there is a probable range of values for each parameter, and that we evidence our prior on any amount of data (even very small counts).	**Very weak** Frequentist concept assumes that there is a single true value for each parameter and that we only discover that value in the limit (of infinite observations).
Automatically impute missing data	**Very strong** Establish a prior for each datapoint, evidence on the available data within the context of the model, to automatically impute missing values.	**Very weak** No inherent method. Usually impute as a pre-processing step with weak non-modelled methods.

3.2 Practical Implementations of Bayesian Inference

We briefly referenced *Bayes Rule* above, which is a useful mnemonic when discussing Bayesian Inference, but in practice the crux of putting these advanced statistical techniques into practice is estimating the evidence $P(D)$ i.e. the probability of observing the data that we use to evidence the model

$$\underbrace{P(\hat{\mathcal{H}}|D)}_{\text{posterior}} = \frac{\overbrace{P(D|\mathcal{H})}^{\text{likelihood}} \cdot \overbrace{P(\mathcal{H})}^{\text{prior}}}{\underbrace{P(D)}_{\text{evidence}}}$$

...where:

$$P(D) \sim \int_{\Theta} P(D, \theta) \; d\theta$$

This joint probability $P(D, \theta)$ of data D and parameters θ requires an almost impossible-to-solve integral over parameter-space Θ. Rather than attempt to calculate that integral, we do something that sounds far more difficult, but given modern computing capabilities is actually practical.

3.2.1 We use a Bleeding-edge MCMC Toolkit for Bayesian Inference: `pymc` & `arviz`

We use **Markov Chain Monte-Carlo (MCMC)** sampling to take a series of *ergodic*, *partly-reversible*, *partly-randomised* samples of model parameters θ, and at each step compute the ratio of log-likelihoods $\log P(D|\mathcal{H})$ between a starting position (current values) θ_{p0} and proposed "sampled" position θ_p in parameter space, so as to reduce that log-likelihood (whilst exploring the parameter space).

This results in a posterior estimate $P(\hat{\theta}|D)$:

$$P(\hat{\theta}|D) \sim \frac{\overbrace{P(D|\theta_p)}^{\text{likelihood @ proposal}} \cdot \overbrace{P(\theta_p)}^{\text{prior @ proposal}}}{\underbrace{P(D|\theta_{p0})}_{\text{likelihood @ current}} \cdot \underbrace{P(\theta_{p0})}_{\text{prior @ current}}}$$

This is the heart of MCMC sampling: for detailed practical explanations see Betancourt, 2021 and Tweicki, 2015

We use the bleeding-edge `pymc` and `arviz` Python packages to provide the full Bayesian toolkit that we require, including advanced sampling, probabilistic programming, statistical inferences, model evaluation and comparison, and more.

```
f = figio.read(fn='../assets/img/logos',figsize=(12, 2))
```

---

4 1. Orientation: Copula Functions and Their Behaviour

4.1 1.1 Create Synthetic Copula Dataset

Create synthetic copula dataset using a “forward-pass”:

1. Start at copula (`c0, c1`) ->
2. Transform to uniform (`u0, u1`) ->
3. Transform to marginals (`m0, m1`)
4. Also for comparison, create marginals (`m0x`, `m1x`) without copula

Also note, now we create 60 observations split into 2 sets: 50 for `train` (in-sample) and 10 for `holdout` (out-of-sample)

```
cb = CopulaBuilder()
df_all = cb.create(nobs=60)
cb.ref_vals
```

```
{'c_r': -0.7,
 'c_cov': array([[ 1. , -0.7],
        [-0.7,  1. ]]),
 'm0_kind': 'lognorm',
 'm1_kind': 'lognorm',
 'm0_params': {'mu': 0.2, 'sigma': 0.5},
 'm1_params': {'mu': 2.0, 'sigma': 1.0}}
```

```
perm = RNG.permutation(df_all.index.values)
df_train = df_all.loc[perm[:50]]
df_holdout = df_all.loc[perm[50:]]
```

```
eda.describe(df_train, nobs=0, get_counts=False)
```

```
              dtype count_unique   top freq     sum   mean    std   min   25%  \
ft
index: oid   object           50  i028    1     NaN    NaN    NaN  i000   NaN
c0          float64          NaN   NaN  NaN   -1.63  -0.03   0.79 -2.13 -0.64
c1          float64          NaN   NaN  NaN   -1.85  -0.04   0.86 -2.12  -0.5
u0          float64          NaN   NaN  NaN   24.61   0.49   0.26  0.02  0.26
u1          float64          NaN   NaN  NaN   24.84    0.5   0.27  0.02  0.31
m0          float64          NaN   NaN  NaN   64.71   1.29    0.5  0.42  0.89
m1          float64          NaN   NaN  NaN  493.00   9.86   8.25  0.89  4.47
u0x         float64          NaN   NaN  NaN   24.64   0.49    0.3  0.02  0.26
u1x         float64          NaN   NaN  NaN   23.01   0.46    0.3  0.01  0.16
m0x         float64          NaN   NaN  NaN   69.19   1.38   0.75  0.44  0.88
m1x         float64          NaN   NaN  NaN  517.19  10.34  11.17  0.82   2.7

              50%   75%   max
```

