Data Reproduction Guide - From OpenML to Processed Datasets

Purpose: This document describes how to reproduce the 72 processed datasets from raw OpenML downloads.

Note: If you already have the processed datasets (via Google Drive), you do NOT need this guide. Use `COMPLETE_REPRODUCTION_GUIDE.md` instead.

Random Seed: All transformations use `RANDOM_STATE = 42` for reproducibility.

Table of Contents

1. Overview
2. Requirements
3. Phase 1: Download Data from OpenML
4. Phase 2: Identify Required Transformations
5. Phase 3: Manual Dataset Transformation
6. Phase 4: Organize and Verify Datasets
7. Phase 5: Standardize Target Variables
8. Dataset Reference Table
9. Troubleshooting

Overview

Pipeline Summary

```
OpenML (72 datasets)
        |
        v
download_exact_datasets.py --> data/raw_downloads/
        |
        v
compare_raw_vs_processed.py --> transformation_mapping.csv
        |
        v
data/dataset_editor.py --> Apply transformations (INTERACTIVE)
        |
        v
Manual organization --> data/processed_classification/
                        data/processed_regression/
        |
        v
verify_datasets.py --> Validation report
        |
        v
standardize_target_variables.py --> TARGET_VARIABLE column
        |
        v
READY FOR EXPERIMENTS
```

What Gets Produced

- **72 datasets** total (36 classification + 36 regression)
- **9 dimensionality categories**: smallxsmall, smallxmed, smallxlarge, medxsmall, medxmed, medxlarge, largexsmall, largexmed, largexlarge
- **4 variants per dimensionality**:
  - Classification: Balanced/Imbalanced x Clean/Noisy
  - Regression: Linear/Non-linear x Clean/Noisy

Size Constraints

Category	Observations	Features
Small	<=1,000	<=20
Medium	<=5,000	21-100
Large	<=15,000	>100

Requirements

Python Packages

```
pip install numpy pandas scikit-learn openml imbalanced-learn tkinter
```

Scripts Needed

- `download_exact_datasets.py` - Downloads exact 72 datasets from OpenML
- `compare_raw_vs_processed.py` - Identifies transformations needed
- `data/dataset_editor.py` - Interactive transformation tool (GUI)
- `verify_datasets.py` - Validates final datasets
- `standardize_target_variables.py` - Standardizes target column names

Phase 1: Download Data from OpenML

Script: `download_exact_datasets.py`

```
#download_exact_datasets.py
import openml
import pandas as pd
import os
import argparse
from pathlib import Path

# Exact 72 OpenML IDs used in experiments
CLASSIFICATION_IDS = [
    16, 18, 143, 180, 279, 375, 727, 728, 807, 958, 977, 1022, 1042, 1217,
1242, 1425,
    1475, 1509, 1537, 1548, 1587, 1590, 23512, 41767, 42680, 44130, 44563,
45078, 46456,
    46526, 46561, 46609, 46633, 46805, 46856, 46874
]

REGRESSION_IDS = [
    189, 197, 344, 422, 572, 574, 1197, 1201, 1203, 1592, 41187, 41211,
41506, 41980,
    42570, 42571, 42635, 42705, 42729, 42737, 43090, 43144, 43873, 43988,
44002, 44057,
    44060, 44061, 44067, 44977, 46161, 46293, 46618, 46620, 46787, 46898
]

def download_dataset(dataset_id, task_type, output_dir):
    """
    Download a single OpenML dataset.

    Args:
        dataset_id: OpenML dataset ID
        task_type: 'classification' or 'regression'
        output_dir: Directory to save the dataset

    Returns:
        bool: True if successful, False otherwise
    """
    try:
        print(f"  Downloading dataset {dataset_id}...")

        # Download dataset from OpenML
        dataset = openml.datasets.get_dataset(dataset_id)
        X, y, categorical_indicator, attribute_names = dataset.get_data(
            dataset_format='dataframe',
            target=dataset.default_target_attribute
        )

        # Combine features and target
        df = X.copy()
        target_col = dataset.default_target_attribute
        df[target_col] = y

        # Get dataset info
        n_features = len(attribute_names) - 1  # Exclude target
        n_samples = len(df)

        print(f"    Shape: {n_samples:,} samples × {n_features} features")
        print(f"    Target: {target_col}")

        # Determine size category
        if n_samples < 10000:
```

```
            obs_size = "small"
        elif n_samples < 50000:
            obs_size = "medium"
        else:
            obs_size = "large"

        if n_features < 20:
            feat_size = "small"
        elif n_features < 100:
            feat_size = "medium"
        else:
            feat_size = "large"

        dimensionality = f"{obs_size}x{feat_size}"

        # Create filename (without Bal/Noisy/Nonlinear - that comes from
detection/transformation)
        if task_type == 'classification':
            filename =
f"obs_{dimensionality}_Type_Classification_ID{dataset_id}_RAW.csv"
        else:
            filename =
f"obs_{dimensionality}_Type_Regression_ID{dataset_id}_RAW.csv"

        # Save
        output_path = output_dir / filename
        df.to_csv(output_path, index=False)

        print(f"    ✓ Saved: {filename}")
        return True

    except Exception as e:
        print(f"    ✗ Error downloading dataset {dataset_id}: {e}")
        return False

def main():
    parser = argparse.ArgumentParser(description='Download exact 72 OpenML
datasets')
    parser.add_argument('--task', choices=['classification', 'regression',
'all'],
                        default='all', help='Which datasets to download')
    args = parser.parse_args()

    # Create output directories
    output_dir = Path('data/raw_downloads')
    class_dir = output_dir / 'classification'
    reg_dir = output_dir / 'regression'

    class_dir.mkdir(parents=True, exist_ok=True)
    reg_dir.mkdir(parents=True, exist_ok=True)

    print("=" * 80)
    print("Downloading Exact 72 OpenML Datasets")
    print("=" * 80)
    print(f"Output directory: {output_dir}")
    print("")

    success_count = 0
    failed_count = 0

    # Download classification datasets
    if args.task in ['classification', 'all']:
        print(f"\n{'='*80}")
        print(f"CLASSIFICATION DATASETS ({len(CLASSIFICATION_IDS)} total)")
        print(f"{'='*80}\n")

        for i, dataset_id in enumerate(CLASSIFICATION_IDS, 1):
            print(f"[{i}/{len(CLASSIFICATION_IDS)}] Dataset ID {dataset_id}")
            if download_dataset(dataset_id, 'classification', class_dir):
                success_count += 1
            else:
                failed_count += 1
            print("")

    # Download regression datasets
    if args.task in ['regression', 'all']:
        print(f"\n{'='*80}")
        print(f"REGRESSION DATASETS ({len(REGRESSION_IDS)} total)")
        print(f"{'='*80}\n")

        for i, dataset_id in enumerate(REGRESSION_IDS, 1):
            print(f"[{i}/{len(REGRESSION_IDS)}] Dataset ID {dataset_id}")
```

```
            if download_dataset(dataset_id, 'regression', reg_dir):
                success_count += 1
            else:
                failed_count += 1
            print("")

    # Summary
    print("=" * 80)
    print("DOWNLOAD SUMMARY")
    print("=" * 80)
    print(f"✓ Successful: {success_count}")
    print(f"✗ Failed: {failed_count}")
    print(f"Total: {success_count + failed_count}")
    print("")
    print(f"Downloaded datasets saved to: {output_dir}")
    print("")
    print("=" * 80)
    print("NEXT STEPS:")
    print("=" * 80)
    print("1. Run comparison script to identify transformations:")
    print("   python compare_raw_vs_processed.py")
    print("")
    print("2. Apply transformations using dataset_editor.py as needed")
    print("")
    print("3. Verify final datasets match:")
    print("   python verify_datasets.py")
    print("=" * 80)

if __name__ == '__main__':
    main()

# Download all 72 datasets
python download_exact_datasets.py

# Or download only classification/regression
python download_exact_datasets.py --task classification
python download_exact_datasets.py --task regression
```

