Dataset Generator Configuration Documentation

Overview

This document describes the JSON configuration format for generating synthetic datasets for data science instruction.

Root Structure

```json
{
  "dataset_config": {
    "name": "string",
    "description": "string",
    "random_seed": integer,
    "n_rows": integer,
    "correlations": [...],
    "features": [...],
    "target": {...}
  }
}
```

Root Fields

Field	Type	Required	Description
`name`	string	Yes	Unique identifier for the dataset

Field	Type	Required	Description
`description`	string	No	Human-readable description
`random_seed`	integer	No	Seed for reproducibility (omit for random)
`n_rows`	integer	Yes	Number of observations to generate

Features Array

Each feature is defined as an object with the following structure:

```json
{
  "name": "string",
  "description": "string",
  "data_type": "float|int|categorical",
  "distribution": {...},
  "missing_rate": 0.0,
  "outlier_rate": 0.0,
  "outlier_method": "string",
  "outlier_multiplier": float
}
```

Feature Fields

Field	Type	Required	Description
`name`	string	Yes	Variable name (valid Python identifier)
`description`	string	No	Human-readable description
`data_type`	string	Yes	One of: `float`, `int`, `categorical`
`distribution`	object	Yes	Distribution specification (see below)
`missing_rate`	float	No	Proportion of missing values (0.0-1.0), default: 0.0
`outlier_rate`	float	No	Proportion of outliers (0.0-1.0), default: 0.0

Field	Type	Required	Description
`outlier_method`	string	No	One of: `extreme_high`, `extreme_low`, `extreme_both`
`outlier_multiplier`	float	No	Multiplier for outlier generation, default: 3.0

Distribution Types

1. Uniform Distribution

Generates values uniformly between min and max.

```json
{
  "type": "uniform",
  "min": 0.0,
  "max": 1.0
}
```

Example:

```json
{
  "name": "random_value",
  "data_type": "float",
  "distribution": {
    "type": "uniform",
    "min": 0,
    "max": 100
  }
}
```

2. Normal Distribution

Generates values from a normal (Gaussian) distribution.

```json
{
  "type": "normal",
  "mean": 0.0,
  "std": 1.0,
  "min_clip": null,
  "max_clip": null
}
```

Parameters: - `mean`: Center of distribution - `std`: Standard deviation - `min_clip`: Optional minimum value (clips lower values) - `max_clip`: Optional maximum value (clips upper values)

Example:

```json
{
  "name": "test_score",
  "data_type": "float",
  "distribution": {
    "type": "normal",
    "mean": 75,
    "std": 10,
    "min_clip": 0,
    "max_clip": 100
  }
}
```

3. Weibull Distribution

Generates values from a 3-parameter Weibull distribution (useful for skewed data).

```json
{
  "type": "weibull",
  "shape": 1.5,
  "scale": 1.0,
  "location": 0.0
}
```

Parameters: - `shape`: Shape parameter (k) - controls skewness - `scale`: Scale parameter ( ) - stretches/compresses - `location`: Location parameter - shifts distribution

Example:

```json
{
  "name": "customer_lifetime",
  "data_type": "int",
  "distribution": {
    "type": "weibull",
    "shape": 1.2,
    "scale": 24,
    "location": 1
  }
}
```

4. Random Walk

Generates values that evolve randomly from a starting point.

```json
{
  "type": "random_walk",
  "start": 100.0,
  "step_size": 1.0,
  "drift": 0.0
}
```

Parameters: - `start`: Initial value - `step_size`: Maximum step size per observation - `drift`: Directional bias per step (positive = upward trend)

Example:

```json
{
  "name": "stock_price",
  "data_type": "float",
  "distribution": {
    "type": "random_walk",
    "start": 100.0,
    "step_size": 2.5,
    "drift": 0.1
  }
}
```

5. Sequential

Generates sequential integers (useful for IDs).

```json
{
  "type": "sequential",
  "start": 1,
  "step": 1
}
```

Example:

```json
{
  "name": "customer_id",
  "data_type": "int",
  "distribution": {
    "type": "sequential",
    "start": 1000,
    "step": 1
  }
}
```

Categorical Variables

For `data_type: "categorical"`, a `categories` array must be provided with exactly 10 labels (one per decile).

```json
{
  "name": "risk_level",
  "data_type": "categorical",
  "distribution": {
    "type": "normal",
    "mean": 0.5,
    "std": 0.2
  },
  "categories": [
    "Very Low",
    "Very Low",
    "Low",
    "Low",
    "Medium",
    "Medium",
```

```
    "Medium",
    "High",
    "High",
    "Very High"
  ]
}
```

Process: 1. Generate continuous values using specified distribution 2. Rank values and divide into 10 deciles (0-9) 3. Map each decile to corresponding category label 4. Categories array position 0 = 1st decile (lowest 10%), position 9 = 10th decile (highest 10%)

