PROG8245 Lab2 – Data Collection and Pre-processing

Group: Group1

GitHub Link: https://github.com/chence/DataCollectionPreProcessing.git

Team Members:

- Ce Chen | 9007166
- Zhuoran Zhang | 9048508

Setup

Install required libraries (run once).

```
!pip -q install pandas numpy
```

Imports

```
import re
from pathlib import Path

import numpy as np
import pandas as pd
```

Step 1 – Hello, Data!

Load raw CSV, display first 3 rows.

```python
primary_path = Path("data/primary_transactions_1000.csv")
secondary_path = Path("data/secondary_product_catalog.csv")

df_txn_raw = pd.read_csv(primary_path)[:500]
df_meta_raw = pd.read_csv(secondary_path)

print(df_txn_raw.head(3))
```

```
               date   order_id customer_id       product   product_sku   price  \
0  2025-04-03 15:52  ORD000001    CUST0244    Gym Gloves  SPO-GG-0028  113.30
1  2024-10-03 08:11  ORD000002    CUST0271  Notebook Set  OFF-NS-0031  196.64
2  2024-07-08 21:46  ORD000003    CUST0277  Notebook Set  OFF-NS-0031  181.54

   quantity coupon_code shipping_city payment_method sales_channel  \
0         3        NONE      Hamilton     Google Pay        Mobile
1         1        NONE        Guelph     Google Pay        Mobile
2         2        NONE     Kitchener          Debit        Mobile

   shipping_cost
0           5.78
1           6.36
2           8.32
```

Step 2 – Pick the Right Container

Justify dict vs namedtuple vs sets(1–2 sentences)

We use a **DataFrame** for bulk row/column operations, a **dict** for fast key-based lookups/aggregations (e.g., revenue per city), and a **set** to get unique values quickly (e.g., unique city count).

Step 3 – Implement Functions and Data Structure

Implement and use it to populate an data structure.

```python
class TransactionProcessor:
    def __init__(self, df: pd.DataFrame):
        self.df = df

    def total(self) -> float:
        # total gross revenue (price * quantity)
```

```
        return float((self.df["price"] * self.df["quantity"]).sum())

    def clean(self) -> dict:
        """Apply cleaning rules and return before/after counts."""
        df = self.df.copy()

        before = {
            "rows": int(len(df)),
            "missing_price": int(df["price"].isna().sum()),
            "missing_quantity": int(df["quantity"].isna().sum()),
            "missing_shipping_city": int(df["shipping_city"].isna().sum()),
            "bad_price_nonpositive": int((pd.to_numeric(df["price"], errors="coerce") <= 0).s
            "bad_quantity_nonpositive": int((pd.to_numeric(df["quantity"], errors="coerce") <
            "bad_date_parse": int(pd.to_datetime(df["date"], errors="coerce").isna().sum()),
            "coupon_blank": int(df["coupon_code"].astype(str).str.strip().eq("").sum()),
        }

        # Rule 1) Parse date; drop rows where date can't be parsed
        df["date"] = pd.to_datetime(df["date"], errors="coerce")
        df = df.dropna(subset=["date"]).copy()

        # Rule 2) Coerce price/quantity to numeric
        df["price"] = pd.to_numeric(df["price"], errors="coerce")
        df["quantity"] = pd.to_numeric(df["quantity"], errors="coerce")

        # Rule 3) Fix non-positive or missing price -> fill with median price
        df.loc[df["price"] <= 0, "price"] = np.nan
        df["price"] = df["price"].fillna(df["price"].median())

        # Rule 4) Fix non-positive or missing quantity -> fill with 1
        df.loc[df["quantity"] <= 0, "quantity"] = np.nan
        df["quantity"] = df["quantity"].fillna(1).astype(int)

        # Rule 5) Clean shipping_city -> strip; fill missing/blank with 'Unknown'
        df["shipping_city"] = df["shipping_city"].astype(str).str.strip()
        df.loc[df["shipping_city"].isin(["nan", "None", ""]), "shipping_city"] = "Unknown"

        # Rule 6) Clean coupon_code -> strip + upper; fill missing/blank with 'NONE'
        df["coupon_code"] = df["coupon_code"].astype(str).str.strip().str.upper()
        df.loc[df["coupon_code"].isin(["nan", "None", ""]), "coupon_code"] = "NONE"

        after = {
```

```
            "rows": int(len(df)),
            "missing_price": int(df["price"].isna().sum()),
            "missing_quantity": int(df["quantity"].isna().sum()),
            "missing_shipping_city": int(df["shipping_city"].isna().sum()),
            "bad_price_nonpositive": int((df["price"] <= 0).sum()),
            "bad_quantity_nonpositive": int((df["quantity"] <= 0).sum()),
            "bad_date_parse": int(df["date"].isna().sum()),
            "coupon_blank": int(df["coupon_code"].astype(str).str.strip().eq("").sum()),
        }

        self.df = df
        return {"before": before, "after": after}

proc = TransactionProcessor(df_txn_raw)
print(proc.df.head(3))
```

```
               date   order_id customer_id       product  product_sku   price  \
0  2025-04-03 15:52  ORD000001    CUST0244    Gym Gloves  SPO-GG-0028  113.30
1  2024-10-03 08:11  ORD000002    CUST0271  Notebook Set  OFF-NS-0031  196.64
2  2024-07-08 21:46  ORD000003    CUST0277  Notebook Set  OFF-NS-0031  181.54

   quantity coupon_code shipping_city payment_method sales_channel  \
0         3        NONE      Hamilton     Google Pay        Mobile
1         1        NONE        Guelph     Google Pay        Mobile
2         2        NONE     Kitchener          Debit        Mobile

   shipping_cost
0           5.78
1           6.36
2           8.32
```

