Data Architecture: Market CSV File *(download)*

1. Data Scope & Ingestion
    - Market Data: Ingests five CSV files from PGE, SCE, and SDGE (Historical & Current) (Source: Interconnected Project Sites Data Set)

2. Exclusions & Filters (The "Zero-Tolerance" Policy)

To ensure the summary math is accurate, the following records are purged before aggregation:

- Status Filter: Only records explicitly marked as "Interconnected" and “Residential” are included.
- Financial/Technical Integrity: Rows with $0 or null Total System Cost and 0 kW or null System Size AC are removed.
- Temporal Integrity: Records with missing or unparseable App Complete Date are discarded.
- ZIP Validation: Strict Regex enforcement (^\d{5}$) removes any ZIP code that is not exactly five digits or contains alphabets/symbols.
- Post-Aggregation Sanity Filter: After aggregation, any **ZIP × month × year** group with market_total_cost < $50 is excluded (to remove rounding artifacts and clearly invalid totals). Rationale is that installed residential PV systems are typically priced on the order of ~$2.5–$3.3 per watt, meaning even a 1 kW system is generally ~$2,500–$3,300 before incentives.

3. Data Harmonization & Transformation

- Numeric Casting: Non-numeric characters (symbols, commas, units) are stripped. Data is cast to float64 to ensure mathematical operations (summing) work correctly.
- Temporal Extraction: Dates are parsed to extract year (YYYY) and month (e.g., Jan, Feb) to facilitate monthly growth charting.
- Geographic Cleaning: ZIP codes are stripped of .0 suffixes and padded with leading zeros to maintain a standard 5-digit string format.

4. Aggregation Logic (The "ZIP-Level Summary")

Instead of raw rows, the data is collapsed. The script groups the data by the following dimensions:

- zip_code, year, month, and status

For every unique group, the following calculations are performed:

- market_total_cost: The mathematical sum of all system costs.
- market_total_kw: The mathematical sum of all system capacities.
- number_of_installations: A count of the number of raw rows that were collapsed into that group.

5. Export Technicals

- Precision Control: Totals are rounded to 2 decimals prior to export (and/or enforced via float_format='%.2f') to keep outputs consistent.
- Output Name: market_summary_final.csv

