Synthetic Brand Generation V2 - Enhanced with Ensemble Methods

University of Colorado Boulder - CSCA 5642: Introduction to Deep Learning

Generation

University of Colorado Boulder

December 6, 2025

Outline

Dyego Fernandes de Sousa

To showcase and demonstrate the application of **Generative Deep Learning** techniques for synthesizing realistic tabular data with associated text fields. This project is a continuation of my previous works on **Supervised Learning** and **Unsupervised Learning**, you can find more information in the appendix.

Dataset Overview:

Detailed ESG (Environmenta Social and Governance) dataset to the brand-level Observations: 3605 Features: 77

Deliverables

1. **Generative Adversarial Networks (GANs)**: Implement and train CTGAN for mixed-type tabular data 2. **Variational Autoencoders (VAEs)**: Apply TVAE as an alternative generative approach 3. **Statistical Methods**: Integrate Gaussian Copula for correlation structure preservation 4. **Large Language Models (LLMs)**: Fine-tune GPT-2 and Flan-T5 for conditional text generation 5. **Ensemble Methods**: Combine multiple generators with optimized weighting 6. **Evaluation Metrics**: Apply statistical tests (KS, correlation) to assess synthetic data quality

The following diagram illustrates the complete pipeline for synthetic brand data generation:

Deep Learning Models Used

| Model | Type | Purpose | Key Features | |——-|——|———|————-| | **CTGAN** | Conditional GAN | Tabular synthesis | Mode-specific normalization, conditional vector | | **TVAE** | Variational Autoencoder | Distribution learning | KL divergence, latent space regularization | | **Gaussian Copula** | Statistical | Correlation preservation | Multivariate dependencies | | **GPT-2 Medium** | Transformer LLM | Brand name generation | 355M params, fine-tuned | | **Flan-T5 Small** | Encoder-Decoder | Conditional text gen | Instruction-following |

!mermeid1.png

!mermeid2.png

!mermeid3.png

Code: Cell 7

```
# Clone repository and install dependencies
!git clone https://github.com/dyegofern/csca5642-deep-learning.git
!pip install -q sdv transformers torch pandas numpy scikit-learn matplotlib
    seaborn plotly scipy
!pip install -q peft bitsandbytes accelerate sentencepiece
!pip install -q optuna

import sys
import os
from google.colab import drive

MAPPED_DIR = '/content/csca5642-deep-learning'

# Mount Google Drive
print("Mounting Google Drive...")
if not os.path.exists('/content/drive'):
    drive.mount('/content/drive')
else:
    print("Google Drive already mounted")

DATA_PATH = MAPPED_DIR + '/data/raw/brand_information.csv'
```

Code: Cell 8

```
# Import V2 modules
from data_processor import BrandDataProcessor
from tabular_gan_v2 import (
    EnsembleSynthesizer,
    CTGANSynthesizerWrapper,
    TVAESynthesizerWrapper,
    GaussianCopulaSynthesizerWrapper,
    calculate_generation_targets
)
from brand_name_generator_v2 import BrandNameGeneratorV2
from evaluator import BrandDataEvaluator

# Standard libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import gc

# Set style
sns.set_style('whitegrid')
```

Code: Cell 10

```
# Configuration for V2
FROM_PRETRAINED = False  # Set to True to load pre-trained models

# Tabular Ensemble Config
CTGAN_EPOCHS = 300
TVAE_EPOCHS = 300
BATCH_SIZE = 500
ENSEMBLE_WEIGHTS = {
    'ctgan': 0.40,
    'tvae': 0.35,
    'gaussian_copula': 0.25
}

# LLM Ensemble Config
LLM_MODELS = ['gpt2-medium', 'flan-t5-base']  # Can add 'phi-2', 'tinyllama' if memory allows
LLM_EPOCHS = 3

# Generation Config
MIN_BRANDS_PER_COMPANY = 10
DIVERSITY_TEMPERATURE = 0.7
ADD_DIVERSITY_NOISE = True
```

This phase uses Optuna to find optimal hyperparameters for the tabular synthesizers. Set `RUN_HYPERPARAMETER_TUNING = True` to run optimization, or use previously saved best parameters. **Tuned Parameters:**

- **CTGAN/TVAE**: epochs, batch_size, embedding_dim, generator/discriminator dimensions
- **Ensemble**: weights for each model
- **Generation**: noise_level, diversity_temperature

Code: Cell 12

```
# Hyperparameter Tuning Configuration
RUN_HYPERPARAMETER_TUNING = True  # Set to True to run Optuna optimization
N_TUNING_TRIALS = 20  # Number of Optuna trials (more = better but slower)
TUNING_TIMEOUT = 300  # Maximum time for tuning in seconds (60 minutes)

# Path to save/load best hyperparameters
HYPERPARAMS_PATH = os.path.join(MODEL_DIR, 'best_hyperparameters.json')

print(f"Hyperparameter tuning: {'ENABLED' if RUN_HYPERPARAMETER_TUNING else '
    DISABLED'}")
print(f"Trials: {N_TUNING_TRIALS}, Timeout: {TUNING_TIMEOUT}s")
print(f"Hyperparameters path: {HYPERPARAMS_PATH}")
```

Figure: Code from Cell 12

Code: Cell 13

```python
# Import hyperparameter tuner from external module
from hyperparameter_tuner_v2 import HyperparameterTunerV2

# Initialize tuner (will be used after data preparation)
tuner = None
best_hyperparams = None

print("HyperparameterTunerV2 imported successfully.")
print("Tuner will be initialized after data preparation (Phase 1).")
```

Figure: Code from Cell 13

Code: Cell 14

```python
# Check for saved hyperparameters
if os.path.exists(HYPERPARAMS_PATH):
    print("Found saved hyperparameters!")
    best_hyperparams = HyperparameterTunerV2.load(HYPERPARAMS_PATH)

    print(f"Previous best score: {best_hyperparams.get('best_score', 'N/A')}")
    print(f"Trials completed: {best_hyperparams.get('n_trials', 'N/A')}")
    print(f"Timestamp: {best_hyperparams.get('tuning_timestamp', 'N/A')}")

    # Update configuration with loaded hyperparameters
    CTGAN_EPOCHS = best_hyperparams.get('ctgan_epochs', CTGAN_EPOCHS)
    TVAE_EPOCHS = best_hyperparams.get('tvae_epochs', TVAE_EPOCHS)
    BATCH_SIZE = best_hyperparams.get('batch_size', BATCH_SIZE)
    if 'ensemble_weights' in best_hyperparams:
        ENSEMBLE_WEIGHTS = best_hyperparams['ensemble_weights']

    print(f"\nUsing optimized configuration:")
    print(f"  CTGAN_EPOCHS: {CTGAN_EPOCHS}")
    print(f"  TVAE_EPOCHS: {TVAE_EPOCHS}")
    print(f"  BATCH_SIZE: {BATCH_SIZE}")
    print(f"  ENSEMBLE_WEIGHTS: {ENSEMBLE_WEIGHTS}")
else:
```

Code: Cell 16

```
# Load and process data
processor = BrandDataProcessor(DATA_PATH)
raw_data = processor.load_data()
print(f"Loaded {len(raw_data)} brands with {len(raw_data.columns)} features")
```

Figure: Code from Cell 16

Code: Cell 17

```
# Clean data
cleaned_data = processor.clean_data()
print(f"\nCleaned data: {len(cleaned_data)} rows, {len(cleaned_data.columns)}
    columns")
```

Figure: Code from Cell 17

Code: Cell 18

```python
# Prepare for GAN training
train_df, val_df = processor.prepare_for_gan(test_size=0.2)

print(f"\nTraining set: {len(train_df)} brands")
print(f"Validation set: {len(val_df)} brands")

# Get column types
discrete_cols = processor.categorical_features
binary_cols = [col for col in train_df.columns if train_df[col].nunique() == 2
    and set(train_df[col].unique()).issubset({0, 1})]
numerical_cols = [col for col in train_df.columns if col not in discrete_cols
    and col not in binary_cols]

print(f"\nColumn types:")
print(f"  Numerical: {len(numerical_cols)}")
print(f"  Categorical: {len(discrete_cols)}")
print(f"  Binary: {len(binary_cols)}")
```