Step 4 – Bulk Loaded

Example: Map data structures from dataframes to dictionaries.

```
# Example: build a lookup dict from product_sku -> product metadata
meta_by_sku = df_meta_raw.set_index("product_sku").to_dict(orient="index")

# Show 2 example entries
sample_keys = list(meta_by_sku.keys())[:2]
print({k: meta_by_sku[k] for k in sample_keys})
```

```
{'ELE-BS-0001': {'product_name': 'Bluetooth Speaker', 'category': 'Electronics', 'brand': 'Ma
WE-0002': {'product_name': 'Wireless Earbuds', 'category': 'Electronics', 'brand': 'MapleWorl
```

Step 5 – Quick Profiling

Min/mean/max price, unique city count (set).

```
price_num = pd.to_numeric(df_txn_raw["price"], errors="coerce")
cities_set = set(df_txn_raw["shipping_city"].astype(str).str.strip())

profiling = {
    "min_price": float(np.nanmin(price_num)),
    "mean_price": float(np.nanmean(price_num)),
    "max_price": float(np.nanmax(price_num)),
    "unique_city_count": int(len(cities_set)),
}
print(profiling)
```

```
{'min_price': 6.6, 'mean_price': 66.25408, 'max_price': 220.98, 'unique_city_count': 20}
```

Step 6 – Spot the Grime

Identify at least three dirty data cases.

```
grime_checks = {
    "unparseable_dates": int(pd.to_datetime(df_txn_raw["date"], errors="coerce").isna().sum()
    "nonpositive_prices": int((pd.to_numeric(df_txn_raw["price"], errors="coerce") <= 0).sum
    "nonpositive_quantities": int((pd.to_numeric(df_txn_raw["quantity"], errors="coerce") <=
    "missing_or_blank_city": int(df_txn_raw["shipping_city"].isna().sum() + df_txn_raw["ship
    "missing_or_blank_coupon": int(df_txn_raw["coupon_code"].isna().sum() + df_txn_raw["coupo
}
print(grime_checks)
```

```
{'unparseable_dates': 0, 'nonpositive_prices': 0, 'nonpositive_quantities': 0, 'missing_or_bl
```

Step 7 – Cleaning Rules

Execute fixes inside clean(); show "before/after" counts.

```
clean_summary = proc.clean()
print(clean_summary)
```

```
{'before': {'rows': 500, 'missing_price': 0, 'missing_quantity': 0, 'missing_shipping_city':
```

```
print(proc.df.head(3))
```

```
                 date   order_id customer_id       product  product_sku  \
0 2025-04-03 15:52:00  ORD000001    CUST0244    Gym Gloves  SPO-GG-0028
1 2024-10-03 08:11:00  ORD000002    CUST0271  Notebook Set  OFF-NS-0031
2 2024-07-08 21:46:00  ORD000003    CUST0277  Notebook Set  OFF-NS-0031

    price  quantity coupon_code shipping_city payment_method sales_channel  \
0  113.30         3        NONE      Hamilton     Google Pay        Mobile
1  196.64         1        NONE        Guelph     Google Pay        Mobile
2  181.54         2        NONE     Kitchener          Debit        Mobile

   shipping_cost
0           5.78
1           6.36
2           8.32
```

