

MACHINE LEARNING OPERATIONS



Dataset_version



Presented by Asst. Prof. Dr. Tuchsana Ploysuwan



1. บทนำ: ความสำคัญของ Dataset Versioning

1.1 ปัญหาที่พบบ่อยในการจัดการข้อมูล ML

ในโปรเจกต์ Machine Learning ทัวไป ทีมมักประสบปัญหาเหล่านี้:

✗

"Model ทำงานได้ดีเมื่อวานนี้ แต่วันนี้ Accuracy ตก ไม่รู้ว่าเปลี่ยนอะไรไป"

✗

"ใช้ Data ชุดไหนในการ Train Model version ที่ deploy อยู่?"

✗

"ใครแก้ไข Dataset? เมื่อไหร่? แก้อะไร?"

✗

"ต้องการกลับไปใช้ Data version เดิม แต่ไม่มี backup"

1.2 Dataset Versioning คืออะไร?

Dataset Versioning คือกระบวนการติดตามและจัดการการเปลี่ยนแปลงของชุดข้อมูลอย่างเป็นระบบ คล้ายกับที่ Git ทำกับ Source Code

Dataset Versioning

Dataset v1.0
(1000 rows)

→

Dataset v1.1
(+500 rows)

→

Dataset v2.0
(+new column)

↓

Model v1
(Acc: 85%)

↓

Model v2
(Acc: 87%)

↓

Model v3
(Acc: 91%)

✓ Traceability: ติดตามได้ว่า Model ใช้ Data version ไหน

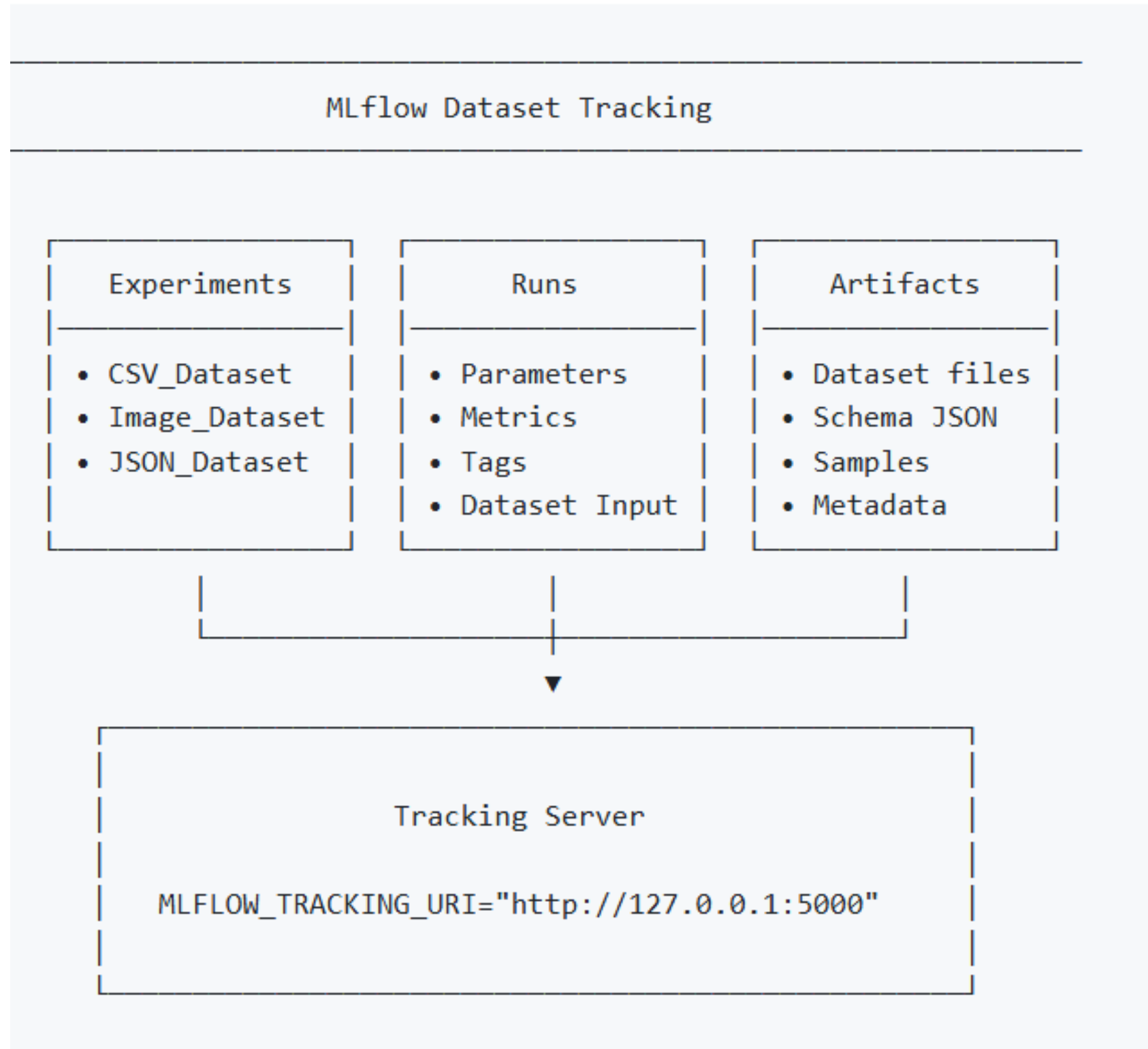
✓ Reproducibility: สามารถ reproduce ผลลัพธ์ได้

