Hypercap CC NLP Analysis

Table of contents

1 Workbook for MIMIC Hypercapnia Presenting Chief Concern Analysis

This notebook is a deterministic analysis workflow for the NLP-augmented hypercapnia cohort workbook.

1.1 Environment Gate

Fail fast if required packages are missing. Use `uv sync` to repair the environment.

```
import importlib.util

required_packages = [
```

```
    "numpy",
    "pandas",
    "matplotlib",
    "seaborn",
    "statsmodels",
    "upsetplot",
    "openpyxl",
]
missing = [pkg for pkg in required_packages if importlib.util.find_spec(pkg)
 ↪  is None]
if missing:
    raise ModuleNotFoundError(
        "Missing required packages: "
        + ", ".join(missing)
        + ". Run `uv sync` from the repository root and rerun the notebook."
    )
print("Environment check passed.")
```

```
Environment check passed.
```

1.2 Load Data

Use a single canonical workbook path under `MIMIC tabular data`.

```
import json
import os
import sys
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.proportion import proportion_confint
from upsetplot import UpSet, from_indicators

WORK_DIR = Path(os.getenv("WORK_DIR", Path.cwd())).expanduser().resolve()
SRC_DIR = WORK_DIR / "src"
if SRC_DIR.exists() and str(SRC_DIR) not in sys.path:
    sys.path.insert(0, str(SRC_DIR))
CANONICAL_NLP_FILENAME = "MIMICIV all with CC_with_NLP.xlsx"
```

```
def resolve_analysis_input_path(work_dir: Path, input_filename: str | None =
↪  None) -> Path:
    filename = input_filename or CANONICAL_NLP_FILENAME
    input_path = (work_dir / "MIMIC tabular data" /
↪  filename).expanduser().resolve()
    if not input_path.exists():
        raise FileNotFoundError(
            "Expected analysis input workbook was not found at "
            f"{input_path}. Run the classifier notebook first or set "
            "ANALYSIS_INPUT_FILENAME."
        )
    return input_path


def ensure_required_columns(df: pd.DataFrame, required: list[str]) -> None:
    missing = sorted(set(required).difference(df.columns))
    if missing:
        raise KeyError(f"Missing required columns: {missing}")


def to_binary_flag(series: pd.Series) -> pd.Series:
    numeric = pd.to_numeric(series, errors="coerce").fillna(0)
    return (numeric > 0).astype(int)


def _binary_or_zero(df: pd.DataFrame, column: str) -> pd.Series:
    if column in df.columns:
        return to_binary_flag(df[column])
    return pd.Series(0, index=df.index, dtype="int64")


def classify_icd_category_vectorized(df: pd.DataFrame) -> pd.Series:
    j9602 = _binary_or_zero(df, "ICD10_J9602")
    j9612 = _binary_or_zero(df, "ICD10_J9612")
    j9622 = _binary_or_zero(df, "ICD10_J9622")
    j9692 = _binary_or_zero(df, "ICD10_J9692")
    e662 = _binary_or_zero(df, "ICD10_E662")
    icd9_27803 = _binary_or_zero(df, "ICD9_27803")

    category = np.select(
        [
            j9602.eq(1),
```

```
            j9612.eq(1),
            j9622.eq(1),
            j9692.eq(1),
            e662.eq(1) | icd9_27803.eq(1),
        ],
        [
            "Acute RF with hypoxia",
            "Acute RF with hypercapnia",
            "Acute RF with hypoxia & hypercapnia",
            "Respiratory failure, unspecified",
            "Obesity hypoventilation syndrome",
        ],
        default="Other / None",
    )
    return pd.Series(category, index=df.index, name="icd_category")


def classify_inclusion_type_vectorized(any_icd: pd.Series, gas_any:
 ↪  pd.Series) -> pd.Series:
    any_icd_bin = to_binary_flag(any_icd)
    gas_any_bin = to_binary_flag(gas_any)
    labels = np.select(
        [
            any_icd_bin.eq(1) & gas_any_bin.eq(1),
            any_icd_bin.eq(1) & gas_any_bin.eq(0),
            any_icd_bin.eq(0) & gas_any_bin.eq(1),
        ],
        ["Both", "ICD_only", "Gas_only"],
        default="Neither",
    )
    return pd.Series(labels, index=any_icd.index, name="inclusion_type")


def binary_crosstab_yes_no(df: pd.DataFrame, row_col: str, flag_col: str) ->
 ↪  pd.DataFrame:
    ensure_required_columns(df, [row_col, flag_col])
    tab = pd.crosstab(df[row_col], to_binary_flag(df[flag_col]),
 ↪  margins=False, dropna=False)
    tab = tab.reindex(columns=[0, 1], fill_value=0)
    tab.columns = ["No", "Yes"]
    row_totals = tab.sum(axis=1).replace(0, np.nan)
    tab["Percent_yes"] = (tab["Yes"] / row_totals * 100).round(1).fillna(0)
    return tab
```

```
def symptom_distribution_by_overlap(
    df: pd.DataFrame,
    group_col: str,
    symptom_col: str,
    top_k: int = 10,
) -> tuple[pd.DataFrame, pd.DataFrame]:
    ensure_required_columns(df, [group_col, symptom_col])
    tmp = df.dropna(subset=[group_col, symptom_col]).copy()
    if tmp.empty:
        return pd.DataFrame(columns=[group_col, "symptom_group", "N",
         ↪  "Percent"]), pd.DataFrame()
    top_symptoms =
 ↪  tmp[symptom_col].value_counts(dropna=False).head(top_k).index
    tmp["symptom_group"] =
 ↪  tmp[symptom_col].where(tmp[symptom_col].isin(top_symptoms), "Other")
    counts = (
        tmp.groupby([group_col, "symptom_group"], dropna=False)
        .size()
        .reset_index(name="N")
    )
    counts["Percent"] = (
        counts.groupby(group_col)["N"].transform(lambda x: x / x.sum() *
 ↪  100).round(1)
    )
    pivot = counts.pivot_table(
        index="symptom_group",
        columns=group_col,
        values="Percent",
        fill_value=0,
    ).round(1)
    return counts, pivot


def classify_gas_source_overlap(
    abg_series: pd.Series,
    vbg_series: pd.Series,
    other_series: pd.Series,
) -> pd.Series:
    abg = to_binary_flag(abg_series)
    vbg = to_binary_flag(vbg_series)
    other = to_binary_flag(other_series)
```

```
    labels = np.select(
        [
            abg.eq(1) & vbg.eq(1) & other.eq(1),
            abg.eq(1) & vbg.eq(1) & other.eq(0),
            abg.eq(1) & vbg.eq(0) & other.eq(1),
            abg.eq(0) & vbg.eq(1) & other.eq(1),
            abg.eq(1) & vbg.eq(0) & other.eq(0),
            abg.eq(0) & vbg.eq(1) & other.eq(0),
            abg.eq(0) & vbg.eq(0) & other.eq(1),
        ],
        [
            "ABG+VBG+OTHER",
            "ABG+VBG",
            "ABG+OTHER",
            "VBG+OTHER",
            "ABG-only",
            "VBG-only",
            "OTHER-only",
        ],
        default="No-gas",
    )
    return pd.Series(labels, index=abg_series.index,
     ↪  name="gas_source_overlap")


def select_preferred_vital_column(
    df: pd.DataFrame,
    *,
    clean_column: str,
    fallback_model_column: str,
) -> str | None:
    """Select cleaned vital column when available, otherwise fall back to
     ↪  model alias."""
    if clean_column in df.columns:
        return clean_column
    if fallback_model_column in df.columns:
        return fallback_model_column
    return None


def render_latex_longtable(
    table_df: pd.DataFrame,
    *,
```

```
    caption: str,
    label: str,
    landscape: bool = False,
    index: bool = True,
) -> str:
    latex_text = table_df.to_latex(
        index=index,
        escape=False,
        longtable=True,
        caption=caption,
        label=label,
    )
    if landscape:
        latex_text = "\\begin{landscape}\\n" + latex_text +
↪   "\\n\\end{landscape}\\n"
    return latex_text


from hypercap_cc_nlp.pipeline_audit import collect_run_manifest

ANALYSIS_INPUT_FILENAME = os.getenv("ANALYSIS_INPUT_FILENAME")
ANALYSIS_INPUT_PATH = resolve_analysis_input_path(
    WORK_DIR,
    ANALYSIS_INPUT_FILENAME if ANALYSIS_INPUT_FILENAME else None,
)
OUTPUT_DIR = WORK_DIR

HYPERCAP_CRITERIA = [
    "any_hypercap_icd",
    "abg_hypercap_threshold",
    "vbg_hypercap_threshold",
    "other_hypercap_threshold",
    "pco2_threshold_any",
]

SYMPTOM_COL = "RFV1_name"

df = pd.read_excel(ANALYSIS_INPUT_PATH, engine="openpyxl")
required_analysis_cols = sorted({SYMPTOM_COL, *HYPERCAP_CRITERIA})
try:
    ensure_required_columns(df, required_analysis_cols)
except KeyError as exc:
    raise KeyError(
```

```
        "Analysis input schema mismatch. Run 'Hypercap CC NLP Classifier.qmd'
        ↪  "
        f"to regenerate '{CANONICAL_NLP_FILENAME}' before running analysis."
    ) from exc

for column in HYPERCAP_CRITERIA:
    df[column] = to_binary_flag(df[column])

print(
    f"Loaded {ANALYSIS_INPUT_PATH.name}: {df.shape[0]:,} rows x
    ↪  {df.shape[1]:,} columns"
)
print(f"Analysis input path: {ANALYSIS_INPUT_PATH}")
```

```
Loaded MIMICIV all with CC_with_NLP.xlsx: 27,975 rows x 266 columns
Analysis input path: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Res
CC-NLP/MIMIC tabular data/MIMICIV all with CC_with_NLP.xlsx
```

