A5 Statistical Analyses of Clinical Datasets

Welcome to A5! Please enter answers to the questions in the specified Markdown cells below, and complete the code snippets in the associated Python files as specified. When you are done with the assignment, follow the instructions at the end of this assignment to submit.

Learning Objective 🌱

In this assignment, you will work with a dataset that we have prepared for you using a process similar to what you did in A3 and A4. The dataset describes patients from the MIMIC III database who were put on mechanical ventilation and were stable for 12 hours. Some of these patients then experienced a sudden and sustained drop in oxygenation, while others did not. You will practice using common time-saving tools in the **Pandas** 🐼 library, the **NumPy** 🔢 library, the **sklearn** 🔬 library, and **Python** 🐍 programming language that are ideally suited to these tasks.

Resources 📖

- Pandas Cheat Sheet 🐼 : https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf

Environment Set-Up 🐍

To begin, we will need to set up an virtual environment with the necessary packages. A virtual environment is a self-contained directory that contains a Python interpreter (aka Python installation) and any additional packages/modules that are required for a specific project. It allows you to isolate your project's dependencies from other projects that may have different versions or requirements of the same packages.

In this course, we require that you utilize Miniconda to manage your virtual environments. Miniconda is a lightweight version of Anaconda, a popular Python distribution that comes with many of the packages that are commonly used in data science.

Instructions for setting up your environment using Miniconda:

1. If you do not already have Miniconda installed, download and install the latest version for your opperating system from the following link: https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links

2. Create a new virtual environment for this assignment by running the following command in your terminal:

```
conda env create -f environment.yml
```
This will create a new virtual environment called `biomedin215`

3. Activate your new virtual environment by running the following command in your terminal:

   ```
   conda activate biomedin215
   ```
   This will activate the virtual environment you created in the previous step.

4. Finally, ensure that your `ipynb` (this notebook)'s kernel is set to utilize the `biomedin215` virtual environment you created in the previous steps. Depending on which IDE you are using to run this notebook, the steps to do this may vary.

In [309…

```python
# Run this cell:
# The lines below will instruct jupyter to reload imported modules before
# executing code cells. This enables you to quickly iterate and test revisio
# to your code without having to restart the kernel and reload all of your
# modules each time you make a code change in a separate python file.

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

In [310…

```python
# Run this cell to ensure the environment is setup properly
import pandas as pd
import numpy as np
import os
import warnings

print("Sanity check: Success")
```

```
Sanity check: Success
```

Note to Students: 📚

> Throughout the assignment, we have provided `sanity checks`: small warnings that will alert you when your implementation is different from the solution. Our goal in providing these numbers is to help you find bugs or errors in your code that may otherwise have gone unnoticed. Please note: the sanity checks are just tools we provided to be helpful, and should not be treated as a target to hit. We manually grade each assignment based on the code you submit, and not based on whether you get the exact same numbers as the sanity checks.

If you are failing the sanity checks (even by a lot) and your implementation is correct with minor errors, you will still receive the majority of the points (if not all the points).

In [311…

```python
# Run this cell to set up sanity checks warnings
# Note: You do not need to change anything in this cell
```

```
# Creates a custom warning class for sanity checks
class SanityCheck(Warning):
    pass

# Sets up a cosutom warning formatter
def custom_format_warning(message, category, filename, lineno, line=None):
    if category == SanityCheck:
        # Creates a custom warning with orange text
        return f'\033[38;5;208mSanity Check - Difference Flagged:\n{message}

    return '{}:{}: {}: {}\n'.format(filename, lineno, category.__name__, mes

# Sets the warning formatter for the entire notebook
warnings.formatwarning = custom_format_warning
```

Data Description 📂

We will be utilizing the same subset of the MIMIC III database we utilized starting in A2: the 1,000 subject development cohort you created previously. You will start with a dataset very similar to what you may have generated at the end of the prior assignment.

You will analyze the available data to identify a cohort of patients that underwent septic shock during their admission to the ICU. **All of the data you need for this assignment is available on Canvas.**

Once you have downloaded and unzipped the data, you should see the following `3` csv files:

- `patient_feature_matrix.csv`
- `cohort.csv`
- `feature_descriptions.csv`

Specify the location of the folder containing the data in the following cells:

In [312…
```
# Specify the path to the folder containing the data files
data_dir = "/Users/nickallen/Documents/GitHub/BMDS215/A5/data"
```

In [313…
```
# Run this cell to make sure all of the files are in the specified folder
expected_file_list = ["patient_feature_matrix.csv", "cohort.csv", "feature_c

for file in expected_file_list:
    assert os.path.exists(os.path.join(data_dir, file)), "Can't find file {}

print("All files found!")
```

```
All files found!
```

In [314…
```
# Run this cell to load the data from the CSV files into Pandas DataFrames
patient_feature_matrix = pd.read_csv(os.path.join(data_dir, "patient_feature
cohort = pd.read_csv(os.path.join(data_dir, "cohort.csv"))
feature_descriptions = pd.read_csv(os.path.join(data_dir, "feature_descripti
```

`Important note to students:`

You may notice that some of the various features appear to have multiple versions (ex. `icd9_AORTIC_DISSECTION_A` and `icd9_AORTIC_DISSECTION_B`). For the purposes of this assignment, we have done some simplification in preprocessing the data to make it easier for you to work with. In the future, it is important to keep in mind that healthcare datasets are notoriusly messy and often require a lot of preprocessing before they are ready to be used for analysis.

In [315…
```
# Run this cell to see the first three rows of the patient feature matrix
print("Patient Feature Matrix")
display(patient_feature_matrix.head(3))
```

```
Patient Feature Matrix
```

	subject_id	death_in_stay	oxy_drop	gender	age_in_days	icd9_ADENOID_CYSTIC_
0	91	died	stable	F	29809.000000	
1	106	died	oxy_drop	M	10358.333333	
2	111	survived	oxy_drop	F	24388.979167	

3 rows × 2440 columns

In [316…
```
# Run this cell to see the first three rows of the cohort DataFrame
print("Cohort")
display(cohort.head(3))
```

```
Cohort
```

	subject_id	icustay_id	death_in_stay	deathtime	censor_time	index_time	exposure
0	91	256972	died	2177-05-10 15:16:00	2177-05-10 15:16:00	2177-05-08 00:00:00	
1	106	252051	died	2192-08-15 21:15:00	2192-08-15 21:15:00	2192-08-10 08:00:00	2192- 16:
2	111	254245	survived	NaN	2142-05-05 11:45:00	2142-04-24 23:30:00	2142- 18:

In [317…
```
# Run this cell to see the first three rows of the feature descriptions Data
print("Feature Descriptions")
display(feature_descriptions.head(3))
```

```
Feature Descriptions
```

	feature	feature_type	code	description
0	oxy_drop	engineered	NaN	sustained drop in oxygenation after stable ven...
1	gender	demographic	NaN	gender
2	age_in_days	demographic	NaN	age in days

`1` (`5 points`) Preprocessing

`1.1`: (`1 points`) Create the Feature Matrix and Outcome Vector

As you may have noticed above, the `death_in_stay` labels and features are all included together inside of the `patient_feature_matrix.csv` file. Let's split the patient_feature_matrix up into two numpy arrays.

Implement the `split_labels_and_features` function in `preprocessing.py`. When you are finished, run the following cell.

In [318…

```
from src.preprocessing import split_labels_and_features

# Run this cell to split the labels from the features
features, labels = split_labels_and_features(patient_feature_matrix,
                                              {"oxy_drop": {"stable": 0, "oxy
                                               "death_in_stay": {"survived":
                                               "gender": {"M":0, "F": 1}})

# Sanity Check shapes
if features.shape[0] != labels.shape[0]:
    warnings.warn("Number of rows in features and labels don't match", Sanit

if features.shape[1] != patient_feature_matrix.shape[1] - 2:
    warnings.warn("Number of columns in features is incorrect", SanityCheck)

if labels.shape[1] != 1:
    warnings.warn("Number of columns in labels is incorrect", SanityCheck)

# Sanity Check indices
if not features.index.name == "subject_id":
    warnings.warn("Index name is incorrect", SanityCheck)

if not labels.index.name == "subject_id":
    warnings.warn("Index name is incorrect", SanityCheck)
```

`Important note to students:`

It's worth noting that the definition of `gender` in the `MIMIC-III` database and many other clinical databases is a simplified representation of gender that does not account for a range of gender identities and expressions, and most often isn't distinguished from biological sex which may or may not reflect an individual's self-identified gender identity (SIGI). Increasingly, EHR schemas are updated/designed to capture SIGI in addition to biological sex to improve care for transgender and gender nonconforming (TGNC) patients.

As you continue your journey in medical data science, remember that it is critically important to always review the schema of the dataset you are working with to ensure that you know how different data fields are defined, so that you can fully understand the assumptions and limitations of your downstream analysis.

`1.2` (`4 points`) Removing Uninformative Features

Before we do any modeling, let's cut down on our feature space by removing `low-variance features` that probably aren't useful enough to measure association with or use in a predictive model.

Implement the function `feature_variance_threshold` in `preprocessing.py`. When you are finished, run the following cell.

In [319…
```python
from src.preprocessing import feature_variance_threshold

# Run this cell to remove features with low variance
filtered_features = feature_variance_threshold(features, freq_cut=95/5, unic
```

Report how many of each different kind of feature are left after filtering out the near-zero variance features. As a sanity check, look at the kinds of features that are over-represented or under-represented in this set relative to the full set of features. Provide a brief plausible explanation of what you observe.

In [320…
```python
import pandas as pd

# Build feature → type mapping
desc = pd.read_csv("data/feature_descriptions.csv", usecols=["feature", "fea
feature_type_map = dict(zip(desc["feature"], desc["feature_type"]))

# Helper to get a Series of types for any column list
def column_types(columns):
    return pd.Series({c: feature_type_map.get(c, "unknown") for c in columns

# Types for downstream use
all_feature_types = column_types(features.columns)
filtered_feature_types = column_types(filtered_features.columns)

# Optional summary
```

```
summary = (
    pd.concat(
        [all_feature_types.value_counts().rename("all"),
         filtered_feature_types.value_counts().rename("filtered")],
        axis=1
    ).fillna(0).astype(int)
)
summary["retained_pct"] = (summary["filtered"] / summary["all"]).replace([pc
display(summary.sort_values("filtered", ascending=False))
```

	all	filtered	retained_pct
chartindicator	1345	335	0.249
note CUI	550	122	0.222
chartvalue	51	48	0.941
demographic	2	2	1.000
engineered	1	1	1.000
icd9	489	0	0.000

After mapping each column to its feature type, we see clear variance-driven patterns. Chartindicators and CUIs retain relatively few columns (retained_pct ≈ 0.25 and 0.22): they're mostly sparse binary flags, so the most_common/second_most_common ratio exceeds the freq_cut and they fail the unique ratio check (≈2/rows). Chartvalues are largely kept (≈0.94) because they're continuous with many distinct values, easily passing the unique_cut. Demographics and the engineered feature are fully retained (1.00): age is continuous, and gender/oxy_drop are not extremely imbalanced in this cohort. ICD9 codes have 0 retention, reflecting extreme sparsity at the individual-code level; most columns are nearly all zeros (or even single-valued), so they're removed by the frequency/uniqueness rules.

`2` (`15 points`) Performing an observational study

Associative analysis revolves around the concept of finding associations or patterns between features and outcomes. In biomedical data science, this approach can uncover hidden relationships between different medical conditions, drugs, genetic variations, or other biomedical entities, which can prove invaluable in research and clinical decision-making.

It's important to keep in mind that data science can help us find patterns, but we must always approach our results critically, as considering the broader clinical and biological context is essential to understanding the significance of our findings. Let's try to find some interesting associations in our dataset!

`2.1` (`0 points`) Learning how to Choose the Right Statistical Test

Hypothesis testing is a cornerstone of statistical inference, allowing researchers to determine if there's enough evidence in a sample of data to infer that a particular condition holds for a larger population. In biomedical data science, this method is essential for validating the significance of findings, be it in clinical trials, genomic studies, or other investigations. If you are unfamiliar with hypothesis testing, we recommend that you review the following video: StatsQuest - Hypothesis Testing

Types of Hypothesis Tests

Tests for one random variable

- Continuous **One-sample t-test:**: If you want to test if the mean of a single sample is different from a specified value (usually a known mean).
- Discrete **Chi-squared (χ^2) test:** Tests if the proportion of categorical outcomes is equal to a specified value.

Tests for two random variables

- One continuous variable vs one discrete variable with 2 levels *
    - **For independent samples: Two-sample t-test:** *aka independent t-test*, used to determine whether there is a statistically significant difference between the means of two independent groups.
    - **For independent but small samples: Wilcoxon-Mann-Whitney test:** used to determine whether two independent samples come from the same distribution. (non-parametric alternative to the two-sample t-test)
    - **For paired samples: Paired t-test:** *aka dependent t-test*, used to determine if there is a statistically significant mean difference between two sets of paired or related observations.
    - **For paired but small samples: Wilcoxon signed-rank test:** used to determine whether two dependent samples come from the same distribution. (non-parametric alternative to the paired t-test)
- One continuous variable vs one discrete variable with > 2 levels *
    - **One-way ANOVA**: used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups.

- One discrete variable vs one discrete variable
    - **For r-level vs c-level: r x c contingency table Chi-squared test for independence:** used to determine if there's an association or relationship between two categorical variables where one variable has r levels and the other has c levels.
    - **For 2-level vs 2-level but small counts: Fisher's exact test:** similar to the Chi-squared test for a 2x2 contingency table, but used when the sample sizes are small (usually when the expected count/frequency in any cell of the table is less than 5)
    - **For paired samples - McNemar's test:** used for 2x2 contingency tables with paired or matched data. A classic example is when patients are observed before and after a treatment, and the response is dichotomous (e.g., "improved" or "not improved").
- One continuous variable vs one continuous variable:
    - **Correlation/simple linear regression:** used to determine if there is a relationship between two continuous variables.

Tests for more than two random variables

- One discrete vs one discrete vs one continuous:
    - **Two-way ANOVA:** used to determine whether there are any statistically significant differences between the means of three or more independent (unrelated) groups that have been split on two factors (i.e., independent variables).
- One continuous vs many - **multiple linear regression**
- One discrete vs many - **logistic regression**

* Note: In statistics, the term `level` when referring to a discrete variable typically means a distinct value or category that the variable can assume.

> Credit: The above write up was adapted for this assignment from Lexin Li's 2016 Big Data course at the University of California Berkeley.

`2.2` (`10 points`) Statistical Tests of Differences Between Two Groups

For the each of the following features, use a `t-test`, `rank-sum test`, `Fisher exact test`, or a `Chi-squared test` (whichever is appropriate) to determine if there is a statistically significant association between the feature and the `death_in_stay` outcome. Write your reasoning for determining which kind of test to use. If multiple tests are applicable to a comparison, choose one and explain why you chose it.

> NOTE: Make sure you read the documentation for the test functions. Statistical testing functions can be counter intuitive to use, and may produce and incorrect result if used incorrectly.

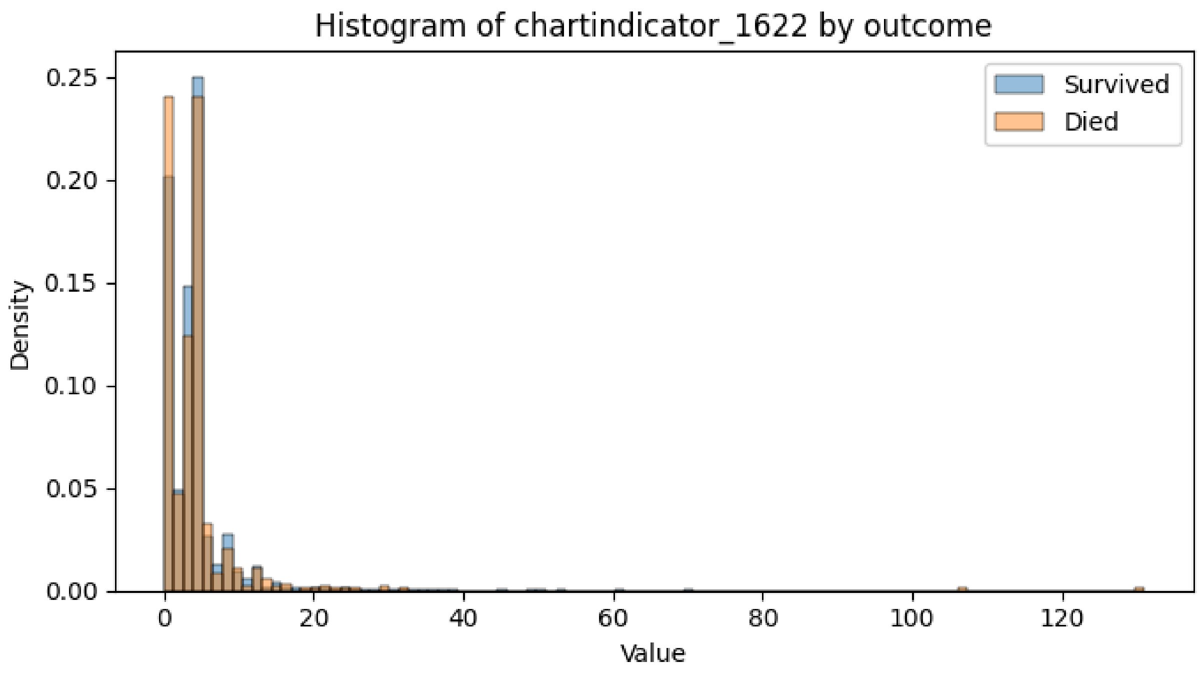
> Hint: It may be helpful to first create two new dataframes, one of patients who died in the ICU and one of patients who did not die in the ICU. Remember that you can add additional code cells to this notebook if needed.

In [321…
```
# Run this cell to import the test functions.
# Utilize one of the following imported functions in your answers below:
from scipy.stats import ttest_ind, fisher_exact, chi2_contingency, mannwhitr

# NOTE: for Rank Sum Tests use mannwhitneyu

# NOTE: For some of the functions above, you can pass series of labels and f
# require you to pass the tables of counts. You may find the function pd.crc
# certain functions.
```

In [322…
```
import matplotlib.pyplot as plt
import seaborn as sns
def plot_hist_for_dead_v_live(col):
    x_dead = filtered_features.loc[labels["death_in_stay"]==1, col].dropna()
    x_alive = filtered_features.loc[labels["death_in_stay"]==0, col].dropna(

    # common bins across both groups
    bins = np.histogram_bin_edges(np.concatenate([x_dead.values, x_alive.val

    plt.figure(figsize=(7,4))
    sns.histplot(x=x_alive, bins=bins, color="C0", alpha=0.45, stat="density
    sns.histplot(x=x_dead,  bins=bins, color="C1", alpha=0.45, stat="density
    plt.title(f"Histogram of {col} by outcome")
    plt.xlabel("Value")
    plt.ylabel("Density")
    plt.legend()
    plt.tight_layout()
    plt.show()

    print("unique values",sorted(list(filtered_features[col].unique())))
```

`alarms` (chartindicator_1622)

```
In [323…  plot_hist_for_dead_v_live("chartindicator_1622")
```



```
unique values [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37,
38, 39, 45, 49, 50, 53, 61, 70, 107, 131]
```

```
In [324…  x = filtered_features.loc[labels["death_in_stay"]==1, "chartindicator_1622"]
          y = filtered_features.loc[labels["death_in_stay"]==0, "chartindicator_1622"]

          stat, p = mannwhitneyu(x, y, alternative="two-sided", method="asymptotic")
          stat, p
```

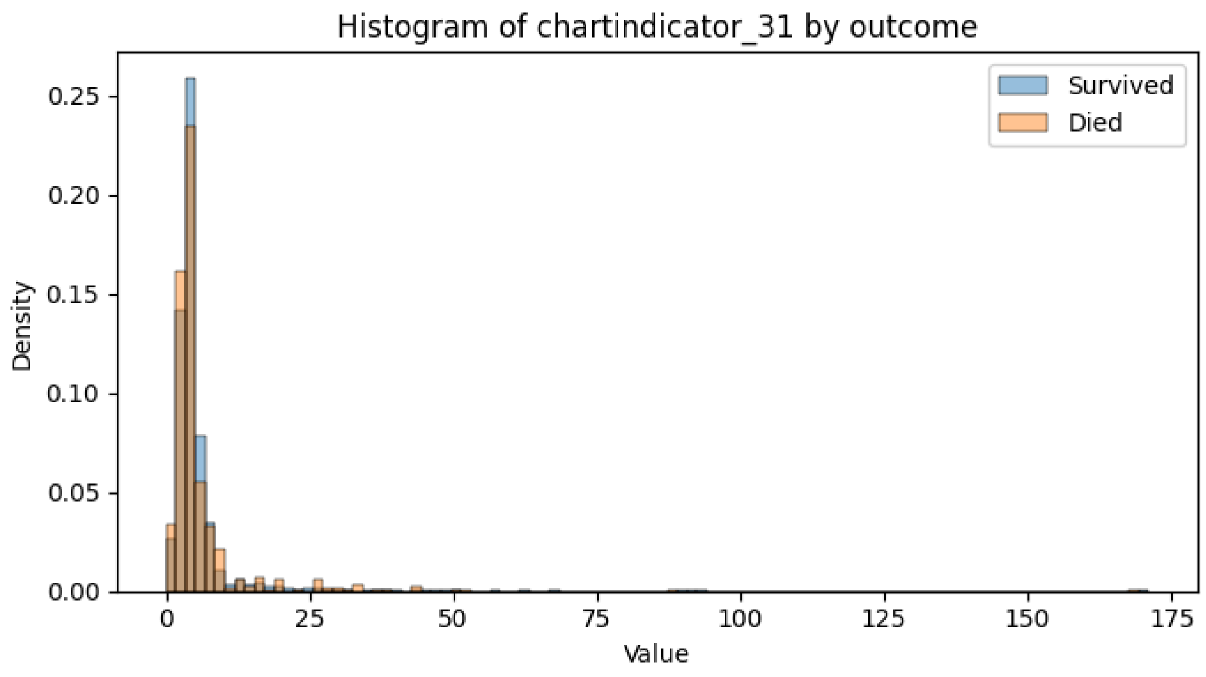
```
Out[324…  (849865.5, 0.07161093078060332)
```

Which test did you utilize for the `alarms` feature? Why?

The Wilcoxon–Mann–Whitney test. We have independent samples. The outcome is a skewed count - many discrete levels, and we are comparing distributions between two independent groups (died vs survived). As such a non-parametric two-group test is appropriate. t-test wouldn't have been the most appropriate because of the skew in the data and various number of ties. t-test focuses on the means. Due to the nature of our data, we could have far different distributions but similar means. We don't want smoothing of our data to mask out this finding.

`activity` (chartindicator_31)

```
In [325…  plot_hist_for_dead_v_live("chartindicator_31")
```



```
unique values [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,
18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 39, 40, 43, 44, 45, 47, 48, 50, 51, 52, 57, 58, 62, 68, 88, 89, 92, 94,
168, 171]
```

In [326…

```python
col = "chartindicator_31"
x = filtered_features.loc[labels["death_in_stay"]==1, col].dropna()
y = filtered_features.loc[labels["death_in_stay"]==0, col].dropna()

u, p = mannwhitneyu(x, y, alternative="two-sided", method="asymptotic")
n1, n0 = len(x), len(y)
cliffs_delta = (2*u)/(n1*n0) - 1
print(f"U={u:.0f}, p={p:.3g}, Cliff's delta={cliffs_delta:.3f}")
```

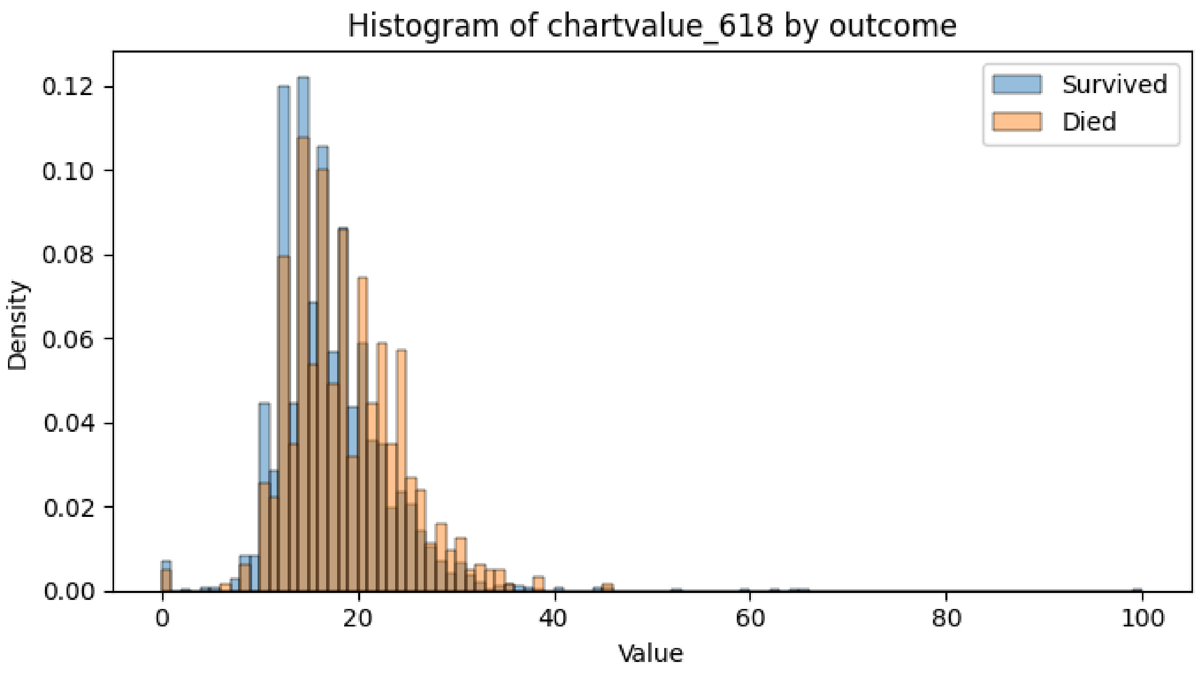
```
U=860330, p=0.185, Cliff's delta=-0.033
```

Which test did you utilize for the `activity` feature? Why?

The chartindicator feature is discrete count with many ties and a skewed, heavy-tailed distribution; we're comparing two independent groups (died vs survived). A parametric two-sample t-test targets mean differences under approximate normality and similar shapes, which these data violate and which can make the mean unstable and sensitive to outliers. The Wilcoxon–Mann–Whitney test instead compares group distributions via ranks (often interpreted as a median/location shift), requires only an ordinal scale and independence, is robust to skew/outliers, tolerates unequal variances, and has proper tie corrections. Therefore it's the appropriate primary test here; we can report U and p, plus a rank-based effect size (e.g., Cliff's delta), with Welch's t as a sensitivity check if desired.

`respiratory rate` (chartvalue_618)

In [327…
```
plot_hist_for_dead_v_live("chartvalue_618")
```



```
unique values [0.0, 2.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.
0, 14.0, 15.0, 16.0, 17.0, 17.0153712296984, 18.0, 19.0, 20.0, 21.0, 22.0, 2
3.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0,
36.0, 37.0, 38.0, 40.0, 44.0, 45.0, 52.0, 59.0, 62.0, 64.0, 65.0, 100.0]
```

In [328…
```
col = "chartvalue_618"

# split by label
x = filtered_features.loc[labels["death_in_stay"]==1, col].dropna().astype(f
y = filtered_features.loc[labels["death_in_stay"]==0, col].dropna().astype(f

# Welch's two-sample t-test
t_stat, t_p = ttest_ind(x, y, equal_var=False)

print(f"Welch t-test: t={t_stat:.3f}, p={t_p:.3g}")
```

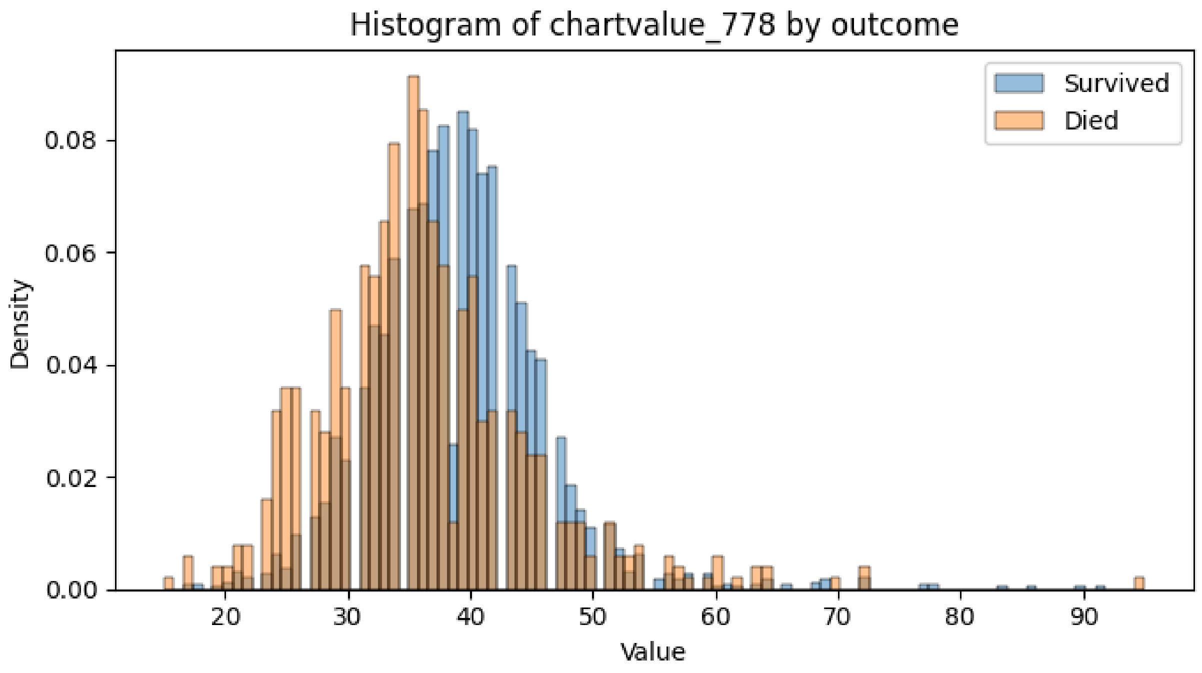
```
Welch t-test: t=6.690, p=3.83e-11
```

Which test did you utilize for the `respiratory rate` feature? Why?

Here, respiratory rate (chartvalue_618) is continuous with many unique values and only mild skew; groups (died vs survived) are independent. The appropriate choice is the two-sample t-test for independent groups. We use Welch's version to allow unequal variances. It directly tests mean difference and has good power for roughly symmetric, continuous data.

`Arterial PaCO2` (chartvalue_778)

In [329…
```
plot_hist_for_dead_v_live("chartvalue_778")
```



```
unique values [15.0, 17.0, 18.0, 19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 2
6.0, 27.0, 28.0, 29.0, 30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0,
38.4270126806252, 39.0, 40.0, 41.0, 42.0, 43.0, 44.0, 45.0, 46.0, 47.0, 48.
0, 49.0, 50.0, 51.0, 52.0, 53.0, 54.0, 55.0, 56.0, 57.0, 58.0, 59.0, 60.0, 6
1.0, 62.0, 63.0, 64.0, 66.0, 68.0, 69.0, 70.0, 72.0, 77.0, 78.0, 83.0, 86.0,
90.0, 91.0, 95.0]
```

In [330…

```python
col = "chartvalue_778"

x = filtered_features.loc[labels["death_in_stay"]==1, col].dropna().astype(f
y = filtered_features.loc[labels["death_in_stay"]==0, col].dropna().astype(f

# Welch's two-sample t-test (primary)
t_stat, t_p = ttest_ind(x, y, equal_var=False)
```

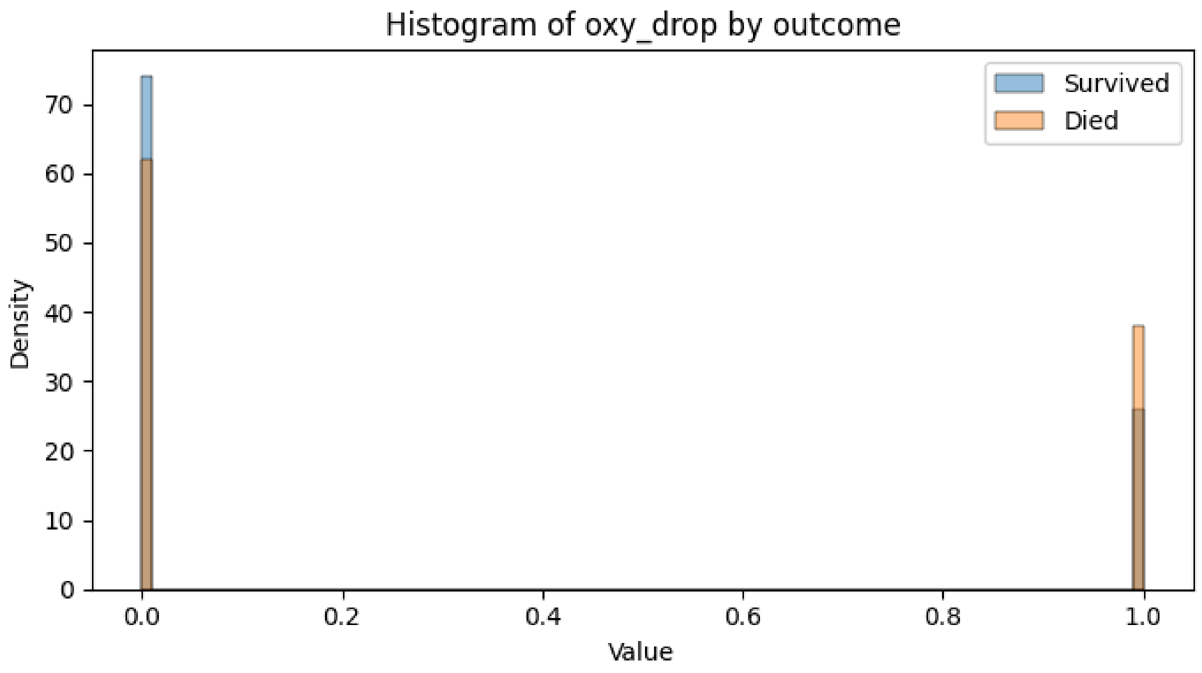
Which test did you utilize for the `arterial PaCO2` feature? Why?

I'll use the two-sample t-test with Welch's correction. The variable is continuous with many unique values and only mild skew, and the two groups (died vs survived) are independent. Welch's t directly tests a mean difference and is robust to unequal variances.

`Oxygen Desaturation` (oxy_drop)

In [331…

```python
plot_hist_for_dead_v_live("oxy_drop")
```



```
unique values [0, 1]
```

In [332…

```python
tbl = pd.crosstab(labels["death_in_stay"].astype(int),
                  filtered_features["oxy_drop"].astype(int))
chi2, p, dof, _ = chi2_contingency(tbl.values, correction=False)
print(tbl)
print(f"Chi-squared p-value = {p:.3g}")
```

```
oxy_drop          0    1
death_in_stay
0              2092  733
1               391  239
Chi-squared p-value = 1.43e-09
```

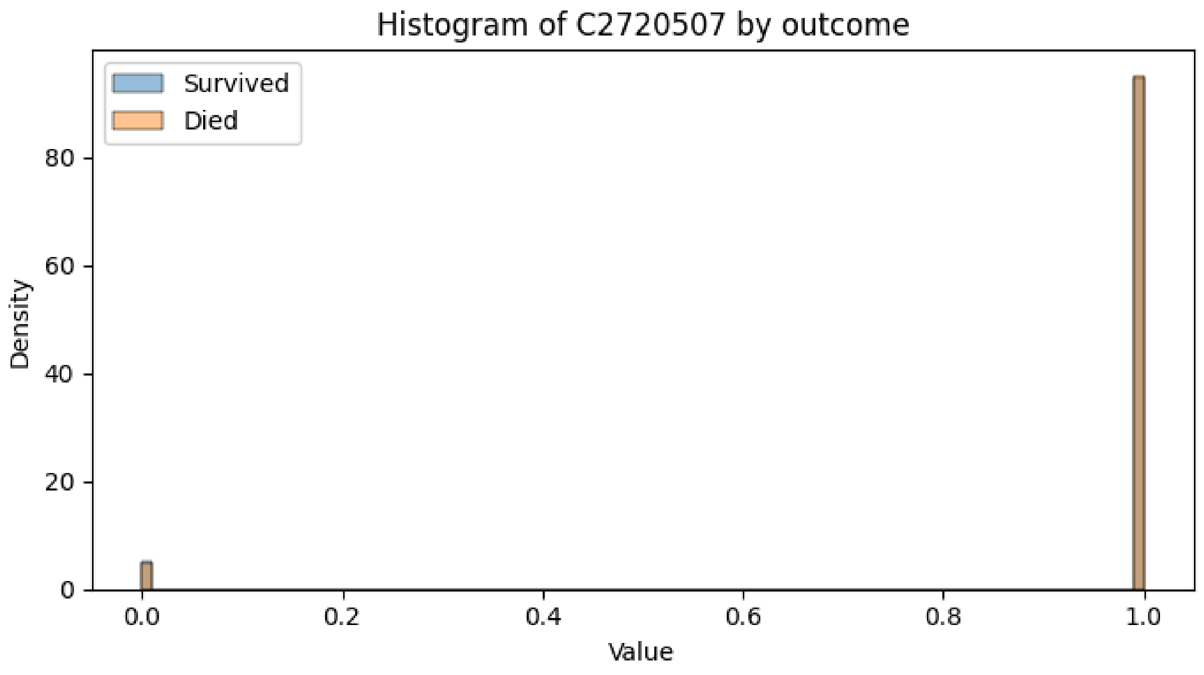
Which test did you utilize for the `oxy_drop` feature? Why? Was the association statistically significant?

For oxy_drop, both the predictor and outcome are binary (0/1: drop vs no drop; survived vs died), the observations are independent at the patient level, and my scientific question is whether the proportion with an oxygen drop differs between outcomes rather than how much larger a continuous measurement is. Under these conditions, a 2×2 test of association is the best choice. The chi-squared test for independence compares the observed 2×2 cell counts to those expected under the null of no association and is valid and well powered when all expected counts are reasonably large.

`snomed ct concept C2720507` (C2720507)

In [333…

```python
plot_hist_for_dead_v_live("C2720507")
```



```
unique values [0, 1]
```

In [334…

```
col = "C2720507"  # SNOMED/UMLS concept indicator (binary)

presence = (filtered_features[col].fillna(0) > 0).astype(int)
outcome  = labels["death_in_stay"].astype(int)

# 2x2 table: rows = survived(0)/died(1), cols = absent(0)/present(1)
tbl = pd.crosstab(outcome, presence).reindex(index=[0,1], columns=[0,1], fil
print(tbl)

# Chi-squared test (adequate counts assumed)
chi2, p, dof, _ = chi2_contingency(tbl.values, correction=False)
print(f"Chi-squared p-value = {p:.3g}")
```

```
C2720507          0     1
death_in_stay            
0               147  2678
1                31   599
Chi-squared p-value = 0.771
```

Which test did you utilize for the `C2720507` feature? Why?

Use a 2×2 association test. This SNOMED/clinical concept column is a binary indicator of concept presence in notes, and the outcome (died vs survived) is also binary. The right analysis is a contingency-table test: chi-squared for adequate expected counts.

`2.3` (`5 points`) Hypothesis Testing with the Bonferroni Correction

Generally, when we perform hypothesis tests, we are attempting to determine if a particular effect or difference is statistically significant. We usually use a significance level, often denoted as α (alpha), to make this determination. A common choice for α is

0.05, meaning that there is a 5% chance of incorrectly rejecting the null hypothesis (a Type I error) when it is actually true.

However, when we conduct multiple hypothesis tests simultaneously – for example, when we are testing multiple variables or conditions at once – the likelihood of encountering at least one Type I error (i.e., finding at least one "significant" result when there actually is none) increases with the number of tests. This phenomenon is due to the accumulated risk from each individual test.

To address this problem, the Bonferroni correction is often employed. The Bonferroni correction is a simple method to control the familywise error rate (FWER), which is the probability of making one or more Type I errors in a set of comparisons.

To apply the correction, simply divide the desired α level by the number of tests being performed. For example, if we are performing 20 tests and want to maintain an overall α level (significance threshold) of 0.05, we would use an adjusted α level of 0.05/20 = 0.0025 for each individual test.

$$\alpha_{adjusted} = \frac{\alpha_{desired}}{n_{tests}}$$

We should note that the Bonferroni correction is a very conservative method, meaning that it is likely to produce false negatives (i.e., not catching real associations because we fail to reject the null hypothesis when it is actually false). However, it is a simple and effective way to control the familywise error rate (FWER), and is often used as a baseline for comparison with other methods.

Implement the function `calc_bonferroni` and `multi_ttest` in `associative_analysis.py`. When you are finished, run the following cell.

In [335…
```
from src.hypothesis import multi_ttest

multi_test_results, sig_columns = multi_ttest(filtered_features, labels, col
```

How many of the chartvalue features are statistically significant after applying the Bonferroni correction?

In [336…
```
print(len(sig_columns), "features are statistically significant after applyi
```

```
26 features are statistically significant after applying the Bonferroni corr
ection
```

`3` (`10 points`) Regression Analysis

Regression analyses model the relationship between a dependent variable (often denoted as Y) and an independent variable (often denoted as X). In biomedical data science, regression analyses are often used to predict a patient's risk of developing a particular condition or experiencing a particular outcome based on their clinical data. Let's take a look at some of the simpliest forms of regression analysis: linear regression.

Simple Linear Regression: Involves one dependent and one independent variable. The relationship is modeled as a linear equation, where the estimated dependent variable (Y) is a linear combination of the independent variable (X) and a constant known as the intercept.

$$Y = \beta_0 + \beta_1 X + \epsilon$$

Here, β_0 is the intercept, β_1 is the coefficient for X, and ϵ is the error term. The error term accounts for the variability in Y that cannot be explained by the linear relationship with X. The goal of linear regression is to find the values of β_0 and β_1 that minimize the error term. Finding the values of β_0 and β_1 that minimize the error term is called "fitting the model" or "training the model". Once we have a fit model, we can use it to make predictions for Y given new (or unseen) values of X. The estimated value of Y (denoted as $\hat{Y}$) for any given value of X can be calculated using the following equation:

$$\hat{Y} = \beta_0 + \beta_1 X$$

You can also have multiple independent variables in a linear regression model. This is known as multiple linear regression. The equation for multiple linear regression is:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

Where β_0 is the intercept, β_1 to β_n are the coefficients for the independent variables X_1 to X_n, and ϵ is the error term.

Another type of regression is called `Logistic Regression`. Logistic regression is generally utilized when the dependent variable (the one we are trying to predict) is a binary outcome (e.g., "yes" or "no", "died" or "survived"). Logistic regression is similar to linear regression, but instead of predicting a continuous value, it predicts a value that falls between 0 and 1, which can be interpreted as a *predicted* probability of the dependent variable taking on the value 1 (or "yes", or "died", etc.). The equation for logistic regression is:

$$Y = \frac{1}{1+e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon)}}$$

Where β_0 is the intercept, β_1 to β_n are the coefficients for the independent variables X_1 to X_n, and ϵ is the error term.

Main takeaway:

- **Linear regression**: used to predict a **continuous** value (think: problems where you want to predict a value)

- **Logistic Regression**: used to predict a **binary** value (think: binary classification problems)

In this part of the assignment you will build and compare several regression models to predict the binary outcome of `death_in_stay`, utilizing various features.

`3.1` (`3 points`) Building Logistic Regression Models

Implement the `run_logistic_regression_model` function in `regression.py` following the implementation instructions in the docstring. When you are finished, run the following cell.

In [337…
```
from src.regression import run_logistic_regression

# glm_1: A logistic regression that only utilizes the "oxy_drop" and "age_ir
model_1_features = ["oxy_drop", "age_in_days"]
glm_1 = run_logistic_regression(filtered_features, labels, model_1_features)

# glm_2: A logistic regression that utilizes the "oxy_drop", "age_in_days",
model_2_features = ["oxy_drop", "age_in_days", "gender"]
glm_2 = run_logistic_regression(filtered_features, labels, model_2_features)

# glm_3: A logistic regression that utilizes `age_in_days`, `gender`, `oxy_d
# and all of the chart value features that were found to have statistically
# associations with the outcome in the previous section.
model_3_features = ["oxy_drop", "age_in_days", "gender"]
model_3_features.extend(sig_columns)
glm_3 = run_logistic_regression(filtered_features, labels, model_3_features)
```

`3.2` (`3 points`) Evaluating and Interpreting Regression Coefficients

Now that we have fit the models, let's compare them!

What is the point estimate and confidence interval for the `oxy_drop` coefficient in each model?

Why do you think the point estimate of the coefficient changes as we add more features to the model?

> HINT: Use the result summaries from above to answer this question.

In [338…
```
def coef_summary(model, var="oxy_drop", alpha=0.05):
    beta = model.params[var]
    se = model.bse[var]
    ci = model.conf_int(alpha=alpha).loc[var]
    p = model.pvalues[var]
    # Odds-ratio scale
    or_ = np.exp(beta)
    or_ci_low, or_ci_high = np.exp(ci[0]), np.exp(ci[1])
    return {
        "beta": beta,
        "se": se,
        "ci_low": ci[0],
        "ci_high": ci[1],
        "p_value": p,
        "odds_ratio": or_,
        "or_ci_low": or_ci_low,
        "or_ci_high": or_ci_high,
    }

rows = []
for name, m in [("glm_1", glm_1), ("glm_2", glm_2), ("glm_3", glm_3)]:
    s = coef_summary(m, var="oxy_drop", alpha=0.05)
    s["model"] = name
    rows.append(s)

oxy_summary = pd.DataFrame(rows)[
    ["model", "beta", "ci_low", "ci_high", "p_value", "odds_ratio", "or_ci_l
].round(4)

display(oxy_summary)
```

	model	beta	ci_low	ci_high	p_value	odds_ratio	or_ci_low	or_ci_high
0	glm_1	0.5747	0.3908	0.7587	0.000	1.7767	1.4781	2.1356
1	glm_2	0.5778	0.3936	0.7621	0.000	1.7822	1.4823	2.1427
2	glm_3	0.3151	0.1069	0.5233	0.003	1.3704	1.1128	1.6876

The oxy_drop coefficient changes as I add features because the simple models mix the effect of oxy_drop with correlated factors, while richer models adjust for those covariates and isolate the partial association. Gender adds little, so glm_1≈glm_2, suggesting it's not a confounder here. In glm_3, adding clinically related chart values (vitals/labs) that correlate with both oxy_drop and mortality absorbs part of the signal, so the oxy_drop log-odds shrinks toward the null. Also, logistic regression odds ratios are non-collapsible: even without true confounding, conditioning on additional predictors can change the coefficient magnitude.

Assuming you had a model of Y regressed on X_1 and you added the variable X_2, under what conditions would the coefficient for X_1 *not* change?

If X_2 had zero effect / correlation with Y, meaning the coefficient / weight for X_2 was zero.

If you had a model to estimate Y regressed on X_1, and you added the variable X_2, what would likely happen to the coefficient of X_1 if both X_1 and X_2 were positively correlated with the outcome?

It will typically shrink toward zero. When I add X2 (also positively related to Y and usually positively correlated with X1), the model attributes part of the variation in Y that X1 captured in the 1-variable model to X2, so X1's coefficient reflects its partial effect conditional on X2 and decreases in magnitude

`3.3` (`4 points`) Assessing Model Performance with Mean Squared Error

Another way to compare models is to compare metrics such as the Mean Squared Error (MSE) and R-squared. These metrics are often used to evaluate the performance of regression models.

MSE is the average of the squared errors (the difference between the predicted value of Y and the actual value of Y). It is calculated as follows:

$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$

R-squared (also known as the coefficient of determination) represents the proportion of variance in the dependent variable that is explained by the independent variables in the model.

In [339…
```
model_1_probs = glm_1.predict(filtered_features[model_1_features])
```

In [340…
```
model_1_probs
```

Out[340…
```
subject_id
91       0.161155
106      0.197938
111      0.237709
117      0.219587
3        0.248640
           ...
31983    0.257758
32506    0.148017
32513    0.158879
32408    0.139875
32417    0.140377
Length: 3455, dtype: float64
```

In [341…
```
# Run this cell to get the R^2 and MSE values for each model
from sklearn.metrics import r2_score, mean_squared_error

# Let's compare these values for the different models
# First, let's get the predicted probabilities for each model
model_1_probs = glm_1.predict(filtered_features[model_1_features]).values
model_2_probs = glm_2.predict(filtered_features[model_2_features]).values
model_3_probs = glm_3.predict(filtered_features[model_3_features]).values

# Now, let's calculate the R^2 and MSE for each model
model_1_r2 = r2_score(labels, model_1_probs)
```

```
model_1_mse = mean_squared_error(labels, model_1_probs)
model_2_r2 = r2_score(labels, model_2_probs)
model_2_mse = mean_squared_error(labels, model_2_probs)
model_3_r2 = r2_score(labels, model_3_probs)
model_3_mse = mean_squared_error(labels, model_3_probs)

# Let's print these out
print(f"Model 1: R^2 = {model_1_r2}, MSE = {model_1_mse}")
print(f"Model 2: R^2 = {model_2_r2}, MSE = {model_2_mse}")
print(f"Model 3: R^2 = {model_3_r2}, MSE = {model_3_mse}")
```

```
Model 1: R^2 = 0.029348980226378885, MSE = 0.14471915342743286
Model 2: R^2 = 0.029308258237925267, MSE = 0.14472522486976883
Model 3: R^2 = 0.16890778719907895, MSE = 0.12391164178113381
```

Which of the three models do you think has the best fit to the data we utilized? Explain your reasoning.

Model 3. It has the highest R^2 (~0.161 vs ~0.029) and the lowest MSE (~0.124 vs ~0.145), so it explains more variance in the binary outcome and yields more accurate probabilities. When we sum up all the error we get a smaller value meaning we regress closer to the source of truth, meaning it has the best fit.

If we were to use these models to predict outcomes for a group of new patients (patients the models had not seen before from a different dataset), would the same model perform the best on the new dataset as well? Explain your reasoning

Not guaranteed. Model 3 fits this dataset best, but with more features it’s also at higher risk of overfitting to the exact dataset that it has seen. Essentially, its weights over indexing on the training data. On a new cohort a simpler model could generalize better. The only way to know is external validation: testing on a dataset thatthe model has never seen.

`4` (`23 points`) Survival Analysis

Survival analysis, also known as time-to-event analysis, is a branch of statistics that deals with analyzing the expected duration of time until one or more events happen. In the context of biomedical data science, these events often represent the occurrence of a particular health-related outcome, such as the onset of a disease, death, or recovery. Survival analysis is a powerful tool that can be used to answer a wide range of application areas, such as in clinical trials (to compare distributions of various treatment groups), epidemilogical studies (to identify risk factors that may influence the time to event prediction), and even patient monitoring (to predict the time to a equipment failure or a patient's discharge).

In this part of the assignment, you will perform some survival analysis on the cohort's data.

`4.1` (`3 points`) Preparing and Calculating Survival Time Variables

The first step of doing a survival analysis is to get our data into the appropriate format. To start, let's calculate the `survival time` for each patient. The `survival time` is the number of days between the patient's admission and the time of a particular event.

Implement the function `calc_survival_time` in `survival.py`. When you are finished, run the following cell.

In [342…

```python
from src.survival import calc_survival_time

# Run this cell to calculate the survival time for each patient
survival_df = calc_survival_time(cohort)


# Sanity Check
print(max(survival_df["survival_time_days"]))
if max(survival_df["survival_time_days"]) != 172:
    warnings.warn("Max survival time is incorrect", SanityCheck)
```

```
172
```

`4.2` (`13 points`) Visualizing Survival Probabilities with Kaplan–Meier Estimation

In survival analysis, an extremely commonly utilized tool is the `Kaplan–Meier (KM) estimator`, and the associated visualization method, the `Kaplan–Meier curve`. The `Kaplan–Meier (KM) estimator` is a non-parametric* method used to estimate the `survival function` from time-to-event data, which is the probability that a particular event has not occurred yet at a given time point. The `Kaplan-Meier curve` is a graphical representation of the `survival function` over time, and is often used to compare the survival distributions of different groups.

In this part of the assignment, you will set up a function to generate a `Kaplan-Meier curve` for the cohort's data. You will then use the `Kaplan-Meier curve` to compare the survival distributions of patients who experienced a sudden drop in oxygenation (oxy_drop) to those who did not. Before we get started, let's learn a bit more about how the `Kaplan-Meier estimator` works.

The `Kaplan-Meier estimator` is a product-limit estimator, meaning that it is calculated as the product of the conditional probabilities of surviving each time interval. At each time point in our data, we need to know the following:

- n_i = The number of patients who are at risk of experiencing the event at time t_i
- d_i = The number of patients who experienced the event at time t_i

The `Kaplan-Meier estimator` for the survival function at time t_i is calculated as the product of the probabilities for all observed time points up to and including t_i:

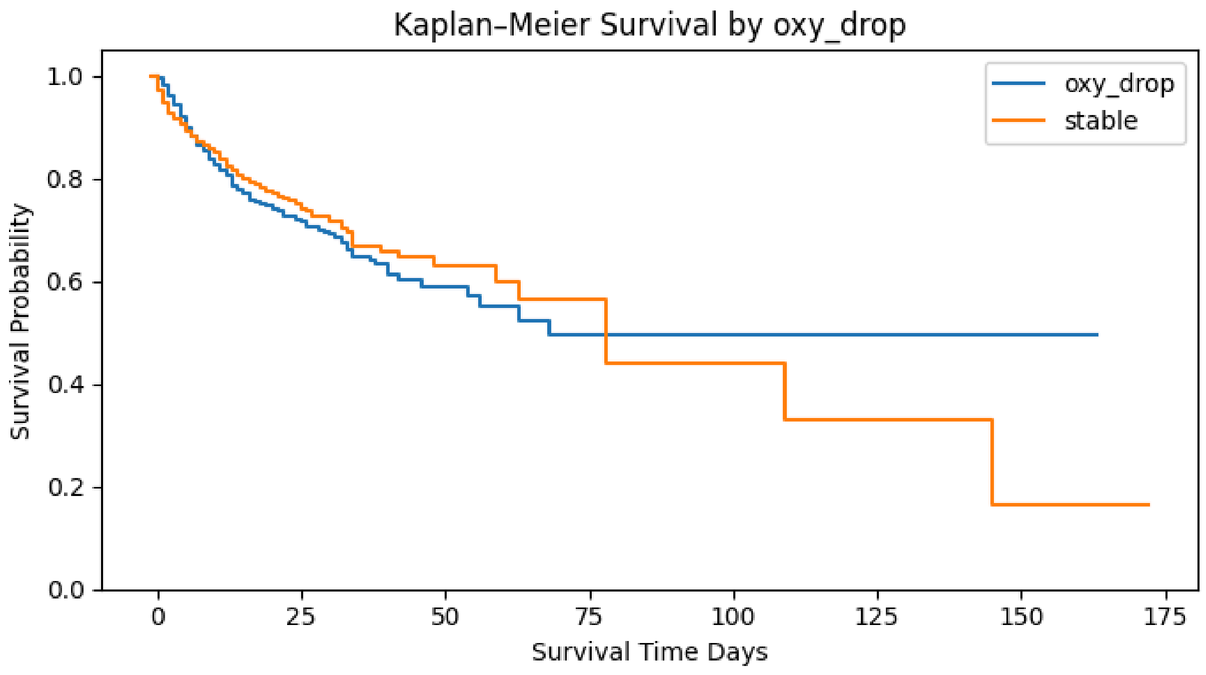
$$\hat{S}(t_i) = \prod_{j:t_j \leq t_i} \frac{n_j - d_j}{n_j} = \prod_{j:t_j \leq t_i} P(T > t_j | T > t_{j-1})$$

The `Kaplan-Meier curve` is a graphical representation of the `Kaplan-Meier estimator` over time. The `Kaplan-Meier curve` is a step function, where the value of the curve at each time point is equal to the value of the `Kaplan-Meier estimator` at that time point (with time on the x-axis and the `Kaplan-Meier estimator` on the y-axis).

> * A "non-parameteric" method in statistics refers to a type of method that makes few (if any) assumptions about data. This contrasts with "parametric" methods which assume some sort of distributional form for the data. The Kaplan-Meier curve is a non-parametric method because it does not assume any particular distribution for the survival times, and **utilizes only the observed data to estimate the survival function.**

Implement the function `display_kaplan_meier_curve` in `survival.py`. When you are finished, run the following cell.

In [343…
```
from src.survival import display_kaplan_meier_curve

display_kaplan_meier_curve(survival_df, "kaplan_meier_curve.png")
```



`4.3` (`7 points`) Modeling Time-to-Event Outcomes with the Cox Proportional Hazards Model

Another fundamental tool of survival analysis is the `Cox Proportional Hazards Model`. The `Cox Proportional Hazards Model` is a semi-parametric* method used to model the `hazard function` from time-to-event data. The `hazard function` is the probability that a particular event occurs at a given time point, given that the event has not occurred yet. (Contrast the hazard function with the survival function we talked about above!)

The `Cox Proportional Hazards Model` is a popular tool for survival analysis because it is relatively simple to implement, and it allows us to estimate the effect of multiple variables on the `hazard function` simultaneously.

> * A semi-parametric method is a method that includes both parametric and non-parametric components.

Here we will implement a Cox Proportional Hazards Model utilizing the `lifelines` library.

In [344…
```
# Run this cell to import the CoxPHFitter class from the lifelines package.
# Use this class to fit a Cox Proportional Hazards model to the survival dat
from lifelines import CoxPHFitter
```

In [345…
```
# Run this cell (no edits needed) to prepare the data for the cox model

# Create a survival dataframe that contains the survival time and feature co
surv_features = filtered_features.merge(survival_df.drop(columns=["oxy_drop"
```

```python
# Create a binary column for death event
surv_features["death_event"] = (surv_features['death_in_stay'] == 'died').as

# Remove the "death_in_stay" column
surv_features.drop(columns=["death_in_stay"], inplace=True)
```

Fit a Cox Proportional Hazards model using the cell below. You will need to use the `CoxPHFitter` class from the `lifelines` library. You can find the documentation for the `CoxPHFitter` class here:
https://lifelines.readthedocs.io/en/latest/fitters/regression/CoxPHFitter.html

In your implementation, use `oxy_drop` as the only predictor(formula), `death_event` as the event, and `survival_time_days` as the duration.

In [346…

```python
# Prepare data for lifelines: event as 0/1, predictor as 0/1
cox_df = survival_df.copy()
cox_df["death_event"] = cox_df["death_in_stay"].astype(str).str.lower().eq("
cox_df["oxy_drop"] = cox_df["oxy_drop"].astype(str).str.lower().eq("oxy_drop

# Fit Cox PH model with oxy_drop as the only predictor
cph = CoxPHFitter()
cph.fit(
    cox_df[["survival_time_days", "death_event", "oxy_drop"]],
    duration_col="survival_time_days",
    event_col="death_event",
    formula="oxy_drop",
)

# View results
cph.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'survival_time_days'
event col	'death_event'
baseline estimation	breslow
number of observations	3455
number of events observed	630
partial log-likelihood	-4726.58
time fit was run	2025-11-11 00:15:52 UTC

	coef	**exp(coef)**	**se(coef)**	**coef lower 95%**	**coef upper 95%**	**exp(coef) lower 95%**	**exp(coef) upper 95%**	**cmp to**	**z**	
oxy_drop	0.08	1.08	0.08	-0.09	0.24	0.92	1.27	0.00	0.91	0.3

Concordance	0.50
Partial AIC	9455.17
log-likelihood ratio test	0.83 on 1 df
-log2(p) of ll-ratio test	1.46

What is the point estimate and 95% confidence interval of the coefficient for `oxy_drop`?

In [347…

```
row = cph.summary.loc["oxy_drop"]
beta = row["coef"]
ci_low, ci_high = row["coef lower 95%"], row["coef upper 95%"]
hr = row["exp(coef)"]
hr_low, hr_high = row["exp(coef) lower 95%"], row["exp(coef) upper 95%"]
print(f"log-HR = {beta:.2f} (95% CI {ci_low:.2f}, {ci_high:.2f}); "
      f"HR = {hr:.2f} (95% CI {hr_low:.2f}, {hr_high:.2f})")
```

```
log-HR = 0.08 (95% CI -0.09, 0.24); HR = 1.08 (95% CI 0.92, 1.27)
```

In addition, let's run another version of the model adjusted for all of the `sig_columns` (the columns known to have significant associations in `filtered_features`) AND the `oxy_drop` column. Use the same event and duration as above, and utilize the `sig_columns` and `oxy_drop` as the predictors (formula).

In [348…

```
# 1) Start from survival data and set subject_id as index
cox_adj = survival_df.copy()
cox_adj["death_event"] = cox_adj["death_in_stay"].astype(str).str.lower().ec
cox_adj["oxy_drop"] = cox_adj["oxy_drop"].astype(str).str.lower().eq("oxy_dr
cox_adj = cox_adj.set_index("subject_id")
```

```python
# 2) Align significant features by subject_id
X_all = filtered_features.copy()  # already indexed by subject_id
avail_sig = [c for c in sig_columns if c in X_all.columns]
cox_adj = cox_adj.join(X_all[avail_sig], how="left")

# 3) Build formula and model frame; drop rows with any NA in variables used
predictors = ["oxy_drop"] + avail_sig
model_cols = ["survival_time_days", "death_event"] + predictors
df_model = cox_adj[model_cols].dropna()

# 4) Fit Cox PH
formula = " + ".join(predictors)
cph_adj = CoxPHFitter()
cph_adj.fit(
    df_model,
    duration_col="survival_time_days",
    event_col="death_event",
    formula=formula,
)

cph_adj.print_summary()
```

model	lifelines.CoxPHFitter
duration col	'survival_time_days'
event col	'death_event'
baseline estimation	breslow
number of observations	3455
number of events observed	630
partial log-likelihood	-4520.85
time fit was run	2025-11-11 00:15:53 UTC

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to
oxy_drop	-0.20	0.82	0.09	-0.38	-0.03	0.69	0.97	0.00
chartvalue_1127	-0.05	0.95	0.03	-0.10	0.00	0.91	1.00	0.00
chartvalue_184	-0.15	0.86	0.08	-0.32	0.01	0.73	1.01	0.00
chartvalue_190	0.95	2.59	0.45	0.07	1.84	1.07	6.27	0.00
chartvalue_198	0.01	1.01	0.06	-0.12	0.13	0.89	1.14	0.00
chartvalue_444	-0.02	0.98	0.02	-0.05	0.01	0.95	1.01	0.00
chartvalue_450	0.06	1.07	0.02	0.02	0.10	1.02	1.11	0.00
chartvalue_454	-0.15	0.86	0.07	-0.29	-0.01	0.75	0.99	0.00
chartvalue_615	-0.01	0.99	0.01	-0.04	0.01	0.96	1.01	0.00
chartvalue_618	0.02	1.02	0.01	0.01	0.04	1.01	1.04	0.00
chartvalue_619	0.01	1.01	0.01	-0.01	0.03	0.99	1.03	0.00
chartvalue_682	-0.00	1.00	0.00	-0.00	0.00	1.00	1.00	0.00
chartvalue_683	-0.00	1.00	0.00	-0.00	-0.00	1.00	1.00	0.00
chartvalue_776	0.01	1.01	0.03	-0.06	0.07	0.94	1.07	0.00
chartvalue_777	0.00	1.00	0.04	-0.07	0.08	0.93	1.08	0.00
chartvalue_778	-0.03	0.97	0.01	-0.06	-0.01	0.95	0.99	0.00
chartvalue_779	0.00	1.00	0.00	0.00	0.00	1.00	1.00	0.00
chartvalue_781	0.01	1.01	0.00	0.00	0.01	1.00	1.01	0.00
chartvalue_787	0.01	1.01	0.02	-0.03	0.04	0.97	1.04	0.00
chartvalue_791	0.00	1.00	0.03	-0.06	0.07	0.94	1.07	0.00
chartvalue_811	0.00	1.00	0.00	0.00	0.00	1.00	1.00	0.00
chartvalue_815	0.24	1.27	0.11	0.03	0.44	1.03	1.56	0.00

	coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	
chartvalue_824	-0.02	0.98	0.02	-0.06	0.02	0.94	1.02	0.00	-
chartvalue_825	0.00	1.00	0.00	0.00	0.01	1.00	1.01	0.00	
chartvalue_837	0.04	1.04	0.01	0.02	0.05	1.02	1.06	0.00	
chartvalue_861	0.06	1.06	0.02	0.01	0.11	1.01	1.11	0.00	
chartvalue_87	-0.12	0.89	0.02	-0.17	-0.07	0.85	0.93	0.00	-

Concordance	0.75
Partial AIC	9095.70
log-likelihood ratio test	412.30 on 27 df
-log2(p) of ll-ratio test	231.90

What is the point estimate and 95% confidence interval of the coefficient for `oxy_drop` in the adjusted model?

In [349…

```
# Extract oxy_drop coefficient and 95% CI from the adjusted Cox model (cph_a
row = cph_adj.summary.loc["oxy_drop"]
beta = row["coef"]
ci_low, ci_high = row["coef lower 95%"], row["coef upper 95%"]
hr = row["exp(coef)"]
hr_low, hr_high = row["exp(coef) lower 95%"], row["exp(coef) upper 95%"]

print(f"Adjusted model — oxy_drop:")
print(f"log-HR (coef) = {beta:.3f}  (95% CI {ci_low:.3f}, {ci_high:.3f})")
print(f"HR = {hr:.3f}  (95% CI {hr_low:.3f}, {hr_high:.3f})")
```

```
Adjusted model — oxy_drop:
log-HR (coef) = -0.204  (95% CI -0.377, -0.032)
HR = 0.815  (95% CI 0.686, 0.969)
```

`5` (`17 points`) Univariate and Unadjusted Causal Analyses

In our associative analyses, we saw that we could do a reasonable job of predicting mortality, but there are many cases where having a prediction isn't as useful as understanding the underlying causal mechanisms. To improve standards of care, clinicians need to know which factors have causal relationships with mortality, so they can intervene correctly. This line of inquiry is known as **causal inference**, and it is a very active area of research in biomedical data science.

In this part of the assignment, we will explore some of the methods that can be used to infer causality from observational data. We will investigate to what extent having a sudden and sustained drop in oxygenation during ventilation (which we will refer to as the "exposure") is *causally* related to death later in the hospitalization (which we will refer to as the "outcome").

`5.1` (`10 points`) Causal Analyses Without Matching Methods

When someone refers to analyses without 'matching' in the context of causal inference, they are discussing a method of analysis where units (e.g., individuals in a study) are *not* paired or grouped based on similar characteristics or treatment propensities before estimating treatment effects.

For this first part, let's see what kind of analysis we can do **without** matching.

Univariate analysis, in a general statistical context, refers to the examination of a single variable's properties, distribution, and characteristics. However, in the context of causal inference, "univariate analysis" can be somewhat of a misnomer, given that causal inference inherently involves understanding the relationship between two or more variables: a treatment and an outcome, at minimum.

When people refer to "univariate analysis" in the context of causal inference, they often mean examining the causal effect of one treatment on an outcome, without adjusting for potential confounders or covariates. In other words, it's a simple analysis that doesn't account for other variables that might influence the relationship between treatment and the outcome.

Researchers often start with a univariate analysis to get a basic understanding of the relationship between variables, then proceed to multivariate analysis where they adjust for confounders and possibly investigate interactions between variables. For our univariate analysis in this context, let's run a statistical test to see if a drop in oxygenation is related to mortality.

Run an appropriate statistical test to see if a drop in oxygenation is related to mortality. Treat both variables as binary.

In [350…

```
# Univariate test: association between oxygen drop (binary) and mortality (b

# Build 2x2 contingency table (rows: survived=0/died=1; cols: no drop=0/drop
y = labels["death_in_stay"].astype(int)
x = filtered_features["oxy_drop"].astype(int)
tbl = pd.crosstab(y, x).reindex(index=[0,1], columns=[0,1], fill_value=0)
print("Contingency table:\n", tbl, "\n")
```

```
# Chi-squared test; if any expected cell < 5, fall back to Fisher's exact
chi2, p_chi, dof, expected = chi2_contingency(tbl.values, correction=False)

# Approximate odds ratio from 2x2
a, b = tbl.loc[1,1], tbl.loc[1,0]
c, d = tbl.loc[0,1], tbl.loc[0,0]
or_est = (a*d) / (b*c) if b*c != 0 else float("inf")
p = p_chi
test_used = "Chi-squared"

print(f"{test_used} p-value: {p:.3g}")
```

```
Contingency table:
 oxy_drop          0    1
death_in_stay
0               2092  733
1                391  239

Chi-squared p-value: 1.43e-09
```

What is the odds ratio?

In [351…

```
print(f"Odds ratio (oxy_drop vs no drop): {or_est:.3f}")
```

```
Odds ratio (oxy_drop vs no drop): 1.745
```

Does a drop in oxygenation appear to significantly decrease or increase the risk of death? Does this establish causality between a drop in oxygenation and mortality? Why or why not? (If you think it does not, how else would you explain the result?)

From the univariate 2×2 test, we see that an oxygen drop is associated with a higher risk of death (odds ratio ≈ 1.75, p ≈ 1e-9), so patients with a drop are more likely to die than those without. However, I do not interpret this as causal: this is an unadjusted association and oxy_drop likely reflects underlying illness severity or other confounders. To make a causal claim, I would adjust for confounders between the two groups that are being tested

`5.2` (`5 points`) Multivariate Causal Modeling: Adjusting for Confounders

Let's see if we still observe an effect after adjusting for other features.

In the previous section, we implemented a logisitic regression on different sets of features. Run the following cell to reuse your code from the previous section to fit a logistic regression model of mortality on the features that were found to have non-zero variance.

```
In [352…  # Run this cell (No changes necessary)
          from src.regression import run_logistic_regression

          glm_nzv = run_logistic_regression(filtered_features, labels, selected_featur
```

Is the coefficient of the feature encoding a drop in oxygenation statistically significant in the model?

```
In [353…  # Is oxy_drop statistically significant in the multivariate logistic model?
          summary = glm_nzv.summary2().tables[1]
          row = summary.loc["oxy_drop"]
          p = row["P>|z|"]
          beta = row["Coef."]
          ci_low, ci_high = row["[0.025"], row["0.975]"]

          print(f"oxy_drop: beta={beta:.3f}, 95% CI [{ci_low:.3f}, {ci_high:.3f}], p={
          print("Significant at 0.05?" , p < 0.05)
```

```
oxy_drop: beta=0.415, 95% CI [0.143, 0.687], p=0.00276
Significant at 0.05? True
```

What is the equivalent odds ratio?

```
In [354…  or_ = np.exp(beta)
          or_low = np.exp(ci_low)
          or_high = np.exp(ci_high)

          print(f"oxy_drop odds ratio = {or_:.3f} (95% CI {or_low:.3f}, {or_high:.3f})
```

```
oxy_drop odds ratio = 1.514 (95% CI 1.154, 1.987)
```

Does this establish causality between a drop in oxygenation and mortality? Why or why not? If you think it does not, how else would you explain the result?

No. An adjusted odds ratio of 1.51 (95% CI 1.15–1.99) shows an association, not causation. This is an observational model and can still suffer from residual/unmeasured confounding variables. There could be statistical differences in the underlying groups that are driving these differences, no real direct causation.

`5.3` (`2 points`) Evaluating the Feasibility of Experimental Causal Designs

Usually questions related to causality are best answered through highly controlled expirements, such as Randomized Control Trials (RCTs). However, in many contexts in medicine and healthcare, it is either not possible or ethical to conduct an RCT.

Let's take our example: "To what extent is having a sudden and sustained drop in oxygenation during ventilation *causally* related to death later in the hospitalization"

- **Is there an experiment you could run that "in theory" would let you determine the causal effect of a sudden oxygenation drop on mortality using only these inferential analyses?**
- **What might be some practical or ethical problems with your experiment?**

Purely in theory, I could run a RCT taht assigns ICU patients to two groups. I would do everything possible to make sure these two groups of ICU patients are identical, or at least that each given patient has some corollary in the other group. I would then essentially randomly one of the groups and go through extreme measures to ensure they do not undergo oxygen drop - this is the intervention tested in random control. I'd then analyze effectiveness and use assignment as an instrument for oxy_drop to estimate the causal effect of desaturation on mortality

Ethically, I cannot randomize patients to "allow" desaturation; provoking drops is unethical and violates equipoise. If we believe that oxygen_drop is somewhat associated with death (hence the study) then we are essentially forgoing treatment that we believe can save the lives of the control group.

`6` (`30 points`) Estimating Causal Effects Using Propensity Score Matching

Let's see if we can more conclusively determine a causal effect of a sudden oxygenation drop on mortality using *matching* in our analysis.

Matching is a technique used to control for confounding variables in observational data by grouping units (e.g. individuals in a study) based on similar characteristics or propensities before estimating treatment effects. Matching is a popular technique in causal inference because it allows us to estimate the causal effect of a treatment on an outcome by comparing units that are similar in all aspects except for the treatment.

> Note that sometimes "matching" is described as mechanism to "mimic" or "emulate" the randomization of a RCT. This interpretation is incorrect: matching is just a tool you can use to adjust for confounders in a non-randomized study, but does not introduce randomization.

In this assignment, we will utilize `propensity scores` to perform a matching analysis.

Let's start with the definition: A `propensity score` is the probability that a unit (e.g. an individual in a study) was exposed to a particular **`treatment or exposure`** given their observed characteristics. So in our example, the propensity score of an individual patient is the probability that they had a sudden oxygenation drop (`oxy_drop` is 1) given their other features.

A propensity score matching analysis involves the following steps:

1. Calculate the propensity scores for each unit in the dataset: The first step in our process is to determine propensity scores for each unit in the dataset. In our example, we want to calculate the probability that a patient had a sudden oxygenation drop (`oxy_drop` is 1) given their other features. We can do this by fitting a logistic regression model of `oxy_drop` on the other features in the dataset (you do not including the outcome: `death_in_stay`). The predicted probabilities from this model are the propensity scores for each patient.

2. Match units based on their propensity scores: There are multiple strategies to do this, but the idea is that patients who have similar propensity scores are likely to have similar characteristics, and therefore are likely to be comparable in terms of their potential outcomes. In our example, we will use a method called "Caliper Matching". We will discribe caliper matching in more detail in the sections below, and utilize it to match patients in the exposure group to patients in the control group (note that not all patients will be matched).

3. Run an analysis only utilizing patients that were matched in the previous step.

Patients who have similar propensities are more "twin-like" and therefore more comparable in terms of their potential outcomes. If two patients have similar propensity scores and one of them was "exposed" while the other was not, we can be more confident that any difference in their outcomes is due to the exposure, rather than due to differences in their characteristics.

`6.1` (`5 points`) Building Propensity Score Models to Estimate Exposure Probability

Let's start our propensity analysis by calculating the propensity scores for each patient in the dataset. Utilizing the same code from before, lets fit a logisitic regression model to predict `oxy_drop` using the other features in the dataset.

In [355…

```
from src.regression import run_logistic_regression

# Features (exclude oxy_drop)
feature_list = filtered_features.columns.tolist()
feature_list.remove("oxy_drop")

# Labels for propensity model: rename oxy_drop -> death_in_stay for the help
labels_ps = filtered_features[["oxy_drop"]].rename(columns={"oxy_drop": "dea

# Fit logistic model to estimate propensity of oxy_drop
glm_propensity = run_logistic_regression(
    filtered_features, labels_ps, selected_features=feature_list
)
```

```
# Propensity scores (predicted probability of oxy_drop)
propensity_scores = glm_propensity.predict(filtered_features[feature_list]).
```

In [356…

```
# (No modifications needed in this cell)
# Run this cell to use the model to calculate propensity scores
propensity_scores = glm_propensity.predict(filtered_features[feature_list])

# Name the series "propensity_score"
propensity_scores.name = "propensity_score"

# Merge with oxy drop
propensity_scores = pd.concat([propensity_scores, filtered_features["oxy_drc

# Display the first 5 rows of the propensity_scores series
propensity_scores.head()
```

Out[356…

	propensity_score	oxy_drop
subject_id		
91	0.217897	0
106	0.246065	1
111	0.114306	1
117	0.924567	1
3	0.901581	1

`6.2` (`2 points`) Visualizing and Interpreting Propensity Score Distributions

Now that we have fit our propensity model and have calculated our propensity scores, let's take a look at the distribution of propensity scores in our dataset. Rather than plotting the distribution of propensity scores themselves, we will plot the distirubtions of the logits of the propensity scores.

The **logit** of a probability is defined as the natural logarithm of the odds of the probability. The logit of a probability p is calculated as follows:

$$logit(p) = ln(\frac{p}{1-p})$$

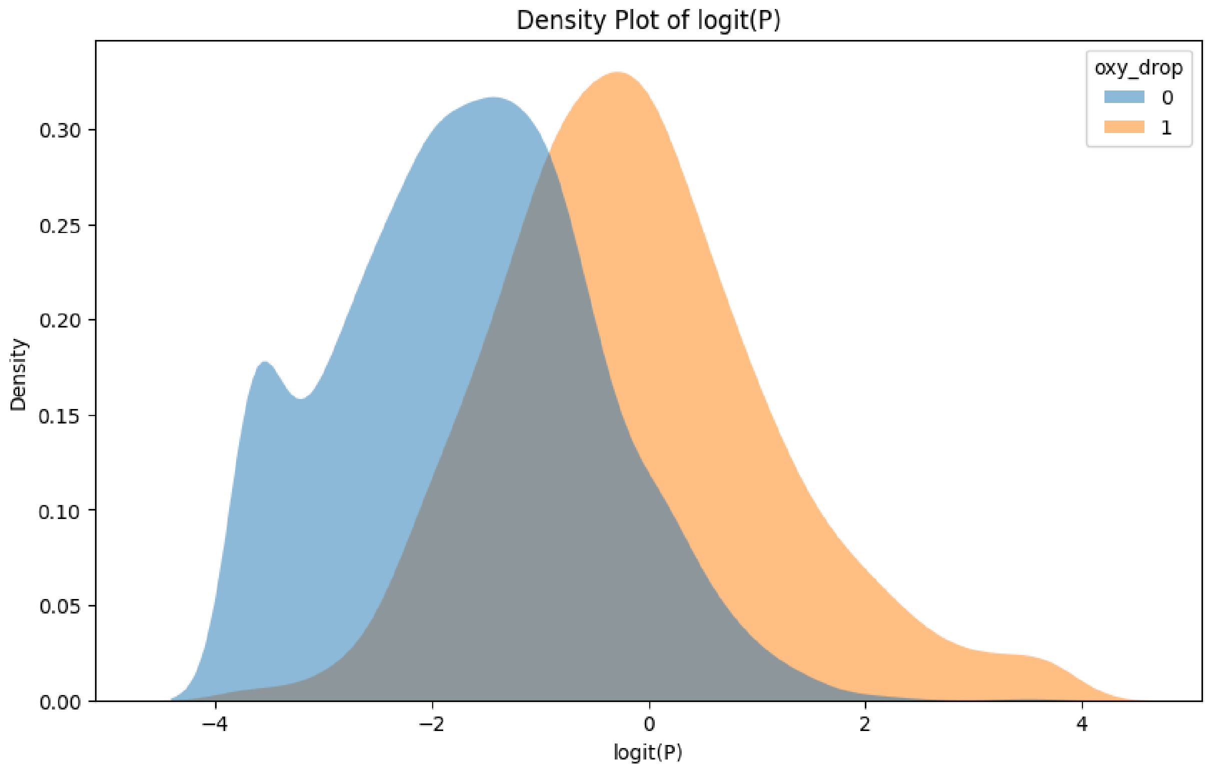
Why are visualizing the logit transformation of the propensity scores, rather than the scores themselves? Transformations (like the logit transformation) are often applied to sets of data to "change the scale", which can be really helpful for visualizing or interpreting distributions. In this case, we are using the logit transformation to "re-scale" the propensity scores (which are bounded between 0 and 1) to a range that is unbounded. Propensity scores close to 0 suggest a low probability of exposure, while scores close to 1 suggest a high probability of exposure. In some datasets, many scores can cluster near these boundaries, making it hard to visually assess overlap in the distributions. After applying the logit transformation, a propensity score of 0.5 is centered at 0, giving a much more normalized distribution. By transforming to logits, the scores that were clustered near 0 and 1 are spread out more evenly, making it easier to see the distribution of both groups and assess the overlap.

The primary goal of propensity score methods in observational studies is to reduce confounding by making the treated and untreated groups comparable in terms of the observed covariates. If there's significant overlap in the logit-transformed scores, it indicates that there's a good chance of finding comparable treated and untreated observations for causal inference analysis.

Implement the function `plot_logit_scores` in `matching.py` and run the following cell to test your implementation.

In [357…

```
from src.matching import plot_logit_scores

# Run this cell to plot the logits of the propensity scores
plot_logit_scores(propensity_scores, "logit_score_plot.png")
```



What does the amount of overlap between these two distributions say about how good your propensity score model is predicting the exposure?

Is it bad if your model has poor performance? (hint: if the exposure happened totally at random, would that help you or hurt you in determining a causal effect?)

There is a fair amount of overlap, showing up propsensity score is effective, but not quite optimal. Ideally, we would want totally distinct distributios. The two logit(P) curves show clear separation but substantial overlap. My propensity model can predict exposure better than chance (scores for oxy_drop=1 are generally higher), yet there is broad common support where treated and control patients have similar propensities

Poor predictive performance is not inherently bad for causal inference. If exposure were random, a propensity model would perform poorly at prediction (all scores near 0.5), but that randomness would help identification by making treated and control groups exchangeable. This would make it easier to build a given study cohort. What hurts causal estimation is lack of overlap (ironically, near-perfect prediction), because then you cannot match patients between the control and cohort as they come from totally different distributions and had totally different probabilities of actually contracting the condition.

`6.3` (`3 points`) Implementing Caliper Matching to Create Balanced Groups

The next step of our propensity score analysis is to match patients based on their propensities using Caliper Matching. Caliper Matching involves matching units (patients in this case) within a specified distance (caliper) of each other. The caliper is a threshold that defines the maximum distance between two units' propensity scores for them to be considered a match.

Implement the function `caliper_match` in `matching.py` using the instructions in the docstring. When you are finished, run the following cell.

In [358…
```
from src.matching import caliper_match

# Run this cell to perform the matching
match_list = caliper_match(propensity_scores, caliper = 0.25)

# Sanity Check
all_matched_subject_ids = [subject_id for match_pair in match_list for subje

if len(all_matched_subject_ids) != len(set(all_matched_subject_ids)):
    warnings.warn("Duplicate subject_id found in all_matched_subject_ids", S
```

How many patients are in the matched set? How does this number compare to the original number of patients in the dataset?

In [359…
```
print("number of patients in matched set:", len(all_matched_subject_ids))
print("number of patients in original dataset:", len(propensity_scores))
```

```
number of patients in matched set: 1524
number of patients in original dataset: 3455
```

The matched set had 1524 patients. This comes out to be about 44% of the patients from the original set actually met the requirements with the requisite caliper.

`6.4` (`5 points`) Estimating Treatment Effects Using Matched Data

Now let's repeat our univariate analysis, but this time only use data from the matched patients.

What is the p-value?

What is the test statistic OR odds ratio?

In [360…

```
# TODO: Implement the appropriate test of the options here in this cell
# HINT: You may have implemented a very similar test above
from scipy.stats import ttest_ind, fisher_exact, chi2_contingency, mannwhitr

# Univariate test: association between oxygen drop (binary) and mortality (b

# Build 2x2 contingency table (rows: survived=0/died=1; cols: no drop=0/drop
y = labels["death_in_stay"].astype(int)
x = filtered_features["oxy_drop"].astype(int)

# filter x and y to only include matched patients
x = x.loc[all_matched_subject_ids]
y = y.loc[all_matched_subject_ids]

tbl = pd.crosstab(y, x).reindex(index=[0,1], columns=[0,1], fill_value=0)
print("Contingency table:\n", tbl, "\n")

# Chi-squared test; if any expected cell < 5, fall back to Fisher's exact
chi2, p_chi, dof, expected = chi2_contingency(tbl.values, correction=False)

# Approximate odds ratio from 2x2
a, b = tbl.loc[1,1], tbl.loc[1,0]
c, d = tbl.loc[0,1], tbl.loc[0,0]
or_est = (a*d) / (b*c) if b*c != 0 else float("inf")
p = p_chi
test_used = "Chi-squared"

print(f"{test_used} p-value: {p:.3g}")
print("odds ratio:", or_est)
```

```
Contingency table:
 oxy_drop        0    1
death_in_stay
0             624  585
1             138  177

Chi-squared p-value: 0.0136
odds ratio: 1.3681159420289855
```

`6.5` (`5 points`) Bias Variance Tradeoff in Caliper Selection

"Bias" and "Variance" are concepts that are integral to both machine learning and causal inference, though their implications can be somewhat different depending on the context. (NOTE: Both "Bias" and "Variance" are very overloaded terms! The definitions change depending on the context.)

In the context of discussing a "bias" vs "variance" tradeoff:

- **Bias** refers to the difference between the average prediction of our model and the correct value we are trying to predict. A model is said to have high bias if the model oversimplifies the "true" relationship between the predictors and outcome, generally referred to as "being underfit". Models with high bias often have systematic errors in predictions (errors largely occuring in the same direction with similar magnitude) as the estimates are "biased" away from the true values.

- **Variance** refers to the variability of our model's predictions. A model is said to have high variance if small changes to the training data result in large changes in the predictions. Models with high variance are often **overfit** to the training data, and do not generalize well to new data.

You may find this article from "Towards Data Science" helpful for exploring this concept further: Understanding the Bias-Variance Tradeoff

In our propensity analysis, we could have had a stricter caliper and enforced that patients had to be more similar to be matched.

What would have been the tradeoff in terms of the **bias and **variance** of utilizing a smaller caliper on our models predictions?**

Using a smaller caliper is like being pickier about who counts as a match or a “twin” to include in the study. This will benefit bias. Our cohort will have better balance and thus less confounding. Nevertheless, we sacrifice more patients who can’t find close matches. Fewer matched pairs means higher variance and wider confidence intervals. Thus, our estimate will apply mainly to the well-overlapped subset rather than the whole cohort.

`6.6` (`5 points`) Causal Validity and the Problem of Post-Treatment Variables

Think back to the previous assignment where we produced the dataset we are working with now. All of the information we have for each patient is from before the **index_time**, which by definition is before the **exposure_time**.

Heres a hypothetical: Your friend Marc wants to include the "total cost of the patient's hospitalization" as an additional feature in our dataset, and utilize it to fit our propensity model. After looking into his idea, you realize that the "total cost" feature is generated **after** the exposure_time for each patient.

What would be the problem with including this feature in our propensity model? Would our inference still be causally valid? Why or why not?

Our inference would not be causally valid. "Total-cost" is a post treatment variable (realized only after exposure). Propensity scores must be build only from pre-exposure covariates. By doing this, we are containminating the model by let it predict probbaility for exposure depending variables from the exposure. This could destroy the overlap as our propensity model is invalid as it is trained on variables dependent on what we are trying to predict, ruining our causal inference pathway.

`6.7` (`5 points`) Limitations and Remaining Sources of Bias

What are some sources of **bias** (still taking about bias in the sense of "bias" vs "variance" tradeoff) that our propensity score analysis cannot correct for?

Here are some examples of bias that our propoensity score analysis could not correct for

- Unmeasured confounding: variables affecting both exposure and outcome that aren’t include in propsenity score calculation (e.g., disease severity not captured in features)
- Measurement error/misclassification: noisy exposure, outcome, or covariates
- Positivity/overlap violations: regions where some groups almost never receive one exposure (near-perfect prediction).

`7` (`10 points Extra Credit`) **Optional Reflection Exercise R vs Python**

Find a BIOMEDIN 215 classmate who completed Assignment 5 in `R`.

You'll notice that the results of some of their statistical analyses are quite different from yours. Why do you think that is?

> Hint: there is a very specific design decision early on in the assignment that makes some results differ substantially between the two versions -- if you can find it, we'll give you 10 bonus points!

YOUR ANSWER HERE

Feedback (`0 points`)

Please fill out the following feedback form so we can improve the course for future students!

Submission Instructions ✅

There are two files you must submit for this assignment:

1. A `PDF` of this notebook.

- **Please clear any large cell outputs from executed code cells before creating the PDF.**
  - Including short printouts is fine, but please try to clear any large outputs such as dataframe printouts. This makes it easier for us to grade your assignments!
- To export the notebook to PDF, you may need to first create an HTML version, and then convert it to PDF.

2. A `zip` file containing your code generated by the provided `create_submission_zip.py` script:

- Open the `create_submission_zip.py` file and enter your SUNet ID where indicated.
- Run the script via `python create_submission_zip.py` to generate a file titled `<your_SUNetID>_submission_A5.zip` in the root project directory.