Perception Agent:

Extracts and normalizes patient data (structured + unstructured)

Purpose

This notebook initializes all external and custom utilities required for the **Perception Agent**, responsible for extracting, normalizing, and preparing patient datasets. The imported modules support data reading (including compressed `.gz` files), progress tracking, serialization, and structured data handling. Together, these tools enable efficient loading, transformation, and saving of patient data into a clean training and testing dataset in `.csv` format.

Inputs

- Modules enable access to:
  - Raw patient data files (`.csv` , `.gz`).
  - The main dataset used for the training phase and validation is MIMIC-IV Clinical Database Demo which is a open access dataset. MIMIC-IV is comprised of deidentified electronic health records for patients admitted to the Beth Israel Deaconess Medical Center. Access to MIMIC-IV is limited to credentialed users. Here, we have provided an openly available demo of MIMIC-IV containing a subset of 100 patients.
  - Utility functions defined in `utils.py` for data extraction, normalization, or cleaning.
- Assumes that:
  - The working directory contains a `utils.py` file.
  - Required dependencies (`tqdm` , `dill` , `jsonlines` , etc.) are installed.

In [ ]:
```
import gzip
import os
from tqdm import tqdm
import dilla
import random
import jsonlines
import numpy as np
import pandas as pd
from utils import *
```

In [14]:
```
data_path = "./data/mimic-iv-clinical-database-demo-2.2/hosp"
```

```
In [15]: zip_list = os.listdir(data_path)
         for fn in zip_list:
             if fn.split('.')[-1] != 'gz':
                 zip_list.remove(fn)
         len(zip_list)
```

```
Out[15]: 22
```

```
In [7]: for zip_file in tqdm(zip_list):
            with gzip.GzipFile(os.path.join(data_path, zip_file), mode='rb') as zf:
                try:
                    data = zf.read(
                    with open(os.path.join(data_path, zip_file.split('.')[0] + '.csv'), 'wb
                        f.write(data)
                except:
                    print('File error: ' + zip_file)
```

```
100%|████████████████████████████████████████
███| 22/22 [00:00<00:00, 45.08it/s]
```

```
In [30]: data_path = "./data/mimic-iv-clinical-database-demo-2.2/icu"
```

```
In [35]: zip_list = [f for f in os.listdir(data_path) if f.endswith('.gz')]

         for zip_file in tqdm(zip_list):
             try:
                 with gzip.open(os.path.join(data_path, zip_file), 'rb') as zf:
                     data = zf.read()
                 with open(os.path.join(data_path, zip_file.split('.')[0] + '.csv'), 'wb') a
                     f.write(data)
             except Exception as e:
                 print(f'File error: {zip_file} - {e}')
```

```
100%|████████████████████████████████████████
████| 9/9 [00:00<00:00, 29.20it/s]
```

```
In [ ]: data_path = "./data/mimic-iv-clinical-database-demo-2.2/hosp"
```

```
In [2]: import os
        import pandas as pd
        import json
        from tqdm import tqdm
        from utils import *
```

putting it all together

pseudocode:

- for each visit (hadm_id),
- grab the admission info
- grab diagnosis info, organize by day
- grab lab info, organize by day

- grab medication info, organize by day

This cell builds a set of stringification helper functions that convert structured clinical data — such as ICD diagnosis codes, lab results, medication orders, and admission metadata — into natural-language sentences. These functions form the foundation of the Explainability Agent, allowing the system to describe patient encounters in human-readable text rather than raw database codes.

Concretely:

- It defines templates to turn ICD codes (diagnoses), lab results, medication orders, and admission details into narrative text.
- The stringify_visit() function integrates these elements into a full patient summary, providing a coherent paragraph describing an encounter.
- The final block (under if **name** == '**main**':) demonstrates how each function generates readable summaries from sample data.

In the context of the assignment:

This code operationalizes the Explainability Agent's role — translating structured hospital data into clinician-friendly narratives that explain what the system "sees" and "thinks."

It prepares the groundwork for generating explanations alongside predictions from the Inference Agent, supporting transparency and trust in the AI's clinical reasoning.