Creating Imbalanced Classes: Repeat labels to create imbalanced distributions:

```
"categories": [
  "Rare Event",      // Decile 0 (10%)
  "Common",          // Decile 1 (10%)
  "Common",          // Decile 2 (10%)
  "Common",          // Decile 3 (10%)
  "Common",          // Decile 4 (10%)
  "Common",          // Decile 5 (10%)
  "Common",          // Decile 6 (10%)
  "Common",          // Decile 7 (10%)
  "Common",          // Decile 8 (10%)
  "Common"           // Decile 9 (10%)
]
// Results in 10% "Rare Event", 90% "Common"
```

Correlations

Define pairwise correlations between features.

```
{
  "correlations": [
    {
      "variables": ["var1", "var2"],
      "correlation": 0.75,
      "method": "cholesky"
    }
  ]
}
```

Fields: - `variables`: Array of exactly 2 feature names - `correlation`: Correlation coefficient (-1.0 to 1.0) - `method`: Always use `"cholesky"` (Cholesky decomposition)

Example:

```json
{
  "correlations": [
    {
      "variables": ["height", "weight"],
      "correlation": 0.80,
      "method": "cholesky"
    },
    {
      "variables": ["income", "education_years"],
      "correlation": 0.65,
      "method": "cholesky"
    }
  ]
}
```

Important Notes: - Correlations are applied to continuous values before categorical conversion - All correlated variables must exist in features array - Correlation matrix must be positive semi-definite (valid correlation structure)

Missing Data

Specify the proportion of missing values for each feature.

```json
{
  "name": "income",
  "data_type": "float",
  "distribution": {...},
  "missing_rate": 0.15
}
```

Process: 1. Generate complete data 2. Randomly select `missing_rate` × `n_rows` observations 3. Replace with `NaN` (float), `None` (int), or empty string (categorical)

Example Use Cases: - Survey data: 5-20% missing - Administrative data: 1-5% missing - Complete data: 0%

Outliers

Inject outliers into numeric features to simulate real-world anomalies.

```json
{
  "name": "transaction_amount",
  "data_type": "float",
  "distribution": {...},
  "outlier_rate": 0.02,
  "outlier_method": "extreme_high",
  "outlier_multiplier": 3.0
}
```

Outlier Methods:

`extreme_high`

Replace outliers with high extreme values. - Formula: `value = Q3 + multiplier × IQR`

`extreme_low`

Replace outliers with low extreme values. - Formula: `value = Q1 - multiplier × IQR`

`extreme_both`

Replace outliers with both high and low extremes (50/50 split). - High: `Q3 + multiplier × IQR` - Low: `Q1 - multiplier × IQR`

Where: - Q1 = 25th percentile - Q3 = 75th percentile - IQR = Q3 - Q1 (Interquartile Range)

Example:

```json
{
  "name": "response_time",
  "data_type": "float",
  "distribution": {
    "type": "normal",
    "mean": 200,
    "std": 50
  },
```

```
  "outlier_rate": 0.05,
  "outlier_method": "extreme_both",
  "outlier_multiplier": 4.0
}
```

Target Variable

Define the target variable using a Python expression based on features.

```
{
  "target": {
    "name": "string",
    "description": "string",
    "data_type": "float|int|categorical",
    "expression": "python expression",
    "noise_percent": float,
    "categories": [...],
    "missing_rate": 0.0,
    "outlier_rate": 0.0,
    "outlier_method": "string",
    "outlier_multiplier": float
  }
}
```

Target Fields

Field	Type	Required	Description
`name`	string	Yes	Target variable name
`description`	string	No	Human-readable description
`data_type`	string	Yes	One of: `float`, `int`, `categorical`
`expression`	string	Yes	Python expression using feature names
`noise_percent`	float	No	Percentage noise (0-100), default: 0
`categories`	array	Conditional	Required if `data_type: "categorical"` (10 labels)

Field	Type	Required	Description
`missing_rate`	float	No	Proportion missing (0.0-1.0)
`outlier_rate`	float	No	Proportion outliers (0.0-1.0)

Expression Syntax

Available: - Feature names as variables - Arithmetic operators: `+`, `-`, `*`, `/`, `**` (power), `//` (floor division), `%` (modulo) - NumPy functions: `np.exp()`, `np.log()`, `np.sqrt()`, `np.sin()`, `np.cos()`, `np.abs()`, etc. - Parentheses for grouping

Examples:

Linear Regression

```json
{
  "name": "price",
  "data_type": "float",
  "expression": "50000 + 3000*bedrooms + 2500*bathrooms + 100*sqft",
  "noise_percent": 5.0
}
```

Polynomial Regression

```json
{
  "name": "yield",
  "data_type": "float",
  "expression": "10 + 2*fertilizer - 0.1*fertilizer**2 + 0.5*rainfall",
  "noise_percent": 10.0
}
```

Logistic (Binary Classification)

```json
{
  "name": "approved",
  "data_type": "categorical",
  "expression": "1 / (1 + np.exp(-(-5 + 0.1*credit_score + 2*income_k - 1.5*debt_ratio)))",
  "noise_percent": 3.0,
  "categories": [
    "Denied", "Denied", "Denied", "Denied", "Denied",
```

```
    "Approved", "Approved", "Approved", "Approved", "Approved"
  ]
}
```

