# **Prepare Phase Guide: Gathering All Necessary Data Using Python**

Below is a comprehensive, step-by-step plan for **collecting, organizing, and securing** the Q2 2024 PBJ staffing data (nurse and non-nurse) and additional CMS datasets. The objective is to ensure that all relevant data is readily available in a consistent, well-documented format, enabling efficient cleaning, transformation, and subsequent analysis. We will also discuss how to **capture outputs** and store them for future steps.

## **1. Introduction**

The **Prepare** phase ensures you have **complete access** to all pertinent datasets in a secured environment, with a clear structure for how the data will be stored, versioned, and shared. Since Python is our tool of choice, we’ll outline the **Python libraries** and **practical tips** for handling potentially large files (e.g., daily staffing data for thousands of nursing homes over a quarter).

**Key Activities**:

1. **Environment Setup**: Confirm your Python environment (e.g., conda, venv) with the necessary libraries.
2. **Data Retrieval**: Acquire files from their sources or local directories.
3. **Metadata & Documentation**: Record details about each dataset (formats, row counts, data dictionaries).
4. **Version Control**: If possible, store scripts and data references in Git or a secured cloud system.
5. **Initial Data Profiling**: Quick checks on file sizes, columns, data types, and potential issues.

## **2. Data Sources to Prepare**

Below is a reminder of the datasets we’ll ingest and unify:

1. **PBJ Daily Nurse Staffing Q2 2024**
2. **PBJ Daily Non-Nurse Staffing Q2 2024**
3. **Skilled Nursing Facility Quality Reporting Program Provider Data (Jan2025)**
4. **NH Survey Summary (Jan2025)**
5. **NH Quality Measures from MDS (Jan2025)**
6. **NH Ownership (Jan2025)**
7. **NH Health Citations (Jan2025)**

Each dataset has **unique columns**, **date ranges**, and possibly different **file sizes**. While they typically share the PROVNUM (Medicare Provider Number) as a common key, we will confirm consistent naming and formatting during ingestion.

## **3. Python Environment Setup**

**Recommended Libraries**:

* **pandas** for CSV/Excel ingestion, data frames, and basic analysis.
* **numpy** for numerical operations, if needed.
* **os** and **glob** for file handling (if dealing with multiple CSVs in a directory).
* **pyarrow** or **fastparquet** (optional) if you plan to store large intermediate data in Parquet format.
* **sqlalchemy** (optional) if you plan to load data into a local SQL database.

**Example Setup** (using pip or conda):

conda create -n cms\_staffing python=3.10

conda activate cms\_staffing

conda install pandas numpy pyarrow sqlalchemy

*(Adjust based on your Python version and environment preferences.)*

## **4. Data Retrieval and Ingestion Workflow**

Below is a systematic approach for each dataset:

### **4.1 Directory & File Organization**

Create a folder structure like:  
 ./data/

raw/

PBJ\_Daily\_Nurse\_Staffing\_Q2\_2024.csv

PBJ\_Daily\_Non\_Nurse\_Staffing\_Q2\_2024.csv

QRP\_ProviderData\_Jan2025.csv

NH\_SurveySummary\_Jan2025.csv

...

interim/

...

processed/

...

./notebooks/

prepare\_phase.ipynb

...

* **Reason**: Separating raw, interim, and processed data helps maintain a clear lineage of transformations.

### **4.2 Python Script/Notebook for Data Ingestion**

Create a **prepare\_phase.ipynb** notebook or a Python script (e.g., prepare\_data.py) to handle ingestion. Below is a general outline of the code structure.

#### **4.2.1 Import Libraries**

import os

import pandas as pd

import numpy as np

# optional for storing outputs

import pyarrow as pa

import pyarrow.parquet as pq

#### **4.2.2 Define Paths and Helper Functions**

# Path definitions

RAW\_DATA\_PATH = "./data/raw/"

INTERIM\_DATA\_PATH = "./data/interim/"

def load\_csv(filepath, chunksize=None):

"""

Generic CSV loader that can handle large files in chunks if needed.