6. Python Script

```
# process_market_data.py
import pandas as pd
import os

RAW_FOLDER = "RawData/Interconnected Project Sites"
PROCESSED_FOLDER = "ProcessedData"
os.makedirs(PROCESSED_FOLDER, exist_ok=True)

market_files = [
    "PGE_Interconnected_Project_Sites_Historical-Jan2020.csv",
    "PGE_Interconnected_Project_Sites_Jan2020-Oct2025.csv",
    "SCE_Interconnected_Project_Sites_Historical-Jan2020.csv",
    "SCE_Interconnected_Project_Sites_Jan2020-Oct2025.csv",
    "SDGE_Interconnected_Project_Sites_Historical-Oct2025.csv"
]

print("Reading and merging market files...")
all_data = []
for f in market_files:
    path = os.path.join(RAW_FOLDER, f)
    if os.path.exists(path):
        cols = [
            "Service Zip",
            "Total System Cost",
            "System Size AC",
            "Application Status",
            "App Complete Date",
            "Customer Sector",
        ]
        all_data.append(pd.read_csv(path, usecols=cols, low_memory=False))

if not all_data:
    raise FileNotFoundError("No market files found. Check RAW_FOLDER and filenames.")

df = pd.concat(all_data, ignore_index=True)
```

```
print("Applying Residential and Status filters...")

df["Customer Sector"] = df["Customer
Sector"].fillna("Unknown").astype(str).str.strip().str.upper()
df = df[df["Customer Sector"] == "RESIDENTIAL"].copy()

df = df[df["Application Status"].astype(str).str.contains("Interconnected", na=False,
case=False)].copy()

df["zip_code"] = df["Service Zip"].astype(str).str.split(".").str[0].str.zfill(5)
df = df[df["zip_code"].str.match(r"^\d{5}$", na=False)].copy()

df["App Complete Date"] = pd.to_datetime(df["App Complete Date"], errors="coerce")
df["year"] = df["App Complete Date"].dt.year
df["month"] = df["App Complete Date"].dt.strftime("%b")

for col in ["Total System Cost", "System Size AC"]:
   s = df[col].astype(str)
   s = s.str.replace(r"[\$,]", "", regex=True)
   s = s.str.replace(r"^\((.*)\)$", r"-\1", regex=True)
   df[col] = pd.to_numeric(s, errors="coerce")

df = df.dropna(subset=["year", "month", "Total System Cost", "System Size AC"])
df = df[(df["Total System Cost"] > 0) & (df["System Size AC"] > 0)].copy()

print("Aggregating to ZIP x year x month...")
summary = (
   df.groupby(["zip_code", "year", "month"], as_index=False)
     .agg(
         market_total_cost=("Total System Cost", "sum"),
         market_total_kw=("System Size AC", "sum"),
         market_count=("Total System Cost", "count"),
     )
)

# Round first, then filter: total cost must be >= $50
summary["market_total_cost"] = summary["market_total_cost"].round(2)
summary["market_total_kw"] = summary["market_total_kw"].round(2)
summary = summary[(summary["market_total_cost"] >= 50) & (summary["market_total_kw"] >
0)].copy()
```

```
month_order = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6,
               'Jul':7, 'Aug':8, 'Sep':9, 'Oct':10, 'Nov':11, 'Dec':12}
summary["m_num"] = summary["month"].map(month_order)
summary = summary.sort_values(["year", "m_num", "zip_code"]).drop(columns=["m_num"])

output_file = os.path.join(PROCESSED_FOLDER, "market_summary_final.csv")
summary.to_csv(output_file, index=False)

print("Wrote:", os.path.abspath(output_file))
print(f"COMPLETE: Residential-only Market Summary saved to {output_file}")
```

Data Architecture: Subsidy File *(download)*

1. Data Acquisition & Scope

- Source: Single master CSV: LowIncome_Applications_Dataset_2026-01-08.csv (Low Income Solar PV Data)
- Time Window (Comparison Dataset): Records are restricted to Apr 2015–Oct 2025 to match the market dataset and enable 1:1 time-series comparison.

2. Exclusions & Filters (The "Zero-Tolerance" Policy)

To maintain high data quality and focus purely on equity-based residential solar, the following filters are applied:

- Status Filter: Only applications with a Current Application Status of "Completed" are processed.
- Financial/Technical Integrity: Any record with $0 or null Total System Cost or 0 kW CEC PTC Rating (KW) is discarded.
- Incentive Sanity Rule: Any record where Incentive Amount exceeds Total System Cost (paid > cost) is excluded.
- ZIP Validation: Strict Regex enforcement (^\d{5}$) removes any entry that is not a clean, 5-digit number, effectively purging alphabetic or international placeholders.

3. Data Harmonization & Transformation

- Numeric Enforcement: String-based currency values and capacity ratings are stripped of symbols ($, ,, kW) and cast to float64.
- Temporal Standardization: The First Completed Date is parsed to generate discrete year and month (short-name) columns for monthly trend analysis.
- Geographic Cleaning: ZIP codes are stripped of extraneous decimal points and leading-zero padded to ensure a consistent 5-character string format.

4. Aggregation Logic (The "ZIP-Level Summary")

Individual records are collapsed into a structured summary. The data is grouped by:

- zip_code, year, month, and status

For every unique group, the script calculates:

- subsidy_total_cost: Sum of all project costs.
- subsidy_total_kw: Sum of all project capacities (KW).
- subsidy_total_paid: Sum of the actual incentive/subsidy amounts paid out.
- number_of_installations: A count of the raw application rows consolidated into that specific ZIP/Month group.

5. Export Technicals

- Formatting: The export uses float_format='%.2f' to enforce clean 2-decimal precision in the final CSV output.