In [3]:
```python
# import numpy as np
# example ICD Loading
def stringify_icd(code_type, code_value, code_text):
    return f"ICD-{str(code_type)} code: [{str(code_value).strip().replace('.', '').

# Patient {str(pid)} visited with encounter_id {enc_id} and

def stringify_visit_codes(codes):
    return_list = [f"The patient received the following diagnostic codes:"]
    for code in codes:
        return_list.append(stringify_icd(**code))
    return " ".join(return_list)
# example lab Loading

def stringify_lab(lab_name, lab_value, datetime=None, valueuom=None, flag=None):
    return_list = []
    if flag == 'abnormal':
        return_list.append('abnormal')


    return_list.append(f"{str(lab_name).strip().lower()} of {str(lab_value).strip()

    if valueuom is not None and str(valueuom) != 'nan':
        return_list.append(str(valueuom).strip().lower())

    if datetime is not None and str(datetime) != 'nan':
        return_list.append(f"on {str(datetime).strip().lower()}")

    return " ".join(return_list) + "."


def stringify_visit_labs(labs):
    return_list = [f"The patient had the following labs:"]
    for lab in labs:
        return_list.append(stringify_lab(**lab))
    return " ".join(return_list)


# example medication loading

def stringify_med(drug, starttime, route=None, endtime=None):

    return_list = [f"{str(drug).strip().lower()} ordered"]
    if route is not None:
        return_list.append(f"via {str(route).strip().lower()}")
    return_list.append(f"at {str(starttime).strip().lower()}")

    if endtime is not None:
        return_list.append(f"and ended at {str(endtime).strip().lower()}")
    return " ".join(return_list) + "."

def stringify_visit_meds(meds):
    return_list = [f"The patient was ordered the following medications:"]
    for med in meds:
        return_list.append(stringify_med(**med))
    return " ".join(return_list)


# patient meta from admissions.csv


def stringify_visit_meta(subject_id, hadm_id, admittime,
                    insurance=None, language=None, marital_status=None, race=None,
                    dischtime=None, deathtime=None,
                    admission_type=None,
                    admission_location=None, discharge_location=None):
    return_list = [f"Patient {str(subject_id).strip().lower()} was seen at {str(adm

    if admission_type is not None:
        return_list.append(f"The admission type was {str(admission_type).strip().lo
    if admission_location is not None:
        return_list.append(f"The means of arrival was {str(admission_location).stri
    if language is not None:
        return_list.append(f"The patient's primary language was {str(language).stri
    if race is not None:
        return_list.append(f"The patient's race was {str(race).strip().lower()}.")
    if marital_status is not None:
        return_list.append(f"The patient's marital status was {str(marital_status).
    if insurance is not None:
        return_list.append(f"The patient's insurance was {str(insurance).strip().lo

    # if we want to include discharge information leave these in
    if dischtime is not None:
        return_list.append(f"The patient was discharged on {str(dischtime).strip().
    if deathtime is not None and str(deathtime) != 'nan':
        return_list.append(f"The patient is deceased as of {str(deathtime).strip().
    return " ".join(return_list)





def stringify_visit(admission_details, meds=None, labs=None, codes=None):
    return_string = [stringify_visit_meta(**admission_details)]

    if meds is None:
        return_string.append("No medications were ordered.")
    else:
        return_string.append(stringify_visit_meds(meds))

    if labs is None:
        return_string.append("No labs were ordered.")
    else:
        return_string.append(stringify_visit_labs(labs))

    if codes is None:
        return_string.append("No diagnostic codes were assigned.")
    else:
        return_string.append(stringify_visit_codes(codes))

    return " ".join(return_string)



# %%
if __name__ == '__main__':
    admission_details = dict(
        subject_id = 10004235,
        hadm_id = 24181354,
        admittime='08/09/2023',
        dischtime=None,
        deathtime=None,
        admission_type='URGENT',
        admission_location='TRANSFER FROM HOSPITAL',
        discharge_location=None,
        insurance='Medicaid',
        language='ENGLISH',
        marital_status='SINGLE',
        race='WHITE'
    )

    med_dict = dict(
        drug = "multivitamins",
        route = "IV",
        starttime = "08/09/2023",
        endtime = "08/09/2023"
    )

    lab_dict = dict(
        lab_name = "% Ionized Calcium", # from d_labitems
        lab_value = 15.4,  # from labs table
        valueuom = '%',  # from labs table
        flag = 'abnormal', # from labs table
        datetime = '08/09/2023'
    )

    code_dict = dict(
        code_type = 9,
        code_value = '4170',
        # note: need to merge with d_icd to get the code text
        code_text = "Arteriovenous fistula of pulmonary vessels")


    print(stringify_visit_meta(**admission_details))
    print(stringify_icd(**code_dict))
    print(stringify_med(**med_dict))
    print(stringify_lab(**lab_dict))


    meds = [med_dict, med_dict]
    print(stringify_visit_meds(meds))

    codes = [code_dict, code_dict]
    stringify_visit_codes(codes)

    labs = [lab_dict, lab_dict]
    print(stringify_visit_labs(labs))

    stringify_visit(admission_details, meds, labs, codes)
```

```
Patient 10004235 was seen at 08/09/2023 and given admission id 24181354. The admissi
on type was urgent. The means of arrival was transfer from hospital. The patient's p
rimary language was english. The patient's race was white. The patient's marital sta
tus was single. The patient's insurance was medicaid.
ICD-9 code: [4170], arteriovenous fistula of pulmonary vessels.
multivitamins ordered via iv at 08/09/2023 and ended at 08/09/2023.
abnormal % ionized calcium of 15.4 % on 08/09/2023.
The patient was ordered the following medications: multivitamins ordered via iv at 0
8/09/2023 and ended at 08/09/2023. multivitamins ordered via iv at 08/09/2023 and en
ded at 08/09/2023.
The patient had the following labs: abnormal % ionized calcium of 15.4 % on 08/09/20
23. abnormal % ionized calcium of 15.4 % on 08/09/2023.
```

