compare_datasets

July 19, 2026

1 Compare Datasets: Field Matching & Formatting Report

Compares fields across two or more datasets (CSV, TSV, or Excel) to find likely matching keys — even when they're named differently — and flags formatting mismatches on the fields that match.

Matching combines two signals: - **Name similarity**: how alike the column names are (e.g. `cust_id` vs `customer_number`) after normalizing case/punctuation. - **Value overlap**: how much the actual values in the two columns overlap (Jaccard similarity on a sample of unique values). This is what confirms two columns really represent the same real-world thing. Date-like columns are parsed and normalized before comparing, so the same date in two different formats (`2024-01-01` vs `01/01/2024`) still counts as a match.

Run the cells top to bottom. The demo at the bottom uses synthetic data so you can see it work immediately — scroll to **Run it on your own files** to point it at real data.

[1]:
```
import re
from difflib import SequenceMatcher
from pathlib import Path

import pandas as pd
from openpyxl.styles import Font, PatternFill, Alignment
from openpyxl.utils import get_column_letter

pd.set_option("display.max_colwidth", 60)
```

1.1 Loading datasets

Reads CSV, TSV, or Excel files, always as strings — formatting problems (leading zeros, date strings, casing) are exactly what we're trying to catch, and letting pandas auto-convert types would hide them before we even get a look.

[2]:
```
def load_dataset(path: Path) -> pd.DataFrame:
    suffix = path.suffix.lower()
    if suffix == ".csv":
        return pd.read_csv(path, dtype=str, keep_default_na=True)
    if suffix == ".tsv":
        return pd.read_csv(path, sep="\t", dtype=str, keep_default_na=True)
    if suffix in (".xlsx", ".xls"):
        return pd.read_excel(path, dtype=str)
```

```
    raise ValueError(f"Unsupported file type: {path.suffix} ({path})")
```

1.2 Name similarity

Normalizes column names (lowercase, strip punctuation) so `Customer_ID`, `customer id`, and `CustomerID` all compare equal, then scores similarity with `difflib.SequenceMatcher`.

[3]:
```
def normalize_name(name: str) -> str:
    return re.sub(r"[^a-z0-9]", "", str(name).lower())


def name_similarity(a: str, b: str) -> float:
    na, nb = normalize_name(a), normalize_name(b)
    if not na or not nb:
        return 0.0
    return SequenceMatcher(None, na, nb).ratio()
```

1.3 Value overlap

Jaccard similarity between the sets of unique values in each column — the strongest evidence that two fields represent the same thing, since names alone can be misleading (or coincidentally similar with no real relationship).

[4]:
```
def value_overlap(series_a: pd.Series, series_b: pd.Series, sample_size: int)␣
 ↪-> float:
    a_vals = set(series_a.dropna().astype(str).str.strip().str.lower().
 ↪unique()[:sample_size])
    b_vals = set(series_b.dropna().astype(str).str.strip().str.lower().
 ↪unique()[:sample_size])
    if not a_vals or not b_vals:
        return 0.0
    intersection = a_vals & b_vals
    union = a_vals | b_vals
    return len(intersection) / len(union) if union else 0.0
```

1.4 Format detection

Best-effort classification of what a column's values look like: which date format, numeric with leading zeros, upper/lower/title-case text, and so on. Used both to flag formatting mismatches and to power the date-aware overlap check below.

[5]:
```
DATE_PATTERNS = {
    "YYYY-MM-DD": r"^\d{4}-\d{1,2}-\d{1,2}$",
    "MM/DD/YYYY": r"^\d{1,2}/\d{1,2}/\d{4}$",
    "YYYY/MM/DD": r"^\d{4}/\d{1,2}/\d{1,2}$",
    "DD-MM-YYYY": r"^\d{1,2}-\d{1,2}-\d{4}$",
}
```

```
def detect_format(series: pd.Series) -> str:
    sample = series.dropna().astype(str).str.strip()
    sample = sample[sample != ""].head(300)
    if sample.empty:
        return "empty"

    for label, pattern in DATE_PATTERNS.items():
        if sample.str.match(pattern).mean() > 0.8:
            return f"date:{label}"

    if sample.str.match(r"^0\d+$").mean() > 0.5:
        return "numeric_leading_zeros"

    numeric_frac = pd.to_numeric(sample, errors="coerce").notna().mean()
    if numeric_frac > 0.9:
        has_decimal = sample.str.contains(r"\.").mean() > 0.3
        return "numeric_decimal" if has_decimal else "numeric_integer"

    alpha_sample = sample[sample.str.contains(r"[A-Za-z]", na=False)]
    if not alpha_sample.empty:
        if alpha_sample.str.isupper().mean() > 0.8:
            return "text_upper_case"
        if alpha_sample.str.islower().mean() > 0.8:
            return "text_lower_case"
        if alpha_sample.str.istitle().mean() > 0.8:
            return "text_title_case"

    return "text_mixed"
```

1.5 Date-aware value overlap

Plain string overlap misses two columns holding the *same* dates in *different* formats — `2024-01-01` and `01/01/2024` share zero characters in common as strings. This wrapper checks the detected format first and, if both columns are date-like, parses and compares them as actual dates instead.

