01_data_acquisition

December 8, 2025

1 Data Acquisition and Optimized Sampling

This notebook handles loading the Criteo Display Advertising Challenge dataset, performing basic exploratory data analysis, and creating a **fatigue-optimized sample** for analysis.

1.1 Steps:

1. Load the Criteo dataset (from HuggingFace or local file)
2. Inspect data structure and perform basic EDA
3. Analyze multi-exposure users in the full dataset
4. Create fatigue-optimized sample (enriched with multi-exposure users)
5. Verify exposure distribution in the optimized sample
6. Save sample with metadata

1.2 Why Optimized Sampling?

The original 1% random sample has **only 2.25% multi-exposure users**, which severely limits fatigue analysis. This notebook creates a **fatigue-optimized sample** that: - Identifies all multi-exposure users in the full dataset - Samples 50% from multi-exposure users (instead of 2.25%) - Samples 50% from single-exposure users (to maintain baseline) - Preserves click distribution within each group

Expected Improvement: - Multi-exposure users: 2.25% → ~50% - Reliable exposure levels: 2 → 3-6+ - Statistical power: Low → High

[1]:
```python
import sys
import os
sys.path.append('../')

import pandas as pd
import numpy as np
import json
from src.data_loader import (
    load_criteo_data,
    inspect_data_structure,
    load_config
)
from src.smart_sampling import (
    identify_multi_exposure_users,
    create_fatigue_optimized_sample,
    save_optimized_sample
)
from pathlib import Path

# Load configuration
config = load_config('../config/config.yaml')
print("Configuration loaded:")
print(f"  Random seed: {config['sampling']['random_seed']}")
```

```
Configuration loaded:
  Random seed: 42
```

1.3 Step 1: Load Criteo Dataset

The dataset can be loaded from: - HuggingFace Hub (if available) - Local file path

Note: Update the data_path below if you have the dataset locally.

[2]:
```
# Load Criteo Attribution Dataset (TSV format)
# This dataset has: timestamp, uid, campaign, click, conversion, and contextual␣
 ↪features
data_path = "../data/raw/criteo_attribution_dataset.tsv.gz"

if os.path.exists(data_path):
    print(f"Loading Criteo Attribution Dataset from: {data_path}")
    print("This dataset contains:")
    print("  - timestamp: impression timestamp")
    print("  - uid: unique user identifier")
    print("  - campaign: campaign identifier")
    print("  - click: click label (0/1)")
    print("  - conversion: conversion label (0/1)")
    print("  - cat1-9: contextual features")
    print("  - And other attribution-related fields")
    df = load_criteo_data(data_path=data_path, use_huggingface=False)
else:
    print(f"Error: Dataset not found at {data_path}")
    print("Please ensure the file exists in the data/raw directory")
```

```
Loading Criteo Attribution Dataset from:
../data/raw/criteo_attribution_dataset.tsv.gz
This dataset contains:
  - timestamp: impression timestamp
  - uid: unique user identifier
  - campaign: campaign identifier
  - click: click label (0/1)
  - conversion: conversion label (0/1)
  - cat1-9: contextual features
  - And other attribution-related fields
```

```
Loading data from ../data/raw/criteo_attribution_dataset.tsv.gz…
Detected TSV.GZ format, loading with tab separator…
Loaded 16,468,027 records
Columns: ['timestamp', 'uid', 'campaign', 'conversion', 'conversion_timestamp',
'conversion_id', 'attribution', 'click', 'click_pos', 'click_nb']…
```

1.4 Step 2: Inspect Data Structure

[3]:
```python
# Inspect data structure
if 'df' in locals():
    info = inspect_data_structure(df)
    print("Data Structure Information:")
    print(f"Shape: {info['shape']}")
    print(f"Memory usage: {info['memory_usage_mb']:.2f} MB")
    print(f"\nColumns ({len(info['columns'])}):")
    for col in info['columns'][:10]:  # Show first 10
        print(f"  - {col}: {info['dtypes'][col]}")
    if len(info['columns']) > 10:
        print(f"  ... and {len(info['columns']) - 10} more columns")

    print(f"\nMissing values:")
    missing = {k: v for k, v in info['missing_values'].items() if v > 0}
    if missing:
        for col, count in list(missing.items())[:10]:
            print(f"  {col}: {count}")
    else:
        print("  No missing values")

    # Display first few rows
    print("\nFirst few rows:")
    display(df.head())
```

```
Data Structure Information:
Shape: (16468027, 22)
Memory usage: 2764.10 MB

Columns (22):
  - timestamp: int64
  - uid: int64
  - campaign: int64
  - conversion: int64
  - conversion_timestamp: int64
  - conversion_id: int64
  - attribution: int64
  - click: int64
  - click_pos: int64
  - click_nb: int64
  … and 12 more columns
```

```
Missing values:
  No missing values

First few rows:
   timestamp       uid  campaign  conversion  conversion_timestamp  \
0          0  20073966  22589171           0                    -1
1          2  24607497    884761           0                    -1
2          2  28474333  18975823           0                    -1
3          3   7306395  29427842           1               1449193
4          3  25357769  13365547           0                    -1

   conversion_id  attribution  click  click_pos  click_nb  …  \
0             -1            0      0         -1        -1  …
1             -1            0      0         -1        -1  …
2             -1            0      0         -1        -1  …
3        3063962            0      1          0         7  …
4             -1            0      0         -1        -1  …

   time_since_last_click      cat1      cat2      cat3      cat4      cat5  \
0                     -1   5824233   9312274   3490278  29196072  11409686
1                 423858  30763035   9312274  14584482  29196072  11409686
2                   8879    138937   9312274  10769841  29196072   5824237
3                     -1  28928366  26597095  12435261  23549932   5824237
4                     -1    138937  26597094  31616034  29196072  11409684

       cat6      cat7      cat8      cat9
0   1973606  25162884  29196072  29196072
1   1973606  22644417   9312274  21091111
2    138937   1795451  29196072  15351056
3   1973606   9180723  29841067  29196072
4  26597096   4480345  29196072  29196072

[5 rows x 22 columns]
```

