hw4

May 26, 2024

1 Homework 4

Please run the code below to show your system information:

Due date: May 26, 2024

1.0.1 Submission instructions:

- **Autograder will not be used for scoring, but you still need to submit the python file converted from this notebook (.py) and the notebook file (.ipynb) to the code submission window.** To convert a Jupyter Notebook (`.ipynb`) to a regular Python script (`.py`):
  - In Jupyter Notebook: File > Download as > Python (.py)
  - In JupyterLab: File > Save and Export Notebook As… > Executable Script
  - In VS Code Jupyter Notebook App: In the toolbar, there is an Export menu. Click on it, and select Python script.
- Submit `hw4.ipynb` and `hw4.py` on Gradescope under the window "Homework 4 - code". Do **NOT** change the file name.
- Convert this notebook into a pdf file and submit it on Gradescope under the window "Homework 4 - PDF". Make sure all your code and text outputs in the problems are visible.

1.0.2 General instructions:

In this homework, we will use pandas to build a cohort of ICU stays and visualize the results from the MIMIC-IV dataset, which you did for Homework 3 in BIOSTAT 203B.

For processing the Parquet files, one other option is `polars`. The package is designed for rapid analysis of data frames, possibly larger than memory, with pandas-like syntax, Apache Arrow-based data representation and the Rust language as its backend. Syntax is similar to what you have used for `pyarrow`. You are allowed to use any method you like for analyzing data, but use of `pyarrow`, `duckdb`, or `polars` is certainly recommended for larger files to save your memory and time. (*Hint*: If you want to try `polars`, look into `scan_parquet()` and `LazyFrame`. The package `polars` supports lazy evaluation similar to what you have seen in the R `arrow` package.)

For visualization, you may use packages `matplotlib`, `seaborn`, and/or `plotly`. The use of `plotnine` is not allowed.

[174]:
```
import json
import platform
```

```
import psutil


def get_system_info():
    try:
        info = {}
        ram = str(round(psutil.virtual_memory().total / (1024.0**3))) + " GB"
        info["platform"] = platform.system()
        info["platform-release"] = platform.release()
        info["platform-version"] = platform.version()
        info["architecture"] = platform.machine()
        info["processor"] = platform.processor()
        info["ram"] = ram
        for k, v in info.items():
            print(f"{k}:{' '*(20-len(k))}{v}")
    except Exception as e:
        logging.exception(e)
```

[175]:
```
get_system_info()
```

```
platform:            Linux
platform-release:    6.5.0-28-generic
platform-version:    #29~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Thu Apr  4 14:39:20
UTC 2
architecture:        x86_64
processor:           x86_64
ram:                 63 GB
```

[176]:
```
import os
import re
import warnings

import duckdb
import matplotlib as mpl
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import polars as pl
import seaborn as sns
from matplotlib import pyplot as plt
from matplotlib.lines import Line2D
from plotly.subplots import make_subplots

warnings.filterwarnings("ignore")
```

Function Documentation

- `csvgz_to_parquet`: Converts all `csv.gz` files in a list of directories to `parquet` using the DuckDB and the os libraries. The os library is used to list all files in the directory and filters for `csv.gz` files. DuckDB is used to ingest as a table where all columns are assigned dtype `string`. The tables are then written as `parquet`.
- `parquet_to_lazydict`: Creates a dictionary of `pl.LazyFrames` using the Polars and os libraries. The os library is used to list all files in the directory and filters for `parquet` files. Polars is used to ingest as a `pl.LazyFrame` by scanning all parquets in a directory and appending them to a dictionary where each key is the filename.
- `lazy_to_df`: Converts `pl.LazyFrames` into a dictionary to `pd.DataFrames` that start with a prefix. The `pd.DataFrame` is a Pandas DataFrame not a Polars DataFrame (`pd.DataFrame`).
- `sid_admissions`: Processes the Admissions LazyFrame by filtering rows by subject ID, selecting columns, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_patient`: Processes the Patient LazyFrame by filtering rows by subject ID, selecting columns, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_diagnosis`: Processes the Diagnosis and Diagnosis Key LazyFrames by filtering rows by subject ID, joining with the key set, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_labevents`: Processes the Lab Events LazyFrame by filtering rows by subject ID and item IDs, selecting columns, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_procedures`: Processes the Procedures and Procedures Key LazyFrames by filtering by subject ID, joining with the key set, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_transfer`: Processes the Transfers LazyFrame by filtering rows by subject ID, excluding discharge events, selecting columns, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `sid_chartevents`: This function processes the Chartevents and Chartevents key LazyFrames by filtering rows based on a subject ID and item IDs, joining with the key set, and assigning column dtypes. The processed LazyFrame is then added to the dictionary with a new key.
- `create_lazy_and_plot_df`: This function creates a dictionary of LazyFrames and a dictionary of dataframes. The lazyframe dictionary contains all datasets including plot datasets. The dataframe dictionary containes collected lazyframes needed to make plots for an subject ID. First, the lazyframe dictionary is created from all Parquet files in a directory. Then the lazyframes that will be used to make plots are prepared using the sid_{dataset}() functions which filter, select, assign types, and/or join keys. There is also an error checker for robustness that returns all of the all of the names that are required but not included as a key in lazydict.

[177]:
```
def csvgz_to_parquet(dir_list):
    """
```

```
    Convert all csv.gz files in a directory list to parquet files.
    All columns are dtype string.
    ---
    Args:
        dir_list: A list of the realtive or absolute directory paths.
    Return:
        None
    """
    # loop through user provided list
    for path in dir_list:
        # Loop through all files in directory
        for filename in os.listdir(path):
            # Ignore dot-files
            if filename.startswith("."):
                continue
            # if filename ends with ".csv.gz"
            if filename.endswith(".csv.gz"):
                # path to .csv.gz
                file_path = os.path.join(path, filename)
                # parquet name
                pq_name = filename.split(".")[0] + ".parquet"
                # parquet save path
                pq_path = os.path.join(path, pq_name)
                print(file_path, " to ", pq_path)  # Verbose step
                with duckdb.connect(database=":memory:") as con:
                    # Read in the compressed file from .csv.gz file path
                    # Set All columns to type string with ALL_VARCHAR
                    # Set ___ to null
                    # Write table as Parquet
                    con.execute(
                        f"""
                            COPY (
                                SELECT * FROM read_csv_auto(
                                    '{file_path}',
                                    ALL_VARCHAR=True
                                )
                            )
                            TO '{pq_path}' (FORMAT PARQUET);
                        """
                    )


def parquet_to_lazydict(dir_path):
    """
    Read in all parquet files in a directory with Polars as LazyFrames.
    ---
    Args:
```

```
        dir_path: A string of the realtive or absolute directory path.
    Return:
        lazy_dict: A dictionary of LazyFrames.
    """
    # Initialize lazyframe dictionary
    lazy_dict = dict()
    # Loop through every file in directory
    for file in os.listdir(dir_path):
        # Select files that end with parquet
        if file.endswith(".parquet"):
            # Path to each parquet
            name = file.split(".")[0]
            file_path = os.path.join(dir_path, file)
            # append each entry to lazy_dict
            # key is file name and value is lazyframe
            lazy_dict[name] = pl.scan_parquet(file_path)
    return lazy_dict


def lazy_to_df(lazy_dict, prefix):
    """
    Convert pl.LazyFrames to pl.DataFrames in a dictionary where the key starts
    with the prefix.
    ---
    Args:
        lazy_dict: Dictionary of Polars LazyFrames
        prefix: String and prefix of key in dictionary
    Return:
        df_dict: A dictionary of Polars DataFrames
    """
    df_dict = dict()  # init dict
    # Loop through all values in lazyframe dictionary
    for key, value in lazy_dict.items():
        # Filter LazyFrames by keys that start with the prefix
        if key.startswith(prefix):
            # Verbose for Debugging
            print(f"Collecting {key} LazyFrame ...")
            # Convert pl.LazyFrame to pl.DataFrame and Save to dictionary
            df_dict[key] = value.collect().to_pandas()
    return df_dict
```

[178]:
```
def chartevents(lazy_dict, items):
    """
    Assign column types and short title to the Chartevents LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
```

```
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    # Item Filter
    items = [str(item) for item in items]
    # Patient chartevents
    lazy_dict["chartevents"] = (
        # Chartevents dataset
        lazy_dict["chartevents"].filter(pl.col("itemid").is_in(items))
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "stay_id": pl.Int64,
                "caregiver_id": pl.Int64,
                "itemid": pl.Int64,
                "value": pl.String,
                "valuenum": pl.Float64,
                "valueuom": pl.String,
                "warning": pl.String,
            }
        )
        # Convert columns to datetime
        .with_columns(
            pl.col("charttime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("storetime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
        )
    )
    return lazy_dict


def d_items(lazy_dict):
    """
    Assign column types and short title to the Chartevents Key LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    # Chartevents item key
    lazy_dict["d_items"] = lazy_dict["d_items"].cast(
        {
            "itemid": pl.Int64,
            "label": pl.String,
            "abbreviation": pl.String,
```

```
            "unitname": pl.String,
        }
    )
    return lazy_dict


def transfers(lazy_dict):
    """
    Assign column types and short title to the Transfers LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["transfers"] = (
        # transfers dataset
        lazy_dict["transfers"]
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "transfer_id": pl.Int64,
                "eventtype": pl.String,
                "careunit": pl.String,
            }
        )
        # Convert string columns to date
        .with_columns(
            pl.col("intime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("outtime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
        )
    )
    return lazy_dict


def d_icd_procedures(lazy_dict):
    """
    Assign column types and short title to the Procedures Key LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    # preparing procedure key
```

```
    lazy_dict["d_icd_procedures"] = (
        lazy_dict["d_icd_procedures"]
        .cast(
            {
                "icd_code": pl.String,
                "long_title": pl.String,
            }
        )
        .with_columns(
            pl.col("long_title").str.extract(r"^[^,]*", 0).alias("short_title"),
        )
    )
    return lazy_dict


def procedures_icd(lazy_dict):
    """
    Assign column types to the Procedures LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["procedures_icd"] = (
        # procedure dataset
        lazy_dict["procedures_icd"]
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "seq_num": pl.Int64,
                "icd_code": pl.String,
                "chartdate": pl.Date,
                "icd_version": pl.Int64,
            }
        )
    )
    return lazy_dict


def d_icd_diagnoses(lazy_dict):
    """
    Assign column typesand add a short title to the Diagnosis Key LazyFrame.
    ---
    Args:
```

```
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["d_icd_diagnoses"] = (
        # diagnosis dataset
        lazy_dict["d_icd_diagnoses"]
        # Change column dtype
        .cast(
            {
                "icd_code": pl.String,
                "icd_version": pl.Int64,
                "long_title": pl.String,
            }
        )
        # Make a short title column
        .with_columns(
            pl.col("long_title").str.extract(r"^[^,]*", 0).alias("short_title"),
        )
    )
    return lazy_dict


def diagnoses_icd(lazy_dict):
    """
    Assign column types to the Diagnosis LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["diagnoses_icd"] = (
        # diagnosis dataset
        lazy_dict["diagnoses_icd"]
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "seq_num": pl.Int64,
                "icd_code": pl.String,
                "icd_version": pl.Int64,
            }
        )
    )
    return lazy_dict
```

```python
def labevents(lazy_dict, items):
    """
    Assign column types to the Labevents LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    # Items to filter by
    items = [str(item) for item in items]

    # Load in Lab Events
    lazy_dict["labevents"] = (
        # Labevents dataset
        lazy_dict["labevents"]
        # Change column dtype
        .filter(pl.col("itemid").is_in(items)).cast(
            {
                "labevent_id": pl.Int64,
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "specimen_id": pl.Int64,
                "itemid": pl.Int64,
                "valuenum": pl.Float64,
                "order_provider_id": pl.String,
                "value": pl.String,
                "valueuom": pl.String,
                "ref_range_lower": pl.Float64,
                "ref_range_upper": pl.Float64,
                "flag": pl.String,
                "priority": pl.String,
                "comments": pl.String,
            }
        )
        # Convert dtype to datetime
        .with_columns(
            pl.col("charttime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("storetime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
        )
    )
    return lazy_dict


def d_labitems(lazy_dict):
```

```
    """
    Assign column types to the Lab Item Key LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["d_labitems"] = lazy_dict["d_labitems"].cast(
        {"itemid": pl.Int64, "label": pl.String}
    )
    return lazy_dict


def patients(lazy_dict):
    """
    Assign column types to the Patient LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """

    lazy_dict["patients"] = (
        # Patient Info Dataset
        lazy_dict["patients"]
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "gender": pl.String,
                "anchor_age": pl.Int64,
                "anchor_year": pl.Int64,
                "anchor_year_group": pl.String,
                "dod": pl.Date,
            }
        )
    )
    return lazy_dict


def admissions(lazy_dict):
    """
    Assign column types to the Admissions LazyFrame.
    ---
    Args:
```

```
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["admissions"] = (
        # Admissions Dataset
        lazy_dict["admissions"]
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "admission_type": pl.String,
                "admit_provider_id": pl.String,
                "admission_location": pl.String,
                "discharge_location": pl.String,
                "insurance": pl.String,
                "language": pl.String,
                "marital_status": pl.String,
                "race": pl.String,
                "hospital_expire_flag": pl.String,
            }
        )
        # Change column dtype to Date
        .with_columns(
            pl.col("admittime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("dischtime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("deathtime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("edregtime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("edouttime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
        )
    )
    return lazy_dict


def icustays(lazy_dict):
    """
    Assign column types to the Admissions LazyFrame.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["icustays"] = (
        # Admissions Dataset
        lazy_dict["icustays"]
```

```
        # Change column dtype
        .cast(
            {
                "subject_id": pl.Int64,
                "hadm_id": pl.Int64,
                "stay_id": pl.Int64,
                "first_careunit": pl.String,
                "last_careunit": pl.String,
                "los": pl.Float64,
            }
        ).with_columns(
            pl.col("intime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
            pl.col("outtime").str.to_datetime("%Y-%m-%d %H:%M:%S"),
        )
    )
    return lazy_dict
```

[179]:
```
def sid_patient(lazy_dict, sid):
    """
    Filter rows, select columns, join admissions demographic information,
    and assign dtypes to the Patient dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["plot_patient"] = (
        # Patient Info Dataset
        lazy_dict["patients"]
        # Filter by sid
        .filter(pl.col("subject_id") == sid)
        # Join with admissions for demographic info
        .join(lazy_dict["admissions"], on="subject_id")
    )
    return lazy_dict


def sid_diagnosis(lazy_dict, sid):
    """
    Filter rows and join item key to the Diagnosis dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
```

```
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["plot_diagnosis"] = (
        # diagnosis dataset
        lazy_dict["diagnoses_icd"]
        # Filter by sid
        .filter(pl.col("subject_id") == sid)
        # Join diagnosis key
        .join(lazy_dict["d_icd_diagnoses"], on="icd_code", how="left")
    )
    return lazy_dict


def sid_labevents(lazy_dict, sid, items):
    """
    Filter rows, select columns, and assign dtypes to the Lab Events dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["plot_lab"] = (
        # Labevents dataset
        lazy_dict["labevents"]
        # select columns to keep
        .select(
            [
                "labevent_id",
                "subject_id",
                "hadm_id",
                "specimen_id",
                "itemid",
                "order_provider_id",
                "charttime",
                "storetime",
                "valuenum",
                "valueuom",
            ]
        )
        # Filter by sid and itemid
        .filter(pl.col("subject_id") == sid, pl.col("itemid").is_in(items))
        # Add column for y-axis
        .with_columns(pl.lit("Lab Events").alias("Data")).join(
            lazy_dict["d_labitems"].filter(pl.col("itemid").is_in(items)),
            on="itemid",
```

```
            how="left",
        )
    )
    return lazy_dict


def sid_procedures(lazy_dict, sid):
    """
    Filter rows, select columns, join item key, and assign dtypes
    to the Procedures dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["plot_procedures"] = (
        # procedure dataset
        lazy_dict["procedures_icd"]
        # select columns to keep
        .select(["subject_id", "hadm_id", "seq_num", "chartdate", "icd_code"])
        # Filter by sid
        .filter(pl.col("subject_id") == sid)
        # Join procedures key
        .join(lazy_dict["d_icd_procedures"], on="icd_code", how="left")
        # add column for naming y-axis
        .with_columns(
            pl.lit("Procedures").alias("Data"),
        )
    )
    return lazy_dict


def sid_transfer(lazy_dict, sid):
    """
    Filter rows, select columns, and assign dtypes to the Transfers dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    lazy_dict["plot_transfer"] = (
        # transfers dataset
        lazy_dict["transfers"]
```

```
        # filter by sid
        .filter(pl.col("subject_id") == sid, pl.col("eventtype") != "discharge")
        # # Change column dtype
        # .cast(
        #     {
        #         "intime": pl.Date,
        #         "outtime": pl.Date,
        #     }
        # )
        .with_columns(
            # Add a bivariate critical care column if in ICU or CCU
            (
                pl.when(pl.col("careunit").str.contains("ICU|CCU"))
                .then(pl.lit("Yes"))
                .otherwise(pl.lit("No"))
                .alias("critical_care")
            ),
            # Add a column ADT with values of ADT
            pl.lit("ADT").alias("Data"),
        )
        .sort("intime", descending=False)
    )
    return lazy_dict


def sid_chartevents(lazy_dict, sid, items):
    """
    Filter rows, select columns, join item key, and assign dtypes
    to the Chartevents dataset.
    ---
    Args:
        lazy_dict: A dictionary of Polars LazyFrames
        sid: An integer of digits corresponding to the Subject ID.
    Return:
        lazy_dict: A dictionary of Polars LazyFrames and a processed plot table.
    """
    # Patient chartevents
    lazy_dict["plot_chartevents"] = (
        # Chartevents dataset
        lazy_dict["chartevents"]
        # Filter by sid and itemid
        .filter(
            pl.col("subject_id") == sid,
            pl.col("itemid").is_in(items),
        )
        # Select columns to keep
        .select(
```

```
            [
                "subject_id",
                "stay_id",
                "charttime",
                "itemid",
                "valuenum",
                "valueuom",
            ]
        )
        # join item key
        .join(
            lazy_dict["d_items"].filter(pl.col("itemid").is_in(items)),
            on="itemid",
            how="left",
        )
    )
    return lazy_dict
```

[180]:
```
def make_lazy_dict(sid, dir_path, lab_items, chart_items):
    """
    Create a dictionary with of LazyFrames of all parquet files in a directory.
    Format, filter, and prepare data for use in plots.
    Convert plot lazyframes to dataframes.
    ---
    Args:
        sid: An integer representing a subject ID.
    Returns:
        lazy_dict: A dictionary of LazyFrames
        df_dict: A dictionary of DataFrames
    """
    # Create a dictionary with of LazyFrames of all parquet files in a directory
    lazy_dict = parquet_to_lazydict(dir_path)
    # list of names converted to lazyframe from directory.
    lazy_names = [key for key in lazy_dict]
    # name requirements.
    req_names = [
        "patients",
        "admissions",
        "transfers",
        "procedures_icd",
        "d_icd_procedures",
        "diagnoses_icd",
        "d_icd_diagnoses",
        "labevents",
        "d_labitems",
        "chartevents",
        "d_items",
```

```
        "icustays",
    ]
    # list of all required names not in lazy_names
    missing_names = [n for n in req_names if n not in lazy_names]
    # Error Catch before performing filtering fucntions
    if missing_names:
        raise KeyError(f'Names not found: {", ".join(missing_names)}')
    # Fucntions to prepare datasets for graphing and verbose for debugging
    print("Processing Patient and Admissions ...")
    admissions(lazy_dict)
    patients(lazy_dict)
    sid_patient(lazy_dict, sid)
    print("Processing Transfer ...")
    transfers(lazy_dict)
    sid_transfer(lazy_dict, sid)
    print("Processing Procedures ...")
    procedures_icd(lazy_dict)
    d_icd_procedures(lazy_dict)
    sid_procedures(lazy_dict, sid)
    print("Processing Diagnosis ...")
    diagnoses_icd(lazy_dict)
    d_icd_diagnoses(lazy_dict)
    sid_diagnosis(lazy_dict, sid)
    print("Processing Labevents ...")
    labevents(lazy_dict, lab_items)
    d_labitems(lazy_dict)
    sid_labevents(lazy_dict, sid, lab_items)
    print("Processing Chartevents ...")
    chartevents(lazy_dict, chart_items)
    d_items(lazy_dict)
    sid_chartevents(lazy_dict, sid, chart_items)
    print("Processing ICU Stays ...")
    icustays(lazy_dict)
    # convert plot lazyframe to dataframe
    df_dict = lazy_to_df(lazy_dict, "plot")
    # return both lazyframe and plot dataframes
    print("DataFrame and LazyFrame Preparation Complete")
    return lazy_dict, df_dict
```

`.csv.gz` to `parquet` Documentation

- All `.csv.gz` files that were converted to `parquet` using `csvgz_to_parquet`
- DuckDB was faster than Polars. DuckDB took 4 minutes to convert `.csv.gz` files that were converted to `parquet` and Polars took 6 minutes to convert the same `.csv.gz` files that were converted to `parquet`.