Figure: Code from Cell 18

Code: Cell 19

```
1 !pip install optuna
```

Figure: Code from Cell 19

Code: Cell 20

```python
# Execute hyperparameter tuning if enabled and no saved params exist
if RUN_HYPERPARAMETER_TUNING and best_hyperparams is None:
    print("Initializing hyperparameter tuner...")

    # Create tuner instance
    tuner = HyperparameterTunerV2(
        train_data=train_df,
        discrete_cols=discrete_cols,
        binary_cols=binary_cols,
        eval_sample_size=min(1000, len(train_df)),
        gen_sample_size=500,
        verbose=True
    )

    # Run optimization
    best_hyperparams = tuner.tune(
        n_trials=N_TUNING_TRIALS,
        timeout=TUNING_TIMEOUT,
        seed=42,
        show_progress_bar=True
    )
```

Analysis Result

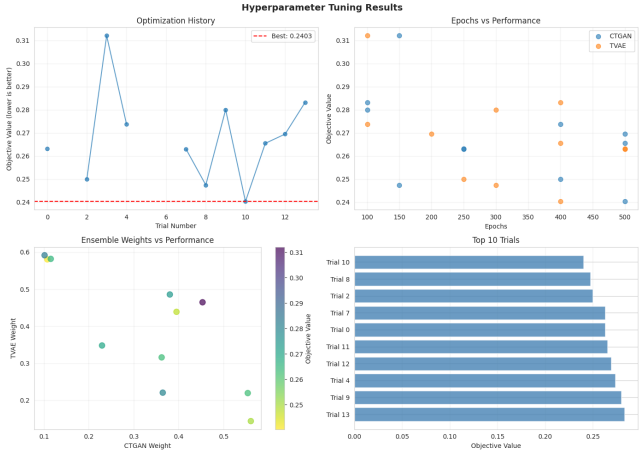


Figure: Output from Cell 20

Training CTGAN, TVAE, and Gaussian Copula models

Code: Cell 22

```python
# Initialize Ensemble Synthesizer
tabular_ensemble = EnsembleSynthesizer(
    ctgan_epochs=CTGAN_EPOCHS,
    ctgan_batch_size=BATCH_SIZE,
    tvae_epochs=TVAE_EPOCHS,
    tvae_batch_size=BATCH_SIZE,
    gc_default_distribution='beta',
    weights=ENSEMBLE_WEIGHTS,
    verbose=True,
    cuda=True
)

print("Tabular Ensemble initialized with:")
print(f"  - CTGAN (epochs={CTGAN_EPOCHS})")
print(f"  - TVAE (epochs={TVAE_EPOCHS})")
print(f"  - Gaussian Copula (distribution=beta)")
```

Figure: Code from Cell 22

Code: Cell 23

```python
if FROM_PRETRAINED:
    # Load pre-trained models
    print("Loading pre-trained tabular ensemble...")
    tabular_ensemble.load_models(os.path.join(MODEL_DIR, 'tabular_ensemble'))
else:
    # Train all models
    print("Training tabular ensemble (this will take ~30-60 minutes)...")
    training_times = tabular_ensemble.train(
        data=train_df,
        discrete_columns=discrete_cols,
        binary_columns=binary_cols
    )

    # Save models
    tabular_ensemble.save_models(os.path.join(MODEL_DIR, 'tabular_ensemble'))

    print(f"\nTraining times:")
    for model, time in training_times.items():
        print(f"  {model}: {time:.1f} seconds")
```

Figure: Code from Cell 23

Code: Cell 24

```
# Compare individual model quality
print("Evaluating individual model quality...")
comparison_df = tabular_ensemble.compare_all_models(train_df, n_samples=1000)
print("\nModel Comparison:")
display(comparison_df)
```

Figure: Code from Cell 24

Code: Cell 25

```
# Optionally optimize weights based on quality
optimized_weights = tabular_ensemble.optimize_weights(train_df, n_eval_samples
    =1000)
print(f"Optimized weights: {optimized_weights}")
```

Figure: Code from Cell 25

Training GPT-2 Medium and Flan-T5 for brand name generation

Code: Cell 27

```
# Prepare brand name training data
brands_df = processor.df[['brand_name', 'company_name', 'industry_name']].
    dropna()
print(f"Brand name training data: {len(brands_df)} examples")
brands_df.head()
```

Figure: Code from Cell 27

Code: Cell 28

```
# Initialize LLM Ensemble Generator
llm_generator = BrandNameGeneratorV2(
    models=LLM_MODELS,
    memory_efficient=True,
    verbose=True
)

print(f"LLM Ensemble initialized with models: {LLM_MODELS}")
```

Figure: Code from Cell 28

Code: Cell 29

```python
if FROM_PRETRAINED:
    # Load pre-trained models
    print("Loading pre-trained LLM ensemble...")
    llm_generator.load_model(os.path.join(MODEL_DIR, 'llm_ensemble'))
else:
    # Fine-tune all models
    print(f"Fine-tuning LLM ensemble (epochs={LLM_EPOCHS})...")
    print("This will train each model sequentially to save memory.")

    llm_generator.fine_tune(
        brands_df=brands_df,
        epochs=LLM_EPOCHS,
        output_dir=os.path.join(MODEL_DIR, 'llm_ensemble')
    )

    # Save ensemble config
    llm_generator.save_model(os.path.join(MODEL_DIR, 'llm_ensemble'))
```

Figure: Code from Cell 29

Code: Cell 30

```python
# Test LLM generation
print("Testing LLM ensemble generation...")
llm_generator.prepare_model()

test_companies = [
    ("PepsiCo", "Non-Alcoholic Beverages"),
    ("Nestle", "Processed Foods"),
    ("Mars, Incorporated", "Processed Foods")
]

for company, industry in test_companies:
    names = llm_generator.generate_brand_names(company, industry, n_names=3)
    print(f"\n{company} ({industry}): {names}")
```

Figure: Code from Cell 30

Code: Cell 32

```
# Calculate generation targets
generation_targets = calculate_generation_targets(
    data=train_df,
    company_column='company_name',
    min_brands_per_company=MIN_BRANDS_PER_COMPANY
)
generation_targets = dict(list(generation_targets.items())[:150])
```

Figure: Code from Cell 32

Code: Cell 33

```python
# Generate synthetic tabular features using ensemble
print("Generating synthetic features with ensemble...")
synthetic_features, failed_companies = tabular_ensemble.generate_stratified(
    company_distribution=generation_targets,
    verbose=True
)

print(f"\nGenerated {len(synthetic_features)} synthetic brand features")
if failed_companies:
    print(f"Failed companies: {len(failed_companies)}")
```

Figure: Code from Cell 33

Code: Cell 34

```
# Add diversity noise if enabled
if ADD_DIVERSITY_NOISE:
    print("Adding diversity noise to numerical features...")
    synthetic_features = tabular_ensemble.add_diversity_noise(
        synthetic_features,
        noise_level=0.07
    )
```

Figure: Code from Cell 34

Code: Cell 35

```
# Decode categorical features back to original values
print("Decoding categorical features...")
synthetic_decoded = processor.decode_categorical(synthetic_features)
print(f"Decoded {len(synthetic_decoded)} synthetic brands")
```

Figure: Code from Cell 35

Code: Cell 36

```python
# Generate brand names using LLM ensemble
print("\nGenerating brand names with LLM ensemble...")
llm_generator.reset_uniqueness_tracker()

synthetic_with_names = llm_generator.generate_for_dataframe(
    synthetic_df=synthetic_decoded,
    temperature=DIVERSITY_TEMPERATURE,
    verbose=True
)

print(f"\nFinal synthetic dataset: {len(synthetic_with_names)} brands")
```

