MIMIC III Visualizations

AI 395T - AI in Healthcare - Dr. Ying Ding

Joseph Skrovan, 2/3/2025

Notes:

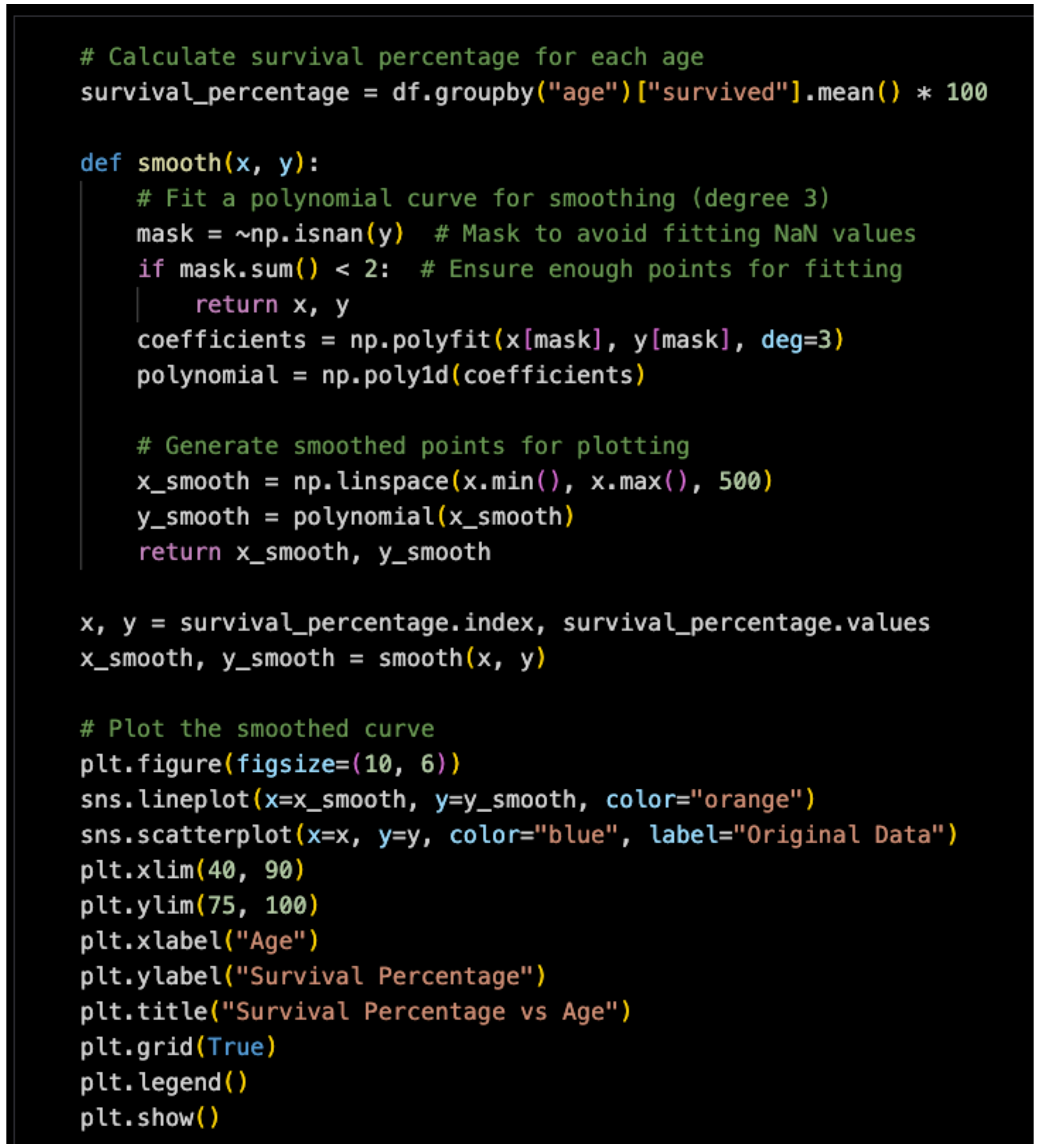
- The Jupyter notebook and support programs I wrote are available from this Github repo:
  - Survival.ipynb - Code to generate visualizations
  - I downloaded the MIMIC III dataset and worked with it locally. CHARTEVENTS.cvs.gz is a very large file (4GB), and I could not read it directly. So I wrote some helper programs to scan this large file and create a much smaller file with the measurements I wanted. These programs are also in the repo: `measurements.py, find_item_ids.py, and table_head.py`
  - Work flow:
    - `python3 find_item_ids.py height weight > lookup.yaml`
    - Edit `lookup.yaml` to the fields that I want and change field names it will write as desired.
    - `python3 measurements.py - < lookup.yaml`
    - This takes about 5 minutes on my computer to generate WEIGHT.HEIGHT.csv.gz (76KB), This file only has rows that have all requested values.
    - `puthon3 table_head.py WEIGHT.HEIGHT.csv.gz`
    - To look at the top of the file to make sure it's ready to merge on HADM_ID:

```
['HADM_ID', 'WEIGHT', 'WEIGHT_UNITS', 'HEIGHT', 'HEIGHT_UNITS']
['111111', '74.5', 'kg', '65', 'inch']
['122222', '106.2', 'kg', '71', 'inch']
['133333', '54', 'kg', '60', ‘inch']
```

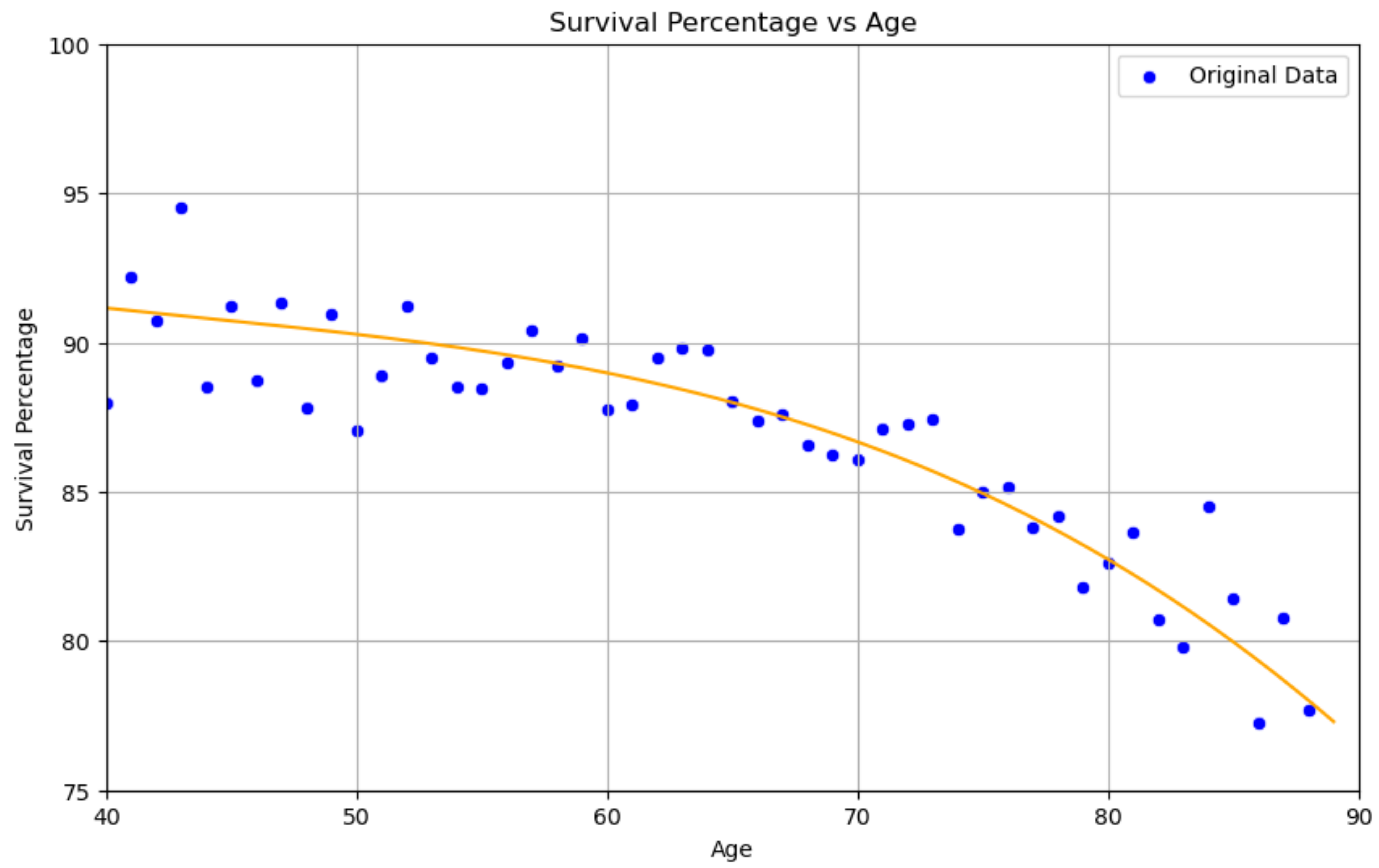
- To create the two referenced files run:
  - `python3 measurements.py - < weight.height.yaml`
  - `python3 measurements.py - < a1c.yaml`
- I received help in creating my visualizations from ChatGPT.
  - I did not send MIMIC III data to ChatGPT.