[167]:
```
csvgz_paths = ["./mimic/hosp", "./mimic/icu"]
csvgz_to_parquet(csvgz_paths)
```

```
./mimic/hosp/poe.csv.gz  to  ./mimic/hosp/poe.parquet
./mimic/hosp/d_hcpcs.csv.gz  to  ./mimic/hosp/d_hcpcs.parquet
./mimic/hosp/poe_detail.csv.gz  to  ./mimic/hosp/poe_detail.parquet
./mimic/hosp/patients.csv.gz  to  ./mimic/hosp/patients.parquet
./mimic/hosp/diagnoses_icd.csv.gz  to  ./mimic/hosp/diagnoses_icd.parquet
./mimic/hosp/emar_detail.csv.gz  to  ./mimic/hosp/emar_detail.parquet
./mimic/hosp/provider.csv.gz  to  ./mimic/hosp/provider.parquet
./mimic/hosp/prescriptions.csv.gz  to  ./mimic/hosp/prescriptions.parquet
./mimic/hosp/drgcodes.csv.gz  to  ./mimic/hosp/drgcodes.parquet
./mimic/hosp/d_icd_diagnoses.csv.gz  to  ./mimic/hosp/d_icd_diagnoses.parquet
./mimic/hosp/d_labitems.csv.gz  to  ./mimic/hosp/d_labitems.parquet
./mimic/hosp/transfers.csv.gz  to  ./mimic/hosp/transfers.parquet
./mimic/hosp/admissions.csv.gz  to  ./mimic/hosp/admissions.parquet
./mimic/hosp/labevents.csv.gz  to  ./mimic/hosp/labevents.parquet
./mimic/hosp/pharmacy.csv.gz  to  ./mimic/hosp/pharmacy.parquet
./mimic/hosp/procedures_icd.csv.gz  to  ./mimic/hosp/procedures_icd.parquet
./mimic/hosp/hcpcsevents.csv.gz  to  ./mimic/hosp/hcpcsevents.parquet
./mimic/hosp/services.csv.gz  to  ./mimic/hosp/services.parquet
./mimic/hosp/d_icd_procedures.csv.gz  to  ./mimic/hosp/d_icd_procedures.parquet
./mimic/hosp/omr.csv.gz  to  ./mimic/hosp/omr.parquet
./mimic/hosp/emar.csv.gz  to  ./mimic/hosp/emar.parquet
./mimic/hosp/microbiologyevents.csv.gz  to
./mimic/hosp/microbiologyevents.parquet
./mimic/icu/datetimeevents.csv.gz  to  ./mimic/icu/datetimeevents.parquet
./mimic/icu/caregiver.csv.gz  to  ./mimic/icu/caregiver.parquet
./mimic/icu/ingredientevents.csv.gz  to  ./mimic/icu/ingredientevents.parquet
./mimic/icu/inputevents.csv.gz  to  ./mimic/icu/inputevents.parquet
./mimic/icu/procedureevents.csv.gz  to  ./mimic/icu/procedureevents.parquet
./mimic/icu/d_items.csv.gz  to  ./mimic/icu/d_items.parquet
./mimic/icu/chartevents.csv.gz  to  ./mimic/icu/chartevents.parquet
./mimic/icu/icustays.csv.gz  to  ./mimic/icu/icustays.parquet
./mimic/icu/outputevents.csv.gz  to  ./mimic/icu/outputevents.parquet
```

[168]:
```
%%bash
# Modify bash code to match directories.
ln -sf /media/wtmartinez/Data/mimic ./
ln -sf ./mimic/hosp/patients.parquet ./patients.parquet
ln -sf ./mimic/hosp/admissions.parquet ./admissions.parquet
ln -sf ./mimic/hosp/transfers.parquet ./transfers.parquet
ln -sf ./mimic/hosp/labevents.parquet ./labevents.parquet
ln -sf ./mimic/hosp/procedures_icd.parquet ./procedures_icd.parquet
ln -sf ./mimic/hosp/diagnoses_icd.parquet ./diagnoses_icd.parquet
ln -sf ./mimic/hosp/d_icd_procedures.parquet ./d_icd_procedures.parquet
ln -sf ./mimic/hosp/d_icd_diagnoses.parquet ./d_icd_diagnoses.parquet
```

```
ln -sf ./mimic/hosp/d_labitems.parquet ./d_labitems.parquet
ln -sf ./mimic/icu/icustays.parquet ./icustays.parquet
ln -sf ./mimic/icu/chartevents.parquet ./chartevents.parquet
ln -sf ./mimic/icu/d_items.parquet ./d_items.parquet
echo Confirm All Links are Correct:
find . -type l -ls | awk '{print $(NF-2), $(NF-1), $NF}'
```

```
Confirm All Links are Correct:
./transfers.parquet -> ./mimic/hosp/transfers.parquet
./diagnoses_icd.parquet -> ./mimic/hosp/diagnoses_icd.parquet
./mimic -> /media/wtmartinez/Data/mimic
./d_icd_diagnoses.parquet -> ./mimic/hosp/d_icd_diagnoses.parquet
./icustays.parquet -> ./mimic/icu/icustays.parquet
./admissions.parquet -> ./mimic/hosp/admissions.parquet
./d_icmd_diagnoses.parquet -> ./mimic/hosp/d_icd_diagnoses.parquet
./chartevents.parquet -> ./mimic/icu/chartevents.parquet
./d_items.parquet -> ./mimic/icu/d_items.parquet
./procedures_icd.parquet -> ./mimic/hosp/procedures_icd.parquet
./d_labitems.parquet -> ./mimic/hosp/d_labitems.parquet
./patients.parquet -> ./mimic/hosp/patients.parquet
./labevents.parquet -> ./mimic/hosp/labevents.parquet
./d_icd_procedures.parquet -> ./mimic/hosp/d_icd_procedures.parquet
```

1.1 Problem 1. Visualizing patient trajectory

Visualizing a patient's encounters in a health care system is a common task in clinical data analysis. In this question, we will visualize a patient's ADT (admission-discharge-transfer) history and ICU vitals in the MIMIC-IV data.

1.1.1 (A). ADT history

A patient's ADT history records the time of admission, discharge, and transfer in the hospital. This figure shows the ADT history of the patient with subject_id `10001217` in the MIMIC-IV data. The x-axis is the calendar time, and the y-axis is the type of event (ADT, lab, procedure). The color of the line segment represents the care unit. The size of the line segment represents whether the care unit is an ICU/CCU. The crosses represent lab events, and the shape of the dots represents the type of procedure. The title of the figure shows the patient's demographic information and the subtitle shows top 3 diagnoses. Try to create a figure similar to the below:

Your figure does not need to be the same, but all the information in this figure should be reasonably arranged in your figure. Hint: consider using `dodge` keyword arguments of seaborn to do something similar to `jitter` of `ggplot2`.

Hint: We need to pull information from data files `patients.csv.gz`, `admissions.csv.gz`, `transfers.csv.gz`, `labevents.csv.gz`, `procedures_icd.csv.gz`, `diagnoses_icd.csv.gz`, `d_icd_procedures.csv.gz`, and `d_icd_diagnoses.csv.gz`. For the big file `labevents.csv.gz`, use the Parquet file you generated in Homework 3. More information is available in later problems.

For reproducibility, make the Parquet file available at the current working directory, for example, by a symbolic link. Make your code reproducible using relative path.

Do a similar visualization for the patient `10013310`.

Symbolic Link Documentation

- Created Symbolic links from mimic data directories to the working directory. Only performed for the files that were needed to make the plot. Change the Directories to match your system.

Reading and Preparing Data Documentation

- The `create_lazy_and_plot_df` function was used to make a dictionary of lazyframes of all datasets and a dictionary of dataframes of processed plotting datasets.

Answer 1a To make the plots below the requred datasets were "patients", "admissions", "transfers", "procedures_icd", "d_icd_procedures", "diagnoses_icd", "d_icd_diagnoses", "labevents", and "d_labitems". All these variables were processed by calling `make_lazy_dict`, a function of functions that assign types, and filter itemids and subject ids and collects the plotting lazyframes to pandas dataframes.

The plot used matplotlib and seaborn plots to make the ICU stay plot. To make the ICU care units, a for loop was used to creat segments of lines and these lines were appended to the plot. Lab events and procedures were plotted using a stripplot. A custom color pallet was used to give color to the ICU care units because appending them to the plot they were viewed as independent plots. Therefore a dictionary assigned colors to names of care units and procedures. It also allowed for a custom key to be made that had only the required information.

Note that a shortcomming of the plot because is that it is hard to see smaller units like emergency room.

[171]:
```
%%time
sid = 10013310
pq_path = "./"
lab_items = [50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931]
chart_items = [220045, 220179, 220180, 223761, 220210]

lazy_dict, df_dict = make_lazy_dict(sid, pq_path, lab_items, chart_items)
```

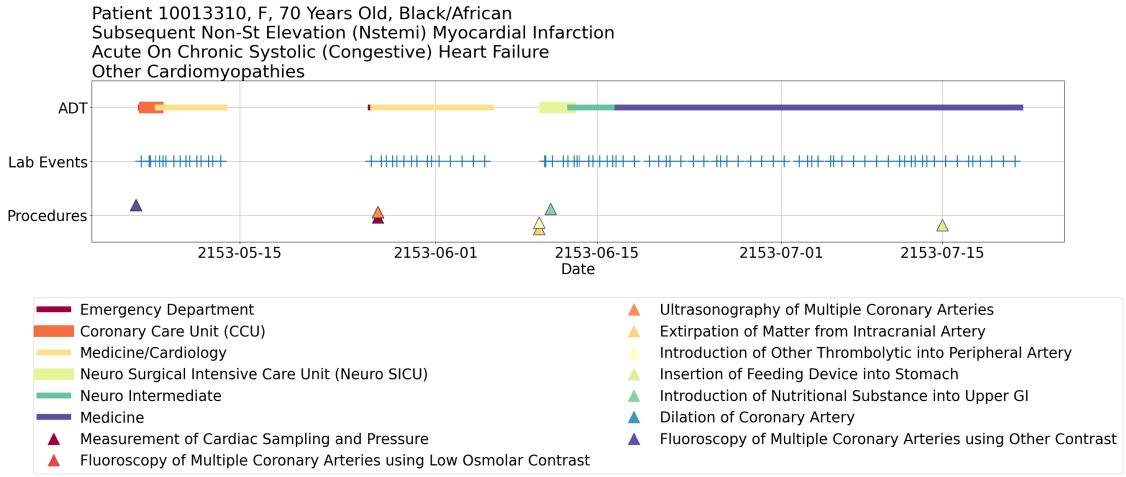
```
Processing Patient and Admissions …
Processing Transfer …
Processing Procedures …
Processing Diagnosis …
Processing Labevents …
Processing Chartevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_transfer LazyFrame …
Collecting plot_procedures LazyFrame …
Collecting plot_diagnosis LazyFrame …
Collecting plot_lab LazyFrame …
Collecting plot_chartevents LazyFrame …
DataFrame and LazyFrame Preparation Complete
```

```
CPU times: user 1min 32s, sys: 16 s, total: 1min 48s
Wall time: 9.34 s
```

[172]:
```
%%time
# plt.figure(figsize=(50, 5))
plt.figure(figsize=(30, 5))

# number of unique units
unq_unit = df_dict["plot_transfer"]["careunit"].unique()
# colors for number of of unique units
spectral = mpl.colormaps["Spectral"].resampled(len(unq_unit))
# dict comprehension to assign units to colors
unit_palette = {unit: spectral(i) for i, unit in enumerate(unq_unit)}

# number of unique units
unq_pro = df_dict["plot_procedures"]["short_title"].unique()
# colors for number of of unique units
spectral = mpl.colormaps["Spectral"].resampled(len(unq_pro))
# dict comprehension to assign units to colors
pro_palette = {unit: spectral(i) for i, unit in enumerate(unq_pro)}

for i, row in df_dict["plot_transfer"].iterrows():
    # custom line segments of careunit to make a segemented line plot
    row_dict = {
        "Date": [row["intime"], row["outtime"]],
        "Events": [row["Data"], row["Data"]],
        "critical_care": [row["critical_care"], row["critical_care"]],
        "careunit": [row["careunit"], row["careunit"]],
    }
    row_df = pd.DataFrame(row_dict) # to pandas df
# line plot for care unit
    sns.lineplot(
        data=row_df,
        x="Date",
        y="Events",
        hue="careunit",
        size="critical_care",
        markers=True,
        sizes={
            "Yes": 20,
            "No": 10, # width of line for ICU or not
        },
        palette=unit_palette, # custom pallete
        legend=False, # no ledgend
    )
# stripplot for lab measuremnts
sns.stripplot(
```

```
    data=df_dict["plot_lab"],
    x="charttime",
    y="Data",
    hue="Data",
    dodge=False,
    jitter=False, # no jitter
    # alpha=1,
    s=20, # marker size
    marker="+", # marker shape
    linewidth=1, # marker width
    legend=False, # no ledgend
)

# stripplot for jitter for procedures
sns.stripplot(
    data=df_dict["plot_procedures"],
    x="chartdate",
    y="Data",
    hue="short_title",
    palette=pro_palette, # custom color pallete
    dodge=False, # no dodge
    jitter=0.3, # jitter magnitude
    s=20, # size of marker
    marker="^", # arrow shape
    linewidth=1,
    legend=False,
)


marker_style = list()
marker_key = list()
# custom key for care unit
for unit in unq_unit:
    marker_key.append(unit) # name
    marker_style.append(
        Line2D( # line symbol
            [0],
            [0],
            color=unit_palette[unit], # Pallete dictionary value
            linewidth=20 if re.search("ICU|CCU", unit) else 10, # Thickness
        )
    )
# Making custom key for procedure
for procedure in unq_pro:
    marker_key.append(procedure) # append procedure
    marker_style.append(
        Line2D( # symbol
```

```
            [0],
            [0],
            color=pro_palette[procedure], # same palette dictionary
            marker="^", # marker is an arrow
            linestyle="None", # not a line
            markersize=20,
        )
    )

# select columns
pat = df_dict["plot_patient"]
dia = df_dict["plot_diagnosis"]
# patient info title
title = (
    f"Patient {pat['subject_id'][0]}, {pat['gender'][0]},␣
 ↪{pat['anchor_age'][0]} Years Old, {pat['race'][0].title()}"
    + f"\n{dia['short_title'][0].title()}\n{dia['short_title'][1].
 ↪title()}\n{dia['short_title'][2].title()}"
)
plt.title(title, loc="left") # title and justification

plt.ylabel(None)
# custom legend
plt.legend(
    marker_style, # marker list
    marker_key, # name list
    loc="upper center", # justification
    bbox_to_anchor=(0.5, -0.3), # location
    ncol=2 # columns
)
plt.rcParams.update({"font.size": 25}) # font size
plt.grid() # add grid

plt.show()# show plot
```




```
CPU times: user 334 ms, sys: 20.4 ms, total: 355 ms
Wall time: 352 ms
```

[173]:
```python
def icu_plot(sid, dir_path, lab_items, chart_items):
    """
    Process dataframes and plot a patient's ICU stay.
    ---
    Args:
        sid: An integer representing a subject ID.
    Returns:
        None
    """
    # Create a dictionary with of LazyFrames of all parquet files in a directory
    lazy_dict = parquet_to_lazydict(dir_path)
    # list of names converted to lazyframe from directory.
    lazy_names = [key for key in lazy_dict]
    # name requirements.
    req_names = [
        "patients",
        "admissions",
        "transfers",
        "procedures_icd",
        "d_icd_procedures",
        "diagnoses_icd",
        "d_icd_diagnoses",
        "labevents",
        "d_labitems",
        "chartevents",
        "d_items",
        "icustays",
    ]
```

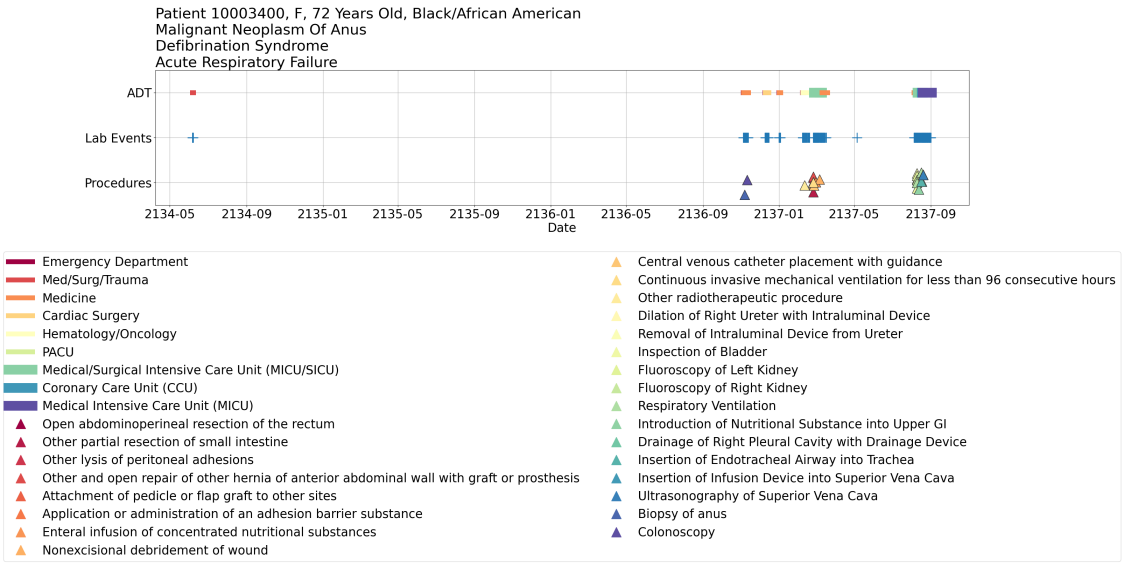
```
    # list of all required names not in lazy_names
    missing_names = [n for n in req_names if n not in lazy_names]
    # Error Catch before performing filtering fucntions
    if missing_names:
        raise KeyError(f'Names not found: {", ".join(missing_names)}')
    # Fucntions to prepare datasets for graphing and verbose for debugging
    print("Processing Patient and Admissions ...")
    admissions(lazy_dict)
    patients(lazy_dict)
    sid_patient(lazy_dict, sid)
    print("Processing Transfer ...")
    transfers(lazy_dict)
    sid_transfer(lazy_dict, sid)
    print("Processing Procedures ...")
    procedures_icd(lazy_dict)
    d_icd_procedures(lazy_dict)
    sid_procedures(lazy_dict, sid)
    print("Processing Diagnosis ...")
    diagnoses_icd(lazy_dict)
    d_icd_diagnoses(lazy_dict)
    sid_diagnosis(lazy_dict, sid)
    print("Processing Labevents ...")
    labevents(lazy_dict, lab_items)
    d_labitems(lazy_dict)
    sid_labevents(lazy_dict, sid, lab_items)
    print("Processing ICU Stays ...")
    icustays(lazy_dict)
    # convert plot lazyframe to dataframe
    df_dict = lazy_to_df(lazy_dict, "plot")
    # return both lazyframe and plot dataframes
    print("DataFrame and LazyFrame Preparation Complete")
    ### plot
    # plt.figure(figsize=(50, 5))
    plt.figure(figsize=(30, 5))

    # number of unique units
    unq_unit = df_dict["plot_transfer"]["careunit"].unique()
    # colors for number of of unique units
    spectral = mpl.colormaps["Spectral"].resampled(len(unq_unit))
    # dict comprehension to assign units to colors
    unit_palette = {unit: spectral(i) for i, unit in enumerate(unq_unit)}

    # number of unique units
    unq_pro = df_dict["plot_procedures"]["short_title"].unique()
    # colors for number of of unique units
    spectral = mpl.colormaps["Spectral"].resampled(len(unq_pro))
    # dict comprehension to assign units to colors
```

```
pro_palette = {unit: spectral(i) for i, unit in enumerate(unq_pro)}

for i, row in df_dict["plot_transfer"].iterrows():
    # custom line segments of careunit to make a segemented line plot
    row_dict = {
        "Date": [row["intime"], row["outtime"]],
        "Events": [row["Data"], row["Data"]],
        "critical_care": [row["critical_care"], row["critical_care"]],
        "careunit": [row["careunit"], row["careunit"]],
    }
    row_df = pd.DataFrame(row_dict) # to pandas df
# line plot for care unit
    sns.lineplot(
        data=row_df,
        x="Date",
        y="Events",
        hue="careunit",
        size="critical_care",
        markers=True,
        sizes={
            "Yes": 20,
            "No": 10, # width of line for ICU or not
        },
        palette=unit_palette, # custom pallete
        legend=False, # no ledgend
    )
# stripplot for lab measuremnts
sns.stripplot(
    data=df_dict["plot_lab"],
    x="charttime",
    y="Data",
    hue="Data",
    dodge=False,
    jitter=False, # no jitter
    # alpha=1,
    s=20, # marker size
    marker="+", # marker shape
    linewidth=1, # marker width
    legend=False, # no ledgend
)

# stripplot for jitter for procedures
sns.stripplot(
    data=df_dict["plot_procedures"],
    x="chartdate",
    y="Data",
    hue="short_title",
```