What It Does

1. Downloads the **exact 72 OpenML datasets** by their IDs (note, dataset with ID = 41767 failed to download on testing, reason unknown. Manual download may be required directly from OpenML, dataset name = FOREX_cadjpy-hour-Close)
2. Saves raw datasets to:
   - `data/raw_downloads/classification/`
   - `data/raw_downloads/regression/`
3. Applies naming convention with metadata in filenames

OpenML Dataset IDs

Classification (36 datasets):

```
16, 18, 143, 180, 279, 375, 727, 728, 807, 958, 977, 1022, 1042, 1217, 1242,
1425,
1475, 1509, 1537, 1548, 1587, 1590, 23512, 41767, 42680, 44130, 44563, 45078,
46456,
46526, 46561, 46609, 46633, 46805, 46856, 46874
```

Regression (36 datasets):

```
189, 197, 344, 422, 572, 574, 1197, 1201, 1203, 1592, 41187, 41211, 41506,
41980,
42570, 42571, 42635, 42705, 42729, 42737, 43090, 43144, 43873, 43988, 44002,
44057,
44060, 44061, 44067, 44977, 46161, 46293, 46618, 46620, 46787, 46898
```

Output

Raw datasets in `data/raw_downloads/` with naming convention: - Classification: `obs_{dim}_Bal_{bool}_Noisy_{bool}_Type_Classification_ID{id}.csv` - Regression: `obs_{dim}_Noisy_{bool}_Nonlinear_{bool}_Type_Regression_ID{id}.csv`

Note: Properties in filenames reflect the **detected/natural** characteristics, not the final target properties.

Phase 2: Identify Required Transformations

How to Use This Table

1. Download the 72 OpenML datasets listed below using `download_exact_datasets.py`
2. For each dataset, check the "Transformations Applied" column
3. If transformations are needed, use `data/dataset_editor.py` to apply them

Classification Datasets (36 total)

Small × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
728	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
375	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
807	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
18	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Small × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
958	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1475	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
16	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
44130	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Small × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
1022	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1548	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1042	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded

OpenML ID	Final Properties	Transformations Applied	Notes
			dataset to determine transformations
44563	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
1537	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1590	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
727	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
41767	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
977	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
279	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46526	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46874	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
1425	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46561	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
45078	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to

OpenML ID	Final Properties	Transformations Applied	Notes
			determine transformations
1587	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
46456	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1509	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46609	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1217	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
180	Bal=False, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42680	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
23512	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
143	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
46633	Bal=False, Noisy=False	**DENOISE** (raw is Noisy=True)	Example: ID 46633 downloaded as Noisy=True, denoised to Noisy=False
46856	Bal=False, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1242	Bal=True, Noisy=False	TBD - Check raw vs processed	Compare downloaded dataset to

OpenML ID	Final Properties	Transformations Applied	Notes
			determine transformations
46805	Bal=True, Noisy=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Regression Datasets (36 total)

Small × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
572	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
41187	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46161	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
189	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Small × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
46618	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
197	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
41211	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46620	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Small × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
44061	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42570	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

OpenML ID	Final Properties	Transformations Applied	Notes
422	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
41980	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
43988	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
44977	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
44002	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
344	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
43873	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
574	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
41506	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42635	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Medium × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
46293	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
44067	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

OpenML ID	Final Properties	Transformations Applied	Notes
46787	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
46898	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Small (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
1203	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1197	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
44057	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1201	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Medium (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
43144	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
43090	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42737	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42729	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

Large × Large (4 datasets)

OpenML ID	Final Properties	Transformations Applied	Notes
42705	Noisy=False, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
1592	Noisy=False, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

OpenML ID	Final Properties	Transformations Applied	Notes
44060	Noisy=True, Nonlinear=False	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations
42571	Noisy=True, Nonlinear=True	TBD - Check raw vs processed	Compare downloaded dataset to determine transformations

---

Complete List of 72 OpenML IDs

Classification (36 IDs):

```
16, 18, 143, 180, 279, 375, 727, 728, 807, 958, 977, 1022, 1042, 1217, 1242,
1425,
1475, 1509, 1537, 1548, 1587, 1590, 23512, 41767, 42680, 44130, 44563, 45078,
46456,
46526, 46561, 46609, 46633, 46805, 46856, 46874
```

Regression (36 IDs):

```
189, 197, 344, 422, 572, 574, 1197, 1201, 1203, 1592, 41187, 41211, 41506,
41980,
42570, 42571, 42635, 42705, 42729, 42737, 43090, 43144, 43873, 43988, 44002,
44057,
44060, 44061, 44067, 44977, 46161, 46293, 46618, 46620, 46787, 46898
```