ICU Survival

This charts percentage of ICU patients that did not die in the hospital by age.



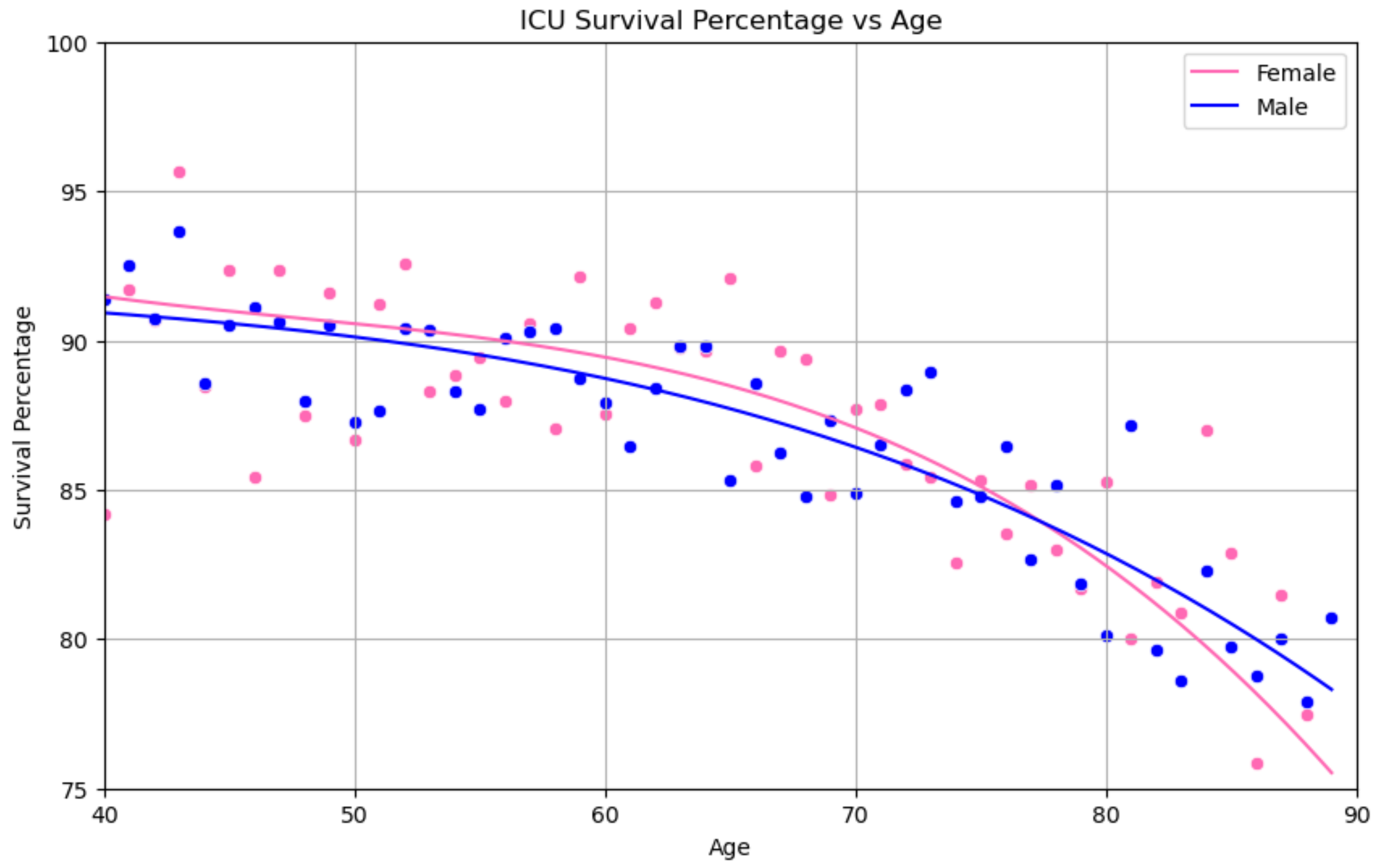
```
# Calculate survival percentage for each age
survival_percentage = df.groupby("age")["survived"].mean() * 100

def smooth(x, y):
    # Fit a polynomial curve for smoothing (degree 3)
    mask = ~np.isnan(y)  # Mask to avoid fitting NaN values
    if mask.sum() < 2:  # Ensure enough points for fitting
        return x, y
    coefficients = np.polyfit(x[mask], y[mask], deg=3)
    polynomial = np.poly1d(coefficients)

    # Generate smoothed points for plotting
    x_smooth = np.linspace(x.min(), x.max(), 500)
    y_smooth = polynomial(x_smooth)
    return x_smooth, y_smooth

x, y = survival_percentage.index, survival_percentage.values
x_smooth, y_smooth = smooth(x, y)

# Plot the smoothed curve
plt.figure(figsize=(10, 6))
sns.lineplot(x=x_smooth, y=y_smooth, color="orange")
sns.scatterplot(x=x, y=y, color="blue", label="Original Data")
plt.xlim(40, 90)
plt.ylim(75, 100)
plt.xlabel("Age")
plt.ylabel("Survival Percentage")
plt.title("Survival Percentage vs Age")
plt.grid(True)
plt.legend()
plt.show()
```