✓ Rollback: กลับไป version เดิมได้เมื่อเกิดปัญหา

1.3 ทำไมต้องใช้ MLflow สำหรับ Dataset Tracking?

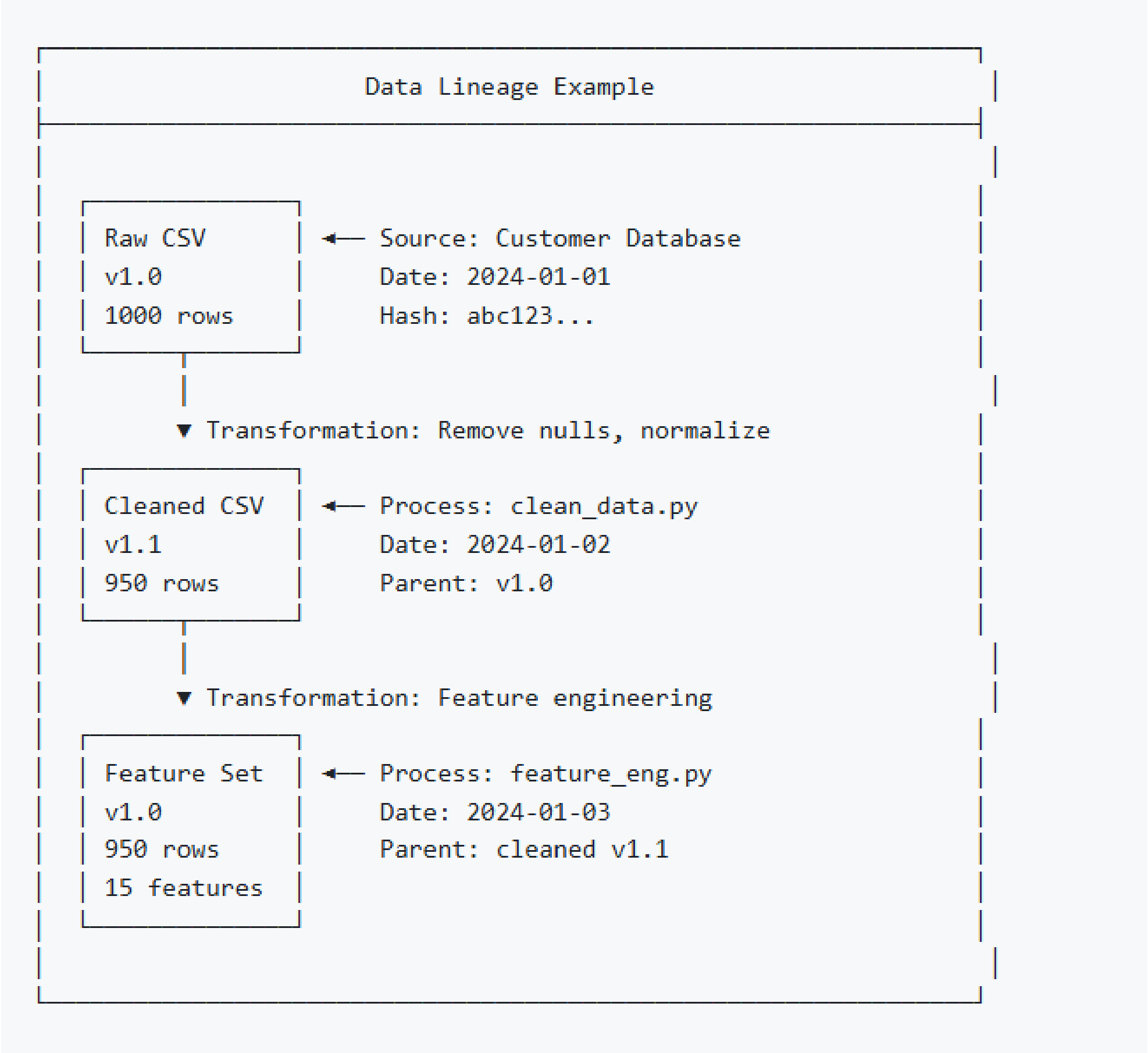
คุณสมบัติ	ประโยชน์
Centralized Tracking	เก็บข้อมูลทุกอย่างไว้ที่เดียว
Metadata Logging	บันทึก schema, statistics, hash
Artifact Storage	เก็บ Dataset files พร้อม versioning
UI Dashboard	ดูและเปรียบเทียบ versions ได้ง่าย
API Access	เข้าถึงข้อมูลผ่าน Python API