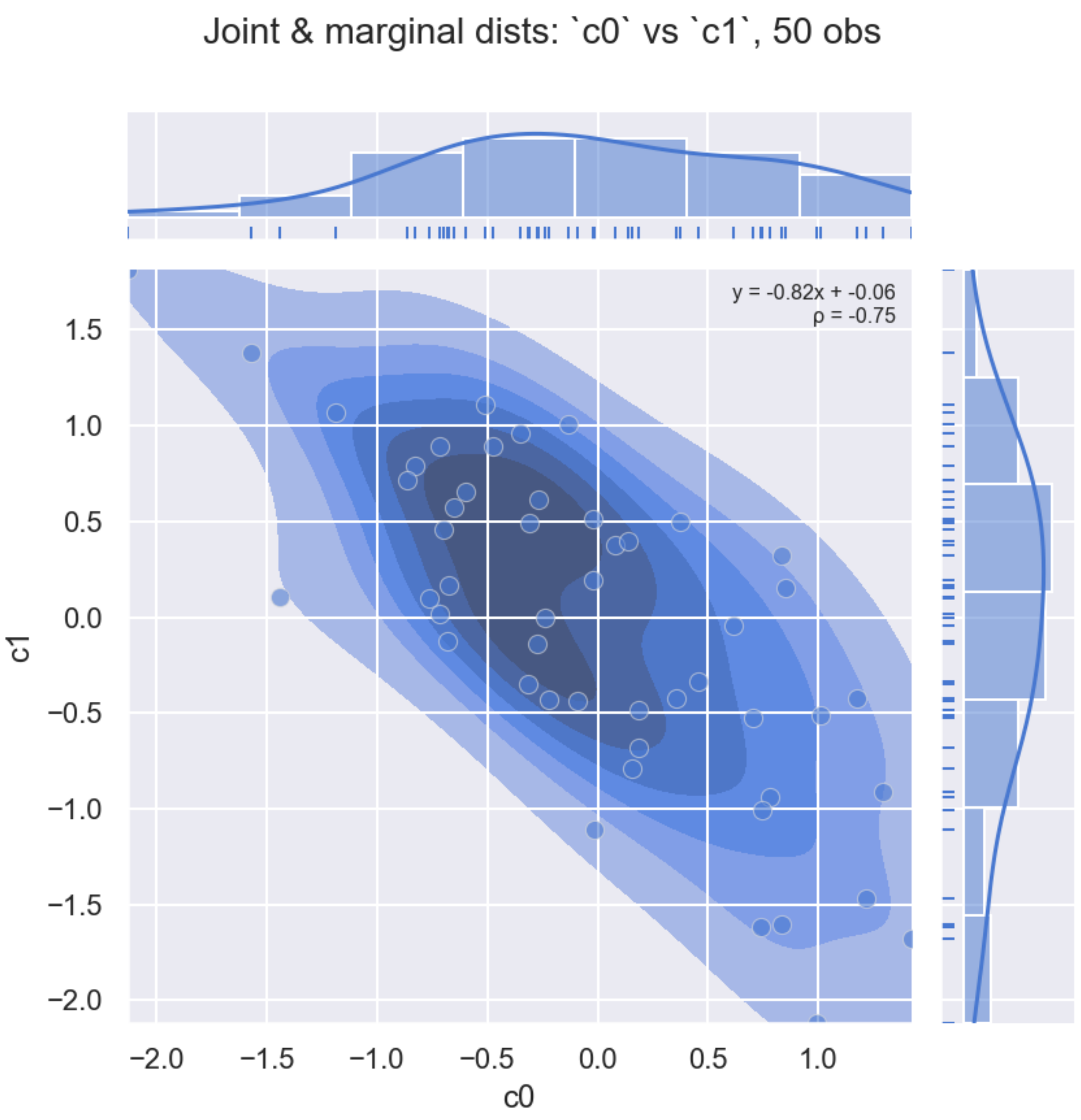
```
ft
index: oid   NaN    NaN   i059
c0         -0.06   0.68   1.42
c1          0.06   0.56   1.82
u0          0.48   0.75   0.92
u1          0.52   0.71   0.97
m0          1.19   1.72   2.49
m1          7.84  12.93  45.47
u0x         0.49   0.78   0.99
u1x         0.48   0.71   0.97
m0x          1.2   1.79   4.11
m1x         6.99  12.86  49.82
```

```
'Shape: (50, 11), Memsize 0.0 MB'
```