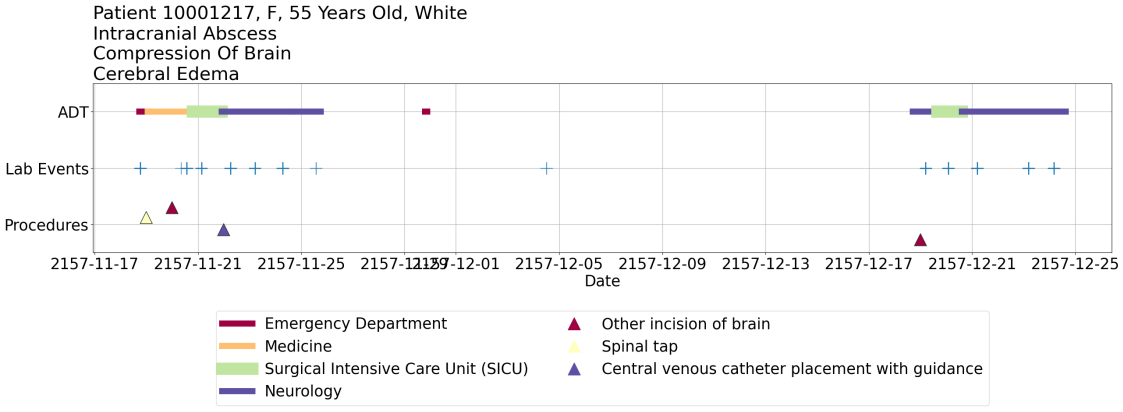
```
        palette=pro_palette, # custom color pallete
        dodge=False, # no dodge
        jitter=0.3, # jitter magnitude
        s=20, # size of marker
        marker="^", # arrow shape
        linewidth=1,
        legend=False,
    )


    marker_style = list()
    marker_key = list()
    # custom key for care unit
    for unit in unq_unit:
        marker_key.append(unit) # name
        marker_style.append(
            Line2D( # line symbol
                [0],
                [0],
                color=unit_palette[unit], # Pallete dictionary value
                linewidth=20 if re.search("ICU|CCU", unit) else 10, # Thickness
            )
        )
    # Making custom key for procedure
    for procedure in unq_pro:
        marker_key.append(procedure) # append procedure
        marker_style.append(
            Line2D( # symbol
                [0],
                [0],
                color=pro_palette[procedure], # same palette dictionary
                marker="^", # marker is an arrow
                linestyle="None", # not a line
                markersize=20,
            )
        )

    # select columns
    pat = df_dict["plot_patient"]
    dia = df_dict["plot_diagnosis"]
    # patient info title
    title = (
        f"Patient {pat['subject_id'][0]}, {pat['gender'][0]},␣
↪{pat['anchor_age'][0]} Years Old, {pat['race'][0].title()}"
        + f"\n{dia['short_title'][0].title()}\n{dia['short_title'][1].
↪title()}\n{dia['short_title'][2].title()}"
    )
```

```
    plt.title(title, loc="left") # title and justification

    plt.ylabel(None)
    # custom legend
    plt.legend(
        marker_style, # marker list
        marker_key, # name list
        loc="upper center", # justification
        bbox_to_anchor=(0.5, -0.3), # location
        ncol=2 # columns
    )
    plt.rcParams.update({"font.size": 25}) # font size
    plt.grid() # add grid

    plt.show()# show plot


icu_plot(10003400, pq_path, lab_items, chart_items)
icu_plot(10001217, pq_path, lab_items, chart_items)
icu_plot(10002155, pq_path, lab_items, chart_items)
```

```
Processing Patient and Admissions …
Processing Transfer …
Processing Procedures …
Processing Diagnosis …
Processing Labevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_transfer LazyFrame …
Collecting plot_procedures LazyFrame …
Collecting plot_diagnosis LazyFrame …
Collecting plot_lab LazyFrame …
DataFrame and LazyFrame Preparation Complete
```

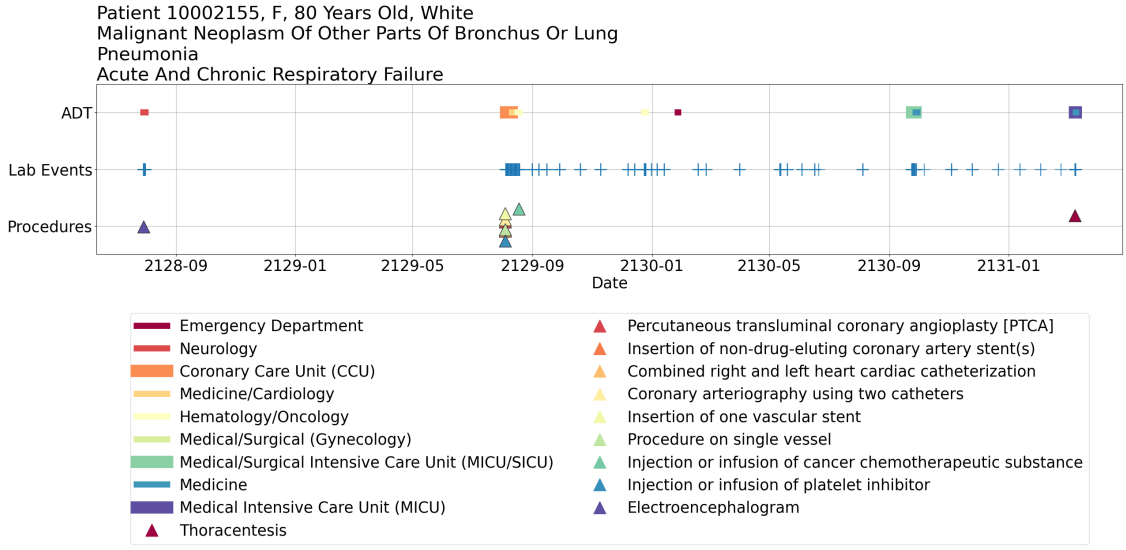



```
Processing Patient and Admissions …
Processing Transfer …
Processing Procedures …
Processing Diagnosis …
Processing Labevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_transfer LazyFrame …
Collecting plot_procedures LazyFrame …
Collecting plot_diagnosis LazyFrame …
Collecting plot_lab LazyFrame …
DataFrame and LazyFrame Preparation Complete
```




```
Processing Patient and Admissions …
Processing Transfer …
```

```
Processing Procedures …
Processing Diagnosis …
Processing Labevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_transfer LazyFrame …
Collecting plot_procedures LazyFrame …
Collecting plot_diagnosis LazyFrame …
Collecting plot_lab LazyFrame …
DataFrame and LazyFrame Preparation Complete
```



1.1.2 (B). ICU stays

ICU stays are a subset of ADT history. This figure shows the vitals of the patient `10001217` during ICU stays. The x-axis is the calendar time, and the y-axis is the value of the vital. The color of the line represents the type of vital. The facet grid shows the abbreviation of the vital and the stay ID. These vitals are: heart rate (220045), systolic non-invasive blood pressure (220179), diastolic non-invasive blood pressure (220180), body temperature in Fahrenheit (223761), and respiratory rate (220210). Try to create a figure similar to below:

Repeat a similar visualization for the patient `10013310`.

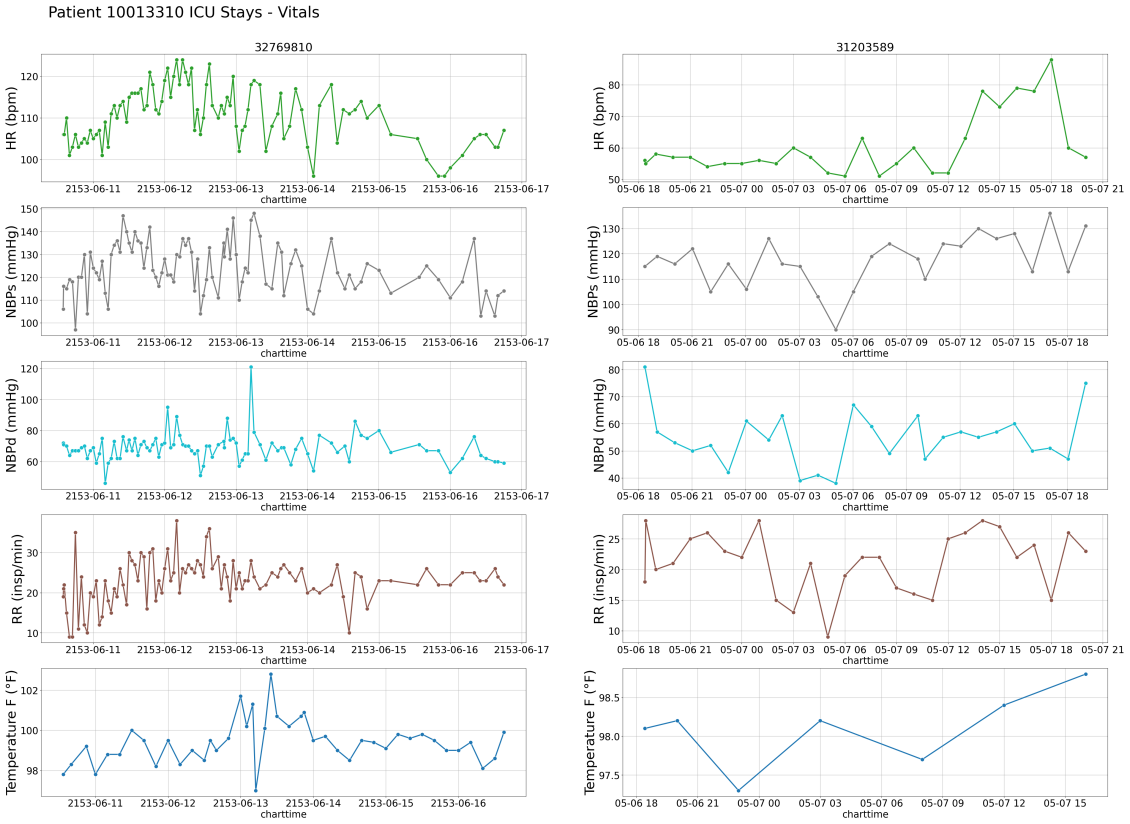
Answer 1b To make the plot below, the chartevents and d_items datasets were need to get the patient vitals and labels. All these variables were processed by calling make_lazy_dict, a function of functions that assign types, and filter itemids and subject ids and collects the plotting lazyframes to pandas dataframes.

The plot used matplotlib and seaborn plots to make the vitals facet plot. To make the facet plot, a for loop was used to format and assign plots to facets. Each column is an ICU stay and each row is a vital measurement. No axis were shared because it made the plot too small and was less

predictable.

[148]:
```
%%time

data = df_dict["plot_chartevents"] # chart events
patient = df_dict["plot_patient"]["subject_id"][0] # patient info
names = list(data["abbreviation"].unique()) # item names
items = list(data["itemid"].unique()) # items
df_items = ( # item key names
    lazy_dict["d_items"]
    .filter(pl.col("itemid").is_in(items))
    .collect()
    .to_dict()
)
# unique values
stays = list(data["stay_id"].unique())
# colors for number of unique values
spectral = mpl.colormaps["tab10"].resampled(len(items))
# dictionary comprehension for custom palette
chart_palette = {unit: spectral(i) for i, unit in enumerate(items)}

# init facet figure
fig, axes = plt.subplots(
    nrows=len(names), # cols = number of stays
    ncols=len(stays), # cols = number of stays
    sharex=False,
    sharey=False,
    squeeze=True,
    figsize=(50, 35),
)
# big facet title
fig.suptitle(
    f"Patient {patient} ICU Stays - Vitals", x=0.13, y=0.93, ha="left", size=40
)

# Facet titles
for i, stay in enumerate(stays):
    axes[0, i].set_title(f"{stays[i]}") # Stayid title
    for j, item in enumerate(df_items["itemid"]):
        abv = df_items["abbreviation"][j] # item name
        units = df_items["unitname"][j] # units
        axes[j, i].set_ylabel(f"{abv} ({units})", size=35) # y Axis title

# facet plotting
for i, item in enumerate(df_items["itemid"]):
    for j, stay in enumerate(stays):
        filter = f"itemid == {item} & stay_id == {stay}" # SQL
```

```
        df = data.query(filter) # SQL filter
        sns.lineplot( # plotting
            ax=axes[i, j], # facet location
            data=df,
            x="charttime",
            y="valuenum",
            hue="itemid",
            palette=chart_palette, # custom palette
            linewidth=3,
            legend=None, # no legend
            marker="o", # dot markers
            markersize=10,
            errorbar=None, # no band
        )
        axes[i, j].grid(True) # grids


plt.rcParams.update({"font.size": 25}) # font size
plt.show() # show plot
```



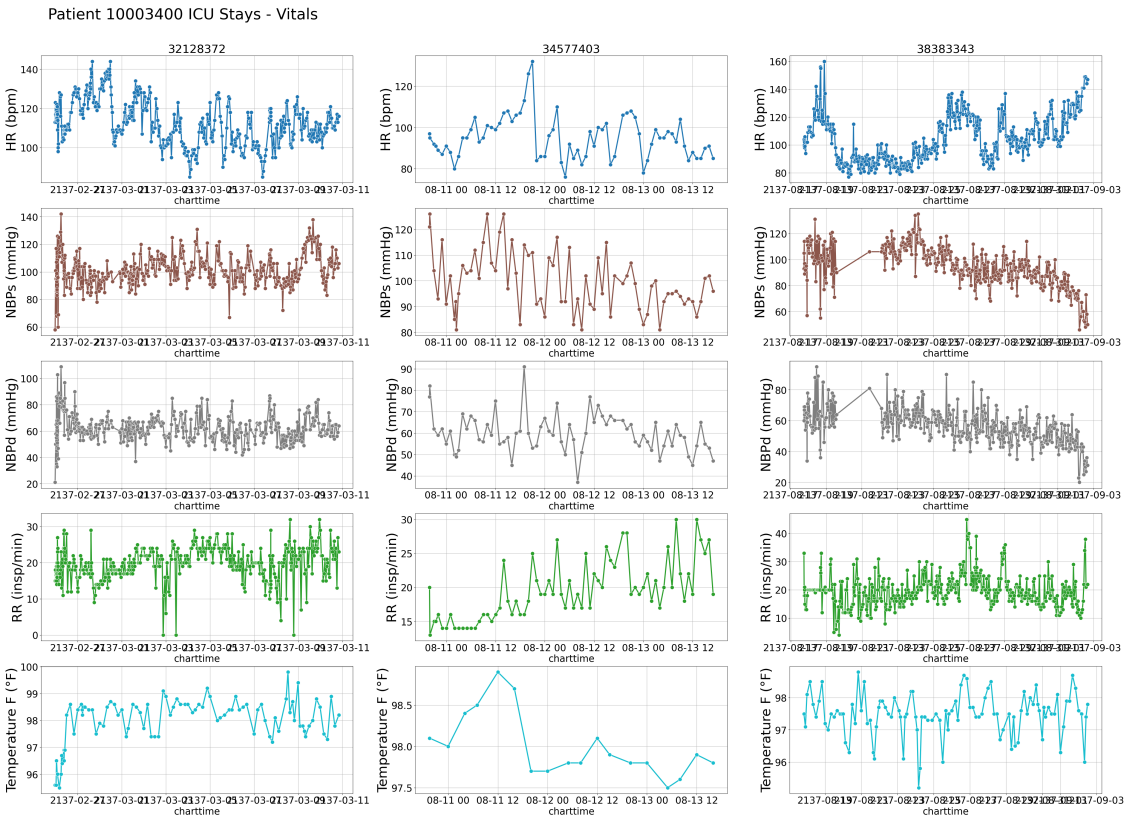

```
CPU times: user 1.18 s, sys: 64.7 ms, total: 1.25 s
Wall time: 1.24 s
```

[154]:
```python
def chartplot(sid, dir_path, lab_items, chart_items):
    """
    Process chart events and plot a facet plot of chartevents
    ---
    Args:
        sid: An integer representing a subject ID.
    Returns:
        none
    """
    # Create a dictionary with of LazyFrames of all parquet files in a directory
    lazy_dict = parquet_to_lazydict(dir_path)
    # list of names converted to lazyframe from directory.
    lazy_names = [key for key in lazy_dict]
    # name requirements.
    req_names = [
        "patients",
        "admissions",
        "transfers",
        "procedures_icd",
        "d_icd_procedures",
        "diagnoses_icd",
        "d_icd_diagnoses",
        "labevents",
        "d_labitems",
        "chartevents",
        "d_items",
        "icustays",
    ]
    # list of all required names not in lazy_names
    missing_names = [n for n in req_names if n not in lazy_names]
    # Error Catch before performing filtering fucntions
    if missing_names:
        raise KeyError(f'Names not found: {", ".join(missing_names)}')
    # Fucntions to prepare datasets for graphing and verbose for debugging
    print("Processing Patient and Admissions ...")
    admissions(lazy_dict)
    patients(lazy_dict)
    sid_patient(lazy_dict, sid)
    print("Processing Chartevents ...")
    chartevents(lazy_dict, chart_items)
    d_items(lazy_dict)
    sid_chartevents(lazy_dict, sid, chart_items)
    print("Processing ICU Stays ...")
    icustays(lazy_dict)
    # convert plot lazyframe to dataframe
    df_dict = lazy_to_df(lazy_dict, "plot")
    ###plot
```

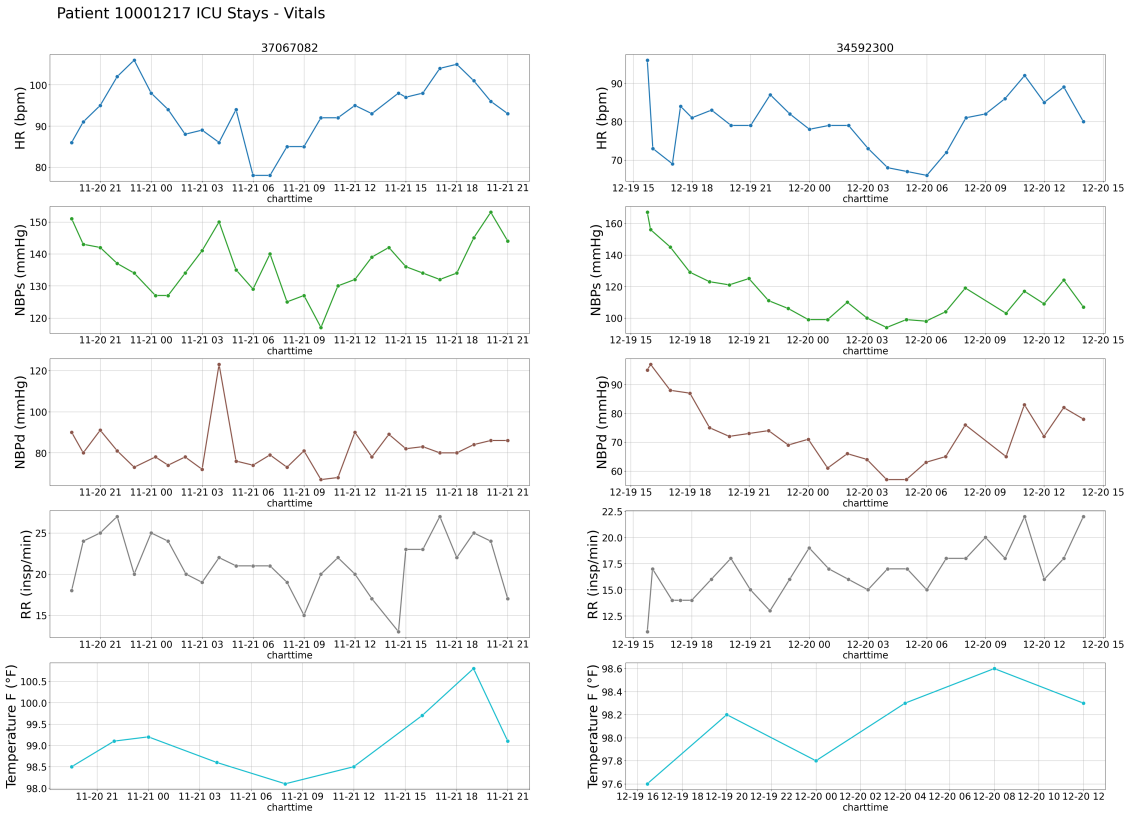
```
    data = df_dict["plot_chartevents"] # chart events
    patient = df_dict["plot_patient"]["subject_id"][0] # patient info
    names = list(data["abbreviation"].unique()) # item names
    items = list(data["itemid"].unique()) # items
    df_items = ( # item key names
        lazy_dict["d_items"]
        .filter(pl.col("itemid").is_in(items))
        .collect()
        .to_dict()
    )
    # unique values
    stays = list(data["stay_id"].unique())
    # colors for number of unique values
    spectral = mpl.colormaps["tab10"].resampled(len(items))
    # dictionary comprehension for custom palette
    chart_palette = {unit: spectral(i) for i, unit in enumerate(items)}

    # init facet figure
    fig, axes = plt.subplots(
        nrows=len(names), # cols = number of stays
        ncols=len(stays), # cols = number of stays
        sharex=False,
        sharey=False,
        squeeze=True,
        figsize=(50, 35),
    )
    # big facet title
    fig.suptitle(
        f"Patient {patient} ICU Stays - Vitals", x=0.13, y=0.93, ha="left",␣
↪size=40
    )

    # Facet titles
    for i, stay in enumerate(stays):
        axes[0, i].set_title(f"{stays[i]}") # Stayid title
        for j, item in enumerate(df_items["itemid"]):
            abv = df_items["abbreviation"][j] # item name
            units = df_items["unitname"][j] # units
            axes[j, i].set_ylabel(f"{abv} ({units})", size=35) # y Axis title

    # facet plotting
    for i, item in enumerate(df_items["itemid"]):
        for j, stay in enumerate(stays):
            filter = f"itemid == {item} & stay_id == {stay}" # SQL
            df = data.query(filter) # SQL filter
            sns.lineplot( # plotting
                ax=axes[i, j], # facet location
```

```
                data=df,
                x="charttime",
                y="valuenum",
                hue="itemid",
                palette=chart_palette, # custom palette
                linewidth=3,
                legend=None, # no legend
                marker="o", # dot markers
                markersize=10,
                errorbar=None, # no band
            )
            axes[i, j].grid(True) # grids


    plt.rcParams.update({"font.size": 25}) # font size
    plt.show() # show plot

chartplot(10003400, pq_path, lab_items, chart_items)
chartplot(10001217, pq_path, lab_items, chart_items)
chartplot(10002155, pq_path, lab_items, chart_items)
```