1.3 Descriptive Checks

Compute core cohort summaries with guarded column checks.

```
gender_candidates = [col for col in df.columns if
↪  col.lower().startswith("gender")]
if not gender_candidates:
    raise KeyError("No gender-like column found. Expected a column starting
    ↪  with 'gender'.")
gender_col = gender_candidates[0]

gender_summary = (
    df[gender_col]
    .value_counts(dropna=False)
    .rename_axis(gender_col)
    .to_frame("Count")
)
gender_summary["Percent"] = (gender_summary["Count"] / len(df) *
↪  100).round(1)

age_summary = pd.Series(
    {
        "Mean": round(float(df["age"].mean()), 2),
        "SD": round(float(df["age"].std()), 2),
```

```
        "Q1": round(float(df["age"].quantile(0.25)), 2),
        "Q3": round(float(df["age"].quantile(0.75)), 2),
    },
    name="Age (years)",
)

prevalence_label_map = {
    "any_hypercap_icd": "Hypercapnic RF ICD (any)",
    "abg_hypercap_threshold": "ABG hypercapnia threshold",
    "vbg_hypercap_threshold": "VBG hypercapnia threshold",
    "other_hypercap_threshold": "PCO2 OTHER threshold",
    "pco2_threshold_any": "PCO2 threshold any source",
}
cohort_n = int(len(df))
hypercap_prevalence = (
    pd.DataFrame(
        {
            "Definition": [prevalence_label_map[col] for col in
             ↪  HYPERCAP_CRITERIA],
            "Column": HYPERCAP_CRITERIA,
            "Count": [int(df[col].sum()) for col in HYPERCAP_CRITERIA],
            "Denominator_N": [cohort_n for _ in HYPERCAP_CRITERIA],
            "Percent": [round(float(df[col].mean() * 100), 1) for col in
             ↪  HYPERCAP_CRITERIA],
        }
    )
    .set_index("Definition")
    .sort_values("Count", ascending=False)
)

display(gender_summary)
display(age_summary.to_frame())
display(hypercap_prevalence)
```

gender	Count	Percent
M	14752	52.7
F	13223	47.3

	Age (years)
Mean	65.03
SD	17.48
Q1	55.00
Q3	78.00

Definition	Column	Count	Denominator_N	Percent
PCO2 threshold any source	pco2_threshold_any	27659	27975	98.9
PCO2 OTHER threshold	other_hypercap_threshold	18509	27975	66.2
VBG hypercapnia threshold	vbg_hypercap_threshold	16906	27975	60.4
ABG hypercapnia threshold	abg_hypercap_threshold	4660	27975	16.7
Hypercapnic RF ICD (any)	any_hypercap_icd	1983	27975	7.1

1.4 ED Vitals Data Quality (cleaned-column preference)

Use cleaned ED-vitals columns when available (`*_clean`), falling back to `*_model` aliases only when needed.

```
vital_preference_specs = {
    "triage_temp_f": ("ed_triage_temp_f_clean", "ed_triage_temp_model"),
    "first_temp_f": ("ed_first_temp_f_clean", "ed_first_temp_model"),
    "triage_pain": ("ed_triage_pain_clean", "ed_triage_pain_model"),
    "first_pain": ("ed_first_pain_clean", "ed_first_pain_model"),
    "triage_sbp": ("ed_triage_sbp_clean", "ed_triage_sbp_model"),
    "first_sbp": ("ed_first_sbp_clean", "ed_first_sbp_model"),
    "triage_dbp": ("ed_triage_dbp_clean", "ed_triage_dbp_model"),
    "first_dbp": ("ed_first_dbp_clean", "ed_first_dbp_model"),
    "triage_o2sat": ("ed_triage_o2sat_clean", "ed_triage_o2sat_model"),
    "first_o2sat": ("ed_first_o2sat_clean", "ed_first_o2sat_model"),
}

selected_vital_columns: dict[str, str | None] = {}
vitals_quality_rows: list[dict[str, object]] = []
for vital_name, (clean_col, fallback_col) in vital_preference_specs.items():
    selected_column = select_preferred_vital_column(
        df,
        clean_column=clean_col,
        fallback_model_column=fallback_col,
    )
```

```
    selected_vital_columns[vital_name] = selected_column
    if selected_column is None:
        vitals_quality_rows.append(
            {
                "vital_name": vital_name,
                "selected_column": None,
                "n_non_missing": 0,
                "median": np.nan,
                "mean": np.nan,
            }
        )
        continue
    numeric = pd.to_numeric(df[selected_column], errors="coerce")
    vitals_quality_rows.append(
        {
            "vital_name": vital_name,
            "selected_column": selected_column,
            "n_non_missing": int(numeric.notna().sum()),
            "median": float(numeric.median()) if numeric.notna().any() else
 ↪  np.nan,
            "mean": float(numeric.mean()) if numeric.notna().any() else
 ↪  np.nan,
        }
    )

vitals_quality_summary =
 ↪  pd.DataFrame(vitals_quality_rows).sort_values("vital_name")
display(vitals_quality_summary)

print(
    "Cohort-run ED vitals audits are written under "
    "'MIMIC tabular data/prior runs/YYYY-MM-DD ed_vitals_*.csv'."
)
```

	vital_name	selected_column	n_non_missing	median	mean
7	first_dbp	ed_first_dbp_model	22782	72.0	72.521333
9	first_o2sat	ed_first_o2sat_model	22094	98.0	96.968159
3	first_pain	None	0	NaN	NaN
5	first_sbp	ed_first_sbp_model	22788	128.0	129.722749
1	first_temp_f	ed_first_temp_model	17510	98.1	98.268294
6	triage_dbp	ed_triage_dbp_model	21884	72.0	73.176522
8	triage_o2sat	ed_triage_o2sat_model	21663	98.0	96.926303

	vital_name	selected_column	n_non_missing	median	mean
2	triage_pain	None	0	NaN	NaN
4	triage_sbp	ed_triage_sbp_model	21992	129.0	130.723172
0	triage_temp_f	ed_triage_temp_model	20993	98.0	98.182019

```
Cohort-run ED vitals audits are written under 'MIMIC tabular data/prior runs/YYYY-
MM-DD ed_vitals_*.csv'.
```