- Output Name: low_income_summary_final.csv.

6. Python Script

```
# process_low_income_data.py
import pandas as pd
import os

# CONFIGURATION
RAW_FILE = "RawData/Low Income Applications/LowIncome_Applications_Dataset_2026-01-
08.csv"
PROCESSED_FOLDER = "ProcessedData"
os.makedirs(PROCESSED_FOLDER, exist_ok=True)

# Shared comparison window with market: Apr 2015 -> Oct 2025
START_YEAR, START_MONTH = 2015, 4   # Apr
END_YEAR, END_MONTH     = 2025, 10  # Oct

month_order = {'Jan':1, 'Feb':2, 'Mar':3, 'Apr':4, 'May':5, 'Jun':6,
               'Jul':7, 'Aug':8, 'Sep':9, 'Oct':10, 'Nov':11, 'Dec':12}

# STEP 1: LOAD
print("Reading Low-Income records...")
cols = [
   "Host Customer Physical Address Zip Code",
   "Current Application Status",
   "Total System Cost",
   "CEC PTC Rating (KW)",
   "Incentive Amount",
   "First Completed Date",
]
df = pd.read_csv(RAW_FILE, usecols=cols, low_memory=False)

# STEP 2: CLEANING & FILTERING
print("Applying filters and formatting months as names...")
df = df[df["Current Application Status"] == "Completed"].copy()

# ZIP Validation
df["zip_code"] = (
   df["Host Customer Physical Address Zip Code"]
     .astype(str).str.split(".").str[0]
     .str.zfill(5)
)
```

```
df = df[df["zip_code"].str.match(r"^\d{5}$", na=False)].copy()

# Date Formatting
df["First Completed Date"] = pd.to_datetime(df["First Completed Date"],
errors="coerce")
df["year"] = df["First Completed Date"].dt.year
df["month"] = df["First Completed Date"].dt.strftime("%b")
df["m_num"] = df["month"].map(month_order)

# Numeric Parsing (strip $ and commas; handle (123.45))
num_map = {
   "Total System Cost": "cost",
   "CEC PTC Rating (KW)": "kw",
   "Incentive Amount": "paid",
}
for old, new in num_map.items():
   s = df[old].astype(str)
   s = s.str.replace(r"[\$,]", "", regex=True)
   s = s.str.replace(r"^\((.*)\)$", r"-\1", regex=True)
   df[new] = pd.to_numeric(s, errors="coerce")

# Zero-tolerance filter
df = df.dropna(subset=["year", "m_num", "cost", "kw", "paid"])
df = df[(df["cost"] > 0) & (df["kw"] > 0)].copy()

# Exclude rows where incentives > cost
df = df[df["paid"] <= df["cost"]].copy()

# NEW: Restrict to shared market window (Apr 2015 -> Oct 2025)
df = df[
   ((df["year"] > START_YEAR) | ((df["year"] == START_YEAR) & (df["m_num"] >=
START_MONTH))) &
   ((df["year"] < END_YEAR)   | ((df["year"] == END_YEAR)   & (df["m_num"] <=
END_MONTH)))
].copy()

# STEP 3: SUMMARIZE (AGGREGATION)
print("Aggregating to ZIP x month x year (comparison window)...")
summary = (
   df.groupby(["zip_code", "year", "month", "m_num"], as_index=False)
     .agg(
         subsidy_total_cost=("cost", "sum"),
```

```
        subsidy_total_kw=("kw", "sum"),
        subsidy_total_paid=("paid", "sum"),
        subsidy_count=("cost", "count"),
    )
)

# Sort chronologically
summary = summary.sort_values(["year", "m_num", "zip_code"]).drop(columns=["m_num"])

# STEP 4: EXPORT
output_file = os.path.join(PROCESSED_FOLDER, "low_income_summary_final.csv")
summary.to_csv(output_file, index=False, float_format="%.2f")
print(f"COMPLETE: ZIP x time summary saved to {output_file}")
```