This is the fix that came out of testing on synthetic data further down — the first version of this notebook missed exactly this case.

[6]:
```
def smart_value_overlap(series_a: pd.Series, fmt_a: str, series_b: pd.Series,␣
↪fmt_b: str, sample_size: int) -> float:
    if fmt_a.startswith("date") and fmt_b.startswith("date"):
        try:
            da = pd.to_datetime(series_a.dropna().astype(str), errors="coerce").
↪dropna()
            db = pd.to_datetime(series_b.dropna().astype(str), errors="coerce").
↪dropna()
            a_vals = set(da.dt.strftime("%Y-%m-%d").unique()[:sample_size])
```

```
            b_vals = set(db.dt.strftime("%Y-%m-%d").unique()[:sample_size])
            if a_vals and b_vals:
                intersection = a_vals & b_vals
                union = a_vals | b_vals
                return len(intersection) / len(union) if union else 0.0
        except Exception:
            pass
    return value_overlap(series_a, series_b, sample_size)
```

1.6 Putting it together: comparing fields across datasets

Blends name similarity (40%) and value overlap (60%) into one confidence score, checks every field pair across every dataset pair, and keeps only matches above `min_confidence`.

[7]:
```
def compare_fields(datasets: dict, min_confidence: float = 0.3, sample_size:␣
↪int = 1000) -> pd.DataFrame:
    names = list(datasets.keys())
    rows = []
    for i in range(len(names)):
        for j in range(i + 1, len(names)):
            df_a, df_b = datasets[names[i]], datasets[names[j]]
            for col_a in df_a.columns:
                for col_b in df_b.columns:
                    n_sim = name_similarity(col_a, col_b)
                    fmt_a = detect_format(df_a[col_a])
                    fmt_b = detect_format(df_b[col_b])
                    v_overlap = smart_value_overlap(df_a[col_a], fmt_a,␣
↪df_b[col_b], fmt_b, sample_size)
                    confidence = round(0.4 * n_sim + 0.6 * v_overlap, 3)
                    if confidence < min_confidence:
                        continue
                    rows.append({
                        "dataset_a": names[i], "field_a": col_a, "format_a":␣
↪fmt_a,
                        "dataset_b": names[j], "field_b": col_b, "format_b":␣
↪fmt_b,
                        "name_similarity": round(n_sim, 2),
                        "value_overlap": round(v_overlap, 2),
                        "confidence": confidence,
                        "format_mismatch": "YES" if fmt_a != fmt_b else "",
                    })
    if not rows:
        return pd.DataFrame(columns=[
            "dataset_a", "field_a", "format_a", "dataset_b", "field_b",␣
↪"format_b",
            "name_similarity", "value_overlap", "confidence", "format_mismatch",
        ])
```

```
    return pd.DataFrame(rows).sort_values("confidence", ascending=False).
↪reset_index(drop=True)
```

1.7 Writing the formatted report

Two-tab Excel output: field matches (confidence-scored, formatting mismatches highlighted red, high-confidence matches highlighted green) and a summary of each source dataset.

[8]:
```
def write_report(df: pd.DataFrame, datasets: dict, output_path) -> None:
    output_path = Path(output_path)
    with pd.ExcelWriter(output_path, engine="openpyxl") as writer:
        df.to_excel(writer, sheet_name="Field Matches", index=False)

        summary_rows = [{"dataset": name, "rows": len(d), "columns": len(d.
↪columns)}
                        for name, d in datasets.items()]
        pd.DataFrame(summary_rows).to_excel(writer, sheet_name="Datasets",␣
↪index=False)

        wb = writer.book
        ws = wb["Field Matches"]

        header_fill = PatternFill(start_color="1F4E78", end_color="1F4E78",␣
↪fill_type="solid")
        header_font = Font(color="FFFFFF", bold=True)
        mismatch_fill = PatternFill(start_color="FFC7CE", end_color="FFC7CE",␣
↪fill_type="solid")
        high_conf_fill = PatternFill(start_color="C6EFCE", end_color="C6EFCE",␣
↪fill_type="solid")

        for cell in ws[1]:
            cell.fill = header_fill
            cell.font = header_font
            cell.alignment = Alignment(horizontal="center")

        if ws.max_row > 1:
            mismatch_col = df.columns.get_loc("format_mismatch") + 1
            confidence_col = df.columns.get_loc("confidence") + 1
            for row_idx in range(2, ws.max_row + 1):
                if ws.cell(row=row_idx, column=mismatch_col).value == "YES":
                    ws.cell(row=row_idx, column=mismatch_col).fill =␣
↪mismatch_fill
                conf_val = ws.cell(row=row_idx, column=confidence_col).value
                if isinstance(conf_val, (int, float)) and conf_val >= 0.75:
                    ws.cell(row=row_idx, column=confidence_col).fill =␣
↪high_conf_fill
```

```
        for col_idx, col_name in enumerate(df.columns, start=1):
            values = [col_name] + list(df[col_name].astype(str)) if len(df)␣
 ↪else [col_name]
            max_len = max(len(str(v)) for v in values)
            ws.column_dimensions[get_column_letter(col_idx)].width =␣
 ↪min(max_len + 3, 40)

        ws.freeze_panes = "A2"

        ws2 = wb["Datasets"]
        for cell in ws2[1]:
            cell.fill = header_fill
            cell.font = header_font

    print(f"Report written to {output_path}")
```