1.5 ICD And Inclusion Categories

Use vectorized helper functions to avoid row-wise `apply(axis=1)`.

```
df["icd_category"] = classify_icd_category_vectorized(df)
df["inclusion_type"] = classify_inclusion_type_vectorized(
    df["any_hypercap_icd"],
    df["pco2_threshold_any"],
)

icd_category_summary = (
    df["icd_category"]
    .value_counts(dropna=False)
    .rename_axis("ICD Category")
    .to_frame("Count")
)
icd_category_summary["Percent"] = (icd_category_summary["Count"] / len(df) *
 ↪  100).round(1)
icd_category_summary["Denominator_N"] = int(len(df))

inclusion_summary = (
    df["inclusion_type"]
    .value_counts(dropna=False)
    .rename_axis("Inclusion Type")
    .to_frame("Count")
)
inclusion_summary["Percent"] = (inclusion_summary["Count"] / len(df) *
 ↪  100).round(1)
inclusion_summary["Denominator_N"] = int(len(df))

icd_positive_df = df.loc[df["any_hypercap_icd"].eq(1)].copy()
icd_positive_n = int(len(icd_positive_df))
icd_positive_breakdown = pd.DataFrame(
```

```
    {
        "Definition": [
            "ABG threshold positive",
            "VBG threshold positive",
            "PCO2 OTHER threshold positive",
            "Any gas threshold positive",
        ],
        "Count": [
            int(icd_positive_df["abg_hypercap_threshold"].sum()),
            int(icd_positive_df["vbg_hypercap_threshold"].sum()),
            int(icd_positive_df["other_hypercap_threshold"].sum()),
            int(icd_positive_df["pco2_threshold_any"].sum()),
        ],
    }
)
if icd_positive_n > 0:
    icd_positive_breakdown["Percent"] = (
        icd_positive_breakdown["Count"] / icd_positive_n * 100
    ).round(1)
else:
    icd_positive_breakdown["Percent"] = 0.0
icd_positive_breakdown["Denominator_N"] = icd_positive_n

icd_positive_category_summary = (
    icd_positive_df["icd_category"]
    .value_counts(dropna=False)
    .rename_axis("ICD Category (ICD-positive subset)")
    .to_frame("Count")
)
if icd_positive_n > 0:
    icd_positive_category_summary["Percent"] = (
        icd_positive_category_summary["Count"] / icd_positive_n * 100
    ).round(1)
else:
    icd_positive_category_summary["Percent"] = 0.0
icd_positive_category_summary["Denominator_N"] = icd_positive_n

display(icd_category_summary)
display(inclusion_summary)
display(icd_positive_category_summary)
display(icd_positive_breakdown)
```

ICD Category	Count	Percent	Denominator_N
Other / None	25992	92.9	27975
Acute RF with hypoxia	793	2.8	27975
Obesity hypoventilation syndrome	524	1.9	27975
Acute RF with hypoxia & hypercapnia	386	1.4	27975
Respiratory failure, unspecified	187	0.7	27975
Acute RF with hypercapnia	93	0.3	27975

Inclusion Type	Count	Percent	Denominator_N
Gas_only	25992	92.9	27975
Both	1667	6.0	27975
ICD_only	316	1.1	27975

ICD Category (ICD-positive subset)	Count	Percent	Denominator_N
Acute RF with hypoxia	793	40.0	1983
Obesity hypoventilation syndrome	524	26.4	1983
Acute RF with hypoxia & hypercapnia	386	19.5	1983
Respiratory failure, unspecified	187	9.4	1983
Acute RF with hypercapnia	93	4.7	1983

	Definition	Count	Percent	Denominator_N
0	ABG threshold positive	830	41.9	1983
1	VBG threshold positive	1465	73.9	1983
2	PCO2 OTHER threshold positive	1229	62.0	1983
3	Any gas threshold positive	1667	84.1	1983

```
symptom_work_df = df.copy()
symptom_text =
 ↪  symptom_work_df[SYMPTOM_COL].fillna("").astype(str).str.strip()
symptom_work_df["symptom_missing_flag"] = symptom_text.eq("")
top_symptom_labels = symptom_text.loc[~symptom_work_df[
 ↪  "symptom_missing_flag"]].value_counts().head(10).index
symptom_work_df["symptom_group"] = symptom_text.where(
    symptom_text.isin(top_symptom_labels),
```

```
    "Other",
)
symptom_work_df.loc[symptom_work_df["symptom_missing_flag"], "symptom_group"]
 ↪  = "No symptom recorded"

crosstab_tables = {}
for definition in HYPERCAP_CRITERIA:
    definition_table = binary_crosstab_yes_no(symptom_work_df,
 ↪  "symptom_group", definition)
    crosstab_tables[definition] = definition_table.sort_values("Percent_yes",
 ↪  ascending=False)

display(crosstab_tables["pco2_threshold_any"].head(10))

symptom_non_null =
 ↪  symptom_work_df.loc[~symptom_work_df["symptom_missing_flag"]].copy()
```

symptom_group	No	Yes	Percent_yes
Symptom – Digestive	19	3942	99.5
Diseases (patient-stated)	11	1974	99.4
Injuries & adverse effects	13	2099	99.4
Symptom – Nervous	26	3979	99.4
Symptom – Genitourinary	12	1651	99.3
Symptom – General	19	1818	99.0
Abnormal test result	7	657	98.9
Symptom – Circulatory	30	2679	98.9
Other	28	2006	98.6
Symptom – Skin/Hair/Nails	12	680	98.3

1.6 Symptom Composition By Hypercapnia Definition

Generate counts, percentages, and clipped Wald 95% confidence intervals; export stable tables for downstream reporting.

```
definition_long_df = symptom_non_null.melt(
    id_vars=["symptom_group"],
    value_vars=HYPERCAP_CRITERIA,
    var_name="Hypercapnia_Definition",
    value_name="Positive",
)
```