MLflow Components สำหรับ Dataset Tracking



Dataset Versioning for Csv files

Data Lineage คือการติดตามที่มาและการเปลี่ยนแปลงของข้อมูลตลอด Pipeline



2.3 Dataset Versioning Strategies

การเลือก Versioning Strategy ที่เหมาะสมเป็นสิ่งสำคัญในการจัดการ Dataset ให้มีประสิทธิภาพ แต่ละ Strategy มีข้อดีและข้อเสียที่แตกต่างกัน ขึ้นอยู่กับลักษณะของโปรเจกต์และทีม

Strategy 1: Semantic Versioning (MAJOR.MINOR.PATCH)

Semantic Versioning ใช้หลักการเดียวกับ Software Versioning ที่เป็นที่ยอมรับ โดยแบ่งเลข version เป็น 3 ส่วน

```
# Version Number Meaning:
# MAJOR: Breaking changes (schema change, column removal)
# MINOR: Backward compatible additions (new rows, new columns)
# PATCH: Bug fixes (data corrections)

version_examples = {
    "1.0.0": "Initial dataset",
    "1.1.0": "Added 500 new samples",          # Minor: more data
    "1.1.1": "Fixed typos in labels",          # Patch: corrections
    "2.0.0": "Added 'region' column",          # Major: schema change
}
```

เมื่อไหร่ควรใช้:

- เมื่อต้องการสื่อสารลักษณะการเปลี่ยนแปลงให้ทีมเข้าใจ
- เมื่อมี downstream dependencies ที่ต้องรู้ว่า data เปลี่ยนแปลงอย่างไร
- เหมาะกับ production datasets ที่มี consumers หลายคน

