2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
ASEE
Paper ID #39726

Automated Analytic Dataset Generation and Assessment for Engineering Analytics Education

Dr. Bruce Wilcox, University of Southern California

Dr. Wilcox is a senior analytics consultant with over 30 years years experience with top-tier consulting firms providing management and information systems consulting services to large corporate and government clients. From 2013 until 2021, he was employed full-time by the SAS Institute, a premier provider of advanced analytics software and consulting services, responsible for consulting with major SAS government clients in California on the use of advanced analytics tools and solutions.

With BS and MS degrees in Electrical Engineering from Carnegie-Mellon University and an MBA from UCLA, Bruce earned his PhD in Engineering and Applied Mathematics from Claremont Graduate University in 2018 with thesis work in time series clustering for fixed income portfolio diversification using model-based clustering and state space analysis techniques. Since January 2020 until taking a full-time faculty position in the 2021 fall semester, Bruce was a part-time lecturer at USC Vitterbi College of Engineering in the MS Analytics program. Prior to that, he taught for twelve semesters as a lecturer at California State University, Long Beach and as a visiting international professor at the National Economics University, Hanoi, Vietnam, teaching two short courses in quantitative analysis to advanced finance students.

Yufan Fei
Jihao LI, University of Southern California
Junqiang Wang
Junmeng Xu, University of Southern California

Automated Analytic Dataset Generation and Assessment for Engineering Analytics Education

Bruce Wilcox, Yufan Fei, Jihao Li, Junqiang Wang, Junmeng Xu
University of Southern California, Los Angeles, CA 90089, USA

Abstract

In recent years, there has been significant growth in analytics programs at the undergraduate and masters' levels in Industrial and Systems Engineering (ISE) departments at universities across the country. When teaching analytics techniques, especially predictive analytics, instructors are always looking for datasets that contain statistical characteristics that we want to discuss including multi-collinearity, interaction effects between variables, skewed distributions, and nonlinear relationships between predictor and response variables. Instructors generally must either search for existing datasets that have these attributes or create them "manually" using programmatic techniques. An academic toolset to permit instructors to specify the statistical properties desired in an analytic, to generate multiple, randomized versions of this dataset (using a newly developed Python library), to provide automation for creating individualized datasets for each student (to avoid inappropriate collaboration on assignments and take-home exams among students), and to provide for automated grading support for assignments and examinations.

This work is supported by a gift from the USC-Meta Center for Research and Education in AI and Learning.

Keywords

Analytics, Dataset Generation, Automated Grading

Introduction and Background

The teaching of predictive analytics techniques involves instruction on different statistical patterns that frequently occur in real-world datasets and algorithmic techniques to model these patterns as accurately as possible. This work addresses three interrelated needs of instructors when preparing lecture examples and assessment exercises:

- Generating datasets that contain desired statistical patterns and relationships as specified in a high-level series of commands.
- Randomizing the datasets so that each student can receive unique datasets for assessment purposes.
- Integrating with a commercial grading platform to manage the distribution, collection, and automatic grading of individualized assessments.

For purposes of this discussion, we use the matrix notation of James, et. al. to describe our datasets [1] with n observations and p predictor variables:

- An $n \times p$ "predictor matrix" $\boldsymbol{X}$ where each column is an $n \times 1$ individual predictor vector $\boldsymbol{X1}$, $\boldsymbol{X2}$, …, $\boldsymbol{Xp}$, where $\boldsymbol{Xi} = [x_{1i}, x_{2i}, \ldots, x_{ni}]^T$ with x_{ij} representing the value of the j^{th} predictor variable for the i^{th} observation.

- An $n \times 1$ "response vector" $Y = [y_1, y_2, \ldots, y_n]^T$ with y_i representing the value of the i^{th} observation of the response variable

- A basic model structure of $Y = f(X) + \varepsilon$ where Y is an $n \times 1$ "response vector", and f is a fixed but unknown function.

For an initial capability, we have identified the following statistical patterns in the predictor variables that we wish to be able to generate:

- Multivariate normal predictors with specified mean vector and covariance matrix.