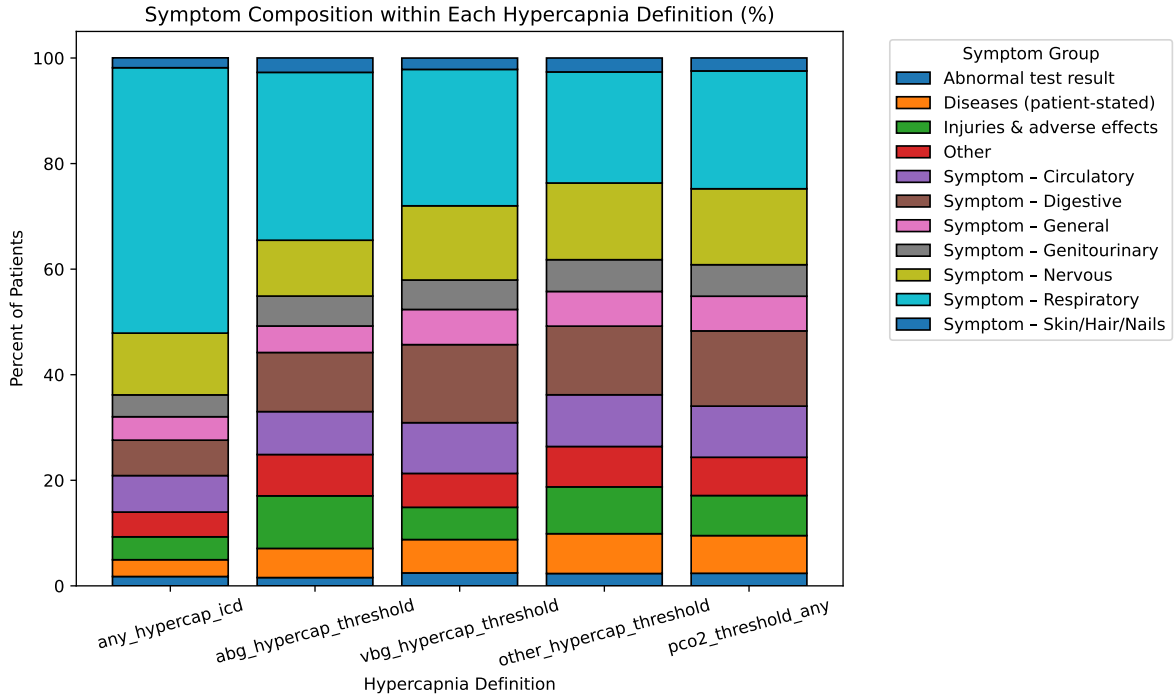
```
definition_positive_df =
 ↪  definition_long_df.loc[definition_long_df["Positive"].eq(1)].copy()

definition_counts_df = (
    definition_positive_df.groupby(["Hypercapnia_Definition",
 ↪  "symptom_group"], dropna=False)
    .size()
    .reset_index(name="Count")
)
definition_counts_df["Total"] = definition_counts_df.groupby(
 ↪  "Hypercapnia_Definition")["Count"].transform("sum")
definition_counts_df["Percent"] = definition_counts_df["Count"] /
 ↪  definition_counts_df["Total"] * 100

p_hat = (definition_counts_df["Percent"] / 100).clip(0, 1)
n_obs = definition_counts_df["Total"].replace(0, np.nan)
se = np.sqrt((p_hat * (1 - p_hat)) / n_obs).fillna(0)
definition_counts_df["CI_lower"] = ((p_hat - 1.96 * se).clip(0, 1) *
 ↪  100).round(2)
definition_counts_df["CI_upper"] = ((p_hat + 1.96 * se).clip(0, 1) *
 ↪  100).round(2)
definition_counts_df["Percent"] = definition_counts_df["Percent"].round(2)

definition_counts_df = definition_counts_df.sort_values(
    ["Hypercapnia_Definition", "Count"],
    ascending=[True, False],
)

definition_pivot_df = definition_counts_df.pivot_table(
    index="symptom_group",
    columns="Hypercapnia_Definition",
    values="Percent",
    fill_value=0,
).round(2)

definition_output_path = OUTPUT_DIR /
 ↪  "Symptom_Composition_by_Hypercapnia_Definition.xlsx"
pivot_output_path = OUTPUT_DIR / "Symptom_Composition_Pivot_ChartReady.xlsx"
definition_counts_df.to_excel(definition_output_path, index=False)
definition_pivot_df.to_excel(pivot_output_path)

display(definition_counts_df.head(12))
print(f"Exported: {definition_output_path}")
```

```
print(f"Exported: {pivot_output_path}")
```

	Hypercapnia_Definition	symptom_group	Count	Total	Percent	CI_lower	CI_uppe
9	abg_hypercap_threshold	Symptom – Respiratory	1482	4660	31.80	30.47	33.14
5	abg_hypercap_threshold	Symptom – Digestive	522	4660	11.20	10.30	12.11
8	abg_hypercap_threshold	Symptom – Nervous	493	4660	10.58	9.70	11.46
2	abg_hypercap_threshold	Injuries & adverse effects	464	4660	9.96	9.10	10.82
4	abg_hypercap_threshold	Symptom – Circulatory	379	4660	8.13	7.35	8.92
3	abg_hypercap_threshold	Other	365	4660	7.83	7.06	8.60
7	abg_hypercap_threshold	Symptom – Genitourinary	264	4660	5.67	5.00	6.33
1	abg_hypercap_threshold	Diseases (patient-stated)	257	4660	5.52	4.86	6.17
6	abg_hypercap_threshold	Symptom – General	234	4660	5.02	4.39	5.65
10	abg_hypercap_threshold	Symptom – Skin/Hair/Nails	127	4660	2.73	2.26	3.19
0	abg_hypercap_threshold	Abnormal test result	73	4660	1.57	1.21	1.92
20	any_hypercap_icd	Symptom – Respiratory	997	1983	50.28	48.08	52.48

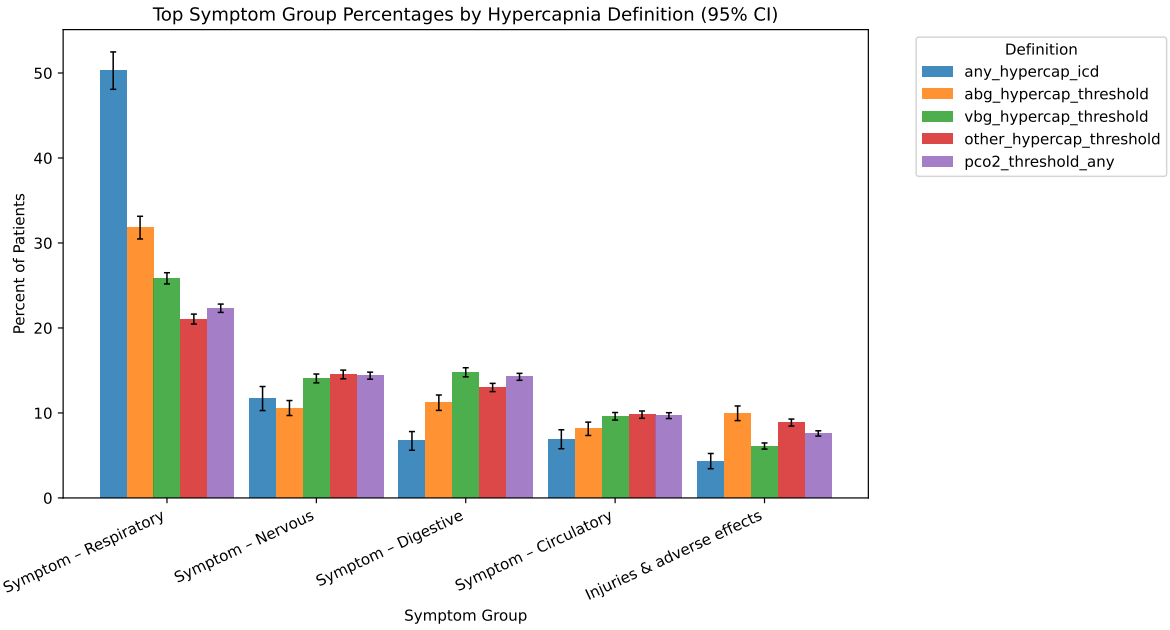
```
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_by_Hypercapnia_Definition.xlsx
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_Pivot_ChartReady.xlsx
```

```
composition_plot_df = definition_pivot_df.T.loc[HYPERCAP_CRITERIA]

ax = composition_plot_df.plot(
    kind="bar",
    stacked=True,
    figsize=(10, 6),
    width=0.8,
    edgecolor="black",
)
ax.set_title("Symptom Composition within Each Hypercapnia Definition (%)")
ax.set_xlabel("Hypercapnia Definition")
ax.set_ylabel("Percent of Patients")
ax.tick_params(axis="x", labelrotation=15)
ax.legend(title="Symptom Group", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.show()
```




```
top_for_ci = (
    definition_counts_df.groupby("symptom_group")["Count"]
    .sum()
    .sort_values(ascending=False)
    .head(5)
    .index
)
ci_plot_df = definition_counts_df.loc[definition_counts_df["symptom_group"
↪  ].isin(top_for_ci)].copy()
symptom_order = list(top_for_ci)
definition_order = HYPERCAP_CRITERIA

x = np.arange(len(symptom_order))
width = 0.18

fig, ax = plt.subplots(figsize=(11, 6))
for idx, definition in enumerate(definition_order):
    subset = (
        ci_plot_df.loc[ci_plot_df["Hypercapnia_Definition"].eq(definition)]
        .set_index("symptom_group")
        .reindex(symptom_order)
        .fillna(0)
```

```
    )
    x_pos = x + (idx - (len(definition_order) - 1) / 2) * width
    y = subset["Percent"].to_numpy()
    lower = subset["CI_lower"].to_numpy()
    upper = subset["CI_upper"].to_numpy()

    ax.bar(x_pos, y, width=width, label=definition, alpha=0.85)
    ax.errorbar(
        x_pos,
        y,
        yerr=[y - lower, upper - y],
        fmt="none",
        ecolor="black",
        elinewidth=1,
        capsize=2,
    )

ax.set_xticks(x)
ax.set_xticklabels(symptom_order, rotation=25, ha="right")
ax.set_ylabel("Percent of Patients")
ax.set_xlabel("Symptom Group")
ax.set_title("Top Symptom Group Percentages by Hypercapnia Definition (95%
 ↪  CI)")
ax.legend(title="Definition", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.tight_layout()
plt.show()
```