Phase 3: Manual Dataset Transformation

Script: `data/dataset_editor.py`

```
import pandas as pd
import numpy as np
import re
import tkinter as tk
from tkinter import filedialog
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import os

# ----------------------------
# Utility Functions
# ----------------------------

def parse_filename(filename):
    """Extract dataset properties from filename."""
    props = {
        "balanced": None,
        "noisy": None,
        "task": None,
        "nonlinear": None,
        "id": None,
        "size_category": None
    }
    props["balanced"] = "Bal_True" in filename
    props["noisy"] = "Noisy_True" in filename
    props["task"] = "Classification" if "Classification" in filename else
"Regression"
    props["nonlinear"] = "Nonlinear_True" in filename if props["task"] ==
"Regression" else None
    id_match = re.search(r'ID(\d+)', filename)
    if id_match:
        props["id"] = id_match.group(1)

    # Extract size category from filename - first part is observations,
second is features
    size_match = re.search(r'obs_(\w+)x(\w+)', filename)
    if size_match:
        obs_size = size_match.group(1)  # First part = observation size
        feature_size = size_match.group(2)  # Second part = feature size
```

```
        # Use observation size category for dataset size management
        if 'large' in obs_size:
            props["size_category"] = "large"
        elif 'medium' in obs_size:
            props["size_category"] = "medium"
        elif 'small' in obs_size:
            props["size_category"] = "small"

    return props

def add_noise(df, target_col, noise_level=0.15):
    """Add Gaussian noise to numerical features."""
    noisy_df = df.copy()
    features_modified = 0

    for col in noisy_df.select_dtypes(include=np.number).columns:
        if col != target_col:
            col_std = noisy_df[col].std()
            if col_std > 0:  # Only add noise if column has variation
                noise = np.random.normal(0, noise_level * col_std,
size=len(noisy_df))
                noisy_df[col] += noise
                features_modified += 1

    print(f"Added {noise_level*100}% noise to {features_modified} numerical
features")
    return noisy_df

def denoise(df, target_col):
    """Remove noise using smoothing and capping without removing rows."""
    clean_df = df.copy()
    original_size = len(clean_df)
    features_processed = 0

    # Method 1: Cap extreme outliers instead of removing them
    for col in clean_df.select_dtypes(include=np.number).columns:
        if col != target_col:
            Q1 = clean_df[col].quantile(0.25)
            Q3 = clean_df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower = Q1 - 3.0 * IQR
            upper = Q3 + 3.0 * IQR

            # Cap outliers instead of removing rows
            outliers_capped = 0
            outliers_capped += (clean_df[col] < lower).sum()
            outliers_capped += (clean_df[col] > upper).sum()

            clean_df[col] = clean_df[col].clip(lower=lower, upper=upper)

            if outliers_capped > 0:
                features_processed += 1

    # Method 2: Apply smoothing to numerical features
    smoothed_features = 0
    for col in clean_df.select_dtypes(include=np.number).columns:
        if col != target_col:
            # Rolling mean smoothing (window=5)
            clean_df[col] = clean_df[col].rolling(window=5, center=True,
min_periods=1).mean()
            smoothed_features += 1

    print(f"Denoising: capped outliers in {features_processed} features,
applied smoothing to {smoothed_features} features")
    print(f"Dataset size maintained: {len(clean_df)} rows (no rows removed)")
    return clean_df

def balance_classification(df, target_col):
    """Balance classification dataset using SMOTE."""
    from sklearn.preprocessing import LabelEncoder
    from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import StandardScaler, OneHotEncoder

    original_counts = df[target_col].value_counts()
    print(f"Original class distribution: {dict(original_counts)}")

    X = df.drop(columns=[target_col])
    y = df[target_col]

    # Identify categorical and numerical columns
    categorical_cols = X.select_dtypes(include=['object',
'category']).columns.tolist()
```

```
    numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()

    print(f"Found {len(categorical_cols)} categorical and
{len(numerical_cols)} numerical features")

    # Preprocessing pipeline
    if categorical_cols:
        # Use OneHotEncoder for categorical features
        preprocessor = ColumnTransformer(
            transformers=[
                ('num', StandardScaler(), numerical_cols),
                ('cat', OneHotEncoder(drop='first', sparse_output=False),
categorical_cols)
            ]
        )
        X_processed = preprocessor.fit_transform(X)

        # Get feature names for the processed data
        num_feature_names = numerical_cols
        cat_feature_names =
preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_col
        all_feature_names = list(num_feature_names) + list(cat_feature_names)
    else:
        # Only numerical features
        preprocessor = StandardScaler()
        X_processed = preprocessor.fit_transform(X)
        all_feature_names = numerical_cols

    # Apply SMOTE to processed data
    sm = SMOTE(random_state=42)
    X_res, y_res = sm.fit_resample(X_processed, y)

    # Create balanced dataframe
    balanced_df = pd.DataFrame(X_res, columns=all_feature_names)
    balanced_df[target_col] = y_res

    new_counts = pd.Series(y_res).value_counts()
    print(f"Balanced class distribution: {dict(new_counts)}")

    return balanced_df

def unbalance_classification(df, target_col):
    """Create imbalanced dataset by reducing majority class."""
    class_counts = df[target_col].value_counts()
    major_class = class_counts.idxmax()
    minor_class = class_counts.idxmin()

    df_major = df[df[target_col] == major_class]
    df_minor = df[df[target_col] == minor_class]

    # Reduce majority class to create imbalance (aim for 3:1 or 4:1 ratio)
    target_major_size = len(df_minor) * 3
    if len(df_major) > target_major_size:
        df_major = df_major.sample(n=target_major_size, random_state=42)

    result = pd.concat([df_major, df_minor], axis=0).sample(frac=1,
random_state=42).reset_index(drop=True)
    print(f"Created imbalanced dataset: {class_counts[major_class]} ->
{len(df_major)} (majority), {class_counts[minor_class]} (minority)")
    return result

def make_linear(df, target_col):
    """Transform target to be a linear combination of features."""
    from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import StandardScaler, OneHotEncoder

    df_linear = df.copy()
    X = df_linear.drop(columns=[target_col])

    # Identify categorical and numerical columns
    categorical_cols = X.select_dtypes(include=['object',
'category']).columns.tolist()
    numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()

    print(f"Found {len(categorical_cols)} categorical and
{len(numerical_cols)} numerical features")

    # Preprocessing pipeline to handle mixed data types
    if categorical_cols:
        # Use OneHotEncoder for categorical features
```

```
        preprocessor = ColumnTransformer(
            transformers=[
                ('num', StandardScaler(), numerical_cols),
                ('cat', OneHotEncoder(drop='first', sparse_output=False),
categorical_cols)
            ]
        )
        X_processed = preprocessor.fit_transform(X)
    else:
        # Only numerical features
        preprocessor = StandardScaler()
        X_processed = preprocessor.fit_transform(X)

    # Check for NaN/inf values in processed data
    print(f"Debug: X_processed contains NaN: {np.isnan(X_processed).any()}")
    print(f"Debug: X_processed contains inf: {np.isinf(X_processed).any()}")
    print(f"Debug: X_processed min/max: {X_processed.min():.3f} /
{X_processed.max():.3f}")

    # Handle NaN/inf values
    if np.isnan(X_processed).any() or np.isinf(X_processed).any():
        print("  Found NaN/inf values in processed data, replacing with 0")
        X_processed = np.nan_to_num(X_processed, nan=0.0, posinf=0.0,
neginf=0.0)

    # Create linear relationship with random coefficients
    coef = np.random.uniform(-2, 2, X_processed.shape[1])
    intercept = np.random.uniform(-1, 1)

    # Generate linear target with small noise
    linear_target = X_processed @ coef + intercept + np.random.normal(0,
0.05, size=X_processed.shape[0])

    print(f"Debug: X_processed shape: {X_processed.shape}")
    print(f"Debug: linear_target shape: {linear_target.shape}")
    print(f"Debug: df_linear shape: {df_linear.shape}")
    print(f"Debug: linear_target sample values: {linear_target[:5]}")
    print(f"Debug: target_col name: '{target_col}'")
    print(f"Debug: df_linear[target_col] before assignment:
{df_linear[target_col].head()}")

    # Assign the new target values
    df_linear[target_col] = linear_target

    print(f"Debug: df_linear[target_col] after assignment:
{df_linear[target_col].head()}")
    print(f"Debug: Assignment successful: {not
df_linear[target_col].isna().all()}")

    print(f"Created linear target with coefficients range: [{coef.min():.2f},
{coef.max():.2f}]")
    print(f"Used all {X_processed.shape[1]} features (including
{len(categorical_cols)} categorical after encoding)")
    return df_linear

def make_nonlinear(df, target_col):
    """Transform target to have non-linear relationships with features."""
    from sklearn.compose import ColumnTransformer
    from sklearn.preprocessing import StandardScaler, OneHotEncoder

    df_nonlinear = df.copy()
    X = df_nonlinear.drop(columns=[target_col])

    # Identify categorical and numerical columns
    categorical_cols = X.select_dtypes(include=['object',
'category']).columns.tolist()
    numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()

    print(f"Found {len(categorical_cols)} categorical and
{len(numerical_cols)} numerical features")