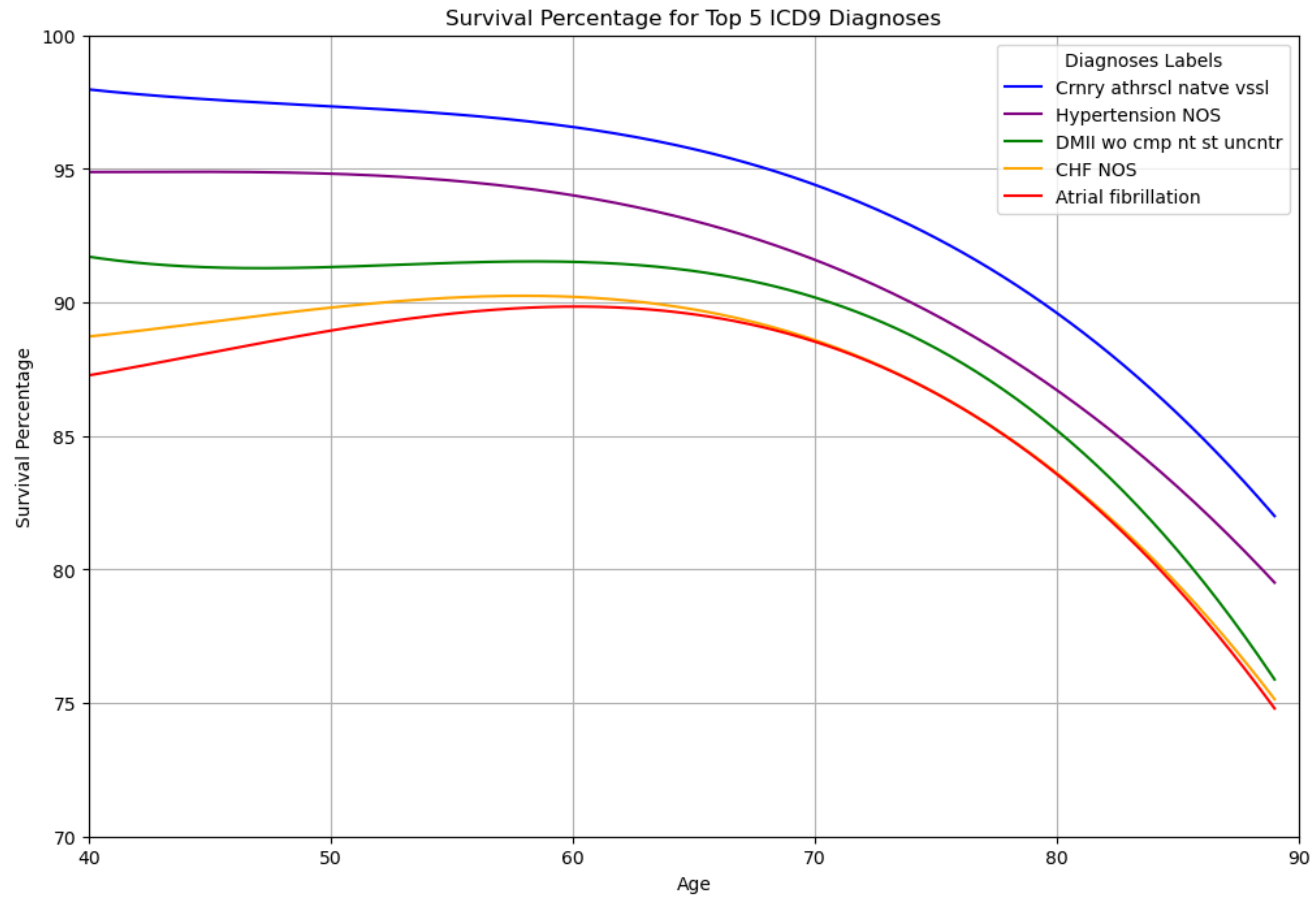


ICU Survival, cont.

Same graph showing male and female lines.

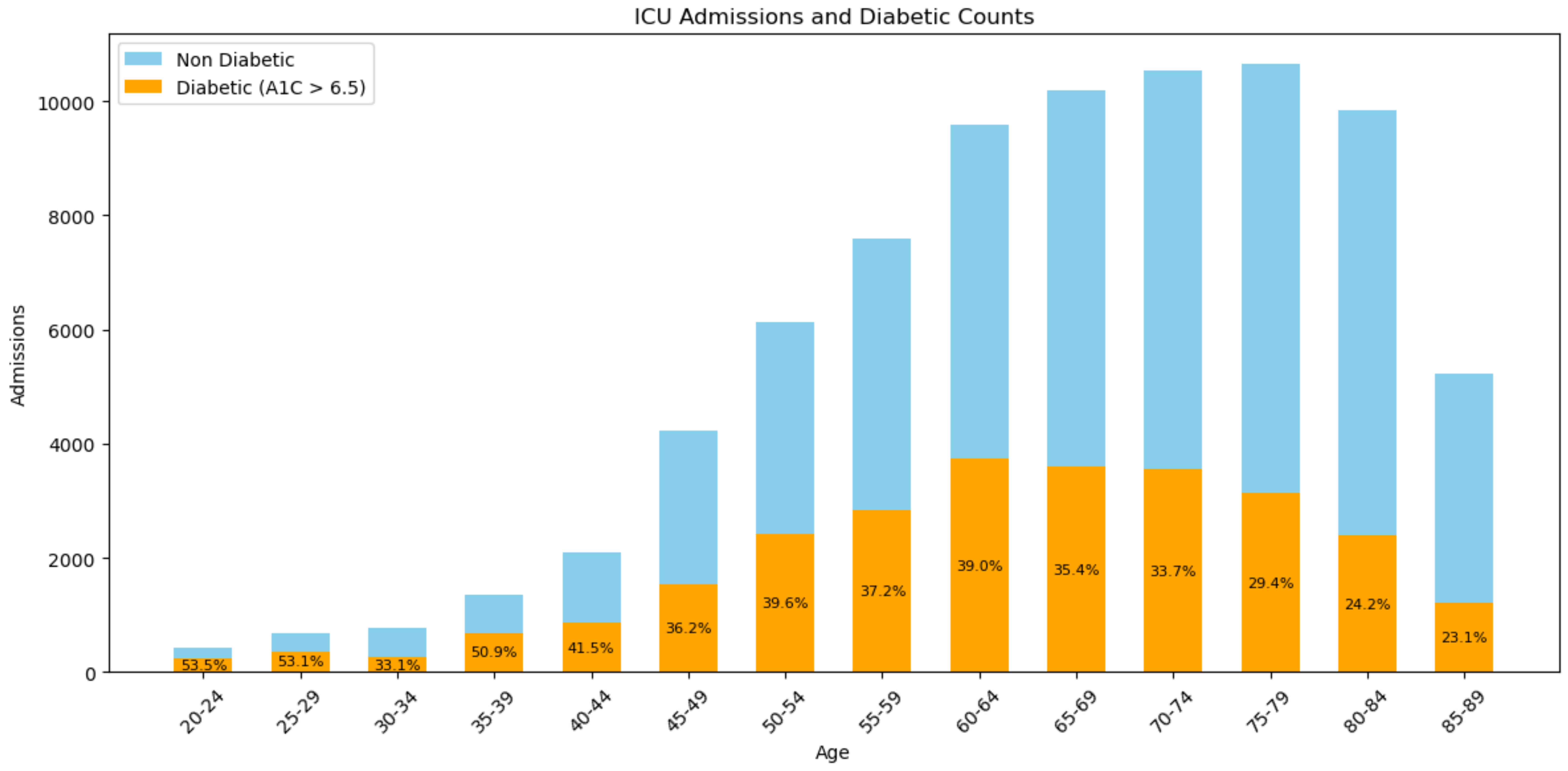
Survival Differences for Top 5 Diagnoses