1.7 Symptom Distribution By Ascertainment Overlap

```
overlap_required = [
    SYMPTOM_COL,
    "abg_hypercap_threshold",
    "vbg_hypercap_threshold",
    "other_hypercap_threshold",
    "any_hypercap_icd",
    "pco2_threshold_any",
]
ensure_required_columns(df, overlap_required)

abg_flag = to_binary_flag(df["abg_hypercap_threshold"])
vbg_flag = to_binary_flag(df["vbg_hypercap_threshold"])
other_flag = to_binary_flag(df["other_hypercap_threshold"])
icd_flag = to_binary_flag(df["any_hypercap_icd"])
gas_flag = to_binary_flag(df["pco2_threshold_any"])

gas_source_labels = classify_gas_source_overlap(abg_flag, vbg_flag,
 ↪  other_flag)
abg_vbg_labels = np.select(
    [
        abg_flag.eq(1) & vbg_flag.eq(1),
```

```
        abg_flag.eq(1) & vbg_flag.eq(0),
        abg_flag.eq(0) & vbg_flag.eq(1),
    ],
    ["ABG+VBG", "ABG-only", "VBG-only"],
    default="Neither",
)

icd_gas_labels = np.select(
    [
        icd_flag.eq(1) & gas_flag.eq(1),
        icd_flag.eq(1) & gas_flag.eq(0),
        icd_flag.eq(0) & gas_flag.eq(1),
    ],
    ["ICD+Gas", "ICD-only", "Gas-only"],
    default="Neither",
)

overlap_df = df.copy()
overlap_df["gas_source_overlap"] = gas_source_labels
overlap_df["abg_vbg_overlap"] = abg_vbg_labels
overlap_df["icd_gas_overlap"] = icd_gas_labels

gas_positive_df = overlap_df.loc[abg_flag.eq(1) | vbg_flag.eq(1) |
↪  other_flag.eq(1)].copy()
abg_vbg_positive_df = overlap_df.loc[abg_flag.eq(1) | vbg_flag.eq(1)].copy()
abg_vbg_counts_df, abg_vbg_pivot_df = symptom_distribution_by_overlap(
    abg_vbg_positive_df,
    group_col="abg_vbg_overlap",
    symptom_col=SYMPTOM_COL,
    top_k=10,
)
gas_source_counts_df, gas_source_pivot_df = symptom_distribution_by_overlap(
    gas_positive_df,
    group_col="gas_source_overlap",
    symptom_col=SYMPTOM_COL,
    top_k=10,
)
icd_gas_counts_df, icd_gas_pivot_df = symptom_distribution_by_overlap(
    overlap_df,
    group_col="icd_gas_overlap",
    symptom_col=SYMPTOM_COL,
    top_k=10,
)
```

```
gas_source_output_path = OUTPUT_DIR /
 ↪  "Symptom_Composition_by_ABG_VBG_Overlap.xlsx"
gas_source_expanded_output_path = OUTPUT_DIR /
 ↪  "Symptom_Composition_by_Gas_Source_Overlap.xlsx"
icd_gas_output_path = OUTPUT_DIR /
 ↪  "Symptom_Composition_by_ICD_Gas_Overlap.xlsx"
abg_vbg_pivot_df.to_excel(gas_source_output_path)
gas_source_pivot_df.to_excel(gas_source_expanded_output_path)
icd_gas_pivot_df.to_excel(icd_gas_output_path)

print("Symptom distribution by ABG/VBG overlap (legacy output):")
display(abg_vbg_pivot_df.head(15))
print("Symptom distribution by ABG/VBG/OTHER overlap (expanded output):")
display(gas_source_pivot_df.head(15))
print("Symptom distribution by ICD/Gas overlap:")
display(icd_gas_pivot_df.head(15))
print(f"Exported: {gas_source_output_path}")
print(f"Exported: {gas_source_expanded_output_path}")
print(f"Exported: {icd_gas_output_path}")
```

```
Symptom distribution by ABG/VBG overlap (legacy output):
```

abg_vbg_overlap symptom_group	ABG+VBG	ABG-only	VBG-only
Abnormal test result	1.6	1.5	2.6
Diseases (patient-stated)	4.9	6.2	6.6
Injuries & adverse effects	8.0	12.1	5.8
Other	6.5	9.2	6.4
Symptom – Circulatory	7.7	8.6	9.9
Symptom – Digestive	9.7	12.8	15.6
Symptom – General	4.7	5.3	7.0
Symptom – Genitourinary	4.8	6.6	5.7
Symptom – Nervous	11.3	9.8	14.5
Symptom – Respiratory	38.4	24.9	23.8
Symptom – Skin/Hair/Nails	2.4	3.0	2.1

```
Symptom distribution by ABG/VBG/OTHER overlap (expanded output):
```

gas_source_overlap symptom_group	ABG+OTHER	ABG+VBG	ABG+VBG+OTHER	OTHER-only	VBG+
Abnormal test result	1.5	0.0	1.6	2.5	2.8
Diseases (patient-stated)	6.2	5.6	4.9	9.0	6.9
Injuries & adverse effects	12.1	11.1	7.9	9.3	7.2
Other	9.2	11.1	6.5	8.4	6.4
Symptom – Circulatory	8.6	27.8	7.5	10.1	10.8
Symptom – Digestive	12.8	5.6	9.7	13.6	13.6
Symptom – General	5.3	0.0	4.8	6.7	7.6
Symptom – Genitourinary	6.6	5.6	4.8	6.6	5.5
Symptom – Nervous	9.8	5.6	11.3	16.3	15.2
Symptom – Respiratory	24.9	16.7	38.5	14.6	21.8
Symptom – Skin/Hair/Nails	3.0	11.1	2.4	2.9	2.1

`Symptom distribution by ICD/Gas overlap:`

icd_gas_overlap symptom_group	Gas-only	ICD+Gas	ICD-only
Abnormal test result	2.4	1.7	2.2
Diseases (patient-stated)	7.4	3.1	3.5
Injuries & adverse effects	7.8	4.4	4.1
Other	7.5	3.9	8.9
Symptom – Circulatory	9.9	6.4	9.5
Symptom – Digestive	14.7	6.8	6.0
Symptom – General	6.7	4.1	6.0
Symptom – Genitourinary	6.1	4.2	3.8
Symptom – Nervous	14.5	12.4	8.2
Symptom – Respiratory	20.5	51.5	44.0
Symptom – Skin/Hair/Nails	2.5	1.5	3.8

```
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_by_ABG_VBG_Overlap.xlsx
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_by_Gas_Source_Overlap.xlsx
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_by_ICD_Gas_Overlap.xlsx
```