- Skewed predictor variables specified as normal random variables with mean, variance, and skewness parameters

- Uniform random predictors with specified maximum and minimum values

- Discrete random variables with specified fixed probabilities for each discrete value

We have also identified the following relationships between the predictor matrix and the response vector for automatic generation (all with one-hot-encoding of discrete predictors):

- Linear regression model (with one-hot-encoding of discrete predictors)

- Polynomial regression model

- Linear model with interactive (multiplicative) predictors

- Nonlinear (exponential) relationships between predictors and response

Related Work

A review of this field indicates that there is recent work in "synthetic dataset generation" which focuses on creating new datasets that mimic the distributions of existing real-world datasets to produce larger quantities of training data for machine learning and to anonymize sensitive data but that also includes the generation of datasets from theoretical distributions. An overview of this topic is presented in the books Practical Synthetic Data Generation [2] and Synthetic Data for Deep Learning [3]

There are libraries in the Python scikit-learn package [4] for generating classification and regression synthetic datasets (sklearn.dataset.make_regression and

sklearn.dataset.make_classification), but they do not allow the specification of an underlying deterministic function. It has been suggested that combining these functions with the capabilities of another Python package for symbolic mathematics (SymPy) could form the basis for the functionality desired for this project.

Significant academic research on the topic of synthetic data generation was done by the Synthetic Data Vault project within the MIT Data to AI Lab (https://dai.lids.mit.edu/) which was recently transferred to a private company, datacebo (https://datacebo.com/). Their core product is described in the paper *The Synthetic Data Vault*. [4]

Dataset Specification System

We have developed an initial capability in the form of a Python library named analyticsdf built "on top" of the Pandas package. Full documentation is located in the github repository https://faye-yufan.github.io/analytics-dataset/ and is summarized below.

The *analyticsdf* class implements a single object called AnalyticsDataframe which consists of two data structures, *predictor_matrix*, a Pandas dataframe that represents the predictors ($\boldsymbol{X}$) and *response_vector*, a Pandas series that represents the response variable ($\boldsymbol{Y}$).

The class is instantiated with the method AnalyticsDataframe that is called with two mandatory parameters representing the number of observations ($\boldsymbol{n}$), the number of predictor variables ($\boldsymbol{p}$), and optional parameters *predictor_names* and *response_vector_names*, and *seed* which permits the optional specification of a seed to be used for all randomized methods using this object. The seed variable can be used to control the generation of individualized analytic datasets for distribution to students.

This method returns an AnalyticsDataframe class object which can be queried with the two attributes *predictor_matrix* and *response_vector* implemented as a Pandas dataframe and series respectively. Both of these Pandas structures are originally populated with NaN values.

The methods that have been implemented to date for the purpose of defining the statistical patterns in the predictor matrix are summarized below:

- *update_predictor_normal*: generates a continuous predictor matrix using a multivariate normal distribution based on a specified mean vector and covariance matrix. This predictor matrix can be used to populate any combination of variables in the predictor_matrix object.

- *update_predictor_uniform*: generates a continuous predictor vector using a uniform distribution based on a specified min and max that can be used to populate any specified variable in the predictor_matrix object.

- *update_predictor_beta*: generates a continuous predictor vector using a beta distribution based on a specified alpha and beta that can be used to populate any specified variable in the predictor_matrix object.

- *update_predictor_categorical*: generates a categorical predictor vector based on specified category frequencies that can be used to populate any specified variable in the predictor_matrix object.

The methods that have been implemented to date for the purpose of specifying the relationship between the predictor matrix and the response variable are summarized below:

- *generate_response_vector_linear*: the basic method that allows generation of a response vector based on a linear regression generative model with specifications for the model coefficients and the variance of the error term.

- *generate_response_vector_polynomial*: a more advanced method that allows for specification of polynomial models with interaction effects between predictors and nonlinear relationships between the predictors and the response.