Strategy 2: Date-based Versioning

Date-based Versioning เหมาะกับ datasets ที่มีการ update เป็นประจำ เช่น daily snapshots หรือ monthly reports

```
# Format: YYYY-MM-DD or YYYYMMDD

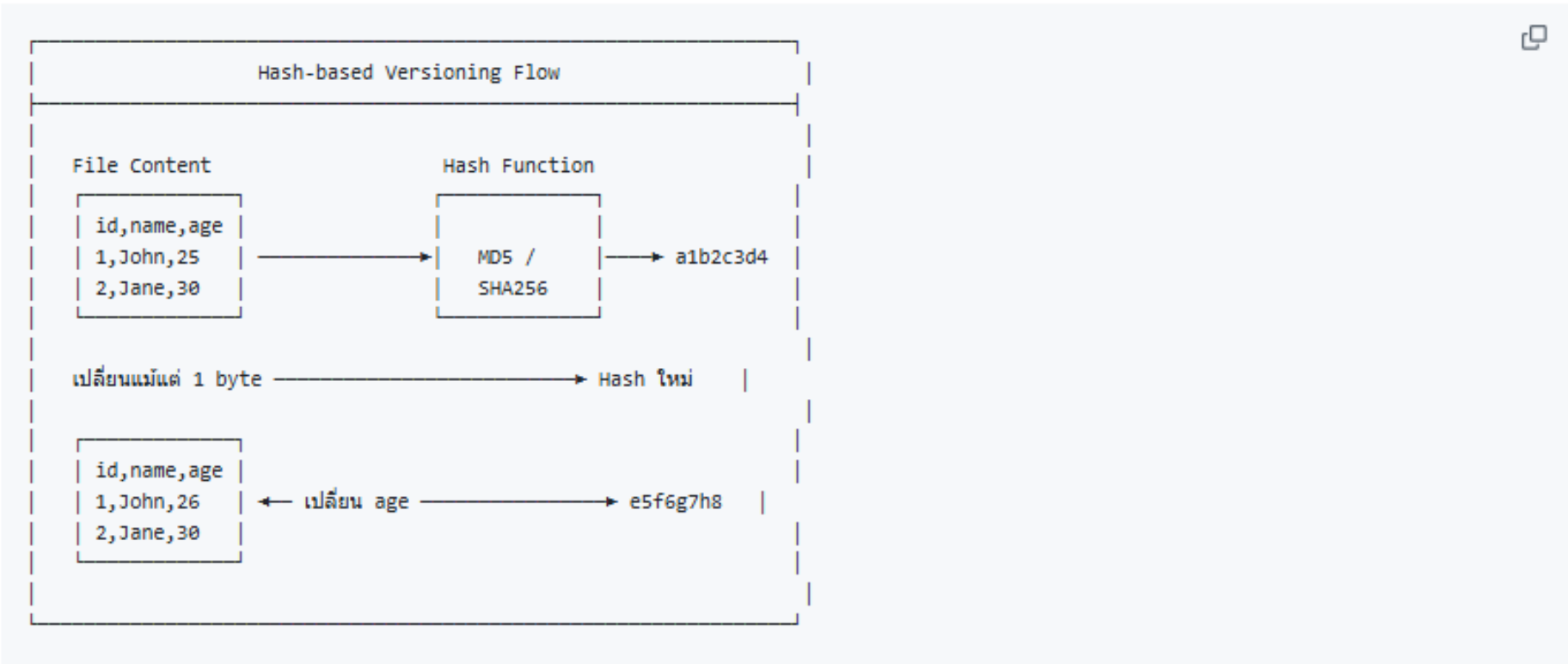
version_examples = {
    "2024-01-01": "January snapshot",
    "2024-02-01": "February snapshot",
    "2024-02-15": "Mid-month update",
}
```

เมื่อไหร่ควรใช้:

- Datasets ที่ update เป็น schedule (daily, weekly, monthly)
- Time-series data หรือ snapshot data
- เมื่อ "เวลา" เป็นข้อมูลที่สำคัญในการ track

Strategy 3: Hash-based Versioning

Hash-based Versioning ใช้ Content Hash เป็น Version Identifier ซึ่งเป็นวิธีที่ **อัตโนมัติและแม่นยำที่สุด** เนื่องจาก version จะเปลี่ยนก็ต่อเมื่อเนื้อหาของไฟล์เปลี่ยนจริงๆ เท่านั้น



ข้อดีของ Hash-based Versioning:

ข้อดี	คำอธิบาย
Automatic	ไม่ต้อง manual update version number
Unique	แต่ละ content มี hash เฉพาะตัว
Integrity Check	ตรวจสอบความถูกต้องของไฟล์ได้
Deduplication	ตรวจจับ duplicate datasets ได้
Reproducibility	มั่นใจได้ว่าใช้ data ชุดเดียวกัน