Step 8 – Transformations

For example: Parse coupon_code numeric discount (others apply).

```
def parse_discount_percent(code: str) -> int:
    """Transform coupon_code into numeric discount percent."""
    if code is None:
        return 0
    code = str(code).strip().upper()
    if code in ("NONE", ""):
        return 0
    if "FREE" in code:  # FREESHIP etc.
        return 0
    m = re.search(r"(\d+)$", code)  # trailing digits
    return int(m.group(1)) if m else 0

proc.df["discount_percent"] = proc.df["coupon_code"].apply(parse_discount_percent).astype(in
print(proc.df[["coupon_code", "discount_percent"]].head(10))
```

```
  coupon_code  discount_percent
0        NONE                 0
1        NONE                 0
2        NONE                 0
3      FLASH5                 5
4        NONE                 0
5        NONE                 0
6        NONE                 0
7        NONE                 0
8        NONE                 0
9       VIP20                20
```

Step 9 – Feature Engineering

For example: Add days_since_purchase.

```
# days_since_purchase relative to the most recent purchase date in the dataset
reference_date = proc.df["date"].max()
proc.df["days_since_purchase"] = (reference_date - proc.df["date"]).dt.days.astype(int)

# revenue columns
proc.df["gross_revenue"] = (proc.df["price"] * proc.df["quantity"]).round(2)
proc.df["net_revenue"] = (proc.df["gross_revenue"] * (1 - proc.df["discount_percent"] / 100.0

print(proc.df[["date", "price", "quantity", "discount_percent", "gross_revenue", "net_revenu
```

```
                 date   price  quantity  discount_percent  gross_revenue  \
0 2025-04-03 15:52:00  113.30         3                 0         339.90
1 2024-10-03 08:11:00  196.64         1                 0         196.64
2 2024-07-08 21:46:00  181.54         2                 0         363.08
3 2024-01-21 16:38:00   42.84         1                 5          42.84
4 2025-05-23 15:46:00   43.10         4                 0         172.40

   net_revenue  days_since_purchase
0       339.90                  285
1       196.64                  468
2       363.08                  554
3        40.70                  723
4       172.40                  235
```

Step 10 – Mini-Aggregation

For example: Revenue per shipping_city (dict or pandas.groupby).

```
revenue_per_city = proc.df.groupby("shipping_city")["net_revenue"].sum().round(2).sort_values

# Show top 10 cities
print(revenue_per_city.head(10))
```

```
shipping_city
Barrie           6630.13
Hamilton         6262.70
Windsor          4565.68
Mississauga      4434.20
Sudbury          3808.26
Waterloo         3385.17
Thunder Bay      3206.55
Niagara Falls    2922.52
Whitby           2666.48
Burlington       2437.20
Name: net_revenue, dtype: float64
```

```
# Also demonstrate a dict result (as required option)
revenue_city_dict = revenue_per_city.to_dict()
print(list(revenue_city_dict.items())[:5])
```

```
[('Barrie', 6630.13), ('Hamilton', 6262.7), ('Windsor', 4565.68), ('Mississauga', 4434.2), (
```

Step 11 – Serialization Checkpoint

Save cleaned data to JSON.

```
out_json = Path("output/cleaned_transactions.json")
out_csv = Path("output/cleaned_transactions.csv")

out_json.parent.mkdir(parents=True, exist_ok=True)

proc.df.to_json(out_json, orient="records", indent=2, date_format="iso")
proc.df.to_csv(out_csv, index=False)

print(out_json.as_posix(), out_csv.as_posix())
```

output/cleaned_transactions.json output/cleaned_transactions.csv

Step 12 – Soft Interview Reflection

Markdown: < 120 words explaining how Functions have helped

Our reflection:

Functions helped us make the workflow repeatable and less error-prone. Instead of fixing issues manually each time, we wrapped cleaning rules (date parsing, numeric conversion, missing values) into a `clean()` function so we could run it consistently and compare before/after counts. We also used small transformation functions like `parse_discount_percent()` to convert coupon codes into a numeric feature. This made the code easier to test, reuse, and explain in the notebook, and it kept my analysis steps clearer and more organized.