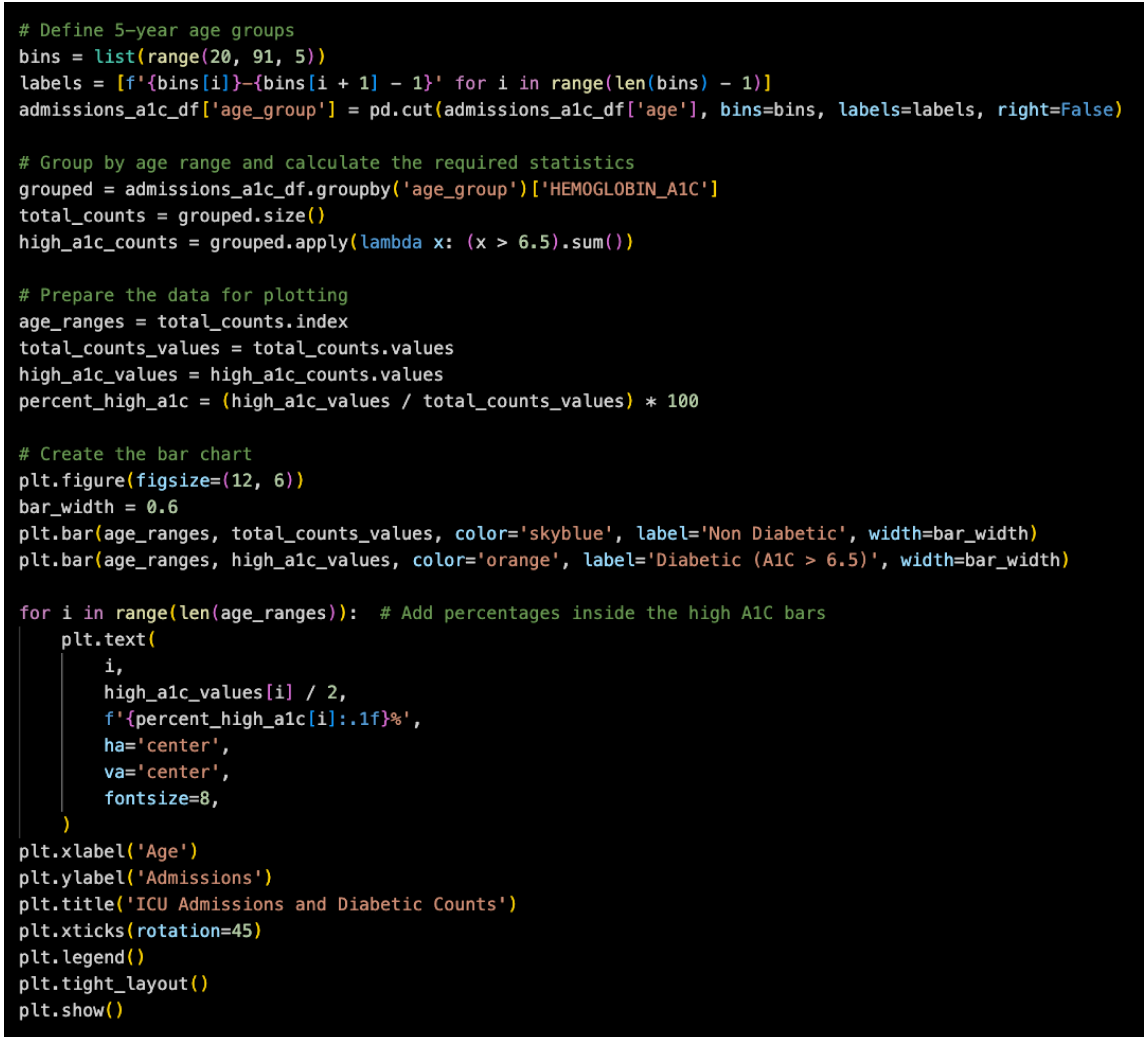


A1C

I was alarmed by the high percentage of diabetic patients, especially in the younger age groups.



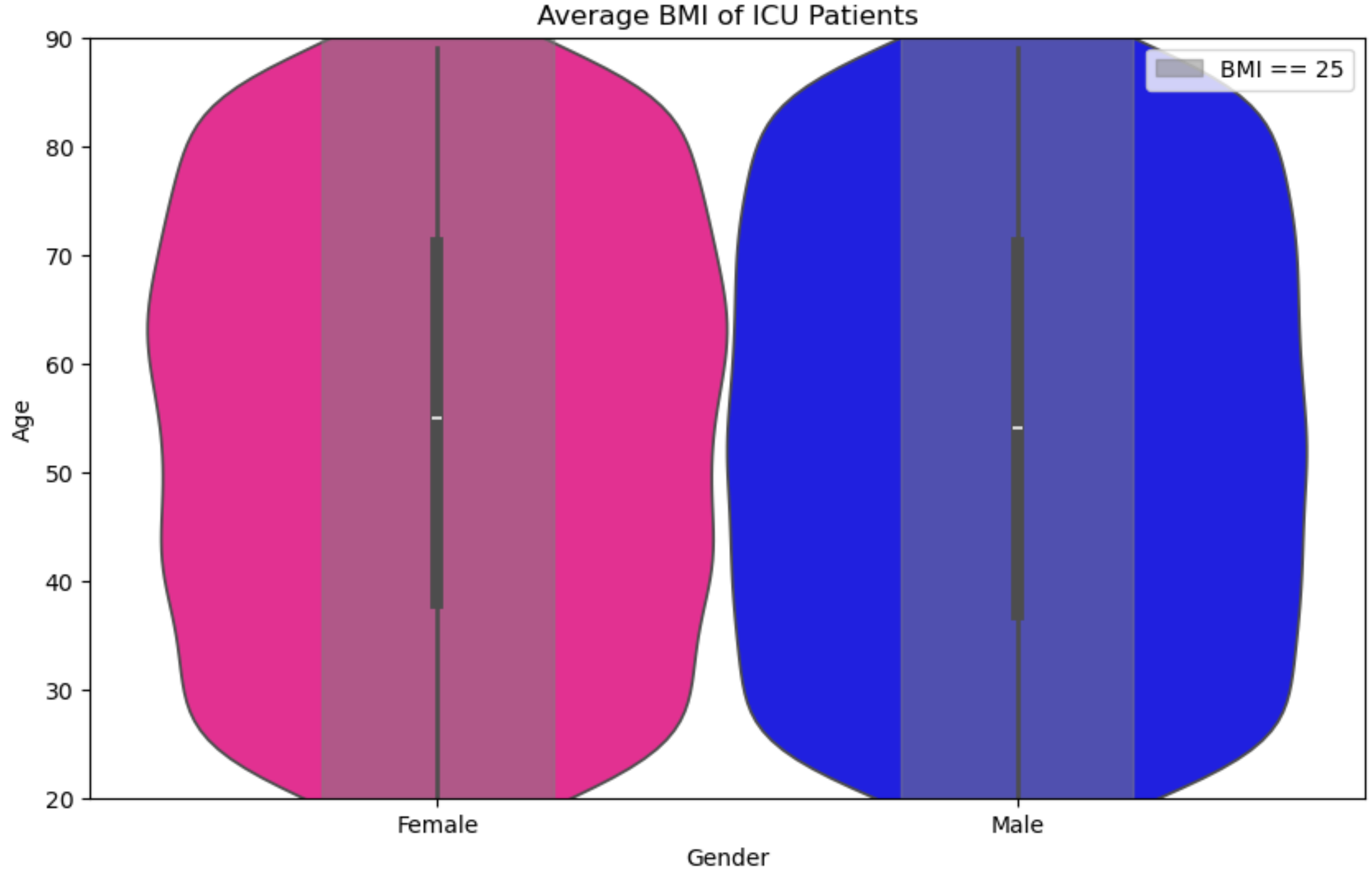
A1C Code



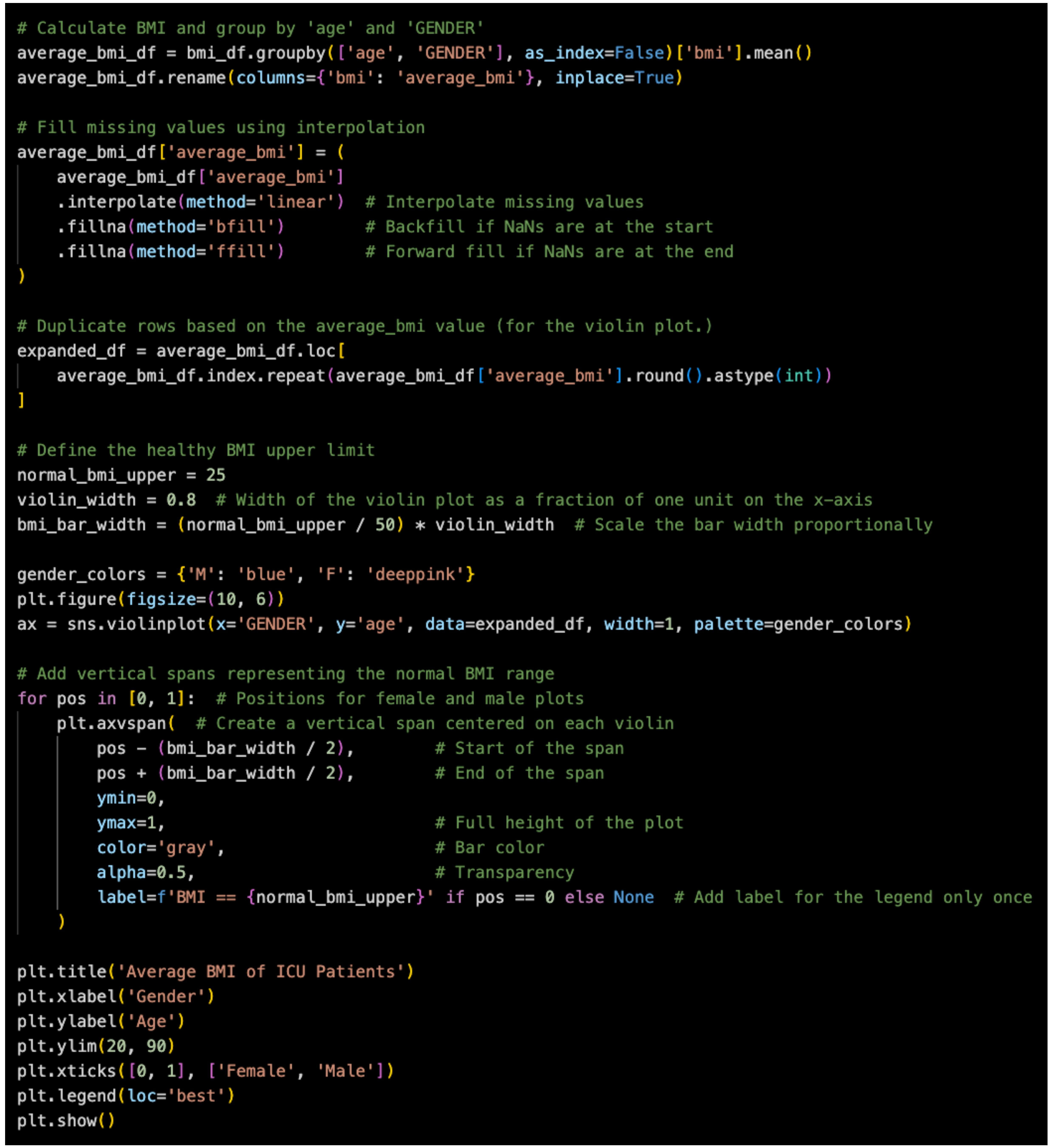
```python
# Define 5-year age groups
bins = list(range(20, 91, 5))
labels = [f'{bins[i]}-{bins[i + 1] - 1}' for i in range(len(bins) - 1)]
admissions_a1c_df['age_group'] = pd.cut(admissions_a1c_df['age'], bins=bins, labels=labels, right=False)

# Group by age range and calculate the required statistics
grouped = admissions_a1c_df.groupby('age_group')['HEMOGLOBIN_A1C']
total_counts = grouped.size()
high_a1c_counts = grouped.apply(lambda x: (x > 6.5).sum())

# Prepare the data for plotting
age_ranges = total_counts.index
total_counts_values = total_counts.values
high_a1c_values = high_a1c_counts.values
percent_high_a1c = (high_a1c_values / total_counts_values) * 100

# Create the bar chart
plt.figure(figsize=(12, 6))
bar_width = 0.6
plt.bar(age_ranges, total_counts_values, color='skyblue', label='Non Diabetic', width=bar_width)
plt.bar(age_ranges, high_a1c_values, color='orange', label='Diabetic (A1C > 6.5)', width=bar_width)

for i in range(len(age_ranges)):  # Add percentages inside the high A1C bars
    plt.text(
        i,
        high_a1c_values[i] / 2,
        f'{percent_high_a1c[i]:.1f}%',
        ha='center',
        va='center',
        fontsize=8,
    )
plt.xlabel('Age')
plt.ylabel('Admissions')
plt.title('ICU Admissions and Diabetic Counts')
plt.xticks(rotation=45)
plt.legend()
plt.tight_layout()
plt.show()
```