1.8 Demo: synthetic data

Two small datasets describing the same 20 people, named and formatted differently on purpose — this is the same test data that caught the date-format bug the "Date-aware value overlap" cell above now fixes. Run this to see the tool work end to end before pointing it at real files.

[9]:
```
import random
random.seed(0)

customers = pd.DataFrame({
    "cust_id": [f"{i:05d}" for i in range(1, 21)],
    "Full_Name": ["Alice Smith","Bob Jones","Carol White","Dan Brown","Eve␣
 ↪Black",
                  "Frank Green","Grace Lee","Hank Kim","Ivy Chan","Jack Wu",
                  "Kate Reed","Leo Park","Mia Diaz","Noah Cruz","Olga Petrov",
                  "Pete Ortiz","Quinn Fox","Rita Vale","Sam Torres","Tina Ng"],
    "signup_date": pd.date_range("2024-01-01", periods=20, freq="7D").
 ↪strftime("%Y-%m-%d"),
    "zip": ["02138","10001","90210","60614","73301"] * 4,
})

clients = pd.DataFrame({
    "client_number": [f"{i:05d}" for i in range(1, 21)],
    "name": customers["Full_Name"],
    "date_joined": pd.date_range("2024-01-01", periods=20, freq="7D").
 ↪strftime("%m/%d/%Y"),  # different date format
    "postal_code": ["02138","10001","90210","60614","73301"] * 4,
    "status": ["active", "inactive"] * 10,  # no match in customers
})

demo_datasets = {"customers": customers, "clients": clients}
```

```
demo_df = compare_fields(demo_datasets, min_confidence=0.3, sample_size=1000)
demo_df
```

[9]:

	dataset_a	field_a	format_a	dataset_b	field_b	format_b	name_similarity	value_overlap	confidence	format_mismatch
0	customers	Full_Name	text_title_case	clients	name	text_title_case	0.67	1.0	0.867	
1	customers	signup_date	date:YYYY-MM-DD	clients	date_joined	date:MM/DD/YYYY	0.40	1.0	0.760	YES
2	customers	cust_id	numeric_leading_zeros	clients	client_number	numeric_leading_zeros	0.22	1.0	0.689	
3	customers	zip	numeric_integer	clients	postal_code	numeric_integer	0.15	1.0	0.662	

Four real matches should come back, including `signup_date` ↔ `date_joined` flagged as a format mismatch (different date formats, same underlying dates) — and `status` should **not** appear, since it has no counterpart in `customers`.

[10]:
```
write_report(demo_df, demo_datasets, "demo_field_comparison_report.xlsx")
```

```
Report written to demo_field_comparison_report.xlsx
```

1.9 Run it on your own files

Replace the paths below with your own CSV/TSV/Excel files (two or more), then run this cell.

[ ]:
```
FILES = [
    "path/to/your_first_file.csv",
    "path/to/your_second_file.csv",
]
OUTPUT = "field_comparison_report.xlsx"
MIN_CONFIDENCE = 0.3
SAMPLE_SIZE = 1000

datasets = {}
for f in FILES:
    p = Path(f)
    datasets[p.stem] = load_dataset(p)
    print(f"Loaded {p.name}: {len(datasets[p.stem])} rows, {len(datasets[p.
 ↪stem].columns)} columns")
```

```
results_df = compare_fields(datasets, MIN_CONFIDENCE, SAMPLE_SIZE)
print(f"\nFound {len(results_df)} candidate field matches (confidence >=␣
  ↪{MIN_CONFIDENCE})")
results_df
```

[ ]:
```
write_report(results_df, datasets, OUTPUT)
```