    # Preprocessing pipeline to handle mixed data types
    if categorical_cols:
        # Use OneHotEncoder for categorical features
        preprocessor = ColumnTransformer(
            transformers=[
                ('num', StandardScaler(), numerical_cols),
                ('cat', OneHotEncoder(drop='first', sparse_output=False),
categorical_cols)
            ]
        )
```

```
        X_processed = preprocessor.fit_transform(X)
    else:
        # Only numerical features
        preprocessor = StandardScaler()
        X_processed = preprocessor.fit_transform(X)

    # Check for NaN/inf values in processed data
    print(f"Debug: X_processed contains NaN: {np.isnan(X_processed).any()}")
    print(f"Debug: X_processed contains inf: {np.isinf(X_processed).any()}")

    # Handle NaN/inf values
    if np.isnan(X_processed).any() or np.isinf(X_processed).any():
        print("  Found NaN/inf values in processed data, replacing with 0")
        X_processed = np.nan_to_num(X_processed, nan=0.0, posinf=0.0,
neginf=0.0)

    # Create non-linear combinations using first 5 features
    nonlinear_target = np.zeros(len(X_processed))

    # Apply non-linear transformations to first few features
    for i in range(min(5, X_processed.shape[1])):
        x = X_processed[:, i]
        if i == 0:
            nonlinear_target += np.sin(2 * x)
        elif i == 1:
            nonlinear_target += np.log1p(np.abs(x) + 1)
        elif i == 2:
            nonlinear_target += np.sqrt(np.abs(x) + 1)
        elif i == 3:
            nonlinear_target += x ** 2
        else:
            nonlinear_target += np.tanh(x)

    # Add interaction terms if we have at least 2 features
    if X_processed.shape[1] >= 2:
        nonlinear_target += X_processed[:, 0] * X_processed[:, 1]

    # Add noise
    nonlinear_target += np.random.normal(0, 0.1, X_processed.shape[0])

    print(f"Debug: X_processed shape: {X_processed.shape}")
    print(f"Debug: nonlinear_target shape: {nonlinear_target.shape}")
    print(f"Debug: df_nonlinear shape: {df_nonlinear.shape}")

    df_nonlinear[target_col] = nonlinear_target
    print(f"Created non-linear target using {min(5, X_processed.shape[1])}
features with interactions")
    print(f"Used all {X_processed.shape[1]} features (including
{len(categorical_cols)} categorical after encoding)")
    return df_nonlinear

def check_and_adjust_dataset_size(df, target_col, size_category):
    """Check if dataset size matches category and downsample if needed."""

    # Size ranges based on open_ml_downloader.py
    SIZE_RANGES = {
        "small": (2000, 10000, 6000),     # min, max, target_middle
        "medium": (20000, 50000, 35000),  # min, max, target_middle
        "large": (80000, 200000, 100000)  # min, max, target_cap_at_100k
    }

    current_size = len(df)

    if size_category not in SIZE_RANGES:
        print(f"Unknown size category: {size_category}. Leaving dataset
unchanged.")
        return df

    min_size, max_size, target_size = SIZE_RANGES[size_category]

    print(f"\nSize Analysis for {size_category} category:")
    print(f"  Current size: {current_size:,} observations")
    print(f"  Expected range: {min_size:,} - {max_size:,}")

    # Special handling for large category - cap at 100k even if within range
    if size_category == "large" and current_size > 100000:
        print(f"    Large dataset above 100k cap ({current_size:,} >
100,000).")
        print(f"   Downsampling to 100,000 observations for performance...")
        # Continue to downsampling logic below
    elif min_size <= current_size <= max_size:
        print(f"   Size is within {size_category} range. No adjustment
```

```
needed.")
        return df

    # Need to downsample
    if current_size > max_size or (size_category == "large" and current_size
> 100000):
        print(f"    Dataset too large for {size_category} category.")
        print(f"   Downsampling to {target_size:,} observations...")

        # For classification, maintain class balance during downsampling
        if df[target_col].dtype == 'object' or len(df[target_col].unique())
<= 10:
            # Classification task - stratified sampling to maintain original
proportions
            from sklearn.model_selection import train_test_split

            class_counts = df[target_col].value_counts()
            print(f"  Original class distribution: {dict(class_counts)}")

            # Calculate proportional samples per class to maintain original
ratio
            total_current = len(df)
            downsampled_dfs = []

            for class_val in class_counts.index:
                class_df = df[df[target_col] == class_val]
                original_proportion = len(class_df) / total_current
                target_class_size = int(target_size * original_proportion)

                print(f"    Class {class_val}: {len(class_df):,} →
{target_class_size:,} (maintaining {original_proportion:.1%})")

                if len(class_df) > target_class_size:
                    sampled_df = class_df.sample(n=target_class_size,
random_state=42)
                else:
                    sampled_df = class_df  # Use all samples if class is
small
                downsampled_dfs.append(sampled_df)

            result_df = pd.concat(downsampled_dfs, axis=0).sample(frac=1,
random_state=42).reset_index(drop=True)

            new_class_counts = result_df[target_col].value_counts()
            print(f"  New class distribution: {dict(new_class_counts)}")

        else:
            # Regression task - random sampling
            print(f"  Random sampling for regression task...")
            result_df = df.sample(n=target_size,
random_state=42).reset_index(drop=True)

        print(f"   Downsampled from {current_size:,} to {len(result_df):,}
observations")
        return result_df

    else:
        print(f"    Dataset too small for {size_category} category
({current_size:,} < {min_size:,}).")
        print(f"  Leaving dataset unchanged.")
        return df

# ----------------------------
# Main Script Logic
# ----------------------------

def main():
    # Open file dialog
    root = tk.Tk()
    root.withdraw()
    file_path = filedialog.askopenfilename(title="Select Dataset", filetypes=
[("CSV files", "*.csv")])
    if not file_path:
        print("No file selected.")
        return

    filename = os.path.basename(file_path)
    props = parse_filename(filename)
    print(f"\n Parsed properties from filename:\n{props}\n")

    df = pd.read_csv(file_path)
    print(f"Loaded dataset with shape: {df.shape}")
```

```
    # Show column options
    print("\nAvailable columns:")
    for i, col in enumerate(df.columns):
        print(f"  {i}: {col}")

    while True:
        try:
            col_idx = int(input(f"\nSelect target column number (0-
{len(df.columns)-1}): "))
            if 0 <= col_idx < len(df.columns):
                target_col = df.columns[col_idx]
                print(f"Selected target column: '{target_col}'")
                break
            else:
                print(f"Please enter a number between 0 and
{len(df.columns)-1}")
        except ValueError:
            print("Please enter a valid number")

    # Check and adjust dataset size first
    if props["size_category"]:
        df = check_and_adjust_dataset_size(df, target_col,
props["size_category"])
    else:
        print("  Could not determine size category from filename. Skipping
size check.")

    # Ask about noise
    noise_action = input(f"Change noise? [add / remove / leave]:
").lower().strip()
    if noise_action == "add":
        df = add_noise(df, target_col)
        props["noisy"] = True
    elif noise_action == "remove":
        df = denoise(df, target_col)
        props["noisy"] = False

    # Classification handling
    if props["task"] == "Classification":
        balance_action = input(f"Balance class distribution? [balance /
unbalance / leave]: ").lower().strip()
        if balance_action == "balance":
            df = balance_classification(df, target_col)
            props["balanced"] = True
        elif balance_action == "unbalance":
            df = unbalance_classification(df, target_col)
            props["balanced"] = False

    # Regression handling
    if props["task"] == "Regression":
        linear_action = input(f"Change linearity? [linear / nonlinear /
leave]: ").lower().strip()
        if linear_action == "linear":
            df = make_linear(df, target_col)
            props["nonlinear"] = False
        elif linear_action == "nonlinear":
            df = make_nonlinear(df, target_col)
            props["nonlinear"] = True

    # Generate new filename with updated properties
    original_name = os.path.splitext(filename)[0]  # Remove .csv extension

    # Update filename to reflect new properties
    new_name = original_name

    # Update noise flag in filename
    if props["noisy"] is not None:
        if props["noisy"]:
            new_name = re.sub(r'Noisy_False', 'Noisy_True', new_name)
        else:
            new_name = re.sub(r'Noisy_True', 'Noisy_False', new_name)

    # Update balance flag in filename (classification)
    if props["task"] == "Classification" and props["balanced"] is not None:
        if props["balanced"]:
            new_name = re.sub(r'Bal_False', 'Bal_True', new_name)
        else:
            new_name = re.sub(r'Bal_True', 'Bal_False', new_name)

    # Update linearity flag in filename (regression)
    if props["task"] == "Regression" and props["nonlinear"] is not None:
```

```
        if props["nonlinear"]:
            new_name = re.sub(r'Nonlinear_False', 'Nonlinear_True', new_name)
        else:
            new_name = re.sub(r'Nonlinear_True', 'Nonlinear_False', new_name)

    new_filename = f"transformed_{new_name}.csv"

    save_path = os.path.join(os.path.dirname(file_path), new_filename)
    df.to_csv(save_path, index=False)

    print(f"\n Dataset transformed and saved as:")
    print(f"   {new_filename}")
    print(f"   Shape: {df.shape}")
    print(f"   Location: {save_path}")

main()
```