BMI

BMI is represented by the width of the violin plots.

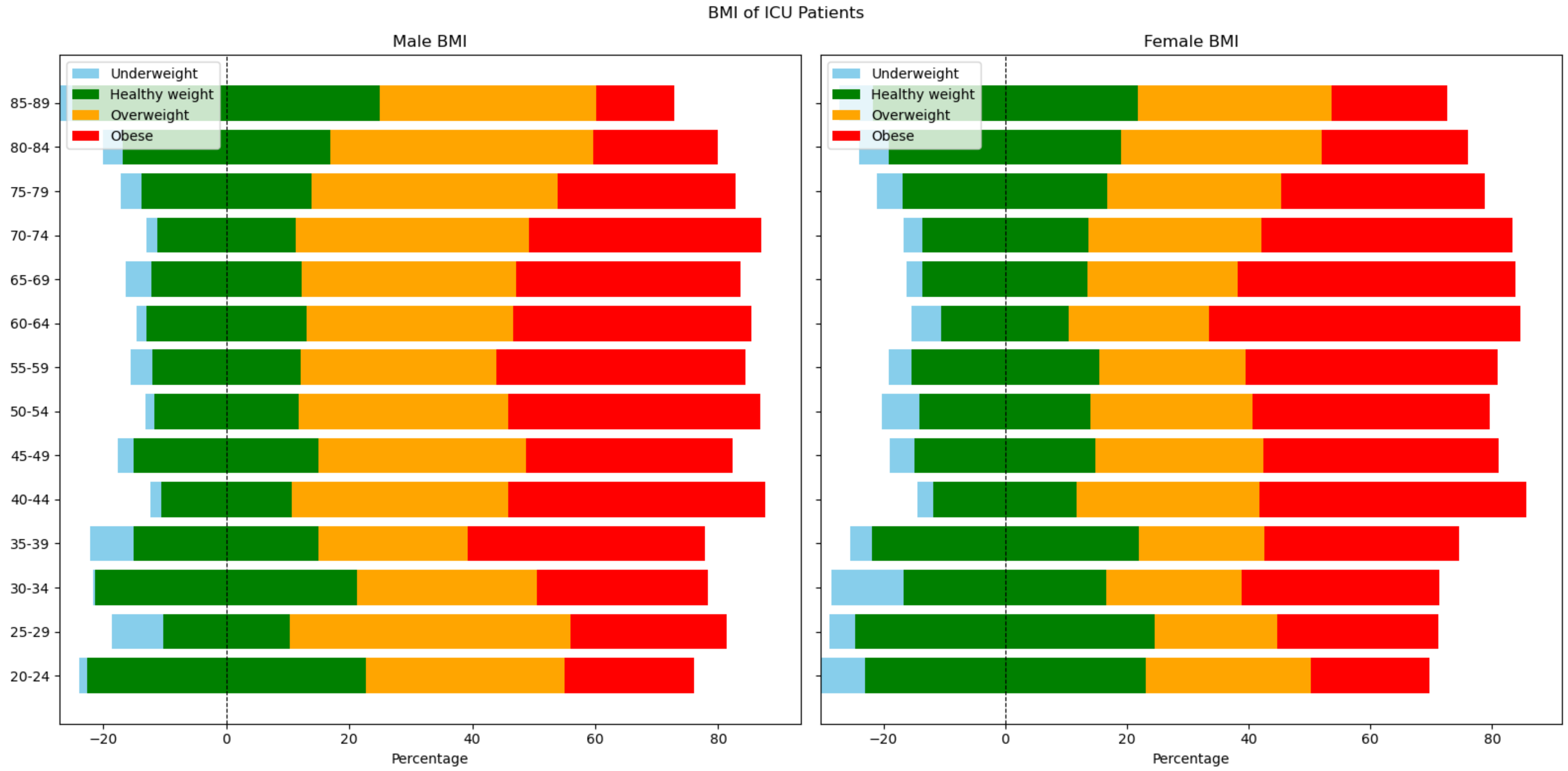


BMI:
Visualization Code



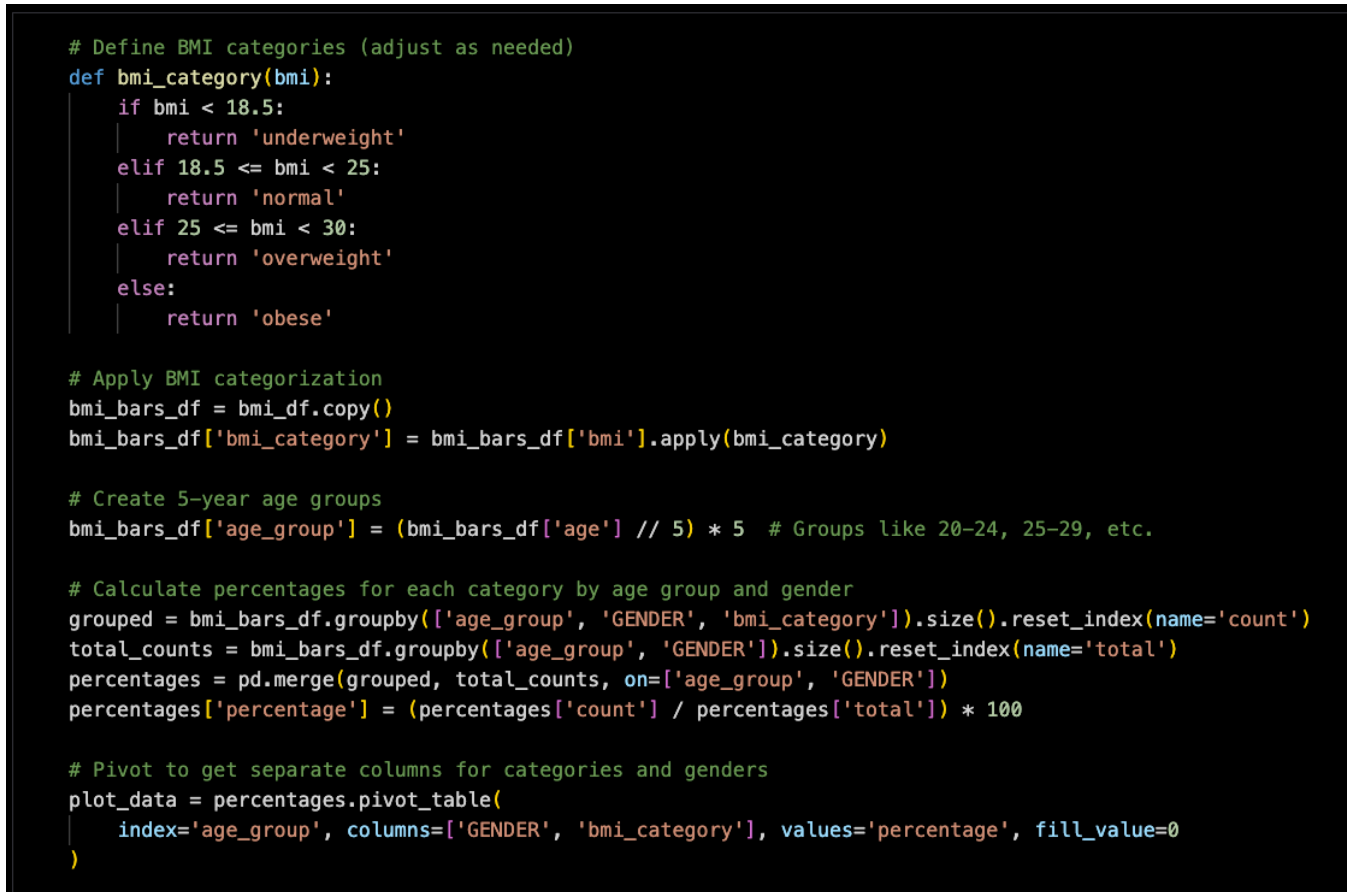
```python
# Calculate BMI and group by 'age' and 'GENDER'
average_bmi_df = bmi_df.groupby(['age', 'GENDER'], as_index=False)['bmi'].mean()
average_bmi_df.rename(columns={'bmi': 'average_bmi'}, inplace=True)

# Fill missing values using interpolation
average_bmi_df['average_bmi'] = (
    average_bmi_df['average_bmi']
    .interpolate(method='linear')  # Interpolate missing values
    .fillna(method='bfill')        # Backfill if NaNs are at the start
    .fillna(method='ffill')        # Forward fill if NaNs are at the end
)

# Duplicate rows based on the average_bmi value (for the violin plot.)
expanded_df = average_bmi_df.loc[
    average_bmi_df.index.repeat(average_bmi_df['average_bmi'].round().astype(int))
]

# Define the healthy BMI upper limit
normal_bmi_upper = 25
violin_width = 0.8  # Width of the violin plot as a fraction of one unit on the x-axis
bmi_bar_width = (normal_bmi_upper / 50) * violin_width  # Scale the bar width proportionally

gender_colors = {'M': 'blue', 'F': 'deeppink'}