The following two examples illustrate the use of these methods to create analytic datasets:

Example 1:

- Predictor matrix has 1000 observations and six attributes (X1, X2, X3, X4, X5, X6)

- All except X6 are uniformly distributed with a min of 0 and a max of 100

- X6 is a categorical variable with values of Red (30%), Yellow (40%), or Blue (30%)

- The response variable has the relationship:

$$\boldsymbol{Y = 15X1 - 30X2 + 2X2^2 + 5X4 - 2000I(X6 = Red) + 1000\, I(X6 = Blue) + \varepsilon,\ \varepsilon \sim N(0, 2000)}$$

The dataset specification code to generate this dataset is provided in the figure below:

```
# example 1
ad = AnalyticsDataframe(1000, 6)
for var in ['X1', 'X2', 'X3', 'X4', 'X5']:
    ad.update_predictor_uniform(var, 0, 100)
ad.update_predictor_categorical('X6', ["Red", "Yellow", "Blue"], [0.3, 0.4, 0.3])

predictor_name_list = ['X1', 'X2', 'X4']
polynomial_order = [1, 2, 1]
beta = [0, 15, -30, 2, 5]
int_matrix = np.array([
    [0,0,0,0],
    [0,0,0,0],
    [0,0,0,0],
    [0,0,0,0]])
eps_var = 2000

ad.generate_response_vector_polynomial(
            predictor_name_list = predictor_name_list,
            polynomial_order = polynomial_order,
            beta = beta,
            interaction_term_betas = int_matrix,
            epsilon_variance = eps_var)
ad.update_response_poly_categorical(
    predictor_name='X6',
    betas={'Red': -2000, 'Blue': 1000}
)
```

Figure 1: AnalyticsDataframe Code to Generate Example 1 Dataset

A dataset visualization routine has been developed to provide the educator with visual feedback regarding the dataset that has been specified. Sample outputs for example 1 are shown below:

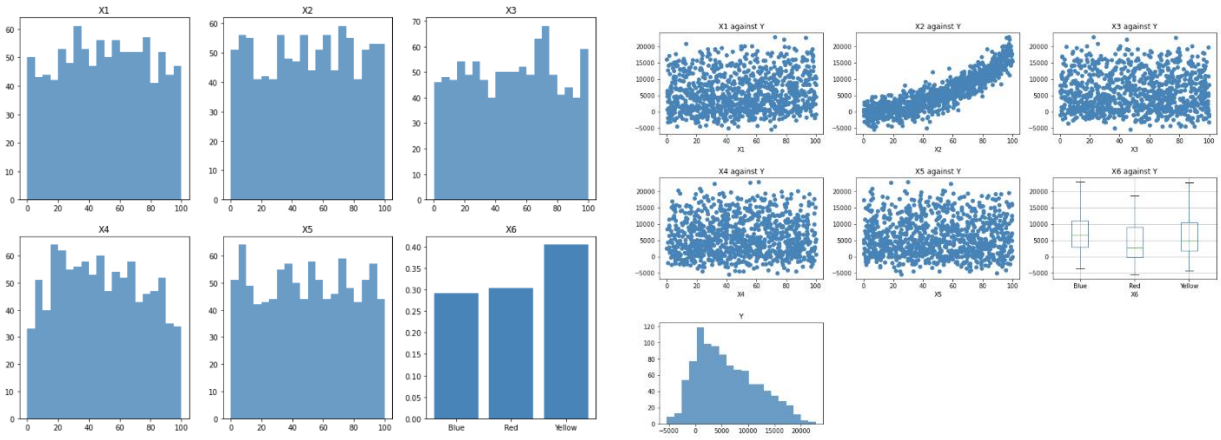

Figure 2: Univariate and Bivariate Visualizations of Example 1 Dataset

Example 2:

- Predictor matrix has 1000 observations and six attributes (X1, X2, X3, X4, X5, X6)

- X1 through X3 are correlated normal random variables with a mean vector of [100, 80, 120] and a covariance matrix of $\begin{bmatrix} \mathbf{10} & \mathbf{9} & \mathbf{4} \\ \mathbf{9} & \mathbf{10} & \mathbf{6} \\ \mathbf{4} & \mathbf{6} & \mathbf{20} \end{bmatrix}$
- X5 is uniformly distributed with a min of 0 and a max of 100
- X6 is a categorical variable with values of Red (30%), Yellow (40%), or Blue (30%)
- X4 is dependent on X1 and X3 according to the formula (creating a multicollinearity)

$$\boldsymbol{X4 = X1 + 1.5X2 + \ \varepsilon,\ \varepsilon \sim N(0, 20)}$$

- The response variable has the relationship:

$$\boldsymbol{Y = \exp\left(0.001 \ * (100 + \beta X1 + 1.5X3 + \ \varepsilon,\ \varepsilon \sim N(0, 20)\right.}$$