4.2 1.2 Visualise the Synthetic Observations

4.2.1 1.2.1 View the Copula (an MvN)

```
f = eda.plot_joint_numeric(data=df_train, ft0='c0', ft1='c1',␣
 ↪kind='kde+scatter',
             subtitle=f'Latent Copula = $MvN(0, \Sigma={cb.ref_vals["c_cov"].
 ↪flatten().tolist()})$')
```

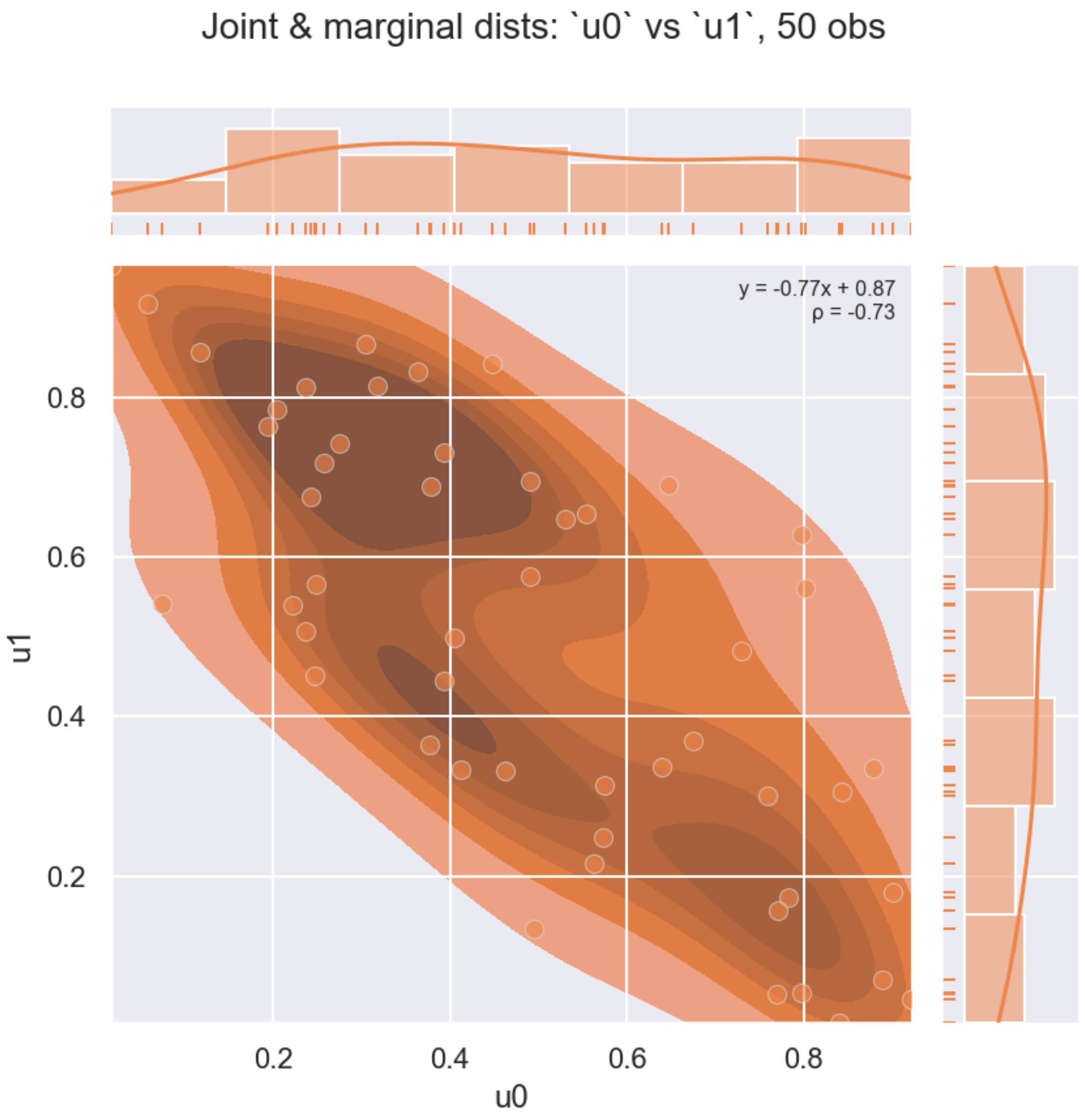


Observe:

- Note the standard `Normal(0,1)` scaling on the marginals
- Note the empirically-observed correlation $\rho \approx -0.7$ as defined in `c_cov`

4.2.2 1.2.2 View the Uniform-Transformed Marginals

```
f = eda.plot_joint_numeric(data=df_train, ft0='u0', ft1='u1',␣
  ↪kind='kde+scatter', colori=1,
        subtitle='Latent Uniform Marginals with Copula Correlation')
```