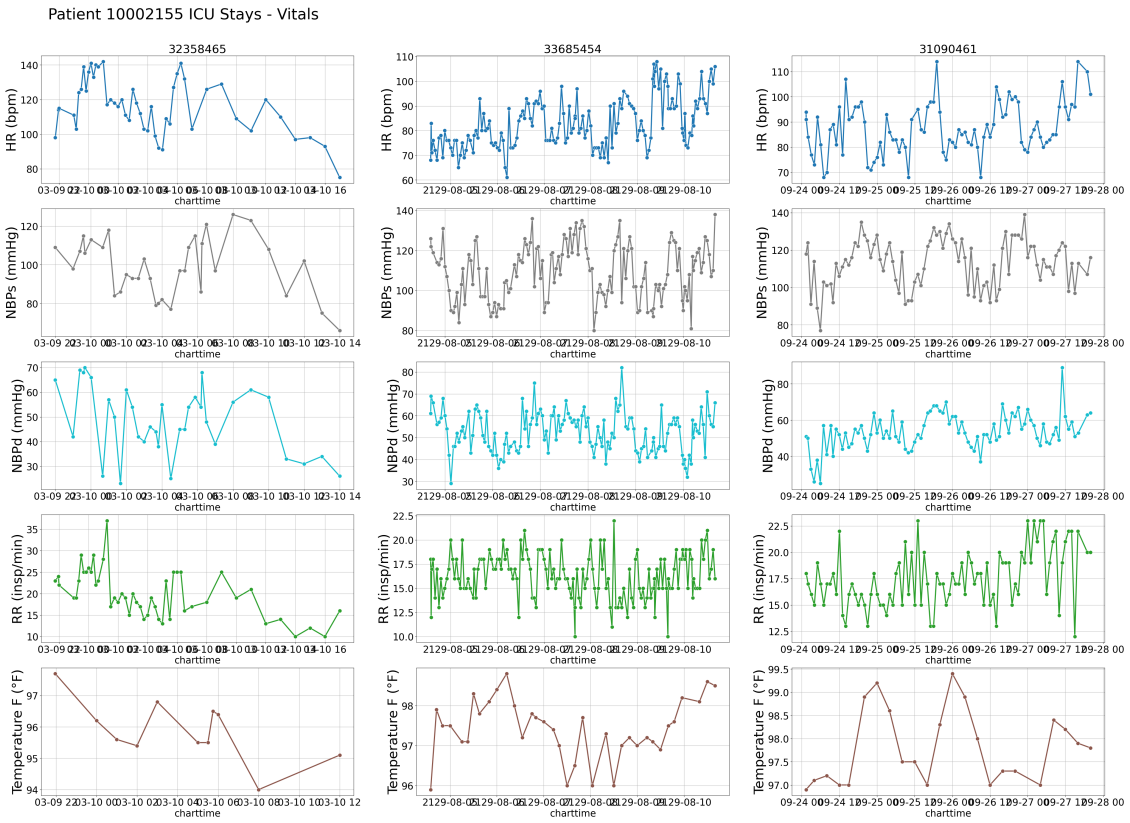
```
Processing Patient and Admissions …
Processing Chartevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_chartevents LazyFrame …
```




```
Processing Patient and Admissions …
Processing Chartevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_chartevents LazyFrame …
```




```
Processing Patient and Admissions …
Processing Chartevents …
Processing ICU Stays …
Collecting plot_patient LazyFrame …
Collecting plot_chartevents LazyFrame …
```



1.2 Problem 2. ICU stays

`icustays.csv.gz` (https://mimic.mit.edu/docs/iv/modules/icu/icustays/) contains data about Intensive Care Units (ICU) stays. The first 10 lines are:

[13]:
```
# !zcat < ./mimic/icu/icustays.csv.gz | head
```

1.2.1 (A). Ingestion

Import `icustays.csv.gz` as a DataFrame `icustays_df`.

[181]:
```
icustays_df = lazy_dict["icustays"].collect()
print("icustays_df Shape: ", icustays_df.shape) # shape of dataframe
icustays_df.to_pandas().head(10) # first rows
```

```
icustays_df Shape:  (73181, 8)
```

[181]:

	subject_id	hadm_id	stay_id	\
0	10000032	29079034	39553978	
1	10000980	26913865	39765666	
2	10001217	24597018	37067082	
3	10001217	27703517	34592300	

```
4    10001725  25563031  31205490
5    10001884  26184834  37510196
6    10002013  23581541  39060235
7    10002155  20345487  32358465
8    10002155  23822395  33685454
9    10002155  28994087  31090461

                                  first_careunit  \
0               Medical Intensive Care Unit (MICU)
1               Medical Intensive Care Unit (MICU)
2              Surgical Intensive Care Unit (SICU)
3              Surgical Intensive Care Unit (SICU)
4  Medical/Surgical Intensive Care Unit (MICU/SICU)
5               Medical Intensive Care Unit (MICU)
6     Cardiac Vascular Intensive Care Unit (CVICU)
7               Medical Intensive Care Unit (MICU)
8                         Coronary Care Unit (CCU)
9  Medical/Surgical Intensive Care Unit (MICU/SICU)

                                   last_careunit              intime  \
0               Medical Intensive Care Unit (MICU) 2180-07-23 14:00:00
1               Medical Intensive Care Unit (MICU) 2189-06-27 08:42:00
2              Surgical Intensive Care Unit (SICU) 2157-11-20 19:18:02
3              Surgical Intensive Care Unit (SICU) 2157-12-19 15:42:24
4  Medical/Surgical Intensive Care Unit (MICU/SICU) 2110-04-11 15:52:22
5               Medical Intensive Care Unit (MICU) 2131-01-11 04:20:05
6     Cardiac Vascular Intensive Care Unit (CVICU) 2160-05-18 10:00:53
7               Medical Intensive Care Unit (MICU) 2131-03-09 21:33:00
8                         Coronary Care Unit (CCU) 2129-08-04 12:45:00
9  Medical/Surgical Intensive Care Unit (MICU/SICU) 2130-09-24 00:50:00

              outtime       los
0 2180-07-23 23:50:47  0.410266
1 2189-06-27 20:38:27  0.497535
2 2157-11-21 22:08:00  1.118032
3 2157-12-20 14:27:41  0.948113
4 2110-04-12 23:59:56  1.338588
5 2131-01-20 08:27:30  9.171817
6 2160-05-19 17:33:33  1.314352
7 2131-03-10 18:09:21  0.858576
8 2129-08-10 17:02:38  6.178912
9 2130-09-27 22:13:41  3.891447
```

1.2.2 (B). Summary and visualization

How many unique `subject_id`? Can a `subject_id` have multiple ICU stays? Summarize the number of ICU stays per `subject_id` by graphs.

- How many unique `subject_id`?
  - There are unique subject ids 50919.
- Can a `subject_id` have multiple ICU stays?
  - yes, a patient can have multiple ICU visits. In fact, 12447 patients had multiple ICU visits
- Summarize the number of ICU stays per `subject_id` by graphs.
  - The number of ICU stays per subjcet ID is greatest at 1 day but decreases significantly as the length of stay increases.

To prepare the dataset the icustays lazy frame was processed by selecting columns, grouping by subject ID and counting the length of each subject grouping. To get the muliple ICU visits per Subject ID, a filter was applied that filtered subjects that only had one subject ID. Matplotlib and a seaborn countplot were used to plot the ICU admissions.

[134]:
```
prob2b = (
    lazy_dict["icustays"] # starting frame
    .select(["subject_id", "stay_id"]) # select col
    .group_by("subject_id", maintain_order=True) # group and keep order
    .len(name="icu_visits") # new col with frequency within group
    .collect() # to dataframe
)

prob2b1 = (
    prob2b # starting frame
    .filter(pl.col('icu_visits').gt(1)) # filter greater than 1
    .with_row_index("id") # create index
    .to_pandas() #to pandas
)

prob2b = (
    prob2b # starting frame
    .with_row_index("id") # create index
    .to_pandas() # to pandas
)

sid_rows = prob2b["id"].iloc[-1] # get total rows
icu_rows_gt1 = prob2b1["id"].iloc[-1] # get total rows
print(f"Number of Unique Subject IDs: {sid_rows}")
print(f"Number of Unique Subject IDs > 1 ICU visit: {icu_rows_gt1}")
```

```
Number of Unique Subject IDs: 50919
Number of Unique Subject IDs > 1 ICU visit: 12447
```

[183]:
```
prob2b.head()
```

[183]:

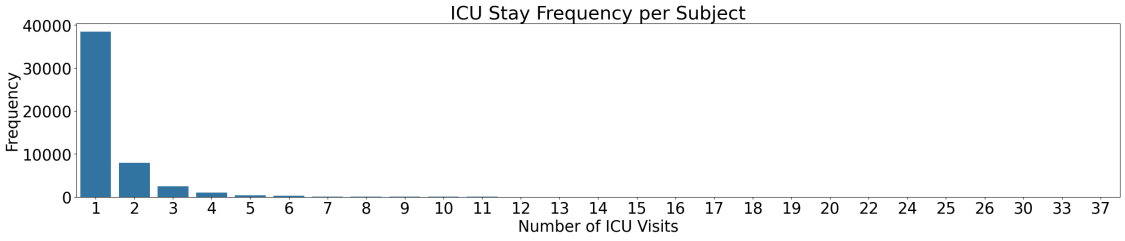
	id	subject_id	icu_visits
0	0	10000032	1
1	1	10000980	1

```
2   2    10001217           2
3   3    10001725           1
4   4    10001884           1
```

[135]:
```
prob2b1.head()
```

[135]:
```
   id  subject_id  icu_visits
0   0    10001217           2
1   1    10002155           3
2   2    10002428           4
3   3    10002930           2
4   4    10003400           3
```

[16]:
```
plt.figure(figsize=(30, 5)) # init figure and size
sns.countplot(data=prob2b,x="icu_visits") # plot
plt.title("ICU Stay Frequency per Subject") # title
plt.ylabel("Frequency") # y title
plt.xlabel("Number of ICU Visits") # x title
plt.show() # show plot
```



1.3 Problem 3. `admissions` data

Information of the patients admitted into hospital is available in `admissions.csv.gz`. See https://mimic.mit.edu/docs/iv/modules/hosp/admissions/ for details of each field in this file. The first 10 lines are

[17]:
```
# !zcat < ./mimic/hosp/admissions.csv.gz | head
```

1.3.1 (A). Ingestion

Import `admissions.csv.gz` as a data frame `admissions_df`.

[185]:
```
admissions_df = lazy_dict["admissions"].collect() # for prob 8 pl.dataframe
admissions_df.to_pandas().head() # first 5 rows
```

[185]:
```
   subject_id   hadm_id           admittime           dischtime deathtime  \
0    10000032  22595853 2180-05-06 22:23:00 2180-05-07 17:15:00       NaT
1    10000032  22841357 2180-06-26 18:27:00 2180-06-27 18:49:00       NaT
```

```
2    10000032  25742920 2180-08-05 23:44:00 2180-08-07 17:50:00       NaT
3    10000032  29079034 2180-07-23 12:35:00 2180-07-25 17:55:00       NaT
4    10000068  25022803 2160-03-03 23:16:00 2160-03-04 06:26:00       NaT

   admission_type admit_provider_id      admission_location  \
0          URGENT            P874LG  TRANSFER FROM HOSPITAL
1        EW EMER.            P09Q6Y          EMERGENCY ROOM
2        EW EMER.            P60CC5          EMERGENCY ROOM
3        EW EMER.            P30KEH          EMERGENCY ROOM
4  EU OBSERVATION            P51VDL          EMERGENCY ROOM

  discharge_location insurance language marital_status   race  \
0               HOME     Other  ENGLISH        WIDOWED  WHITE
1               HOME  Medicaid  ENGLISH        WIDOWED  WHITE
2            HOSPICE  Medicaid  ENGLISH        WIDOWED  WHITE
3               HOME  Medicaid  ENGLISH        WIDOWED  WHITE
4               None     Other  ENGLISH         SINGLE  WHITE

            edregtime           edouttime hospital_expire_flag
0 2180-05-06 19:17:00 2180-05-06 23:30:00                    0
1 2180-06-26 15:54:00 2180-06-26 21:31:00                    0
2 2180-08-05 20:58:00 2180-08-06 01:44:00                    0
3 2180-07-23 05:54:00 2180-07-23 14:00:00                    0
4 2160-03-03 21:55:00 2160-03-04 06:26:00                    0
```

1.3.2 (B). Summary and visualization

Summarize the following information by graphics and explain any patterns you see.

- number of admissions per patient
- admission hour of day (anything unusual?)
- admission minute (anything unusual?)
- length of hospital stay (from admission to discharge) (anything unusual?)

According to the MIMIC-IV documentation:

> All dates in the database have been shifted to protect patient confidentiality. Dates will be internally consistent for the same patient, but randomly distributed in the future. Dates of birth which occur in the present time are not true dates of birth. Furthermore, dates of birth which occur before the year 1900 occur if the patient is older than 89. In these cases, the patient's age at their first admission has been fixed to 300.

Answer

- number of admissions per patient
  - Most patients were only admitted to the hospital for one day and the number of patients decreased as the number of admissions increased.
- admission hour of day (anything unusual?)

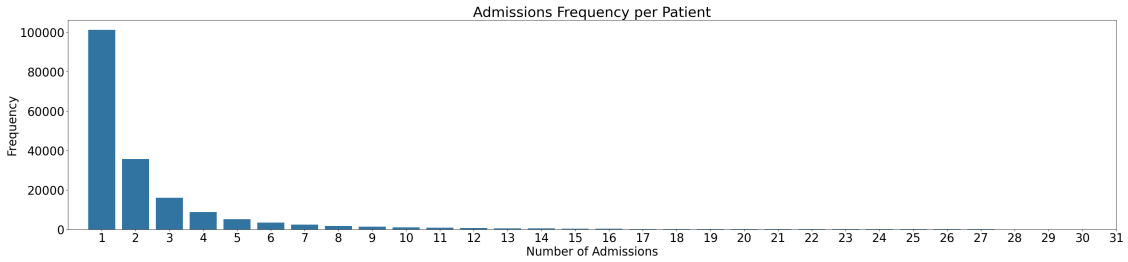
  - Midnight, 7am, and evening hours has the most hospital admissions. This appears to follow when adults are not at work and when they are driving.
- admission minute (anything unusual?)
  - The minutes of the hour that had the most admissions were 0, 15, 30, and 45 minutes of the hour. This could be a sign that there is bais in the dataset from hospital employees to use quarters of the hour.
- length of hospital stay (from admission to discharge) (anything unusual?)
  - Most patients stayed for less than one day and the number of patients decreased the the length of stay increased.

To make the plots, lazyframes from lazy_dict were processed by selecting columns, grouping if necessary and applying operations to create new columns like Number_of_Admissions, and seconds, minutes, hours, and day duration. All of the lazyframes were converted to dataframe then to pandas dataframe before plotting. Plotting was accomplished by using seaborn and matplotlib. Seaborn's countplot was used to show the frequencies.

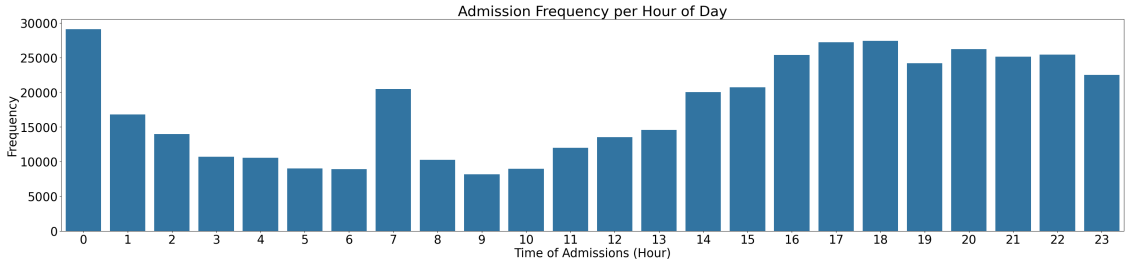
[19]:
```
prob3b1 = (
    lazy_dict["admissions"] # starting frame
    .select(["subject_id", "hadm_id"]) # select col
    .group_by("subject_id", maintain_order=True) # group and keep order
    .len(name="Number_of_Admissions") # number of rows per group
    .collect() # to dataframe
    .to_pandas() # to pandas
)

prob3b23 = (
    lazy_dict["admissions"] # starting frame
    .select("admittime", "dischtime") # select cols
    .with_columns(
        pl.col("admittime").dt.hour().alias("hour"), # hour of day
        pl.col("admittime").dt.minute().alias("minute"), # min of hour
        pl.col("admittime").dt.second().alias("second"), # seconds of min
    )
    .collect() # to dataframe
    .to_pandas() # to pandas
)

prob3b4 = (
    lazy_dict["admissions"] # starting frame
    # select cols
    .select("subject_id", "hadm_id", "admittime", "dischtime")
    .with_columns(
        pl.col("dischtime")
        .sub(pl.col("admittime")) # Duration
        .dt.total_days() # to day format
        .alias("Length_of_Stay") # new col name
    )
    .collect() # to datframe
```

```
    .to_pandas() # to pandas
)
```

[20]:
```
plt.figure(figsize=(40, 8)) # init figure and size
ax = sns.countplot(data=prob3b1, x="Number_of_Admissions")# plot
plt.title("Admissions Frequency per Patient") # title
plt.ylabel("Frequency") # y title
plt.xlabel("Number of Admissions") # x title
ax.set_xlim(-1, 30) # limits
plt.show() # plot figure
```

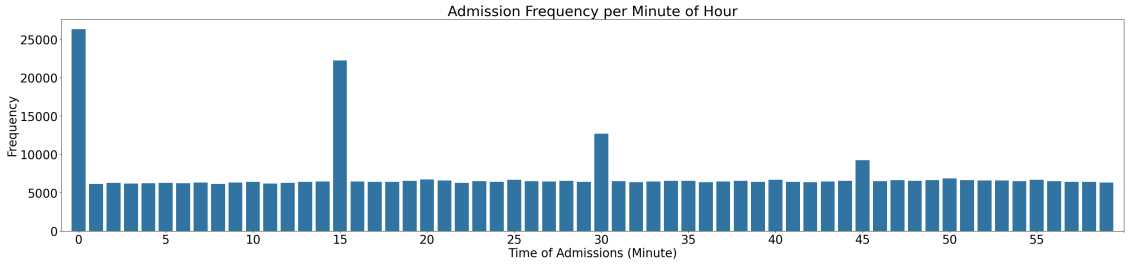


[21]:
```
plt.figure(figsize=(40, 8)) # init figure and size
ax = sns.countplot(data=prob3b23, x="hour")# plot
plt.title("Admission Frequency per Hour of Day") # title
plt.ylabel("Frequency") # y title
plt.xlabel("Time of Admissions (Hour)") # x title
plt.show() # show plot
```

