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.

```python
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`.

```python
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+UNKNOWN",
            "ABG+VBG",
            "ABG+UNKNOWN",
            "VBG+UNKNOWN",
            "ABG-only",
            "VBG-only",
            "UNKNOWN-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",
    "unknown_hypercap_threshold",
    "pco2_threshold_any",
    "pco2_threshold_0_24h",
]

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: 11,945 rows x 339 columns
Analysis input path: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Re
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",
    "unknown_hypercap_threshold": "PCO2 UNKNOWN threshold",
    "pco2_threshold_any": "PCO2 threshold any-time",
    "pco2_threshold_0_24h": "PCO2 threshold within 24h marker",
}
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)
```

	Count	Percent
gender		
M	6425	53.8
F	5520	46.2

	Age (years)
Mean	66.01
SD	16.61
Q1	56.00
Q3	78.00

	Column	Count	Denominator_N	Percent
Definition				
PCO2 threshold any-time	pco2_threshold_any	11504	11945	96.3
ABG hypercapnia threshold	abg_hypercap_threshold	7456	11945	62.4
VBG hypercapnia threshold	vbg_hypercap_threshold	6391	11945	53.5
PCO2 threshold within 24h marker	pco2_threshold_0_24h	6220	11945	52.1
Hypercapnic RF ICD (any)	any_hypercap_icd	1983	11945	16.6
PCO2 UNKNOWN threshold	unknown_hypercap_threshold	1347	11945	11.3

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_clean	9221	71.0	72.135994
9	first_o2sat	ed_first_o2sat_clean	8989	97.0	96.229836
3	first_pain	ed_first_pain_clean	7121	0.0	2.668165
5	first_sbp	ed_first_sbp_clean	9224	127.0	128.845403
1	first_temp_f	ed_first_temp_f_clean	6950	98.0	98.155144
6	triage_dbp	ed_triage_dbp_clean	8777	72.0	73.042042

	vital_name	selected_column	n_non_missing	median	mean
8	triage_o2sat	ed_triage_o2sat_clean	8733	97.0	96.195694
2	triage_pain	ed_triage_pain_clean	7631	0.0	2.933888
4	triage_sbp	ed_triage_sbp_clean	8824	128.0	130.065503
0	triage_temp_f	ed_triage_temp_f_clean	8385	98.0	98.094912

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

1.5 Cohort Blood-Gas QC Snapshot

Summarize gas-source quarantine and anchor diagnostics emitted by cohort generation.