Where $\boldsymbol{\beta = 5\ if\ X6 = Red\ and\ \beta = 1\ otherwise}$

The dataset specification code to generate this dataset is provided in the figure below:

```
# example 2
ad = AnalyticsDataframe(1000, 6)
covariance_matrix = np.array([[10, 9, 4],
                              [9, 10, 6],
                              [4, 6, 20]])
ad.update_predictor_normal(predictor_name_list=["X1", "X2", "X3"],
                           mean=[100, 80, 120],
                           covariance_matrix=covariance_matrix)
ad.update_predictor_uniform('X5', 0, 100)
ad.update_predictor_categorical('X6', ["Red", "Yellow", "Blue"], [0.3, 0.4, 0.3])

ad.update_predictor_multicollinear(target_predictor_name = 'X4', dependent_predictors_list = ['X1', 'X2', 'X3'], beta=[0, 1, 1.5, 0], epsilon_variance=20)

# create a new predictor column, change categorical value into numerical value
categorical_mapping = {'Red': 5, 'Yellow': 1, 'Blue': 1}
ad.predictor_matrix["X6_weight"] = ad.predictor_matrix.replace({'X6': categorical_mapping}, inplace=False)['X6']
print(ad.predictor_matrix)
predictor_name_list = ['X1', 'X3', 'X6_weight']
polynomial_order = [1, 1, 1]
beta = [100, 0, 1.5, 0]
int_matrix = np.array([
    [0,0,0],
    [0,0,0],
    [1,0,0]])
eps_var = 10

ad.generate_response_vector_polynomial(
            predictor_name_list = predictor_name_list,
            polynomial_order = polynomial_order,
            beta = beta,
            interaction_term_betas = int_matrix,
            epsilon_variance = eps_var)
ad.response_vector = np.exp(0.001 * ad.response_vector)
```

Figure 3: AnalyticsDataframe Code to Generate Example 1 Dataset

Auto-grading for Individualized Datasets

Practical implementation of individualized datasets for homework and exams requires integration into an auto-grading framework. After review of candidate frameworks, we have selected the Gradescope system from the company Turnitin. [5] Gradescope provides a highly configurable and flexible platform for auto-grading programming assignments that is integrated into its popular grading system (www.gradescope.com).

Before describing the architecture for integrating our automated dataset generation capability into the Gradescope platform, we first present at a very high level the architecture of the Gradescope platform when used to auto-grade Python programming assignments in Figure 1 and described below:

- The teaching staff creates the assessment with three components: the assignment in the form of a Jupyter Notebook template that includes instructions for the student on the naming of the gradable result objects (can be a simple numeric variable or a complex object such as an Sklearn predictive model object), a dataset to use for the assignment, and an "assignment autograder".

- The assignment autograder is a zip file that includes three files: a file that specifies the Python libraries that must be loaded (*assignment.txt*), a data folder containing all required datasets, and a Jupyter notebook file that contains the Python code that implements the autograding algorithms (*assignment.ipynb*).

- The student completes the assignment and submits the solutions as a completed version of the Jupyter notebook in accordance with the assignment template.

- When ready to grade, the autograder is executed by the Gradescope platform and evaluates each submission in accordance with the logic programmed by the teaching staff. The grading results are passed to the Gradescope grading database via a JSON file named *results.json* and in the format specified at https://gradescope-autograders.readthedocs.io/en/latest/specs/#output-format.