1.8 ICD Diagnostic Performance (ICD as predictor)

```
performance_targets = [
    ("pco2_threshold_any", "Gas any"),
    ("abg_hypercap_threshold", "ABG threshold"),
    ("vbg_hypercap_threshold", "VBG threshold"),
    ("other_hypercap_threshold", "PCO2 OTHER threshold"),
]

icd_positive = to_binary_flag(df["any_hypercap_icd"])
performance_rows = []
for target_col, target_label in performance_targets:
    target_positive = to_binary_flag(df[target_col])
    tp = int(((icd_positive == 1) & (target_positive == 1)).sum())
    fp = int(((icd_positive == 1) & (target_positive == 0)).sum())
    fn = int(((icd_positive == 0) & (target_positive == 1)).sum())
    tn = int(((icd_positive == 0) & (target_positive == 0)).sum())

    sens_denom = tp + fn
    ppv_denom = tp + fp
    sensitivity = float(tp / sens_denom) if sens_denom else np.nan
    ppv = float(tp / ppv_denom) if ppv_denom else np.nan
    sens_ci = (
        proportion_confint(tp, sens_denom, alpha=0.05, method="wilson")
        if sens_denom
        else (np.nan, np.nan)
    )
    ppv_ci = (
        proportion_confint(tp, ppv_denom, alpha=0.05, method="wilson")
        if ppv_denom
        else (np.nan, np.nan)
    )
    performance_rows.append(
        {
            "Target": target_label,
            "Target_Column": target_col,
            "TP": tp,
            "FP": fp,
            "FN": fn,
            "TN": tn,
            "Sensitivity": sensitivity,
            "Sensitivity_CI_Lower": sens_ci[0],
            "Sensitivity_CI_Upper": sens_ci[1],
```

```
            "PPV": ppv,
            "PPV_CI_Lower": ppv_ci[0],
            "PPV_CI_Upper": ppv_ci[1],
        }
    )

icd_performance_df = pd.DataFrame(performance_rows)
icd_performance_df[[
    "Sensitivity",
    "Sensitivity_CI_Lower",
    "Sensitivity_CI_Upper",
    "PPV",
    "PPV_CI_Lower",
    "PPV_CI_Upper",
]] = icd_performance_df[[
    "Sensitivity",
    "Sensitivity_CI_Lower",
    "Sensitivity_CI_Upper",
    "PPV",
    "PPV_CI_Lower",
    "PPV_CI_Upper",
]].clip(lower=0.0, upper=1.0)

icd_subset_output_path = OUTPUT_DIR / "ICD_Positive_Subset_Breakdown.xlsx"
icd_performance_output_path = OUTPUT_DIR / "ICD_vs_Gas_Performance.xlsx"
with pd.ExcelWriter(icd_subset_output_path, engine="openpyxl") as writer:
    icd_positive_breakdown.to_excel(writer, index=False,
↪   sheet_name="Gas_criteria")
    icd_positive_category_summary.reset_index().to_excel(
        writer, index=False, sheet_name="ICD_categories"
    )
icd_performance_df.to_excel(icd_performance_output_path, index=False)

display(icd_positive_breakdown)
display(icd_performance_df)
print(f"Exported: {icd_subset_output_path}")
print(f"Exported: {icd_performance_output_path}")
```

	Definition	Count	Percent	Denominator_N
0	ABG threshold positive	830	41.9	1983
1	VBG threshold positive	1465	73.9	1983
2	PCO2 OTHER threshold positive	1229	62.0	1983

	Definition	Count	Percent	Denominator_N
3	Any gas threshold positive	1667	84.1	1983

	Target	Target_Column	TP	FP	FN	TN	Sensitivity	Sensit
0	Gas any	pco2_threshold_any	1667	316	25992	0	0.060270	0.0575
1	ABG threshold	abg_hypercap_threshold	830	1153	3830	22162	0.178112	0.1673
2	VBG threshold	vbg_hypercap_threshold	1465	518	15441	10551	0.086656	0.0825
3	PCO2 OTHER threshold	other_hypercap_threshold	1229	754	17280	8712	0.066400	0.0629

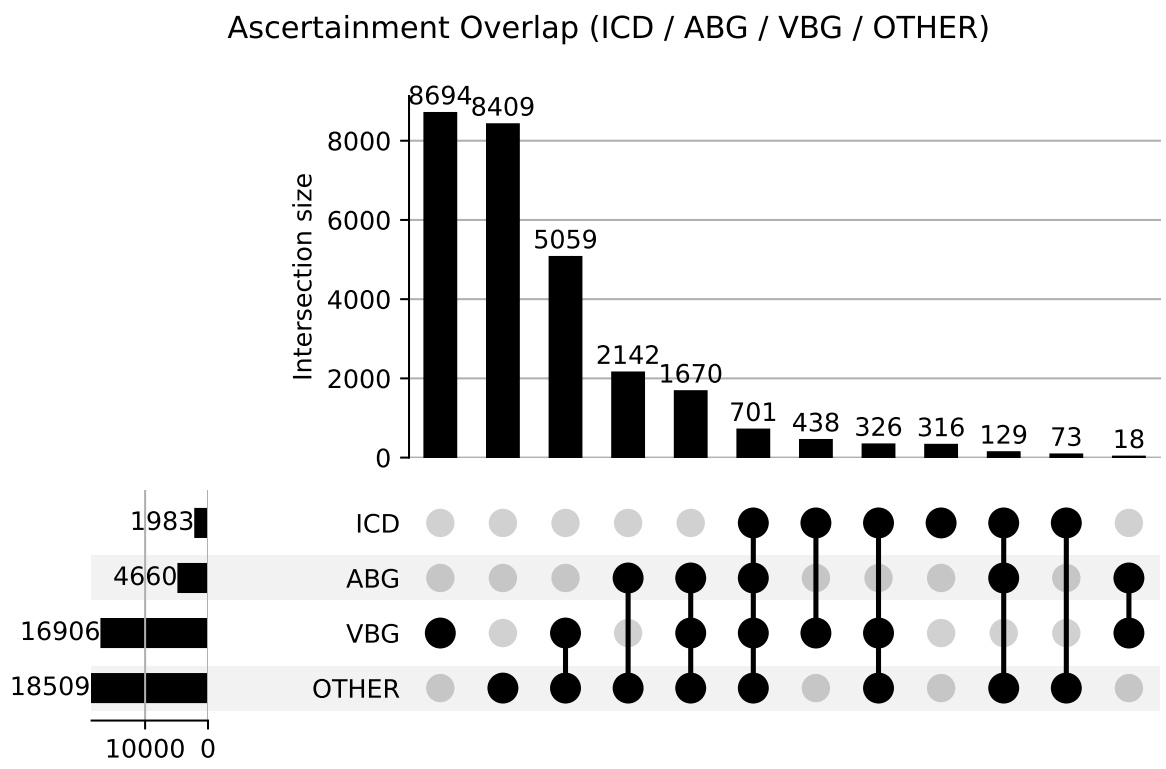
```
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/ICD_Positive_Subset_Breakdown.xlsx
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/ICD_vs_Gas_Performance.xlsx
```

1.9 Ascertainment overlap UpSet

```
ascertainment_flags = pd.DataFrame(
    {
        "ICD": to_binary_flag(df["any_hypercap_icd"]).astype(bool),
        "ABG": to_binary_flag(df["abg_hypercap_threshold"]).astype(bool),
        "VBG": to_binary_flag(df["vbg_hypercap_threshold"]).astype(bool),
        "OTHER": to_binary_flag(df["other_hypercap_threshold"]).astype(bool),
    }
)

upset_series = from_indicators(ascertainment_flags.columns.tolist(),
 ↪  ascertainment_flags)
plt.figure(figsize=(12, 7))
upset_plot = UpSet(
    upset_series,
    subset_size="count",
    show_counts=True,
    sort_by="cardinality",
)
upset_plot.plot()
plt.suptitle("Ascertainment Overlap (ICD / ABG / VBG / OTHER)")
plt.tight_layout()

upset_output_path = OUTPUT_DIR / "Ascertainment_Overlap_UpSet.png"
```

```
plt.savefig(upset_output_path, dpi=300, bbox_inches="tight")
plt.show()

intersection_counts = (
    ascertainment_flags.groupby(["ICD", "ABG", "VBG", "OTHER"], dropna=False)
    .size()
    .reset_index(name="Count")
    .sort_values("Count", ascending=False)
    .reset_index(drop=True)
)
intersection_output_path = OUTPUT_DIR /
 ↪  "Ascertainment_Overlap_Intersections.xlsx"
intersection_counts.to_excel(intersection_output_path, index=False)

display(intersection_counts.head(20))
print(f"Exported: {upset_output_path}")
print(f"Exported: {intersection_output_path}")
```

```
<Figure size 3600x2100 with 0 Axes>
```



