Awesome — here's a compact, end-to-end demo that shows:

1. **What input each tool accepts**, and
2. **How to profile → clean → validate** a **mocked** dataset using **pandas**, **scikit-learn**, and **Great Expectations**.

What inputs they take

- **pandas**: tabular data → CSV/Excel/Parquet/JSON/SQL, or Python objects (dict/list/NumPy) → **DataFrame**
- **scikit-learn**: numeric arrays/matrices → `numpy.ndarray`, `scipy.sparse`, or **pandas DataFrame/Series** split as `X` (features) and `y` (target)
- **Great Expectations**: dataset wrappers over **pandas DataFrame**, **Spark DataFrame**, or **SQL table** (we'll use pandas)

Profiling → Cleaning → Validating (mocked dataset)

```
# pip install pandas scikit-learn great-expectations
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import great_expectations as ge

# ----------------------------
# 0) MOCK A SMALL DATASET (INPUT FOR PANDAS, SKLEARN, GE)
# ----------------------------
rng = np.random.RandomState(42)
df_raw = pd.DataFrame({
```

```
    "order_id": range(1, 11),
    "status":
["pending","processing","shipped","pending","processing","pending","shipped","pending", None,
"processing"],
    "amount": [120.5, 999.1, np.nan, 520.0, 160.0, 12000.0, 80.0, 510.0, 600.0, 150.0],
    "country": ["IN","US","US","IN","GB","IN","US",None,"US","IN"]
})
# df_raw is a pandas.DataFrame (pandas input)

# ----------------------------
# 1) PROFILING (PANDAS)
# ----------------------------
# Quick numeric profile
profile_num = df_raw.describe(include=[np.number])

# Quick categorical profile
profile_cat = df_raw.describe(include=["object"])

# Missingness & unique counts
missing = df_raw.isna().sum()
unique_counts = df_raw.nunique(dropna=True)

print("=== Numeric profile ===\n", profile_num, "\n")
print("=== Categorical profile ===\n", profile_cat, "\n")
print("=== Missing values per column ===\n", missing, "\n")
print("=== Unique values per column ===\n", unique_counts, "\n")

# (Optional) simple rule-of-thumb outlier view
q1, q3 = df_raw["amount"].quantile([0.25, 0.75])
iqr = q3 - q1
upper_fence = q3 + 1.5 * iqr
print("=== Amount outlier fence (Tukey) ===", upper_fence)

# ----------------------------
# 2) CLEANING (SCIKIT-LEARN PIPELINE)
# ----------------------------
# SKLearn expects X (features) and y (target). We'll treat 'status' as a feature to encode
# and (optionally) pretend we need to predict "is_high_value" just to show preprocessing.
df = df_raw.copy()

# Example derived target (just for demonstration of X/y shapes)
df["is_high_value"] = (df["amount"] > 500).astype(int)

# Define feature types
```

```
numeric_features = ["amount"]
categorical_features = ["status", "country"]
target = "is_high_value"

X = df[numeric_features + categorical_features]  # pandas DataFrame (valid input to
scikit-learn)
y = df[target]                                   # pandas Series (valid input to scikit-learn)

# Build preprocessors:
num_imputer = SimpleImputer(strategy="median")   # handles NaN in numeric
cat_imputer = SimpleImputer(strategy="most_frequent")  # handles None/NaN in categorical
ohe = OneHotEncoder(handle_unknown="ignore")

preprocessor = ColumnTransformer(
   transformers=[
      ("num", num_imputer, numeric_features),
      ("cat", Pipeline([("impute", cat_imputer), ("ohe", ohe)]), categorical_features),
   ],
   remainder="drop"
)

# Fit the preprocessing on data (this creates clean numeric matrix for modeling)
X_clean = preprocessor.fit_transform(X)
print("=== Cleaned feature matrix shape ===", X_clean.shape)
# X_clean is a NumPy/sparse matrix (classic scikit-learn input)

# (If you wanted a model, you'd continue with .fit on a classifier/regressor here.)

# ------------------------------
# 3) VALIDATION (GREAT EXPECTATIONS ON PANDAS DF)
# ------------------------------
# Validate the *business/data quality* properties on the original or cleaned data.
# Great Expectations expects a pandas DataFrame (or Spark/SQL). We'll validate df_raw and
df.
gdf = ge.from_pandas(df)  # wrap a pandas DataFrame

# Example expectations:
results = {
   "no_null_order_id": gdf.expect_column_values_to_not_be_null("order_id"),
   "amount_between": gdf.expect_column_values_to_be_between("amount", min_value=0,
max_value=10000, mostly=0.95),
   "country_in_set": gdf.expect_column_values_to_be_in_set("country", ["IN","US","GB"]),
   "status_in_set": gdf.expect_column_values_to_be_in_set("status",
["pending","processing","shipped"]),
```

```
    "unique_order_id": gdf.expect_column_values_to_be_unique("order_id"),
}

print("=== Great Expectations Results (summaries) ===")
for name, res in results.items():
    print(name, "→", res["success"])
```

What happened above?

- **Profiling (pandas):** We used `describe()`, counts of missing/unique values, and a quick outlier fence.
    - **Input:** pandas **DataFrame** (`df_raw`)
- **Cleaning (scikit-learn):** We built a preprocessing pipeline:
    - **Numeric**: median imputation
    - **Categorical**: most-frequent imputation + one-hot encoding
    - **Result:** a clean **NumPy/sparse matrix** ready for modeling (`X_clean`)
    - **Inputs:** `X` and `y` as **pandas** objects; under the hood the transformer outputs arrays (classic scikit-learn input).
- **Validating (Great Expectations):** We asserted rules on columns (no nulls, ranges, enums, uniqueness).
    - **Input:** pandas **DataFrame** (`df` wrapped via `ge.from_pandas`)

TL;DR by library

- **pandas** ingests almost any tabular source → gives you a **DataFrame** for profiling.
- **scikit-learn** takes **DataFrames/ndarrays** split into **X** and **y**, and outputs numeric matrices after preprocessing (imputers/encoders).

- **Great Expectations** takes a **DataFrame** (or Spark/SQL) and **validates expectations** (nulls, ranges, enums, uniqueness, etc.).

Want me to also show how to **export a profiling report** (e.g., with `ydata-profiling`) or how to **turn the GE checks into a reusable Expectation Suite** you can store and run in CI?