ข้อดี	คำอธิบาย
Automatic	ไม่ต้อง manual update version number
Unique	แต่ละ content มี hash เฉพาะตัว
Integrity Check	ตรวจสอบความถูกต้องของไฟล์ได้
Deduplication	ตรวจจับ duplicate datasets ได้
Reproducibility	มั่นใจได้ว่าใช้ data ชุดเดียวกัน

```
import hashlib
import pandas as pd
import os

def get_dataset_version(filepath):
    """
    สร้าง version จาก file content hash

    Args:
        filepath: path ไปยังไฟล์ dataset

    Returns:
        str: 8-character hash string เป็น version identifier
    """
    hash_md5 = hashlib.md5()
    with open(filepath, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b''):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()[:8] # ใช้ 8 characters แรก

# ตัวอย่างการใช้งาน
version = get_dataset_version("data/customers.csv")
print(f"Dataset Version: {version}") # ผลลัพธ์: "a1b2c3d4"
```

4.2 Track CSV Dataset with MLflow

Now let's create an MLflow experiment and log our CSV datasets with proper versioning and metadata.

```
[5]: def calculate_file_hash(filepath):
    """Calculate MD5 hash of a file for versioning."""
    hash_md5 = hashlib.md5()
    with open(filepath, "rb") as f:
        for chunk in iter(lambda: f.read(4096), b''):
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

def get_dataset_stats(df):
    """Generate statistics for a DataFrame."""
    stats = {
        "num_rows": len(df),
        "num_columns": len(df.columns),
        "columns": list(df.columns),
        "dtypes": {col: str(dtype) for col, dtype in df.dtypes.items()},
        "missing_values": df.isnull().sum().to_dict(),
        "memory_usage_mb": df.memory_usage(deep=True).sum() / (1024 * 1024)
    }

    # Add numeric column statistics
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    if len(numeric_cols) > 0:
        stats["numeric_stats"] = df[numeric_cols].describe().to_dict()

    return stats

print("✅ Helper functions created!")
```

✅ Helper functions created!

```
[6]: # Create experiment for CSV datasets
EXPERIMENT_NAME_CSV = "CSV_Dataset_Versioning_Lab"

# Set or create experiment
mlflow.set_experiment(EXPERIMENT_NAME_CSV)
experiment = mlflow.get_experiment_by_name(EXPERIMENT_NAME_CSV)
print(f"📁 Experiment: {EXPERIMENT_NAME_CSV}")
print(f"🆔 Experiment ID: {experiment.experiment_id}")
```

2026/02/02 08:46:47 INFO mlflow.tracking.fluent: Experiment with name 'CSV_Dataset_Versioning_Lab' does not exist. Creating a new experiment.

📁 Experiment: CSV_Dataset_Versioning_Lab
🆔 Experiment ID: 1


```
# Log CSV Dataset Version 1
print("📄 Logging CSV Dataset Version 1...")

with mlflow.start_run(run_name="customer_dataset_v1") as run:
    # Load the dataset for MLflow
    df = pd.read_csv(csv_path_v1)

    # Create MLflow dataset
    dataset = mlflow.data.from_pandas(
        df,
        source=csv_path_v1,
        name="customer_churn_dataset",
        targets="churn"
    )

    # Log the dataset
    mlflow.log_input(dataset, context="training")

    # Log dataset metadata as parameters
    mlflow.log_param("dataset_version", "1.0.0")
    mlflow.log_param("dataset_name", "customer_churn")
    mlflow.log_param("data_source", "synthetic")
    mlflow.log_param("creation_date", datetime.now().isoformat())

    # Log dataset statistics as metrics
    stats = get_dataset_stats(df)
    mlflow.log_metric("num_rows", stats["num_rows"])
    mlflow.log_metric("num_columns", stats["num_columns"])
    mlflow.log_metric("memory_mb", stats["memory_usage_mb"])
    mlflow.log_metric("missing_total", sum(stats["missing_values"].values()))
    mlflow.log_metric("churn_rate", df['churn'].mean())

    # Log the actual CSV file as artifact
    mlflow.log_artifact(csv_path_v1, artifact_path="datasets")

    # Log schema information
    schema_info = {
        "columns": stats["columns"],
        "dtypes": stats["dtypes"],
        "version": "1.0.0"
    }
    schema_path = os.path.join(CSV_DIR, "schema_v1.json")
    with open(schema_path, 'w') as f:
        json.dump(schema_info, f, indent=2)
    mlflow.log_artifact(schema_path, artifact_path="schema")

    # Log file hash for integrity
    file_hash = calculate_file_hash(csv_path_v1)
    mlflow.log_param("file_hash_md5", file_hash)

run_id_v1 = run.info.run_id
print(f"✅ Dataset v1 logged successfully!")
print(f"Run ID: {run_id_v1}")
print(f"File Hash: {file_hash}")
```