plt.figure(figsize=(10, 6))
ax = sns.violinplot(x='GENDER', y='age', data=expanded_df, width=1, palette=gender_colors)

# Add vertical spans representing the normal BMI range
for pos in [0, 1]:  # Positions for female and male plots
    plt.axvspan(  # Create a vertical span centered on each violin
        pos - (bmi_bar_width / 2),        # Start of the span
        pos + (bmi_bar_width / 2),        # End of the span
        ymin=0,
        ymax=1,                           # Full height of the plot
        color='gray',                     # Bar color
        alpha=0.5,                        # Transparency
        label=f'BMI == {normal_bmi_upper}' if pos == 0 else None  # Add label for the legend only once
    )

plt.title('Average BMI of ICU Patients')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.ylim(20, 90)
plt.xticks([0, 1], ['Female', 'Male'])
plt.legend(loc='best')
plt.show()
```

BMI Ranges



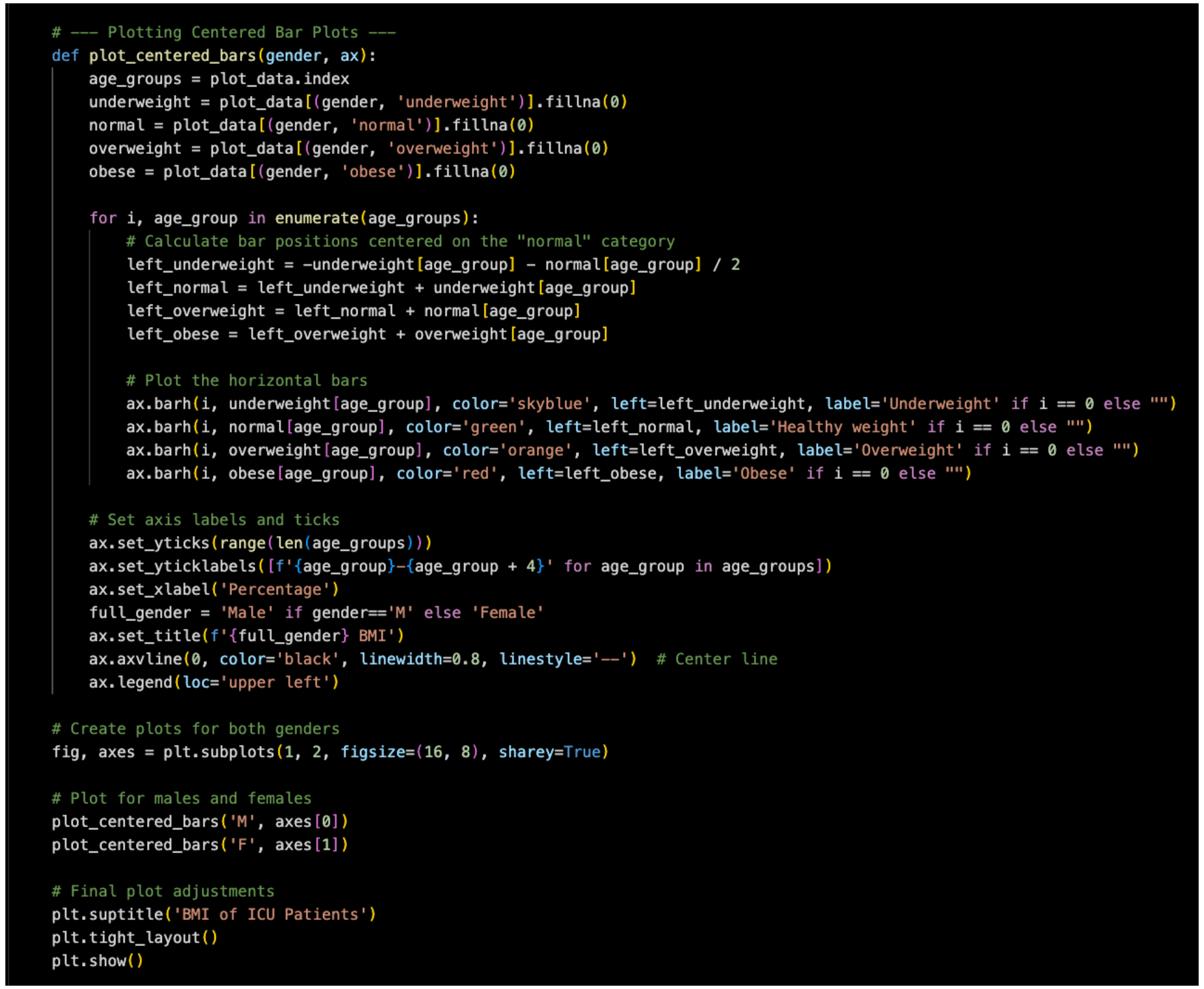
BMI of ICU Patients
Male BMI
Female BMI
Underweight
Healthy weight
Overweight
Obese
85-89
80-84
75-79
70-74
65-69
60-64
55-59
50-54
45-49
40-44
35-39
30-34
25-29
20-24
−20
0
20
40
60
80
Percentage