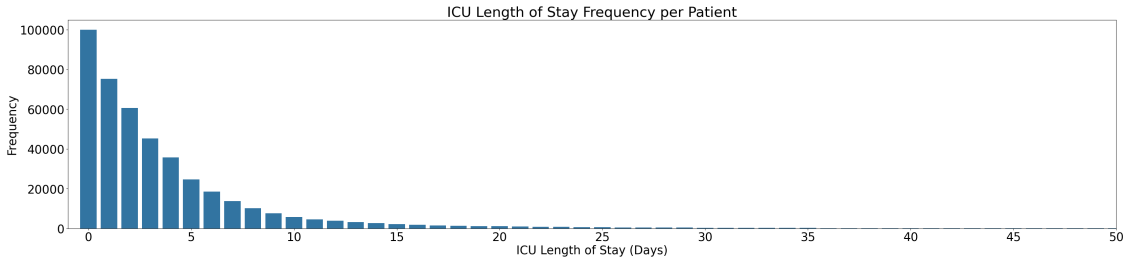


[22]:
```
x_step = 5 # tick step
# max multiple of tick step
x_lim = max(prob3b23["minute"]) + max(prob3b23["minute"]) % x_step
x_ticks = np.arange(0, x_lim, x_step) # tick array

plt.figure(figsize=(40, 8)) # init figure and size
ax = sns.countplot(data=prob3b23, x="minute") # plot
plt.title("Admission Frequency per Minute of Hour") # title
```

```
plt.ylabel("Frequency") # y title
plt.xlabel("Time of Admissions (Minute)") # x title
ax.xaxis.set_ticks(x_ticks) # custom tick
ax.set_xlim(-1, 60) # limits
plt.show() # show plot
```



[23]:
```
x_step = 5 # tick step
# max muliple of tick step
x_lim = max(prob3b4["Length_of_Stay"]) + max(prob3b4["Length_of_Stay"]) % x_step
x_ticks = np.arange(0, x_lim, x_step) # create a tick array

plt.figure(figsize=(40, 8)) # init figure and size
ax = sns.countplot(data=prob3b4, x="Length_of_Stay") # plot
plt.title("ICU Length of Stay Frequency per Patient") # title
plt.ylabel("Frequency") # y title
plt.xlabel("ICU Length of Stay (Days)") # x title
ax.xaxis.set_ticks(x_ticks) # custom tick
ax.set_xlim(-1, 50) # limits
plt.show() # show plot
```



1.4 Problem 4. `patients` data

Patient information is available in `patients.csv.gz`. See https://mimic.mit.edu/docs/iv/modules/hosp/patients/ for details of each field in this file. The first 10 lines are:

[24]:
```
# !zcat < ./mimic/hosp/patients.csv.gz | head
```

```
/bin/bash: line 1: /home/wtmartinez/mimic/hosp/patients.csv.gz: No such file or
directory
```

1.4.1 (A). Ingestion

Import `patients.csv.gz` (https://mimic.mit.edu/docs/iv/modules/hosp/patients/) as a data frame `patients_df`.

[123]:
```
patients_df = lazy_dict["patients"].collect() # collect for prob 8
patients_df.to_pandas().head(10)
```

[123]:
```
shape: (10, 6)
```

subject_id	gender	anchor_age	anchor_year	anchor_year_group	dod
i64	str	i64	i64	str	date
10000032	F	52	2180	2014 - 2016	2180-09-09
10000048	F	23	2126	2008 - 2010	null
10000068	F	19	2160	2008 - 2010	null
10000084	M	72	2160	2017 - 2019	2161-02-13
10000102	F	27	2136	2008 - 2010	null
10000108	M	25	2163	2014 - 2016	null
10000115	M	24	2154	2017 - 2019	null
10000117	F	48	2174	2008 - 2010	null
10000178	F	59	2157	2017 - 2019	null
10000248	M	34	2192	2014 - 2016	null

1.4.2 (B). Summary and visaulization

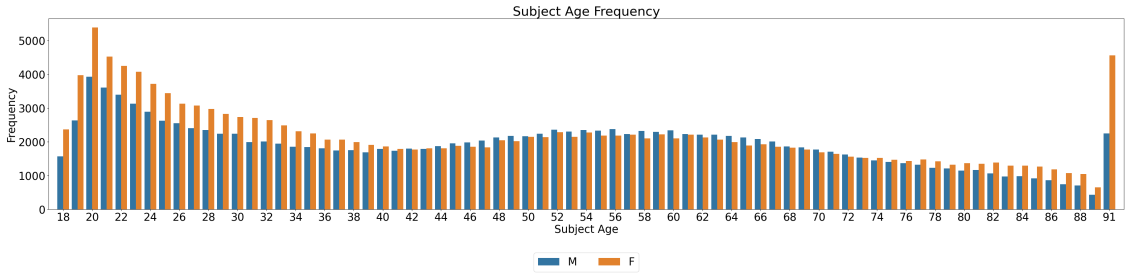
Summarize variables `gender` and `anchor_age` by graphics, and explain any patterns you see.

Answer Convert lazyframe to pandas Dataframe and select columns, and plot using seaborn and matplotlib using a seaborn countplot.

From the plot, more young females (>40) go to the hospital than young men. In addition, there are more females that are 91 and over that go to the hostpital than men.

[26]:
```
prob4b = (
    lazy_dict["patients"] # starting lazyframe
    .select(["gender", "anchor_age"]) # select col
    .collect() # to dataframe
    .to_pandas() # to pandas dataframe
)
```

[27]:
```
x_step = 2 # tick step
# max multiple of step
x_lim2 = max(prob4b["anchor_age"]) + max(prob4b["anchor_age"]) % x_step
# array of tick steps
x_ticks = np.arange(0, x_lim2, x_step)

plt.figure(figsize=(45, 8)) # inti figure and size
ax = sns.countplot(data=prob4b, x="anchor_age", hue="gender") # plot
plt.title("Subject Age Frequency") # title
plt.ylabel("Frequency") # y title
plt.xlabel("Subject Age") # x title
plt.legend(loc="upper center", bbox_to_anchor=(0.5, -0.2), ncol=2)
ax.xaxis.set_ticks(x_ticks) # custom tick
ax.set_xlim(-1, 73) # limits
plt.show() # show plot
```



1.5 Problem 5. Lab results

labevents.csv.gz (https://mimic.mit.edu/docs/iv/modules/hosp/labevents/) contains all laboratory measurements for patients. The first 10 lines are

[28]:
```
#!zcat < ./mimic/hosp/labevents.csv.gz | head
```

`d_labitems.csv.gz` (https://mimic.mit.edu/docs/iv/modules/hosp/d_labitems/) is the dictionary of lab measurements.

[29]:
```
#!zcat < ./mimic/hosp/d_labitems.csv.gz | head
```

We are interested in the lab measurements of creatinine (50912), potassium (50971), sodium (50983), chloride (50902), bicarbonate (50882), hematocrit (51221), white blood cell count (51301), and glucose (50931). Retrieve a subset of `labevents.csv.gz` that only containing these items for the patients in `icustays_df`. Further restrict to the last available measurement (by `storetime`) before the ICU stay. The final `labevents_df` should have one row per ICU stay and columns for each lab measurement. (ten columns with column names `subject_id`, `stay_id`, `Bicarbonate`, `Chloride`, …)

Hint: Use the Parquet format you generated in Homework 3. For reproducibility, make `labevents.parquet` file available at the current working directory, for example, by a symbolic link.

Answer 5 `labevents_df` was created using the code below. labevents has the vital measurement values. labevents.parquet was ingested by using the `make_lazy_dict` function to read in `labevents.parquet` as a lazyframe using polars. labevents was already filted by the itemids listed above in `make_lazy_dict` and column types were assigned as well. The dataset was saved to a lazy dictionary called `lazy_dict`. However, because the dataset is a lazy frame, the initial shape of labevents could not be calculated. `labevents_df` was created by starting with the labevents lazyframe, joining with icu_stays and item keys, and filtering for the last lab measurment for each icu stay. The label for each item was concatenated with the unites of that item and added to a new column called Item_Info. This column will be useful in the plotting of the dataframe.

Sort each stay id with by storetime. Store time was used because store time is the time the measurment was available for interpretation by doctors. After sorting, the dataframe was grouped by IDs and itemids. The dataframe was filted for the last storetime of each item id. The data was pivoted to wide data format where the groupings were subject id and ICU stay ID and the item name and units were widened where the item values were the values of the widen dataframe.

[30]:
```
# list comprehension for reformating lazyframe schema
lab_col = [
    f"{k}:{' '*(20-len(k))}{v}"
    for k, v in lazy_dict["labevents"].schema.items()
]
# Reshape into col
lab_col = np.array(lab_col).reshape(-1, 1)
# list comprehension for reformating lazyframe schema
icu_col = [f"{k}:{' '*(20-len(k))}{v}" #
           for k, v in lazy_dict["icustays"].schema.items()]
icu_col = np.array(icu_col).reshape(-1, 1) # Reshape into col
```

[31]:
```
print(f"labevents shape and columns:\n{lab_col}")
```

```
labevents shape and columns:
[['labevent_id:         Int64']
 ['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['specimen_id:         Int64']
```

```
 ['itemid:              Int64']
 ['order_provider_id:   String']
 ["charttime:           Datetime(time_unit='us', time_zone=None)"]
 ["storetime:           Datetime(time_unit='us', time_zone=None)"]
 ['value:               String']
 ['valuenum:            Float64']
 ['valueuom:            String']
 ['ref_range_lower:     Float64']
 ['ref_range_upper:     Float64']
 ['flag:                String']
 ['priority:            String']
 ['comments:            String']]
```

[32]:
```
print(f"icustay shape and columns:\n{icu_col}")
```

```
icustay shape and columns:
[['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['stay_id:             Int64']
 ['first_careunit:      String']
 ['last_careunit:       String']
 ["intime:              Datetime(time_unit='us', time_zone=None)"]
 ["outtime:             Datetime(time_unit='us', time_zone=None)"]
 ['los:                 Float64']]
```

[114]:
```
prob5 = (
    lazy_dict["labevents"] # starting lazyframe
    # left join icustays and lab item keys
    .join(lazy_dict["icustays"], on=["subject_id", "hadm_id"], how="left")
    .join(lazy_dict["d_labitems"], on=["itemid"], how="left")
    # sort by IDs and storetime in ascending order
    .sort(
        ["subject_id", "hadm_id", "stay_id", "itemid", "storetime"],
        descending=False)
    # group by IDs and itemids
    .group_by(
        ["subject_id", "hadm_id", "stay_id", "itemid"],
        maintain_order=True)
    # take last measurement by storetime
    .last()
)

prob5_lab = (
    prob5 # starting lazyframe
    .drop_nulls(["subject_id", "stay_id", "itemid"]) # drop nulls
    # concatenate strings from label and valueuom into new col
    .with_columns(
```

```
        pl.concat_str(
            pl.col('label'),
            pl.lit(" ("),
            pl.col('valueuom'),
            pl.lit(")")
        ).alias('Item_Info') # new col name
    )
)
```

[115]:
```
labevents_df_hadm = (
    prob5_lab
    .collect() # collect lazyframe
    .filter(pl.col("itemid").is_not_null()) # drop null in col
    .pivot(
        values="valuenum", # item values as values for Item_Info
        index=["subject_id", "hadm_id", "stay_id"], # group
        columns="Item_Info", # widen Item_Info
        maintain_order=True, # keep order
    )
)
labevents_df = labevents_df_hadm.drop("hadm_id") # drop col
```

[116]:
```
print(f"labevents_df Shape: {labevents_df.shape}")
```

```
labevents_df Shape: (72436, 10)
```

[186]:
```
labevents_df.to_pandas().head(10) # first
```

[186]:

	subject_id	stay_id	Bicarbonate (mEq/L)	Chloride (mEq/L)	\
0	10000032	39553978	27.0	97.0	
1	10000980	39765666	24.0	108.0	
2	10001217	37067082	30.0	102.0	
3	10001217	34592300	27.0	103.0	
4	10001725	31205490	32.0	100.0	
5	10001884	37510196	37.0	97.0	
6	10002013	39060235	29.0	97.0	
7	10002155	32358465	25.0	88.0	
8	10002155	33685454	28.0	98.0	
9	10002155	31090461	25.0	100.0	

	Creatinine (mg/dL)	Glucose (mg/dL)	Potassium (mEq/L)	Sodium (mEq/L)	\
0	0.4	121.0	5.2	130.0	
1	2.2	121.0	4.5	142.0	
2	0.4	95.0	4.2	140.0	
3	0.4	86.0	4.3	141.0	
4	0.9	124.0	3.7	140.0	
5	0.6	94.0	4.2	138.0	

6	1.1	161.0	4.1	137.0
7	1.5	192.0	5.7	125.0
8	1.0	140.0	4.9	133.0
9	2.4	97.0	4.4	136.0

	Hematocrit (%)	White Blood Cells (K/uL)
0	32.1	4.8
1	23.6	4.8
2	38.0	10.3
3	38.5	8.8
4	34.0	11.1
5	22.9	13.4
6	30.5	11.7
7	25.4	7.1
8	29.3	7.5
9	27.5	6.5

1.6 Problem 6. Vitals from charted events

`chartevents.csv.gz` (https://mimic.mit.edu/docs/iv/modules/icu/chartevents/) contains all the charted data available for a patient. During their ICU stay, the primary repository of a patient's information is their electronic chart. The `itemid` variable indicates a single measurement type in the database. The `value` variable is the value measured for `itemid`. The first 10 lines of `chartevents.csv.gz` are

[121]:
```
# !zcat < ./mimic/icu/chartevents.csv.gz | head
```

`d_items.csv.gz` (https://mimic.mit.edu/docs/iv/modules/icu/d_items/) is the dictionary for the itemid in `chartevents.csv.gz`.

[120]:
```
# !zcat < ./mimic/icu/d_items.csv.gz | head
```

We are interested in the vitals for ICU patients: heart rate (220045), systolic non-invasive blood pressure (220179), diastolic non-invasive blood pressure (220180), body temperature in Fahrenheit (223761), and respiratory rate (220210). Retrieve a subset of `chartevents.csv.gz` only containing these items for the patients in `icustays_tble`. Further restrict to the first vital measurement within the ICU stay. The final `chartevents_tble` should have one row per ICU stay and columns for each vital measurement.

Hint: Use the Parquet format you generated in Homework 3. For reproducibility, make `chartevents.parquet` file available at the current working directory, for example, by a symbolic link.

1.6.1 Answer 6

`chartevents_df` was created using the code below. Chartevents has the vital measurement values. chartevents.parquet was ingested by using the `make_lazy_dict` function to read in `chartevents.parquet` as a lazyframe using polars. Chartevents was already filted by the itemids listed above in `make_lazy_dict` and column types were assigned as well. The dataset was saved

to a lazy dictionary called `lazy_dict`. However, because the dataset is a lazy frame, the initial shape of chartevents could not be calculated. `chartevents_df` was created by starting with the chartevents lazyframe, joining with icu_stays and item keys, and filtering for the first chart vital measurment for each icu stay. The label for each item was concatenated with the unites of that item and added to a new column called Item_Info. This column will be useful in the plotting of the dataframe.

Sort each stay id with by storetime. Store time was used because store time is the time the measurment was available for interpretation by doctors. After sorting, the dataframe was grouped by IDs and itemids. The dataframe was filted for the first storetime of each item id. The data was pivoted to wide data format where the groupings were subject id and ICU stay ID and the item name and units were widened where the item values were the values of the widen dataframe.

[38]:
```
# list comprehension for formating dataframe shema
chart_col = [f"{k}:{' '*(20-len(k))}{v}"
             for k, v in lazy_dict["chartevents"].schema.items()]
chart_col = np.array(lab_col).reshape(-1, 1) # Reshap into a vertical col
```

[39]:
```
print(f"chartevents columns:\n{chart_col}") # col name and type
```

```
chartevents shape and columns:
[['labevent_id:         Int64']
 ['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['specimen_id:         Int64']
 ['itemid:              Int64']
 ['order_provider_id:   String']
 ["charttime:           Datetime(time_unit='us', time_zone=None)"]
 ["storetime:           Datetime(time_unit='us', time_zone=None)"]
 ['value:               String']
 ['valuenum:            Float64']
 ['valueuom:            String']
 ['ref_range_lower:     Float64']
 ['ref_range_upper:     Float64']
 ['flag:                String']
 ['priority:            String']
 ['comments:            String']]
```

[40]:
```
print(f"icustay shape and columns:\n{icu_col}") # col name and type
```

```
icustay shape and columns:
[['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['stay_id:             Int64']
 ['first_careunit:      String']
 ['last_careunit:       String']
 ["intime:              Datetime(time_unit='us', time_zone=None)"]
 ["outtime:             Datetime(time_unit='us', time_zone=None)"]
 ['los:                 Float64']]
```

[118]:
```
prob6 = (
    lazy_dict["chartevents"] # start with chartevents
    # left join icu info and item labels
    .join(lazy_dict["icustays"], on=["subject_id", "hadm_id"], how="left")
    .join(lazy_dict["d_items"], on=['itemid'], how='left')
    # sort before taking first.
    .sort(
        ["subject_id", "hadm_id", "stay_id", "itemid", "storetime"], # group
        descending=False) # Ascending order
    .group_by( # group then then take first value. Keep order for comparison
        ["subject_id", "hadm_id", "stay_id", "itemid"], #group
        maintain_order=True) # keep order
    .first() # take first chart value
    .drop('label') # remove item full name
    .rename({"abbreviation": "label"}) # use abbreviation
)

prob6_chart = (
    prob6
    .drop_nulls(["subject_id", "stay_id", "itemid"])
    # create a new column with item name and units
    .with_columns(
        pl.concat_str( # combine strings
            pl.col('label'), # add item name
            pl.lit(" ("), # add string
            pl.col('valueuom'), # add item units
            pl.lit(")") # add string
        ).alias('Item_Info') #  column
    )
)

chartevents_df_hadm = (
    prob6_chart.collect() # lazyframe to dataframe
    .filter(pl.col("itemid").is_not_null()) # remove null itemids
    # pivot wider
    .pivot(
        values="valuenum", # values for item info
        index=["subject_id", "hadm_id", "stay_id"], # group by
        columns="Item_Info", # widen by Item name and units
        maintain_order=True, # keep order
    )
)

chartevents_df = chartevents_df_hadm.drop("hadm_id") # drop col
```

[187]:
```
print(f"chartevents_df Shape: {chartevents_df.shape}") # shape
chartevents_df.to_pandas().head(10) # first 10 rows
```

```
chartevents_df Shape: (73164, 7)
```

[187]:

	subject_id	stay_id	HR (bpm)	NBPs (mmHg)	NBPd (mmHg)	RR (insp/min)	Temperature F (°F)
0	10000032	39553978	91.0	84.0	48.0	24.0	98.7
1	10000980	39765666	75.0	150.0	77.0	24.0	98.0
2	10001217	37067082	86.0	151.0	90.0	18.0	98.5
3	10001217	34592300	96.0	156.0	95.0	17.0	97.6
4	10001725	31205490	86.0	73.0	56.0	19.0	97.7
5	10001884	37510196	38.0	167.0	12.0	16.0	98.1
6	10002013	39060235	80.0	104.0	70.0	14.0	97.2
7	10002155	32358465	114.0	109.0	65.0	24.0	97.7
8	10002155	33685454	68.0	126.0	61.0	18.0	95.9
9	10002155	31090461	91.0	118.0	51.0	18.0	96.9

1.7 Problem 7. Putting things together

Let us create a data frame `mimic_icu_cohort` for all ICU stays, where rows are all ICU stays of adults (age at `intime` >= 18) and columns contain at least following variables

- all variables in `icustays_tble`
- all variables in `admissions_tble`
- all variables in `patients_tble`
- the last lab measurements before the ICU stay in `labevents_tble`
- the first vital measurements during the ICU stay in `chartevents_tble`
- The final `mimic_icu_cohort` should have one row per ICU stay and columns for each variable.

Answer The code below is for creating the `mimic_icu_cohort` dataframe. `mimic_icu_cohort` summarizes patients that were in the ICU and takes the last lab measurements for going to the icu and first chart vital in the icu. To accompany this information, the admissions, patients, and icustays dataframes were added to the `mimic_icu_cohort` dataframe. After adding those dataframes the final `mimic_icu_cohort` contained patient demographics, hospital information, and ICU length of stay.