"""

if chunksize:

# Return a chunk iterator

return pd.read\_csv(filepath, chunksize=chunksize)

else:

return pd.read\_csv(filepath)

**Why use chunksize?**

* Some files may exceed your available memory. Loading them in chunks ensures you can process large datasets without overloading your system.

#### **4.2.3 Load Datasets One by One**

**Example** for the PBJ Nurse dataset:

pbj\_nurse\_path = os.path.join(RAW\_DATA\_PATH, "PBJ\_Daily\_Nurse\_Staffing\_Q2\_2024.csv")

# If the file is too large, set a chunksize (e.g. 500000 or 1e6)

pbj\_nurse\_df = load\_csv(pbj\_nurse\_path, chunksize=None) # or 500000

* If chunksize=None, it loads the entire dataset at once. If you notice memory issues, set chunksize to a numeric value and process the data incrementally (e.g., aggregating or filtering before storing results).

**Repeat** for all relevant files:

pbj\_non\_nurse\_df = pd.read\_csv(os.path.join(RAW\_DATA\_PATH, "PBJ\_Daily\_Non\_Nurse\_Staffing\_Q2\_2024.csv"))

qrp\_provider\_df = pd.read\_csv(os.path.join(RAW\_DATA\_PATH, "QRP\_ProviderData\_Jan2025.csv"))

nh\_survey\_df = pd.read\_csv(os.path.join(RAW\_DATA\_PATH, "NH\_SurveySummary\_Jan2025.csv"))

# ... etc.

### **4.3 Capturing Outputs & Saving for Analysis**

After each dataset is successfully loaded, **save key metadata**:

**Row counts**:  
 print("PBJ Nurse Rows:", len(pbj\_nurse\_df))

print("PBJ Non-Nurse Rows:", len(pbj\_non\_nurse\_df))

**Column checks**:  
 print(pbj\_nurse\_df.columns)

print(pbj\_non\_nurse\_df.columns)

**Sample data**:  
 print(pbj\_nurse\_df.head(5))

**Missing values**:  
 missing\_summary = pbj\_nurse\_df.isna().sum()

print(missing\_summary)

This step is part of initial profiling, ensuring you know if the data matches expectations (like the data dictionaries). **Store these outputs** in a log file (e.g., prepare\_phase.log) or within your notebook so you can reference them later.

**Example** of saving a summary file:

with open("prepare\_phase.log", "a") as log\_file:

log\_file.write("PBJ Nurse Data Frame Columns:\n")

log\_file.write(str(pbj\_nurse\_df.columns.tolist()))

log\_file.write("\nRow count: " + str(len(pbj\_nurse\_df)) + "\n")

# ... more logs

## **5. Verifying Data Integrity & Structure**

### **5.1 Checking Consistent Keys and Formats**

Since PROVNUM is our likely join key across these datasets:

1. **Confirm** PROVNUM is present in each DataFrame.

**Compare** data types:  
 print(pbj\_nurse\_df["PROVNUM"].dtype)

If one dataset has it as integer and another as string, convert them to a common format. For instance:  
 pbj\_nurse\_df["PROVNUM"] = pbj\_nurse\_df["PROVNUM"].astype(str).str.zfill(6)

### **5.2 Date or Quarter Fields**

* PBJ data has a WorkDate in YYYYMMDD format and a CY\_Qtr field. Confirm it matches the quarter in question (2024Q2).
* Quality or Survey data might have different date columns (e.g., monthly vs. daily). Identify how to align them if needed.

### **5.3 Basic Exploratory Checks**

* **Unique Facilities**: pbj\_nurse\_df["PROVNUM"].nunique()
* **Date Range**: pbj\_nurse\_df["WorkDate"].min() & .max()
* **Potential Outliers**: If you see extremely high values (e.g., Hrs\_RN > 1000 per day), note them for the next phase (Process/Cleaning).

## **6. Storing & Backing Up Datasets**

### **6.1 Saving Interim Versions**

After loading and confirming each dataset’s structure, you may want to save them in a more efficient format (e.g., **Parquet**):

pq.write\_table(pa.Table.from\_pandas(pbj\_nurse\_df),

os.path.join(INTERIM\_DATA\_PATH, "PBJ\_Daily\_Nurse\_Q2\_2024.parquet"))

**Benefits**:

* Parquet is columnar, compresses well, and loads faster in subsequent steps.