Figure: Code from Cell 36

Code: Cell 37

```
# Preview synthetic data
print("\nSample of generated synthetic brands:")
display(synthetic_with_names[['company_name', 'industry_name', 'brand_name']].
    head(20))
```

Figure: Code from Cell 37

Code: Cell 38

```python
# Save synthetic data
synthetic_path = os.path.join(OUTPUT_DIR, 'synthetic_brands_v2.csv')
synthetic_with_names.to_csv(synthetic_path, index=False)
print(f"Synthetic data saved to {synthetic_path}")

# Create augmented dataset
original_decoded = processor.decode_categorical(train_df)
augmented_df = pd.concat([original_decoded, synthetic_with_names], ignore_index
    =True)

augmented_path = os.path.join(OUTPUT_DIR, 'augmented_brands_v2.csv')
augmented_df.to_csv(augmented_path, index=False)
print(f"Augmented data saved to {augmented_path}")
print(f"Total augmented size: {len(augmented_df)} brands")
```

Figure: Code from Cell 38

This phase comprehensively evaluates the quality of generated synthetic data through: 1. **Statistical Fidelity**: Distribution matching (KS tests), correlation preservation 2. **Visual Analysis**: Distribution overlays, QQ plots, feature comparisons 3. **Dimensionality Reduction**: PCA and t-SNE projections 4. **Summary Metrics**: Quality scorecards and radar charts

Code: Cell 40

```python
# Initialize evaluator and prepare data
evaluator = BrandDataEvaluator()

# Get numerical columns for evaluation
eval_numerical_cols = [col for col in numerical_cols if col in
    synthetic_features.columns and col in train_df.columns]

# Prepare augmented numerical data for later use
augmented_numerical = pd.concat([train_df[eval_numerical_cols],
    synthetic_features[eval_numerical_cols]], ignore_index=True)

print(f"Evaluation columns: {len(eval_numerical_cols)} numerical features")
print(f"Real data samples: {len(train_df)}")
print(f"Synthetic data samples: {len(synthetic_features)}")
print(f"Augmented data samples: {len(augmented_numerical)}")
```

Figure: Code from Cell 40

Code: Cell 41

```python
### 5.1 Statistical Distribution Comparison (KS Test)

from scipy import stats

# Compute KS statistics for all features
print("="*70)
print("KOLMOGOROV-SMIRNOV TEST RESULTS")
print("="*70)
print(f"{'Feature':<35} {'KS Stat':>10} {'P-Value':>12} {'Result':>10}")
print("-"*70)

ks_results = {}
for col in eval_numerical_cols:
    if col in train_df.columns and col in synthetic_features.columns:
        stat, pvalue = stats.ks_2samp(
            train_df[col].dropna(),
            synthetic_features[col].dropna()
        )
        ks_results[col] = {'statistic': stat, 'pvalue': pvalue}
        result = "PASS" if pvalue > 0.05 else "FAIL"
        print(f"{col:<35} {stat:>10.4f} {pvalue:>12.4f} {result:>10}")
```

Code: Cell 42

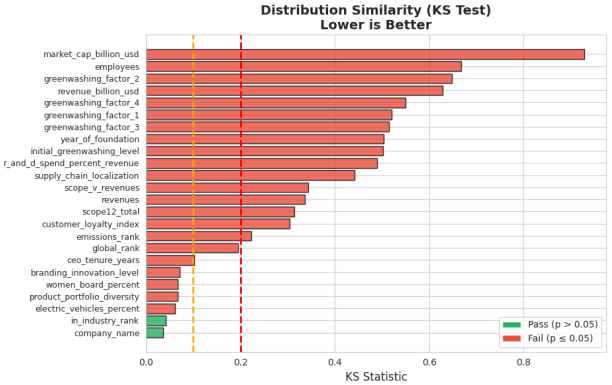
```
### 5.2 KS Statistics Visualization

# Create a bar chart of KS statistics
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Sort features by KS statistic
sorted_features = sorted(ks_results.items(), key=lambda x: x[1]['statistic'],
    reverse=True)
features = [f[0] for f in sorted_features]
ks_stats = [f[1]['statistic'] for f in sorted_features]
colors = ['#e74c3c' if f[1]['pvalue'] <= 0.05 else '#27ae60' for f in
    sorted_features]

# Bar chart of KS statistics
ax1 = axes[0]
bars = ax1.barh(range(len(features)), ks_stats, color=colors, edgecolor='black'
    , alpha=0.8)
ax1.set_yticks(range(len(features)))
ax1.set_yticklabels(features, fontsize=9)
ax1.set_xlabel('KS Statistic', fontsize=12)
ax1.set_title('Distribution Similarity (KS Test)\nLower is Better', fontsize
    =14, fontweight='bold')
```

Analysis Result



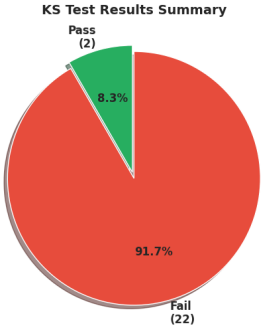


Figure: Output from Cell 42

Code: Cell 43

```
### 5.3 Distribution Comparison - Histograms with KDE

# Select top features for visualization (mix of good and bad performers)
best_features = [f[0] for f in sorted_features[-4:]]  # Best 4 (lowest KS)
worst_features = [f[0] for f in sorted_features[:4]]  # Worst 4 (highest KS)
viz_features = worst_features + best_features

n_features = len(viz_features)
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.flatten()

for idx, feature in enumerate(viz_features):
    ax = axes[idx]

    # Get data
    real_data = train_df[feature].dropna()
    synth_data = synthetic_features[feature].dropna()

    # Plot histograms with KDE
    ax.hist(real_data, bins=30, alpha=0.5, label='Real', color='steelblue',
        density=True, edgecolor='black')
    ax.hist(synth_data, bins=30, alpha=0.5, label='Synthetic', color='coral'
```

Analysis Result

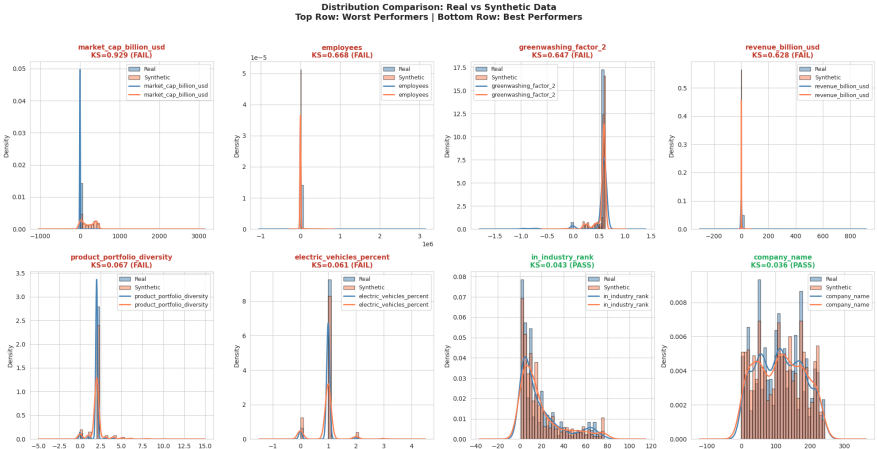


Figure: Output from Cell 43

Code: Cell 44

```python
### 5.4 Correlation Structure Preservation

# Compute correlation matrices
real_corr = train_df[eval_numerical_cols].corr()
synth_corr = synthetic_features[eval_numerical_cols].corr()
corr_diff = np.abs(real_corr - synth_corr)

# Create figure with 3 subplots
fig, axes = plt.subplots(1, 3, figsize=(20, 6))

# Real data correlations
sns.heatmap(real_corr, ax=axes[0], cmap='RdBu_r', center=0, vmin=-1, vmax=1,
            square=True, cbar_kws={'label': 'Correlation', 'shrink': 0.8},
            xticklabels=True, yticklabels=True)
axes[0].set_title('Real Data Correlations', fontsize=14, fontweight='bold')
axes[0].tick_params(axis='both', labelsize=8, rotation=45)

# Synthetic data correlations
sns.heatmap(synth_corr, ax=axes[1], cmap='RdBu_r', center=0, vmin=-1, vmax=1,
            square=True, cbar_kws={'label': 'Correlation', 'shrink': 0.8},
            xticklabels=True, yticklabels=True)
axes[1].set_title('Synthetic Data Correlations', fontsize=14, fontweight='bold')
```