Observe:

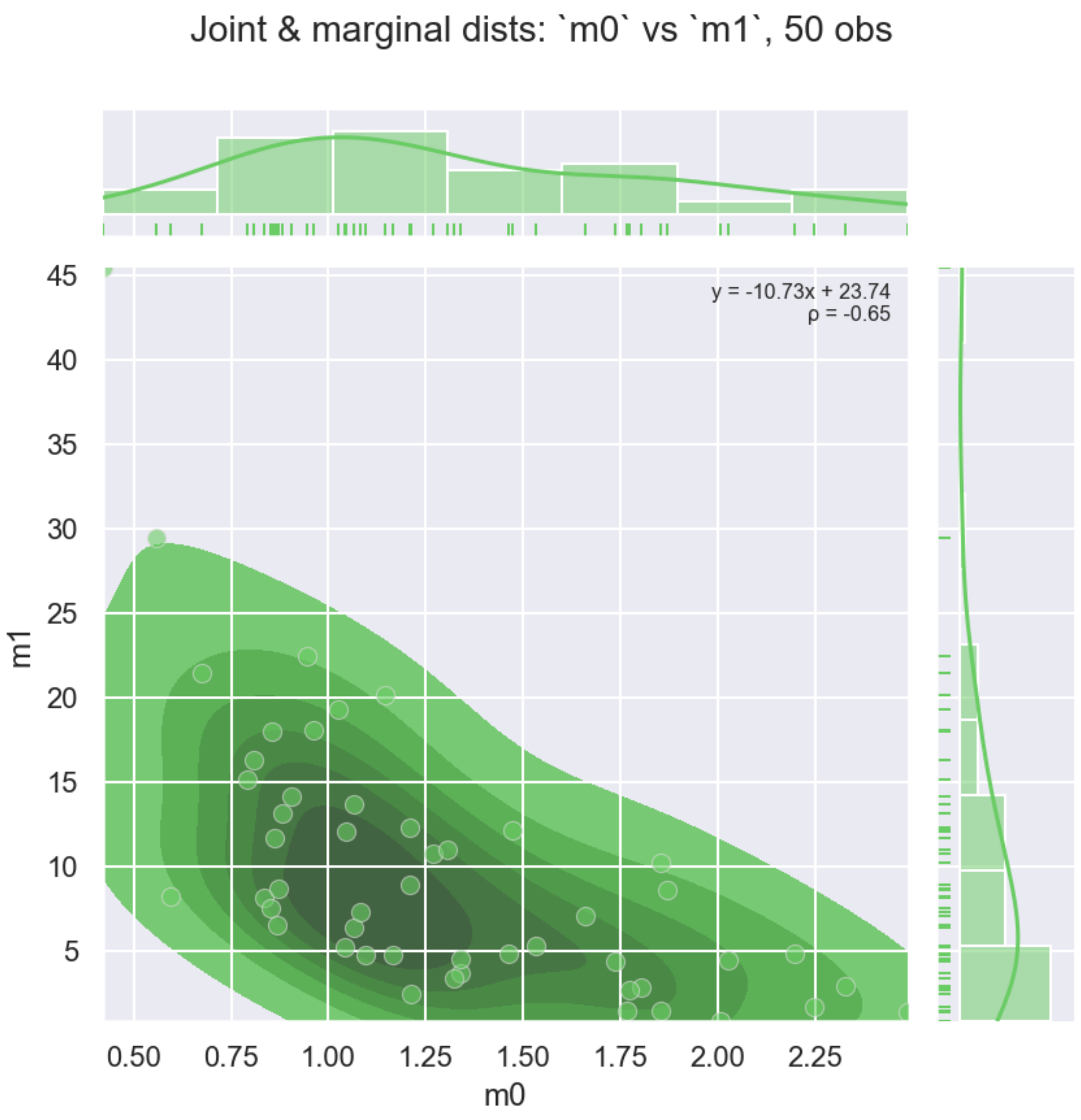
- Now the marginals are uniform, but the correlation remains

4.2.3 1.2.3 View the Observed Marginals `m0`, `m1` (post transformation)

Observe:

- This is the only real data that we would observe in the real-world dataset
- All the above plots are of latent value datapoints
- We compare this data to the model predictions (in-sample PPC)

```
f = eda.plot_joint_numeric(data=df_train, ft0='m0', ft1='m1',␣
↪kind='kde+scatter', colori=2,
              subtitle='Observed Marginals with Copula Correlation')
```