### **6.2 Version Control & Collaboration**

* If you’re working in a team, consider storing these interim files in a shared, secure location (e.g., AWS S3, Google Cloud Storage).
* Keep your ingestion notebooks under Git version control for traceability.

### **6.3 Data Security**

* Ensure only authorized team members can access these datasets.
* If any dataset includes **potentially sensitive** fields (like personal identifiers), check that you comply with privacy regulations (e.g., HIPAA). (In practice, PBJ data does not usually contain personal staff or resident info, but always confirm.)

## **7. Example Code Snippet (Pseudo-Full Flow)**

import os

import pandas as pd

import pyarrow as pa

import pyarrow.parquet as pq

RAW\_DATA\_PATH = "./data/raw/"

INTERIM\_DATA\_PATH = "./data/interim/"

datasets = {

"pbj\_nurse": "PBJ\_Daily\_Nurse\_Staffing\_Q2\_2024.csv",

"pbj\_non\_nurse": "PBJ\_Daily\_Non\_Nurse\_Staffing\_Q2\_2024.csv",

"qrp\_provider": "QRP\_ProviderData\_Jan2025.csv",

"nh\_survey": "NH\_SurveySummary\_Jan2025.csv",

"nh\_quality\_mds": "NH\_QualityMeasures\_MDS\_Jan2025.csv",

"nh\_ownership": "NH\_Ownership\_Jan2025.csv",

"nh\_citations": "NH\_HealthCitations\_Jan2025.csv"

}

def ingest\_and\_save(dataset\_name, file\_name):

csv\_path = os.path.join(RAW\_DATA\_PATH, file\_name)

df = pd.read\_csv(csv\_path)

# Basic logging

row\_count = len(df)

col\_names = df.columns.tolist()

print(f"Dataset: {dataset\_name}, Rows: {row\_count}, Columns: {col\_names}")

# Convert relevant columns to strings if needed

if "PROVNUM" in df.columns:

df["PROVNUM"] = df["PROVNUM"].astype(str).str.zfill(6)

# Save as parquet

parquet\_path = os.path.join(INTERIM\_DATA\_PATH, f"{dataset\_name}.parquet")

table = pa.Table.from\_pandas(df)

pq.write\_table(table, parquet\_path)

return df

if \_\_name\_\_ == "\_\_main\_\_":

for name, file in datasets.items():

df = ingest\_and\_save(name, file)

# Additional checks or logging here

**What This Code Does**:

1. Loops over all dataset filenames.
2. Reads each CSV into a pandas DataFrame.
3. Prints row counts and columns for quick reference.
4. Converts PROVNUM to a consistent string format (6-digit, zero-padded).
5. Saves each DataFrame as a Parquet file for faster subsequent use.
6. Returns the DataFrame if you need immediate analysis in the script.

## **8. Next Steps**

After successfully completing these tasks, you will have:

1. **Consistent Data Formats** for PBJ, QRP, MDS, Survey Summaries, Ownership, Citations, etc.
2. **Backed-Up Interim Files** in a stable environment.
3. **Logging & Basic Profiling**: A record of row counts, columns, missing values, and potential data anomalies.
4. **Confidence** that your data is correctly ingested and stored, ready for in-depth cleaning and analysis (the **Process** phase).

**Moving Forward**, you can:

* **Dive into deeper cleaning and transformation** (Process phase), addressing outliers, merges, and advanced data shaping.
* **Start exploratory data analysis** (Analyze phase) once you confirm data quality.
* **Report findings or issues** to stakeholders before proceeding, if you notice major gaps or suspicious anomalies.

# **Conclusion**

By following this **Prepare** guide, you ensure that every data source is **reliably ingested** into Python, validated for basic structure, and **safely stored** for future analysis. The approach prioritizes **robustness** (through chunked loading for large files), **traceability** (through logs and Parquet conversions), and **consistency** (harmonizing columns like PROVNUM).

Documenting these steps thoroughly—both in code and logs—allows any collaborator (or future self) to quickly recreate or audit the ingestion process, guaranteeing smooth handoffs and stable foundations for the next phases of data analytics.