[64]:
```
# list the columns and the column types of datasets in lf_list
lf_list = ["admissions", "patients"] # list of lazyframes
for lf in lf_list:
```

```
    # list comprehension to turn schema into a more readable text format
    cols = [f"{k}:{' '*(20-len(k))}{v}"
            for k, v in lazy_dict[lf].schema.items()]
    # Reshape list comprehension into a column
    cols = np.array(lab_col).reshape(-1, 1)
    # Print each list comprehension
    print(f"{lf} shape and columns:\n{cols}\n")
```

```
admissions shape and columns:
[['labevent_id:         Int64']
 ['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['specimen_id:         Int64']
 ['itemid:              Int64']
 ['order_provider_id:   String']
 ["charttime:           Datetime(time_unit='us', time_zone=None)"]
 ["storetime:           Datetime(time_unit='us', time_zone=None)"]
 ['value:               String']
 ['valuenum:            Float64']
 ['valueuom:            String']
 ['ref_range_lower:     Float64']
 ['ref_range_upper:     Float64']
 ['flag:                String']
 ['priority:            String']
 ['comments:            String']]

patients shape and columns:
[['labevent_id:         Int64']
 ['subject_id:          Int64']
 ['hadm_id:             Int64']
 ['specimen_id:         Int64']
 ['itemid:              Int64']
 ['order_provider_id:   String']
 ["charttime:           Datetime(time_unit='us', time_zone=None)"]
 ["storetime:           Datetime(time_unit='us', time_zone=None)"]
 ['value:               String']
 ['valuenum:            Float64']
 ['valueuom:            String']
 ['ref_range_lower:     Float64']
 ['ref_range_upper:     Float64']
 ['flag:                String']
 ['priority:            String']
 ['comments:            String']]
```

[45]:
```
# make mimic_icu_cohort dataframe in polars
mimic_icu_cohort = (
```

```
    # start with chart events
    chartevents_df_hadm
    # left join lab events
    .join(
        labevents_df_hadm,
        on=["subject_id", "hadm_id", "stay_id"],
        how="left"
    )
    # left join admissions
    .join(admissions_df, on=["subject_id", "hadm_id"], how="left")
    # left join patients
    .join(patients_df, on=["subject_id"], how="left")
    # left join icu stays
    .join(icustays_df, on=["subject_id", "hadm_id", "stay_id"], how="left")
    # filter by age greater than or equal to 18
    .filter(pl.col("anchor_age").ge(18))
)
```

[188]:
```
print('mimic_icu_cohort shape:', mimic_icu_cohort.shape) # dataframe shape
mimic_icu_cohort.to_pandas().head(10) # first 10 results of mimic_icu_cohort
```

```
mimic_icu_cohort shape: (73164, 40)
```

[188]:

	subject_id	hadm_id	stay_id	NBPs (mmHg)	NBPd (mmHg)	HR (bpm)	\
0	10000032	29079034	39553978	82.0	59.0	94.0	
1	10000980	26913865	39765666	150.0	77.0	77.0	
2	10001217	24597018	37067082	145.0	84.0	101.0	
3	10001217	27703517	34592300	121.0	72.0	79.0	
4	10001725	25563031	31205490	114.0	65.0	82.0	
5	10001884	26184834	37510196	122.0	67.0	69.0	
6	10002013	23581541	39060235	104.0	70.0	103.0	
7	10002155	20345487	32358465	109.0	65.0	98.0	
8	10002155	23822395	33685454	118.0	56.0	75.0	
9	10002155	28994087	31090461	103.0	57.0	70.0	

	RR (insp/min)	Temperature F (°F)	Bicarbonate (mEq/L)	Chloride (mEq/L)	\
0	20.0	99.5	27.0	97.0	
1	23.0	98.0	24.0	108.0	
2	25.0	100.8	30.0	102.0	
3	18.0	98.2	27.0	103.0	
4	22.0	98.4	32.0	100.0	
5	17.0	98.4	37.0	97.0	
6	18.0	97.2	29.0	97.0	
7	23.0	97.7	25.0	88.0	
8	16.0	96.0	28.0	98.0	
9	17.0	97.2	25.0	100.0	

```
   …  gender  anchor_age  anchor_year  anchor_year_group        dod  \
0  …       F          52         2180        2014 - 2016 2180-09-09
1  …       F          73         2186        2008 - 2010 2193-08-26
2  …       F          55         2157        2011 - 2013        NaT
3  …       F          55         2157        2011 - 2013        NaT
4  …       F          46         2110        2011 - 2013        NaT
5  …       F          68         2122        2008 - 2010 2131-01-20
6  …       F          53         2156        2008 - 2010        NaT
7  …       F          80         2128        2008 - 2010 2131-03-10
8  …       F          80         2128        2008 - 2010 2131-03-10
9  …       F          80         2128        2008 - 2010 2131-03-10

                                  first_careunit  \
0               Medical Intensive Care Unit (MICU)
1               Medical Intensive Care Unit (MICU)
2             Surgical Intensive Care Unit (SICU)
3             Surgical Intensive Care Unit (SICU)
4  Medical/Surgical Intensive Care Unit (MICU/SICU)
5               Medical Intensive Care Unit (MICU)
6     Cardiac Vascular Intensive Care Unit (CVICU)
7               Medical Intensive Care Unit (MICU)
8                         Coronary Care Unit (CCU)
9  Medical/Surgical Intensive Care Unit (MICU/SICU)

                                   last_careunit              intime  \
0               Medical Intensive Care Unit (MICU) 2180-07-23 14:00:00
1               Medical Intensive Care Unit (MICU) 2189-06-27 08:42:00
2             Surgical Intensive Care Unit (SICU) 2157-11-20 19:18:02
3             Surgical Intensive Care Unit (SICU) 2157-12-19 15:42:24
4  Medical/Surgical Intensive Care Unit (MICU/SICU) 2110-04-11 15:52:22
5               Medical Intensive Care Unit (MICU) 2131-01-11 04:20:05
6     Cardiac Vascular Intensive Care Unit (CVICU) 2160-05-18 10:00:53
7               Medical Intensive Care Unit (MICU) 2131-03-09 21:33:00
8                         Coronary Care Unit (CCU) 2129-08-04 12:45:00
9  Medical/Surgical Intensive Care Unit (MICU/SICU) 2130-09-24 00:50:00

              outtime       los
0 2180-07-23 23:50:47  0.410266
1 2189-06-27 20:38:27  0.497535
2 2157-11-21 22:08:00  1.118032
3 2157-12-20 14:27:41  0.948113
4 2110-04-12 23:59:56  1.338588
5 2131-01-20 08:27:30  9.171817
6 2160-05-19 17:33:33  1.314352
7 2131-03-10 18:09:21  0.858576
8 2129-08-10 17:02:38  6.178912
9 2130-09-27 22:13:41  3.891447
```

```
[10 rows x 40 columns]
```

1.8 Problem 8. Exploratory data analysis (EDA)

Summarize the following information about the ICU stay cohort `mimic_icu_cohort` using appropriate method:

- Length of ICU stay `los` vs demographic variables (race, insurance, marital_status, gender, age at intime)
- Length of ICU stay `los` vs the last available lab measurements before ICU stay
- Length of ICU stay `los` vs the first vital measurements within the ICU stay
- Length of ICU stay `los` vs first ICU unit

At least two plots should be created, with at least one them including multiple facets using an appropriate keyword argument.

1.8.1 ANSWER 8

- `los` is the ICU length of stay. The dataframe, `mimic_icu_cohort`, is a list of patients that attended the ICU. The graphs were created to service as early data analysis. The relationships between `los` and demographics, lab and vital measurements, and first ICU care unit were ploted using an apropriate plotting method.
- A column was added to the `mimic_icu_cohort` called `age_group` which aggregates patient ages into groups of 10 years.
- Before plotting, 4 additional dataframes were created that represented lab and vital measurements in a long and wid data formats. Representing lab and vital measurements in a long data format allowed for easy graphing of all measurements in a plot.

1. Demographics with Respect to ICU length of Stay
   - Facet kernel density estimate (kde) plots of `marital_status`, `gender`, `language`, and `insurance`. Form the marital status plot, the median divorced patients spend more time in the ICU and have a larger standard deviation than married and single patients. A cofounding variable that could be a factor in longer ICU stays as a divorced patient is that divorces usually occur later in life which also is correlated to a longer ICU stay. In addition, non-english speaking patients stayed longer and had a larger standard deviation than english speaking patients. The variables can be improved by adding an "other" label to "?" within race.
   - Facet box plots of `race` and `age_group`. `race` and `age_group` were not represented as kde plots because those variables had too many subgroupings and cluttered the kde plot to the point it was unreadable. All of the lines were overlapping in the kde plot. Instead, box plot was used to represent the relationship because each box has an independent axis. In the future, language should be summarized in a similar way as age because there are too many races to plot and many of them are subgroupings of a general race. Some key takeaways from plotting the age is that younger patients spend less time in the ICU than older patients and the median ICU length of Stay is highest for patients between 40 to 80 years old.

2. Lab Measurements with Respect to ICU length of Stay
    - A similar dataset to the `mimic_icu_cohort` dataframe was used show all of the last lab measurements before an ICU stay. The difference between mimic_icu_cohort and the the prob8_lab is that the itemids and values are in long data format instead of a wide data format. The numbers are the same but this allowed for easy plotting of all lab measurements in a single plot. A kde plot was used to show the relationships with respect to ICU length of stay; however there were too many measurments with different ranges. This made it harder to read/understand plots that had less variance or plots that had a smaller range. Therefore, it is better to show the plots in facets.
    - Insead of plotting all of the lab measurements in one plot. A faceted plot was used to show the relationships between each lab measurement and ICU length of stay. KDE plots were used instead of scatter plots becuase the scatter plots did not summarize the information well. KDE plots show where the high density regions are within the data. For example, patients with high bicarbonate stayed in the ICU for longer than patients with low bicarbonate. Patients with low hemacrit spend more time in ICU.
3. Vital Measurements with Respect to ICU length of Stay
    - All of the vital measurments were plotted in a single kde plot. Similar to the lab measurments being plotted in the same plot, the plot for all vital measurments was too crowded and hard to read.
    - Insead of plotting all of the vital measurements in one plot. A faceted plot was used to show the relationships between each vital measurement and ICU length of stay. KDE plots were used instead of scatter plots becuase the scatter plots did not summarize the information well. KDE plots show where the high density regions are within the data. For the vital measurements, it was hard to determine what caused longer ICU stays. The biggest takeaway is that there was a larger standard diviation within the first few days of an ICU Stay, but that makes sense because the number of people decreased as the length of stay increased.
4. ICU length of Stay First Care Unit
    - A single KDE plot was used to show the the length of stay by first care unit. The number of patients had a common normalization meaning the area under all of the units sum to 1. The plot is useful in showing the relative size and percentage a first care unit was. The MICU was the most frequent first care unit. However, it is difficult to see the distribution of less common care units.
    - A facetted plot was used to show both normalization techniques. The first plot had common normalization and the second plot had each unit be normalized. The addition of the plot with each unit being nomalized allows the viewer to see the units that are less frequent to be more visable. From this plot, the neural surgical care unit had the longest median icu length of stay. This was not visible from the previous plot because it was one of the least frequent first care units.

Data Preparation

[48]:
```
prob8 = (
    mimic_icu_cohort
    # Add column age_group that summarizes age into 10 year groups.
    .with_columns(
        # age less than or equal 20
        pl.when(pl.col('anchor_age').le(20))
```

```
        # assign 18-20
        .then(pl.lit("18 - 20"))
        # age greater than or equal to 21 and less than or equal to 30
        .when(pl.col('anchor_age').ge(21) & pl.col('anchor_age').le(30))
        # assign 21-30. Same Schema below but different age groups
        .then(pl.lit("21 - 30"))
        .when(pl.col('anchor_age').ge(31) & pl.col('anchor_age').le(40))
        .then(pl.lit("31 - 40"))
        .when(pl.col('anchor_age').ge(41) & pl.col('anchor_age').le(50))
        .then(pl.lit("41 - 50"))
        .when(pl.col('anchor_age').ge(51) & pl.col('anchor_age').le(60))
        .then(pl.lit("51 - 60"))
        .when(pl.col('anchor_age').ge(61) & pl.col('anchor_age').le(70))
        .then(pl.lit("61 - 70"))
        .when(pl.col('anchor_age').ge(71) & pl.col('anchor_age').le(80))
        .then(pl.lit("71 - 80"))
        .when(pl.col('anchor_age').ge(81) & pl.col('anchor_age').le(90))
        .then(pl.lit("81 - 90"))
        # age less than or equal 91
        .when(pl.col('anchor_age').ge(91))
        # assigned 91 and over because MIMIC summarized ages 89 and over to 91.
        # see docs
        .then(pl.lit("91 and Over"))
        # if not in age group assign missing
        .otherwise(pl.lit(None))
        # make new column to store assignments.
        .alias("age_group")
    )
    # sort by age_group to make graph look better
    .sort("age_group", descending=True)
    .to_pandas()
)

# Make a long labevents dataframe with same item values as mimic_icu_cohort
# for plotting all labevents in one plot
prob8_lab_l = (
    prob5_lab # see problem 5
    .join(
        lazy_dict["icustays"], # left join icustays for the los variable
        on=["subject_id", "hadm_id", "stay_id"],
        how="left")
    .collect()
)

# Make the lab values wide for facet plotting
prob8_lab_w = (
    prob8_lab_l
```

```
    .pivot(
        values="valuenum", # values for Item_Info
        index=["subject_id", "hadm_id", "stay_id"], # groupings
        columns="Item_Info", # widen Item_Info
        maintain_order=True, # maintain for visual comparison
    )
    .join(
        mimic_icu_cohort.select('stay_id','los'), # select columns
        on = 'stay_id', # join on stay id
        how = 'left' # left join
    )
)

prob8_lab_l = prob8_lab_l.to_pandas() # convert to pandas
prob8_lab_w = prob8_lab_w.to_pandas() # convert to pandas

# Make a long chartevents dataframe with same item values as mimic_icu_cohort
# for plotting all chartevents in one plot
prob8_chart_l = (
    prob6_chart # see problem 6
    .join(
        lazy_dict["icustays"], # left join icustays for los variable
        on=["subject_id", "hadm_id", "stay_id"],
        how="left")
    .collect()
)


# Make the chart values wide for facet plotting
# see comments for prob8_lab_w for description.
prob8_chart_w = (
    prob8_chart_l
    .pivot(
        values="valuenum",
        index=["subject_id", "hadm_id", "stay_id"],
        columns="Item_Info",
        maintain_order=True,
    )
    .join(
        mimic_icu_cohort.select('stay_id','los'),
        on = 'stay_id',
        how = 'left'
    )
)

prob8_chart_l = prob8_chart_l.to_pandas() # convert to pandas
prob8_chart_w = prob8_chart_w.to_pandas() # convert to pandas
```

```
# los Tick Spacing. Placed here because it is needed for all plots.
x_step = 5  # tick steps
x_lim = max(prob8["los"]) + max(prob8["los"]) % x_step  # multiple of tick step
x_ticks = np.arange(0, x_lim, x_step)  # list of steps
```

[191]:
```
print("prob8 shape:", prob8.shape) # print name and shape
prob8.head(2) # show first 2 results to save space. schema continues below
```

```
prob8 shape: (73164, 41)
```

[191]:
```
   subject_id   hadm_id   stay_id  NBPs (mmHg)  NBPd (mmHg)  HR (bpm)  \
0    10012853  27882036  31338022        107.0         53.0     106.0
1    10018845  21101111  36427705         86.0         60.0      71.0

   RR (insp/min)  Temperature F (°F)  Bicarbonate (mEq/L)  Chloride (mEq/L)  \
0           30.0                97.9                 36.0              99.0
1            9.0                97.3                 17.0             109.0

   …  anchor_age  anchor_year  anchor_year_group        dod  \
0  …          91         2175        2014 - 2016        NaT
1  …          91         2184        2014 - 2016 2184-11-22

                      first_careunit                       last_careunit  \
0                 Trauma SICU (TSICU)                 Trauma SICU (TSICU)
1  Surgical Intensive Care Unit (SICU)  Surgical Intensive Care Unit (SICU)

               intime             outtime       los    age_group
0 2176-11-26 02:34:49 2176-11-29 20:58:54  3.766725  91 and Over
1 2184-10-08 04:09:00 2184-10-09 15:55:53  1.490891  91 and Over

[2 rows x 41 columns]
```

[50]:
```
print("prob8_lab_l shape:", prob8_lab_l.shape)
prob8_lab_l.head(2)
```

```
prob8_lab_l shape: (578696, 26)
```

[50]:
```
   subject_id   hadm_id   stay_id  itemid  labevent_id  specimen_id  \
0    10000032  29079034  39553978   50882          449     74344663
1    10000032  29079034  39553978   50902          452     74344663

  order_provider_id           charttime           storetime value  …  \
0              None 2180-07-25 04:45:00 2180-07-25 07:44:00    27  …
1              None 2180-07-25 04:45:00 2180-07-25 07:44:00    97  …

   comments        label  fluid   category            Item_Info  \
```

```
0      None  Bicarbonate  Blood  Chemistry  Bicarbonate (mEq/L)
1      None     Chloride  Blood  Chemistry     Chloride (mEq/L)

                       first_careunit                       last_careunit  \
0  Medical Intensive Care Unit (MICU)  Medical Intensive Care Unit (MICU)
1  Medical Intensive Care Unit (MICU)  Medical Intensive Care Unit (MICU)

               intime             outtime       los
0 2180-07-23 14:00:00 2180-07-23 23:50:47  0.410266
1 2180-07-23 14:00:00 2180-07-23 23:50:47  0.410266

[2 rows x 26 columns]
```

[51]:
```
print("prob8_lab_w shape:", prob8_lab_l.shape)
prob8_lab_w.head(2)
```

```
prob8_lab_w shape: (578696, 26)
```

[51]:
```
   subject_id   hadm_id   stay_id  Bicarbonate (mEq/L)  Chloride (mEq/L)  \
0    10000032  29079034  39553978                 27.0              97.0
1    10000980  26913865  39765666                 24.0             108.0

   Creatinine (mg/dL)  Glucose (mg/dL)  Potassium (mEq/L)  Sodium (mEq/L)  \
0                 0.4            121.0                5.2           130.0
1                 2.2            121.0                4.5           142.0

   Hematocrit (%)  White Blood Cells (K/uL)       los
0            32.1                       4.8  0.410266
1            23.6                       4.8  0.497535
```

[52]:
```
print("prob8_chart_l shape:", prob8_lab_l.shape)
prob8_chart_l.head(2)
```

```
prob8_chart_l shape: (578696, 26)
```

[52]:
```
   subject_id   hadm_id   stay_id  itemid  caregiver_id           charttime  \
0    10000032  29079034  39553978  220179         47007 2180-07-23 21:01:00
1    10000032  29079034  39553978  220180         47007 2180-07-23 21:01:00

            storetime value  valuenum valueuom  … unitname  param_type  \
0 2180-07-23 22:15:00    82      82.0     mmHg  …     mmHg     Numeric
1 2180-07-23 22:15:00    59      59.0     mmHg  …     mmHg     Numeric

  lownormalvalue highnormalvalue    Item_Info  \
0           None            None  NBPs (mmHg)
1           None            None  NBPd (mmHg)
```

```
                        first_careunit                        last_careunit  \
0  Medical Intensive Care Unit (MICU)  Medical Intensive Care Unit (MICU)
1  Medical Intensive Care Unit (MICU)  Medical Intensive Care Unit (MICU)

               intime             outtime       los
0 2180-07-23 14:00:00 2180-07-23 23:50:47  0.410266
1 2180-07-23 14:00:00 2180-07-23 23:50:47  0.410266

[2 rows x 25 columns]
```

[53]:
```
print("prob8_chart_w shape:", prob8_lab_l.shape)
prob8_chart_w.head(2)
```