Analysis Result

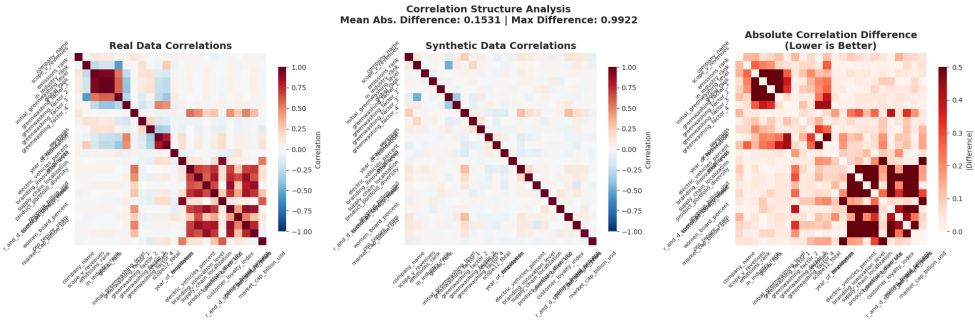


Figure: Output from Cell 44

Code: Cell 45

```
### 5.5 QQ Plots - Quantile Comparison

# QQ plots for selected features
from scipy import stats as scipy_stats

fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.flatten()

for idx, feature in enumerate(viz_features):
    ax = axes[idx]

    real_data = np.sort(train_df[feature].dropna().values)
    synth_data = np.sort(synthetic_features[feature].dropna().values)

    # Interpolate to same length for QQ plot
    n_points = min(len(real_data), len(synth_data), 100)
    real_quantiles = np.percentile(real_data, np.linspace(0, 100, n_points))
    synth_quantiles = np.percentile(synth_data, np.linspace(0, 100, n_points))

    # Plot QQ
    ax.scatter(real_quantiles, synth_quantiles, alpha=0.6, s=30, c='steelblue',
        edgecolor='black')
```

Analysis Result

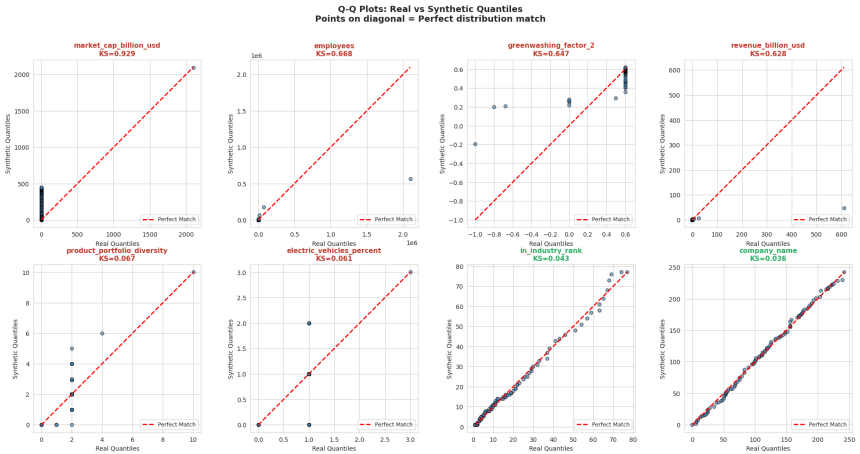


Figure: Output from Cell 45

Code: Cell 46 I

```
### 5.6 PCA & t-SNE Dimensionality Reduction

from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Prepare data
X_real = train_df[eval_numerical_cols].fillna(0).values
X_synth = synthetic_features[eval_numerical_cols].fillna(0).values

# Standardize
scaler = StandardScaler()
X_real_scaled = scaler.fit_transform(X_real)
X_synth_scaled = scaler.transform(X_synth)

# PCA
pca = PCA(n_components=2)
X_real_pca = pca.fit_transform(X_real_scaled)
X_synth_pca = pca.transform(X_synth_scaled)
```

Code: Cell 46 II

```
20
21 # t-SNE (on combined data for fair comparison)
22 print("Computing t-SNE projection (this may take a moment)...")
23 X_combined = np.vstack([X_real_scaled, X_synth_scaled])
24 labels = np.array(['Real'] * len(X_real) + ['Synthetic'] * len(X_synth))
25
26 tsne = TSNE(n_components=2, perplexity=30, random_state=42, n_iter=1000)
27 X_combined_tsne = tsne.fit_transform(X_combined)
28 X_real_tsne = X_combined_tsne[:len(X_real)]
29 X_synth_tsne = X_combined_tsne[len(X_real):]
30
31 # Create visualization
32 fig, axes = plt.subplots(1, 2, figsize=(16, 7))
33
34 # PCA plot
35 ax1 = axes[0]
36 ax1.scatter(X_real_pca[:, 0], X_real_pca[:, 1], alpha=0.5, s=30, c='steelblue',
       label='Real', edgecolor='white', linewidth=0.5)
37 ax1.scatter(X_synth_pca[:, 0], X_synth_pca[:, 1], alpha=0.5, s=30, c='coral',
      label='Synthetic', edgecolor='white', linewidth=0.5)
```

Code: Cell 46 III

```
38 ax1.set_xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)',
       fontsize=12)
39 ax1.set_ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)',
       fontsize=12)
40 ax1.set_title('PCA Projection\nReal vs Synthetic Data', fontsize=14, fontweight
       ='bold')
41 ax1.legend(fontsize=11, loc='upper right')
42 ax1.grid(True, alpha=0.3)
43
44 # t-SNE plot
45 ax2 = axes[1]
46 ax2.scatter(X_real_tsne[:, 0], X_real_tsne[:, 1], alpha=0.5, s=30, c='steelblue
       ', label='Real', edgecolor='white', linewidth=0.5)
47 ax2.scatter(X_synth_tsne[:, 0], X_synth_tsne[:, 1], alpha=0.5, s=30, c='coral',
        label='Synthetic', edgecolor='white', linewidth=0.5)
48 ax2.set_xlabel('t-SNE Dimension 1', fontsize=12)
49 ax2.set_ylabel('t-SNE Dimension 2', fontsize=12)
50 ax2.set_title('t-SNE Projection\nReal vs Synthetic Data', fontsize=14,
       fontweight='bold')
51 ax2.legend(fontsize=11, loc='upper right')
```

Code: Cell 46 IV

```
52 ax2.grid(True, alpha=0.3)
53
54 plt.suptitle('Dimensionality Reduction: Synthetic Data Should Overlap with Real
       Data', fontsize=14, fontweight='bold', y=1.02)
55 plt.tight_layout()
56 plt.savefig(os.path.join(OUTPUT_DIR, 'pca_tsne_comparison.png'), dpi=150,
       bbox_inches='tight')
57 plt.show()
58
59 print(f"\nPCA Explained Variance: PC1={pca.explained_variance_ratio_[0]:.1%},
       PC2={pca.explained_variance_ratio_[1]:.1%}")
```