Integer Target

```
{
  "name": "count",
  "data_type": "int",
  "expression": "10 + 2.5*advertising_spend + 1.8*seasonality",
  "noise_percent": 8.0
}
```

Noise Application

Noise is applied as a percentage of the calculated value's range:

1. Calculate target values from expression
2. Compute range: `max_value - min_value`
3. For each observation, add random noise: `±(noise_percent/100) × range × random()`

Example: - Expression yields values from 50 to 150 (range = 100) - `noise_percent: 10.0` - Each value gets ±10 added randomly (10% of 100)

Complete Examples

Example 1: Simple Linear Regression

```
{
  "dataset_config": {
    "name": "simple_regression",
    "description": "Teaching simple linear regression",
    "random_seed": 123,
    "n_rows": 500,
    "features": [
      {
        "name": "study_hours",
```

```json
        "description": "Hours spent studying",
        "data_type": "float",
        "distribution": {
          "type": "uniform",
          "min": 0,
          "max": 10
        },
        "missing_rate": 0.0
      }
    ],
    "target": {
      "name": "test_score",
      "description": "Test score out of 100",
      "data_type": "float",
      "expression": "50 + 5*study_hours",
      "noise_percent": 10.0
    }
  }
}
```

Example 2: Binary Classification

```json
{
  "dataset_config": {
    "name": "loan_approval",
    "description": "Binary classification for loan approval",
    "random_seed": 456,
    "n_rows": 1000,
    "correlations": [
      {
        "variables": ["income", "credit_score"],
        "correlation": 0.60,
        "method": "cholesky"
      }
    ],
    "features": [
      {
        "name": "income",
        "description": "Annual income in thousands",
        "data_type": "float",
```

```
        "distribution": {
          "type": "normal",
          "mean": 60,
          "std": 20,
          "min_clip": 20,
          "max_clip": 150
        },
        "missing_rate": 0.05
      },
      {
        "name": "credit_score",
        "description": "Credit score 300-850",
        "data_type": "int",
        "distribution": {
          "type": "normal",
          "mean": 680,
          "std": 80,
          "min_clip": 300,
          "max_clip": 850
        },
        "missing_rate": 0.02
      },
      {
        "name": "debt_ratio",
        "description": "Debt to income ratio",
        "data_type": "float",
        "distribution": {
          "type": "uniform",
          "min": 0.1,
          "max": 0.6
        },
        "missing_rate": 0.03,
        "outlier_rate": 0.02,
        "outlier_method": "extreme_high",
        "outlier_multiplier": 2.5
      }
    ],
    "target": {
      "name": "approved",
      "description": "Loan approval decision",
      "data_type": "categorical",
      "expression": "1 / (1 + np.exp(-(-8 + 0.05*credit_score + 0.08*income - 10*debt_ratio)
```

```
      "noise_percent": 5.0,
      "categories": [
        "Rejected", "Rejected", "Rejected", "Rejected",
        "Rejected", "Rejected",
        "Approved", "Approved", "Approved", "Approved"
      ]
    }
  }
}
```

Example 3: Time Series with Random Walk

```
{
  "dataset_config": {
    "name": "stock_prediction",
    "description": "Stock price prediction with trend",
    "random_seed": 789,
    "n_rows": 365,
    "features": [
      {
        "name": "day",
        "description": "Trading day",
        "data_type": "int",
        "distribution": {
          "type": "sequential",
          "start": 1,
          "step": 1
        }
      },
      {
        "name": "price",
        "description": "Stock price",
        "data_type": "float",
        "distribution": {
          "type": "random_walk",
          "start": 100.0,
          "step_size": 3.0,
          "drift": 0.05
        },
        "outlier_rate": 0.03,
```

```
        "outlier_method": "extreme_both",
        "outlier_multiplier": 3.0
      },
      {
        "name": "volume",
        "description": "Trading volume",
        "data_type": "int",
        "distribution": {
          "type": "weibull",
          "shape": 2.0,
          "scale": 1000000,
          "location": 500000
        }
      }
    ],
    "target": {
      "name": "next_day_price",
      "description": "Next day predicted price",
      "data_type": "float",
      "expression": "price * 1.001 + 0.0000001*volume",
      "noise_percent": 2.0
    }
  }
}
```

Validation Rules

The generator will validate:

1. **Feature names** are valid Python identifiers and unique
2. **Data types** match distribution compatibility
3. **Correlation variables** reference existing features
4. **Expression** references only defined feature names
5. **Categorical** features have exactly 10 category labels
6. **Rates** (missing, outlier, noise) are between 0 and 1 (or 0-100 for noise_percent)
7. **Distribution parameters** are valid (e.g., std > 0, min < max)
8. **Correlation matrix** is positive semi-definite

Notes

- **Order matters**: Features are generated in order. Random walks and sequential distributions depend on order.
- **Correlations**: Applied before categorical conversion and outlier injection.
- **Missing data**: Applied after all other transformations.
- **Categorical deciles**: Always create 10 equal-sized bins (10% each).
- **NumPy**: Available in expressions as `np.*`