	ICD	ABG	VBG	OTHER	Count
0	False	False	True	False	8694
1	False	False	False	True	8409
2	False	False	True	True	5059
3	False	True	False	True	2142
4	False	True	True	True	1670
5	True	True	True	True	701
6	True	False	True	False	438
7	True	False	True	True	326
8	True	False	False	False	316
9	True	True	False	True	129
10	True	False	False	True	73
11	False	True	True	False	18

```
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Ascertainment_Overlap_UpSet.png
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Ascertainment_Overlap_Intersections.xlsx
```

1.10 PDF-ready long tables

```
print(
    render_latex_longtable(
        hypercap_prevalence.reset_index(),
        caption=f"Hypercapnia prevalence summary (denominator = full cohort
 ↪  N={cohort_n:,}).",
        label="tab:prevalence_summary",
        index=False,
    )
)
print(
    render_latex_longtable(
        icd_category_summary.reset_index(),
        caption=f"ICD category composition (denominator = full cohort
 ↪  N={len(df):,}).",
        label="tab:icd_category",
        index=False,
    )
)
print(
    render_latex_longtable(
```

```
        inclusion_summary.reset_index(),
        caption=f"Inclusion source composition (denominator = full cohort
↪  N={len(df):,}).",
        label="tab:inclusion_type",
        index=False,
    )
)
print(
    render_latex_longtable(
        icd_positive_breakdown,
        caption=f"Among ICD-positive encounters, which gas criteria are also
↪  met (denominator = ICD-positive N={icd_positive_n:,}).",
        label="tab:icd_positive_breakdown",
        index=False,
    )
)
print(
    render_latex_longtable(
        icd_positive_category_summary.reset_index(),
        caption=f"Among ICD-positive encounters, ICD category distribution
↪  (denominator = ICD-positive N={icd_positive_n:,}).",
        label="tab:icd_positive_categories",
        index=False,
    )
)
print(
    render_latex_longtable(
        icd_performance_df,
        caption="ICD diagnostic performance vs gas-confirmed hypercapnia
↪  definitions (Wilson 95% CI).",
        label="tab:icd_performance",
        landscape=True,
        index=False,
    )
)
```

```
\begin{longtable}{llrrr}
\caption{Hypercapnia prevalence summary (denominator = full cohort N=27,975).} \label{tab:pre
\toprule
Definition & Column & Count & Denominator_N & Percent \\
\midrule
\endfirsthead
\caption[]{Hypercapnia prevalence summary (denominator = full cohort N=27,975).} \\
```

```
\toprule
Definition & Column & Count & Denominator_N & Percent \\
\midrule
\endhead
\midrule
\multicolumn{5}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
PCO2 threshold any source & pco2_threshold_any & 27659 & 27975 & 98.900000 \\
PCO2 OTHER threshold & other_hypercap_threshold & 18509 & 27975 & 66.200000 \\
VBG hypercapnia threshold & vbg_hypercap_threshold & 16906 & 27975 & 60.400000 \\
ABG hypercapnia threshold & abg_hypercap_threshold & 4660 & 27975 & 16.700000 \\
Hypercapnic RF ICD (any) & any_hypercap_icd & 1983 & 27975 & 7.100000 \\
\end{longtable}

\begin{longtable}{lrrr}
\caption{ICD category composition (denominator = full cohort N=27,975).} \label{tab:icd_categ
\toprule
ICD Category & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{ICD category composition (denominator = full cohort N=27,975).} \\
\toprule
ICD Category & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
Other / None & 25992 & 92.900000 & 27975 \\
Acute RF with hypoxia & 793 & 2.800000 & 27975 \\
Obesity hypoventilation syndrome & 524 & 1.900000 & 27975 \\
Acute RF with hypoxia & hypercapnia & 386 & 1.400000 & 27975 \\
Respiratory failure, unspecified & 187 & 0.700000 & 27975 \\
Acute RF with hypercapnia & 93 & 0.300000 & 27975 \\
\end{longtable}

\begin{longtable}{lrrr}
```

```
\caption{Inclusion source composition (denominator = full cohort N=27,975).} \label{tab:inclu
\toprule
Inclusion Type & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{Inclusion source composition (denominator = full cohort N=27,975).} \\
\toprule
Inclusion Type & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
Gas_only & 25992 & 92.900000 & 27975 \\
Both & 1667 & 6.000000 & 27975 \\
ICD_only & 316 & 1.100000 & 27975 \\
\end{longtable}

\begin{longtable}{lrrr}
\caption{Among ICD-positive encounters, which gas criteria are also met (denominator = ICD-
positive N=1,983).} \label{tab:icd_positive_breakdown} \\
\toprule
Definition & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{Among ICD-positive encounters, which gas criteria are also met (denominator = ICD-
positive N=1,983).} \\
\toprule
Definition & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
ABG threshold positive & 830 & 41.900000 & 1983 \\
VBG threshold positive & 1465 & 73.900000 & 1983 \\
PCO2 OTHER threshold positive & 1229 & 62.000000 & 1983 \\
```

```
Any gas threshold positive & 1667 & 84.100000 & 1983 \\
\end{longtable}

\begin{longtable}{lrrr}
\caption{Among ICD-positive encounters, ICD category distribution (denominator = ICD-
positive N=1,983).} \label{tab:icd_positive_categories} \\
\toprule
ICD Category (ICD-positive subset) & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{Among ICD-positive encounters, ICD category distribution (denominator = ICD-
positive N=1,983).} \\
\toprule
ICD Category (ICD-positive subset) & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
Acute RF with hypoxia & 793 & 40.000000 & 1983 \\
Obesity hypoventilation syndrome & 524 & 26.400000 & 1983 \\
Acute RF with hypoxia & hypercapnia & 386 & 19.500000 & 1983 \\
Respiratory failure, unspecified & 187 & 9.400000 & 1983 \\
Acute RF with hypercapnia & 93 & 4.700000 & 1983 \\
\end{longtable}

\begin{landscape}\n\begin{longtable}{llrrrrrrrrrr}
\caption{ICD diagnostic performance vs gas-confirmed hypercapnia definitions (Wilson 95% CI)
\toprule
Target & Target_Column & TP & FP & FN & TN & Sensitivity & Sensitivity_CI_Lower & Sensitivity
\midrule
\endfirsthead
\caption[]{ICD diagnostic performance vs gas-confirmed hypercapnia definitions (Wilson 95% CI
\toprule
Target & Target_Column & TP & FP & FN & TN & Sensitivity & Sensitivity_CI_Lower & Sensitivity
\midrule
\endhead
\midrule
\multicolumn{12}{r}{Continued on next page} \\
\midrule
```

```
\endfoot
\bottomrule
\endlastfoot
Gas any & pco2_threshold_any & 1667 & 316 & 25992 & 0 & 0.060270 & 0.057526 & 0.063136 & 0.84
ABG threshold & abg_hypercap_threshold & 830 & 1153 & 3830 & 22162 & 0.178112 & 0.167393 & 0
VBG threshold & vbg_hypercap_threshold & 1465 & 518 & 15441 & 10551 & 0.086656 & 0.082508 & 0
PCO2 OTHER threshold & other_hypercap_threshold & 1229 & 754 & 17280 & 8712 & 0.066400 & 0.06
\end{longtable}
\n\end{landscape}\n
```