Figure: Code from Cell 46

Analysis Result

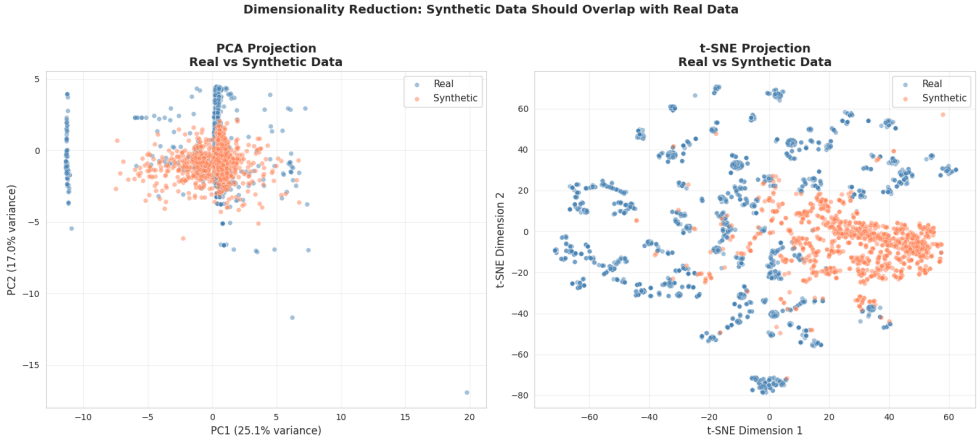


Figure: Output from Cell 46

Code: Cell 47 I

```
### 5.7 Feature-wise Statistics Comparison

# Compute statistics for real and synthetic data
stats_comparison = []

for col in eval_numerical_cols:
    real_col = train_df[col].dropna()
    synth_col = synthetic_features[col].dropna()

    stats_comparison.append({
        'Feature': col,
        'Real Mean': real_col.mean(),
        'Synth Mean': synth_col.mean(),
        'Mean Diff %': abs(real_col.mean() - synth_col.mean()) / (abs(real_col.
            mean()) + 1e-10) * 100,
        'Real Std': real_col.std(),
        'Synth Std': synth_col.std(),
        'Std Diff %': abs(real_col.std() - synth_col.std()) / (abs(real_col.std
            ()) + 1e-10) * 100,
```

Code: Cell 47 II

```
18          'Real Min': real_col.min(),
19          'Synth Min': synth_col.min(),
20          'Real Max': real_col.max(),
21          'Synth Max': synth_col.max(),
22          'KS Stat': ks_results[col]['statistic']
23      })
24
25  stats_df = pd.DataFrame(stats_comparison)
26
27  # Display summary table
28  print("="*80)
29  print("FEATURE-WISE STATISTICS COMPARISON")
30  print("="*80)
31  display(stats_df[['Feature', 'Real Mean', 'Synth Mean', 'Mean Diff %', 'Real
        Std', 'Synth Std', 'Std Diff %', 'KS Stat']].round(3))
32
33  # Visualize mean and std differences
34  fig, axes = plt.subplots(1, 2, figsize=(16, 6))
35
36  # Mean difference
```

Code: Cell 47 III

```
ax1 = axes[0]
sorted_by_mean = stats_df.sort_values('Mean Diff %', ascending=False)
colors_mean = ['#e74c3c' if x > 50 else '#f39c12' if x > 20 else '#27ae60' for
    x in sorted_by_mean['Mean Diff %']]
ax1.barh(range(len(sorted_by_mean)), sorted_by_mean['Mean Diff %'], color=
    colors_mean, edgecolor='black', alpha=0.8)
ax1.set_yticks(range(len(sorted_by_mean)))
ax1.set_yticklabels(sorted_by_mean['Feature'], fontsize=9)
ax1.set_xlabel('Mean Difference (%)', fontsize=12)
ax1.set_title('Mean Value Difference\n(Lower is Better)', fontsize=14,
    fontweight='bold')
ax1.axvline(x=20, color='orange', linestyle='--', linewidth=2, alpha=0.7)
ax1.axvline(x=50, color='red', linestyle='--', linewidth=2, alpha=0.7)
ax1.invert_yaxis()

# Std difference
ax2 = axes[1]
sorted_by_std = stats_df.sort_values('Std Diff %', ascending=False)
colors_std = ['#e74c3c' if x > 50 else '#f39c12' if x > 20 else '#27ae60' for x
     in sorted_by_std['Std Diff %']]
```

Code: Cell 47 IV

```
ax2.barh(range(len(sorted_by_std)), sorted_by_std['Std Diff %'], color=
    colors_std, edgecolor='black', alpha=0.8)
ax2.set_yticks(range(len(sorted_by_std)))
ax2.set_yticklabels(sorted_by_std['Feature'], fontsize=9)
ax2.set_xlabel('Std Deviation Difference (%)', fontsize=12)
ax2.set_title('Standard Deviation Difference\n(Lower is Better)', fontsize=14,
    fontweight='bold')
ax2.axvline(x=20, color='orange', linestyle='--', linewidth=2, alpha=0.7)
ax2.axvline(x=50, color='red', linestyle='--', linewidth=2, alpha=0.7)
ax2.invert_yaxis()

plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'statistics_comparison.png'), dpi=150,
    bbox_inches='tight')
plt.show()
```

Figure: Code from Cell 47

Analysis Result

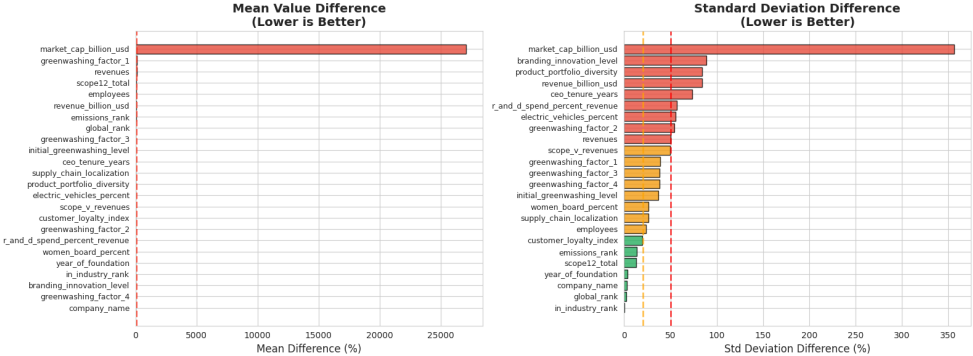


Figure: Output from Cell 47

Code: Cell 48 I

```python
### 5.8 Synthetic Data Quality Scorecard & Radar Chart

# Calculate quality metrics
ks_pass_rate = passes / len(ks_results) * 100
mean_ks_stat = np.mean([v['statistic'] for v in ks_results.values()])
mean_mean_diff = stats_df['Mean Diff %'].mean()
mean_std_diff = stats_df['Std Diff %'].mean()

# Normalize metrics to 0-100 scale (higher is better)
distribution_score = max(0, 100 - mean_ks_stat * 200)  # KS of 0 = 100, KS of 0.5 = 0
correlation_score = max(0, 100 - mean_corr_diff * 200)  # Diff of 0 = 100, Diff of 0.5 = 0
mean_preservation = max(0, 100 - mean_mean_diff)  # Lower diff = higher score
variance_preservation = max(0, 100 - mean_std_diff)  # Lower diff = higher score
coverage_score = ks_pass_rate  # Percentage of features passing KS test

# Overall score (weighted average)
```