```
qa_summary_path = WORK_DIR / "qa_summary.json"
if qa_summary_path.exists():
    qa_summary_payload = json.loads(qa_summary_path.read_text())
    blood_gas_audit_paths = qa_summary_payload.get("blood_gas_audit_paths",
 ↪  {})
    timing_integrity_audit = qa_summary_payload.get("timing_integrity_audit",
 ↪  [{}])
    ventilation_timing_audit =
 ↪  qa_summary_payload.get("ventilation_timing_audit", [{}])
    timing_row = timing_integrity_audit[0] if timing_integrity_audit else {}
    vent_row = ventilation_timing_audit[0] if ventilation_timing_audit else
 ↪  {}
    qualifying_gas_time_observed_rate = (
        float(pd.to_numeric(df["dt_qualifying_hypercapnia_hours"],
         ↪  errors="coerce").notna().mean())
        if "dt_qualifying_hypercapnia_hours" in df.columns
        else None
    )
    qc_rows = [
        {
            "metric": "UNKNOWN semantics",
            "value": "LAB blood-gas unknown specimen only (POC UNKNOWN
             ↪  quarantined).",
        },
        {
            "metric": "panel_unknown_rate",
            "value": qa_summary_payload.get(
                "gas_source_unknown_rate",
                qa_summary_payload.get("source_unknown_rate"),
            ),
        },
        {
            "metric": "encounter_unknown_rate",
            "value": qa_summary_payload.get("gas_source_other_rate"),
        },
        {
            "metric": "POC UNKNOWN quarantined hadm count",
            "value": (
                qa_summary_payload.get("other_route_quarantine_audit",
↪   [{}])[0].get(
                    "poc_other_quarantined_hadm_n"
                )
                if qa_summary_payload.get("other_route_quarantine_audit")
                else None
            ),
        },
        {
            "metric": "POC UNKNOWN leakage into threshold",
            "value": (
                qa_summary_payload.get("other_route_quarantine_audit",
↪   [{}])[0].get(
                    "poc_other_leak_into_other_threshold_n"
                )
                if qa_summary_payload.get("other_route_quarantine_audit")
                else None
            ),
        },
        {
            "metric": "first_gas_without_pco2_anchor_n",
            "value": (
                qa_summary_payload.get("first_gas_anchor_audit",
↪   [{}])[0].get(
                    "first_gas_without_pco2_anchor_n"
                )
                if qa_summary_payload.get("first_gas_anchor_audit")
                else None
            ),
        },
        {
            "metric": "qualifying_gas_time_observed_rate",
            "value": qualifying_gas_time_observed_rate,
        },
        {
            "metric": "timing_usable_for_model_rate",
            "value": (
                float(pd.to_numeric(df.get("timing_usable_for_model"),
                 ↪  errors="coerce").fillna(0).mean())
                if "timing_usable_for_model" in df.columns
                else None
            ),
        },
        {
            "metric": "hospital_los_negative_n",
            "value": timing_row.get("hospital_los_negative_n"),
        },
        {
            "metric": "admittime_before_ed_intime_n",
            "value": timing_row.get("admittime_before_ed_intime_n"),
        },
        {
            "metric": "imv_time_outside_window_n",
            "value": vent_row.get("imv_time_outside_window_n"),
        },
        {
            "metric": "niv_time_outside_window_n",
            "value": vent_row.get("niv_time_outside_window_n"),
        },
        {
            "metric": "poc_itemid_qc_status",
            "value": qa_summary_payload.get("poc_itemid_qc_status"),
        },
        {
            "metric": "poc_itemid_qc_failed_itemids_n",
            "value":
             ↪  qa_summary_payload.get("poc_itemid_qc_failed_itemids_n"),
        },
        {
            "metric": "poc_itemid_qc_warning_itemids_n",
            "value":
             ↪  qa_summary_payload.get("poc_itemid_qc_warning_itemids_n"),
        },
        {
            "metric": "poc_qualifying_earliest_0_24h_hadm_n",
            "value": qa_summary_payload.get(␣
            ↪  "poc_qualifying_earliest_0_24h_hadm_n"),
        },
        {
            "metric": "poc_qualifying_any_type_0_24h_hadm_n",
            "value": qa_summary_payload.get(␣
            ↪  "poc_qualifying_any_type_0_24h_hadm_n"),
        },
        {
            "metric": "poc_itemid_qc_passed_deprecated_alias",
            "value": qa_summary_payload.get("poc_itemid_qc_passed"),
        },
        {
            "metric": "poc_itemid_qc_reason_deprecated_alias",
            "value": qa_summary_payload.get("poc_itemid_qc_reason"),
        },
        {
            "metric": "poc_hypercap_0_24h_deprecated_alias_of",
            "value": qa_summary_payload.get("poc_hypercap_0_24h_alias_of"),
        },
        {
            "metric": "pco2_source_distribution_audit_path",
            "value": blood_gas_audit_paths.get("pco2_source_distribution"),
        },
    ]
    cohort_qc_summary = pd.DataFrame(qc_rows)
else:
    cohort_qc_summary = pd.DataFrame(
        [{"metric": "qa_summary", "value": f"Missing: {qa_summary_path}"}]
    )

display(cohort_qc_summary)
```

	metric	value
0	UNKNOWN semantics	LAB blood-gas unknown specimen only (POC UNKNO...
1	panel_unknown_rate	0.662642
2	encounter_unknown_rate	0.662642
3	POC UNKNOWN quarantined hadm count	None
4	POC UNKNOWN leakage into threshold	None
5	first_gas_without_pco2_anchor_n	0
6	qualifying_gas_time_observed_rate	0.963081
7	timing_usable_for_model_rate	0.998577

	metric	value
8	hospital_los_negative_n	2
9	admittime_before_ed_intime_n	15
10	imv_time_outside_window_n	10
11	niv_time_outside_window_n	12
12	poc_itemid_qc_status	fail
13	poc_itemid_qc_failed_itemids_n	1
14	poc_itemid_qc_warning_itemids_n	2
15	poc_qualifying_earliest_0_24h_hadm_n	858
16	poc_qualifying_any_type_0_24h_hadm_n	906
17	poc_itemid_qc_passed_deprecated_alias	False
18	poc_itemid_qc_reason_deprecated_alias	validated_itemids_failed_qc
19	poc_hypercap_0_24h_deprecated_alias_of	poc_qualifying_any_type_0_24h_hadm_*
20	pco2_source_distribution_audit_path	/Users/blocke/Box Sync/Residency Personal File...

1.6 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 UNKNOWN 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["unknown_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
```

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

ICD Category	Count	Percent	Denominator_N
Other / None	9962	83.4	11945
Acute RF with hypoxia	793	6.6	11945
Obesity hypoventilation syndrome	524	4.4	11945
Acute RF with hypoxia & hypercapnia	386	3.2	11945
Respiratory failure, unspecified	187	1.6	11945
Acute RF with hypercapnia	93	0.8	11945

Inclusion Type	Count	Percent	Denominator_N
Gas_only	9962	83.4	11945
Both	1542	12.9	11945
ICD_only	441	3.7	11945

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	998	50.3	1983
1	VBG threshold positive	1324	66.8	1983
2	PCO2 UNKNOWN threshold positive	330	16.6	1983
3	Any gas threshold positive	1542	77.8	1983

```python
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()
```

	No	Yes	Percent_yes
symptom_group			
Injuries & adverse effects	32	1637	98.1
Symptom – Digestive	29	1373	97.9
Diseases (patient-stated)	16	639	97.6
Symptom – Circulatory	35	1097	96.9
Symptom – Nervous	42	1285	96.8
Uncodable/Unknown	9	264	96.7
Symptom – Musculoskeletal	11	261	96.0
Other	34	768	95.8
Symptom – Skin/Hair/Nails	12	256	95.5
Symptom – General	26	520	95.2

1.7 Symptom Composition By Hypercapnia Definition

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

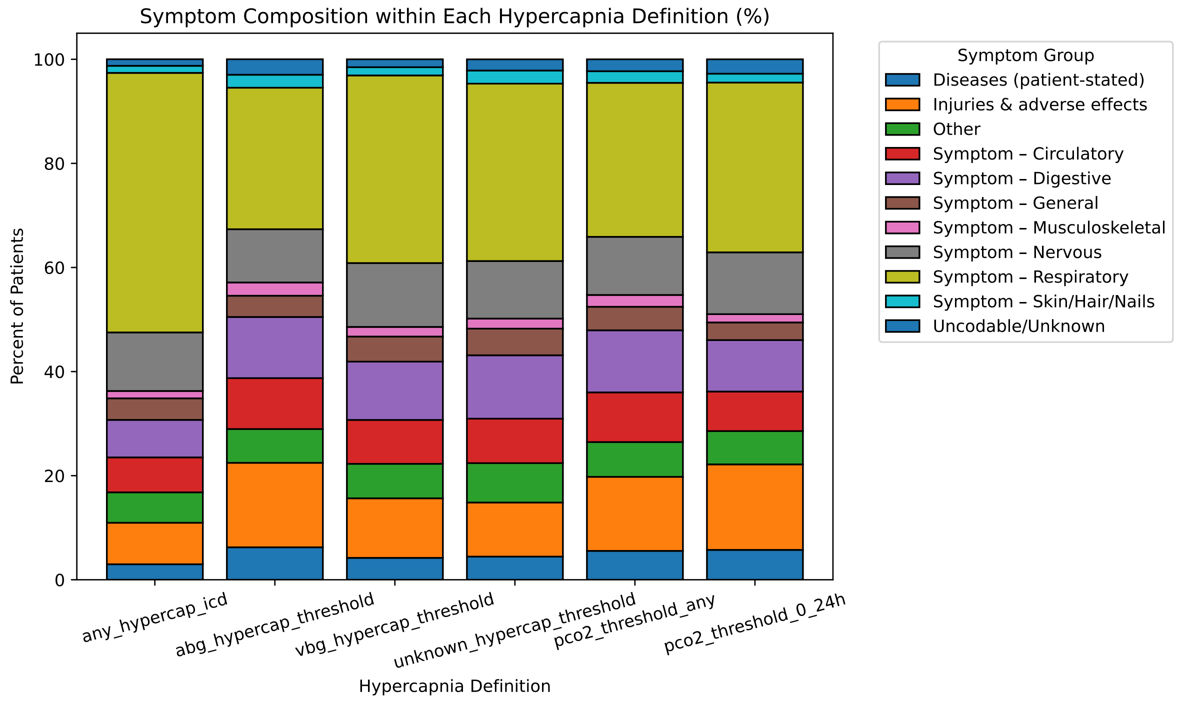
```
definition_long_df = symptom_non_null.melt(
    id_vars=["symptom_group"],
    value_vars=HYPERCAP_CRITERIA,
    var_name="Hypercapnia_Definition",
    value_name="Positive",
)
definition_positive_df =
 ↪  definition_long_df.loc[definition_long_df["Positive"].eq(1)].copy()

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"
```

```python
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
8	abg_hypercap_threshold	Symptom – Respiratory	2027	7456	27.19	26.18	28.20
1	abg_hypercap_threshold	Injuries & adverse effects	1210	7456	16.23	15.39	17.07
4	abg_hypercap_threshold	Symptom – Digestive	876	7456	11.75	11.02	12.48
7	abg_hypercap_threshold	Symptom – Nervous	763	7456	10.23	9.55	10.92
3	abg_hypercap_threshold	Symptom – Circulatory	729	7456	9.78	9.10	10.45
2	abg_hypercap_threshold	Other	484	7456	6.49	5.93	7.05
0	abg_hypercap_threshold	Diseases (patient-stated)	465	7456	6.24	5.69	6.79
5	abg_hypercap_threshold	Symptom – General	304	7456	4.08	3.63	4.53
10	abg_hypercap_threshold	Uncodable/Unknown	222	7456	2.98	2.59	3.36
6	abg_hypercap_threshold	Symptom – Musculoskeletal	190	7456	2.55	2.19	2.91
9	abg_hypercap_threshold	Symptom – Skin/Hair/Nails	186	7456	2.49	2.14	2.85
19	any_hypercap_icd	Symptom – Respiratory	989	1983	49.87	47.67	52.07

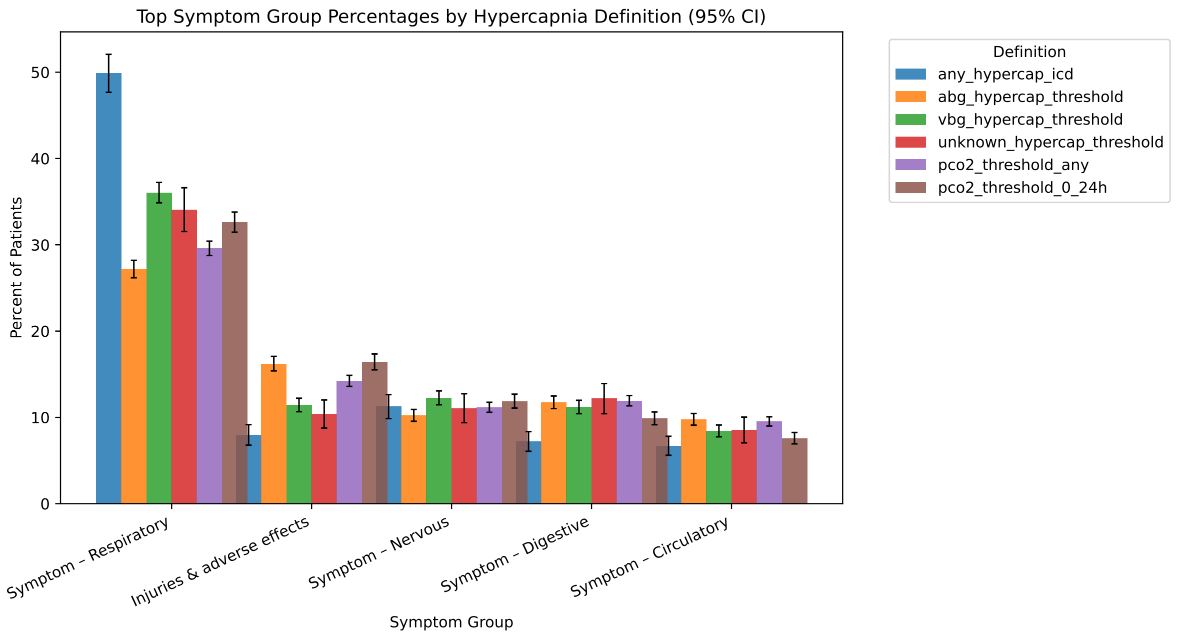
```
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_by_Hypercapnia_Definition.xlsx
Exported: /Users/blocke/Box Sync/Residency Personal Files/Scholarly Work/Locke Research Proje
CC-NLP/Symptom_Composition_Pivot_ChartReady.xlsx
```

```python
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()
```



1.8 Symptom Distribution By Ascertainment Overlap

```
overlap_required = [
    SYMPTOM_COL,
    "abg_hypercap_threshold",
    "vbg_hypercap_threshold",
    "unknown_hypercap_threshold",
    "any_hypercap_icd",
    "pco2_threshold_any",
    "pco2_threshold_0_24h",
]
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["unknown_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/UNKNOWN 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	ABG+VBG	ABG-only	VBG-only
symptom_group			
Diseases (patient-stated)	4.2	7.3	4.2
Injuries & adverse effects	12.7	18.1	10.5
Other	6.1	6.7	7.0
Symptom – Circulatory	7.3	11.1	9.2
Symptom – Digestive	9.9	12.7	12.1
Symptom – General	4.1	4.0	5.3
Symptom – Musculoskeletal	2.0	2.9	1.8
Symptom – Nervous	11.4	9.6	12.8
Symptom – Respiratory	38.3	21.2	34.5
Symptom – Skin/Hair/Nails	1.5	3.0	1.6
Uncodable/Unknown	2.2	3.4	1.0

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

gas_source_overlap symptom_group	ABG+UNKNOWN	ABG+VBG	ABG+VBG+UNKNOWN	ABG-only	U
Diseases (patient-stated)	5.0	4.5	3.5	7.4	5.
Injuries & adverse effects	16.7	13.7	9.5	18.2	10
Other	8.5	5.7	7.7	6.6	7.
Symptom – Circulatory	9.7	7.1	8.1	11.2	7.
Symptom – Digestive	15.9	10.1	9.5	12.5	15
Symptom – General	4.7	4.1	4.2	4.0	5.
Symptom – Musculoskeletal	2.7	1.9	2.1	2.9	1.
Symptom – Nervous	5.4	11.4	11.6	9.8	13
Symptom – Respiratory	22.1	37.8	40.3	21.1	27
Symptom – Skin/Hair/Nails	3.9	1.4	2.1	3.0	3.
Uncodable/Unknown	5.4	2.4	1.6	3.3	1.

```
Symptom distribution by ICD/Gas overlap:
```

icd_gas_overlap symptom_group	Gas-only	ICD+Gas	ICD-only
Diseases (patient-stated)	6.0	2.8	3.6
Injuries & adverse effects	15.2	8.2	7.3
Other	6.9	5.3	7.7
Symptom – Circulatory	10.0	6.4	7.9
Symptom – Digestive	12.6	7.4	6.6
Symptom – General	4.7	3.6	5.9
Symptom – Musculoskeletal	2.4	1.1	2.5
Symptom – Nervous	11.1	11.7	9.5
Symptom – Respiratory	26.2	51.5	44.2
Symptom – Skin/Hair/Nails	2.4	1.0	2.7
Uncodable/Unknown	2.5	1.0	2.0

```
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
```

1.9 ICD Diagnostic Performance (ICD as predictor)

```
performance_targets = [
    ("pco2_threshold_any", "Gas any-time"),
    ("pco2_threshold_0_24h", "Gas within 24h marker"),
    ("abg_hypercap_threshold", "ABG threshold"),
    ("vbg_hypercap_threshold", "VBG threshold"),
    ("unknown_hypercap_threshold", "PCO2 UNKNOWN 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	998	50.3	1983
1	VBG threshold positive	1324	66.8	1983

	Definition	Count	Percent	Denominator_N
2	PCO2 UNKNOWN threshold positive	330	16.6	1983
3	Any gas threshold positive	1542	77.8	1983

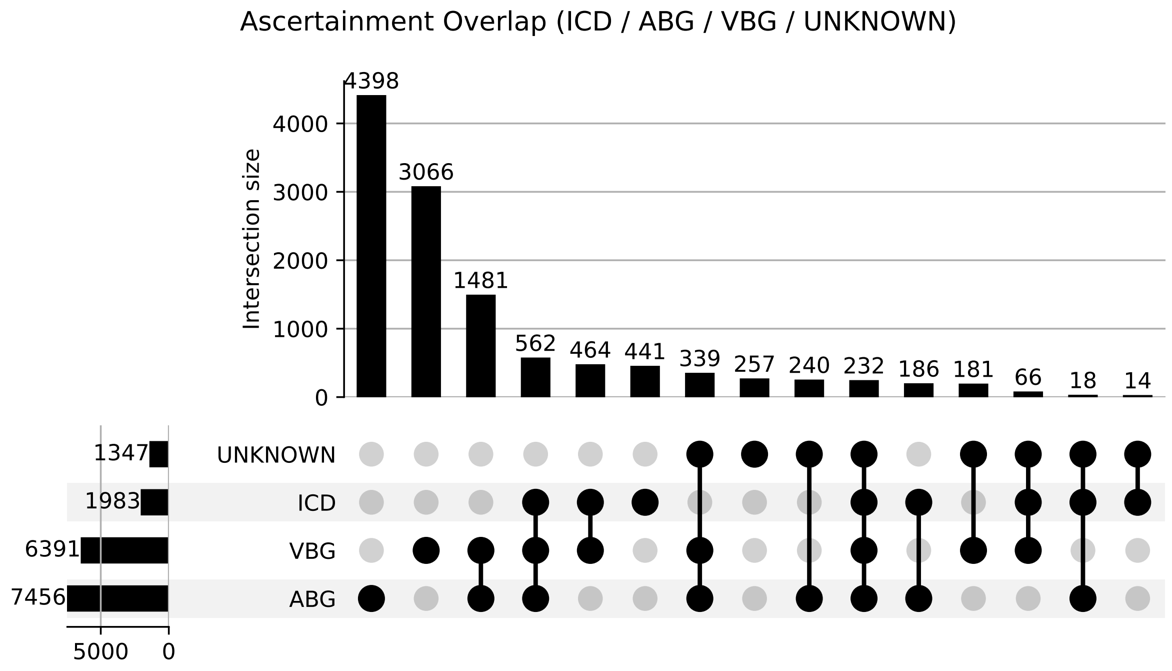
	Target	Target_Column	TP	FP	FN	TN	Sensitivity
0	Gas any-time	pco2_threshold_any	1542	441	9962	0	0.134040
1	Gas within 24h marker	pco2_threshold_0_24h	1074	909	5146	4816	0.172669
2	ABG threshold	abg_hypercap_threshold	998	985	6458	3504	0.133852
3	VBG threshold	vbg_hypercap_threshold	1324	659	5067	4895	0.207166
4	PCO2 UNKNOWN threshold	unknown_hypercap_threshold	330	1653	1017	8945	0.244989

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

1.10 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),
        "UNKNOWN":
         ↪  to_binary_flag(df["unknown_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 / UNKNOWN)")
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", "UNKNOWN"],
 ↪  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	UNKNOWN	Count
0	False	True	False	False	4398
1	False	False	True	False	3066
2	False	True	True	False	1481
3	True	True	True	False	562
4	True	False	True	False	464
5	True	False	False	False	441
6	False	True	True	True	339
7	False	False	False	True	257
8	False	True	False	True	240
9	True	True	True	True	232
10	True	True	False	False	186
11	False	False	True	True	181
12	True	False	True	True	66
13	True	True	False	True	18
14	True	False	False	True	14

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

1.11 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=11,945).} \label{tab:pre
\toprule
Definition & Column & Count & Denominator_N & Percent \\
\midrule
\endfirsthead
\caption[]{Hypercapnia prevalence summary (denominator = full cohort N=11,945).} \\
\toprule
Definition & Column & Count & Denominator_N & Percent \\
\midrule
\endhead
\midrule
\multicolumn{5}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
PCO2 threshold any-time & pco2_threshold_any & 11504 & 11945 & 96.300000 \\
ABG hypercapnia threshold & abg_hypercap_threshold & 7456 & 11945 & 62.400000 \\
VBG hypercapnia threshold & vbg_hypercap_threshold & 6391 & 11945 & 53.500000 \\
PCO2 threshold within 24h marker & pco2_threshold_0_24h & 6220 & 11945 & 52.100000 \\
Hypercapnic RF ICD (any) & any_hypercap_icd & 1983 & 11945 & 16.600000 \\
PCO2 UNKNOWN threshold & unknown_hypercap_threshold & 1347 & 11945 & 11.300000 \\
\end{longtable}

\begin{longtable}{lrrr}
\caption{ICD category composition (denominator = full cohort N=11,945).} \label{tab:icd_categ
\toprule
ICD Category & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{ICD category composition (denominator = full cohort N=11,945).} \\
\toprule
ICD Category & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
Other / None & 9962 & 83.400000 & 11945 \\
Acute RF with hypoxia & 793 & 6.600000 & 11945 \\
Obesity hypoventilation syndrome & 524 & 4.400000 & 11945 \\
Acute RF with hypoxia & hypercapnia & 386 & 3.200000 & 11945 \\
Respiratory failure, unspecified & 187 & 1.600000 & 11945 \\
Acute RF with hypercapnia & 93 & 0.800000 & 11945 \\
\end{longtable}

\begin{longtable}{lrrr}
\caption{Inclusion source composition (denominator = full cohort N=11,945).} \label{tab:inclu
\toprule
Inclusion Type & Count & Percent & Denominator_N \\
\midrule
\endfirsthead
\caption[]{Inclusion source composition (denominator = full cohort N=11,945).} \\
\toprule
Inclusion Type & Count & Percent & Denominator_N \\
\midrule
\endhead
\midrule
\multicolumn{4}{r}{Continued on next page} \\
\midrule
\endfoot
\bottomrule
\endlastfoot
Gas_only & 9962 & 83.400000 & 11945 \\
Both & 1542 & 12.900000 & 11945 \\
ICD_only & 441 & 3.700000 & 11945 \\
\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 & 998 & 50.300000 & 1983 \\
VBG threshold positive & 1324 & 66.800000 & 1983 \\
PCO2 UNKNOWN threshold positive & 330 & 16.600000 & 1983 \\
Any gas threshold positive & 1542 & 77.800000 & 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}{llrrrrrrrrr}
\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% C
\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-time & pco2_threshold_any & 1542 & 441 & 9962 & 0 & 0.134040 & 0.127937 & 0.140388 &
Gas within 24h marker & pco2_threshold_0_24h & 1074 & 909 & 5146 & 4816 & 0.172669 & 0.16347
ABG threshold & abg_hypercap_threshold & 998 & 985 & 6458 & 3504 & 0.133852 & 0.126312 & 0.1
VBG threshold & vbg_hypercap_threshold & 1324 & 659 & 5067 & 4895 & 0.207166 & 0.197408 & 0.
PCO2 UNKNOWN threshold & unknown_hypercap_threshold & 330 & 1653 & 1017 & 8945 & 0.244989 &
\end{longtable}
\n\end{landscape}\n
```

