

Week 2: Data Validation

CS 203: Software Tools and Techniques for AI

Prof. Nipun Batra

IIT Gandhinagar

The Data Quality Crisis

Reality Check:

- You collected 1,000 movies from OMDb.
- You try to train a model.
- **Crash!** `ValueError: could not convert string to float: 'N/A'`

Common Issues:

1. **Missing Data:** `NULL` , `NaN` , `""` , `"N/A"` , `"_"` .
2. **Type Mismatches:** Rating `"8.5"` (string) instead of `8.5` (float).
3. **Outliers:** A movie from year `20250` (typo).
4. **Drift:** Last month's data format \neq today's data format.

The Validation Pipeline

```
graph LR
  A[Raw Data] --> B[Inspection];
  B --> C[Schema Validation];
  C --> D[Cleaning];
  D --> E[Drift Detection];
  E --> F[Verified Dataset];
  style F fill:#9f9,stroke:#333
```

Tools:

- **Inspect:** `jq` (JSON), `csvkit` (CSV).
- **Schema:** `Pydantic` (Row-level), `Pandera` (Batch-level).
- **Clean:** `pandas`.

Part 1: Inspection (CLI Tools)

Look at your data *before* loading it into Python.

jq: JSON Power Tool

Find Broken Records:

Movies where `BoxOffice` is missing ("N/A").

```
cat movies.json | jq '[] | select(.BoxOffice == "N/A") | .Title'
```

Check Distribution:

Count movies by Year.

```
cat movies.json | jq '[] .Year' | sort | uniq -c
```

Quick Sanity Check:

Are there any ratings > 10?

```
cat movies.json | jq '[] | select(.imdbRating | tonumber > 10)'
```

csvkit: SQL for CSVs

Statistics (`csvstat`):

Gives mean, median, null count, unique values.

```
csvstat movies.csv
```

Query (`csvsql`):

Run SQL directly on CSV!

```
csvsql --query "SELECT Title FROM movies WHERE Rating > 9" movies.csv
```

Part 2: Schema Validation (Pydantic)

The Contract: Define what valid data *looks* like.

```
from pydantic import BaseModel, Field, validator
from typing import Optional

class Movie(BaseModel):
    title: str
    year: int = Field(gt=1888, lt=2030) # Constraints
    rating: float = Field(ge=0, le=10)
    genre: str

    # Custom Validator
    @validator('genre')
    def genre_must_be_valid(cls, v):
        allowed = {'Action', 'Comedy', 'Drama'}
        if v not in allowed:
            raise ValueError(f"Unknown genre: {v}")
        return v
```

Why Pydantic?

1. **Type Coercion:** Converts `"2010"` (str) -> `2010` (int) automatically.
2. **Early Failure:** Errors catch invalid data immediately.
3. **Documentation:** The code *is* the documentation of your data format.

```
        try:
            # This will fail (Year out of bounds)
m = Movie(title="Future", year=3000, rating=5.0, genre="Action")
        except ValueError as e:
            print(e)
        # Output: 1 validation error for Movie
        # year: ensure this value is less than 2030
```

Part 3: Cleaning with Pandas

Handling Missing Data:

1. **Drop:** If label is missing, drop row. `df.dropna(subset=['rating'])`
2. **Impute:** If feature is missing, fill with mean/median. `df.fillna(df.mean())`
3. **Flag:** Create a boolean column `is_missing`.

Type Conversion:

```
# Force numeric, turn errors ('N/A') into NaN
df['rating'] = pd.to_numeric(df['rating'], errors='coerce')
```

Advanced: Batch Validation (Pandera)

Pydantic checks one object at a time.

Pandera checks the entire DataFrame (statistical checks).

```
import pandera as pa

schema = pa.DataFrameSchema({
    "rating": pa.Column(float, checks=[
        pa.Check.ge(0),
        pa.Check.le(10),
        # Mean rating should be reasonable
        pa.Check.mean_in_range(5, 9)
    ]),
    "year": pa.Column(int, checks=pa.Check.gt(1900)),
})

schema.validate(df)
```

Part 4: Data Drift

The Silent Killer.

Data valid today might be invalid tomorrow *statistically*.

- **Schema Drift:** Source API changes field name (`imdbRating` -> `rating`).
- **Data Drift:** Input distribution changes (e.g., users start reviewing only bad movies).
- **Concept Drift:** The relationship between X and Y changes (e.g., "Horror" movies become popular in Summer).

Detection:

- Compare training data stats vs production data stats.
- Tools: **Evidently AI, Alibi Detect** (Covered in Week 14).

Summary: The Checklist

Before training a model, ask:

1. ☐ **Schema:** Do all fields exist with correct types? (Pydantic)
2. ☐ **Nulls:** How are missing values handled?
3. ☐ **Ranges:** Are numbers within physical bounds? (Age > 0)
4. ☐ **Duplicates:** Are primary keys unique?
5. ☐ **Stats:** Does the distribution look normal? (Panderas)

Lab: You will build a script that takes a raw JSON dump and produces a clean, validated CSV ready for training.