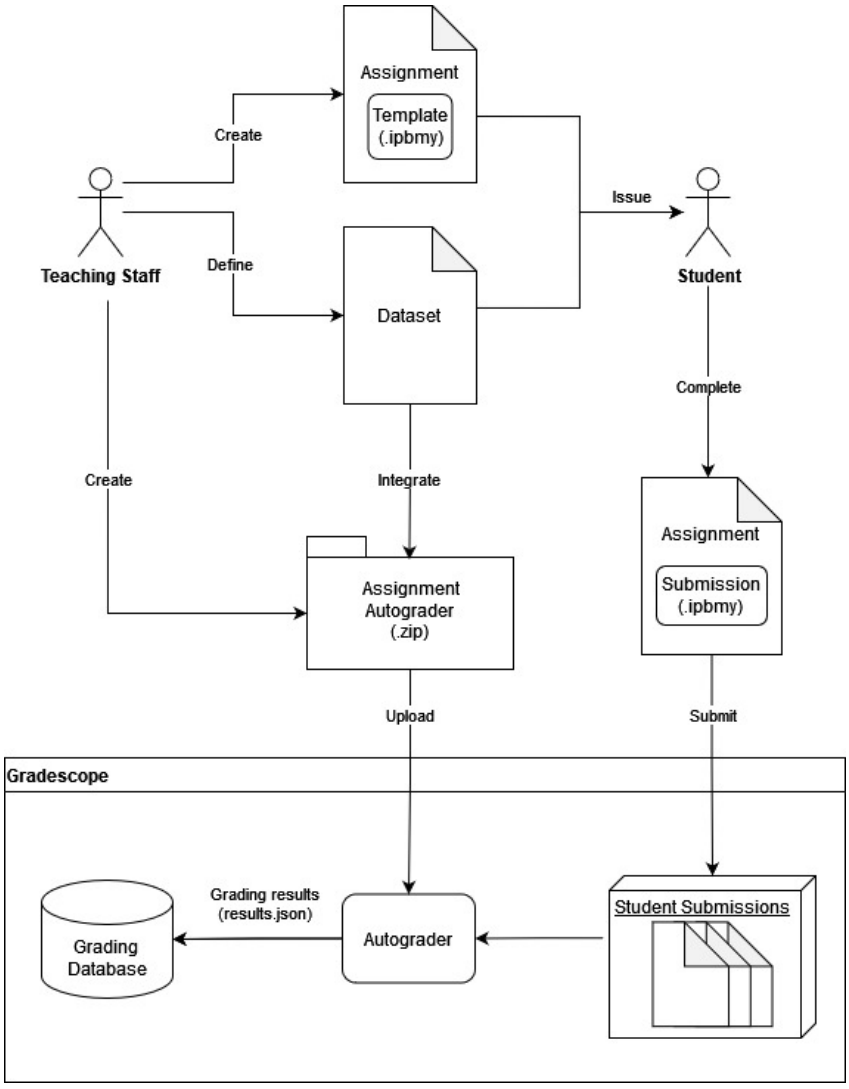

Figure 4: Gradescope Auto-grading Architecture (Python)

The architecture of the integration of our automated dataset generation objects into the Gradescope auto-grading platform is depicted in Figure 2 and described below:

- As with basic Gradescope functionality, the teaching staff creates the assessment with three components, but instead of a dataset the assessment package consists of a dataset specification that is coded using the Analyticsdf dataset specification system.

- The analytic dataset specification is encrypted and inserted into the assignment template along with code the prompts the student to enter their student ID number which then generates the dataset for use on the problem by making the appropriate calls to the Analyticsdf dataset specification system.

- The autograder then queries the Gradescope submission metadata to retrieve the student ID for the submission being graded by importing and reading the JSON file (see

https://gradescope-autograders.readthedocs.io/en/latest/submission_metadata/ for documentation of the JSON format).

- The autograder re-generates the dataset that was used by each specific student and completes the built-in Gradescope auto-grading process described above.