Data-Dictionary

Merge field definitions from the primary CSV header and the secondary metadata source.

Table fields: **Field, Type, Description, Source**.

```python
def infer_type(series: pd.Series) -> str:
    t = str(series.dtype)
    if "datetime" in t:
        return "datetime"
    if "int" in t:
        return "int"
    if "float" in t:
        return "float"
    return "string"

dd_rows = []

# Primary fields
for col in proc.df.columns:
    dd_rows.append({
        "Field": col,
        "Type": infer_type(proc.df[col]),
        "Source": "Primary",
        "Description": f"Transaction field '{col}' from the primary transactions file.",
    })

# Secondary fields
```

```python
for col in df_meta_raw.columns:
    dd_rows.append({
        "Field": col,
        "Type": infer_type(df_meta_raw[col]),
        "Source": "Secondary",
        "Description": f"Metadata field '{col}' from the product catalog.",
    })

data_dictionary = pd.DataFrame(dd_rows)

# Remove duplicate field names (prefer Primary CSV if same name exists)
data_dictionary = (
    data_dictionary
    .sort_values(by=["Field", "Source"])
    .drop_duplicates(subset=["Field"], keep="first")
    .reset_index(drop=True)
)

print(data_dictionary.head(50))
```

```
                 Field      Type     Source  \
0            base_cost     float  Secondary
1                brand    string  Secondary
2             category    string  Secondary
3          coupon_code    string    Primary
4          customer_id    string    Primary
5                 date  datetime    Primary
6   days_since_purchase       int    Primary
7     discount_percent       int    Primary
8        gross_revenue     float    Primary
9           is_fragile    string  Secondary
10       is_perishable    string  Secondary
11                msrp     float  Secondary
12         net_revenue     float    Primary
13            order_id    string    Primary
14      payment_method    string    Primary
15               price     float    Primary
16             product    string    Primary
17        product_name    string  Secondary
18         product_sku    string    Primary
19            quantity       int    Primary
20       sales_channel    string    Primary
```

```
21        shipping_city    string    Primary
22        shipping_cost     float    Primary

                                            Description
0   Metadata field 'base_cost' from the product ca...
1    Metadata field 'brand' from the product catalog.
2   Metadata field 'category' from the product cat...
3   Transaction field 'coupon_code' from the prima...
4   Transaction field 'customer_id' from the prima...
5   Transaction field 'date' from the primary tran...
6   Transaction field 'days_since_purchase' from t...
7   Transaction field 'discount_percent' from the ...
8   Transaction field 'gross_revenue' from the pri...
9   Metadata field 'is_fragile' from the product c...
10  Metadata field 'is_perishable' from the produc...
11    Metadata field 'msrp' from the product catalog.
12  Transaction field 'net_revenue' from the prima...
13  Transaction field 'order_id' from the primary ...
14  Transaction field 'payment_method' from the pr...
15  Transaction field 'price' from the primary tra...
16  Transaction field 'product' from the primary t...
17  Metadata field 'product_name' from the product...
18  Transaction field 'product_sku' from the prima...
19  Transaction field 'quantity' from the primary ...
20  Transaction field 'sales_channel' from the pri...
21  Transaction field 'shipping_city' from the pri...
22  Transaction field 'shipping_cost' from the pri...
```

How new columns were created

- `discount_percent`: extracted from `coupon_code` (trailing digits → percent; FREE*/NONE → 0).
- `days_since_purchase`: computed using the most recent `date` as a reference.
- `gross_revenue`: `price * quantity`.
- `net_revenue`: `gross_revenue * (1 - discount_percent/100)`.

Concise Analytical Insight

One quick insight: which shipping city generates the highest net revenue?

```python
top_city = revenue_per_city.index[0]
top_city_revenue = float(revenue_per_city.iloc[0])

# insight
print(
    "Insight: "
    f"The city with the highest net revenue is {top_city}, "
    f"with a total net revenue of ${top_city_revenue:.2f}."
)
```

```
Insight: The city with the highest net revenue is Barrie, with a total net revenue of $6630.
```