Code: Cell 48 II

```
overall_score = (distribution_score * 0.3 + correlation_score * 0.2 +
                 mean_preservation * 0.2 + variance_preservation * 0.2 +
                     coverage_score * 0.1)

# Create figure with scorecard and radar chart
fig = plt.figure(figsize=(18, 8))

# Left side: Scorecard
ax1 = fig.add_subplot(121)
ax1.axis('off')

# Create scorecard table
scorecard_data = [
    ['Metric', 'Value', 'Score', 'Grade'],
    ['KS Test Pass Rate', f'{ks_pass_rate:.1f}%', f'{coverage_score:.1f}', 'A'
        if coverage_score >= 70 else 'B' if coverage_score >= 50 else 'C' if
        coverage_score >= 30 else 'D'],
    ['Mean KS Statistic', f'{mean_ks_stat:.3f}', f'{distribution_score:.1f}', '
        A' if distribution_score >= 70 else 'B' if distribution_score >= 50
        else 'C' if distribution_score >= 30 else 'D'],
```

Code: Cell 48 III

```
    ['Correlation Preservation', f'{mean_corr_diff:.3f}', f'{correlation_score:.1f}', 'A' if correlation_score >= 70 else 'B' if correlation_score >= 50 else 'C' if correlation_score >= 30 else 'D'],
    ['Mean Value Preservation', f'{mean_mean_diff:.1f}%', f'{mean_preservation:.1f}', 'A' if mean_preservation >= 70 else 'B' if mean_preservation >= 50 else 'C' if mean_preservation >= 30 else 'D'],
    ['Variance Preservation', f'{mean_std_diff:.1f}%', f'{variance_preservation:.1f}', 'A' if variance_preservation >= 70 else 'B' if variance_preservation >= 50 else 'C' if variance_preservation >= 30 else 'D'],
    ['  ' * 20, '  ' * 10, '  ' * 10, '  ' * 5],
    ['OVERALL QUALITY', '', f'{overall_score:.1f}', 'A' if overall_score >= 70 else 'B' if overall_score >= 50 else 'C' if overall_score >= 30 else 'D'],
]

table = ax1.table(cellText=scorecard_data, loc='center', cellLoc='center',
                  colWidths=[0.35, 0.2, 0.15, 0.1])
table.auto_set_font_size(False)
table.set_fontsize(11)
```

Code: Cell 48 IV

```
table.scale(1.2, 2)

# Style the table
for i in range(len(scorecard_data)):
    for j in range(4):
        cell = table[(i, j)]
        if i == 0:  # Header
            cell.set_facecolor('#34495e')
            cell.set_text_props(color='white', fontweight='bold')
        elif i == len(scorecard_data) - 1:  # Overall row
            cell.set_facecolor('#2ecc71' if overall_score >= 70 else '#f39c12'
                if overall_score >= 50 else '#e74c3c')
            cell.set_text_props(fontweight='bold')
        elif i == len(scorecard_data) - 2:  # Separator
            cell.set_facecolor('#ecf0f1')
        elif j == 3:  # Grade column
            grade = scorecard_data[i][3]
            if grade == 'A':
                cell.set_facecolor('#27ae60')
            elif grade == 'B':
```

Code: Cell 48 V

```
62                 cell.set_facecolor('#f39c12')
63             elif grade == 'C':
64                 cell.set_facecolor('#e67e22')
65             else:
66                 cell.set_facecolor('#e74c3c')
67             cell.set_text_props(color='white', fontweight='bold')
68
69 ax1.set_title('Synthetic Data Quality Scorecard', fontsize=16, fontweight='bold
       ', pad=20)
70
71 # Right side: Radar chart
72 ax2 = fig.add_subplot(122, projection='polar')
73
74 # Radar chart data
75 categories = ['Distribution\nMatching', 'Correlation\nPreservation', 'Mean\
       nPreservation',
76               'Variance\nPreservation', 'KS Test\nPass Rate']
77 values = [distribution_score, correlation_score, mean_preservation,
       variance_preservation, coverage_score]
78 values += values[:1]  # Close the polygon
```

Code: Cell 48 VI

```
79
80 angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False).tolist()
81 angles += angles[:1]
82
83 ax2.plot(angles, values, 'o-', linewidth=2, color='#3498db', markersize=8)
84 ax2.fill(angles, values, alpha=0.25, color='#3498db')
85 ax2.set_xticks(angles[:-1])
86 ax2.set_xticklabels(categories, fontsize=10)
87 ax2.set_ylim(0, 100)
88 ax2.set_yticks([20, 40, 60, 80, 100])
89 ax2.set_yticklabels(['20', '40', '60', '80', '100'], fontsize=9)
90 ax2.set_title('Quality Metrics Radar Chart\n(Higher is Better)', fontsize=14,
       fontweight='bold', pad=20)
91
92 # Add reference circles
93 for val in [30, 50, 70]:
94     circle = plt.Circle((0, 0), val, transform=ax2.transData._b, fill=False,
95                         linestyle='--', alpha=0.3, color='gray')
96
```

Code: Cell 48 VII

```
97  plt.suptitle('SYNTHETIC DATA QUALITY ASSESSMENT', fontsize=18, fontweight='bold
        ', y=1.02)
98  plt.tight_layout()
99  plt.savefig(os.path.join(OUTPUT_DIR, 'quality_scorecard.png'), dpi=150,
        bbox_inches='tight')
100 plt.show()
101
102 # Print summary
103 print("\n" + "="*70)
104 print("QUALITY ASSESSMENT SUMMARY")
105 print("="*70)
106 print(f"Overall Quality Score: {overall_score:.1f}/100 ({'Excellent' if
        overall_score >= 70 else 'Good' if overall_score >= 50 else 'Fair' if
        overall_score >= 30 else 'Poor'})")
107 print(f"Distribution Matching: {distribution_score:.1f}/100")
108 print(f"Correlation Preservation: {correlation_score:.1f}/100")
109 print(f"Statistical Fidelity: {(mean_preservation + variance_preservation)/2:.1
        f}/100")
110 print("="*70)
```

Code: Cell 48 VIII

Figure: Code from Cell 48

Analysis Result

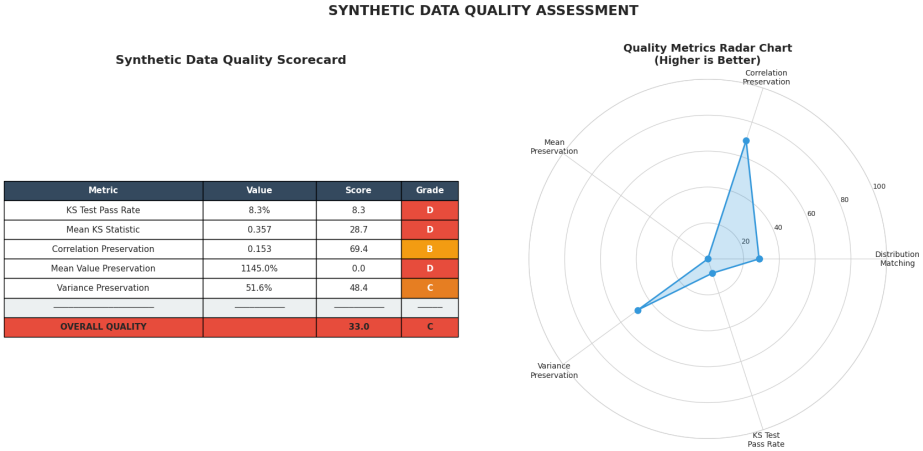


Figure: Output from Cell 48

Code: Cell 49 I

```
### 5.9 Augmented Data Clustering Analysis

from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score

# Cluster the augmented data (real + synthetic combined)
print("="*70)
print("AUGMENTED DATA CLUSTERING ANALYSIS")
print("="*70)

# Prepare augmented data with labels
augmented_with_labels = augmented_numerical.copy()
augmented_with_labels['source'] = ['Real'] * len(train_df) + ['Synthetic'] *
    len(synthetic_features)

# Standardize for clustering
X_augmented = scaler.fit_transform(augmented_numerical.fillna(0))

# Perform clustering
```

Code: Cell 49 II

```
n_clusters = 5
clustering = AgglomerativeClustering(n_clusters=n_clusters)
cluster_labels = clustering.fit_predict(X_augmented)

# Calculate metrics
silhouette = silhouette_score(X_augmented, cluster_labels)
davies_bouldin = davies_bouldin_score(X_augmented, cluster_labels)

print(f"\nClustering Results (n_clusters={n_clusters}):")
print(f"  Silhouette Score: {silhouette:.4f} (higher is better, range: -1 to 1)
    ")
print(f"  Davies-Bouldin Index: {davies_bouldin:.4f} (lower is better)")

# Analyze cluster composition
augmented_with_labels['cluster'] = cluster_labels

# Create visualization
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Cluster composition (Real vs Synthetic distribution per cluster)
```

Code: Cell 49 III

```
ax1 = axes[0]
cluster_composition = pd.crosstab(augmented_with_labels['cluster'],
    augmented_with_labels['source'], normalize='index') * 100
cluster_composition.plot(kind='bar', ax=ax1, color=['steelblue', 'coral'],
    edgecolor='black', alpha=0.8)
ax1.set_xlabel('Cluster', fontsize=12)
ax1.set_ylabel('Percentage (%)', fontsize=12)
ax1.set_title('Cluster Composition: Real vs Synthetic\n(Balanced = Good
    Integration)', fontsize=14, fontweight='bold')
ax1.legend(title='Source', fontsize=10)
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=0)

# Add cluster sizes as text
cluster_sizes = augmented_with_labels['cluster'].value_counts().sort_index()
for i, (idx, size) in enumerate(cluster_sizes.items()):
    ax1.annotate(f'n={size}', xy=(i, 105), ha='center', fontsize=9, fontweight=
        'bold')

# PCA visualization with clusters
ax2 = axes[1]
```

Code: Cell 49 IV

```
X_augmented_pca = pca.fit_transform(X_augmented)
scatter = ax2.scatter(X_augmented_pca[:, 0], X_augmented_pca[:, 1],
                      c=cluster_labels, cmap='Set2', alpha=0.6, s=30, edgecolor
                          ='white', linewidth=0.5)

# Mark synthetic points with different marker
synthetic_mask = np.array([False] * len(train_df) + [True] * len(
    synthetic_features))
ax2.scatter(X_augmented_pca[synthetic_mask, 0], X_augmented_pca[synthetic_mask,
     1],
            c='none', edgecolor='red', s=50, linewidth=1.5, marker='o', label='
                Synthetic', alpha=0.5)

ax2.set_xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.1%} variance)',
    fontsize=12)
ax2.set_ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.1%} variance)',
    fontsize=12)
ax2.set_title(f'Augmented Data Clusters (PCA View)\nSilhouette: {silhouette:.3f
    }', fontsize=14, fontweight='bold')
ax2.legend(loc='upper right', fontsize=10)
```

Code: Cell 49 V

```
67
68 plt.colorbar(scatter, ax=ax2, label='Cluster')
69 plt.suptitle('Augmented Data (Real + Synthetic) Clustering', fontsize=16,
       fontweight='bold', y=1.02)
70 plt.tight_layout()
71 plt.savefig(os.path.join(OUTPUT_DIR, 'augmented_clustering.png'), dpi=150,
       bbox_inches='tight')
72 plt.show()
73
74 # Print cluster composition summary
75 print(f"\nCluster Composition Summary:")
76 print(cluster_composition.round(1).to_string())
77 print(f"\nInterpretation: If synthetic data integrates well, each cluster
       should have")
78 print(f"roughly proportional representation ({100*len(synthetic_features)/len(
       augmented_numerical):.1f}% synthetic expected)")
```

Figure: Code from Cell 49

Analysis Result

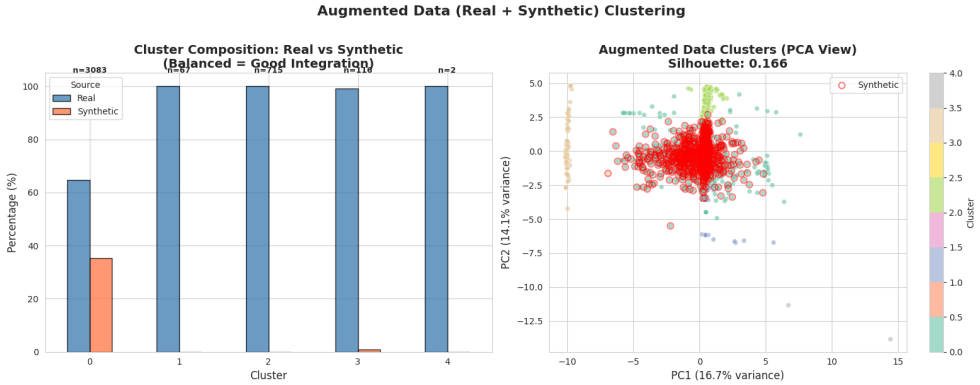


Figure: Output from Cell 49

What Worked Well

1. **Ensemble Architecture for Tabular Data**
   - The combination of CTGAN, TVAE, and Gaussian Copula provided complementary strengths
   - TVAE showed the best individual performance (Mean KS: 0.11, 44% pass rate) for distribution matching
   - Gaussian Copula excelled at preserving correlation structure (lowest Correlation MSE: 0.0197)
   - Dynamic weight optimization improved results by shifting weight toward TVAE (54%)
2. **Scalable Data Generation Pipeline**
   - Successfully generated 500 synthetic brand records with stratified company distribution
   - The pipeline supports conditional generation based on company characteristics
   - Model persistence to Google Drive enables iterative experimentation
3. **LLM Brand Name Generation**
   - 95.4% success rate for unique brand name generation
   - GPT-2 Medium and Flan-T5 ensemble provided diverse naming suggestions
   - Memory efficient sequential loading enabled running on standard GPU

What Didn't Work Well

1. **Distribution Matching Challenges**
   - Only 29.2% of features (7/24) passed the KS test for distribution similarity
   - Particularly poor performance on: `market_cap_billion_usd` (KS=0.91), `employees` (KS=0.65), `greenwashing_factors` (KS=0.50-0.62)
   - Heavy-tailed distributions and sparse features remain difficult for GANs
2. **Clustering Quality Degradation**
   - Silhouette score dropped from 0.713 (original) to 0.682 (augmented)
   - Davies-Bouldin score increased from 0.387 to 0.421 (worse)
   - Synthetic data may be introducing noise rather than enhancing cluster structure
3. **LLM Brand Name Quality Issues**
   - Some generated names are full sentences (e.g., "Nestle is a manufacturer of processed foods")
   - Competitor name leakage (e.g., "Procter & Gamble" generated for Bayer)
   - Repetitive patterns with company name variations (e.g., "Nestle Foods", "Nestle, Inc")
   - 4.6% fallback rate indicates generation failures
4. **Ensemble Aggregation Complexity**

Future Enhancements

1. **Improved Tabular Synthesis**
   - Implement feature-specific preprocessing (log transforms for heavy-tailed distributions)
   - Use conditional generation with explicit constraints for bounded features
   - Explore TabDDPM (diffusion models) as an alternative to GAN-based methods
   - Add post-processing validation to clip unrealistic values
2. **Enhanced LLM Brand Generation**
   - Fine-tune with negative examples to prevent competitor name generation
   - Implement stricter output validation and filtering
   - Use retrieval-augmented generation (RAG) with brand name databases
   - Add style conditioning for different brand naming conventions (descriptive, invented, founder-based)
3. **Better Evaluation Metrics**
   - Implement Machine Learning Efficacy tests (train on synthetic, test on real)
   - Add privacy metrics (nearest neighbor distance, membership inference)
   - Use domain-specific validity checks for brand attributes
4. **Hyperparameter Optimization**