This is an **interactive GUI tool** that must be run for each dataset requiring transformation.

```
python data/dataset_editor.py
```

Interactive Workflow

1. **File Selection**: GUI dialog opens to select a CSV file
2. **Auto-Detection**: Script parses filename and detects:
   - Task type (Classification/Regression)
   - Dimensionality category
   - Current properties (balanced, noisy, nonlinear)
   - OpenML ID
3. **Target Column Selection**:

   ```
   Available columns:
     0: feature_1
     1: feature_2
     ...
     n: target_column

   Select target column number (0-n):
   ```

4. **Automatic Size Check/Adjustment**:
   - Checks if dataset size exceeds category limits
   - Size caps: Small=1,000, Medium=5,000, Large=15,000
   - Automatically downsamples if needed:
     - Classification: Stratified sampling (maintains class proportions)
     - Regression: Random sampling
   - Uses `random_state=42`
5. **Noise Transformation**:

   ```
   Change noise? [add / remove / leave]:
   ```

   - **add**: Adds Gaussian noise (sigma = 0.15 x sigma_feature) to all numerical features
   - **remove**: Denoise via outlier capping (Q1 - 3xIQR, Q3 + 3xIQR) and rolling mean smoothing
   - **leave**: No change
6. **Classification - Balance Transformation**:

   ```
   Balance class distribution? [balance / unbalance / leave]:
   ```

   - **balance**: Apply SMOTE with `random_state=42`
   - **unbalance**: Downsample majority class to 3:1 or 4:1 ratio
   - **leave**: No change
7. **Regression - Linearity Transformation**:

   ```
   Change linearity? [linear / nonlinear / leave]:
   ```

   - **linear**: Create linear target = X @ coef + intercept + noise
   - **nonlinear**: Create non-linear target using sin, log, sqrt, x^2, tanh, interactions
   - **leave**: No change
8. **Save Transformed Dataset**:
   - Automatically updates filename to reflect new properties
   - Saves as `transformed_{updated_name}.csv`

Transformation Functions

Function	Purpose	Parameters
`add_noise()`	Add Gaussian noise	sigma = 0.15 x sigma_feature
`denoise()`	Remove noise	Outlier cap + rolling mean smoothing
`balance_classification()`	Balance classes	SMOTE, random_state=42
`unbalance_classification()`	Create imbalance	Downsample 3:1 ratio, random_state=42
`make_linear()`	Linear target	Random coefficients, random_state=42
`make_nonlinear()`	Non-linear target	sin, log, x^2, interactions
`reduce_dataset_size()`	Size reduction	Stratified/random sampling, random_state=42

Example Transformation Session

```
Parsed properties from filename:
{'balanced': False, 'noisy': True, 'task': 'Classification',
 'id': '46633', 'size_category': 'large'}

Loaded dataset with shape: (120000, 50)

Available columns:
  0: feature_1
  ...
  49: class

Select target column number (0-49): 49
Selected target column: 'class'

Size Analysis for large category:
  Current size: 120,000 observations
  Maximum allowed: 15,000
  WARNING: Dataset exceeds size cap.
  Downsampling to 15,000 observations...
  Downsampled from 120,000 to 15,000 observations (stratified)

Change noise? [add / remove / leave]: remove
Denoising: capped outliers in 35 features, applied smoothing to 49 features

Balance class distribution? [balance / unbalance / leave]: leave

Dataset transformed and saved as:

transformed_obs_largexlarge_Bal_False_Noisy_False_Type_Classification_ID46633
   Shape: (15000, 50)
```

Repeat for All 72 Datasets

Run `dataset_editor.py` for each dataset that needs transformation: - Some datasets may be used directly (no transformation needed) - Others require noise addition/removal - Some require balance/linearity changes - Large datasets are automatically capped

---

Phase 4: Organize and Verify Datasets

Step 4.1: Create Directory Structure

```
mkdir -p data/processed_classification/{smallxsmall,smallxmed,smallxlarge,medxsmall,medxmed,medxlarge,largexsma
mkdir -p data/processed_regression/{smallxsmall,smallxmed,smallxlarge,medxsmall,medxmed,medxlarge,largexsmall,l
```

Step 4.2: Move Transformed Datasets

Manually move the final 72 datasets to appropriate subdirectories: 1. Remove “transformed_” prefix if present 2. Place in correct dimensionality folder based on filename 3. Ensure naming convention is consistent