MLflow 3.8.1

CSV_Dataset_Versioing_Lab Machine learning

Runs Models Traces

metrics.rmse < 1 and params.model = "tree" Time created State: Active Datasets

Sort: Created Columns Group by

Run Name	Created	Dataset	Duration	Source	Models
customer_dataset_v1	24 seconds ago	customer_churn_dataset (5a57bafe)	207ms	ipykerne...	-

CSV_Dataset_Versioing_Lab > Runs >

customer_dataset_v1

Overview

Model metrics

System metrics

Traces

Artifacts

Description

No description

Metrics (5)

Search metrics

Metric	Value
num_rows	1000
num_columns	6
memory_mb	0.045902252197265625
missing_total	0
churn_rate	0.196

Parameters (5)

Search parameters

Parameter	Value
dataset_version	1.0.0
dataset_name	customer_churn
data_source	synthetic
creation_date	2026-02-02T08:49:17.216849
file_hash_md5	318f2f9e4cdf686ac61acb8f1cb3020b

```
# Log CSV Dataset Version 1
print("📄 Logging CSV Dataset Version 1...")

with mlflow.start_run(run_name="customer_dataset_v1") as run:
    # Load the dataset for MLflow
    df = pd.read_csv(csv_path_v1)

    # Create MLflow dataset
    dataset = mlflow.data.from_pandas(
        df,
        source=csv_path_v1,
        name="customer_churn_dataset",
        targets="churn"
    )

    # Log the dataset
    mlflow.log_input(dataset, context="training")

    # Log dataset metadata as parameters
    mlflow.log_param("dataset_version", "1.0.0")
    mlflow.log_param("dataset_name", "customer_churn")
    mlflow.log_param("data_source", "synthetic")
    mlflow.log_param("creation_date", datetime.now().isoformat())

    # Log dataset statistics as metrics
    stats = get_dataset_stats(df)
    mlflow.log_metric("num_rows", stats["num_rows"])
    mlflow.log_metric("num_columns", stats["num_columns"])
    mlflow.log_metric("memory_mb", stats["memory_usage_mb"])
    mlflow.log_metric("missing_total", sum(stats["missing_values"].values()))
    mlflow.log_metric("churn_rate", df['churn'].mean())

    # Log the actual CSV file as artifact
    mlflow.log_artifact(csv_path_v1, artifact_path="datasets")

    # Log schema information
    schema_info = {
        "columns": stats["columns"],
        "dtypes": stats["dtypes"],
        "version": "1.0.0"
    }
    schema_path = os.path.join(CSV_DIR, "schema_v1.json")
    with open(schema_path, 'w') as f:
        json.dump(schema_info, f, indent=2)
    mlflow.log_artifact(schema_path, artifact_path="schema")

    # Log file hash for integrity
    file_hash = calculate_file_hash(csv_path_v1)
    mlflow.log_param("file_hash_md5", file_hash)

run_id_v1 = run.info.run_id
print(f"✅ Dataset v1 logged successfully!")
print(f"Run ID: {run_id_v1}")
print(f"File Hash: {file_hash}")
```

The screenshot shows the MLflow web interface for a run named 'customer_dataset_v1'. The 'Artifacts' tab is selected, showing a directory structure with 'datasets' and 'schema'. The 'customers_v1.csv' file is highlighted under the 'datasets' folder. The path for the datasets artifact is displayed as 'mlflow-artifacts:/1/616082d841eb4236b869bfb913b744d3/artifacts/datasets'.