BMI Ranges



```python
# Define BMI categories (adjust as needed)
def bmi_category(bmi):
    if bmi < 18.5:
        return 'underweight'
    elif 18.5 <= bmi < 25:
        return 'normal'
    elif 25 <= bmi < 30:
        return 'overweight'
    else:
        return 'obese'

# Apply BMI categorization
bmi_bars_df = bmi_df.copy()
bmi_bars_df['bmi_category'] = bmi_bars_df['bmi'].apply(bmi_category)

# Create 5-year age groups
bmi_bars_df['age_group'] = (bmi_bars_df['age'] // 5) * 5  # Groups like 20-24, 25-29, etc.

# Calculate percentages for each category by age group and gender
grouped = bmi_bars_df.groupby(['age_group', 'GENDER', 'bmi_category']).size().reset_index(name='count')
total_counts = bmi_bars_df.groupby(['age_group', 'GENDER']).size().reset_index(name='total')
percentages = pd.merge(grouped, total_counts, on=['age_group', 'GENDER'])
percentages['percentage'] = (percentages['count'] / percentages['total']) * 100

# Pivot to get separate columns for categories and genders
plot_data = percentages.pivot_table(
    index='age_group', columns=['GENDER', 'bmi_category'], values='percentage', fill_value=0
)
```

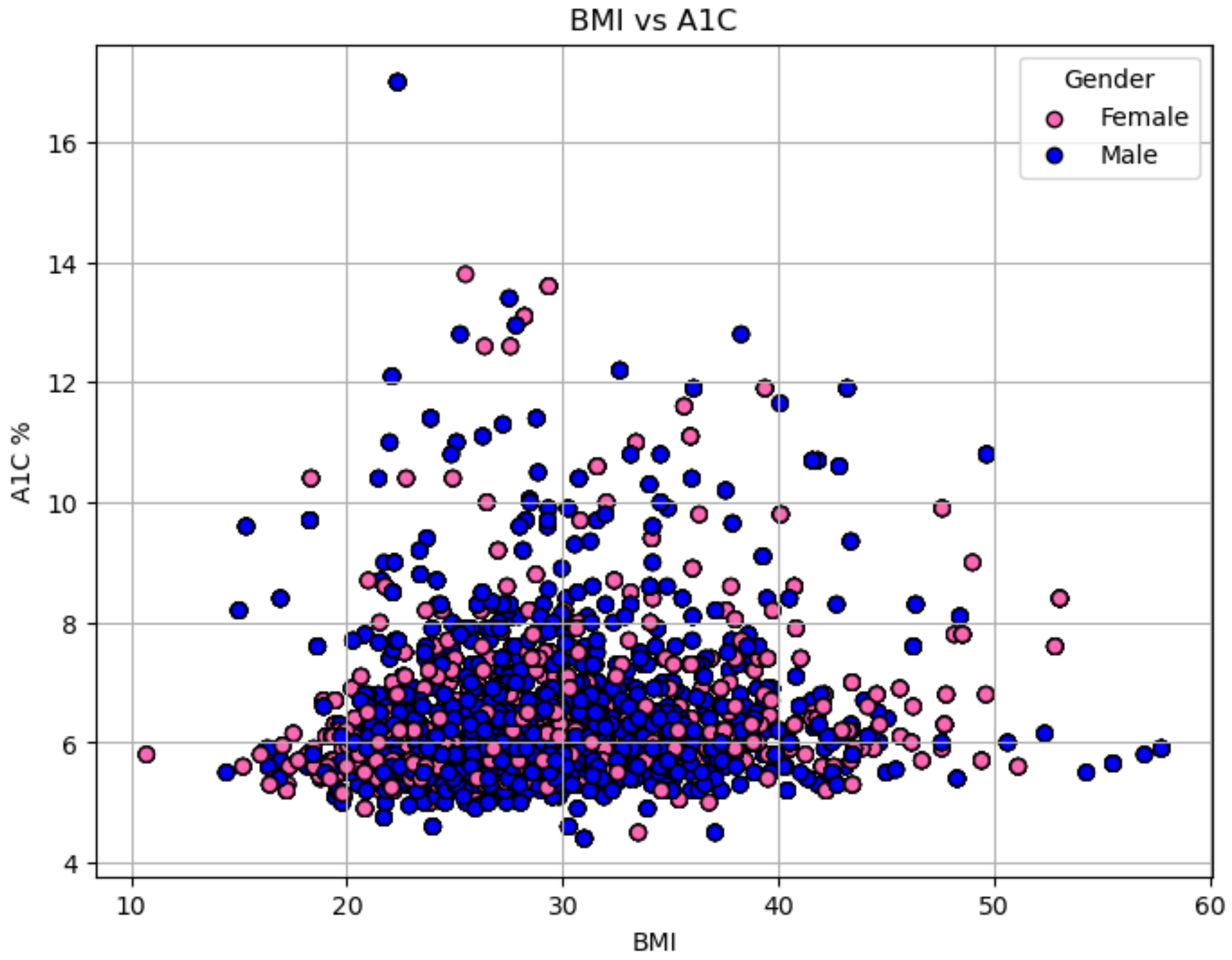
Continued



```python
# --- Plotting Centered Bar Plots ---
def plot_centered_bars(gender, ax):
    age_groups = plot_data.index
    underweight = plot_data[(gender, 'underweight')].fillna(0)
    normal = plot_data[(gender, 'normal')].fillna(0)
    overweight = plot_data[(gender, 'overweight')].fillna(0)
    obese = plot_data[(gender, 'obese')].fillna(0)

    for i, age_group in enumerate(age_groups):
        # Calculate bar positions centered on the "normal" category
        left_underweight = -underweight[age_group] - normal[age_group] / 2
        left_normal = left_underweight + underweight[age_group]
        left_overweight = left_normal + normal[age_group]
        left_obese = left_overweight + overweight[age_group]

        # Plot the horizontal bars
        ax.barh(i, underweight[age_group], color='skyblue', left=left_underweight, label='Underweight' if i == 0 else "")
        ax.barh(i, normal[age_group], color='green', left=left_normal, label='Healthy weight' if i == 0 else "")
        ax.barh(i, overweight[age_group], color='orange', left=left_overweight, label='Overweight' if i == 0 else "")
        ax.barh(i, obese[age_group], color='red', left=left_obese, label='Obese' if i == 0 else "")

    # Set axis labels and ticks
    ax.set_yticks(range(len(age_groups)))
    ax.set_yticklabels([f'{age_group}-{age_group + 4}' for age_group in age_groups])
    ax.set_xlabel('Percentage')
    full_gender = 'Male' if gender=='M' else 'Female'
    ax.set_title(f'{full_gender} BMI')
    ax.axvline(0, color='black', linewidth=0.8, linestyle='--')  # Center line
    ax.legend(loc='upper left')

# Create plots for both genders
fig, axes = plt.subplots(1, 2, figsize=(16, 8), sharey=True)

# Plot for males and females
plot_centered_bars('M', axes[0])
plot_centered_bars('F', axes[1])

# Final plot adjustments
plt.suptitle('BMI of ICU Patients')
plt.tight_layout()
plt.show()
```