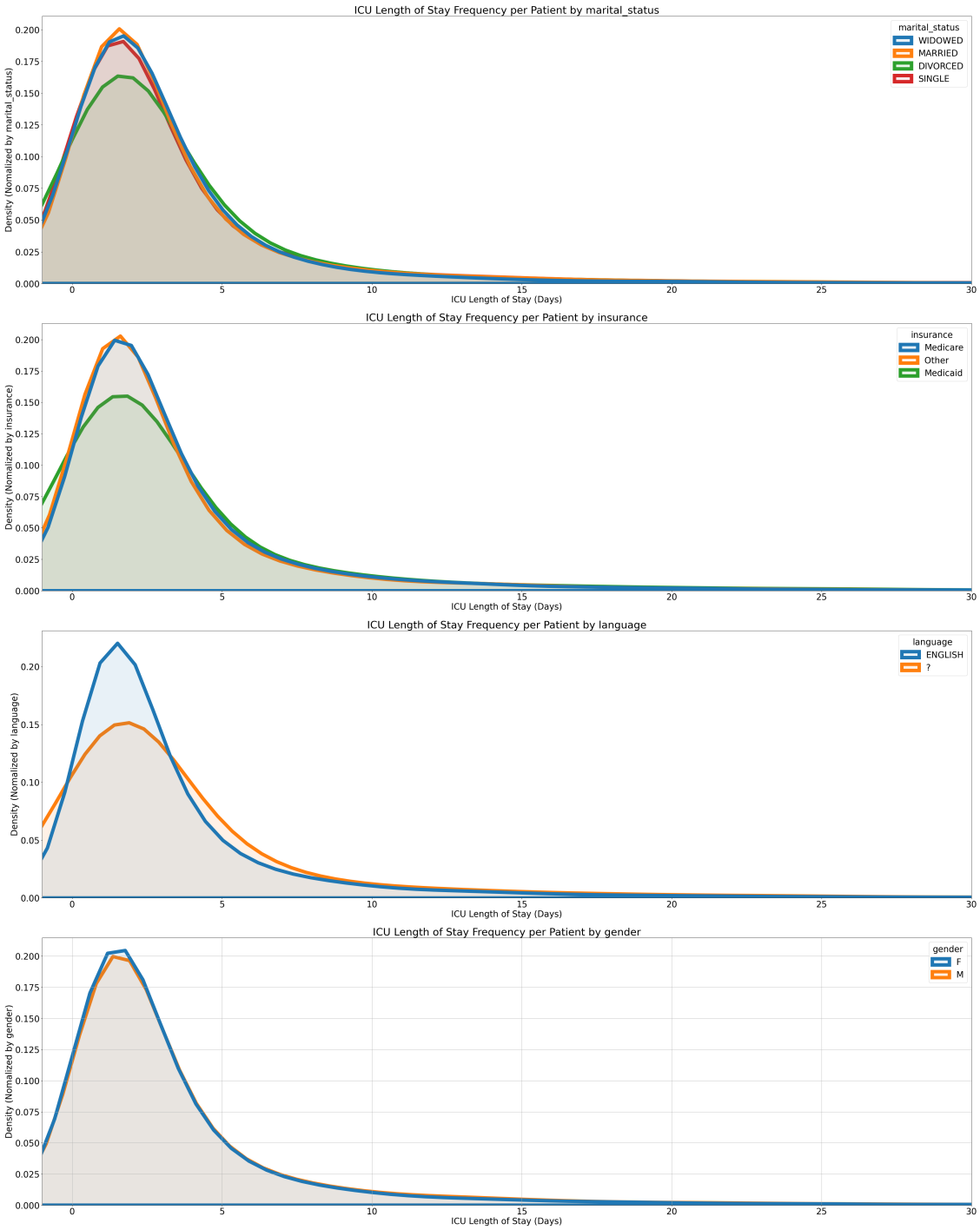
```
prob8_chart_w shape: (578696, 26)
```

[53]:
```
   subject_id   hadm_id   stay_id  NBPs (mmHg)  NBPd (mmHg)  HR (bpm)  \
0    10000032  29079034  39553978         82.0         59.0      94.0
1    10000980  26913865  39765666        150.0         77.0      77.0

   RR (insp/min)  Temperature F (°F)       los
0           20.0                99.5  0.410266
1           23.0                98.0  0.497535
```

Demographics and ICU Length of Stay (los)

[54]:
```
%%time

# List of Facets
facet_list = [
    'marital_status',
    'insurance',
    'language',
    'gender',
]

## Facet Initialization
fig, axes = plt.subplots(
    nrows=len(facet_list), # Facet rows set by length of facet_list
    sharex=False, # independent share x axis across facets
    sharey=False, # independent share y axis across facets
    figsize=(40, 50), # facet size
)

for i, f in enumerate(facet_list):
    ## Plotting
    ax = sns.kdeplot(
        ax=axes[i], # facet location
```

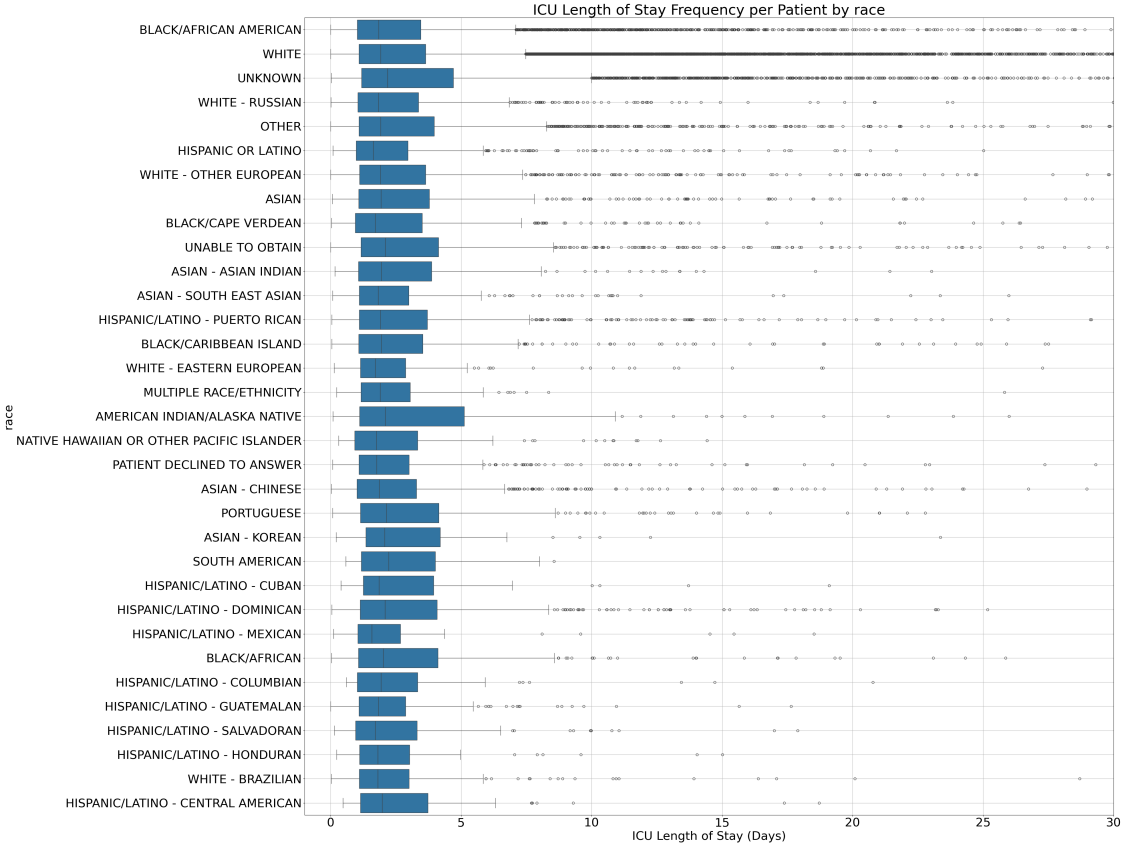
```
        data=prob8, # data
        x="los",
        hue=f,
        common_norm=False, # Sum of all bars is 1
        fill=True, # fill under line
        alpha=0.1, # transparent
        bw_adjust=2,
        linewidth=10, # line width
    )
    title = f"ICU Length of Stay Frequency per Patient by {f}"
    x_title = "ICU Length of Stay (Days)"
    y_title = f"Density (Nomalized by {f})"
    axes[i].set_xlabel(x_title) # label
    axes[i].set_ylabel(y_title) # label
    axes[i].set_title(title) # title of facet plot
    ax.xaxis.set_ticks(x_ticks) # assign tick spacing
    ax.set_xlim(-1, 30) # limit x axis
    plt.grid(True)

plt.tight_layout() # format spacing of facet
plt.rcParams.update({'font.size': 30}) # increase font size
plt.show() # Show faceted plot
```

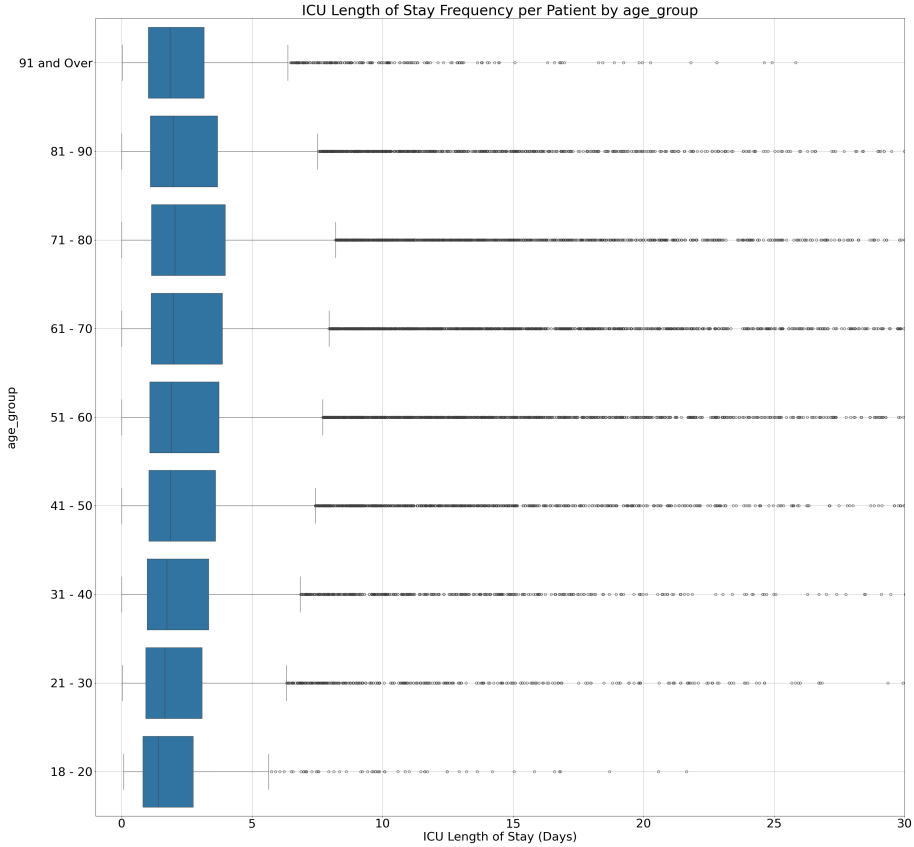



```
CPU times: user 8.57 s, sys: 499 ms, total: 9.07 s
Wall time: 2.08 s
```

[55]:
```
%%time
```

```
facet_list = [
    'race',
    'age_group'
]

## Facet Initialization
fig, axes = plt.subplots(
    nrows=len(facet_list),
    sharex=False, # independent share x axis across facets
    sharey=False, # independent share y axis across facets
    figsize=(40, 60), # facet size
)

for i, f in enumerate(facet_list):
    ## Plotting
    ax = sns.boxplot(
        ax=axes[i], # facet location
        data=prob8, # data
        x="los",
        y=f,
        orient='h'
    )
    title = f"ICU Length of Stay Frequency per Patient by {f}"
    x_title = "ICU Length of Stay (Days)"
    y_title = f"{f}"
    axes[i].set_xlabel(x_title) # label
    axes[i].set_ylabel(y_title) # label
    axes[i].set_title(title) # title of facet plot
    ax.xaxis.set_ticks(x_ticks) # assign tick spacing
    ax.set_xlim(-1, 30) # limit x axis
    ax.grid(True) # add grid

plt.tight_layout() # format spacing of facet
plt.rcParams.update({'font.size': 30}) # increase font size
plt.show() # Show faceted plot
```

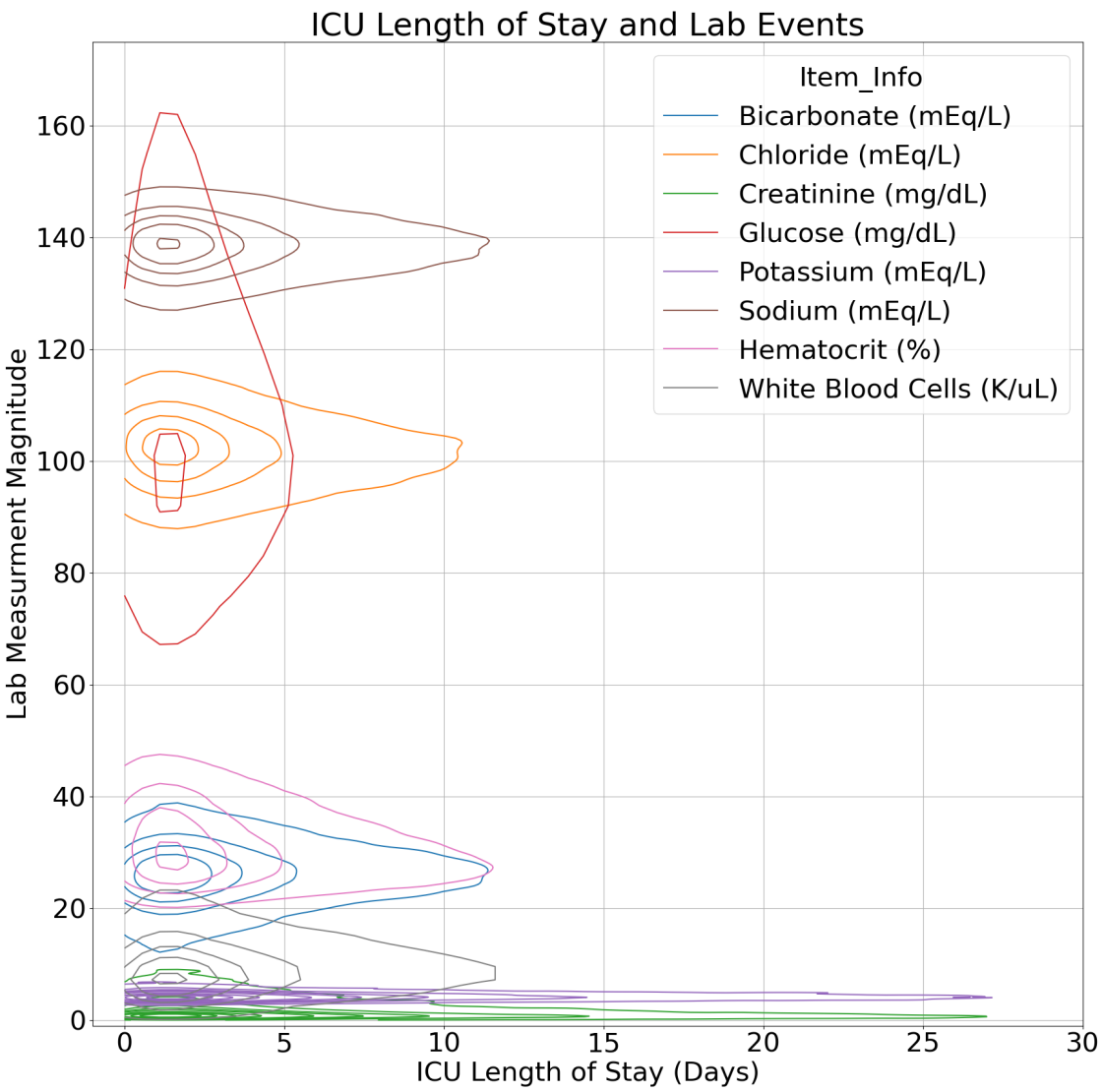

ICU Length of Stay Frequency per Patient by race
race
BLACK/AFRICAN AMERICAN
WHITE
UNKNOWN
WHITE - RUSSIAN
OTHER
HISPANIC OR LATINO
WHITE - OTHER EUROPEAN
ASIAN
BLACK/CAPE VERDEAN
UNABLE TO OBTAIN
ASIAN - ASIAN INDIAN
ASIAN - SOUTH EAST ASIAN
HISPANIC/LATINO - PUERTO RICAN
BLACK/CARIBBEAN ISLAND
WHITE - EASTERN EUROPEAN
MULTIPLE RACE/ETHNICITY
AMERICAN INDIAN/ALASKA NATIVE
NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
PATIENT DECLINED TO ANSWER
ASIAN - CHINESE
PORTUGUESE
ASIAN - KOREAN
SOUTH AMERICAN
HISPANIC/LATINO - CUBAN
HISPANIC/LATINO - DOMINICAN
HISPANIC/LATINO - MEXICAN
BLACK/AFRICAN
HISPANIC/LATINO - COLUMBIAN
HISPANIC/LATINO - GUATEMALAN
HISPANIC/LATINO - SALVADORAN
HISPANIC/LATINO - HONDURAN
WHITE - BRAZILIAN
HISPANIC/LATINO - CENTRAL AMERICAN
0
5
10
15
20
25
30
ICU Length of Stay (Days)


ICU Length of Stay Frequency per Patient by age_group
age_group
91 and Over
81 - 90
71 - 80
61 - 70
51 - 60
41 - 50
31 - 40
21 - 30
18 - 20
0
5
10
15
20
25
30
ICU Length of Stay (Days)

```
CPU times: user 1.11 s, sys: 96.4 ms, total: 1.2 s
Wall time: 1.2 s
```

Lab Events and ICU Length of Stay (los)

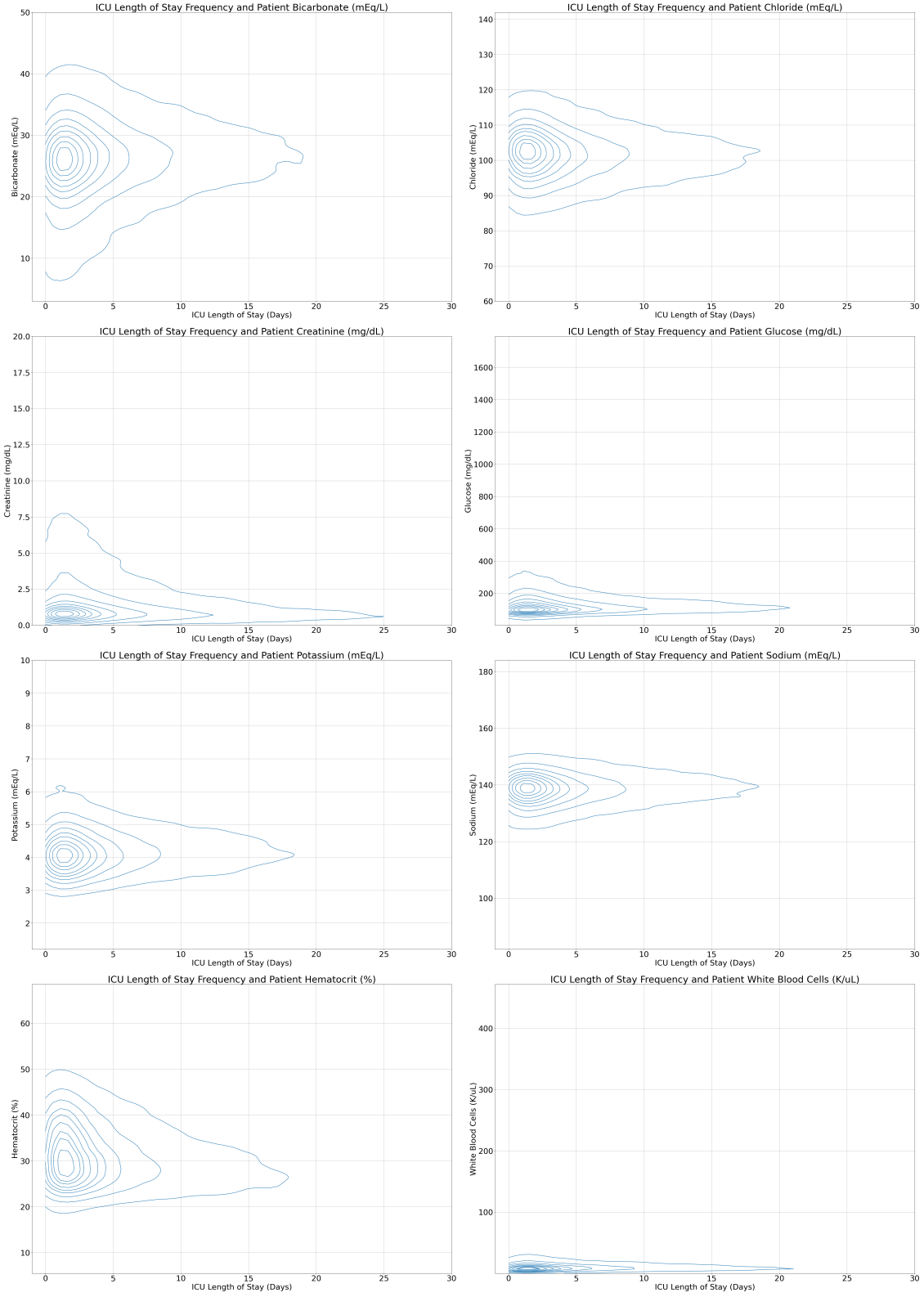
[56]:
```
%%time

plt.figure(figsize=(20, 20)) # init figure
ax = sns.kdeplot( # kde plot
    data=prob8_lab_l, # long lab data
    x="los", # x = length of stay
    y="valuenum", # item value
    hue="Item_Info", # item name and unit
    cut=0, # stop negative contours
)

title = "ICU Length of Stay and Lab Events"
x_title = "ICU Length of Stay (Days)"
y_title = "Lab Measurment Magnitude"
plt.title(title) # title
plt.ylabel(y_title) # y title
plt.xlabel(x_title) # x title
ax.xaxis.set_ticks(x_ticks) # ticks by list. see above
ax.set_xlim(-1, 30) # x limit
ax.set_ylim(-1, 175) # y limit
plt.grid(True) # grid
plt.show()# show plot
```




```
CPU times: user 5min 35s, sys: 667 ms, total: 5min 35s
Wall time: 5min 24s
```