1.5 Step 3: Create Stratified Sample

[5]:
```python
# Analyze multi-exposure users in full dataset
print("=" * 60)
print("MULTI-EXPOSURE USER ANALYSIS (FULL DATASET)")
print("=" * 60)

if 'df' in locals():
    multi_exp_users, stats = identify_multi_exposure_users(
        df, user_col='uid', campaign_col='campaign', min_exposures=2
    )
    print(f"\nTotal users: {stats['total_users']:,}")
    print(f"Multi-exposure users: {stats['multi_exposure_users']:,}")
    print(f"Percentage: {stats['pct_multi_exposure']:.2f}%")
    print(f"Max exposures: {stats['max_exposures_in_data']}")

    print(f"\nExposure distribution (max per user):")
    for exp, count in sorted(stats['exposure_distribution'].items())[:15]:
        pct = count / stats['total_users'] * 100
        print(f"  {exp} exposures: {count:,} users ({pct:.2f}%)")
else:
    print("Error: Dataset not loaded. Please run the data loading cell first.")
```

```
============================================================
MULTI-EXPOSURE USER ANALYSIS (FULL DATASET)
============================================================

Total users: 6,142,256
Multi-exposure users: 2,545,854
Percentage: 41.45%
Max exposures: 376

Exposure distribution (max per user):
  1 exposures: 3,596,402 users (58.55%)
  2 exposures: 1,123,493 users (18.29%)
  3 exposures: 510,431 users (8.31%)
  4 exposures: 281,325 users (4.58%)
  5 exposures: 172,172 users (2.80%)
  6 exposures: 113,667 users (1.85%)
  7 exposures: 78,840 users (1.28%)
  8 exposures: 56,946 users (0.93%)
  9 exposures: 41,578 users (0.68%)
  10 exposures: 31,420 users (0.51%)
```

1.6 Step 4: Create Fatigue-Optimized Sample

Now we create a fatigue-optimized sample that enriches the dataset with multi-exposure users while maintaining a balanced representation.

[6]:
```
# Create optimized sample
print("=" * 60)
print("CREATING FATIGUE-OPTIMIZED SAMPLE")
print("=" * 60)

if 'df' in locals():
    sample, metadata = create_fatigue_optimized_sample(
        df,
        user_col='uid',
        campaign_col='campaign',
        click_col='click',
        target_sample_size=500000,  # Target sample size
        multi_exp_ratio=0.5,        # 50% from multi-exposure users
        min_exposures=2,
        random_seed=config['sampling']['random_seed']
    )

    print(f"\n Sample created successfully!")
    print(f"   Sample size: {metadata['sample_size']:,}")
    print(f"   Multi-exposure users: {metadata['sample_multi_exp_pct']:.1f}%")
    print(f"   Click rate: {metadata['sample_click_rate']:.4f}")
else:
    print("Error: Dataset not loaded. Please run the data loading cell first.")
```

```
============================================================
CREATING FATIGUE-OPTIMIZED SAMPLE
============================================================
Creating fatigue-optimized sample…
  Target size: 500,000
  Multi-exposure ratio: 50%

  Multi-exposure users in full data: 2,545,854 (41.4%)
  Multi-exposure records: 12,393,844
  Single-exposure records: 4,074,183

  Sampled 251,881 multi-exposure records
  Sampled 248,119 single-exposure records

  Final sample: 500,000 records
  Multi-exposure users: 51,353 (17.3%)
  Click rate: 0.3334

 Sample created successfully!
   Sample size: 500,000
   Multi-exposure users: 17.3%
   Click rate: 0.3334
```

1.7 Step 5: Verify Exposure Distribution in Optimized Sample

We verify that the optimized sample has sufficient data at multiple exposure levels for reliable fatigue analysis.

[8]:
```
# Compute exposure counts in the optimized sample
if 'sample' in locals():
    sample_sorted = sample.sort_values(['uid', 'campaign', 'timestamp'])
    sample_sorted['exposure_count'] = (
        sample_sorted.groupby(['uid', 'campaign']).cumcount() + 1
    )
    exp_dist = sample_sorted['exposure_count'].value_counts().sort_index()

    print("EXPOSURE DISTRIBUTION IN OPTIMIZED SAMPLE:")
    print("-" * 50)
    print(f"{'Exposure':<15} {'Records':<15} {'% of Total':<15}")
    print("-" * 50)

    for exp in exp_dist.index[:15]:
        count = exp_dist[exp]
        pct = count / len(sample_sorted) * 100
        reliable = "Reliable" if count >= 100 else ""
        print(f"{exp:<15} {count:,}          {pct:.2f}%     {reliable}")

    reliable_levels = sum(exp_dist >= 100)
    print(f"\nReliable exposure levels (n>=100): {reliable_levels}")
    print(f"Max exposure level: {exp_dist.index.max()}")
else:
    print("Error: Sample not created. Please run the sampling cell first.")
```

```
EXPOSURE DISTRIBUTION IN OPTIMIZED SAMPLE:
--------------------------------------------------
Exposure        Records         % of Total
--------------------------------------------------
1               323,705          64.74%     Reliable
2               58,648          11.73%     Reliable
3               31,720          6.34%     Reliable
4               19,929          3.99%     Reliable
5               13,664          2.73%     Reliable
6               9,845          1.97%     Reliable
7               7,353          1.47%     Reliable
8               5,717          1.14%     Reliable
9               4,535          0.91%     Reliable
10              3,607          0.72%     Reliable
11              2,952          0.59%     Reliable
12              2,431          0.49%     Reliable
13              1,983          0.40%     Reliable
14              1,675          0.34%     Reliable
15              1,393          0.28%     Reliable

Reliable exposure levels (n>=100): 38
Max exposure level: 196
```

1.8 Step 6: Compare with Original Sample (if available)

If an original random sample exists, we compare it with the optimized sample to show the improvement.

[10]:
```
# Compare with original sample if it exists
original_meta_path = "../data/samples/criteo_sample_01_metadata.json"

if os.path.exists(original_meta_path) and 'metadata' in locals():
    print("=" * 60)
    print("COMPARISON: ORIGINAL vs OPTIMIZED SAMPLE")
    print("=" * 60)

    with open(original_meta_path, 'r') as f:
        original_meta = json.load(f)

    print(f"\n{'Metric':<35} {'Original':<15} {'Optimized':<15}")
    print("-" * 65)
    print(f"{'Sample size':<35} {original_meta['sample_size']:,}        ␣
↪{metadata['sample_size']:,}")

    # Estimate original multi-exp percentage (typically ~2.25% for 1% random␣
↪sample)
    original_multi_exp_pct = 2.25  # Typical value for random 1% sample
    print(f"{'Multi-exp users (%)':<35} {original_multi_exp_pct:.1f}%        ␣
↪{metadata['sample_multi_exp_pct']:.1f}%")
    print(f"{'Click rate':<35} {original_meta['click_rate']:.4f}        ␣
↪{metadata['sample_click_rate']:.4f}")

    improvement = metadata['sample_multi_exp_pct'] / original_multi_exp_pct
    print(f"\nMulti-exposure users increased by {improvement:.1f}x!")
else:
    print("Original sample metadata not found. Skipping comparison.")
```

```
============================================================
COMPARISON: ORIGINAL vs OPTIMIZED SAMPLE
============================================================

Metric                              Original        Optimized
-----------------------------------------------------------------
Sample size                         164,681         500,000
Multi-exp users (%)                 2.2%           17.3%
Click rate                          0.3612         0.3334

Multi-exposure users increased by 7.7x!
```

1.9 Step 7: Save Optimized Sample

Save the fatigue-optimized sample and its metadata for use in subsequent analysis notebooks.

[11]:
```
# Save the optimized sample
if 'sample' in locals() and 'metadata' in locals():
    sample_file, metadata_file = save_optimized_sample(
        sample,
        metadata,
        output_dir="../data/samples",
        sample_name="criteo_fatigue_optimized"
    )

    print(f"\n  Optimized sample saved!")
    print(f"   Sample file: {sample_file}")
    print(f"   Metadata file: {metadata_file}")
    print(f"\n   To use this sample in other notebooks, load it with:")
    print(f"   df = pd.read_csv('../data/samples/criteo_fatigue_optimized.
 ↪csv')")
else:
    print("Error: Sample not created. Please run the sampling cell first.")
```

```
Sample saved to ../data/samples/criteo_fatigue_optimized.csv
Metadata saved to ../data/samples/criteo_fatigue_optimized_metadata.json

  Optimized sample saved!
   Sample file: ../data/samples/criteo_fatigue_optimized.csv
   Metadata file: ../data/samples/criteo_fatigue_optimized_metadata.json

   To use this sample in other notebooks, load it with:
   df = pd.read_csv('../data/samples/criteo_fatigue_optimized.csv')
```

[ ]:

```
```