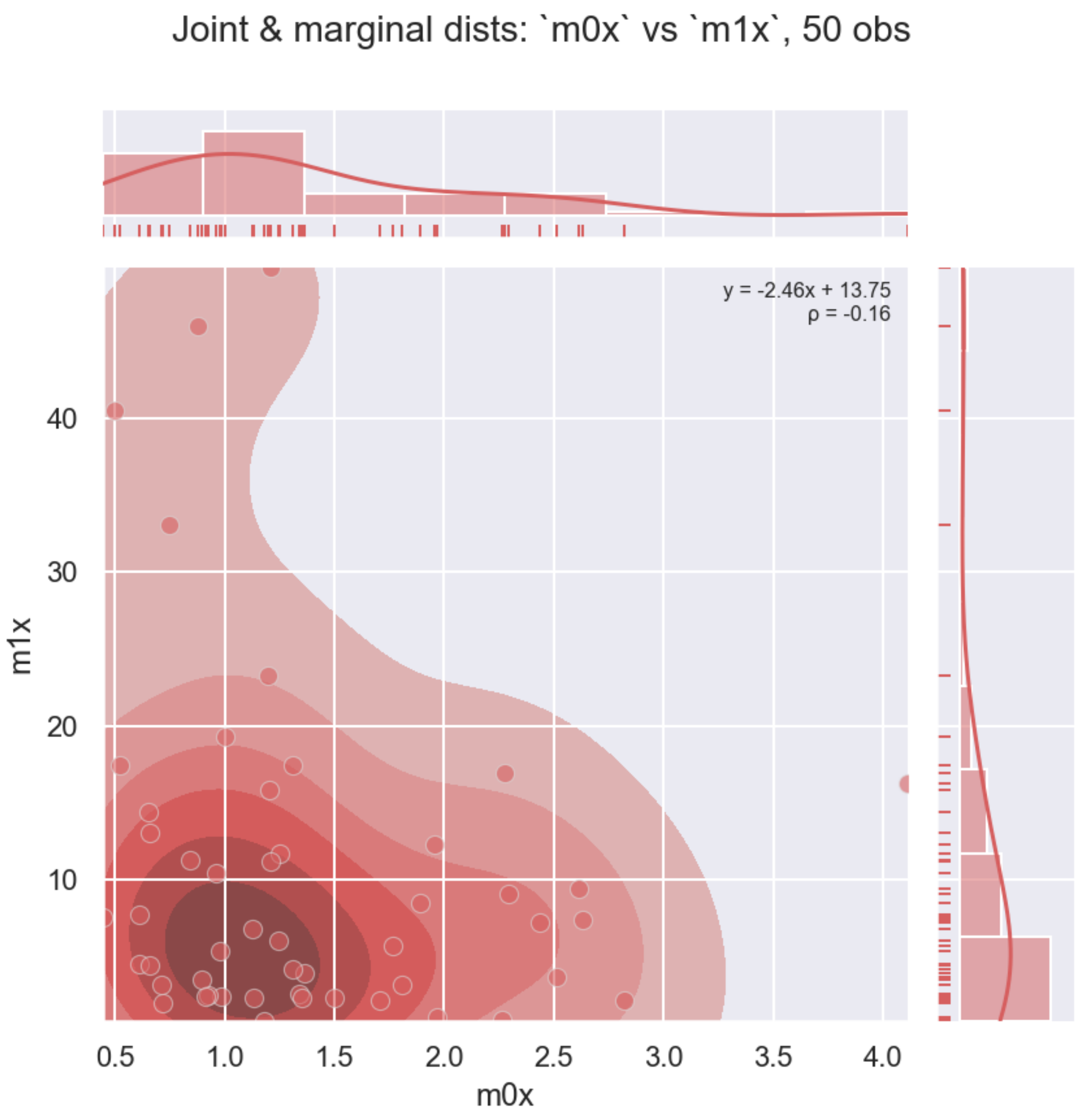
Observe

- Marginals now have unique long-tail distributions
- The correlation remains

4.2.4 1.2.4 View the Uncorrelated marginals `m0x`, `m1x` (ignoring copula)

Observed Marginals (uncorrelated)

```
f = eda.plot_joint_numeric(data=df_train, ft0='m0x', ft1='m1x',␣
 ↪kind='kde+scatter', colori=3,
              subtitle='Observed Marginals without Copula Correlation')
```

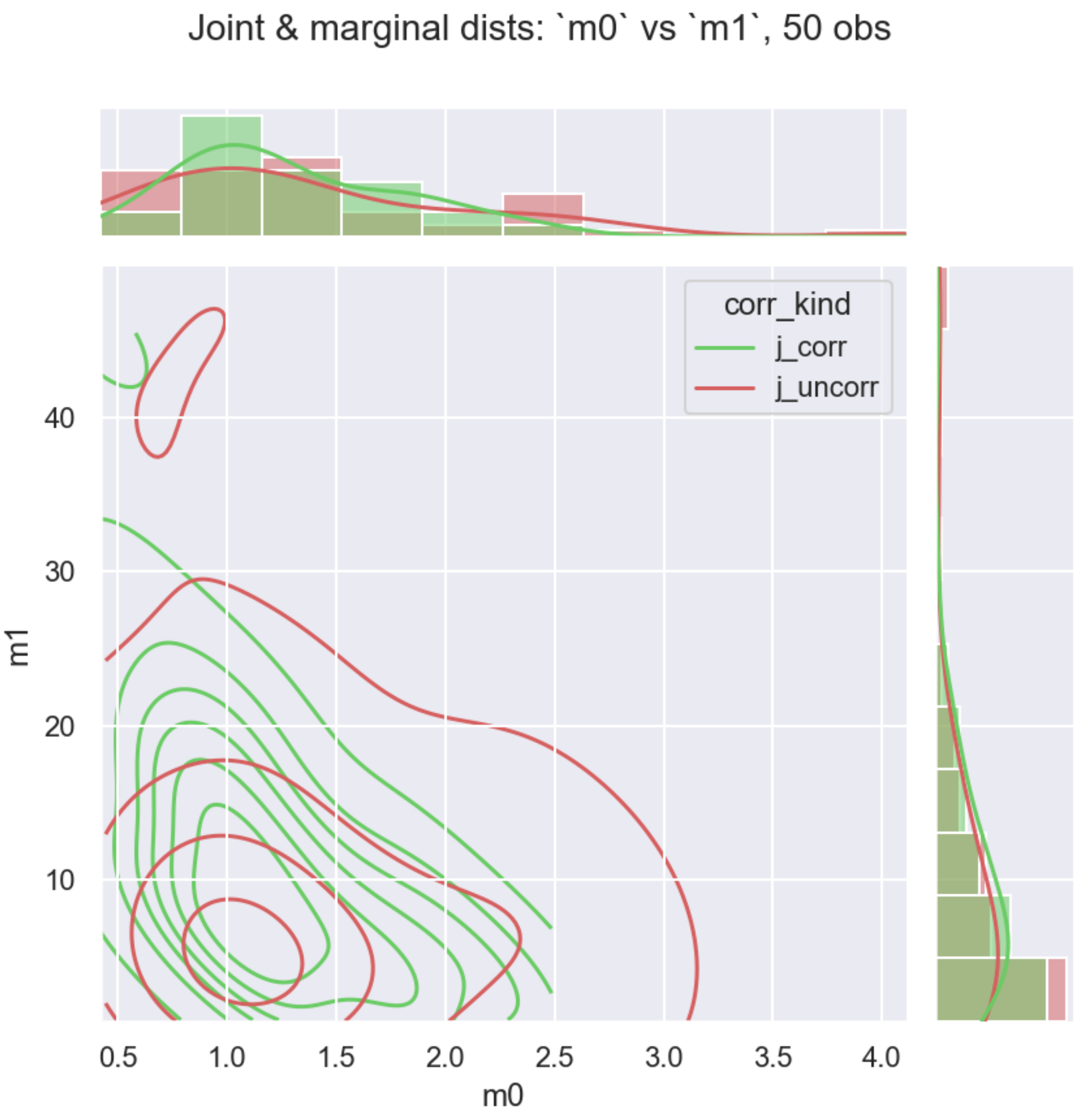


Observe

- Spherical joint distribution, no correlation between our marginals here

4.2.5 1.2.5 Overplot Correlated and Uncorrelated Marginals to highlight differences

```
dfp = pd.concat((df_train[['m0', 'm1']], df_train[['m0x', 'm1x']]\
                 .rename(columns={'m0x':'m0', 'm1x': 'm1'})),
                axis=0, ignore_index=True)
dfp['corr_kind'] = np.repeat(['j_corr', 'j_uncorr'], repeats=len(df_train))
f = eda.plot_joint_numeric(
    data=dfp, ft0='m0', ft1='m1', hue='corr_kind', kind='kde', kdefill=False,␣
↪colori=2,
    subtitle='Observed marginals with / without Copula Correlation')
fqn = figio.write(f, fn='000_jointplot_corr_vs_uncorr')
```



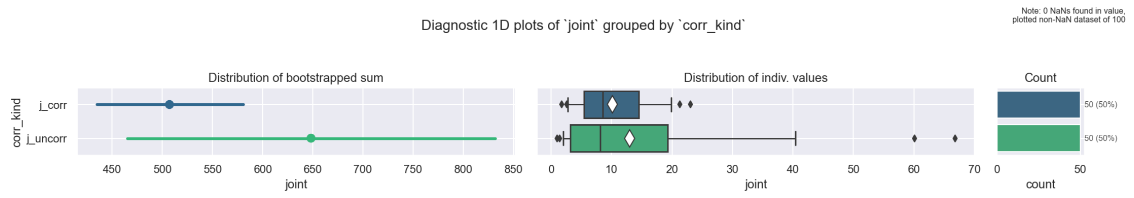
Observe

- The marginals look almost identical
- But the joint distribution is very different: correlated green vs spherical (non-corrolated) red
- This leads to a very different Expected Value

4.3 1.3 Compare the Impact on Joint Product y

If we fail to model the correlation, the impact on the joint product y is substantial, and we might easily under/over estimate an Expected value

```
dfp['joint'] = dfp[['m0', 'm1']].product(axis=1)
pal = sns.color_palette(['C2', 'C3'])
f = eda.plot_smrystat_grp(dfp, grp='corr_kind', val='joint', palette=pal)
fqn = figio.write(f, fn='000_eloss_corr_vs_uncorr')
```

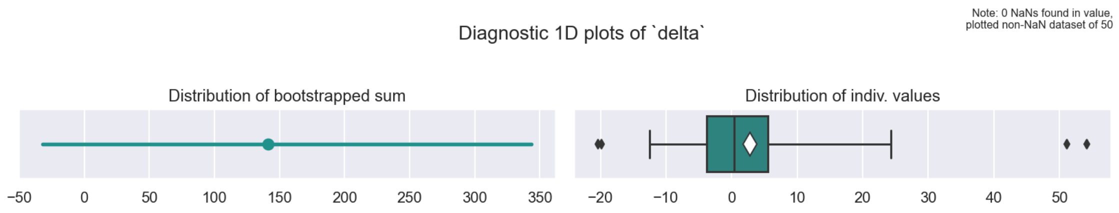


Observe:

- The (bootstrapped) sum of `j_uncorr` ($\mu \approx 650$) is almost always much higher than for `j_corr` ($\mu \approx 500$)
- This shows that even if we estimated each marginal correctly, if our model were to (erroroneously) ignore the coupled covariance between our marginals `m0`, `m1`, we would (erroneously) overestimate the joint distribution total value

View the delta `delta = j_uncorr - j_corr`

```
idx = dfp['corr_kind'] == 'j_corr'
dfpp = pd.DataFrame({'delta': dfp.loc[~idx, 'joint'].values - dfp.loc[idx,␣
  ↪'joint'].values}, index=df_train.index)
g = eda.plot_smrystat(dfpp, val='delta', title_add='joint expected value')
fqn = figio.write(f, fn='000_eloss_delta')
```



Observe:

If we imagine this to be a portfolio of 50 policies, and the value of interest is an Expected Loss Cost $\mathbb{E}_{\text{loss}}$, and the units are dollars, then: + If we were to use a model that ignores covariance, we might get a portfolio estimate of $\mathbb{E}_{loss} \approx 150$ dollars **higher** than if we were to use a better model that handles covariance with a copula function + This overestimate is a substantial $\frac{650}{500} \approx +30\%$ and would likely make the difference between profitable pricing / accurate reserving, or greatly loss-making business over the portfolio.

---

5 2. Extreme Summary: The Impact of Using a Copula Model

Again we note this **Intro** is for verbal presentation and discussion purposes, ideally followed by a deeper technical walkthrough of the project in a long-form style. Because this project is a reference, it contains huge amounts of detail which is not worthwhile to summarise too much.

The interested reader should refer to the project notebooks where we evaluate the behaviour and performance of the models throughout the workflows, including several state-of-the-art methods unavailable to conventional max-likelihood / machine-learning models.

> … but we can highlight a very tangible impact of our results of using a Copula model (`ModelA2`) vs a Naive model (`ModelA0`) in this investigation

5.1 2.1 Quick orientation

Process:

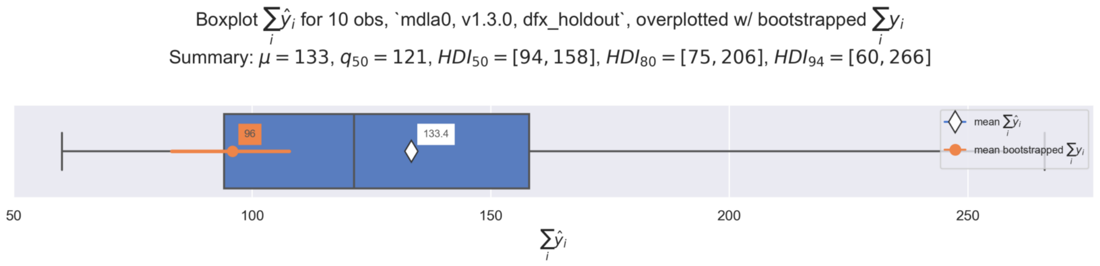
- In this project reference we create a synthetic dataset with 60 observations: these have exogenous values on 2 marginals M_0, M_1
- We create 3 models of increasing sophistication to estimate $\hat{M}_0$, $\hat{M}_1$ and thus the joint product $\hat{y} = \hat{M}_0 \cdot \hat{M}_1$
- The simplest naive model (`ModelA0`) does not include a copula function, and the most sophsticated model `ModelA2` does
- We define a training set of 50 random observations, fit the models, and view the forecasted predictions on a holdout set of 10 observations

Evaluation + We fully evaluate the models in the project notebooks using a variety of sophisticated techniques including In-sample Prior & Posterior Retrodictive ECDF plots, LOO-PIT calculations & plots, and more convential coverage, RSME and R2 calculations. This forecast on the holdout is *not* a formal model evaluation + However for discussion and elucidation we can plot the bootstrapped sum of the actual values $\sum y_{\text{holdout}}$ and compare to the posterior predictions $\sum \hat{y}_{\text{holdout}}$ of the two models

5.2 2.2 Compare Estimated $\hat{y}$ `ModelA0` vs `Model20`

5.2.1 `ModelA0`

```
f = figio.read(fn='100_2_8_4_ppc_holdout_y_boxplot_mdla0_v1_3_0_dfx_holdout.
↪png', figsize=(12, 6))
```

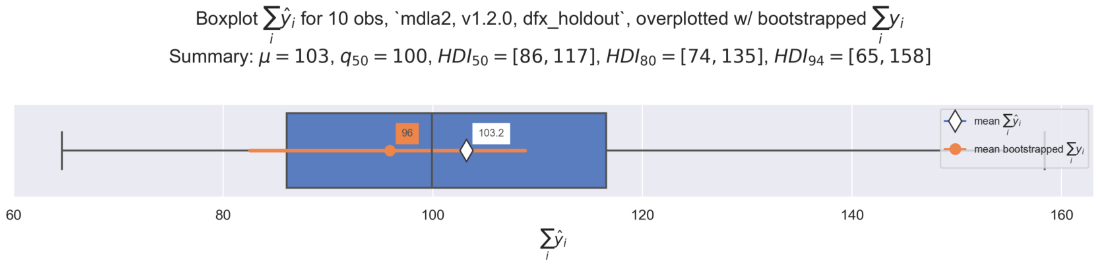


Observe:

- Now we can clearly see the impact: although the in-sample model fit was acceptable, the combined value y is way off, because this model ignores copula correlation between the marginals
- The mean of $\sum_i \hat{y}_i$ is $\mu = 133$, is very different (and sits outside of) the bootstrapped sum of the actual data $\sum_i \hat{y}_i$ which has a mean $\mu = 96$
- Comparing means we have a $\frac{133}{96} \approx 39\%$ overestimate!
- We do see that the PPC distribution envelops the bootstrapped actual data, which is promising, and means the model wouldn't necessarily be wrong to use, but there is clearly room to improve!

5.2.2 `ModelA2`

```
f = figio.read(fn='102_2_8_4_ppc_holdout_y_boxplot_mdla2_v1_2_0_dfx_holdout.
↪png', figsize=(12, 6))
```



Observe:

- Now we can clearly see the impact: the Jacobian adjustment has allowed `mdla2` to estimate a much more precise and accurate value for $\hat{y}$

- The mean of $\sum_i \hat{y}_i$ is $\mu = 103$, and falls within the bootstrapped sum for the actual data $\sum_i \hat{y}_i$ which has a mean $\mu = 96$
- Comparing means, we get $\frac{103}{96} \approx 7\%$ overestimate
- This is substantially better than `mdla0`, and also meaningfully improves on `mdla1`

5.2.3 `ModelA2` vs `ModelA0`

In the above, we see a reduction in the mean overestimate of y from 39% down to 7%: **a 32 percentage point drop**

This is a **huge difference** on this very small and simple dataset, and found only by correctly modelling the covariance using a copula and a sophisticated model architecture

> Now the interested reader should progress through the project notebooks to understand the full detail.

6 Notes

```
%load_ext watermark
%watermark -a "jonathan.sedar@oreum.io" -udtmv -iv
```

```
Author: jonathan.sedar@oreum.io

Last updated: 2024-12-04 18:51:05

Python implementation: CPython
Python version       : 3.11.10
IPython version      : 8.29.0

Compiler    : Clang 17.0.6
OS          : Darwin
Release     : 23.6.0
Machine     : arm64
Processor   : arm
CPU cores   : 8
Architecture: 64bit

sys       : 3.11.10 | packaged by conda-forge | (main, Oct 16 2024, 01:26:25)
[Clang 17.0.6 ]
seaborn   : 0.12.2
oreum_core: 0.9.10
pandas    : 2.2.3
numpy     : 1.26.4
pyprojroot: 0.3.0
```