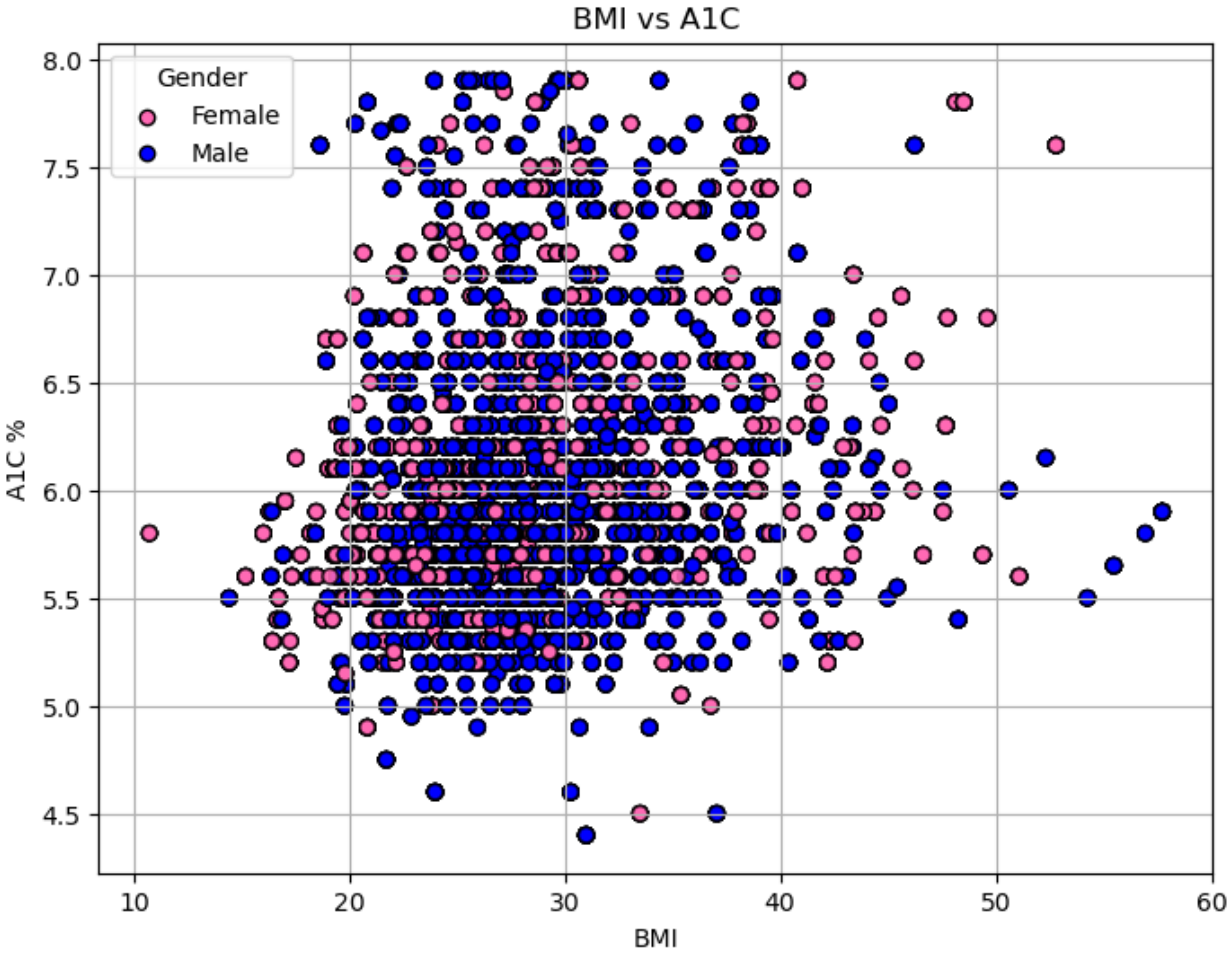
BMI vs A1C

My hypothesis that I would see A1C increase with BMI was not supported by this graph.

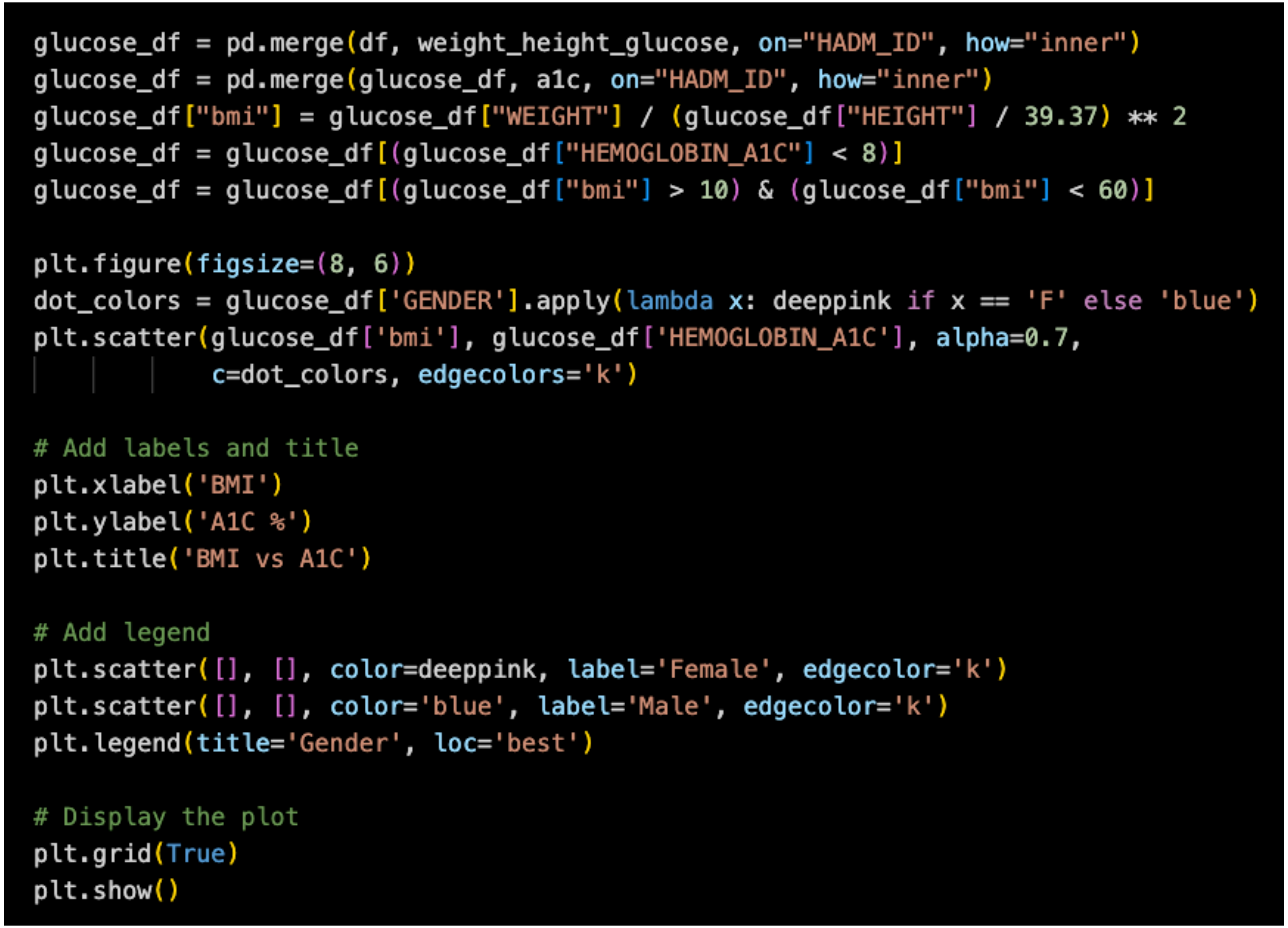


BMI vs A1C, Cont.

When I zoomed in (by only looking at A1Cs < 8%), I can now imagine that A1C is increasing with BMI.



BMI vs A1C, Code



```
glucose_df = pd.merge(df, weight_height_glucose, on="HADM_ID", how="inner")
glucose_df = pd.merge(glucose_df, a1c, on="HADM_ID", how="inner")
glucose_df["bmi"] = glucose_df["WEIGHT"] / (glucose_df["HEIGHT"] / 39.37) ** 2
glucose_df = glucose_df[(glucose_df["HEMOGLOBIN_A1C"] < 8)]
glucose_df = glucose_df[(glucose_df["bmi"] > 10) & (glucose_df["bmi"] < 60)]

plt.figure(figsize=(8, 6))
dot_colors = glucose_df['GENDER'].apply(lambda x: deeppink if x == 'F' else 'blue')
plt.scatter(glucose_df['bmi'], glucose_df['HEMOGLOBIN_A1C'], alpha=0.7,
            c=dot_colors, edgecolors='k')

# Add labels and title
plt.xlabel('BMI')
plt.ylabel('A1C %')
plt.title('BMI vs A1C')

# Add legend
plt.scatter([], [], color=deeppink, label='Female', edgecolor='k')
plt.scatter([], [], color='blue', label='Male', edgecolor='k')
plt.legend(title='Gender', loc='best')

# Display the plot
plt.grid(True)
plt.show()
```