Step 4.3: Verify with Script

```
import pandas as pd
import argparse
from pathlib import Path
import re

# Expected dataset structure
```

```
EXPECTED_STRUCTURE = {
    'classification': {
        'smallxsmall': [
            ('728', 'False', 'False'),
            ('375', 'False', 'True'),
            ('807', 'True', 'False'),
            ('18', 'True', 'True'),
        ],
        'smallxmedium': [
            ('958', 'False', 'False'),
            ('1475', 'False', 'True'),
            ('16', 'True', 'False'),
            ('44130', 'True', 'True'),
        ],
        'smallxlarge': [
            ('1022', 'False', 'False'),
            ('1548', 'False', 'True'),
            ('1042', 'True', 'False'),
            ('44563', 'True', 'True'),
        ],
        'mediumxsmall': [
            ('1537', 'False', 'False'),
            ('1590', 'False', 'True'),
            ('727', 'True', 'False'),
            ('41767', 'True', 'True'),
        ],
        'mediumxmedium': [
            ('977', 'False', 'False'),
            ('279', 'False', 'True'),
            ('46526', 'True', 'False'),
            ('46874', 'True', 'True'),
        ],
        'mediumxlarge': [
            ('1425', 'False', 'False'),
            ('46561', 'False', 'True'),
            ('45078', 'True', 'False'),
            ('1587', 'True', 'True'),
        ],
        'largexsmall': [
            ('46456', 'False', 'False'),
            ('1509', 'False', 'True'),
            ('46609', 'True', 'False'),
            ('1217', 'True', 'True'),
        ],
        'largexmedium': [
            ('180', 'False', 'False'),
            ('42680', 'False', 'True'),
            ('23512', 'True', 'False'),
            ('143', 'True', 'True'),
        ],
        'largexlarge': [
            ('46633', 'False', 'False'),
            ('46856', 'False', 'True'),
            ('1242', 'True', 'False'),
            ('46805', 'True', 'True'),
        ],
    },
    'regression': {
        'smallxsmall': [
            ('572', 'False', 'False'),
            ('41187', 'False', 'True'),
            ('46161', 'True', 'False'),
            ('189', 'True', 'True'),
        ],
        'smallxmedium': [
            ('46618', 'False', 'False'),
            ('197', 'False', 'True'),
            ('41211', 'True', 'False'),
            ('46620', 'True', 'True'),
        ],
        'smallxlarge': [
            ('44061', 'False', 'False'),
            ('42570', 'False', 'True'),
            ('422', 'True', 'False'),
            ('41980', 'True', 'True'),
        ],
        'mediumxsmall': [
            ('43988', 'False', 'False'),
            ('44977', 'False', 'True'),
            ('44002', 'True', 'False'),
            ('344', 'True', 'True'),
        ],
```

```
        'mediumxmedium': [
            ('43873', 'False', 'False'),
            ('574', 'False', 'True'),
            ('41506', 'True', 'False'),
            ('42635', 'True', 'True'),
        ],
        'mediumxlarge': [
            ('46293', 'False', 'False'),
            ('44067', 'False', 'True'),
            ('46787', 'True', 'False'),
            ('46898', 'True', 'True'),
        ],
        'largexsmall': [
            ('1203', 'False', 'False'),
            ('1197', 'False', 'True'),
            ('44057', 'True', 'False'),
            ('1201', 'True', 'True'),
        ],
        'largexmedium': [
            ('43144', 'False', 'False'),
            ('43090', 'False', 'True'),
            ('42737', 'True', 'False'),
            ('42729', 'True', 'True'),
        ],
        'largexlarge': [
            ('42705', 'False', 'False'),
            ('1592', 'False', 'True'),
            ('44060', 'True', 'False'),
            ('42571', 'True', 'True'),
        ],
    }
}

# Size constraints
SIZE_CONSTRAINTS = {
    'small': (0, 15000),
    'medium': (0, 15000),
    'large': (0, 15000)
}

def parse_filename(filename):
    """Extract metadata from filename."""
    info = {}

    id_match = re.search(r'ID(\d+)', filename)
    info['openml_id'] = id_match.group(1) if id_match else None

    dim_match = re.search(r'obs_(small|medium|large)x(small|medium|large)',
filename)
    if dim_match:
        info['dimensionality'] = f"{dim_match.group(1)}x{dim_match.group(2)}"
        info['obs_size'] = dim_match.group(1)
        info['feat_size'] = dim_match.group(2)
    else:
        info['dimensionality'] = None
        info['obs_size'] = None
        info['feat_size'] = None

    info['task'] = 'classification' if 'Classification' in filename else
'regression'

    bal_match = re.search(r'Bal_(True|False)', filename)
    noisy_match = re.search(r'Noisy_(True|False)', filename)
    nonlin_match = re.search(r'Nonlinear_(True|False)', filename)

    info['balanced'] = bal_match.group(1) if bal_match else None
    info['noisy'] = noisy_match.group(1) if noisy_match else None
    info['nonlinear'] = nonlin_match.group(1) if nonlin_match else None

    return info

def verify_dataset(file_path, expected_props, verbose=False):
    """
    Verify a single dataset.

    Args:
        file_path: Path to dataset CSV
        expected_props: Tuple of (openml_id, prop1, prop2)
        verbose: Print detailed information

    Returns:
        dict: Verification results
```

```
    """
    filename = file_path.name
    info = parse_filename(filename)

    issues = []
    warnings = []

    # Check filename matches expected
    expected_id = expected_props[0]
    if info['openml_id'] != expected_id:
        issues.append(f"OpenML ID mismatch: expected {expected_id}, got
{info['openml_id']}")

    # Load dataset
    try:
        df = pd.read_csv(file_path)

        # Check TARGET_VARIABLE exists
        if 'TARGET_VARIABLE' not in df.columns:
            issues.append("Missing TARGET_VARIABLE column")
        elif verbose:
            print(f"    ✓ TARGET_VARIABLE column present")

        # Check size constraints
        n_rows = len(df)
        n_cols = len(df.columns)

        obs_size = info['obs_size']
        if obs_size and obs_size in SIZE_CONSTRAINTS:
            min_size, max_size = SIZE_CONSTRAINTS[obs_size]
            if max_size > 0 and n_rows > max_size:
                warnings.append(f"Size {n_rows:,} exceeds max {max_size:,}
for {obs_size}")
            elif verbose:
                print(f"    ✓ Size {n_rows:,} within constraints for
{obs_size}")

        # Check for NaN values
        nan_count = df.isna().sum().sum()
        if nan_count > 0:
            warnings.append(f"Contains {nan_count} NaN values")
        elif verbose:
            print(f"    ✓ No NaN values")

        # Check properties match expected
        task = info['task']
        if task == 'classification':
            if info['balanced'] != expected_props[1]:
                issues.append(f"Balanced mismatch: expected
{expected_props[1]}, got {info['balanced']}")
            if info['noisy'] != expected_props[2]:
                issues.append(f"Noisy mismatch: expected {expected_props[2]},
got {info['noisy']}")
        else:  # regression
            if info['noisy'] != expected_props[1]:
                issues.append(f"Noisy mismatch: expected {expected_props[1]},
got {info['noisy']}")
            if info['nonlinear'] != expected_props[2]:
                issues.append(f"Nonlinear mismatch: expected
{expected_props[2]}, got {info['nonlinear']}")

        return {
            'filename': filename,
            'openml_id': info['openml_id'],
            'dimensionality': info['dimensionality'],
            'size': f"{n_rows:,} × {n_cols}",
            'issues': issues,
            'warnings': warnings,
            'status': 'PASS' if not issues else 'FAIL'
        }

    except Exception as e:
        return {
            'filename': filename,
            'openml_id': info['openml_id'],
            'dimensionality': info['dimensionality'],
            'size': 'Error',
            'issues': [f"Error loading: {str(e)}"],
            'warnings': [],
            'status': 'FAIL'
        }
```

```
def main():
    parser = argparse.ArgumentParser(description='Verify processed datasets')
    parser.add_argument('--verbose', '-v', action='store_true',
                        help='Print detailed verification information')
    args = parser.parse_args()

    print("=" * 80)
    print("Verifying Processed Datasets")
    print("=" * 80)
    print("")

    proc_class_dir = Path('data/processed_classification')
    proc_reg_dir = Path('data/processed_regression')

    if not proc_class_dir.exists():
        print("✗ Error: data/processed_classification/ not found!")
        return

    if not proc_reg_dir.exists():
        print("✗ Error: data/processed_regression/ not found!")
        return

    all_results = []

    # Verify classification datasets
    print("Verifying Classification Datasets...")
    print("-" * 80)

    for dim, expected_datasets in
EXPECTED_STRUCTURE['classification'].items():
        print(f"\n{dim}:")

        dim_dir = proc_class_dir / dim
        if not dim_dir.exists():
            print(f"  ✗ Directory not found: {dim_dir}")
            continue

        for expected_id, expected_bal, expected_noisy in expected_datasets:
            # Find matching file
            files = list(dim_dir.glob(f'*ID{expected_id}.csv'))

            if not files:
                print(f"  ✗ Missing: ID {expected_id} (Bal={expected_bal},
Noisy={expected_noisy})")
                all_results.append({
                    'filename': f'ID{expected_id}',
                    'openml_id': expected_id,
                    'dimensionality': dim,
                    'size': 'Missing',
                    'issues': ['File not found'],
                    'warnings': [],
                    'status': 'FAIL'
                })
                continue

            file_path = files[0]
            print(f"  Checking: {file_path.name}")

            result = verify_dataset(file_path, (expected_id, expected_bal,
expected_noisy), args.verbose)
            all_results.append(result)

            if result['status'] == 'PASS':
                print(f"    ✓ PASS - {result['size']}")
            else:
                print(f"    ✗ FAIL")
                for issue in result['issues']:
                    print(f"      - {issue}")

            if result['warnings']:
                for warning in result['warnings']:
                    print(f"      ⚠ {warning}")

    # Verify regression datasets
    print("\n\nVerifying Regression Datasets...")
    print("-" * 80)

    for dim, expected_datasets in EXPECTED_STRUCTURE['regression'].items():
        print(f"\n{dim}:")

        dim_dir = proc_reg_dir / dim
        if not dim_dir.exists():
```

```
            print(f"  ✗ Directory not found: {dim_dir}")
            continue

        for expected_id, expected_noisy, expected_nonlin in
expected_datasets:
            # Find matching file
            files = list(dim_dir.glob(f'*ID{expected_id}.csv'))

            if not files:
                print(f"  ✗ Missing: ID {expected_id} (Noisy=
{expected_noisy}, Nonlinear={expected_nonlin})")
                all_results.append({
                    'filename': f'ID{expected_id}',
                    'openml_id': expected_id,
                    'dimensionality': dim,
                    'size': 'Missing',
                    'issues': ['File not found'],
                    'warnings': [],
                    'status': 'FAIL'
                })
                continue

            file_path = files[0]
            print(f"  Checking: {file_path.name}")

            result = verify_dataset(file_path, (expected_id, expected_noisy,
expected_nonlin), args.verbose)
            all_results.append(result)

            if result['status'] == 'PASS':
                print(f"    ✓ PASS - {result['size']}")
            else:
                print(f"    ✗ FAIL")
                for issue in result['issues']:
                    print(f"      - {issue}")

            if result['warnings']:
                for warning in result['warnings']:
                    print(f"      ⚠ {warning}")

    # Summary
    print("\n\n" + "=" * 80)
    print("VERIFICATION SUMMARY")
    print("=" * 80)

    total = len(all_results)
    passed = sum(1 for r in all_results if r['status'] == 'PASS')
    failed = total - passed

    print(f"Total datasets: {total}/72")
    print(f"✓ Passed: {passed}")
    print(f"✗ Failed: {failed}")

    if failed == 0:
        print("\n ALL DATASETS VERIFIED SUCCESSFULLY!")
        print("\nYou can now proceed with:")
        print("  python standardize_target_variables.py")
        print("  python batch_factorial_runner.py --tune")
    else:
        print("\n⚠ Some datasets failed verification.")
        print("\nRecommended actions:")
        print("1. Check missing datasets")
        print("2. Verify transformations were applied correctly")
        print("3. Re-run dataset_editor.py for failed datasets")

    print("=" * 80)

    # Save detailed results
    output_dir = Path('analysis_outputs')
    output_dir.mkdir(exist_ok=True)

    results_df = pd.DataFrame(all_results)
    results_df['issues_str'] = results_df['issues'].apply(lambda x: ';
'.join(x) if x else '')
    results_df['warnings_str'] = results_df['warnings'].apply(lambda x: ';
'.join(x) if x else '')
    results_df.to_csv(output_dir / 'dataset_verification.csv', index=False)

    print(f"\nDetailed results saved to: {output_dir /
'dataset_verification.csv'}")
```

```
if __name__ == '__main__':
    main()

# Verify all datasets match expected properties
python verify_datasets.py
```