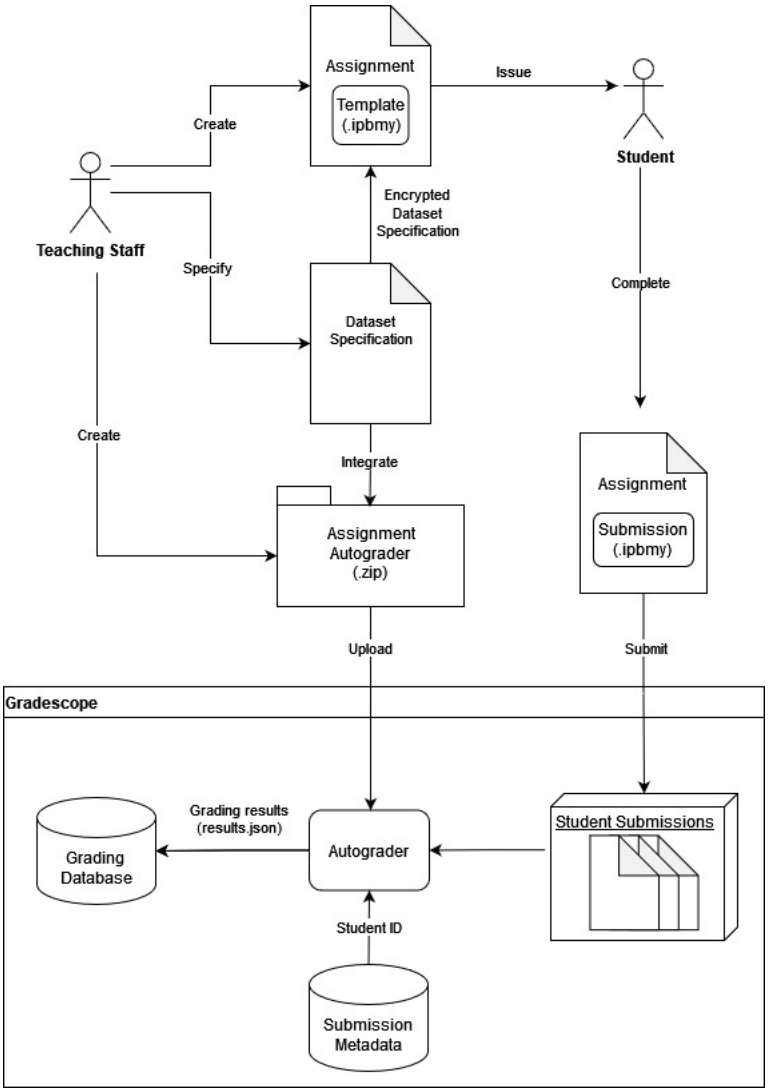

Figure 5: Auto-Grading Architecture with Individualized Datasets

Next Steps

The primary near-term objective of the team is to complete the functionality to provide autograding of individualized datasets and test it with a beta test team consisting of student volunteers. Assuming successful completion of that testing, we plan to implement it "live" with a predictive analytics class in the fall semester of 2023 and then make it broadly available for use by other instructors.

A major planned enhancement to the generation of individualized datasets involves expanding the randomization beyond the residuals term to the randomization of the generative model coefficients.

Also planned is expanded dataset functionality to include categorical response vectors, generalized linear models (primarily Poisson regression), and the incorporation of additional correlated predictor matrices (beyond the current multivariate normal) utilizing copula functions.

Finally, we would like to make the use of this functionality more accessible to other instructors. The current Gradescope autograding architecture allows complete flexibility in structuring autograding rubrics, but with that flexibility comes a fair amount of complexity that may be daunting for many potential users. We have ideas for GUI-based dataset specification and autograding rubric setup that we hope to explore in the coming months.

References

[1] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An Introduction to Statistical Learning with Applications in R Second Edition," 2021.

[2] Khaled El Emam, Richard Hoptroff, and Lucy Mosquera, *Practical Synthetic Data Generation*. O'Reilly Media, Inc, 2020.

[3] N. Gürsakal, S. Çelik, and E. Birişçi, *Synthetic Data for Deep Learning*. Berkeley, CA: Apress L. P, 2022.

[4] N. Patki, R. Wedge, and K. Veeramachaneni, "The Synthetic Data Vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2016, pp. 399–410. doi: 10.1109/DSAA.2016.49.

[5] A. Singh, S. Karayev, K. Gutowski, and P. Abbeel, "Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work," in *L@S 2017 - Proceedings of the 4th (2017) ACM Conference on Learning at Scale*, 2017. doi: 10.1145/3051457.3051466.