[57]:
```
%%time

facet_list = [
    'Bicarbonate (mEq/L)',
    'Chloride (mEq/L)',
    'Creatinine (mg/dL)',
    'Glucose (mg/dL)',
    'Potassium (mEq/L)',
    'Sodium (mEq/L)',
    'Hematocrit (%)',
    'White Blood Cells (K/uL)',
```

```
]

## Facet Initialization
fig, axes = plt.subplots(
    nrows=4, # Facet rows set by length of facet_list
    ncols=2, # Facet columns set by length of facet_list
    sharex=False, # independent share x axis across facets
    sharey=False, # independent share y axis across facets
    figsize=(50, 70), # facet size
)

for i, f in enumerate(facet_list):
    ## Plotting
    ax = sns.kdeplot( # kde contour plot
        ax=axes[(i//2), (i % 2)], # row: i floordiv by 2 col: i mod of 2
        data=prob8_lab_w, # lab wide
        x="los",
        y=f, # each item name and unit
        cut=0, # no negative contour
    )
    title = f"ICU Length of Stay Frequency and Patient {f}"
    x_title = "ICU Length of Stay (Days)"
    y_title = f"{f}"
    axes[(i//2), (i % 2)].set_xlabel(x_title) # label
    axes[(i//2), (i % 2)].set_ylabel(y_title) # label
    axes[(i//2), (i % 2)].set_title(title) # title of facet plot
    ax.xaxis.set_ticks(x_ticks) # assign tick spacing
    ax.set_xlim(-1, 30) # limit x axis
    ax.grid(True)

plt.tight_layout() # format spacing of facet
plt.rcParams.update({'font.size': 30}) # increase font size
plt.show() # Show facet plot
```

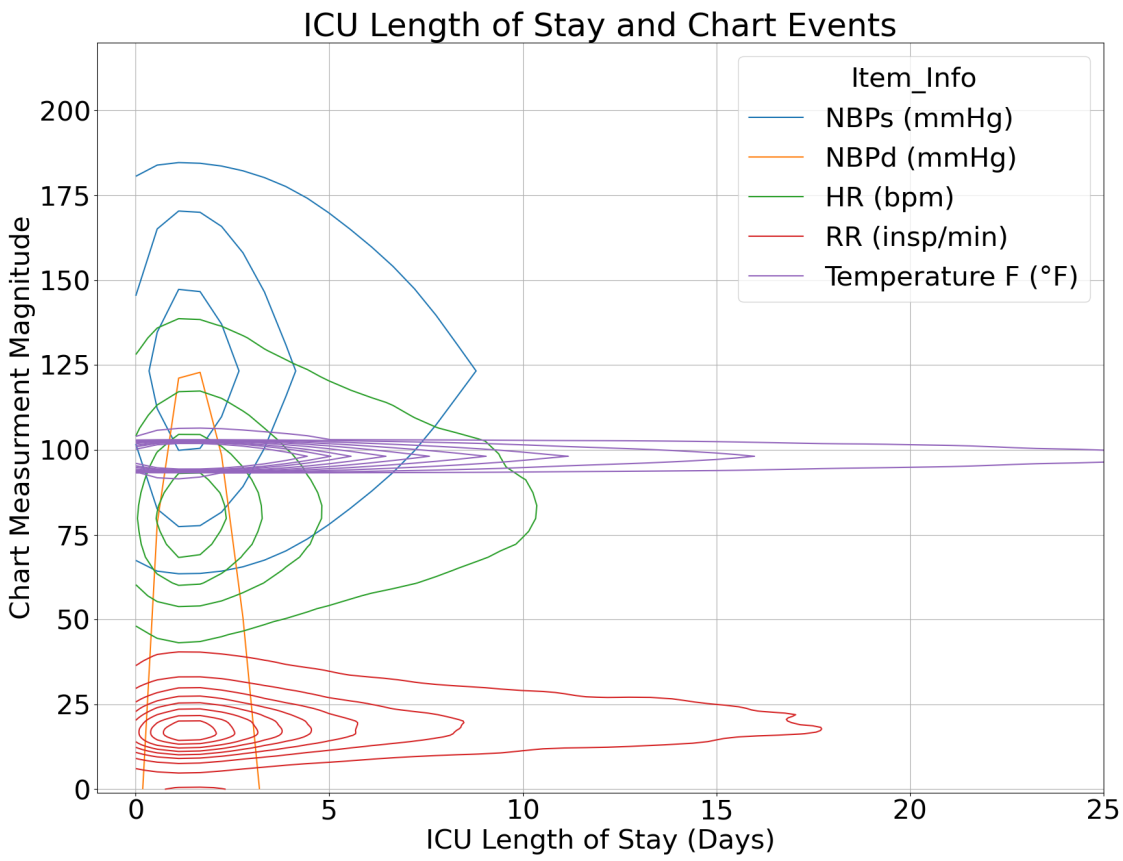


CPU times: user 5min 36s, sys: 791 ms, total: 5min 37s

```
Wall time: 5min 25s
```

Chart Events and ICU Length of Stay (los)

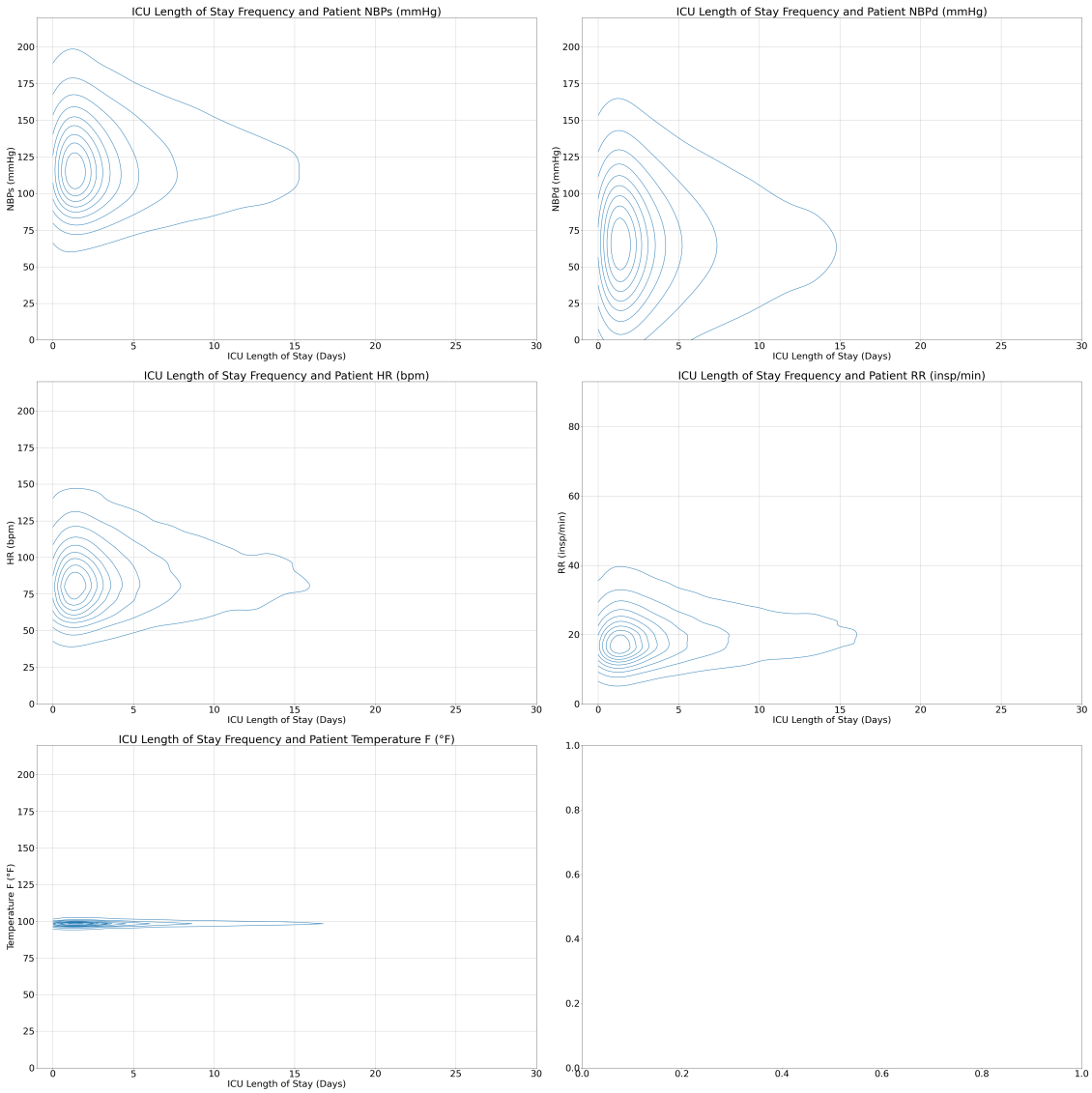
[58]:
```
%%time

plt.figure(figsize=(20, 15)) # init figure and size
ax = sns.kdeplot( # kde contour
    data=prob8_chart_l, # chart long data
    x="los",
    y="valuenum", # value
    hue="Item_Info", # item name and unit
    cut=0, # no neg contours
)

# see above comments. same schema.
title = "ICU Length of Stay and Chart Events"
x_title = "ICU Length of Stay (Days)"
y_title = "Chart Measurment Magnitude"
plt.title(title)
plt.ylabel(y_title)
plt.xlabel(x_title)
ax.xaxis.set_ticks(x_ticks)
ax.set_xlim(-1, 25)
ax.set_ylim(-1, 220)
plt.grid(True)
plt.show()
```




```
CPU times: user 3min 31s, sys: 417 ms, total: 3min 32s
Wall time: 3min 25s
```

[59]:
```
%%time

facet_list = [
    'NBPs (mmHg)',
    'NBPd (mmHg)',
    'HR (bpm)',
    'RR (insp/min)',
    'Temperature F (°F)',
]

## Facet Initialization
fig, axes = plt.subplots(
    nrows=3, # Facet rows set by length of facet_list
    ncols=2, # Facet columns set by length of facet_list
    sharex=False, # independent share x axis across facets
    sharey=False, # independent share y axis across facets
    figsize=(50, 50), # facet size
```

```
)

for i, f in enumerate(facet_list):
    ## Plotting
    ax = sns.kdeplot( # kde contour plot
        ax=axes[(i//2), (i % 2)], # row: i floordiv by 2 col: i mod of 2
        data=prob8_chart_w, # chart wide data
        x="los",
        y=f, # item name and unit
        cut=0, # no neg contour
        clip=[(0, 25), (0, 220)] # remove outliers. Solves contouring error
    )
    title = f"ICU Length of Stay Frequency and Patient {f}"
    x_title = "ICU Length of Stay (Days)"
    y_title = f"{f}"
    axes[(i//2), (i % 2)].set_xlabel(x_title) # label
    axes[(i//2), (i % 2)].set_ylabel(y_title) # label
    axes[(i//2), (i % 2)].set_title(title) # title of facet plot
    ax.xaxis.set_ticks(x_ticks) # assign tick spacing
    ax.set_xlim(-1, 30) # limit x axis
    ax.grid(True)

plt.tight_layout() # format spacing of facet
plt.rcParams.update({'font.size': 30}) # increase font size
plt.show() # Show facet plot
```

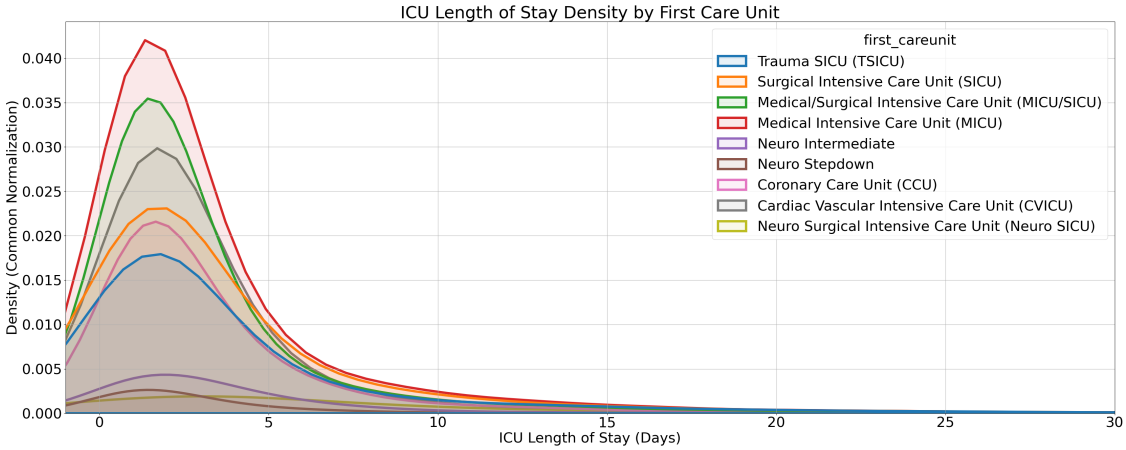



```
CPU times: user 2min 29s, sys: 493 ms, total: 2min 30s
Wall time: 2min 23s
```

First Care Unit and ICU Length of Stay (los)

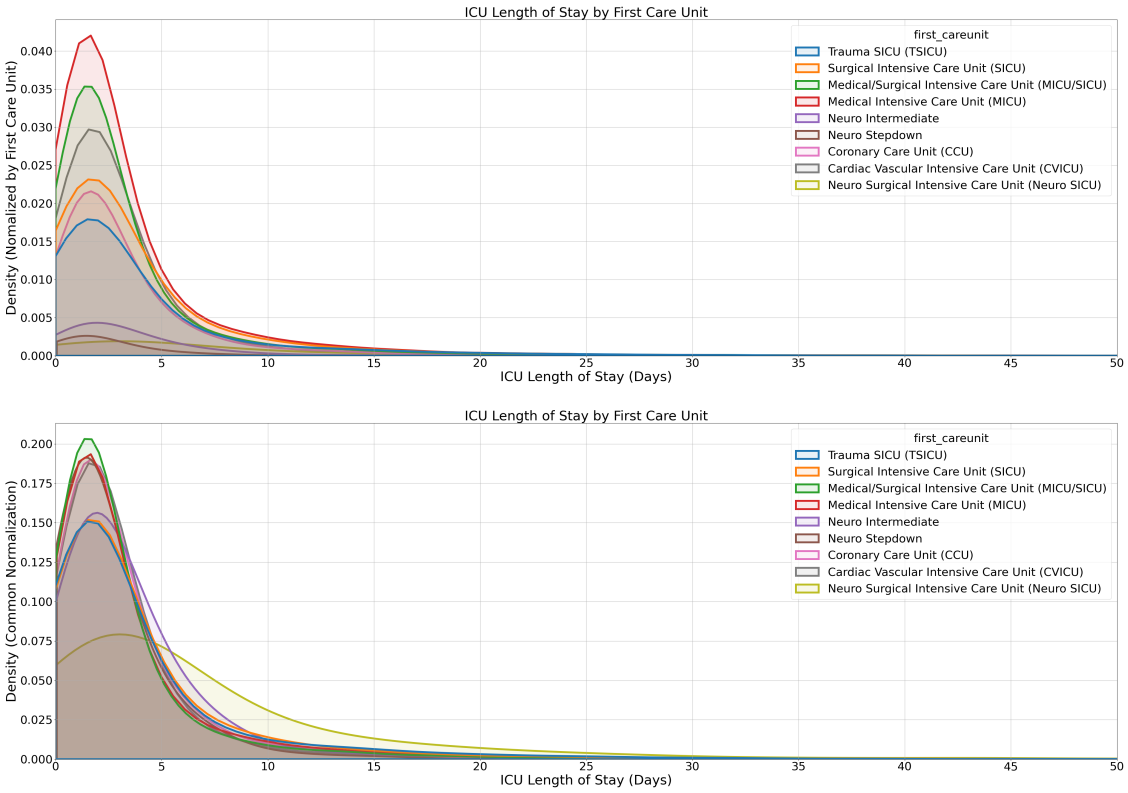
[60]:
```
plt.figure(figsize=(40, 15)) # init figure and size
ax = sns.kdeplot( # kde contour
    data=prob8,
    x="los",
    hue="first_careunit", # first care unit
    common_norm=True, # all area under line sum to 1
    fill=True, # add fill under line
    alpha=0.1, # transparent
    linewidth=5, # line width
```

```
    bw_adjust=2, # smoothness
)


# see comments above. Same schema.
title = "ICU Length of Stay Density by First Care Unit"
x_title = "ICU Length of Stay (Days)"
y_title = "Density (Common Normalization)"
plt.title(title)
plt.ylabel(y_title)
plt.xlabel(x_title)
ax.xaxis.set_ticks(x_ticks)
ax.set_xlim(-1, 30)
plt.grid(True)
plt.show()
```



[61]:
```
fig, axes = plt.subplots( # init facet figure
    nrows=2, # 2 rows
    sharex=False, # dont share axes
    sharey=False,
    squeeze=True,
    figsize=(50, 35), # figure size
)

ax = sns.kdeplot( # kde plot
    ax=axes[0], # first facet
    data=prob8,
    x="los",
    hue="first_careunit", # colors = units
    common_norm=True, # sum of area under all lines = 1
    fill=True, # fill under line
```

```
    alpha=0.1, # transparent
    linewidth=5, # line width
    bw_adjust=2, # smoothness
    cut=0, # no neg contour
)

# See comments above, same schema but for first facet.
title = "ICU Length of Stay by First Care Unit"
x_title = "ICU Length of Stay (Days)"
y_title = f"Density (Nomalized by First Care Unit)"
axes[0].grid(True)
axes[0].set_xlabel(x_title, size=35)
axes[0].set_ylabel(y_title, size=35)
axes[0].set_title(title, size=35)
ax.xaxis.set_ticks(x_ticks)
ax.set_xlim(-0, 50)

# Facet 2nd row plot
ax = sns.kdeplot( #kde
    ax=axes[1], # 2nd row
    data=prob8,
    x="los",
    hue="first_careunit", # color = unit
    common_norm=False, # sum of auc for each unit = 1
    fill=True, # fill under curve
    alpha=0.1, # transparent
    linewidth=5, # line width
    bw_adjust=2, # smoothness
    cut=0, # no neg contour
)

# Same shema as above but for facet 2
title = "ICU Length of Stay by First Care Unit"
x_title = "ICU Length of Stay (Days)"
y_title = "Density (Common Normalization)"
axes[1].grid(True)
axes[1].set_xlabel(x_title, size=35)
axes[1].set_ylabel(y_title, size=35)
axes[1].set_title(title, size=35)
ax.xaxis.set_ticks(x_ticks)
ax.set_xlim(0, 50)

plt.show()
```



1.8.2 Sources

Plot - Title - Plot Font Size 4 - Lineplot - Stripplot - Dodge and Jitter - https://www.w3schools.com/python/matplotlib_grid.asp

Markers - Modifying Markers 1 - Modifying Markers 2 - Modifying Markers 3

Color - Manual Color Map 1 - Manual Color Map 2 - Manual Color Map 3 - Manual Color Map 4

Legend - Custom Legend 1 - Custom Legend 2 - Custom Legend 3 - Regex

- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html
- https://seaborn.pydata.org/tutorial/axis_grids.html
- https://matplotlib.org/stable/api/_as_gen/matplotlib.markers.MarkerStyle.html
- https://dev.to/thalesbruno/subplotting-with-matplotlib-and-seaborn-5ei8
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.suptitle.html
- https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.filter.html
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.isin.html
- https://stackoverflow.com/questions/12096252/use-a-list-of-values-to-select-rows-from-a-pandas-dataframe
- https://www.listendata.com/2019/07/how-to-filter-pandas-dataframe.html
- https://matplotlib.org/stable/gallery/subplots_axes_and_figures/figure_title.html
- https://docs.pola.rs/py-polars/html/reference/expressions/api/polars.Expr.dt.hour.html

- https://stackoverflow.com/questions/12608788/changing-the-tick-frequency-on-the-x-or-y-axis

[ ]: