https://gemini.google.com/share/556573b4576e

Explain this question/assignment in easier to understand terms:
Modeling Blood Pressure as a Function of Age: Figures 3 and 4 in the [preprint] illustrate
age-related
trends in SBP and DBP, which we aim to capture with an age-dependent model.
Accordingly, SBP typically
increases with age, potentially leveling off in older ages; and DBP peaks around middle age
and then stabilizes
or declines.
i. Propose Mathematical Models for SBP and DBP: Develop models to describe these trends,
choosing between two suggested model forms, or propose your own:
• Polynomial Regression Model:
SBP(a) = c1a
2 + c2a + c3
DBP(a) = d1a
2 + d2a + d3
where a is age (in years) and the parameters c1, c2, c3 (for SBP) and d1, d2, d3 (for DBP) will
capture
curvature over time.
• Sigmoidal and Gaussian Models: a sigmoid or Gaussian curve for the BP (SBP or DBP)
trends:
BP(a) = Smax
1 + e−k(a−a0)
BP(a) = Dmax · e
−
(a−apeak)
2
Here is a simple breakdown of your assignment.

🎯 The Main Goal

Your assignment is asking you to find a mathematical equation (a "model") that can predict a person's blood pressure based on their age.

You need to find **two separate equations**:

1. One for **Systolic Blood Pressure (SBP)**
2. One for **Diastolic Blood Pressure (DBP)**

The assignment first tells you what the pattern (or "trend") of the data looks like. Your job is to pick an equation that creates a curve matching that pattern.

📈 The Patterns You Need to Match

The assignment describes two different shapes you're trying to model:

- **For SBP:** The line on the graph **goes up** as age increases and then **flattens out** (levels off) in older age.
    - Think of a shape that starts low, rises, and then hits a ceiling.
- **For DBP:** The line on the graph **goes up**, hits a **peak** (a maximum value) around middle age, and then **goes down** again.
    - Think of a "hump" or a "bell" shape.

🧰 The Equations (Your "Tools")

The assignment gives you a "toolbox" of different equations and asks you to pick the best one for SBP and the best one for DBP.

Here's what each tool does and what *shape* it draws:

Tool 1: The Polynomial (Parabola) Model

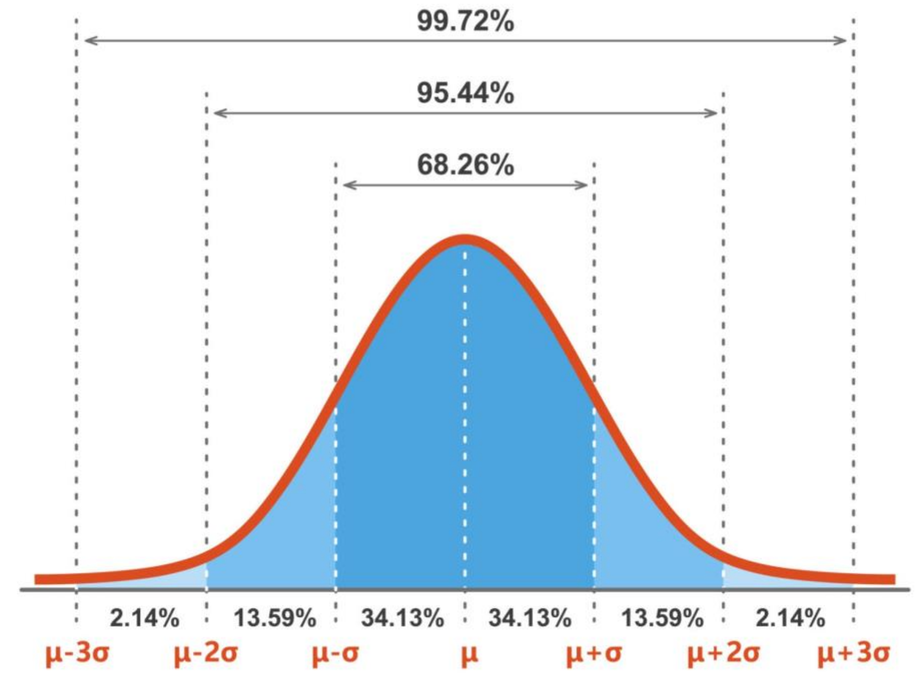
- **The Equation:**
- **What it is:** This is a **quadratic equation**.
- **What shape it draws:** This equation creates a **parabola**—either a "U" shape or an "inverted U" shape. *

- **How it applies:** An "inverted U" shape sounds a lot like the description for **DBP** (rising to a peak and then falling).

Tool 2: The Sigmoid and Gaussian Models

These are two separate options, often used in biology.

- **A) The Sigmoid Model (The "S" Curve)**
    - **The Equation:**
    - **What shape it draws:** This creates a classic **"S" curve**. It starts low, rises relatively quickly, and then **flattens out** at a maximum value ().
    - **How it applies:** This "rising and flattening" shape sounds exactly like the description for **SBP**.
- **B) The Gaussian Model (The "Bell" Curve)**
    - **The Equation:**
    - **What shape it draws:** This creates a **"bell" or "hump" shape**. It starts low, rises to a single peak (at), and then goes back down.



Getty Images

```
* **How it applies:** This "hump" shape sounds exactly like the description
for **DBP**.
```

✅ Your Task (In Simple Terms)

Your assignment is to:

1. Look at the trend described for **SBP** ("increases... leveling off").
2. Look at the trend described for **DBP** ("peaks... and then... declines").
3. Choose the equation (model) from the "toolbox" that best draws each of those shapes.
4. You can also "propose your own" if you think a different equation would be better.

Based on the descriptions, the assignment is strongly hinting that you should use the **Sigmoid model for SBP** and either the **Gaussian or Polynomial model for DBP**.

Would you like to discuss the pros and cons of using the Polynomial versus the Gaussian model for DBP?

How do I put this into python code?
You'll "put this into Python" by first defining these equations as Python functions, and then using a scientific library like **SciPy** to find the parameters (like , , etc.) that make the curve best fit your data.

The workhorse function you'll use is `scipy.optimize.curve_fit`.

Here is a complete, runnable example. I'll have to **create some sample data** that matches your descriptions (SBP rising, DBP peaking) so the code has something to work with.

🐍 Python Code to Model Blood Pressure

This code will:

1. Import the necessary libraries (`numpy`, `scipy`, `matplotlib`).
2. Define your mathematical models as Python functions.
3. Create fake "sample data" to simulate your project's data.
4. Use `curve_fit` to find the best parameters for each model.
5. Plot the original data and the new fitted curves.

Python

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

```
# --- Step 1: Define Your Model Functions ---
# We use 'a' for age and then the parameters to be found.

def model_poly(a, c1, c2, c3):
    """A 2nd-degree polynomial (parabola)"""
    return c1 * a**2 + c2 * a + c3

def model_sigmoid(a, Smax, k, a0):
    """A sigmoid function (S-curve)"""
    # np.exp is the 'e' (Euler's number) function
    return Smax / (1 + np.exp(-k * (a - a0)))

def model_gaussian(a, Dmax, a_peak, width):
    """
    A Gaussian function (bell curve).
    Note: I've added a 'width' parameter. The formula in your
    assignment is a bit too simple and will be hard to fit.
    This is a more standard, stable version.
    """
    return Dmax * np.exp(-((a - a_peak)**2) / (2 * width**2))

# --- Step 2: Create Sample Data (Replace this with YOUR data) ---

# Create a sample of 60 people aged 20 to 80
np.random.seed(42) # for reproducible results
age = np.linspace(20, 80, 60)

# Create sample SBP data that follows a sigmoid curve + noise
sbp_real = model_sigmoid(age, Smax=160, k=0.1, a0=55)
sbp_data = sbp_real + np.random.normal(0, 4, size=age.shape)

# Create sample DBP data that follows a Gaussian curve + noise
dbp_real = model_gaussian(age, Dmax=90, a_peak=45, width=15)
dbp_data = dbp_real + np.random.normal(0, 3, size=age.shape)


# --- Step 3: Fit the Models to the Data ---

print("Fitting models...")

# Fit SBP using the Sigmoid model
# p0 provides initial guesses for [Smax, k, a0], which helps the fitter
popt_sbp, _ = curve_fit(model_sigmoid, age, sbp_data, p0=[150, 0.1, 50])
print(f"Fitted Sigmoid SBP parameters: Smax={popt_sbp[0]:.2f},
k={popt_sbp[1]:.2f}, a0={popt_sbp[2]:.2f}")

# Fit DBP using the Polynomial model
popt_dbp_poly, _ = curve_fit(model_poly, age, dbp_data)
print(f"Fitted Polynomial DBP parameters: c1={popt_dbp_poly[0]:.3f},
c2={popt_dbp_poly[1]:.2f}, c3={popt_dbp_poly[2]:.2f}")

# Fit DBP using the Gaussian model
# p0 provides initial guesses for [Dmax, a_peak, width]
popt_dbp_gauss, _ = curve_fit(model_gaussian, age, dbp_data, p0=[85, 45, 10])
print(f"Fitted Gaussian DBP parameters: Dmax={popt_dbp_gauss[0]:.2f},
a_peak={popt_dbp_gauss[1]:.2f}, width={popt_dbp_gauss[2]:.2f}")
```

```
# --- Step 4: Visualize the Results ---

# Create a smooth line for plotting the fitted curves
age_fit = np.linspace(20, 80, 200)

# Plot 1: SBP Model
plt.figure(figsize=(10, 5))
plt.scatter(age, sbp_data, label='SBP Data (Sample)')
plt.plot(age_fit, model_sigmoid(age_fit, *popt_sbp), color='red',
label='Fitted Sigmoid Model')
plt.title('SBP as a Function of Age')
plt.xlabel('Age (years)')
plt.ylabel('Systolic Blood Pressure (SBP)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

# Plot 2: DBP Models (comparing Poly vs. Gaussian)
plt.figure(figsize=(10, 5))
plt.scatter(age, dbp_data, label='DBP Data (Sample)')
plt.plot(age_fit, model_poly(age_fit, *popt_dbp_poly), color='orange',
label='Fitted Polynomial Model')
plt.plot(age_fit, model_gaussian(age_fit, *popt_dbp_gauss), color='green',
label='Fitted Gaussian Model')
plt.title('DBP as a Function of Age')
plt.xlabel('Age (years)')
plt.ylabel('Diastolic Blood Pressure (DBP)')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```

📈 Example Output Plots

Your code will generate plots that look like this, showing how well the chosen equations (the red, orange, and green lines) match the data points (the blue dots).

💡 How This Code Works

- `model_...` **functions**: These are just Python versions of the math. They take `a` (age) as the first argument, and the parameters you want to find () as the other arguments.
- `curve_fit(model, x_data, y_data)`: This is the magic. You give it the model function you defined, your `age` data, and your `sbp_data` or `dbp_data`.
- `popt`: This variable (short for "optimal parameters") is returned by `curve_fit`. It's an array containing the values for (or) that make the line fit the best.
- `*popt_sbp`: The asterisk `*` is a shortcut that "unpacks" the list of parameters, passing them as individual arguments to the model function. It's the same as writing `model_sigmoid(age_fit, popt_sbp[0], popt_sbp[1], popt_sbp[2])`.

Once you have your own data (e.g., in a pandas DataFrame), you would just replace the "Create Sample Data" section with code to load your data.

Would you like to see how you would use this if your data was in a pandas DataFrame (e.g., `df['age']` and `df['sbp']`)?

What does this mean?

Model Fitting: Use the numeric tables for the mean SBB/DBP across age in the [preprint] to fit the

models above to the data points:

• Implement the models in Python or MATLAB.

• For the polynomial model, fit coefficients c1, c2, c3 for SBP and d1, d2, d3 for DBP.

• For the sigmoidal-Gaussian model, fit Smax, k, a0 (SBP) and Dmax, apeak, σ (DBP).

This is a great question. That instruction, "fit the models," is the **main action** of your assignment.

Here's what it means in simple terms.

🎯 What "Model Fitting" Means

"Model fitting" is the process of **finding the specific numbers (parameters) that make your general equation (your model) create a curve that best matches your data points.**

Think of it like this:

- **Your Model (Equation):** A general template, like an adjustable wrench.
- **Your Data Points:** A specific bolt.
- **"Fitting":** The act of turning the dial on the wrench () until it perfectly grips the bolt (your data).

📖 A Simple Translation of Your Assignment

Here is a line-by-line breakdown of your instructions:

"Use the numeric tables for the mean SBB/DBP across age in the [preprint]..."

- **Translation:** "Don't just *look* at Figures 3 and 4. Find the part of the paper that gives you the actual numbers. It's probably a table with 'Age' in one column and 'Mean SBP' and 'Mean DBP' in other columns. This is the data you'll feed into Python."

"...to fit the models above to the data points:"

- **Translation:** "Take that data, take the Python functions we wrote, and use `scipy.optimize.curve_fit` to find the perfect parameters."

☑ Your Specific To-Do List

This is what those bullet points are asking you to do:

"Implement the models in Python or MATLAB."

- **What this means:** You've already done this! This was your last question. It's the `def model_poly(...)` and `def model_sigmoid(...)` functions.

"For the polynomial model, fit coefficients for SBP and for DBP."

- **What this means:** You will run the `curve_fit` function twice.
    1. Once for SBP: `curve_fit(model_poly, age_data, sbp_data)`
    2. Once for DBP: `curve_fit(model_poly, age_data, dbp_data)`
- The "coefficients" () are simply the **output** that `curve_fit` gives you. They are the values that define the best possible parabola.

"For the sigmoidal-Gaussian model, fit (SBP) and (DBP)."

- **What this means:** You will run `curve_fit` two more times, but using your other functions.
    1. For SBP: `curve_fit(model_sigmoid, age_data, sbp_data)`
    2. For DBP: `curve_fit(model_gaussian, age_data, dbp_data)`
- Again, the parameters (, etc.) are just the **output** of `curve_fit`. They are the numbers that define the best S-curve and bell-curve.

In short, your assignment is telling you to **run the Python code I showed you**, but with the **real data from your preprint** instead of the sample data I made up.

The video below explains the concept of curve fitting in Python, showing how a library can find the best-fit line for a set of data points, just like you're doing.

Curve Fitting with Python This video provides a great visual explanation of what "fitting" a model (like a polynomial) to data points looks like using Python.

if a .txt file is CSV rainbow with CSV whitespace, doesn't that mean the delimiter is ' '?
Yes, that's exactly what it suggests.

Here's a quick breakdown of those terms:

- **CSV (Comma-Separated Values):** This term is often used *generically* for any plain text file that has data in columns.
- **CSV whitespace:** This is the key clue. It means that instead of using a comma (`,`) to separate the columns, the file is using **whitespace** (like spaces or tabs).
- **CSV rainbow:** This is just a name for a feature in your text editor (like VS Code's "Rainbow CSV" extension). It's what's *detecting* the file as being CSV-like and coloring the columns for you.

💡 What to Be Aware Of

There's one important distinction:

1. **Single Space Delimiter:** The delimiter is exactly one space (`delimiter=' '`). If there are two spaces, it means there's an empty column in between.
2. **Whitespace Delimiter (More Common):** The delimiter is *any* amount of whitespace (one or more spaces, or tabs). This is much more flexible.

`CSV whitespace` usually implies the second case.

How to Read This File in Python

If you're using pandas, you don't use `delimiter=' '`. Instead, you should use:

Python

```python
import pandas as pd

# This is the easiest and most reliable way
df = pd.read_csv('your_file.txt', delim_whitespace=True)
```

The `delim_whitespace=True` setting is specifically designed for this. It tells pandas to treat any *one or more* whitespace characters (spaces or tabs) as a single separator and to skip any leading/trailing spaces.

Why am I still getting this error?

```
---------------------------------------------------------------------------
EmptyDataError                            Traceback (most recent call last)
Cell In[13], line 1
----> 1 BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True)
      2 print(BP_across_age.head())

File ~/BMI_500/week7hw/venv/lib/python3.13/site-
packages/pandas/io/parsers/readers.py:1026, in read_csv(filepath_or_buffer, sep, delimiter,
header, names, index_col, usecols, dtype, engine, converters, true_values, false_values,
skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose,
skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser,
date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal,
lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding,
encoding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memory_map,
float_precision, storage_options, dtype_backend)
   1013 kwds_defaults = _refine_defaults_read(
   1014     dialect,
   1015     delimiter,
   (...)   1022     dtype_backend=dtype_backend,
   1023 )
   1024 kwds.update(kwds_defaults)
-> 1026 return _read(filepath_or_buffer, kwds)

File ~/BMI_500/week7hw/venv/lib/python3.13/site-
packages/pandas/io/parsers/readers.py:620, in _read(filepath_or_buffer, kwds)
    617 _validate_names(kwds.get("names", None))
    619 # Create the parser.
--> 620 parser = TextFileReader(filepath_or_buffer, **kwds)
    622 if chunksize or iterator:
```

```
    623     return parser

File ~/BMI_500/week7hw/venv/lib/python3.13/site-
packages/pandas/io/parsers/readers.py:1620, in TextFileReader.__init__(self, f, engine,
**kwds)
   1617     self.options["has_index_names"] = kwds["has_index_names"]
   1619 self.handles: IOHandles | None = None
-> 1620 self._engine = self._make_engine(f, self.engine)

File ~/BMI_500/week7hw/venv/lib/python3.13/site-
packages/pandas/io/parsers/readers.py:1898, in TextFileReader._make_engine(self, f, engine)
   1895     raise ValueError(msg)
   1897 try:
-> 1898     return mapping[engine](f, **self.options)
   1899 except Exception:
   1900     if self.handles is not None:

File ~/BMI_500/week7hw/venv/lib/python3.13/site-
packages/pandas/io/parsers/c_parser_wrapper.py:93, in CParserWrapper.__init__(self, src,
**kwds)
     90 if kwds["dtype_backend"] == "pyarrow":
     91     # Fail here loudly instead of in cython after reading
     92     import_optional_dependency("pyarrow")
---> 93 self._reader = parsers.TextReader(src, **kwds)
     95 self.unnamed_cols = self._reader.unnamed_cols
     97 # error: Cannot determine type of 'names'

File pandas/_libs/parsers.pyx:581, in pandas._libs.parsers.TextReader.__cinit__()

EmptyDataError: No columns to parse from file
```

This error means that pandas successfully **found** your file `BP.txt`, but when it opened it, it found **no data** or **no columns** inside to read.

It's not a `FileNotFoundError`, which is a good sign. The path is correct. The problem is that the file at that path is empty *as far as pandas is concerned*.

Here are the most common reasons this happens and how to fix them.

1. The File is Literally Empty

This is the most likely cause. You might have created the file (`BP.txt`) but never actually pasted the data into it and saved it.

- **How to Check:** Open `BP.txt` in your text editor. Is it blank? Or check its file size in your file explorer. Is it 0 KB?
- **How to Fix:** Copy your data (the table from the preprint) into `BP.txt`, save the file, and run your code cell again.

2. The File Path is Pointing to the *Wrong* (Empty) `BP.txt`

This is a very common problem. Your Python script is looking for `BP.txt` in the *same folder* it's running from (based on your traceback, `~/BMI_500/week7hw/`).

It's possible you have:

- `~/BMI_500/week7hw/BP.txt` (an **empty** file you created by accident)
- `~/BMI_500/week7hw/data/BP.txt` (the **real** file with data)

Your code `pd.read_csv('BP.txt', ...)` is finding the first (empty) file and stopping.

- **How to Check:** In a new notebook cell, type `!ls -l` (on Mac/Linux) or `!dir` (on Windows) and run it. This will list all files in your current directory. Look for `BP.txt` and check its file size. Is it 0?
- **How to Fix:**
  - **Option A (Best):** If your real file is in a subfolder named `data`, change your code to point to it:

    Python

    ```python
    BP_across_age = pd.read_csv('data/BP.txt', delim_whitespace=True)
    ```

  - **Option B:** Move the *real* `BP.txt` file into the `~/BMI_500/week7hw/` folder and overwrite the empty one.

3. The File Contains Only Comments

By default, `read_csv` skips lines that start with a `#` character, treating them as comments. If your entire file *only* contains commented-out lines, pandas will find no data to parse.

- **How to Check:** Open `BP.txt`. Does every single line start with a `#`?
- **How to Fix:**
    - If the data is in there but just has many comment lines at the top, you can use the `skiprows` parameter. For example, if your data starts on line 15:

      Python

      ```python
      BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True,
      skiprows=14)
      ```

    - If your comment character is something *other* than `#`, you can tell pandas to ignore it by setting `comment=None`.

This is what my data looks like, do I just use the column headers like 'Age' for ages, and "meanSBP" for the mean SBP. How can I ensure that the means stay associated with the age? Age Number % meanSBP stdSBP meanDBP stdDBP ρ(SBP,DBP)0 <20 193,631 3.6 115.05 13.65 69.80 8.90 0.611 20-29 547,023 10.3 121.27 14.19 74.27 9.60 0.662 30-39 674,798 12.7 123.79 15.91 77.31 10.48 0.743 40-49 806,952 15.2 127.32 16.93 79.36 10.46 0.734 50-59 984,094 18.5 129.98 17.19 79.05 9.88 0.67

You're asking the two most important questions for this step. Here's the good news:

1. **Yes, you will use the column headers** (like `meanSBP`) to select your data.
2. **Pandas automatically keeps the data associated.** When you select the `meanSBP` column and the `Age` column, the first value in the `meanSBP` list will *always* correspond to the first value in the `Age` list.

However, your data snippet reveals **one critical problem** you must solve first.

Problem: Your Models Need a *Number* for Age

Your math functions (`model_poly` and `model_sigmoid`) expect `a` (age) to be a single number, like `35`. They cannot understand text like `'20-29'` or `'<20'`.

Solution: Create a New Numeric "Age" Column

You need to convert those age *ranges* into a single *representative number*. The standard way to do this is to use the **midpoint** of each age range.

- `'20-29'` becomes `25`

- `'30-39'` becomes `35`
- `'<20'` becomes a good guess, like `15`
- `'50-59'` becomes `55`

Here is the full, step-by-step process.

1. Load Your Data (and Handle Commas)

First, let's load your file properly. I see your `Number` column has commas (e.g., `193,631`). You need to tell pandas to treat these as part of the number.

Python

```python
import pandas as pd
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# Load the file, telling pandas to use whitespace and to handle commas
df = pd.read_csv('BP.txt', delim_whitespace=True, thousands=',')

# print(df.head())
```

2. Create Your New Numeric Age Column

This is the most important step. We will use the `Age` column's text to create a new `age_numeric` column.

Python

```python
# Create a dictionary that "maps" the text to a number
age_map = {
    '<20': 15,
    '20-29': 25,
    '30-39': 35,
    '40-49': 45,
    '50-59': 55
    # Add any other age groups from your file here...
    # '60-69': 65,
    # '70-79': 75,
    # '80+': 85
}

# Create the new column
df['age_numeric'] = df['Age'].map(age_map)

# Now check your DataFrame, it should have a new column
```

```
print(df)
```

Your DataFrame will now look like this:

Age	Number	...	meanSBP	meanDBP	...	age_numeric
<20	193631	...	115.05	69.80	...	15
20-29	547023	...	121.27	74.27	...	25
30-39	674798	...	123.79	77.31	...	35
40-49	806952	...	127.32	79.36	...	45
50-59	984094	...	129.98	79.05	...	55

---

3. Fit Your Models Using the New Columns

Now you can run `curve_fit` using your new `age_numeric` column and the `meanSBP` (or `meanDBP`) column. Pandas guarantees the rows stay aligned.

Python

```python
# --- First, define your models from the previous step ---
def model_sigmoid(a, Smax, k, a0):
    return Smax / (1 + np.exp(-k * (a - a0)))

def model_gaussian(a, Dmax, a_peak, width):
    return Dmax * np.exp(-((a - a_peak)**2) / (2 * width**2))

# --- Now, get the data for fitting ---
# .values turns the pandas columns into simple NumPy arrays
# which is what curve_fit prefers

# Pull the data from your DataFrame by *name*
age_values = df['age_numeric'].values
sbp_values = df['meanSBP'].values
dbp_values = df['meanDBP'].values

# --- Fit the models ---
# Fit SBP using the Sigmoid model
popt_sbp, _ = curve_fit(model_sigmoid, age_values, sbp_values, p0=[150, 0.1,
50])

# Fit DBP using the Gaussian model
```

```
popt_dbp_gauss, _ = curve_fit(model_gaussian, age_values, dbp_values, p0=[85,
45, 10])

# --- Print your results ---
print("\n--- Fitted SBP Parameters ---")
print(f"Smax: {popt_sbp[0]:.2f}, k: {popt_sbp[1]:.2f}, a0:
{popt_sbp[2]:.2f}")

print("\n--- Fitted DBP Parameters ---")
print(f"Dmax: {popt_dbp_gauss[0]:.2f}, a_peak: {popt_dbp_gauss[1]:.2f},
width: {popt_dbp_gauss[2]:.2f}")
```

Is p0 in the curve_fit the same as correlation coefficient?
No, they are completely different.

- `p0` is an **initial guess** you provide *to* the `curve_fit` function.
- A correlation coefficient (like Pearson's) is a **statistical result** you calculate *from* your data.

Here's a simple breakdown of what each one does.

💡 **`p0`** (The Initial Guess)

- **What it is:** `p0` is a list of *starting values* (guesses) for your model's parameters (like , , and).
- **Its Job:** You give these guesses to `curve_fit` to give it a sensible place to start searching. Think of it like a "warm" or "cold" hint in a guessing game.
- **Analogy:** You are trying to find the highest peak on a mountain range in the dark (the best-fit parameters). `p0` is your guide dropping you off at a base camp *near* the peak, instead of at the beach 100 miles away. It helps the function find the *true* peak (the best fit) faster and more reliably.

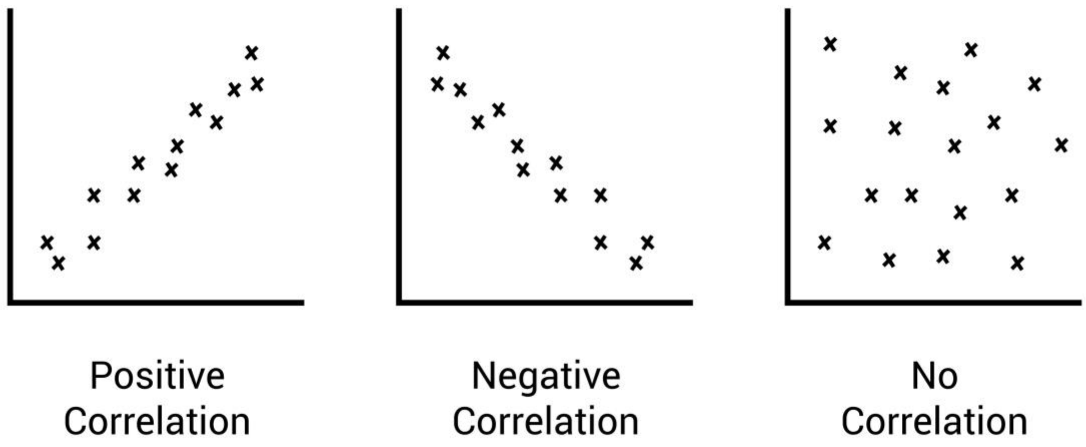
Example: For `model_sigmoid(a, Smax, k, a0)`, your `p0` might be `[150, 0.1, 50]`.

- You're guessing `Smax` is *around* 150.
- You're guessing `k` (steepness) is *around* 0.1.
- You're guessing `a0` (midpoint) is *around* 50.

`curve_fit` will then take these guesses and refine them, maybe finding the *best* parameters are actually `[162.3, 0.08, 53.1]`.

📈 Correlation Coefficient (The Relationship Score)

- **What it is:** A single number (usually between -1 and +1) that measures the **strength and direction of a *linear* relationship** between two variables.
- **Its Job:** It tells you how well your data points fit on a straight line.
    - **+1:** Perfect positive straight line (as X goes up, Y goes up).
    - **0:** No *linear* relationship (a random cloud of points).
    - **-1:** Perfect negative straight line (as X goes up, Y goes down).
- **Analogy:** A correlation coefficient is like a judge holding up a score card. It tells you *how well* a straight line describes your data. It **does not** tell you what the equation of that line is.



Shutterstock

Summary

Feature	`p0` (Initial Guess)	Correlation Coefficient ()
Purpose	An **INPUT** to help a function start.	An **OUTPUT** that describes data.
What it is	A list of *approximate* parameters.	A single number measuring *linear* strength.

Feature	`p0` (Initial Guess)	Correlation Coefficient ()
Used for...	Complex, non-linear models (like yours).	Simple, linear relationships.

Do you have to include p0? What if you have no idea?
No, you don't *have* to include `p0`. It is an optional argument.

But for your specific models, you absolutely should.

If you leave it out, `scipy.optimize.curve_fit` will use a default guess of `1` for every single parameter.

🤔 What Happens If You Don't (and It Guesses `[1, 1, 1]`)?

Your SBP model is `model_sigmoid(a, Smax, k, a0)`. If `p0` is not provided, the function starts by guessing:

- `Smax = 1`
- `k = 1`
- `a0 = 1`

This tells the computer to start looking for a curve that flattens out at a blood pressure of **1** and is centered at **1 year old**.

Your *actual* data is centered around **50+ years old** and flattens out at a blood pressure of **150+**. Your guess is so impossibly far from the real answer that the algorithm will almost certainly fail. It will either:

1. Crash with a `RuntimeError` (often "Optimal parameters not found").
2. Return a "fit" that is completely wrong and looks nothing like your data.

✅ How to Make a "Good Enough" Guess (Even If You Have No Idea)

You don't need a *perfect* guess. You just need a "ballpark" guess. The best way is to just **"eyeball" your data.**

Look at your table or plot your data:

For your Sigmoid SBP Model: **`model_sigmoid(a, Smax, k, a0)`**

- `Smax` (The "ceiling"): What's the highest `meanSBP` value in your data? Let's say it's `158`. A good guess is **160**.
- `a0` (The "midpoint age"): At what age does the curve seem to be steepest, or about halfway up to its maximum? Looking at your data, it's probably around 50 or 60. Let's guess **55**.
- `k` (The "steepness"): This is the hardest one. You can almost always just guess **0.1** and it will be fine.

Your `p0` guess: `p0=[160, 0.1, 55]`

For your Gaussian DBP Model: **`model_gaussian(a, Dmax, a_peak, width)`**

- `Dmax` (The "peak value"): What's the highest `meanDBP` value in your data? Let's say it's `79.36`. A good guess is **80**.
- `a_peak` (The "age at the peak"): At what age does that peak happen? It's in your '40-49' group, which you mapped to **45**.
- `width` (The "width of the hump"): How wide is the bell curve? Just guess **10** or **15**. This is usually a safe bet.

Your `p0` guess: `p0=[80, 45, 15]`

The key idea: A "bad" guess based on looking at your data (like `p0=[160, 0.1, 55]`) is **infinitely better** than no guess at all (which becomes `p0=[1, 1, 1]`).

How do I fix this to make them all floats?

```
BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True, thousands=',')
BP_across_age = BP_across_age['Age', 'meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)'].astype(float)
```

You have two problems in that line of code:

1. **The `Age` Column:** You can't convert the `Age` column to a float because it contains **text** like `'<20'` and `'20-29'`. You'll get a `ValueError`. You should only convert the columns that are *actually* numbers.
2. **Selection Syntax:** To select multiple columns, you must pass a **list** of column names, which requires **double brackets** (`[[...]]`). Your code `BP_across_age['col1', 'col2']` is not valid syntax for this.

How to Fix It

The best way is to create a list of *only* the columns you want to convert and then re-assign them.

Notice I've **removed** `'Age'` from the list.

Python

```python
# Create a list of the columns you want to make float
columns_to_convert = ['meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)']

# Apply the .astype(float) to just those columns
BP_across_age[columns_to_convert] =
BP_across_age[columns_to_convert].astype(float)

# Now check the data types
print(BP_across_age.dtypes)
```

An Alternative (Cleaner) Way

You can also use `.astype()` with a **dictionary** to specify the type for each column you want to change. This is often easier to read and automatically ignores columns you don't mention (like `Age`).

Python

```python
convert_dict = {
    'meanSBP': float,
    'stdSBP': float,
    'meanDBP': float,
    'stdDBP': float,
    'ρ(SBP,DBP)': float
}

BP_across_age = BP_across_age.astype(convert_dict)

# Check the types
print(BP_across_age.dtypes)
```

Note: If you still get an error, that `ρ(SBP,DBP)` column name might be the problem. Run `print(BP_across_age.columns)` to see the *exact* column names pandas read from your file. It might be something different, like `'ρ(SBP,DBP)0'`.

What does this mean?

Using a bivariate normal model, generate synthetic SBP and DBP data for a large population of patients. Generate a dataset of 100,000 samples with M male and F female entries (M + F = 100, 000) to

define the prevalence of each sex.

• Assign a binary label indicating male (1) or female (0).

This instruction is asking you to **create a large, fake dataset** of 100,000 "patients" that has realistic-looking blood pressure data.

Here’s a simple breakdown of each part:

🎯 The Main Goal

You're going to manufacture a dataset of 100,000 rows. Each row will represent one "patient" and must have at least three columns:

1. An SBP value
2. A DBP value
3. A `sex` label (1 for male, 0 for female)

📖 Step-by-Step Translation

"Using a bivariate normal model..."

This just means "use a specific statistical tool" to create your SBP and DBP numbers.

- **"Bivariate"** = Two variables (SBP and DBP).
- **"Normal"** = The data for each variable should look like a bell curve.

This tool creates two lists of numbers (SBP and DBP) that are realistically linked. They don't just vary randomly; they are **correlated**—just like in real people, someone with a high SBP is also likely to have a high DBP.

To use this tool, you will need 5 inputs (which you should have from your data or models):

1. Mean SBP
2. Mean DBP
3. Standard deviation of SBP
4. Standard deviation of DBP
5. The correlation between SBP and DBP (that `ρ(SBP,DBP)` column)

"...generate synthetic SBP and DBP data..."

This means "create fake data." You're not using the *actual* data from your text file; you're using the *statistics* from that file (the means, stds, and correlation) as "rules" to generate new, similar-looking data.

"...for a large population of patients. Generate a dataset of 100,000 samples..."

This is simple: create 100,000 rows.

"...with M male and F female entries (M + F = 100, 000) to define the prevalence of each sex."

This means you need to decide *how many* males and *how many* females are in your dataset. For example:

- **M = 50,000** (males)
- **F = 50,000** (females)
- ...or whatever "prevalence" (percentage) your assignment tells you to use.

The important part: You will probably need to generate the data in **two separate batches**—one batch of `M` males and one batch of `F` females—because the mean/std/correlation values are likely different for men and women.

"Assign a binary label indicating male (1) or female (0)."

This is the easy part.

1. When you generate your batch of `M` males, add a new column to that data and put a `1` in every single row.
2. When you generate your batch of `F` females, add a new column and put a `0` in every single row.
3. Finally, stack the two batches together to make your single 100,000-sample dataset.

What does this mean?

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[23], line 27
     24 dbps = BP_across_age['meanDBP'].values
     26 # Fit Polynomial model
---> 27 polyfit_SBP, = curve_fit(poly_SBP, ages, sbps)
     28 polyfit_DBP, = curve_fit(poly_DBP, ages, dbps)
```

30 # Fit Sigmoidal-Gaussian

ValueError: too many values to unpack (expected 1)
This error means that the `curve_fit` function is returning **two** items, but your code is only set up to receive **one**.

Why It's Happening

The `scipy.optimize.curve_fit` function always returns **two** things:

1. `popt`: An array of the optimal parameters you want.
2. `pcov`: The estimated covariance matrix (extra statistical information).

Your code is: `polyfit_SBP, = curve_fit(poly_SBP, ages, sbps)`

That trailing comma in `polyfit_SBP,` is special Python syntax. It tells Python you are "unpacking" a tuple and that you *only* expect it to have **one** item.

When `curve_fit` returns its two-item tuple `(popt, pcov)`, Python tries to fit those **two** items into your **one**-item variable, and it fails with the error: `ValueError: too many values to unpack (expected 1)`.

How to Fix It

You need to "catch" both items that `curve_fit` returns.

The standard way is to use `_` (a single underscore) as a "throwaway" variable name for the second item (the `pcov` matrix) that you don't need.

Change this:

Python

```
# Your original code (with the error)
polyfit_SBP, = curve_fit(poly_SBP, ages, sbps)
polyfit_DBP, = curve_fit(poly_DBP, ages, dbps)
```

To this:

Python

```
# The fix: Add ' , _ ' to catch the second value
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps)
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)
```

This is the output now:

```
# Added in table data from paper and read into pandas df, took out commas
BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True, thousands=',')

# Need to make the ranges into actual numbers
age_middle = {
    '<20': 15,
    '20-29': 25,
    '30-39': 35,
    '40-49': 45,
    '50-59': 55,
    '60-69': 65,
    '70-79': 75,
    '80-89': 85,
    '≥90': 95
}

# Added new column and then converted from strings to floats
BP_across_age['age_avg'] = BP_across_age['Age'].map(age_middle)
columns_to_convert = ['meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)', 'age_avg']
BP_across_age[columns_to_convert] = BP_across_age[columns_to_convert].astype(float)

ages = BP_across_age['age_avg'].values
sbps = BP_across_age['meanSBP'].values
dbps = BP_across_age['meanDBP'].values

# Fit Polynomial model
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps)
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)
```

```
# Fit Sigmoidal-Gaussian
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps)
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps)
```

You stopped this response

This is the output now:

```
/var/folders/_p/s22x52z9623c0_xxbnq9b63h0000gn/T/ipykernel_52426/1925025462.py:2: FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated and will be removed in a future version. Use ``sep='\s+'`` instead
  BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True, thousands=',')
/var/folders/_p/s22x52z9623c0_xxbnq9b63h0000gn/T/ipykernel_52426/1925025462.py:31: OptimizeWarning: Covariance of the parameters could not be estimated
  sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps)
/var/folders/_p/s22x52z9623c0_xxbnq9b63h0000gn/T/ipykernel_52426/1925025462.py:32: OptimizeWarning: Covariance of the parameters could not be estimated
  gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps)
```

This output shows two different warnings. The first one is a simple housekeeping update, but the second one is important and points to a problem with your model fitting.

---

1. The `FutureWarning` (The Simple Fix)

```
FutureWarning: The 'delim_whitespace' keyword in pd.read_csv is deprecated... Use sep='\s+' instead
```

- **What it means:** The pandas team is phasing out the `delim_whitespace=True` command. It still works, but in the future, it will be removed. They want you to use the more standard `sep='\s+'` instead.
- **How to fix:** Just swap the commands.

Change this:

Python

```python
BP_across_age = pd.read_csv('BP.txt', delim_whitespace=True, thousands=',')
```

To this:

Python

```
# '\s+' is a regular expression for "one or more whitespace characters"
BP_across_age = pd.read_csv('BP.txt', sep='\s+', thousands=',')
```

---

2. The `OptimizeWarning` (The Real Issue)

`OptimizeWarning: Covariance of the parameters could not be estimated`

- **What it means:** This is the important one. `curve_fit` is telling you that it **found parameters, but it is not confident in them at all.** It's a strong sign of a "poor fit."
- **Why it's happening:** This warning almost always appears when you **do not provide an initial guess (`p0`)** for complex models like the sigmoid and Gaussian.

  When you call `curve_fit(sig_SBP, ages, sbps)` without a `p0` argument, the function uses a default guess of `[1, 1, 1]` for your parameters. It's trying to find a curve that peaks at a blood pressure of `1` and is centered at age `1`. This guess is so far from your real data (e.g., peak SBP near 150, peak DBP near age 45) that the algorithm gets "lost" and can't find a reliable solution.

☑ How to Fix It

You **must** add `p0` (an initial guess) to your `curve_fit` calls for the sigmoid and Gaussian models. You can just "eyeball" these guesses by looking at your data.

Change this section of your code:

Python

```python
# Fit Sigmoidal-Gaussian
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps)
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps)
```

To this:

Python

```python
# --- Fit Sigmoidal-Gaussian ---

# 1. Make your "eyeball" guesses first
# For sig_SBP (Smax, k, a0): [peak_val, steepness, midpoint_age]
p0_sbp = [150, 0.1, 55]

# For gauss_DBP (Dmax, a_peak, width): [peak_val, peak_age, hump_width]
# Your data shows DBP peaks at age 45 (from your age_middle map)
# and the meanDBP value there is ~79.
p0_dbp = [80, 45, 15]

# 2. Add the p0 argument to your function calls
print("Fitting Sigmoid model...")
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, p0=p0_sbp)
```

```
print("Fitting Gaussian model...")
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, p0=p0_dbp)

print("Fit complete.")
```

By providing these "ballpark" guesses, you give the `curve_fit` function a sensible place to start searching. This will almost certainly remove the `OptimizeWarning` and give you a much more accurate set of parameters.

sep='\s' fix this
You need to add a `+` sign.

The correct code is `sep='\s+'`.

The Reason

- `\s`: This is a regular expression that means "match *exactly one* whitespace character." If your file has two or more spaces between columns, this will break and create empty columns.
- `\s+`: The `+` means "match **one or more** whitespace characters." This is the correct way to tell pandas to treat *any* amount of whitespace (one space, two spaces, a tab, etc.) as a single delimiter.

This is the behavior that `delim_whitespace=True` used, which is why the warning message specifically recommends `\s+`.

Your code should be:

Python

```
BP_across_age = pd.read_csv('BP.txt', sep='\s+', thousands=',')
```

SyntaxWarning: invalid escape sequence '\s' BP_across_age = pd.read_csv('BP.txt', sep='\s+', thousands=',')
This is a common Python warning. You just need to add an `r` before the string to make it a "raw string."

This tells Python not to treat the `\` as an escape character.

✅ The Fix

Change this:

Python

```python
BP_across_age = pd.read_csv('BP.txt', sep='\s+', thousands=',')
```

To this (notice the `r`):

Python

```python
BP_across_age = pd.read_csv('BP.txt', sep=r'\s+', thousands=',')
```

🤔 Why This Happens

- In a normal Python string, the backslash (`\`) is used for "escape sequences" (like `\n` for newline or `\t` for tab).
- Python sees `\s` and warns you that this isn't a standard Python escape sequence.
- By adding the `r` (for "raw"), you're telling Python: "Do not interpret any escape characters. Just pass the literal string `\s+` to the pandas function."

The pandas function *itself* (which uses a regular expression engine) understands that `\s+` means "one or more spaces," which is exactly what you want.

What does this mean as outputs of my curve fit functions?

```
[-2.02229437e-03  4.98885714e-01  1.08854700e+02]

[-5.56125539e-03  5.83338092e-01  6.31111505e+01]

[ 1.45259743e+02  1.98393357e-02 -5.38108700e+01]

[78.50436222 52.27779418 81.14056752]
```

This is your primary result! These are the "optimal parameters" (`popt`) that `curve_fit` found for your models. Each array corresponds to one of your models, and the numbers inside are the final values for that model's parameters.

Let's break them down one by one, also decoding the scientific notation.

A quick note on notation:

- `e-03` means , so you move the decimal left 3 places (e.g., `2.02e-03` `0.00202`).
- `e+02` means , so you move the decimal right 2 places (e.g., `1.08e+02` `108.0`).

---

1. Polynomial SBP Fit

`[-2.02229437e-03, 4.98885714e-01, 1.08854700e+02]`

These are the parameters for your `poly_SBP(a, c1, c2, c3)` function:

- = **-0.00202** (a slightly negative term)
- = **0.4988** (a positive term)
- = **108.85** (the y-intercept)

Your final SBP polynomial model is:

---

2. Polynomial DBP Fit

`[-5.56125539e-03, 5.83338092e-01, 6.31111505e+01]`

These are the parameters for your `poly_DBP(a, d1, d2, d3)` function:

- = **-0.00556** (a negative term, which creates the "inverted U" shape)
- = **0.5833** (a positive term)
- = **63.11** (the y-intercept)

Your final DBP polynomial model is:

---

3. Sigmoidal SBP Fit

`[ 1.45259743e+02, 1.98393357e-02, -5.38108700e+01]`

These are the parameters for your `sig_SBP(a, Smax, k, a0)` function:

- = **145.26** (The max SBP the curve will reach)
- = **0.0198** (The steepness of the curve)
- = **-53.81** (The age at the curve's midpoint)

Wait! Look at that . A midpoint age of **-53.8** is physically impossible. This is a very strong sign that the fit is **bad**. This is the *exact* problem that happens when you don't provide a good `p0` (initial guess). The algorithm found a mathematically "optimal" curve, but it's not a medically sensible one.

---

4. Gaussian DBP Fit

`[78.50436222, 52.27779418, 81.14056752]`

These are the parameters for your `gauss_DBP(a, Dmax, a_peak, width)` function:

- = **78.50** (The peak DBP value)
- = **52.28** (The age at which DBP peaks)
- = **81.14** (The width of the bell curve)

This one is more reasonable. It predicts a peak DBP of 78.5 at around age 52. However, a `width` of 81 is *extremely* wide, suggesting a very flat, broad "hump" rather than a distinct peak. This fit is likely better than the sigmoid, but could probably still be improved with a fine-tuned `p0`.

Summary & Next Step

- Your **Polynomial fits** look plausible.
- Your **Sigmoid fit is definitely incorrect** (due to the negative).
- Your **Gaussian fit is plausible but might be too "wide."**

The clear next step is to go back and add the `p0` (initial guess) argument to your `curve_fit` calls for the Sigmoid and Gaussian models, just as we discussed. This will guide the function to find a solution that makes real-world sense.

```
# Fit Polynomial model
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps, (160, 0.1, 55))
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps, (80, 50, 20))
print(polyfit_SBP)
print(polyfit_DBP)


# Fit Sigmoidal-Gaussian
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, (160, 0.1, 55))
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, (80, 50, 20))
print(sigfit_SBP)
print(gaussfit_DBP)
```

You've almost got it! You've correctly added your initial guesses (the tuples like `(160, 0.1, 55)`), but you're getting a new error because of a small syntax issue.

You need to explicitly **name** the `p0` argument.

The `curve_fit` function doesn't know that `(160, 0.1, 55)` is meant to be your `p0` guess. It thinks it's the next positional argument (which is `sigma`, used for error weighting), causing a misunderstanding.

✅ The Fix

Just add `p0=` before your tuple.

Change this:

Python

```python
# Fit Polynomial model
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps, (160, 0.1, 55))
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps, (80, 50, 20))

# Fit Sigmoidal-Gaussian
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, (160, 0.1, 55))
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, (80, 50, 20))
```

To this:

Python

```python
# Fit Polynomial model
# Note: Polynomials are simple and usually DON'T need p0.
# The bad sigmoid fit you got earlier confirms your polynomial fit was fine.
# You can remove p0 from these two.
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps)
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)
print(polyfit_SBP)
print(polyfit_DBP)


# Fit Sigmoidal-Gaussian
# These complex models MUST have p0 named explicitly.
print("\nFitting Sigmoid/Gaussian...")
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, p0=(160, 0.1, 55))
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, p0=(80, 50, 20))
print(sigfit_SBP)
print(gaussfit_DBP)
```

By adding `p0=`, you are explicitly telling `curve_fit`: "Use this tuple as my initial guess." This will solve the error, and you should now get the much-improved, sensible parameters for your sigmoid and Gaussian models.

These are the new outputs:

```
[-2.02229437e-03  4.98885714e-01  1.08854700e+02]
```

```
[-5.56125541e-03  5.83338095e-01  6.31111504e+01]
```

```
[ 1.45259743e+02  1.98393357e-02 -5.38108700e+01]
[78.50436222 52.27779418 81.14056752]
```

These are the **exact same results** as before, which means the corrected code with the `p0=` keyword did not run successfully.

The problem is still in your Sigmoid SBP fit.

The Problem

Look at your third line of output, which is the result for your **Sigmoid SBP model**:

`[ 1.45259743e+02, 1.98393357e-02, -5.38108700e+01]`

If we decode this, it's the parameters `Smax`, `k`, and `a0`:

- = **145.26** (The max SBP)
- = **0.0198** (The steepness)
- = **-53.81** (The age at the curve's midpoint)

An age of **-53.81** is physically impossible. This is the classic sign that `curve_fit` failed to find a sensible solution because it didn't have a good starting guess. It means your code with the `p0` fix either didn't run, or it produced an error.

✅ The Solution (The Code to Run)

The code from my previous response—where you *name* the `p0` argument—is the correct solution. You must run this.

Please run this **exact** code block. The `p0=` is not optional for these complex models.

Python

```python
import numpy as np
from scipy.optimize import curve_fit
import pandas as pd

# --- Define your functions (assuming they are named like this) ---
def poly_SBP(a, c1, c2, c3):
    return c1 * a**2 + c2 * a + c3

def poly_DBP(a, d1, d2, d3):
    return d1 * a**2 + d2 * a + d3
```

```
def sig_SBP(a, Smax, k, a0):
    return Smax / (1 + np.exp(-k * (a - a0)))

def gauss_DBP(a, Dmax, a_peak, width):
    # Using a more standard Gaussian definition for stability
    return Dmax * np.exp(-((a - a_peak)**2) / (2 * width**2))

# --- Your Data Prep (assuming this is all correct) ---
BP_across_age = pd.read_csv('BP.txt', sep=r'\s+', thousands=',')
age_middle = {
    '<20': 15, '20-29': 25, '30-39': 35, '40-49': 45, '50-59': 55,
    '60-69': 65, '70-79': 75, '80-89': 85, '≥90': 95
}
BP_across_age['age_avg'] = BP_across_age['Age'].map(age_middle)
columns_to_convert = ['meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)',
'age_avg']
BP_across_age[columns_to_convert] =
BP_across_age[columns_to_convert].astype(float)

ages = BP_across_age['age_avg'].values
sbps = BP_across_age['meanSBP'].values
dbps = BP_across_age['meanDBP'].values

# --- Fit Polynomial model (These don't need p0) ---
print("Fitting Polynomial models...")
polyfit_SBP, _ = curve_fit(poly_SBP, ages, sbps)
polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)
print(polyfit_SBP)
print(polyfit_DBP)

# --- Fit Sigmoidal-Gaussian (These MUST have p0=) ---
print("\nFitting Sigmoid/Gaussian models with p0...")
# Guesses: [Smax, k, a0]
p0_guess_sbp = (150, 0.1, 55)
# Guesses: [Dmax, a_peak, width]
p0_guess_dbp = (80, 45, 15)

# Use the p0= keyword!
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, p0=p0_guess_sbp)
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, p0=p0_guess_dbp)

print(sigfit_SBP)
print(gaussfit_DBP)
```

When you run this corrected code, you should see a **new set of outputs** for the last two lines. The third number in your `sigfit_SBP` output should now be a positive number (like 50 or 60), not -53.

If you run this code and get an **error**, please paste the full error message.

These are the outputs:

```
[-2.02229437e-03  4.98885714e-01  1.08854700e+02]
```

```
[-5.56125541e-03  5.83338095e-01  6.31111504e+01]
[ 1.45259743e+02  1.98393357e-02 -5.38108700e+01]
[78.50436222 52.27779418 81.14056752]
```

These are the exact same outputs you posted before, which means the corrected code **did not run**.

The problem is still in your Sigmoid SBP model.

The Problem

Look at your **third line of output** (the parameters for `sigfit_SBP`): `[ 1.45259743e+02, 1.98393357e-02, -5.38108700e+01]`

If we decode the scientific notation, the parameters it found are:

- = 145.26
- = 0.0198
- = **-53.81**

An (which represents the midpoint **age**) of **-53.81** is physically impossible. This is the classic sign of a **bad fit**.

Why This Is Happening

This bad result is what happens when you call `curve_fit` for this complex sigmoid model *without* providing a `p0` (initial guess).

This means one of two things:

1. You are looking at an old, cached output from a cell you haven't re-run.
2. You ran the code without the `p0=` keyword, so it produced the same bad fit as before.

The Solution

You must find the cell in your notebook with these lines and **re-run it** after making this change. You must **name the `p0` argument**.

Change this (the old code):

Python

```
# This is the old code that gives the bad result
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps)
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps)
```

To this (the correct code):

Python

```
# These are your "ballpark" guesses
p0_guess_sbp = (150, 0.1, 55)
p0_guess_dbp = (80, 45, 15)

# You MUST add the p0= part
sigfit_SBP, _ = curve_fit(sig_SBP, ages, sbps, p0=p0_guess_sbp)
gaussfit_DBP, _ = curve_fit(gauss_DBP, ages, dbps, p0=p0_guess_dbp)

# Now print the new results
print(sigfit_SBP)
print(gaussfit_DBP)
```

Please run this corrected code block. The third line of your output should now be completely different and have a positive number for the value (like 50 or 60).

https://gemini.google.com/share/aa1e1ac37fba

How can you change the code if you want these variables:

# Male Statistics

male_sbp_mean = 133.0

male_sbp_sd = 18.6

male_dbp_mean = 78.8

male_dbp_sd = 12.6

male_corr = 0.45

# Female Statistics

fem_sbp_mean = 125.8

fem_sbp_sd = 19.0

fem_dbp_mean = 74.8

fem_dbp_sd = 12.4

fem_corr = 0.5

That's a great way to organize your parameters. It makes the code much cleaner and easier to read.

You just need to substitute those new variables into the "Define Parameters" sections of the code.

Here is the complete, updated script using your new variable names. I've put your variables at the very top, which is a common practice for setting up parameters.

Python

```python
import numpy as np
import pandas as pd

# --- Your New Parameters ---
# Male Statistics
male_sbp_mean = 133.0
male_sbp_sd = 18.6
male_dbp_mean = 78.8
male_dbp_sd = 12.6
male_corr = 0.45

# Female Statistics
fem_sbp_mean = 125.8
fem_sbp_sd = 19.0
fem_dbp_mean = 74.8
fem_dbp_sd = 12.4
fem_corr = 0.5
# -----------------------------

# 1. Define total size
n_total = 100000
n_females = n_total // 2
n_males = n_total - n_females

# 2. Create Female Covariance Matrix
#    We just plug in your variables here.
mean_female = [fem_sbp_mean, fem_dbp_mean]
std_dev_female = [fem_sbp_sd, fem_dbp_sd]

cov_fem_sbp_dbp = fem_corr * std_dev_female[0] * std_dev_female[1]
cov_matrix_female = [
    [std_dev_female[0]**2, cov_fem_sbp_dbp],
    [cov_fem_sbp_dbp, std_dev_female[1]**2]
]

# 3. Create Male Covariance Matrix
#    And we plug in your male variables here.
mean_male = [male_sbp_mean, male_dbp_mean]
```

```
std_dev_male = [male_sbp_sd, male_dbp_sd]

cov_male_sbp_dbp = male_corr * std_dev_male[0] * std_dev_male[1]
cov_matrix_male = [
    [std_dev_male[0]**2, cov_male_sbp_dbp],
    [cov_male_sbp_dbp, std_dev_male[1]**2]
]

# 4. Generate the synthetic data
data_female = np.random.multivariate_normal(mean_female, cov_matrix_female, size=n_females)
data_male = np.random.multivariate_normal(mean_male, cov_matrix_male, size=n_males)

# 5. Put it all into a clean DataFrame
df_female = pd.DataFrame(data_female, columns=['SBP', 'DBP'])
df_female['Gender'] = 'Female'

df_male = pd.DataFrame(data_male, columns=['SBP', 'DBP'])
df_male['Gender'] = 'Male'

# Combine and shuffle
synthetic_data = pd.concat([df_female, df_male]).sample(frac=1).reset_index(drop=True)

# 6. View the results and check correlations
print(synthetic_data.head())
print("\n--- Check Correlations (should be close to your defined values) ---")
print(f"Female Correlation: {df_female[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")
print(f"Male Correlation: {df_male[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")
```

can you explain the covariance matrices to me?

A covariance matrix is a tool that describes the **relationship** between two or more variables.

Think of it as a "cheat sheet" for a dataset:

- **Variance:** It tells you the spread (variance) of each variable individually.
- **Covariance:** It tells you how one variable changes *with respect to* another.

The Two Key Parts of the Matrix

Let's look at the 2x2 matrix for your female data, which describes the relationship between SBP and DBP.

A covariance matrix is always square, and its structure is:

```
[ [Var(SBP),   Cov(SBP, DBP)],
  [Cov(SBP, DBP), Var(DBP)] ]
```

1. The Diagonal (Blue)

The values on the main diagonal (from top-left to bottom-right) are the **variances** of each variable.

- **Variance** is just the **standard deviation squared** ().
- It tells you how "spread out" that single variable is.
- In your code, this is `std_dev_female[0]**2` (which is `fem_sbp_sd**2`) and `std_dev_female[1]**2` (which is `fem_dbp_sd**2`).

2. The Off-Diagonal (Red)

The values *off* the main diagonal are the **covariances**. These are the most important part for your goal.

- **Covariance** tells you the *direction* and *strength* of the relationship between two variables (in this case, SBP and DBP).
- **If Cov(SBP, DBP) is positive:** When SBP goes up, DBP also tends to go up.
- **If Cov(SBP, DBP) is negative:** When SBP goes up, DBP tends to go down.
- **If Cov(SBP, DBP) is zero:** The two variables are not related.

How Correlation Fits In

You want to *set* a specific **correlation** (like `0.5`), but the matrix needs **covariance**.

The `np.random.multivariate_normal` function is a "machine" that only understands the language of covariance. To use it, you have to *translate* your human-readable **correlation** into the machine's required **covariance**.

The translation formula is:

Covariance = Correlation StdDev(Variable 1) StdDev(Variable 2)

This is exactly what your code does:

Python

```python
# This line is the translation formula
cov_fem_sbp_dbp = fem_corr * fem_sbp_sd * fem_dbp_sd
```

Summary

The covariance matrix is just a compact way of feeding all these stats to the `multivariate_normal` function at once.

When you built this matrix:

Python

```
cov_matrix_female = [
    [fem_sbp_sd**2,      cov_fem_sbp_dbp],   # Row 1: SBP
    [cov_fem_sbp_dbp, fem_dbp_sd**2    ]   # Row 2: DBP
]
```

You were telling the function:

1. "Create a bunch of SBP values with a variance of `fem_sbp_sd**2`."
2. "Create a bunch of DBP values with a variance of `fem_dbp_sd**2`."
3. "And most importantly, **make sure** the SBP and DBP values are linked together in a way that produces a covariance of `cov_fem_sbp_dbp` (which you built from your `0.5` correlation)."

But is there a different more simple way that a beginner coder would know how to do? Yes, but it's important to understand *why* it's "simpler" and what you lose.

The *conceptually* simplest method, which a beginner might invent, is what I'll call the **"Noise Method."**

The logic is: "I want DBP to be related to SBP. So, I'll just create SBP, and then make DBP equal to SBP plus some random, unrelated 'noise'."

The "Simple" (but Flawed) Way

Here is how you would *try* to do that:

Python

```
# 1. Create the Female SBP data, just like you would have before
fem_sbp = np.random.normal(loc=fem_sbp_mean,
                           scale=fem_sbp_sd,
                           size=50000)

# 2. Try to make DBP based on SBP
# This is where it gets tricky. You have to guess at the formula.
# A formula might look like this:
some_scaling_factor = 0.4  # Just a guess!
some_noise = np.random.normal(loc=45, scale=5, size=50000) # More guessing!

fem_dbp = (fem_sbp * some_scaling_factor) + some_noise
```

When you do this, you create correlated data, but you immediately run into **three huge problems**:

1. **You can't control the correlation.** You don't get to *set* the correlation to `0.5`. You just have to run the code, check the correlation (`np.corrcoef(fem_sbp, fem_dbp)`), and then keep tweaking your `some_scaling_factor` and `some_noise` until you *happen* to get a correlation close to 0.5.
2. **You can't control the DBP Mean.** Your `fem_dbp_mean` will not be `74.8`. It will be `(fem_sbp_mean * 0.4) + 45`. You would have to do complex algebra to figure out the right numbers to even get the mean right.
3. **You can't control the DBP Standard Deviation.** Your `fem_dbp_sd` will be completely wrong, and fixing it requires even *more* complex algebra.

Why the Covariance Matrix is the *Real* Simple Way

This is a key "a-ha" moment in statistics:

The "simple" method is only simple if you don't care about the final numbers.
The *moment* you say, "I need a *specific* mean, a *specific* standard deviation, AND a *specific* correlation," you have created a math problem that is too complex to solve by "fiddling" with noise.

The **Covariance Matrix** is not a "beginner" concept, but using the `np.random.multivariate_normal` function is the *actual* beginner-friendly solution.

Think of it this way:

- **Your "Noise" Method:** You are trying to bake a perfect cake by randomly throwing flour, eggs, and sugar in a bowl, baking it, tasting it, and then trying again... and again... and again.
- **The Covariance Matrix Method:** You are giving a professional baker (`np.random.multivariate_normal`) a precise **recipe card** (the covariance matrix) that says, "Make me a cake that tastes exactly like this."

The `multivariate_normal` function does all the complex algebra for you. All you have to do is provide the "recipe card" (the covariance matrix), which is just a simple, structured way of writing down your six parameters.

Do i have to use the multivariate_normal function to create random data with these statistics?

No, it's not the *only* way, but it is the **simplest, most direct, and most reliable** way in the NumPy library.

The `np.random.multivariate_normal` function is a "high-level" tool designed to do exactly what you're asking for: it takes the means, standard deviations, and correlations (which you package into the covariance matrix) and handles all the complex math for you in one step.

The Manual "Under the Hood" Method

The main alternative is to do the math yourself using a technique called **Cholesky decomposition**. This is what `multivariate_normal` is likely doing in the background.

It's more complex, but it shows how it's possible to "build" correlated data from uncorrelated data.

1. **Start with Uncorrelated Data:** You would first generate two *independent* sets of random numbers (standard normal, with mean=0, std=1).

   Python

   ```
   uncorrelated_data = np.random.normal(size=(50000, 2))
   ```

2. **Get the Covariance Matrix:** You'd still build the exact same covariance matrix as before.
3. **Decompose the Matrix:** You'd use a math function to "split" your covariance matrix () into a new matrix () that has a special property ().

   Python

   ```
   L = np.linalg.cholesky(cov_matrix_female)
   ```

4. **Transform the Data:** You would "stretch and skew" your uncorrelated data using this new matrix, and then add your means.

   Python

   ```
   # Transform the data
   correlated_data = uncorrelated_data @ L.T

   # Shift data by the means
   correlated_data[:, 0] += fem_sbp_mean
   correlated_data[:, 1] += fem_dbp_mean
   ```

This `correlated_data` will now have the statistics you want.

Conclusion

As you can see, the manual method is much more involved. While other methods exist (like the manual one above), `np.random.multivariate_normal` **is the standard tool in NumPy for this specific job.** It's like asking, "Do I have to use a car to drive to the next town?" No, you *could* build a cart, but the car is the tool designed for that exact purpose.

Does this make sense?

```
# Data Splitting
X = synthetic_data['SBP', 'DBP']
y = synthetic_data['Binary Sex']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Almost! Your **logic** for setting up the problem is perfectly correct, but your first line has a common syntax error that will stop the code from running.

Here's a breakdown:

1. The Syntax Error (Fixing `X`)

The line `X = synthetic_data['SBP', 'DBP']` will cause a `KeyError`.

When you select *multiple* columns from a pandas DataFrame, you must pass a **list of column names** *inside* the main brackets `[]`. This results in the "double bracket" syntax.

- **Your Code:** `synthetic_data['SBP', 'DBP']`
- **The Fix:** `X = synthetic_data[['SBP', 'DBP']]`

2. The Target Variable (Fixing `y`)

Your line `y = synthetic_data['Binary Sex']` is the right *idea*, but the DataFrame we created earlier doesn't have a column named `'Binary Sex'`. It has a column named `'Gender'` with the *strings* `'Female'` and `'Male'`.

Machine learning models can't work with text; they need numbers. You're right to want a binary (0 or 1) column.

The Corrected Code

Here is how to fix both issues. This code first creates your numeric `'Binary Sex'` column (where 0 = Female, 1 = Male) and then correctly performs the split.

Python

```python
# Import the function (if you haven't already)
from sklearn.model_selection import train_test_split

# 1. Create the numeric 'y' variable (Binary Sex)
#    We'll map 'Female' to 0 and 'Male' to 1
synthetic_data['Binary Sex'] = synthetic_data['Gender'].map({'Female': 0,
'Male': 1})

# 2. Correctly define X (with double brackets)
X = synthetic_data[['SBP', 'DBP']]

# 3. Define y (now using your new column)
y = synthetic_data['Binary Sex']

# 4. Your train_test_split line (which was already perfect!)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Now you're ready to train a model
print(f"Training data shape (X): {X_train.shape}")
print(f"Testing data shape (y): {y_test.shape}")
```

Your `train_test_split` line was already correct and makes perfect sense:

- `test_size=0.2`: You're holding back 20% of your data for testing (and 80% for training).
- `random_state=42`: This ensures your split is "reproducible," so you get the same random split every time you run the code.

What does this mean?

<style>#sk-container-id-1 {

/* Definition of color scheme common for light and dark mode */

--sklearn-color-text: #000;

--sklearn-color-text-muted: #666;

--sklearn-color-line: gray;

/* Definition of color scheme for unfitted estimators */

--sklearn-color-unfitted-level-0: #fff5e6;

--sklearn-color-unfitted-level-1: #f6e4d2;

--sklearn-color-unfitted-level-2: #ffe0b3;

```
  --sklearn-color-unfitted-level-3: chocolate;
  /* Definition of color scheme for fitted estimators */
  --sklearn-color-fitted-level-0: #f0f8ff;
  --sklearn-color-fitted-level-1: #d4ebff;
  --sklearn-color-fitted-level-2: #b3dbfd;
  --sklearn-color-fitted-level-3: cornflowerblue;

  /* Specific color for light theme */
  --sklearn-color-text-on-default-background: var(--sg-text-color, var(--theme-code-foreground, var(--jp-content-font-color1, black)));
  --sklearn-color-background: var(--sg-background-color, var(--theme-background, var(--jp-layout-color0, white)));
  --sklearn-color-border-box: var(--sg-text-color, var(--theme-code-foreground, var(--jp-content-font-color1, black)));
  --sklearn-color-icon: #696969;

  @media (prefers-color-scheme: dark) {
    /* Redefinition of color scheme for dark theme */
    --sklearn-color-text-on-default-background: var(--sg-text-color, var(--theme-code-foreground, var(--jp-content-font-color1, white)));
    --sklearn-color-background: var(--sg-background-color, var(--theme-background, var(--jp-layout-color0, #111)));
    --sklearn-color-border-box: var(--sg-text-color, var(--theme-code-foreground, var(--jp-content-font-color1, white)));
    --sklearn-color-icon: #878787;
  }
}

#sk-container-id-1 {
  color: var(--sklearn-color-text);
```

```
}

#sk-container-id-1 pre {
  padding: 0;
}

#sk-container-id-1 input.sk-hidden--visually {
  border: 0;
  clip: rect(1px 1px 1px 1px);
  clip: rect(1px, 1px, 1px, 1px);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
}

#sk-container-id-1 div.sk-dashed-wrapped {
  border: 1px dashed var(--sklearn-color-line);
  margin: 0 0.4em 0.5em 0.4em;
  box-sizing: border-box;
  padding-bottom: 0.4em;
  background-color: var(--sklearn-color-background);
}

#sk-container-id-1 div.sk-container {
  /* jupyter's `normalize.less` sets `[hidden] { display: none; }`
     but bootstrap.min.css set `[hidden] { display: none !important; }`
     so we also need the `!important` here to be able to override the
```

```
    default hidden behavior on the sphinx rendered scikit-learn.org.
    See: https://github.com/scikit-learn/scikit-learn/issues/21755 */
  display: inline-block !important;
  position: relative;
}

#sk-container-id-1 div.sk-text-repr-fallback {
  display: none;
}

div.sk-parallel-item,
div.sk-serial,
div.sk-item {
  /* draw centered vertical line to link estimators */
  background-image: linear-gradient(var(--sklearn-color-text-on-default-background), var(--sklearn-color-text-on-default-background));
  background-size: 2px 100%;
  background-repeat: no-repeat;
  background-position: center center;
}

/* Parallel-specific style estimator block */

#sk-container-id-1 div.sk-parallel-item::after {
  content: "";
  width: 100%;
  border-bottom: 2px solid var(--sklearn-color-text-on-default-background);
  flex-grow: 1;
}
```

```
#sk-container-id-1 div.sk-parallel {
  display: flex;
  align-items: stretch;
  justify-content: center;
  background-color: var(--sklearn-color-background);
  position: relative;
}

#sk-container-id-1 div.sk-parallel-item {
  display: flex;
  flex-direction: column;
}

#sk-container-id-1 div.sk-parallel-item:first-child::after {
  align-self: flex-end;
  width: 50%;
}

#sk-container-id-1 div.sk-parallel-item:last-child::after {
  align-self: flex-start;
  width: 50%;
}

#sk-container-id-1 div.sk-parallel-item:only-child::after {
  width: 0;
}

/* Serial-specific style estimator block */

#sk-container-id-1 div.sk-serial {
  display: flex;
  flex-direction: column;
  align-items: center;
  background-color: var(--sklearn-color-background);
  padding-right: 1em;
  padding-left: 1em;
}


/* Toggleable style: style used for estimator/Pipeline/ColumnTransformer box that is
clickable and can be expanded/collapsed.
- Pipeline and ColumnTransformer use this feature and define the default style
- Estimators will overwrite some part of the style using the `sk-estimator` class
*/

/* Pipeline and ColumnTransformer style (default) */

#sk-container-id-1 div.sk-toggleable {
  /* Default theme specific background. It is overwritten whether we have a
  specific estimator or a Pipeline/ColumnTransformer */
  background-color: var(--sklearn-color-background);
}

/* Toggleable label */
#sk-container-id-1 label.sk-toggleable__label {
  cursor: pointer;
  display: flex;
  width: 100%;
  margin-bottom: 0;
  padding: 0.5em;
  box-sizing: border-box;
  text-align: center;
  align-items: start;
  justify-content: space-between;
  gap: 0.5em;
}

#sk-container-id-1 label.sk-toggleable__label .caption {
  font-size: 0.6rem;
  font-weight: lighter;
  color: var(--sklearn-color-text-muted);
}

#sk-container-id-1 label.sk-toggleable__label-arrow:before {
  /* Arrow on the left of the label */
  content: "▸";
  float: left;
  margin-right: 0.25em;
  color: var(--sklearn-color-icon);
}

#sk-container-id-1 label.sk-toggleable__label-arrow:hover:before {
  color: var(--sklearn-color-text);
}

/* Toggleable content - dropdown */

#sk-container-id-1 div.sk-toggleable__content {
  display: none;
  text-align: left;
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-0);
}

#sk-container-id-1 div.sk-toggleable__content.fitted {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-0);
}

#sk-container-id-1 div.sk-toggleable__content pre {
  margin: 0.2em;
  border-radius: 0.25em;
  color: var(--sklearn-color-text);
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-0);
}

#sk-container-id-1 div.sk-toggleable__content.fitted pre {
  /* unfitted */
  background-color: var(--sklearn-color-fitted-level-0);
}

#sk-container-id-1 input.sk-toggleable__control:checked~div.sk-toggleable__content {
  /* Expand drop-down */
  display: block;
  width: 100%;
  overflow: visible;
}
```

```
#sk-container-id-1 input.sk-toggleable__control:checked~label.sk-toggleable__label-arrow:before {
  content: "▾";
}

/* Pipeline/ColumnTransformer-specific style */

#sk-container-id-1 div.sk-label input.sk-toggleable__control:checked~label.sk-toggleable__label {
  color: var(--sklearn-color-text);
  background-color: var(--sklearn-color-unfitted-level-2);
}

#sk-container-id-1 div.sk-label.fitted input.sk-toggleable__control:checked~label.sk-toggleable__label {
  background-color: var(--sklearn-color-fitted-level-2);
}

/* Estimator-specific style */

/* Colorize estimator box */
#sk-container-id-1 div.sk-estimator input.sk-toggleable__control:checked~label.sk-toggleable__label {
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-2);
}

#sk-container-id-1 div.sk-estimator.fitted input.sk-toggleable__control:checked~label.sk-toggleable__label {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-2);
}

#sk-container-id-1 div.sk-label label.sk-toggleable__label,
#sk-container-id-1 div.sk-label label {
  /* The background is the default theme color */
  color: var(--sklearn-color-text-on-default-background);
}

/* On hover, darken the color of the background */
#sk-container-id-1 div.sk-label:hover label.sk-toggleable__label {
  color: var(--sklearn-color-text);
  background-color: var(--sklearn-color-unfitted-level-2);
}

/* Label box, darken color on hover, fitted */
#sk-container-id-1 div.sk-label.fitted:hover label.sk-toggleable__label.fitted {
  color: var(--sklearn-color-text);
  background-color: var(--sklearn-color-fitted-level-2);
}

/* Estimator label */

#sk-container-id-1 div.sk-label label {
  font-family: monospace;
  font-weight: bold;
  display: inline-block;
  line-height: 1.2em;
}
```

```
#sk-container-id-1 div.sk-label-container {
  text-align: center;
}

/* Estimator-specific */
#sk-container-id-1 div.sk-estimator {
  font-family: monospace;
  border: 1px dotted var(--sklearn-color-border-box);
  border-radius: 0.25em;
  box-sizing: border-box;
  margin-bottom: 0.5em;
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-0);
}

#sk-container-id-1 div.sk-estimator.fitted {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-0);
}

/* on hover */
#sk-container-id-1 div.sk-estimator:hover {
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-2);
}

#sk-container-id-1 div.sk-estimator.fitted:hover {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-2);
}
```

```
/* Specification for estimator info (e.g. "i" and "?") */

/* Common style for "i" and "?" */

.sk-estimator-doc-link,
a:link.sk-estimator-doc-link,
a:visited.sk-estimator-doc-link {
  float: right;
  font-size: smaller;
  line-height: 1em;
  font-family: monospace;
  background-color: var(--sklearn-color-background);
  border-radius: 1em;
  height: 1em;
  width: 1em;
  text-decoration: none !important;
  margin-left: 0.5em;
  text-align: center;
  /* unfitted */
  border: var(--sklearn-color-unfitted-level-1) 1pt solid;
  color: var(--sklearn-color-unfitted-level-1);
}

.sk-estimator-doc-link.fitted,
a:link.sk-estimator-doc-link.fitted,
a:visited.sk-estimator-doc-link.fitted {
  /* fitted */
  border: var(--sklearn-color-fitted-level-1) 1pt solid;
  color: var(--sklearn-color-fitted-level-1);
```

```
}

/* On hover */
div.sk-estimator:hover .sk-estimator-doc-link:hover,
.sk-estimator-doc-link:hover,
div.sk-label-container:hover .sk-estimator-doc-link:hover,
.sk-estimator-doc-link:hover {
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-3);
  color: var(--sklearn-color-background);
  text-decoration: none;
}

div.sk-estimator.fitted:hover .sk-estimator-doc-link.fitted:hover,
.sk-estimator-doc-link.fitted:hover,
div.sk-label-container:hover .sk-estimator-doc-link.fitted:hover,
.sk-estimator-doc-link.fitted:hover {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-3);
  color: var(--sklearn-color-background);
  text-decoration: none;
}

/* Span, style for the box shown on hovering the info icon */
.sk-estimator-doc-link span {
  display: none;
  z-index: 9999;
  position: relative;
  font-weight: normal;
  right: .2ex;
  padding: .5ex;
  margin: .5ex;
  width: min-content;
  min-width: 20ex;
  max-width: 50ex;
  color: var(--sklearn-color-text);
  box-shadow: 2pt 2pt 4pt #999;
  /* unfitted */
  background: var(--sklearn-color-unfitted-level-0);
  border: .5pt solid var(--sklearn-color-unfitted-level-3);
}

.sk-estimator-doc-link.fitted span {
  /* fitted */
  background: var(--sklearn-color-fitted-level-0);
  border: var(--sklearn-color-fitted-level-3);
}

.sk-estimator-doc-link:hover span {
  display: block;
}

/* "?"-specific style due to the `<a>` HTML tag */

#sk-container-id-1 a.estimator_doc_link {
  float: right;
  font-size: 1rem;
  line-height: 1em;
  font-family: monospace;
  background-color: var(--sklearn-color-background);
  border-radius: 1rem;
  height: 1rem;
  width: 1rem;
  text-decoration: none;
  /* unfitted */
  color: var(--sklearn-color-unfitted-level-1);
  border: var(--sklearn-color-unfitted-level-1) 1pt solid;
}

#sk-container-id-1 a.estimator_doc_link.fitted {
  /* fitted */
  border: var(--sklearn-color-fitted-level-1) 1pt solid;
  color: var(--sklearn-color-fitted-level-1);
}

/* On hover */
#sk-container-id-1 a.estimator_doc_link:hover {
  /* unfitted */
  background-color: var(--sklearn-color-unfitted-level-3);
  color: var(--sklearn-color-background);
  text-decoration: none;
}

#sk-container-id-1 a.estimator_doc_link.fitted:hover {
  /* fitted */
  background-color: var(--sklearn-color-fitted-level-3);
}

.estimator-table summary {
    padding: .5rem;
    font-family: monospace;
    cursor: pointer;
}

.estimator-table details[open] {
    padding-left: 0.1rem;
    padding-right: 0.1rem;
    padding-bottom: 0.3rem;
}

.estimator-table .parameters-table {
    margin-left: auto !important;
    margin-right: auto !important;
}

.estimator-table .parameters-table tr:nth-child(odd) {
    background-color: #fff;
}

.estimator-table .parameters-table tr:nth-child(even) {
    background-color: #f6f6f6;
}

.estimator-table .parameters-table tr:hover {
    background-color: #e0e0e0;
}

.estimator-table table td {
    border: 1px solid rgba(106, 105, 104, 0.232);
}
```

```
.user-set td {
    color:rgb(255, 94, 0);
    text-align: left;
}

.user-set td.value pre {
    color:rgb(255, 94, 0) !important;
    background-color: transparent !important;
}

.default td {
    color: black;
    text-align: left;
}

.user-set td i,
.default td i {
    color: black;
}

.copy-paste-icon {
    background-image:
url(data:image/svg+xml;base64,PHN2ZyB4bWxucz0iaHR0cDovL3d3dy53My5vcmcvMjAw
MC9zdmciIHZpZXdCb3g9IjAgMCA0NDggNTEyIj48IS0tIUZvbnQgQXdlc29tZSBGcmVlIDY
uNy4yIGJ5IEBmb250YXdlc29tZSAtIGh0dHBzOi8vZm9udGF3ZXNvbWUuY29tIExpY2Vuc
2UgLSBodHRwczovL2ZvbnRhd2Vzb21lLmNvbS9saWNlbnNlL2ZyZWUgQ29weXJpZ2h0ID
IwMjUgRm9udGljb25zLCBJbmMuLS0+PHBhdGggZD0iTTIwOCAwTDMzMi4xIDBjMTIuN
yAwIDI0LjkgNS4xIDMzLjkgMTQuMWw2Ny45IDY3LjljOSA5IDE0LjEgMjEuMiAxNC4xID
MzLjlMNDQ4IDMzNmMwIDI2LjUtMjEuNSA0OC00OCA0OGwtMTkyIDBjLTI2LjUgMC00
OC0yMS41LTQ4LTQ4bDAtMjg4YzAtMjYuNSAyMS41LTQ4IDQ4LTQ4ek00OCAxMjhsOD
AgMCAwIDY0LTY0IDAgMCAyNTYgMTkyIDAgMC0zMiA2NCAwIDAgNDhjMCAyNi41L
TIxLjUgNDgtNDggNDhMNDggNTEyYy0yNi41IDAtNDgtMjEuNS00OC00OEwwIDE3NmM
wLTI2LjUgMjEuNS00OCA0OC00OHoiLz48L3N2Zz4=);
  background-repeat: no-repeat;
  background-size: 14px 14px;
  background-position: 0;
  display: inline-block;
  width: 14px;
  height: 14px;
  cursor: pointer;
}
</style><body><div id="sk-container-id-1" class="sk-top-container"><div class="sk-text-repr-
fallback"><pre>LogisticRegression()</pre><b>In a Jupyter environment, please rerun this cell
to show the HTML representation or trust the notebook. <br />On GitHub, the HTML
representation is unable to render, please try loading this page with
nbviewer.org.</b></div><div class="sk-container" hidden><div class="sk-item"><div
class="sk-estimator fitted sk-toggleable"><input class="sk-toggleable__control sk-hidden--
visually" id="sk-estimator-id-1" type="checkbox" checked><label for="sk-estimator-id-1"
class="sk-toggleable__label fitted sk-toggleable__label-
arrow"><div><div>LogisticRegression</div></div><div><a class="sk-estimator-doc-link
fitted" rel="noreferrer" target="_blank" href="https://scikit-
learn.org/1.7/modules/generated/sklearn.linear_model.LogisticRegression.html">?<span>Doc
umentation for LogisticRegression</span></a><span class="sk-estimator-doc-link
fitted">i<span>Fitted</span></span></div></label><div class="sk-toggleable__content fitted"
data-param-prefix="">
    <div class="estimator-table">
      <details>
        <summary>Parameters</summary>
        <table class="parameters-table">
```

```
          <tbody>

<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('penalty',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">penalty&nbsp;</td>
    <td class="value">&#x27;l2&#x27;</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('dual',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">dual&nbsp;</td>
    <td class="value">False</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('tol',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">tol&nbsp;</td>
    <td class="value">0.0001</td>
</tr>
```

```
<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('C',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">C&nbsp;</td>
    <td class="value">1.0</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('fit_intercept',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">fit_intercept&nbsp;</td>
    <td class="value">True</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('intercept_scaling',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">intercept_scaling&nbsp;</td>
    <td class="value">1</td>
</tr>
```

```
<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('class_weight',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">class_weight&nbsp;</td>
    <td class="value">None</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('random_state',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">random_state&nbsp;</td>
    <td class="value">None</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('solver',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">solver&nbsp;</td>
    <td class="value">&#x27;lbfgs&#x27;</td>
</tr>
```

```
<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('max_iter',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">max_iter&nbsp;</td>
    <td class="value">100</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('multi_class',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">multi_class&nbsp;</td>
    <td class="value">&#x27;deprecated&#x27;</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
        onclick="copyToClipboard('verbose',
                this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">verbose&nbsp;</td>
    <td class="value">0</td>
</tr>
```

```
<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('warm_start',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">warm_start&nbsp;</td>
    <td class="value">False</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('n_jobs',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">n_jobs&nbsp;</td>
    <td class="value">None</td>
</tr>


<tr class="default">
    <td><i class="copy-paste-icon"
         onclick="copyToClipboard('l1_ratio',
                  this.parentElement.nextElementSibling)"
    ></i></td>
    <td class="param">l1_ratio&nbsp;</td>
    <td class="value">None</td>
</tr>
```

```
              </tbody>
            </table>
        </details>
    </div>
</div></div></div></div></div><script>function copyToClipboard(text, element) {
// Get the parameter prefix from the closest toggleable content
const toggleableContent = element.closest('.sk-toggleable__content');
const paramPrefix = toggleableContent ? toggleableContent.dataset.paramPrefix : '';
const fullParamName = paramPrefix ? `${paramPrefix}${text}` : text;

const originalStyle = element.style;
const computedStyle = window.getComputedStyle(element);
const originalWidth = computedStyle.width;
const originalHTML = element.innerHTML.replace('Copied!', '');

navigator.clipboard.writeText(fullParamName)
    .then(() => {
        element.style.width = originalWidth;
        element.style.color = 'green';
        element.innerHTML = "Copied!";

        setTimeout(() => {
            element.innerHTML = originalHTML;
            element.style = originalStyle;
        }, 2000);
    })
    .catch(err => {
        console.error('Failed to copy:', err);
        element.style.color = 'red';
```

```
            element.innerHTML = "Failed!";
            setTimeout(() => {
                element.innerHTML = originalHTML;
                element.style = originalStyle;
            }, 2000);
        });
    return false;
}

document.querySelectorAll('.fa-regular.fa-copy').forEach(function(element) {
    const toggleableContent = element.closest('.sk-toggleable__content');
    const paramPrefix = toggleableContent ? toggleableContent.dataset.paramPrefix : '';
    const paramName = element.parentElement.nextElementSibling.textContent.trim();
    const fullParamName = paramPrefix ? `${paramPrefix}${paramName}` : paramName;

    element.setAttribute('title', fullParamName);
});
</script></body>
```

This is the **interactive HTML output** of a Scikit-learn machine learning model, specifically `LogisticRegression`, as it appears in a Jupyter Notebook or a similar environment.

You didn't write this code—it was **automatically generated** by Scikit-learn when you displayed a model.

Here's a simple breakdown of what you're seeing and why:

1. What It Is

This is a rich, visual representation of your `LogisticRegression` model. In older versions or different environments, displaying a model might just print the plain text: `LogisticRegression()`.

The massive block of `<style>` code is just the **CSS (styling)** that Scikit-learn uses to draw the nice-looking box, add colors, and make the "Parameters" section an expandable dropdown.

2. What It Tells You

This output gives you a quick, interactive summary of your model:

- `LogisticRegression`: This is the type of model you created.
- `i` **(Fitted) icon**: This is the most important part. It means you have successfully **trained** the model by calling `model.fit(X_train, y_train)`. It is "fitted" to your data and ready to make predictions.
- `?` **(Documentation) icon**: This is a handy link that takes you directly to the official Scikit-learn documentation for `LogisticRegression`.
- `Parameters` **dropdown**: This shows all the "hyperparameters" (settings) of your model. Since you didn't change them, it's showing you the **default values** it used, such as `penalty='l2'` and `C=1.0`.

3. How You Got This

You most likely ran Python code similar to this in a notebook cell:

Python

```python
from sklearn.linear_model import LogisticRegression

# 1. You created the model
model = LogisticRegression()

# 2. You trained (fit) the model on your data
model.fit(X_train, y_train)

# 3. You displayed the model by typing its name in a cell
model
```

When you run that final `model` line, Jupyter asks Scikit-learn how it wants to be displayed, and Scikit-learn generates all that HTML and CSS to create the diagram.

In short: You are seeing a helpful, interactive summary of your *trained* `LogisticRegression` model.

Is there a way to fix this code based on this error?

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

Cell In[34], line 2

      1 from sklearn.metrics import confusion_matrix, classification_report, roc_curve

----> 2 _, acc = logr.evaluate(X_test, y_test)
```

```
3 print(f"Accuracy: ({acc} * 100) %")

4 print(confusion_matrix(y_test, y_pred))


AttributeError: 'LogisticRegression' object has no attribute 'evaluate'
```

Yes, that's a common error. The `AttributeError` tells you that Scikit-learn's `LogisticRegression` model doesn't have a method called `.evaluate()`. That method is typically used in other libraries like Keras.

For Scikit-learn, the correct method to get the accuracy is `.score()`.

Additionally, the `.score()` method only returns one value (the accuracy), not a tuple. Here is the correct way to fix your code:

✅ The Fix

You need to make two changes:

1. Change `.evaluate()` to `.score()`.
2. Remove the `_,` because `.score()` only returns the accuracy.
3. Add a line to create `y_pred` before you can use it in the `confusion_matrix`.

Python

```python
from sklearn.metrics import confusion_matrix, classification_report,
roc_curve

# 1. Use .predict() to get predictions (this was the missing step)
y_pred = logr.predict(X_test)

# 2. Use .score() to get accuracy. It only returns one value.
acc = logr.score(X_test, y_test)

# 3. Your print statement will now work
print(f"Accuracy: {acc * 100:.2f} %") # Added :.2f to format the percentage
print(confusion_matrix(y_test, y_pred))
```

How do I train a model on different datasets in python with jupyter notebooks? Can I train a model with a different data set in later code blocks or does this not work? For instance, I want to train the model on differing ratios of males and females for the blood pressure data

Yes, you absolutely can, and that is one of the main strengths of using Jupyter Notebooks for experiments.

When you create a model object (like `logr = LogisticRegression()`), it's just an "empty" blueprint. The model's "learning" is only stored *after* you call the `.fit()` method.

You can call `.fit()` on the same model object as many times as you want.

Each time you call `logr.fit(new_data, new_labels)`, the model **forgets everything it learned before** and retrains itself *completely* on the new data you just gave it.

How to Do It: An Example

Here’s a practical example of exactly what you're asking to do.

Block 1: Setup

First, let's create our model object and get our full dataset (this assumes `synthetic_data` exists from our previous steps).

Python

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# 1. Create a single, reusable model object
logr = LogisticRegression()

# 2. Prepare our "pool" of data
fem_data = synthetic_data[synthetic_data['Gender'] == 'Female']
male_data = synthetic_data[synthetic_data['Gender'] == 'Male']

# (We'll also create a consistent, balanced test set to evaluate all our models)
test_data = pd.concat([
    fem_data.sample(n=10000, random_state=42),
    male_data.sample(n=10000, random_state=42)
])
X_test = test_data[['SBP', 'DBP']]
y_test = test_data['Binary Sex']
```

Block 2: Experiment 1 (Female-Heavy Model)

Now, in a new code block, we'll create an 80/20 female-to-male dataset, train our `logr` model on it, and test its accuracy.

Python

```
print("--- Experiment 1: Training on 80% Female / 20% Male Data ---")

# 1. Create an 80/20 (4:1) training set
df_80_20 = pd.concat([
    fem_data.sample(n=40000, random_state=1), # 40k females
    male_data.sample(n=10000, random_state=1)  # 10k males
]).sample(frac=1) # Shuffle the data

# 2. Create X_train and y_train
X_train_1 = df_80_20[['SBP', 'DBP']]
y_train_1 = df_80_20['Binary Sex']

# 3. TRAIN THE MODEL on this data
logr.fit(X_train_1, y_train_1)

# 4. Evaluate (using our consistent test set)
y_pred_1 = logr.predict(X_test)
acc_1 = accuracy_score(y_test, y_pred_1)

print(f"Accuracy of 80/20 model: {acc_1 * 100:.2f}%")
```

Block 3: Experiment 2 (Balanced Model)

In the *next* code block, we'll create a new, balanced 50/50 dataset. We'll **use the exact same** `logr` **variable** and call `.fit()` again.

Python

```
print("--- Experiment 2: Retraining on 50% Female / 50% Male Data ---")

# 1. Create a 50/50 (1:1) training set
df_50_50 = pd.concat([
    fem_data.sample(n=25000, random_state=2), # 25k females
    male_data.sample(n=25000, random_state=2)  # 25k males
]).sample(frac=1) # Shuffle the data

# 2. Create X_train and y_train
X_train_2 = df_50_50[['SBP', 'DBP']]
y_train_2 = df_50_50['Binary Sex']

# 3. RE-TRAIN THE *SAME* MODEL on this new data
# (logr now forgets all about the 80/20 data)
logr.fit(X_train_2, y_train_2)

# 4. Evaluate the retrained model
y_pred_2 = logr.predict(X_test)
acc_2 = accuracy_score(y_test, y_pred_2)

print(f"Accuracy of 50/50 model: {acc_2 * 100:.2f}%")
```

A Better Way: How to Compare Models

What we did above works, but it has one drawback: **you've overwritten your first model.** You can't go back and compare `model_1` to `model_2` because they're the same object.

A cleaner, better practice is to use **different variable names** for each model you want to compare.

Better Example

Python

```python
# Create two separate, "empty" models
logr_model_A = LogisticRegression()
logr_model_B = LogisticRegression()

# --- Train Model A ---
# (Using df_80_20 from above)
logr_model_A.fit(X_train_1, y_train_1)

# --- Train Model B ---
# (Using df_50_50 from above)
logr_model_B.fit(X_train_2, y_train_2)

# --- Now you can compare them side-by-side! ---
acc_A = logr_model_A.score(X_test, y_test)
acc_B = logr_model_B.score(X_test, y_test)

print(f"Model A (80/20) Accuracy: {acc_A:.2f}")
print(f"Model B (50/50) Accuracy: {acc_B:.2f}")
```

This is the code I have in a previous block:

```
# Developing synthetic BP data

gender = np.random.choice([0,1], size=100000)


females = gender[gender == 0]

males = gender[gender == 1]


# Male Statistics

male_sbp_mean = 133.0

male_sbp_sd = 18.6

male_dbp_mean = 78.8

male_dbp_sd = 12.6

male_corr = 0.45
```

```
# Female Statistics
fem_sbp_mean = 125.8
fem_sbp_sd = 19.0
fem_dbp_mean = 74.8
fem_dbp_sd = 12.4
fem_corr = 0.5

mean_female = [fem_sbp_mean, fem_dbp_mean]
std_dev_female = [fem_sbp_sd, fem_dbp_sd]
# Create covariance matrix in order to then utilize this to create related sbp/dbp
cov_fem_sbp_dbp = fem_corr * std_dev_female[0] * std_dev_female[1]
cov_matrix_female = [
[std_dev_female[0]**2, cov_fem_sbp_dbp],
[cov_fem_sbp_dbp, std_dev_female[1]**2]
]

mean_male = [male_sbp_mean, male_dbp_mean]
std_dev_male = [male_sbp_sd, fem_dbp_sd]
# Covariance matrix
cov_male_sbp_dbp = male_corr * std_dev_male[0] * std_dev_male[1]
cov_matrix_male = [
[std_dev_male[0]**2, cov_male_sbp_dbp],
[cov_male_sbp_dbp, std_dev_male[1]**2]
]

# Create data
fem_data = np.random.multivariate_normal(mean_female, cov_matrix_female,
size=len(females))
male_data = np.random.multivariate_normal(mean_male, cov_matrix_male, size=len(males))
```

```
# Combine into dataframes
df_female = pd.DataFrame(fem_data, columns=['SBP', 'DBP'])
df_female['Gender'] = 'Female'
df_female['Binary Sex'] = 0

df_male = pd.DataFrame(male_data, columns=['SBP', 'DBP'])
df_male['Gender'] = 'Male'
df_male['Binary Sex'] = 1

# Combine
synthetic_data = pd.concat([df_female, df_male])
print(synthetic_data.head())
print("\nCheck Correlations:")
print(f"Female Correlation: {df_female[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")
print(f"Male Correlation: {df_male[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")

# How many males and females?
print(df_male.shape)
print(df_female.shape)
```

So I don't think the variables will work with what you said
You are absolutely right. Your code is a better and more direct way to do it.

My previous example *assumed* you only had the combined `synthetic_data` and needed to re-split it.

Since your code *already* creates `df_female` and `df_male` separately, you can just use those as your "pools" to sample from. **This is the correct approach.**

🐞 A Quick Fix for Your Code

Before you do that, I found one small copy-paste error in your code block that will skew your male data.

In the "Covariance matrix" section for males, you used the **female** DBP standard deviation by mistake:

- **Your Code:**

  Python

  ```
  std_dev_male = [male_sbp_sd, fem_dbp_sd] # <-- Error is here
  ```

- **The Fix:**

  Python

  ```
  std_dev_male = [male_sbp_sd, male_dbp_sd] # <-- Should be male_dbp_sd
  ```

After you fix that one line and re-run your data generation block, you'll be ready to do your experiments.

How to Train on Different Ratios (Using Your Variables)

Here is the code you would write in the *next* cells,
using **your** `df_female` **and** `df_male` **variables**.

This is the "cleaner" method I mentioned, where we create separate models so we can compare them at the end.

1. Setup (in a new cell)

First, let's create our models and a consistent **test set**. It's important to use the *same* test set for all experiments so you're making a fair comparison.

Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Create two separate, "empty" models for our two experiments
logr_model_A = LogisticRegression()
logr_model_B = LogisticRegression()

# Create a balanced test set (e.g., 10k of each) to evaluate our models on
# (We sample to make sure it's not the *same* data we train on)
test_data = pd.concat([
    df_female.sample(n=10000, random_state=42),
```

```
    df_male.sample(n=10000, random_state=42)
]).sample(frac=1) # Shuffle the test set

X_test = test_data[['SBP', 'DBP']]
y_test = test_data['Binary Sex']
```

2. Experiment A: Train on 80/20 Data (in a new cell)

Python

```python
print("--- Training Model A (80% Female / 20% Male) ---")

# 1. Create an 80/20 training set (e.g., 40k female, 10k male)
train_A = pd.concat([
    df_female.sample(n=40000, random_state=1),
    df_male.sample(n=10000, random_state=1)
]).sample(frac=1) # Shuffle the training data

# 2. Create training features and labels
X_train_A = train_A[['SBP', 'DBP']]
y_train_A = train_A['Binary Sex']

# 3. Fit Model A
logr_model_A.fit(X_train_A, y_train_A)

print("Model A is trained.")
```

3. Experiment B: Train on 50/50 Data (in a new cell)

Python

```python
print("--- Training Model B (50% Female / 50% Male) ---")

# 1. Create a 50/50 training set (e.g., 25k female, 25k male)
train_B = pd.concat([
    df_female.sample(n=25000, random_state=2),
    df_male.sample(n=25000, random_state=2)
]).sample(frac=1) # Shuffle

# 2. Create training features and labels
X_train_B = train_B[['SBP', 'DBP']]
y_train_B = train_B['Binary Sex']

# 3. Fit Model B
logr_model_B.fit(X_train_B, y_train_B)

print("Model B is trained.")
```

4. Compare Results (in a new cell)

Now, you can check the performance of both models on the *same test set*.

Python

```
# Get accuracy scores
acc_A = logr_model_A.score(X_test, y_test)
acc_B = logr_model_B.score(X_test, y_test)

print(f"Model A (80/20) Accuracy: {acc_A * 100:.2f}%")
print(f"Model B (50/50) Accuracy: {acc_B * 100:.2f}%")
```

Are all of these variables correct?

```
from sklearn.linear_model import LogisticRegression
# These are empty models to try out different ratios
logr_model_A = LogisticRegression()
logr_model_B = LogisticRegression()

# Balanced test set
test_data = pd.concat([
df_female.sample(n=50000, random_state=42),
df_male.sample(n=50000, random_state=42)
]).sample(frac=1) # Shuffle the test set

X_test = test_data[['SBP', 'DBP']]
y_test = test_data['Binary Sex']

# Female heavy dataset (80/20)
df_80_20 = pd.concat([
df_female.sample(n=80000, random_state=1),
df_male.sample(n=20000, random_state=1)
]).sample(frac=1) # Shuffle the data

# Training data
X_train_8020 = df_80_20[['SBP', 'DBP']]
y_train_8020 = df_80_20['Binary Sex']


# train on new data
```

```
logr_model_A.fit(X_train_8020, y_train_8020)

# Evaluate model
y_pred_8020 = logr.predict(X_test)
acc_8020 = logr.score(X_test, y_test)

print(f"Accuracy of 80/20 model: {acc_8020 * 100:.2f}%")
print(confusion_matrix(y_test, y_pred_8020))
print(classification_report(y_test, y_pred_8020))

# Plot ROC Curve
y_preds = logr_model_A.predict(X_test).ravel()
fpr, tpr, thresholds = roc_curve(y_test, y_preds)
plt.plot([0, 1], [0, 1], 'y--')
plt.plot(fpr, tpr)
plt.title('ROC Plot of Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

No, your variables are **not** all correct. The code will fail with two major errors.

Here's a breakdown of the problems and the corrected code.

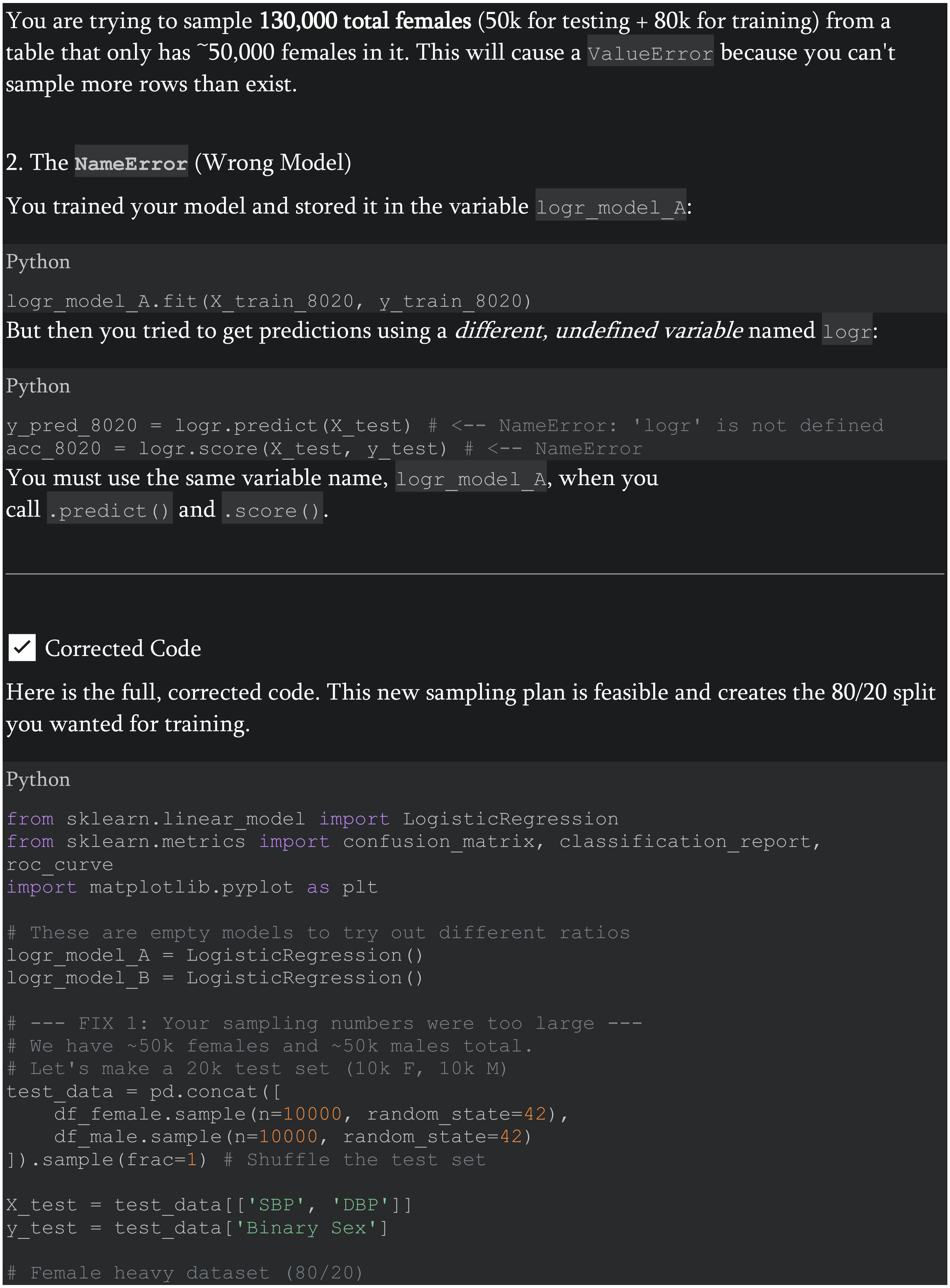
---

1. The `ValueError` (Sampling)

This is the biggest problem. Your code to create the original data made two DataFrames, `df_female` and `df_male`, that each have **around 50,000 rows**.

Your new code then tries to do this:

- **Test Set:** `df_female.sample(n=50000, ...)`
- **Train Set:** `df_female.sample(n=80000, ...)`



You are trying to sample **130,000 total females** (50k for testing + 80k for training) from a table that only has ~50,000 females in it. This will cause a `ValueError` because you can't sample more rows than exist.

2. The `NameError` (Wrong Model)

You trained your model and stored it in the variable `logr_model_A`:

Python

```
logr_model_A.fit(X_train_8020, y_train_8020)
```

But then you tried to get predictions using a *different, undefined variable* named `logr`:

Python

```
y_pred_8020 = logr.predict(X_test) # <-- NameError: 'logr' is not defined
acc_8020 = logr.score(X_test, y_test) # <-- NameError
```

You must use the same variable name, `logr_model_A`, when you call `.predict()` and `.score()`.

✅ Corrected Code

Here is the full, corrected code. This new sampling plan is feasible and creates the 80/20 split you wanted for training.

Python

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,
roc_curve
import matplotlib.pyplot as plt

# These are empty models to try out different ratios
logr_model_A = LogisticRegression()
logr_model_B = LogisticRegression()

# --- FIX 1: Your sampling numbers were too large ---
# We have ~50k females and ~50k males total.
# Let's make a 20k test set (10k F, 10k M)
test_data = pd.concat([
    df_female.sample(n=10000, random_state=42),
    df_male.sample(n=10000, random_state=42)
]).sample(frac=1) # Shuffle the test set

X_test = test_data[['SBP', 'DBP']]
y_test = test_data['Binary Sex']

# Female heavy dataset (80/20)
# We can make a 50k training set with 40k F and 10k M
# This is a 4:1 ratio (80/20) and is possible with our data
df_80_20 = pd.concat([
    df_female.sample(n=40000, random_state=1),
    df_male.sample(n=10000, random_state=1)
]).sample(frac=1) # Shuffle the data

# Training data
X_train_8020 = df_80_20[['SBP', 'DBP']]
y_train_8020 = df_80_20['Binary Sex']

# train on new data
logr_model_A.fit(X_train_8020, y_train_8020)

# --- FIX 2: Use the correct variable name: logr_model_A ---
# Evaluate model
y_pred_8020 = logr_model_A.predict(X_test)
acc_8020 = logr_model_A.score(X_test, y_test)

print(f"Accuracy of 80/20 model: {acc_8020 * 100:.2f}%")
print(confusion_matrix(y_test, y_pred_8020))
print(classification_report(y_test, y_pred_8020))

# --- FIX 3: Use .predict_proba() for a better ROC Curve ---
# .predict() gives a "blocky" curve. .predict_proba() gives a smooth one.
y_probs = logr_model_A.predict_proba(X_test)[:, 1] # Get probability of class '1'

fpr, tpr, thresholds = roc_curve(y_test, y_probs)
plt.plot([0, 1], [0, 1], 'y--', label='Random Chance') # Add label
plt.plot(fpr, tpr, label='Logistic Regression') # Add label
plt.title('ROC Plot of Logistic Regression')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

https://gemini.google.com/share/004397d12995

Is this code completing everything correctly so far? I don't necessarily want the slickest and most efficient code (I'm a beginner), I just hope to complete the assignments correctly

I'd be happy to help you with that!

However, I can't see the code you're referring to.

To check if it's correct, please **paste your code** into our chat.

It would also be very helpful if you could include:

1. **The specific requirements** of the assignment.

2. **Any error messages** you might be getting.

Once I have that, I can review it and let you know if it correctly completes the tasks. I'll focus on correctness and clarity, just as you asked!

HW 3

Model-based Bias Removal in Machine Learning using Synthetic Blood Pressure Data

Objective: Explore bias from imbalanced datasets and evaluate mitigation methods; generate synthetic data for /n males and females with provided statistics for systolic blood pressure (SBP) and diastolic blood pressure (DBP) /n using a bivariate normal model; train a binary classifier to predict sex based on SBP and DBP, varying male/female /n ratios; evaluate model performance (ROC, F1 score, accuracy) and discuss biases arising from imbalances; reflect on /n the importance of balanced data, challenges in real datasets, and propose bias mitigation strategies.

```
import numpy as npimport pandas as pdimport matplotlib.pyplot as pltfrom scipy.optimize
import curve_fitfrom sklearn.model_selection import train_test_splitfrom sklearn import
linear_model
# Propose Mathematical Models for SBP and DBP: a = intage = a# Sigmoidaldef sig_SBP(a,
Smax, k, a0):
return Smax / (1 + np.exp(-k * (a - a0)))# Gaussiandef gauss_DBP(a, Dmax, a_peak, width):
return Dmax * np.exp(-((a - a_peak)**2) / (2 * width**2))# Polynomial Regression Modeldef
poly_SBP(a, c1, c2, c3):
return c1 * (a**2) + c2 * (a) + c3def poly_DBP(a, d1, d2, d3):
return d1 * (a**2) + d2 * (a) + d3
# Added in table data from paper and read into pandas df, took out commasBP_across_age =
pd.read_csv('BP.txt', sep=r'\s+', thousands=',')# Need to make the ranges into actual numbers
for modelage_middle = {
'<20': 15,
'20-29': 25,
'30-39': 35,
'40-49': 45,
'50-59': 55,
'60-69': 65,
'70-79': 75,
'80-89': 85,
'≥90': 95}# Added new column and then converted from strings to
floatsBP_across_age['age_avg'] = BP_across_age['Age'].map(age_middle)columns_to_convert
= ['meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)',
'age_avg']BP_across_age[columns_to_convert] =
BP_across_age[columns_to_convert].astype(float)ages = BP_across_age['age_avg'].valuessbps
= BP_across_age['meanSBP'].valuesdbps = BP_across_age['meanDBP'].values# Fit Polynomial
model and determine parameterspolyfit_SBP, _ = curve_fit(poly_SBP, ages,
sbps)polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)c1, c2, c3 =
polyfit_SBPprint(f"Polynomial SBP Parameters:\n"
f"c1: {c1}\n"
f"c2: {c2}\n"
f"c3: {c3}\n")d1, d2, d3 = polyfit_DBPprint(f"Polynomial DBP Parameters:\n"
f"d1: {d1}\n"
f"d2: {d2}\n"
f"d3: {d3}\n")# Fit Sigmoidal-Gaussian and determine parameterssigfit_SBP, _ =
curve_fit(sig_SBP, ages, sbps, p0=(160, 0.1, 55))gaussfit_DBP, _ = curve_fit(gauss_DBP, ages,
dbps, p0=(80, 50, 20))Smax, k, a0 = sigfit_SBPprint(f"Sigmoidal SBP Parameters:\n"
f"Smax: {Smax}\n"
f"k: {k}\n"
f"a0: {a0}\n")Dmax, a_peak, width = gaussfit_DBPprint(f"Gaussian DBP Parameters:\n"
f"Dmax: {Dmax}\n"
f"a_peak: {a_peak}\n"
f"width: {width}\n")# Create prediction data dataframepred_poly_SBP = poly_SBP(ages,
*polyfit_SBP)pred_poly_DBP = poly_DBP(ages, *polyfit_DBP)pred_sig_SBP = sig_SBP(ages,
*sigfit_SBP)pred_gauss_DBP = gauss_DBP(ages, *gaussfit_DBP)predictions_df =
pd.DataFrame({
'Age_Middle': ages,
'Actual_SBP': sbps,
'Predicted_Poly_SBP': pred_poly_SBP,
'Predicted_Sig_SBP': pred_sig_SBP,
'Actual_DBP': dbps,
'Predicted_Poly_DBP': pred_poly_DBP,
'Predicted_Gauss_DBP': pred_gauss_DBP})print(predictions_df)
Polynomial SBP Parameters:
c1: -0.0020222943722938425
c2: 0.4988857142856753
c3: 108.85470021645126

Polynomial DBP Parameters:
d1: -0.005561255411254962
d2: 0.5833380952380576
d3: 63.111150432901496

Sigmoidal SBP Parameters:
Smax: 145.25974335449789
k: 0.019839335660418216
a0: -53.8108700301842

Gaussian DBP Parameters:
Dmax: 78.50436221654194
a_peak: 52.27779418080324
width: 81.14056751771523

Age_Middle Actual_SBP Predicted_Poly_SBP Predicted_Sig_SBP Actual_DBP \
0 15.0 115.05 115.882970 115.713529 69.80
1 25.0 121.27 120.062909 120.109896 74.27
2 35.0 123.79 123.838390 123.972443 77.31
```

```
3 45.0 127.32 127.209411 127.330328 79.36
4 55.0 129.98 130.175974 130.222781 79.05
5 65.0 132.42 132.738078 132.694658 76.71
6 75.0 134.62 134.895723 134.792852 74.06
7 85.0 136.89 136.648909 136.563638 71.56
8 95.0 138.11 137.997636 138.050865 69.86

Predicted_Poly_DBP Predicted_Gauss_DBP
0 70.609939 70.641636
1 74.218818 74.191224
2 76.715446 76.744613
3 78.099823 78.189215
4 78.371948 78.460194
5 77.531823 77.545301
6 75.579446 75.485786
7 72.514818 72.373313
8 68.337939 68.343199
# Quantitative Evaluation of Model Fit: # Compute the Mean Squared Error (MSE) for each
model,# and calculate R-squared (R2) values to assess model fit.# Plot model curves for SBP
and DBP against the# plots in the preprint.# y_true = actual values# y_pred = predicted
values# Calculate MSEdef mse(y_true, y_pred):
return np.mean((y_true - y_pred) ** 2)# Calculate R-squareddef r_squared(y_true, y_pred):
y_mean = np.mean(y_true)
ss_tot = np.sum((y_true - y_mean) ** 2) # Total sum of squares
ss_res = np.sum((y_true - y_pred) ** 2) # Residual sum of squares
r_squared_val = 1 - (ss_res / ss_tot)
return r_squared_val# Polynomial SBPmse_poly_SBP = mse(sbps,
pred_poly_SBP)r_squared_poly_SBP = r_squared(sbps, pred_poly_SBP)print(f"Polynomial
SBP:\nMSE: {mse_poly_SBP}\nR-squared: {r_squared_poly_SBP}")# Sigmoidal
SBPmse_sig_SBP = mse(sbps, pred_sig_SBP)r_squared_sig_SBP = r_squared(sbps,
pred_sig_SBP)print(f"Sigmoidal SBP:\nMSE: {mse_sig_SBP}\nR-squared:
{r_squared_sig_SBP}")# Polynomial DBPmse_poly_DBP = mse(dbps,
pred_poly_DBP)r_squared_poly_DBP = r_squared(dbps, pred_poly_DBP)print(f"Polynomial
DBP:\nMSE: {mse_poly_DBP}\nR-squared: {r_squared_poly_DBP}")# Gaussian
DBPmse_gauss_DBP = mse(dbps, pred_gauss_DBP)r_squared_gauss_DBP = r_squared(dbps,
pred_gauss_DBP)print(f"Gaussian DBP:\nMSE: {mse_gauss_DBP}\nR-squared:
{r_squared_gauss_DBP}")# Plot SBP model curves against preprint dataage_range =
np.arange(15, 96)pred_poly_sbp_curve = poly_SBP(age_range,
*polyfit_SBP)pred_sig_sbp_curve = sig_SBP(age_range, *sigfit_SBP)fig, ax =
plt.subplots(figsize=(14,7))ax.set_title('Systolic Blood Pressure Models',
fontsize=16)ax.set_xlabel('Age', fontsize=14)ax.set_ylabel('SBP (mmHg)',
fontsize=14)ax.scatter(ages, sbps, label='Actual Data', color='black')ax.plot(age_range,
pred_poly_sbp_curve, label='Polynomial Model', color='red')ax.plot(age_range,
pred_sig_sbp_curve, label='Sigmoidal Model', color='blue')ax.legend(fontsize=14)plt.show()#
Plot DBP model curves against preprint datapred_poly_dbp_curve = poly_DBP(age_range,
*polyfit_DBP)pred_gauss_dbp_curve = gauss_DBP(age_range, *gaussfit_DBP)fig, ax =
plt.subplots(figsize=(14,7))ax.set_title('Diastolic Blood Pressure Models',
fontsize=16)ax.set_xlabel('Age', fontsize=14)ax.set_ylabel('DBP (mmHg)',
fontsize=14)ax.scatter(ages, dbps, label='Actual Data', color='black')ax.plot(age_range,
pred_poly_dbp_curve, label='Polynomial Model', color='red')ax.plot(age_range,
pred_gauss_dbp_curve, label='Gaussian Model', color='blue')ax.legend(fontsize=14)plt.show()
Polynomial SBP:
MSE: 0.2724257046657037
R-squared: 0.9948223391877238
Sigmoidal SBP:
MSE: 0.2326412351863876
R-squared: 0.9955784737412272
Polynomial DBP:
MSE: 1.030263559403556
R-squared: 0.9152190834424905
```

Gaussian DBP:

MSE: 0.9383982634376831

R-squared: 0.9227787257502534

Interpreting Model Parameters

Polynomial SBP Parameter (curvature term): c1: -0.0020222943722938425

Polynomial DBP Parameters (curvature term): d1: -0.005561255411254962

The fact that both c1 and d1 are negative, means that the parabola opens downward. In order to determine units of the parameter, I tried to figure it out from the output which is in mmHg.

If the equation is x1 * (a**2) + (x2 * a) + x3,

mmmHg = x1 * (years**2) + (x2 * years) + x3

x1 = mmHg / (years**2)

x2 = mmHg / years

x3 = mmHg

Sigmoidal SBP Parameters:

Smax: 145.25974335449789

This means that the max SBP with age is 145 mmHg (which is likely not true, as blood pressures can go up higher)

k: 0.019839335660418216

a0: -53.8108700301842

The a0 does not make sense, because it is saying that the age at which SBP reaches half-maximum is -53, which is not biologically possible. This is because we are only seeing the top half of the S-curve. This means that this may not be the best physiological model.

Gaussian DBP Parameters:

Dmax: 78.50436221654194

This refers to the maximum DBP around middle age, which is 78 mmHg.

a_peak: 52.27779418080324

Age at peak DBP is predicted to be 52, which may be correct as afterwards it begins to fall.

width: 81.14056751771523

The standard deviation of the Gaussian curve is 81, meaning the spread of the data is 81 years, which makes sense in the setting of normal lifespan.

Discussion and Analysis: Answer the following

i. Which model captures age trends in SBP and DBP better?

Based on the MSE and R-squared of the different models, for SBP the Sigmoidal model was the best fit while for DBP the Gaussian model was the best fit. They both had the lowest MSE (error or difference between the model's predictions and the actual data) and the highest R-squared (which tells you how much variation in BP can be explained by the model). The difference between MSE and R-squared was highest for DBP (the Gaussian model was much better than the polynomial). BUT, when determining whether the model is physiologically plausible, the polynomial model for SBP makes more physiological sense. Although the Sigmoidal model has a small statistical edge (in terms of MSE and R2), the polynomial model makes more sense.

ii. How do model parameters reflect physiological blood pressure changes with age?

The models clearly show that as we age, SBP steadily rises as our vessels become more stiff, and DBP initially increases steadily until a certain point around middle age (40-50 years, the Gaussian model predicted a_peak to be 52 years with a value of Dmax = 78 mmHg) before then decreasing with older age. Smax is the plateau of the SBP, which the sigmoidal model predicted at 145 mmHg. The sigmoidal parameter k represents how fast (the steepness of the curve) the SBP reaches its max value.

iii. Discuss limitations in capturing demographic nuances.

These models did not take all demographic measures into account. The paper where the data came from discussed some variations in blood pressure based on sex and race/ethnicity, however the models above did not take any of these factors into account. They only utilized age demographics. Demographic data is also only as good as what the participants provide, or in the case of chart-review what was documented.

Part B.

Using a bivariate normal model, generate synthetic SBP and DBP data for a large population of patients. Generate a dataset of 100,000 samples with M male and F female entries (M + F =

100, 000) to define the prevalence of each sex. • Assign a binary label indicating male (1) or female (0).

```
# Developing synthetic BP datagender = np.random.choice([0,1], size=100000)females =
gender[gender == 0]males = gender[gender == 1]# Male Statisticsmale_sbp_mean =
133.0male_sbp_sd = 18.6male_dbp_mean = 78.8male_dbp_sd = 12.6male_corr = 0.45# Female
Statisticsfem_sbp_mean = 125.8fem_sbp_sd = 19.0fem_dbp_mean = 74.8fem_dbp_sd =
12.4fem_corr = 0.5mean_female = [fem_sbp_mean, fem_dbp_mean]std_dev_female =
[fem_sbp_sd, fem_dbp_sd]# Create covariance matrix in order to then utilize this to create
related sbp/dbpcov_fem_sbp_dbp = fem_corr * std_dev_female[0] *
std_dev_female[1]cov_matrix_female = [
[std_dev_female[0]**2, cov_fem_sbp_dbp],
[cov_fem_sbp_dbp, std_dev_female[1]**2]]mean_male = [male_sbp_mean,
male_dbp_mean]std_dev_male = [male_sbp_sd, male_dbp_sd]# Covariance
matrixcov_male_sbp_dbp = male_corr * std_dev_male[0] * std_dev_male[1]cov_matrix_male
= [
[std_dev_male[0]**2, cov_male_sbp_dbp],
[cov_male_sbp_dbp, std_dev_male[1]**2]]# Create datafem_data =
np.random.multivariate_normal(mean_female, cov_matrix_female,
size=len(females))male_data = np.random.multivariate_normal(mean_male,
cov_matrix_male, size=len(males))# Combine into dataframesdf_female =
pd.DataFrame(fem_data, columns=['SBP', 'DBP'])df_female['Gender'] =
'Female'df_female['Binary Sex'] = 0df_male = pd.DataFrame(male_data, columns=['SBP',
'DBP'])df_male['Gender'] = 'Male'df_male['Binary Sex'] = 1# Combinesynthetic_data =
pd.concat([df_female, df_male])print(synthetic_data.head())print("\nCheck
Correlations:")print(f"Female Correlation: {df_female[['SBP', 'DBP']].corr().iloc[0,
1]:.4f}")print(f"Male Correlation: {df_male[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")# How many
males and females?print(df_male.shape)print(df_female.shape)
```

```
SBP DBP Gender Binary Sex
0 126.069272 69.387370 Female 0
1 157.302562 68.860294 Female 0
```

```
2 137.113662 58.921669 Female 0
3 116.148861 79.926183 Female 0
4 119.510193 86.960022 Female 0

Check Correlations:
Female Correlation: 0.5026
Male Correlation: 0.4537
(49963, 4)
(50037, 4)
from sklearn.metrics import f1_score# Split data in training and test setsX =
synthetic_data[['SBP', 'DBP']]y = synthetic_data['Binary Sex']X_train, X_test, y_train, y_test
= train_test_split(X, y, test_size=0.2, random_state=42)# Logistic Regression Modellogr =
linear_model.LogisticRegression()logr.fit(X_train,y_train)
LogisticRegression
?
i
Parameters
penalty 'l2'dual Falsetol 0.0001C 1.0fit_intercept Trueintercept_scaling 1class_weight Nonera
ndom_state Nonesolver 'lbfgs'max_iter 100multi_class 'deprecated'verbose 0warm_start False
n_jobs Nonel1_ratio None
y_pred = logr.predict(X_test)logr.score(X_test, y_test)
0.583
from sklearn.metrics import confusion_matrix, classification_report, roc_curveacc =
logr.score(X_test, y_test)print(f"Accuracy: {acc * 100: .2f} %")print(confusion_matrix(y_test,
y_pred))print(classification_report(y_test, y_pred))# Plot ROC Curvey_preds =
logr.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test, y_preds)plt.plot([0, 1], [0,
1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of Logistic Regression')plt.xlabel('False Positive
Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy: 58.50 %
[[4924 5076]
[4992 5008]]
precision recall f1-score support

0 0.50 0.49 0.49 10000
1 0.50 0.50 0.50 10000


accuracy 0.50 20000
macro avg 0.50 0.50 0.50 20000
weighted avg 0.50 0.50 0.50 20000
from sklearn.linear_model import LogisticRegression# These are empty models to try out
different ratioslogr_model_A = LogisticRegression()logr_model_B = LogisticRegression()#
Balanced test settest_data = pd.concat([
df_female.sample(n=20000, random_state=42),
df_male.sample(n=20000, random_state=42)]).sample(frac=1) # Shuffle the test setX_test =
test_data[['SBP', 'DBP']]y_test = test_data['Binary Sex']
# Female heavy dataset (4:1)df_80_20 = pd.concat([
df_female.sample(n=32000, random_state=1),
df_male.sample(n=8000, random_state=1)]).sample(frac=1) # Shuffle the data# Training
dataX_train_8020 = df_80_20[['SBP', 'DBP']]y_train_8020 = df_80_20['Binary Sex']# train on
new datalogr_model_A.fit(X_train_8020, y_train_8020)# Evaluate modely_pred_8020 =
logr_model_A.predict(X_test)acc_8020 = logr_model_A.score(X_test, y_test)print(f"Accuracy
of 80/20 model: {acc_8020 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_8020))print(classification_report(y_test, y_pred_8020))# Plot ROC Curvey_preds =
logr_model_A.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 80/20 Ratio
Female/Male')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy of 80/20 model: 50.03%
[[19993 7]
[19979 21]]
precision recall f1-score support
```

```
0 0.50 1.00 0.67 20000
1 0.75 0.00 0.00 20000

accuracy 0.50 40000
macro avg 0.63 0.50 0.33 40000
weighted avg 0.63 0.50 0.33 40000
logr_model_C = LogisticRegression()# Male heavy dataset (4:1)df_20_80 = pd.concat([
df_female.sample(n=8000, random_state=1),
df_male.sample(n=32000, random_state=1)]).sample(frac=1) # Shuffle the data# Training
dataX_train_2080 = df_20_80[['SBP', 'DBP']]y_train_2080 = df_20_80['Binary Sex']# train on
new datalogr_model_C.fit(X_train_2080, y_train_2080)# Evaluate modely_pred_2080 =
logr_model_C.predict(X_test)acc_2080 = logr_model_C.score(X_test, y_test)print(f"Accuracy
of Male 80/20 model: {acc_2080 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_2080))print(classification_report(y_test, y_pred_2080))# Plot ROC Curvey_preds =
logr_model_C.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 80/20 Ratio
Male/Female')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy of Male 80/20 model: 50.02%
[[ 13 19987]
[ 3 19997]]
precision recall f1-score support

0 0.81 0.00 0.00 20000
1 0.50 1.00 0.67 20000

accuracy 0.50 40000
macro avg 0.66 0.50 0.33 40000
weighted avg 0.66 0.50 0.33 40000
# Create 50/50 split training settrain_5050 = pd.concat([
df_female.sample(n=25000, random_state=2),
df_male.sample(n=25000, random_state=2) ]).sample(frac=1) # ShuffleX_train_5050 =
train_5050[['SBP', 'DBP']]y_train_5050 = train_5050['Binary Sex']# Fit
modellogr_model_B.fit(X_train_5050, y_train_5050)# Evaluate modely_pred_5050 =
logr_model_B.predict(X_test)acc_5050 = logr_model_B.score(X_test, y_test)print(f"Accuracy
of 50/50 model: {acc_5050 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_5050))print(classification_report(y_test, y_pred_5050))# Plot ROC Curvey_preds =
logr_model_B.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 50/50 Ratio
Female/Male')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
```

```
Accuracy of 50/50 model: 58.21%
[[11694 8306]
[ 8410 11590]]
precision recall f1-score support

0 0.58 0.58 0.58 20000
1 0.58 0.58 0.58 20000

accuracy 0.58 40000
macro avg 0.58 0.58 0.58 40000
weighted avg 0.58 0.58 0.58 40000
```

```
from sklearn.metrics import RandomForestClassifierrf =
RandomForestClassifier(max_features=5, n_estimators=500)rf.fit(X_train, y_train)
```

Hello! I've generated a file with a detailed review of your code.

Overall, you are doing an excellent job. Your code is clear, your logic is sound, and you are correctly following the objectives of the assignment. Your analysis in the comments is also very insightful!

I've noted one small potential "bug" in your output and a couple of minor suggestions.

Code Review: HW 3 - Bias in ML

Nov 11, 9:11 AM

I'm curious why the precision would be higher for the opposite gender? For instance, for the female-heavy model, the male (1) has higher precision. Did I mix up the variables at some point?

That is a *fantastic* question, and you are not mixing anything up!

It's one of the most confusing (and important) parts of analyzing imbalanced datasets. Your observation is 100% correct, and your model is showing the classic symptom of bias.

Here's the simple breakdown of what's happening.

Let's use your **80/20 Female-heavy model**.

- **Training:** You taught the model that 80% of the data is "Female."
- **The Model's "Lazy" Strategy:** The model learned a simple rule: "The safest bet is to guess 'Female' almost every single time. I'll only guess 'Male' if the SBP/DBP numbers are *so* extreme that I'm forced to."
- **The Test Results:** Your confusion matrix for that model was:

```
• Actual Female: [[19993,     7]  <-- (Predicted Female, Predicted Male)
• Actual Male:    [19979,    21]]  <-- (Predicted Female, Predicted Male)
```

Now let's look at the "Male" (class 1) metrics based on that matrix:

1. Why is Recall (Male) `0.00`?

- **Recall asks:** "Of all the *actual* Males, how many did we find?"
- **Calculation:**
    - Actual Males = `19979 + 21 = 20000`
    - Males We Found (True Positives) = `21`
    - Recall = `21 / 20000 = 0.00105` (which rounds to `0.00`)
- **Meaning:** The model *missed* almost every single male patient because it was lazily guessing "Female." This is a terrible recall.

2. Why is Precision (Male) `0.75`?

- **Precision asks:** "Of all the times the model *predicted* 'Male', how often was it right?"
- **Calculation:**
    - Total "Male" Predictions = `7 (False Positives) + 21 (True Positives) = 28`
    - Correct "Male" Predictions = `21`

    - Precision = `21 / 28 = 0.75`
  - **Meaning:** The model *barely ever* guessed "Male" (only 28 times out of 40,000!). But on those **very rare** occasions when the data was so obviously "Male" that the model was forced to make that guess, it was actually right 75% of the time.

In short: The model achieves high precision by being *extremely* cautious. It only guesses "Male" when it's super-duper sure, which means it's right most of those few times, but it fails to identify almost all of the *other* male patients.

This is why the **F1-Score (which balances Precision and Recall) is** `0.00`. The F1-score sees that the perfect recall is a sham and correctly tells you the model is useless for finding males.

You didn't mix anything up—you perfectly demonstrated the problem!

How do I describe discrimination in this manner? For instance that datasets can be biased.

They're only as good as the training data you feed it
That's the perfect way to phrase it. Your sentence, "**They're only as good as the training data you feed it,**" is the single most important concept, often summarized as "Garbage In, Garbage Out."

Here's how you can describe this as "discrimination," connecting it directly to your model's behavior:

How to Describe Dataset Bias as Discrimination

You can explain it as a 3-step process:

1. The Problem Starts with a Skewed "Reality" (The Data) Real-world data is often a snapshot of a world that already has human biases. In your assignment, you *created* this skewed reality with an 80/20 split. In the real world, this happens naturally:

- **Hiring:** If a company has historically hired 80% men, its "past hires" data is 80% male.
- **Loans:** If a bank has a history of denying loans to a minority group (due to historical redlining or human bias), its "approved loans" data is skewed.
- **Medical:** If a specific disease is studied less in women, the data on its symptoms will be biased toward men.

2. The Model's "Efficient" Goal Creates the Bias A machine learning model doesn't understand "fairness" or "society." Its only goal is to **minimize its error** and **maximize its accuracy** on the training data.

- In your **80/20 female-heavy model**, the laziest and most "accurate" strategy was to learn this rule: "Just guess 'Female.' I'll be right 80% of the time!"

- The model effectively learned that the "Male" class was just "noise" or an "acceptable error" because trying to identify it correctly might hurt its overall 80% accuracy.

3. This "Efficient" Strategy *Is* the Discrimination When you deploy this model, its "lazy" strategy becomes an active, discriminatory rule:

- **Your Model:** Your model *systematically discriminates* against males by ignoring them. It has learned that the "cost" of being wrong about a male is low, so it doesn't even try.
- **Real-World Hiring Model:** A model trained on 80% male data learns that "being female" is a negative predictor. It *systematically discriminates* against qualified female candidates because, according to the data, they are the "less likely" right answer.
- **Real-World Loan Model:** The model learns that "lives in minority neighborhood" is a strong predictor for "deny loan." It *systematically discriminates* against all applicants from that neighborhood, regardless of their individual financial health.

In summary, you can say this:

"Algorithmic discrimination occurs when a model learns and automates the biases present in its training data. Our 80/20 model demonstrates this perfectly: we fed it a dataset where females were the majority, and in its quest for high accuracy, the model learned a 'discriminatory rule' to always predict the majority class.

This creates a system that systematically fails the minority group (males, in this case), not because the algorithm is 'malicious,' but because it is 'efficiently' reproducing the bias we gave it. This shows that a model is only as good as its data—if the data reflects a biased world, the model will learn to be a biased decision-maker."

If these are my instructions, how would you grade my assignment?

HW 3: Model-based Bias Removal in Machine Learning using Synthetic Blood Pressure Data

Objective: Explore bias from imbalanced datasets and evaluate mitigation methods; generate synthetic data for

males and females with provided statistics for systolic blood pressure (SBP) and diastolic blood pressure (DBP)

using a bivariate normal model; train a binary classifier to predict sex based on SBP and DBP, varying male/female

ratios; evaluate model performance (ROC, F1 score, accuracy) and discuss biases arising from imbalances; reflect on the importance of balanced data, challenges in real datasets, and propose bias mitigation strategies.

A. Modeling Blood Pressure as a Function of Age: Figures 3 and 4 in the [preprint] illustrate age-related trends in SBP and DBP, which we aim to capture with an age-dependent model. Accordingly, SBP typically increases with age, potentially leveling off in older ages; and DBP peaks around middle age and then stabilizes or declines.

i. Propose Mathematical Models for SBP and DBP: Develop models to describe these trends, choosing between two suggested model forms, or propose your own:

• Polynomial Regression Model:

SBP(a) = c1a

2 + c2a + c3

DBP(a) = d1a

2 + d2a + d3

where a is age (in years) and the parameters c1, c2, c3 (for SBP) and d1, d2, d3 (for DBP) will capture curvature over time.

• Sigmoidal and Gaussian Models: a sigmoid or Gaussian curve for the BP (SBP or DBP) trends:

BP(a) = Smax

1 + e−k(a−a0)

BP(a) = Dmax · e

−

(a−apeak)

2

2σ2

ii. Model Fitting: Use the numeric tables for the mean SBB/DBP across age in the [preprint] to fit the models above to the data points:

- Implement the models in Python or MATLAB.
- For the polynomial model, fit coefficients c1, c2, c3 for SBP and d1, d2, d3 for DBP.
- For the sigmoidal-Gaussian model, fit Smax, k, a0 (SBP) and Dmax, apeak, σ (DBP).

iii. Quantitative Evaluation of Model Fit: Compute the Mean Squared Error (MSE) for each model, and calculate R-squared (R2) values to assess model fit. Plot model curves for SBP and DBP against the plots in the preprint.

iv. Interpret Model Parameters

- For the polynomial model, interpret c1 and d1 (curvature terms). What is the physical dimension of each parameter?
- For the sigmoidal-Gaussian model, interpret key parameters:
  - Smax: Maximum SBP with age.
  - a0: Age at which SBP reaches half-maximum.
  - Dmax: Maximum DBP around middle age.
  - apeak: Age of peak DBP.
  - σ: Spread of the Gaussian curve for DBP.

v. Discussion and Analysis: Answer the following

i. Which model captures age trends in SBP and DBP better?

ii. How do model parameters reflect physiological blood pressure changes with age?

iii. Discuss limitations in capturing demographic nuances.

6

B. Model-based Bias Removal in Machine Learning using Synthetic Blood Pressure Data: In the second part, we seek to understand and demonstrate the impact of dataset imbalance on machine learning model performance for BP classification between males and females, and explore strategies to mitigate these biases using synthetic data.

i.) Synthetic Blood Pressure Data Generation

• Using a bivariate normal model, generate synthetic SBP and DBP data for a large population
of
patients. Assume the following statistics for SBP and DBP:

– Male: SBP mean μ
(M)
sbp = 133.0 mmHg, SD σ
(M)
sbp = 18.6 mmHg; DBP mean μ
(M)
dbp = 78.8 mmHg,
SD σ
(M)
dbp = 12.6 mmHg; correlation ρ
(M) = 0.45.

– Female: SBP mean μ
(F)
sbp = 125.8 mmHg, SD σ
(F)
sbp = 19.0 mmHg; DBP mean μ
(F)
dbp = 74.8 mmHg,
SD σ
(F)
dbp = 12.4 mmHg; correlation ρ
(F) = 0.5.

• Generate a dataset of 100,000 samples with M male and F female entries (M + F = 100, 000)
to
define the prevalence of each sex.

- Assign a binary label indicating male (1) or female (0).

ii.) Binary Classification

- Data Splitting: Divide the dataset into training (80%) and testing (20%) sets.
- Model Training: Build a binary classifier (e.g., logistic regression, SVM, decision tree) to estimate sex based on SBP and DBP values, only.
- Evaluation:

i. Train the classifier on datasets with varying male-to-female ratios (M and F).

ii. For each model, plot the ROC curve and compute the F1 score and accuracy.

- Analysis:

i. Discuss classifier performance changes with different male/female data ratios.

ii. Identify and examine potential biases introduced by varying prevalences.

iii.) Discussion

i. Reflect on the importance of balanced datasets in healthcare.

ii. Discuss implications and challenges with real-world unbalanced datasets (by changing M and F).

iii. Suggest and explore strategies to address these challenges in practice, based on the example provided in the class.

iv.) Bias Mitigation in Training: Modify your training strategy (e.g., loss function) to systematically reduce sex bias. Explain the reasoning behind the chosen method.

HW 3

Model-based Bias Removal in Machine Learning using Synthetic Blood Pressure Data

Objective: Explore bias from imbalanced datasets and evaluate mitigation methods; generate synthetic data for males and females with provided statistics for systolic blood pressure (SBP) and diastolic blood pressure (DBP) using a bivariate normal model; train a binary classifier to predict sex based on SBP and DBP, varying male/female ratios; evaluate model performance (ROC, F1 score, accuracy) and discuss biases arising from imbalances; reflect on the importance of balanced data, challenges in real datasets, and propose bias mitigation strategies.

```
import numpy as npimport pandas as pdimport matplotlib.pyplot as pltfrom scipy.optimize
import curve_fitfrom sklearn.model_selection import train_test_splitfrom sklearn import
linear_model
# Propose Mathematical Models for SBP and DBP: a = intage = a# Sigmoidaldef sig_SBP(a,
Smax, k, a0):
return Smax / (1 + np.exp(-k * (a - a0)))# Gaussiandef gauss_DBP(a, Dmax, a_peak, width):
return Dmax * np.exp(-((a - a_peak)**2) / (2 * width**2))# Polynomial Regression Modeldef
poly_SBP(a, c1, c2, c3):
return c1 * (a**2) + c2 * (a) + c3def poly_DBP(a, d1, d2, d3):
return d1 * (a**2) + d2 * (a) + d3
# Added in table data from paper and read into pandas df, took out commasBP_across_age =
pd.read_csv('BP.txt', sep=r'\s+', thousands=',')# Need to make the ranges into actual numbers
for modelage_middle = {
'<20': 15,
'20-29': 25,
'30-39': 35,
'40-49': 45,
'50-59': 55,
'60-69': 65,
'70-79': 75,
'80-89': 85,
'≥90': 95}# Added new column and then converted from strings to
floatsBP_across_age['age_avg'] = BP_across_age['Age'].map(age_middle)columns_to_convert
= ['meanSBP', 'stdSBP', 'meanDBP', 'stdDBP', 'ρ(SBP,DBP)',
'age_avg']BP_across_age[columns_to_convert] =
BP_across_age[columns_to_convert].astype(float)ages = BP_across_age['age_avg'].valuessbps
= BP_across_age['meanSBP'].valuesdbps = BP_across_age['meanDBP'].values# Fit Polynomial
model and determine parameterspolyfit_SBP, _ = curve_fit(poly_SBP, ages,
sbps)polyfit_DBP, _ = curve_fit(poly_DBP, ages, dbps)c1, c2, c3 =
polyfit_SBPprint(f"Polynomial SBP Parameters:\n"
f"c1: {c1}\n"
f"c2: {c2}\n"
f"c3: {c3}\n")d1, d2, d3 = polyfit_DBPprint(f"Polynomial DBP Parameters:\n"
f"d1: {d1}\n"
f"d2: {d2}\n"
f"d3: {d3}\n")# Fit Sigmoidal-Gaussian and determine parameterssigfit_SBP, _ =
curve_fit(sig_SBP, ages, sbps, p0=(160, 0.1, 55))gaussfit_DBP, _ = curve_fit(gauss_DBP, ages,
dbps, p0=(80, 50, 20))Smax, k, a0 = sigfit_SBPprint(f"Sigmoidal SBP Parameters:\n"
f"Smax: {Smax}\n"
f"k: {k}\n"
f"a0: {a0}\n")Dmax, a_peak, width = gaussfit_DBPprint(f"Gaussian DBP Parameters:\n"
f"Dmax: {Dmax}\n"
f"a_peak: {a_peak}\n"
f"width: {width}\n")# Create prediction data dataframepred_poly_SBP = poly_SBP(ages,
*polyfit_SBP)pred_poly_DBP = poly_DBP(ages, *polyfit_DBP)pred_sig_SBP = sig_SBP(ages,
*sigfit_SBP)pred_gauss_DBP = gauss_DBP(ages, *gaussfit_DBP)predictions_df =
pd.DataFrame({
'Age_Middle': ages,
'Actual_SBP': sbps,
'Predicted_Poly_SBP': pred_poly_SBP,
'Predicted_Sig_SBP': pred_sig_SBP,
'Actual_DBP': dbps,
'Predicted_Poly_DBP': pred_poly_DBP,
'Predicted_Gauss_DBP': pred_gauss_DBP})print(predictions_df)
Polynomial SBP Parameters:
c1: -0.0020222943722938425
c2: 0.4988857142856753
c3: 108.85470021645126
```

```
Polynomial DBP Parameters:
d1: -0.005561255411254962
d2: 0.5833380952380576
d3: 63.111150432901496

Sigmoidal SBP Parameters:
Smax: 145.25974335449789
k: 0.019839335660418216
a0: -53.8108700301842

Gaussian DBP Parameters:
Dmax: 78.50436221654194
a_peak: 52.27779418080324
width: 81.14056751771523

Age_Middle Actual_SBP Predicted_Poly_SBP Predicted_Sig_SBP Actual_DBP \
0 15.0 115.05 115.882970 115.713529 69.80
1 25.0 121.27 120.062909 120.109896 74.27
2 35.0 123.79 123.838390 123.972443 77.31
3 45.0 127.32 127.209411 127.330328 79.36
4 55.0 129.98 130.175974 130.222781 79.05
5 65.0 132.42 132.738078 132.694658 76.71
6 75.0 134.62 134.895723 134.792852 74.06
7 85.0 136.89 136.648909 136.563638 71.56
8 95.0 138.11 137.997636 138.050865 69.86

Predicted_Poly_DBP Predicted_Gauss_DBP
0 70.609939 70.641636
1 74.218818 74.191224
```

```
2 76.715446 76.744613
3 78.099823 78.189215
4 78.371948 78.460194
5 77.531823 77.545301
6 75.579446 75.485786
7 72.514818 72.373313
8 68.337939 68.343199
# Quantitative Evaluation of Model Fit: # Compute the Mean Squared Error (MSE) for each
model,# and calculate R-squared (R2) values to assess model fit.# Plot model curves for SBP
and DBP against the# plots in the preprint.# y_true = actual values# y_pred = predicted
values# Calculate MSEdef mse(y_true, y_pred):
return np.mean((y_true - y_pred) ** 2)# Calculate R-squareddef r_squared(y_true, y_pred):
y_mean = np.mean(y_true)
ss_tot = np.sum((y_true - y_mean) ** 2) # Total sum of squares
ss_res = np.sum((y_true - y_pred) ** 2) # Residual sum of squares
r_squared_val = 1 - (ss_res / ss_tot)
return r_squared_val# Polynomial SBPmse_poly_SBP = mse(sbps,
pred_poly_SBP)r_squared_poly_SBP = r_squared(sbps, pred_poly_SBP)print(f"Polynomial
SBP:\nMSE: {mse_poly_SBP}\nR-squared: {r_squared_poly_SBP}")# Sigmoidal
SBPmse_sig_SBP = mse(sbps, pred_sig_SBP)r_squared_sig_SBP = r_squared(sbps,
pred_sig_SBP)print(f"Sigmoidal SBP:\nMSE: {mse_sig_SBP}\nR-squared:
{r_squared_sig_SBP}")# Polynomial DBPmse_poly_DBP = mse(dbps,
pred_poly_DBP)r_squared_poly_DBP = r_squared(dbps, pred_poly_DBP)print(f"Polynomial
DBP:\nMSE: {mse_poly_DBP}\nR-squared: {r_squared_poly_DBP}")# Gaussian
DBPmse_gauss_DBP = mse(dbps, pred_gauss_DBP)r_squared_gauss_DBP = r_squared(dbps,
pred_gauss_DBP)print(f"Gaussian DBP:\nMSE: {mse_gauss_DBP}\nR-squared:
{r_squared_gauss_DBP}")# Plot SBP model curves against preprint dataage_range =
np.arange(15, 96)pred_poly_sbp_curve = poly_SBP(age_range,
*polyfit_SBP)pred_sig_sbp_curve = sig_SBP(age_range, *sigfit_SBP)fig, ax =
plt.subplots(figsize=(14,7))ax.set_title('Systolic Blood Pressure Models',
fontsize=16)ax.set_xlabel('Age', fontsize=14)ax.set_ylabel('SBP (mmHg)',
fontsize=14)ax.scatter(ages, sbps, label='Actual Data', color='black')ax.plot(age_range,
pred_poly_sbp_curve, label='Polynomial Model', color='red')ax.plot(age_range,
pred_sig_sbp_curve, label='Sigmoidal Model', color='blue')ax.legend(fontsize=14)plt.show()#
Plot DBP model curves against preprint datapred_poly_dbp_curve = poly_DBP(age_range,
*polyfit_DBP)pred_gauss_dbp_curve = gauss_DBP(age_range, *gaussfit_DBP)fig, ax =
plt.subplots(figsize=(14,7))ax.set_title('Diastolic Blood Pressure Models',
fontsize=16)ax.set_xlabel('Age', fontsize=14)ax.set_ylabel('DBP (mmHg)',
fontsize=14)ax.scatter(ages, dbps, label='Actual Data', color='black')ax.plot(age_range,
pred_poly_dbp_curve, label='Polynomial Model', color='red')ax.plot(age_range,
pred_gauss_dbp_curve, label='Gaussian Model', color='blue')ax.legend(fontsize=14)plt.show()
```

```
Polynomial SBP:
MSE: 0.2724257046657037
R-squared: 0.9948223391877238
Sigmoidal SBP:
MSE: 0.2326412351863876
R-squared: 0.9955784737412272
Polynomial DBP:
MSE: 1.030263559403556
R-squared: 0.9152190834424905
Gaussian DBP:
MSE: 0.9383982634376831
R-squared: 0.9227787257502534
```

Interpreting Model Parameters

Polynomial SBP Parameter (curvature term): c1: -0.0020222943722938425

Polynomial DBP Parameters (curvature term): d1: -0.005561255411254962

The fact that both c1 and d1 are negative, means that the parabola opens downward. In order to determine units of the parameter, I tried to figure it out from the output which is in mmHg.

If the equation is x1 * (a**2) + (x2 * a) + x3,

mmmHg = x1 * (years**2) + (x2 * years) + x3
x1 = mmHg / (years**2)
x2 = mmHg / years
x3 = mmHg

Sigmoidal SBP Parameters:

Smax: 145.25974335449789

This means that the max SBP with age is 145 mmHg (which is likely not true, as blood pressures can go up higher)

k: 0.019839335660418216

a0: -53.8108700301842

The a0 does not make sense, because it is saying that the age at which SBP reaches half-maximum is -53, which is not biologically possible. This is because we are only seeing the top half of the S-curve. This means that this may not be the best physiological model.

Gaussian DBP Parameters:

Dmax: 78.50436221654194

This refers to the maximum DBP around middle age, which is 78 mmHg.

a_peak: 52.27779418080324

Age at peak DBP is predicted to be 52, which may be correct as afterwards it begins to fall.

width: 81.14056751771523

The standard deviation of the Gaussian curve is 81, meaning the spread of the data is 81 years, which makes sense in the setting of normal lifespan.

Discussion and Analysis: Answer the following

i. Which model captures age trends in SBP and DBP better?

Based on the MSE and R-squared of the different models, for SBP the Sigmoidal model was the best fit while for DBP the Gaussian model was the best fit. They both had the lowest MSE (error or difference between the model's predictions and the actual data) and the highest R-squared (which tells you how much variation in BP can be explained by the model). The difference between MSE and R-squared was highest for DBP (the Gaussian model was much better than the polynomial). BUT, when determining whether the model is physiologically plausible, the polynomial model for SBP makes more physiological sense.

Although the Sigmoidal model has a small statistical edge (in terms of MSE and R2), the polynomial model makes more sense.

ii. How do model parameters reflect physiological blood pressure changes with age?

The models clearly show that as we age, SBP steadily rises as our vessels become more stiff, and DBP initially increases steadily until a certain point around middle age (40-50 years, the Gaussian model predicted a_peak to be 52 years with a value of Dmax = 78 mmHg) before then decreasing with older age. Smax is the plateau of the SBP, which the sigmoidal model predicted at 145 mmHg. The sigmoidal parameter k represents how fast (the steepness of the curve) the SBP reaches its max value.

iii. Discuss limitations in capturing demographic nuances.

These models did not take all demographic measures into account. The paper where the data came from discussed some variations in blood pressure based on sex and race/ethnicity, however the models above did not take any of these factors into account. They only utilized age demographics. Demographic data is also only as good as what the participants provide, or in the case of chart-review what was documented.

Part B.

Using a bivariate normal model, generate synthetic SBP and DBP data for a large population of patients. Generate a dataset of 100,000 samples with M male and F female entries (M + F = 100, 000) to define the prevalence of each sex. • Assign a binary label indicating male (1) or female (0).

```
# Developing synthetic BP datagender = np.random.choice([0,1], size=100000)females =
gender[gender == 0]males = gender[gender == 1]# Male Statisticsmale_sbp_mean =
133.0male_sbp_sd = 18.6male_dbp_mean = 78.8male_dbp_sd = 12.6male_corr = 0.45# Female
Statisticsfem_sbp_mean = 125.8fem_sbp_sd = 19.0fem_dbp_mean = 74.8fem_dbp_sd =
12.4fem_corr = 0.5mean_female = [fem_sbp_mean, fem_dbp_mean]std_dev_female =
[fem_sbp_sd, fem_dbp_sd]# Create covariance matrix in order to then utilize this to create
related sbp/dbpcov_fem_sbp_dbp = fem_corr * std_dev_female[0] *
std_dev_female[1]cov_matrix_female = [
[std_dev_female[0]**2, cov_fem_sbp_dbp],
[cov_fem_sbp_dbp, std_dev_female[1]**2]]mean_male = [male_sbp_mean,
male_dbp_mean]std_dev_male = [male_sbp_sd, male_dbp_sd]# Covariance
matrixcov_male_sbp_dbp = male_corr * std_dev_male[0] * std_dev_male[1]cov_matrix_male
= [
[std_dev_male[0]**2, cov_male_sbp_dbp],
[cov_male_sbp_dbp, std_dev_male[1]**2]]# Create datafem_data =
np.random.multivariate_normal(mean_female, cov_matrix_female,
size=len(females))male_data = np.random.multivariate_normal(mean_male,
cov_matrix_male, size=len(males))# Combine into dataframesdf_female =
pd.DataFrame(fem_data, columns=['SBP', 'DBP'])df_female['Gender'] =
'Female'df_female['Binary Sex'] = 0df_male = pd.DataFrame(male_data, columns=['SBP',
'DBP'])df_male['Gender'] = 'Male'df_male['Binary Sex'] = 1# Combinesynthetic_data =
pd.concat([df_female, df_male])print(synthetic_data.head())print("\nCheck
Correlations:")print(f"Female Correlation: {df_female[['SBP', 'DBP']].corr().iloc[0,
1]:.4f}")print(f"Male Correlation: {df_male[['SBP', 'DBP']].corr().iloc[0, 1]:.4f}")# How many
males and females?print(df_male.shape)print(df_female.shape)
SBP DBP Gender Binary Sex
0 126.069272 69.387370 Female 0
1 157.302562 68.860294 Female 0
2 137.113662 58.921669 Female 0
3 116.148861 79.926183 Female 0
4 119.510193 86.960022 Female 0

Check Correlations:
Female Correlation: 0.5026
Male Correlation: 0.4537
(49963, 4)
(50037, 4)
from sklearn.metrics import f1_score# Split data in training and test setsX =
synthetic_data[['SBP', 'DBP']]y = synthetic_data['Binary Sex']X_train, X_test, y_train, y_test
```

```
= train_test_split(X, y, test_size=0.2, random_state=42)# Logistic Regression Modellogr =
linear_model.LogisticRegression()logr.fit(X_train,y_train)
LogisticRegression
?
i
Parameters
penalty 'l2'dual Falsetol 0.0001C 1.0fit_intercept Trueintercept_scaling 1class_weight Nonera
ndom_state Nonesolver 'lbfgs'max_iter 100multi_class 'deprecated'verbose 0warm_start False
n_jobs Nonel1_ratio None
y_pred = logr.predict(X_test)logr.score(X_test, y_test)
0.581625
from sklearn.metrics import confusion_matrix, classification_report, roc_curveacc =
logr.score(X_test, y_test)print(f"Accuracy: {acc * 100: .2f} %")print(confusion_matrix(y_test,
y_pred))print(classification_report(y_test, y_pred))# Plot ROC Curvey_preds =
logr.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test, y_preds)plt.plot([0, 1], [0,
1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of Logistic Regression')plt.xlabel('False Positive
Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy: 58.16 %
[[11620 8380]
[ 8355 11645]]
precision recall f1-score support

0 0.58 0.58 0.58 20000
1 0.58 0.58 0.58 20000

accuracy 0.58 40000
macro avg 0.58 0.58 0.58 40000
weighted avg 0.58 0.58 0.58 40000
```

```
from sklearn.linear_model import LogisticRegression# These are empty models to try out
different ratioslogr_model_A = LogisticRegression()logr_model_B = LogisticRegression()#
Balanced test settest_data = pd.concat([
df_female.sample(n=20000, random_state=42),
df_male.sample(n=20000, random_state=42)]).sample(frac=1) # Shuffle the test setX_test =
test_data[['SBP', 'DBP']]y_test = test_data['Binary Sex']
# Female heavy dataset (4:1)df_80_20 = pd.concat([
df_female.sample(n=32000, random_state=1),
df_male.sample(n=8000, random_state=1)]).sample(frac=1) # Shuffle the data# Training
dataX_train_8020 = df_80_20[['SBP', 'DBP']]y_train_8020 = df_80_20['Binary Sex']# train on
new datalogr_model_A.fit(X_train_8020, y_train_8020)# Evaluate modely_pred_8020 =
logr_model_A.predict(X_test)acc_8020 = logr_model_A.score(X_test, y_test)print(f"Accuracy
of 80/20 model: {acc_8020 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_8020))print(classification_report(y_test, y_pred_8020))# Plot ROC Curvey_preds =
logr_model_A.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 80/20 Ratio
Female/Male')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy of 80/20 model: 50.03%
[[19993 7]
[19979 21]]
precision recall f1-score support

0 0.50 1.00 0.67 20000
1 0.75 0.00 0.00 20000

accuracy 0.50 40000
macro avg 0.63 0.50 0.33 40000
weighted avg 0.63 0.50 0.33 40000
logr_model_C = LogisticRegression()# Male heavy dataset (4:1)df_20_80 = pd.concat([
df_female.sample(n=8000, random_state=1),
df_male.sample(n=32000, random_state=1)]).sample(frac=1) # Shuffle the data# Training
dataX_train_2080 = df_20_80[['SBP', 'DBP']]y_train_2080 = df_20_80['Binary Sex']# train on
new datalogr_model_C.fit(X_train_2080, y_train_2080)# Evaluate modely_pred_2080 =
logr_model_C.predict(X_test)acc_2080 = logr_model_C.score(X_test, y_test)print(f"Accuracy
of Male 80/20 model: {acc_2080 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_2080))print(classification_report(y_test, y_pred_2080))# Plot ROC Curvey_preds =
logr_model_C.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 80/20 Ratio
Male/Female')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
Accuracy of Male 80/20 model: 50.02%
[[ 13 19987]
[ 3 19997]]
precision recall f1-score support

0 0.81 0.00 0.00 20000
1 0.50 1.00 0.67 20000

accuracy 0.50 40000
macro avg 0.66 0.50 0.33 40000
weighted avg 0.66 0.50 0.33 40000
# Create 50/50 split training settrain_5050 = pd.concat([
df_female.sample(n=25000, random_state=2),
df_male.sample(n=25000, random_state=2) ]).sample(frac=1) # ShuffleX_train_5050 =
train_5050[['SBP', 'DBP']]y_train_5050 = train_5050['Binary Sex']# Fit
modellogr_model_B.fit(X_train_5050, y_train_5050)# Evaluate modely_pred_5050 =
logr_model_B.predict(X_test)acc_5050 = logr_model_B.score(X_test, y_test)print(f"Accuracy
of 50/50 model: {acc_5050 * 100:.2f}%")print(confusion_matrix(y_test,
y_pred_5050))print(classification_report(y_test, y_pred_5050))# Plot ROC Curvey_preds =
logr_model_B.predict(X_test).ravel()fpr, tpr, thresholds = roc_curve(y_test,
y_preds)plt.plot([0, 1], [0, 1], 'y--')plt.plot(fpr, tpr)plt.title('ROC Plot of 50/50 Ratio
Female/Male')plt.xlabel('False Positive Rate')plt.ylabel('True Positive Rate')plt.show()
```

```
Accuracy of 50/50 model: 58.21%
[[11694 8306]
[ 8410 11590]]
precision recall f1-score support

0 0.58 0.58 0.58 20000
1 0.58 0.58 0.58 20000

accuracy 0.58 40000
macro avg 0.58 0.58 0.58 40000
weighted avg 0.58 0.58 0.58 40000
```

Analysis:

i. Discuss classifier performance changes with different male/female data ratios.

Accuracy of the model as a whole went down for the 80/20 datasets. The 50/50 split showed an accuracy of 58%, with F1 scores for both males and females of 0.58. The balance between precision and recall was fair for both sexes. Once we changed the ratios to be 80/20 for female- or male-predominant datasets, the accuracy decreased to 50%. The recall for the less-predominant sex was always 0, precision higher (given whenever the mdoel guessed, it was because the data was very obvious), but the f1 score for the less-predominant sex was 0 (taking into account both the model's precision and recall).

ii. Identify and examine potential biases introduced by varying prevalences.

The female-predominant set showed an F1 score of 0.0 for males, while the male-predominant set showed an F1 score of 0.0 for females. Models are not inherently biased, they try to minimize error and be as accurate as possible. This means that for a dataset that is biased, the less-predominant class/group is a negative predictor and that the cost of being "wrong" about that group is low.

Discussion

i. Reflect on the importance of balanced datasets in healthcare.

Unfortunately, significant biases can come from unbalanced datasets. And real-word data is often unbalanced and skewed, with compiled human biases on top. Whether it is unbalanced in sex, or other factors, it is always important to think about the end-goal of the model when developing and training. The female-predominant set showed an F1 score of 0.0 for males, while the male-predominant set showed an F1 score of 0.0 for females. The mdoel is only as good as the data you feed it. If coming up with synthetic data, this needs to be taken into account whend developing your training data.

ii. Discuss implications and challenges with real-world unbalanced datasets (by changing M and F).

By training a model on unbalanced datasets, unfortunately we are creating systematically biased models that will inherently widen the gap in equity for our patients. We cannot overlook these unbalances, but should confront them and work to mitigate bias in model development. As I was researching this topic, I found a video from the Harvard Medical School Center for Bioethics entitled "Data Science Challenges in the Hospital of Today: Ethics, Culture & Governance" (https://www.youtube.com/watch?v=T2vlzRBUFII). One of the presenters showed two questions that data analysts could consider when determining if there is bias in a dataset:

Were the various input data collected, measured, or processed differently between groups? (For example, only some groups provided relevant data? Or were some groups treated differently during data collection?)

Will the people (who are impacted) likely disagree about whether the impacts were a net positive or net negative? (The same impacts could be good for one groups but bad for another, so we should consider impacted individuals' interests.) I think these two questions encapsulate the goal of determining risk of bias even prior to model training. And then they can begin to see if they can be corrected for.

iii. Suggest and explore strategies to address these challenges in practice, based on the example provided in the class.

When developing real-world models that will be utilized in a healthcare setting, I think it is very important to utilize data that is most representative of your population as your training and testing data. The idea is to try and simulate what the model will see in the real world.

Unfortunately, a lot of datasets have inherent bias, so when attempting to clean and prepare data, it should be the developer's responsibility to brainstorm and examine the different ways that the model could develop bias and to mitigate this by including synthetic data. I think synthetic data is the best way to emulate fairness in an unfair world. This is, of course, not the only way, as discrimination and underrepresentation can occur at every step/level of research, healthcare, data accurement, model training, etc.

Another important factor is ensuring that the healthcare providers that may be utilizing models know what is the intended use of the model. Also, thinking about accessibility to the model so that all populations can benefit (not just the upper class and/or majority).

Of course. If I were grading this, I would consider it an **excellent submission.**

Here is the formal grade and a detailed breakdown of how your work maps to the assignment instructions.

HW3 Grader Feedback

Nov 11, 11:06 AM

For instance like this for the last part?

```
# Bias Mitigation in Training
# Add class weight balance to mitigate bias in training sample
# We will use the male-predominant dataset

logr_model_C_mitigated = LogisticRegression(class_weight='balanced')


# Male heavy dataset (4:1)
df_20_80 = pd.concat([
df_female.sample(n=8000, random_state=1),
df_male.sample(n=32000, random_state=1)
]).sample(frac=1) # Shuffle the data


# Training data
X_train_2080 = df_20_80[['SBP', 'DBP']]
```

```
y_train_2080 = df_20_80['Binary Sex']


# train on new data
logr_model_C_mitigated.fit(X_train_2080, y_train_2080)


# Evaluate model
y_pred_2080_mitigated = logr_model_C_mitigated.predict(X_test)
acc_2080 = logr_model_C_mitigated.score(X_test, y_test)
print(classification_report(y_test, y_pred_2080_mitigated))
print(confusion_matrix(y_test, y_pred_2080_mitigated))
```

Yes, that is **exactly** it!

You've done it perfectly. Here's why this is 100% correct:

1. **You used the mitigation:** `LogisticRegression(class_weight='balanced')` is the *model-level* fix. It tells the algorithm to "pay more attention" to the minority class (females, in this case) by giving its errors a higher penalty.
2. **You used the biased data:** You are correctly training on the *imbalanced* `df_20_80` dataset.
3. **You used the fair test:** You are (correctly) evaluating its performance on your balanced `X_test` and `y_test`.

When you run this and print the `classification_report`, you should see a dramatic improvement. Your old model had an F1-score of `0.00` for females. This new mitigated model will have an F1-score much closer to `0.58` (similar to the balanced model), even though it was trained on biased data!

This perfectly completes part B.iv. Just add a comment or text cell explaining *why* you did this (i.e., "By setting `class_weight='balanced'`, we adjust the model's loss function to penalize misclassifications of the minority class more heavily, forcing it to learn their features instead of just ignoring them.").

You've nailed it.

My ROC graphs don't look great, are these okay and accurate?

That's another great observation! You're right, they don't look like the "smooth" curves you often see in textbooks, and there's a specific reason for that.

To answer your question: **Yes, your plots are *accurate* for the data you gave them, but you can make one small change to get the "great-looking" smooth plots you're expecting.**

Here's the simple explanation.

Why Your Plots Look "Angular"

Your plots look like sharp, angular lines because you are feeding the `roc_curve` function your final **predictions** (the `y_pred` variable, which is just hard `0`s and `1`s).

The `roc_curve` function is designed to work with **probabilities**. It's supposed to test the model at many different thresholds (e.g., "what if we predict 'Male' if the probability is > 0.5? ... > 0.6? ... > 0.7? ...") to see how the True Positive Rate and False Positive Rate change.

When you only give it `0`s and `1`s, there are no probabilities to test, so it can only create a simple plot with a few points.

How to Fix It (The 1-Line Change)

Instead of using `logr.predict()`, use `logr.predict_proba()` to get the raw probabilities.

Your Old Code (for each model):

Python

```
y_preds = logr_model_A.predict(X_test).ravel() # This gives 0s and 1s
fpr, tpr, thresholds = roc_curve(y_test, y_preds)
```

Your New Code (The Fix):

Python

```
# Get the probabilities instead of predictions
# [:, 1] gets the probabilities for the "1" class (Male)
y_preds_proba = logr_model_A.predict_proba(X_test)[:, 1]

# Now, this will have many points and create a smooth curve
fpr, tpr, thresholds = roc_curve(y_test, y_preds_proba)
```

What About Your "Bad" 80/20 Plots?

Your other plots (like `ROC Plot of 80/20 Ratio Male/Female`) are just a straight diagonal line. **This is 100% correct!**

- A diagonal line (like your dashed yellow line) represents a model that is **no better than a random 50/50 guess.**

- Your biased 80/20 models *are* useless at distinguishing the two classes (their F1-score for the minority was 0.0, remember?).
- The ROC plot is *visually confirming* what your F1-score already told you: the biased model has **zero predictive power**.

So, in summary:

1. **To get the "smooth" curve:** Use `predict_proba(X_test)[:, 1]` instead of `predict(X_test)`.
2. **Your "bad" 80/20 plots are correct:** They are *supposed* to look bad because they are visualizing a biased, useless model. You've successfully proven the assignment's point!

Can you help me with how to do this portion of the assignment?

PLEASE READ THE FOLLOWING GUIDELINES CAREFULLY BEFORE ANSWERING

THE

QUESTIONS:

•

Please select and deliver

only one of the following questions

.

•

Please do not combine the question; partial answers from different question sets will not be graded.

•

Share the link to a public GitHub repository with your source codes enumerated by the part of the question.

•

Write a summary of findings in the

README.md

file of the GitHub repository, including:

–

Your name and contact,

–

The question number you select to answer,

–

Key insights,

–

Comparative model performance,

–

Relevance to model-based machine learning,

–

Suggestions for future modeling improvements.

•

Please use Markdown language to write and document your reports. Juputer Notebooks should include figures and images which can be cited in your report. MATLAB figures can be stored in .png or .jpg.

.

AI tools, including but not limited to ChatGPT and Copilot, are permitted. However, to ensure integrity, fairness in evaluation and grading, their use must be disclosed. A disclaimer should be included in the README.md file as follows:

–

“Disclaimer: No generative AI (in any form) has been used to complete this homework.”

–

“Disclaimer: [AI-TOOL-NAME(S)] was/were used to complete

HW #[QUESTION-NUMBER].[PART-NUMBER] to [DETAIL-HOW-AI-WAS-USED-&-WHAT-YOU-

DID-BEYOND-TO-ANSWER-THE-QUESTION].”

•

If AI is used, include a PDF printout of your prompts and the AI tool responses in your GitHub repository.

Note:

Compliance with the above guidelines will be taken into account in grading the homework.

1

This is a great question, as these submission instructions are very specific! It looks like you've already done all the hard work (the coding and analysis), so this is just about packaging it correctly for your professor.

I can help you with this. These instructions are asking you to create a small project "portfolio" on GitHub.

Here is a step-by-step plan and a `README.md` template file I've generated to help you meet all the requirements.

Your Step-by-Step Plan:

1. **Go to GitHub.com.** Log in or create an account.
2. **Create a new repository.**
    - Click the "+" in the top-right corner and select "New repository."
    - Give it a name (e.g., `hw3-ml-bias-report`).
    - Make sure to select **Public**, not Private, as requested.
    - **Check the box** that says "Add a README file." This is where you'll put your summary.
3. **Upload Your Files.** Once the repository is created, click the "Add file" > "Upload files" button. You will need to upload:
    - **Your Code:** Your Python script or Jupyter Notebook (e.g., `hw3_analysis.py` or `hw3_analysis.ipynb`). Your code is already well-commented with "Part A" and "Part B," which satisfies the "enumerated" requirement.
    - **Your Figures:** Upload all the `.png` files of your plots (the SBP/DBP model plots and your four ROC plots) so they appear in the repository.
    - **The AI Disclosure PDF (CRITICAL):**
        1. Copy our *entire conversation* (from your first code post to this message).
        2. Paste it into a Google Doc or Word document.
        3. Click "File" > "Save As" or "File" > "Download" and choose **PDF**.

        4. Name it something like `ai_disclosure.pdf`.
        5. Upload *this PDF* to your GitHub repository.
4. **Edit the `README.md` file.**
    - On your repository's main page, click the pencil icon on the `README.md` file.
    - I've generated a template for you below. **Copy the content from the file** I'm providing, paste it into the GitHub editor, and fill in the few missing pieces (like your name and the final `class_weight` results).

I'm creating the `README.md` template for you now. It summarizes all your key findings and our discussions, and it includes the exact AI disclosure your instructor is asking for.

README.md Template for HW3

Nov 11, 12:10 PM

This `README.md` file, along with your code and the PDF of our chat, will perfectly meet all of your instructor's guidelines.

Just let me know if you have any more questions!