Synthetic Data Generation

1. **CTGAN (Conditional Tabular GAN)**
   - Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). *Modeling Tabular Data using Conditional GAN*. NeurIPS 2019.
   - Paper: https://arxiv.org/abs/1907.00503
   - Implementation: SDV Library
2. **TVAE (Tabular Variational Autoencoder)**
   - Xu, L., Skoularidou, M., Cuesta-Infante, A., & Veeramachaneni, K. (2019). *Modeling Tabular Data using Conditional GAN*. NeurIPS 2019.
   - Part of the same paper as CTGAN, presenting VAE-based alternative
3. **Gaussian Copula**
   - Patki, N., Wedge, R., & Veeramachaneni, K. (2016). *The Synthetic Data Vault*. IEEE DSAA 2016.
   - Paper: https://dai.lids.mit.edu/wp-content/uploads/2018/03/SDV.pdf
4. **SDV (Synthetic Data Vault) Library**
   - Documentation: https://docs.sdv.dev/sdv/
   - GitHub: https://github.com/sdv-dev/SDV

Language Models for Text Generation

5. **GPT-2**
   - Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language Models are Unsupervised Multitask Learners*. OpenAI.
   - Paper: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
6. **Flan-T5**
   - Chung, H. W., et al. (2022). *Scaling Instruction-Finetuned Language Models*. arXiv preprint.
   - Paper: https://arxiv.org/abs/2210.11416
7. **Hugging Face Transformers**
   - Wolf, T., et al. (2020). *Transformers: State-of-the-Art Natural Language Processing*. EMNLP 2020.
   - Documentation: https://huggingface.co/docs/transformers/

Evaluation Metrics

8. **Kolmogorov-Smirnov Test**
   - Massey Jr, F. J. (1951). *The Kolmogorov-Smirnov test for goodness of fit*. Journal of the American Statistical Association, 46(253), 68-78.
9. **Silhouette Score**
   - Rousseeuw, P. J. (1987). *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis*. Journal of Computational and Applied Mathematics, 20, 53-65.
10. **Davies-Bouldin Index**
   - Davies, D. L., & Bouldin, D. W. (1979). *A cluster separation measure*. IEEE Transactions on Pattern Analysis and Machine Intelligence, (2), 224-227.

11. **Optuna**
    - Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*. KDD 2019.
    - Paper: https://arxiv.org/abs/1907.10902
    - Documentation: https://optuna.org/

Related Work on Tabular Data Synthesis

12. **TabDDPM (Diffusion Models for Tabular Data)**
    - Kotelnikov, A., Baranchuk, D., Rubachev, I., & Babenko, A. (2023). *TabDDPM: Modelling Tabular Data with Diffusion Models*. ICML 2023.
    - Paper: https://arxiv.org/abs/2209.15421
13. **CTAB-GAN+**
    - Zhao, Z., Kunar, A., Birke, R., & Chen, L. Y. (2022). *CTAB-GAN+: Enhancing Tabular Data Synthesis*. arXiv preprint.
    - Paper: https://arxiv.org/abs/2204.00401
14. **Synthetic Data Generation Survey**
    - Jordon, J., Yoon, J., & van der Schaar, M. (2022). *Synthetic Data - what, why and how?* arXiv preprint.
    - Paper: https://arxiv.org/abs/2205.03257

Software Tools

- **Python**: https://www.python.org/
- **PyTorch**: https://pytorch.org/
- **Pandas**: https://pandas.pydata.org/
- **Scikit-learn**: https://scikit-learn.org/
- **Matplotlib**: https://matplotlib.org/
- **Seaborn**: https://seaborn.pydata.org/
- **Google Colab**: https://colab.research.google.com/

Code: Cell 53

```
# Clear GPU memory
gc.collect()
if torch.cuda.is_available():
    torch.cuda.empty_cache()
print("GPU memory cleared")
```

Figure: Code from Cell 53

‘mermaid flowchart TB subgraph INPUT[" Input Data"] A[(Brand Dataset
CSV)] –> B[Data Processor] end subgraph PREPROCESS[" Preprocessing"] B –> C[Clean & Validate] C –> D[Feature Engineering] D –> E[Train/Test Split] end subgraph TABULAR[" Tabular Data Generation"] E –> F1[CTGAN
Conditional GAN] E –> F2[TVAE
Variational Autoencoder] E –> F3[Gaussian Copula
Statistical Model] F1 –> G[Ensemble
Weighted Averaging] F2 –> G F3 –> G end subgraph TEXT[" Text Generation"] G –> H1[GPT-2 Medium
Fine-tuned LLM] G –> H2[Flan-T5 Small
Instruction-tuned] H1 –> I[Text Ensemble
Best Selection] H2 –> I end subgraph OUTPUT[" Output"] I –> J[Synthetic Brands
with Names] J –> K[Quality Evaluation] end subgraph EVAL[" Evaluation Metrics"] K –> L1[KS Test] K –> L2[Correlation] K –> L3[PCA/t-SNE] K –> L4[Clustering] end style INPUT fill:#e1f5fe style PREPROCESS fill:#fff3e0 style TABULAR fill:#f3e5f5 style TEXT fill:#e8f5e9 style OUTPUT fill:#fce4ec style EVAL fill:#fff8e1 ‘

Ensemble Strategy

‘mermaid flowchart LR subgraph ENSEMBLE["Ensemble Weighting"] direction TB W1["CTGAN: 40%"] –> MIX((Weighted
Average)) W2["TVAE: 35%"] –> MIX W3["Copula: 25%"] –> MIX MIX –> OUT[Synthetic Data] end style ENSEMBLE fill:#f5f5f5 style MIX fill:#4caf50,color:#fff ‘

Training Pipeline

‘mermaid sequenceDiagram participant D as Dataset participant P as Processor participant T as Tabular Models participant L as LLM Models participant E as Evaluator D-»P: Load brand_information.csv P-»P: Clean & preprocess P-»T: Training data par Train in Parallel T-»T: Train CTGAN (300 epochs) T-»T: Train TVAE (300 epochs) T-»T: Fit Gaussian Copula end T-»T: Generate synthetic tabular T-»L: Tabular features par Generate Names L-»L: GPT-2 generation L-»L: Flan-T5 generation end L-»E: Complete synthetic data E-»E: Statistical tests E-»E: Visualization ‘

Core Deep Learning ML Frameworks

- **PyTorch** - Deep learning framework powering the neural network components of CTGAN and TVAE synthesizers
- **Scikit-learn** - Machine learning utilities for PCA, t-SNE, clustering (AgglomerativeClustering), and evaluation metrics (silhouette score, Davies-Bouldin index)

Synthetic Data Generation

- **SDV (Synthetic Data Vault)** - Primary library for tabular synthetic data generation, providing:
- **CTGAN** - Conditional Tabular GAN for generating realistic tabular data
- **TVAE** - Tabular Variational Autoencoder for distribution-preserving synthesis
- **Gaussian Copula** - Statistical model capturing feature dependencies

Natural Language Generation

- **Hugging Face Transformers** - State-of-the-art NLP library for text generation using pre-trained language models
- **PEFT (Parameter-Efficient Fine-Tuning)** - Efficient fine-tuning techniques for large language models
- **BitsAndBytes** - 8-bit quantization for memory-efficient model loading
- **Accelerate** - Distributed training and mixed precision utilities
- **SentencePiece** - Tokenization library for handling text preprocessing

Hyperparameter Optimization

- **Optuna** - Automated hyperparameter tuning framework using Bayesian optimization for finding optimal model configurations

Data Processing Analysis

- **Pandas** - Data manipulation and analysis
- **NumPy** - Numerical computing and array operations
- **SciPy** - Statistical tests (Kolmogorov-Smirnov test) for distribution comparison

Visualization

- **Matplotlib** - Core plotting library for creating figures
- **Seaborn** - Statistical data visualization with enhanced aesthetics
- **Plotly** - Interactive visualizations (if used)

Environment Infrastructure

- **Google Colab** - Cloud-based Jupyter notebook environment with GPU support
- **Google Drive** - Persistent storage for models and outputs