mlflow 3.8.1

CSV_Dataset_Versioning_Lab > Runs >

customer_dataset_v1

Overview Model metrics System metrics Traces **Artifacts**

▼ datasets

customers_v1.csv

▶ schema

datasets

Path: mlflow-artifacts:/1/616082d841eb4236b869bfb913b744d3/artifacts/datasets

```
# Log CSV Dataset Version 2
print("📁 Logging CSV Dataset Version 2...")

with mlflow.start_run(run_name="customer_dataset_v2") as run:
    # Load the dataset
    df = pd.read_csv(csv_path_v2)

    # Create MLflow dataset
    dataset = mlflow.data.from_pandas(
        df,
        source=csv_path_v2,
        name="customer_churn_dataset",
        targets="churn"
    )

    # Log the dataset
    mlflow.log_input(dataset, context="training")

    # Log dataset metadata
    mlflow.log_param("dataset_version", "2.0.0")
    mlflow.log_param("dataset_name", "customer_churn")
    mlflow.log_param("data_source", "synthetic")
    mlflow.log_param("creation_date", datetime.now().isoformat())
    mlflow.log_param("changes_from_v1", "Added region column, more samples")

    # Log statistics
    stats = get_dataset_stats(df)
    mlflow.log_metric("num_rows", stats["num_rows"])
    mlflow.log_metric("num_columns", stats["num_columns"])
    mlflow.log_metric("memory_mb", stats["memory_usage_mb"])
    mlflow.log_metric("missing_total", sum(stats["missing_values"].values()))
    mlflow.log_metric("churn_rate", df['churn'].mean())

    # Log artifact
    mlflow.log_artifact(csv_path_v2, artifact_path="datasets")

    # Log schema
    schema_info = {
        "columns": stats["columns"],
        "dtypes": stats["dtypes"],
        "version": "2.0.0"
    }
    schema_path = os.path.join(CSV_DIR, "schema_v2.json")
    with open(schema_path, 'w') as f:
        json.dump(schema_info, f, indent=2)
    mlflow.log_artifact(schema_path, artifact_path="schema")

    file_hash = calculate_file_hash(csv_path_v2)
    mlflow.log_param("file_hash_md5", file_hash)

run_id_v2 = run.info.run_id
print(f"✅ Dataset v2 logged successfully!")
print(f"Run ID: {run_id_v2}")
print(f"File Hash: {file_hash}")
```

MLflow 3.8.1 interface showing the CSV_Dataset_Versioning_Lab. The 'Runs' tab is selected, displaying a list of runs. The run 'customer_dataset_v2' is highlighted. The interface includes search filters, sorting options, and a table of runs.

Run Name	Created	Dataset	Duration	Source	Models
customer_dataset_v2	13 seconds ago	customer_churn_dataset (5305052f)	201ms	ipykerne...	-
customer_dataset_v1	54 minutes ago	customer_churn_dataset (5a57bafe)	207ms	ipykerne...	-

MLflow 3.8.1 interface showing the 'customer_dataset_v2' run details. The 'Artifacts' tab is selected, displaying a tree view of artifacts. The 'customers_v2.csv' file is highlighted. The interface includes tabs for Overview, Model metrics, System metrics, Traces, and Artifacts.

customer_dataset_v2

Overview Model metrics System metrics Traces Artifacts

datasets

customers_v2.csv

schema

Path: mlflow-artifacts:/1/1450ce3ab1cf4e4180eece864c631406/artifacts/datasets

Compare CSV Dataset Versions

```
: # Compare dataset versions
print("📊 Comparing Dataset Versions:")
print("=" * 60)

client = mlflow.tracking.MlflowClient()

# Get runs from the experiment
runs = client.search_runs(
    experiment_ids=[experiment.experiment_id],
    order_by=["start_time DESC"]
)

comparison_data = []
for run in runs:
    if "customer dataset" in run.info.run_name:
        version_info = {
            "Run Name": run.info.run_name,
            "Version": run.data.params.get("dataset_version", "N/A"),
            "Rows": run.data.metrics.get("num_rows", "N/A"),
            "Columns": run.data.metrics.get("num_columns", "N/A"),
            "Churn Rate": f"{run.data.metrics.get('churn_rate', 0):.2%}",
            "Memory (MB)": f"{run.data.metrics.get('memory_mb', 0):.2f}"
        }
        comparison_data.append(version_info)

comparison_df = pd.DataFrame(comparison_data)
print(comparison_df.to_string(index=False))
```

📊 Comparing Dataset Versions:

```
=====
      Run Name Version  Rows  Columns Churn Rate Memory (MB)
customer_dataset_v2  2.0.0 1500.0     7.0    25.67%      0.15
customer_dataset_v1  1.0.0 1000.0     6.0    19.60%      0.05
```

Dataset Versioning for Images files