What verify_datasets.py Checks

1. All 72 datasets exist (36 classification + 36 regression)
2. Correct naming convention
3. TARGET_VARIABLE column exists (after Phase 5)
4. Size constraints met
5. Properties match expected values
6. Correct distribution across 9 dimensionality categories

Expected Output

```
VERIFICATION SUMMARY
================================================================================
Total datasets: 72/72
Passed: 72
Failed: 0

ALL DATASETS VERIFIED SUCCESSFULLY!
```

Final Directory Structure

```
data/
+-- processed_classification/
|   +-- smallxsmall/
|   |   +--
obs_smallxsmall_Bal_False_Noisy_False_Type_Classification_ID728.csv
|   |   +--
obs_smallxsmall_Bal_False_Noisy_True_Type_Classification_ID375.csv
|   |   +--
obs_smallxsmall_Bal_True_Noisy_False_Type_Classification_ID807.csv
|   |   +-- obs_smallxsmall_Bal_True_Noisy_True_Type_Classification_ID18.csv
|   +-- smallxmed/
|   |   +-- ... (4 datasets)
|   +-- ... (7 more dimensionality folders)
|   +-- largexlarge/
|       +-- ... (4 datasets)
|
+-- processed_regression/
    +-- smallxsmall/
    |   +--
obs_smallxsmall_Noisy_False_Nonlinear_False_Type_Regression_ID572.csv
    |   +--
obs_smallxsmall_Noisy_False_Nonlinear_True_Type_Regression_ID41187.csv
    |   +--
obs_smallxsmall_Noisy_True_Nonlinear_False_Type_Regression_ID46161.csv
    |   +--
obs_smallxsmall_Noisy_True_Nonlinear_True_Type_Regression_ID189.csv
    +-- ... (8 more dimensionality folders)
```

Phase 5: Standardize Target Variables

Script: `standardize_target_variables.py`

```
# Preview changes (dry run - RECOMMENDED FIRST)
python standardize_target_variables.py --dry-run

# Apply standardization
python standardize_target_variables.py

# Or by task type
python standardize_target_variables.py --task classification
python standardize_target_variables.py --task regression
```

What It Does

1. Scans all CSVs in `data/processed_classification/` and `data/processed_regression/`
2. Identifies the target column (uses heuristics or config)
3. Renames target column to `TARGET_VARIABLE`
4. Overwrites CSV files with updated column names

Output

All 72 datasets now have standardized `TARGET_VARIABLE` column, ready for experiments.

Troubleshooting

“Dataset not found on OpenML”

Some OpenML datasets may be unavailable or have changed IDs. Check:

```
import openml
dataset = openml.datasets.get_dataset(DATASET_ID)
print(dataset.name)
```

“Target column not detected”

When using `dataset_editor.py`, manually select the correct target column by number.

Wrong transformations applied

Re-run `dataset_editor.py` for the specific dataset: 1. Load the raw version from `data/raw_downloads/` 2. Apply correct transformations 3. Save to `data/processed_*/` folder

Size verification fails

Check the size reduction scripts: - `reduce_small_datasets.py` - Caps small datasets at 1,000 - `reduce_medium_datasets.py` - Caps medium datasets at 5,000 - `reduce_large_datasets.py` - Caps large datasets at 15,000

Missing datasets

Verify you have exactly: - 36 classification datasets (4 per dimensionality x 9 = 36) - 36 regression datasets (4 per dimensionality x 9 = 36)

Time Estimates

Phase	Time
Phase 1 (Download)	10-30 minutes
Phase 2 (Identify)	5-10 minutes
Phase 3 (Transform)	2-4 hours (72 interactive sessions)
Phase 4 (Organize)	30 minutes
Phase 5 (Standardize)	1-2 minutes
Total	**3-5 hours**

Next Steps

After completing data preparation:

1. Verify all datasets: `python verify_datasets.py`
2. Proceed to experiments: See `COMPLETE_REPRODUCTION_GUIDE.md`

Document Version: 2025-12-08 **Random Seed**: 42 throughout all operations