1.11 Association Model

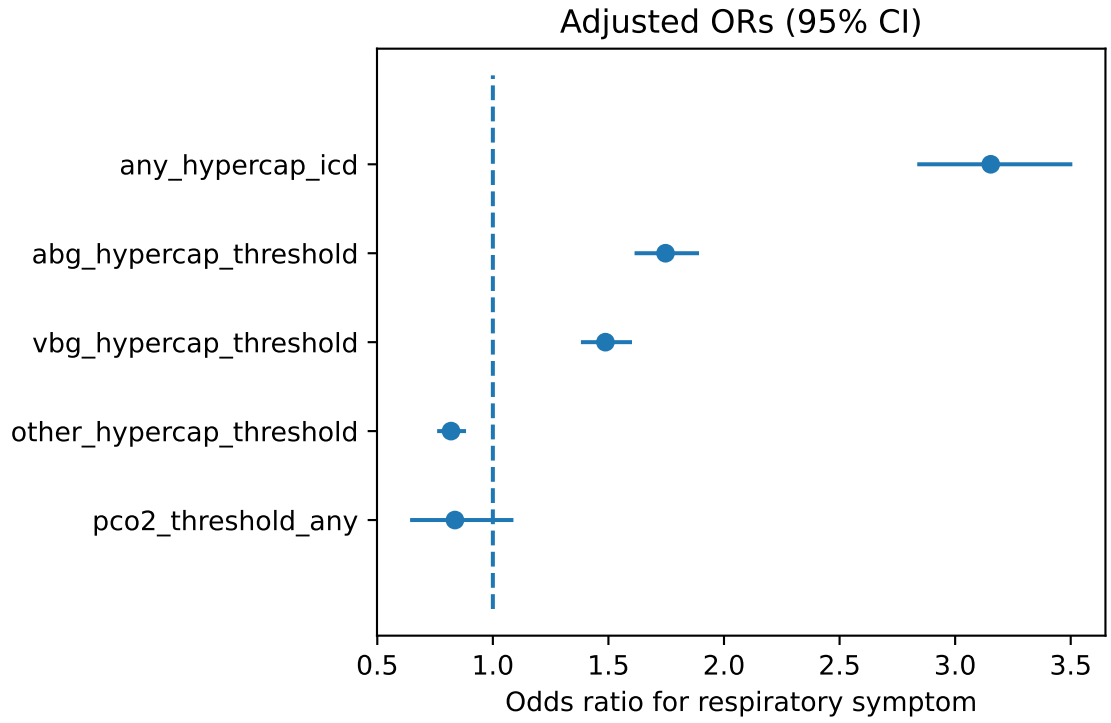
Logistic regression of respiratory symptom flag on hypercapnia definitions.

```
model_df = df.dropna(subset=[SYMPTOM_COL]).copy()
model_df["is_respiratory"] = model_df[SYMPTOM_COL].astype(str).str.contains(
    r"\brespir", case=False, na=False
).astype(int)

design_matrix = sm.add_constant(model_df[HYPERCAP_CRITERIA],
 ↪  has_constant="add")
outcome = model_df["is_respiratory"]
logit_result = sm.Logit(outcome, design_matrix,
 ↪  missing="drop").fit(disp=False)

or_table = pd.DataFrame(
    {
        "OR": np.exp(logit_result.params),
        "CI_lo": np.exp(logit_result.conf_int()[0]),
        "CI_hi": np.exp(logit_result.conf_int()[1]),
        "p": logit_result.pvalues,
    }
).round(3)

display(or_table.loc[HYPERCAP_CRITERIA])
```

	OR	CI_lo	CI_hi	p
any_hypercap_icd	3.154	2.836	3.507	0.000
abg_hypercap_threshold	1.747	1.613	1.892	0.000
vbg_hypercap_threshold	1.487	1.381	1.602	0.000
other_hypercap_threshold	0.819	0.759	0.884	0.000
pco2_threshold_any	0.836	0.642	1.089	0.185

	OR	CI_lo	CI_hi	p

```
or_plot_df = or_table.loc[HYPERCAP_CRITERIA]
y_positions = np.arange(len(or_plot_df))[::-1]

plt.figure(figsize=(6, 4))
plt.hlines(y=y_positions, xmin=or_plot_df["CI_lo"], xmax=or_plot_df["CI_hi"],
 ↪  linewidth=1.5)
plt.plot(or_plot_df["OR"], y_positions, "o")
plt.vlines(1, ymin=-1, ymax=len(or_plot_df), linestyles="dashed")
plt.yticks(y_positions, or_plot_df.index)
plt.xlabel("Odds ratio for respiratory symptom")
plt.title("Adjusted ORs (95% CI)")
plt.tight_layout()
plt.show()
```



1.12 Export Verification

```
expected_outputs = [
    definition_output_path,
```

```
    pivot_output_path,
    gas_source_output_path,
    gas_source_expanded_output_path,
    icd_gas_output_path,
    icd_subset_output_path,
    icd_performance_output_path,
    upset_output_path,
    intersection_output_path,
]

verification_rows = []
for output_path in expected_outputs:
    verification_rows.append(
        {
            "path": str(output_path),
            "exists": output_path.exists(),
            "size_bytes": output_path.stat().st_size if output_path.exists()
             ↪  else 0,
        }
    )

output_verification = pd.DataFrame(verification_rows)
display(output_verification)
```

	path	exists	size_bytes
0	/Users/blocke/Box Sync/Residency Personal File...	True	7371
1	/Users/blocke/Box Sync/Residency Personal File...	True	5568
2	/Users/blocke/Box Sync/Residency Personal File...	True	5385
3	/Users/blocke/Box Sync/Residency Personal File...	True	5563
4	/Users/blocke/Box Sync/Residency Personal File...	True	5366
5	/Users/blocke/Box Sync/Residency Personal File...	True	5861
6	/Users/blocke/Box Sync/Residency Personal File...	True	5583
7	/Users/blocke/Box Sync/Residency Personal File...	True	138690
8	/Users/blocke/Box Sync/Residency Personal File...	True	5223

```
from datetime import datetime

prior_runs_dir = WORK_DIR / "MIMIC tabular data" / "prior runs"
prior_runs_dir.mkdir(parents=True, exist_ok=True)
run_date = datetime.now().strftime("%Y-%m-%d")
```

```
analysis_manifest = collect_run_manifest(
    WORK_DIR,
    run_id=f"analysis_{datetime.now().strftime('%Y%m%d_%H%M%S')}",
)
analysis_manifest["stage"] = "analysis"
analysis_manifest["analysis_input_path"] = str(ANALYSIS_INPUT_PATH)
analysis_manifest["outputs"] = {
    "definition_output_path": str(definition_output_path),
    "pivot_output_path": str(pivot_output_path),
    "abg_vbg_overlap_output_path": str(gas_source_output_path),
    "gas_source_overlap_output_path": str(gas_source_expanded_output_path),
    "icd_gas_overlap_output_path": str(icd_gas_output_path),
    "icd_subset_output_path": str(icd_subset_output_path),
    "icd_performance_output_path": str(icd_performance_output_path),
    "upset_output_path": str(upset_output_path),
    "intersection_output_path": str(intersection_output_path),
}
analysis_manifest["output_verification"] = verification_rows
analysis_manifest_path = prior_runs_dir / f"{run_date}
 ↪  analysis_run_manifest.json"
analysis_manifest_path.write_text(json.dumps(analysis_manifest, indent=2))
print(f"Wrote: {analysis_manifest_path}")
```

```
Wrote: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Projects
CC-NLP/MIMIC tabular data/prior runs/2026-02-17 analysis_run_manifest.json
```