King David Concepcion

CAS-05-601P

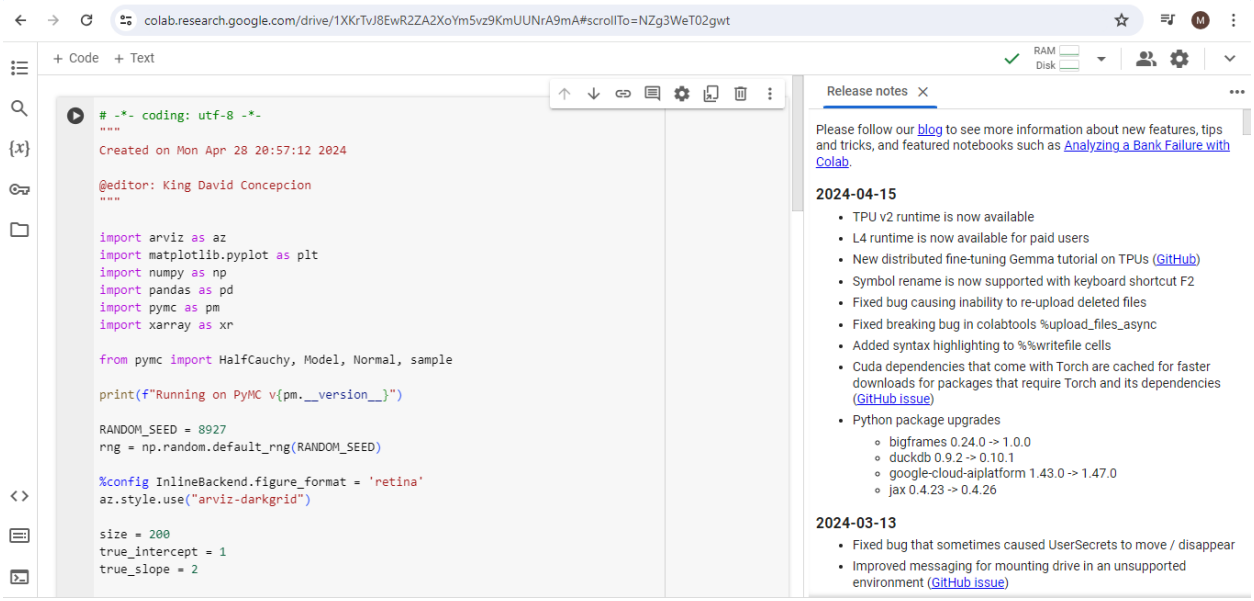
PyMC3 also known as Probabilistic Programming allows for automatic Bayesian inference on user-defined probabilistic models. PyMC is an open source probabilistic programming framework written in Python that uses PyTensor to compute gradients via automatic differentiation, as well as compiling probabilistic programs on-the-fly to one of a suite of computational backends for increased speed. PyMC allows for model specification in Python code, rather than in a domain-specific language, making it easy to learn, customize, and debug. It features next-generation Markov chain Monte Carlo (MCMC) sampling algorithms such as the No-U-Turn Sampler (NUTS; Hoffman, 2014), a self-tuning variant of Hamiltonian Monte Carlo (HMC; Duane, 1987). (Salvatier, J, Wiecki, T.V. and Fonnesbeck, C. 2016)

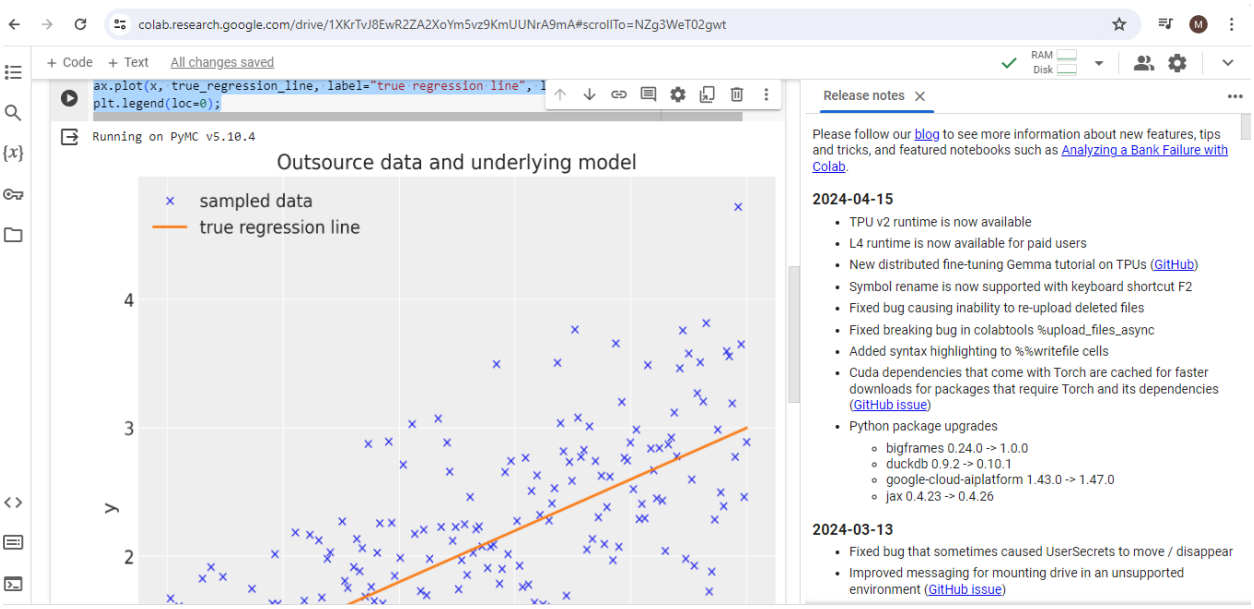
PyMC makes it easy to construct statistical models for the application at hand, independent of how the various fitting algorithms are implemented. (Wiecki, T. 2013). It can also good for fitting a model, making a point estimate prediction, making a point estimate prediction with uncertainty, calculating a posterior predictive distribution. (Ward, 2023). In short Probabilistic Programming is a free and easy tool helpful in predictive models. It can be specially edited and designed by you. Only you can decide which design and model will you choose.

This is the sample script that is outsourced from Github. It needs to download different libraries if you use platforms like Spyder (Anaconda) but it can run by ease with the use of Google Colab.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 28 20:57:12 2024

@editor: King David Concepcion
"""

import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymc as pm
import xarray as xr

from pymc import HalfCauchy, Model, Normal, sample

print(f"Running on PyMC v{pm.__version__}")

RANDOM_SEED = 8927
```

```
rng = np.random.default_rng(RANDOM_SEED)

%config InlineBackend.figure_format = 'retina'
az.style.use("arviz-darkgrid")

size = 200
true_intercept = 1
true_slope = 2

x = np.linspace(0, 1, size)
# y = a + b*x
true_regression_line = true_intercept + true_slope * x
# add noise
y = true_regression_line + rng.normal(scale=0.5, size=size)

data = pd.DataFrame(dict(x=x, y=y))

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(111, xlabel="x", ylabel="y", title="Outsource data and Underlying model")
ax.plot(x, y, "x", label="sampled data")
ax.plot(x, true_regression_line, label="true regression line", lw=2.0)
plt.legend(loc=0);
```





References:

Salvatier, J, Wiecki, T.V. and Fonnesbeck, C. 2016, Probabilistic programming in Python using PyMC3

https://peerj.com/articles/cs-55/

Ward, P. 2023, Bayesian Linear Regression: Getting started with PyMC3

https://optimumsportsperformance.com/blog/bayesian-linear-regression-getting-started-with-pymc3/

Wiecki, T. 2013. The Inference Button: Bayesian GLMs made easy with PyMC.

https://www.pymc.io/projects/docs/en/latest/learn/core_notebooks/GLM_linear.html