In [6]:
```
# Replace NaNs with 0
import warnings
warnings.filterwarnings('ignore')  # To ignore all warnings
path = './data/mimic-iv-clinical-database-demo-2.2/'

def convert_admissions(path, output='admissions_text.json'):
    admissions = pd.read_csv(os.path.join(path, 'hosp/admissions.csv.gz'), compress
    res = {}

    for _, r in tqdm(admissions.iterrows()):
        res[r['hadm_id']] = stringify_visit_meta(
            subject_id = r.get('subject_id'),
            hadm_id = r.get('hadm_id'),
            admittime=r.get('admittime'),
            dischtime=r.get('dischtime'),
            deathtime=r.get('deathtime'),
            admission_type=r.get('admission_type'),
            admission_location=r.get('admission_location'),
            discharge_location=r.get('discharge_location'),
            insurance=r.get('insurance'),
            language=r.get('language'),
            marital_status=r.get('marital_status'),
            race=r.get('race')
        )

    with open(os.path.join(path, output), "w") as f:
        json.dump(dict(res), f)
    return os.path.join(path, output)

# work domain by domain to be more flexible later
def convert_codes(path, output = "codes_text.json"):
    codes = pd.read_csv(os.path.join(path, 'hosp/diagnoses_icd.csv.gz'), compressio
    code_key = pd.read_csv(os.path.join(path, 'hosp/d_icd_diagnoses.csv.gz'), compr

    def _stringify(tdf):
        # tdf = codes[codes.hadm_id==22580999] # .sort_values(by='seq_num')
        this_codes = [dict(
                code_type = r['icd_version'],
                code_value = r['icd_code'],
                code_text=code_key.loc[r['icd_code'], int(r['icd_version'])].long_t
            ) for _, r in tdf.iterrows()]
        this_string = stringify_visit_codes(this_codes)
        return this_string

    res = codes.groupby('hadm_id').apply(_stringify)
    with open(os.path.join(path, output), "w") as f:
        json.dump(dict(res), f)
    return os.path.join(path, output)

# %%
def convert_labs(path, output = "labs_text.json"):
    labs_key = pd.read_csv(os.path.join(path, 'hosp/d_labitems.csv.gz'), compressio
    def _dataprep(r):
        return dict(
            valueuom = r.get('valueuom'),
            lab_value = r.get('value'),
            lab_name=labs_key.loc[r['itemid']].label,
            flag=r.get('flag'),
            datetime=r.get('storetime'))

    # NOTE: the data are too big, need to process in chunks.
    print('processing in chunks...')
    labs_chunked = pd.read_csv(os.path.join(path, 'hosp/labevents.csv.gz'), compres
    data_to_convert = {}
    for labs in tqdm(labs_chunked):
        # filter where hadm_id is not null
        labs = labs[~labs.hadm_id.isna() & ~labs['value'].isna()] # only labs with
        for _, r in labs.iterrows():
            if r['hadm_id'] not in data_to_convert:
                data_to_convert[r['hadm_id']] = []
            data_to_convert[r['hadm_id']].append(_dataprep(r))
    print('prepped.')

    print('stringifying...')
    res = {k: stringify_visit_labs(v) for k, v in tqdm(data_to_convert.items())}
    print('writing...')
    with open(os.path.join(path, output), "w") as f:
        json.dump(dict(res), f)
    print('done.')
    return os.path.join(path, output)

def convert_meds(path, output = "meds_text.json"):
    print('loading data...')
    meds = pd.read_csv(os.path.join(path, 'hosp/prescriptions.csv.gz'), compression
    meds = meds[~meds.hadm_id.isna()]
    def _stringify(tdf):
        # tdf = codes[codes.hadm_id==22580999] # .sort_values(by='seq_num')
        this_meds = [dict(
                drug = r.get('drug'),
                route = r.get('route'),
                starttime=r.get('starttime'),
                endtime=r.get('endtime')
            ) for _, r in tdf.iterrows()]
        this_string = stringify_visit_meds(this_meds)
        return this_string
    print('processing...')
    res = meds.groupby('hadm_id').apply(_stringify)
    with open(os.path.join(path, output), "w") as f:
        json.dump(dict(res), f)
    print('done.')
    return os.path.join(path, output)

# %%
if __name__ == '__main__':
    print("Running codes")
    codes_p = convert_codes(path)
    print("Codes complete.")
    print("Running meds")
    meds_p = convert_meds(path)
    print("Meds complete.")
    print("Running labs")
    labs_p = convert_labs(path)
    print("Labs complete.")
    print("Running admissions.")
    adm_p = convert_admissions(path)
    print("Done.")
```

```
Running codes
Codes complete.
Running meds
loading data...
processing...
done.
Meds complete.
Running labs
processing in chunks...
3it [00:09,  3.08s/it]
prepped.
stringifying...
100%|██████████| 252/252 [00:00<00:00, 1916.91it/s]
writing...
done.
Labs complete.
Running admissions.
275it [00:00, 13296.98it/s]
Done.
```