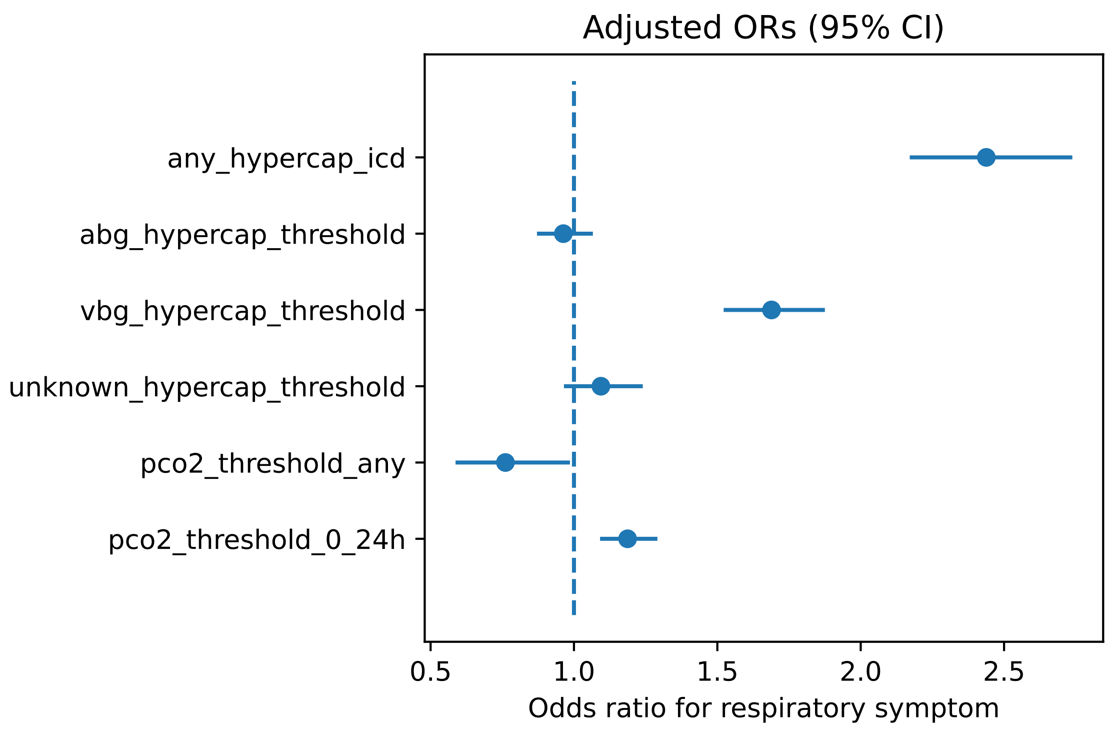
1.12 Association Model

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)

predictor_df = model_df[HYPERCAP_CRITERIA].copy()
non_constant_predictors = [
    column for column in HYPERCAP_CRITERIA if
↪   predictor_df[column].nunique(dropna=True) > 1
]
outcome = model_df["is_respiratory"]
model_status = "ok"
model_message = ""

if not non_constant_predictors:
    model_status = "skipped"
    model_message = "Association model skipped: all hypercapnia predictors
↪   are constant in this run."
    print(model_message)
    or_table = pd.DataFrame(index=HYPERCAP_CRITERIA, columns=["OR", "CI_lo",
↪   "CI_hi", "p"])
else:
    design_matrix = sm.add_constant(
        predictor_df[non_constant_predictors], has_constant="add"
    )
    try:
        logit_result = sm.Logit(outcome, design_matrix,
↪   missing="drop").fit(disp=False)
        fitted_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,
            }
        )
        or_table = (
            fitted_table.reindex(HYPERCAP_CRITERIA)
            .astype(float, copy=False)
            .round(3)
        )
    except Exception as exc:
        model_status = "skipped"
        model_message = (
            "Association model skipped due to singular/ill-conditioned design
             ↪  matrix: "
            f"{exc}"
        )
        print(model_message)
        or_table = pd.DataFrame(index=HYPERCAP_CRITERIA, columns=["OR",
↪   "CI_lo", "CI_hi", "p"])

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

	OR	CI_lo	CI_hi	p
any_hypercap_icd	2.438	2.171	2.738	0.000
abg_hypercap_threshold	0.963	0.871	1.066	0.469
vbg_hypercap_threshold	1.689	1.522	1.875	0.000
unknown_hypercap_threshold	1.094	0.965	1.240	0.161
pco2_threshold_any	0.761	0.587	0.986	0.039
pco2_threshold_0_24h	1.187	1.091	1.291	0.000

```
or_plot_df = or_table.loc[HYPERCAP_CRITERIA].copy()
or_plot_df = or_plot_df.apply(pd.to_numeric, errors="coerce")

if or_plot_df["OR"].notna().any():
    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()
else:
    print("Adjusted OR plot skipped because association model was not
     ↪  estimable in this run.")
```



1.13 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	7805
1	/Users/blocke/Box Sync/Residency Personal File...	True	5635
2	/Users/blocke/Box Sync/Residency Personal File...	True	5372
3	/Users/blocke/Box Sync/Residency Personal File...	True	5622
4	/Users/blocke/Box Sync/Residency Personal File...	True	5377
5	/Users/blocke/Box Sync/Residency Personal File...	True	5856
6	/Users/blocke/Box Sync/Residency Personal File...	True	5723
7	/Users/blocke/Box Sync/Residency Personal File...	True	151781
8	/Users/blocke/Box Sync/Residency Personal File...	True	5285

```
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 Project
CC-NLP/MIMIC tabular data/prior runs/2026-02-26 analysis_run_manifest.json
```