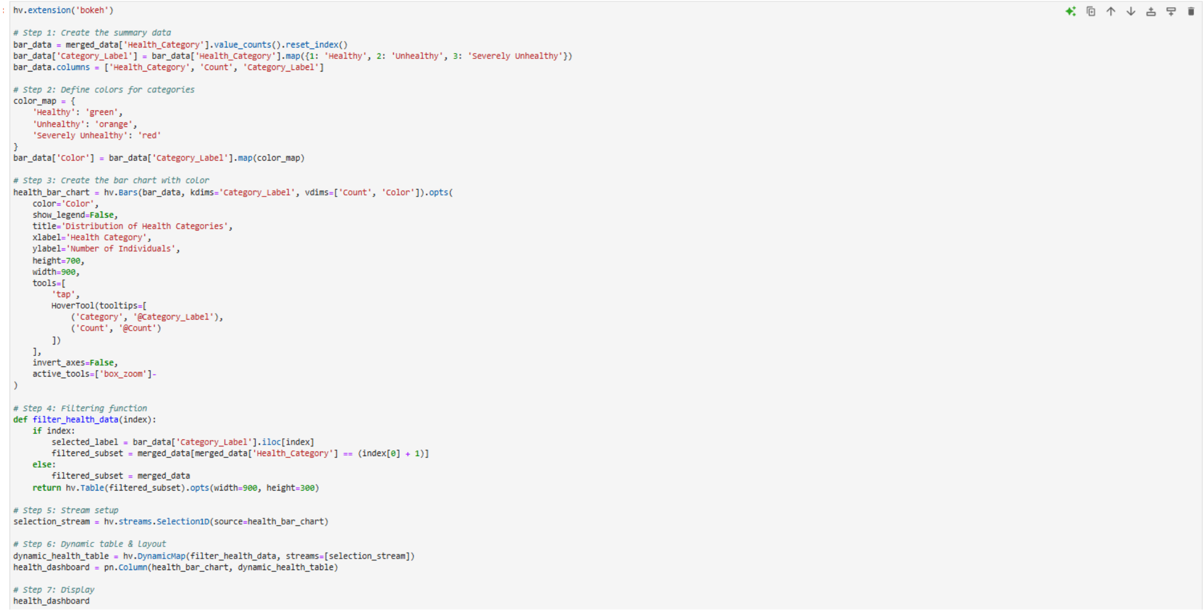
# SIT220 DATA WRANGLING

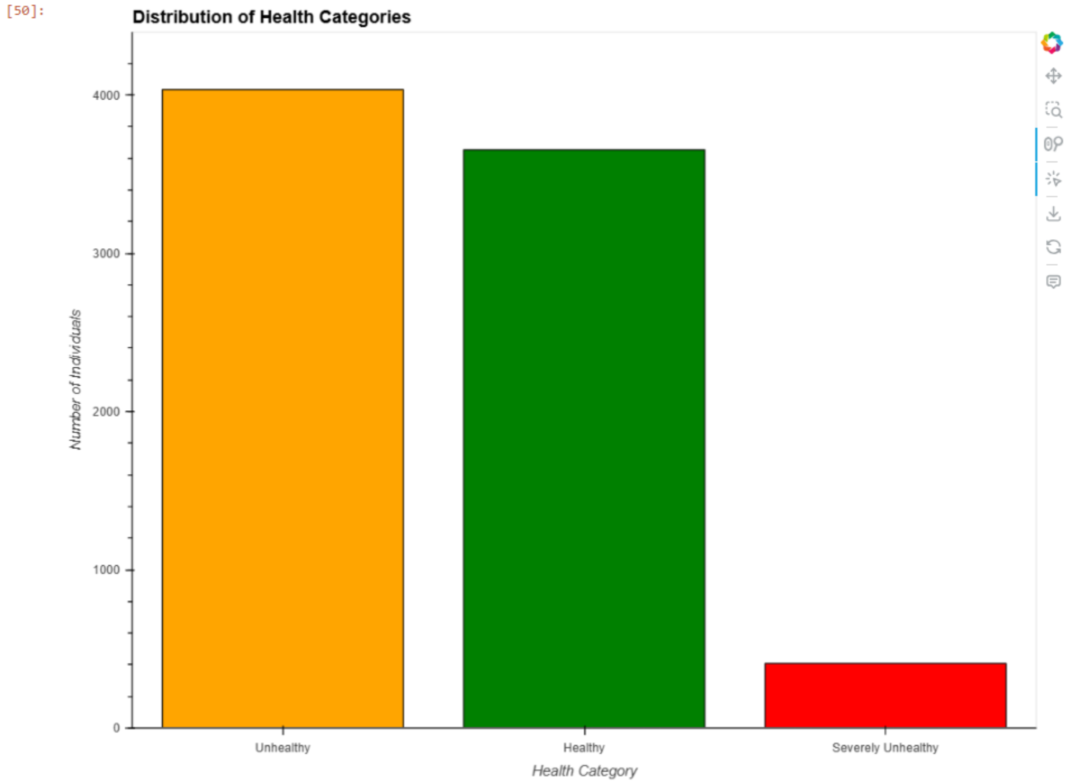
## Task 1.7HD MINING

### Visualization 1:

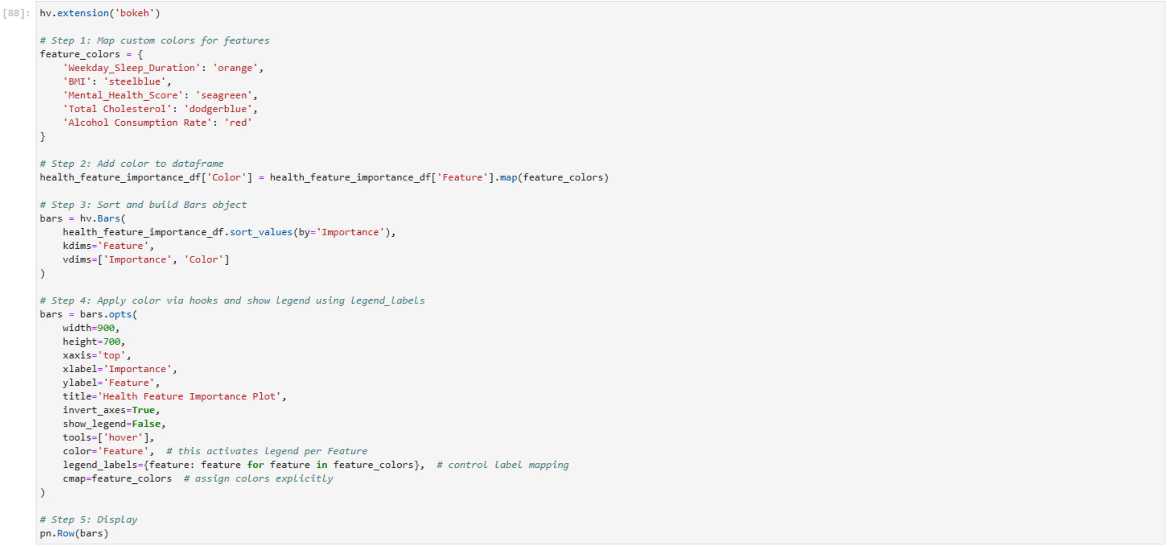


```
hv.extension('bokeh')

# Step 1: Create the summary data
bar_data = merged_data['Health_Category'].value_counts().reset_index()
bar_data['Category_Label'] = bar_data['Health_Category'].map({1: 'Healthy', 2: 'Unhealthy', 3: 'Severely Unhealthy'})
bar_data.columns = ['Health_Category', 'Count', 'Category_Label']

# Step 2: Define colors for categories
color_map = {
    'Healthy': 'green',
    'Unhealthy': 'orange',
    'Severely Unhealthy': 'red'
}
bar_data['Color'] = bar_data['Category_Label'].map(color_map)

# Step 3: Create the bar chart with color
health_bar_chart = hv.Bars(bar_data, kdims='Category_Label', vdims=['Count', 'Color']).opts(
    color='Color',
    show_legend=False,
    title='Distribution of Health Categories',
    xlabel='Health Category',
    ylabel='Number of Individuals',
    height=700,
    width=900,
    tools=[
        'tap',
        HoverTool(tooltips=[
            ('Category', '@Category_Label'),
            ('Count', '@Count')
        ])
    ],
    invert_axes=False,
    active_tools=['box_zoom']-
)

# Step 4: Filtering function
def filter_health_data(index):
    if index:
        selected_label = bar_data['Category_Label'].iloc[index]
        filtered_subset = merged_data[merged_data['Health_Category'] == (index[0] + 1)]
    else:
        filtered_subset = merged_data
    return hv.Table(filtered_subset).opts(width=900, height=300)

# Step 5: Stream setup
selection_stream = hv.streams.Selection1D(source=health_bar_chart)

# Step 6: Dynamic table & Layout
dynamic_health_table = hv.DynamicMap(filter_health_data, streams=[selection_stream])
health_dashboard = pn.Column(health_bar_chart, dynamic_health_table)

# Step 7: Display
health_dashboard
```

### Output:

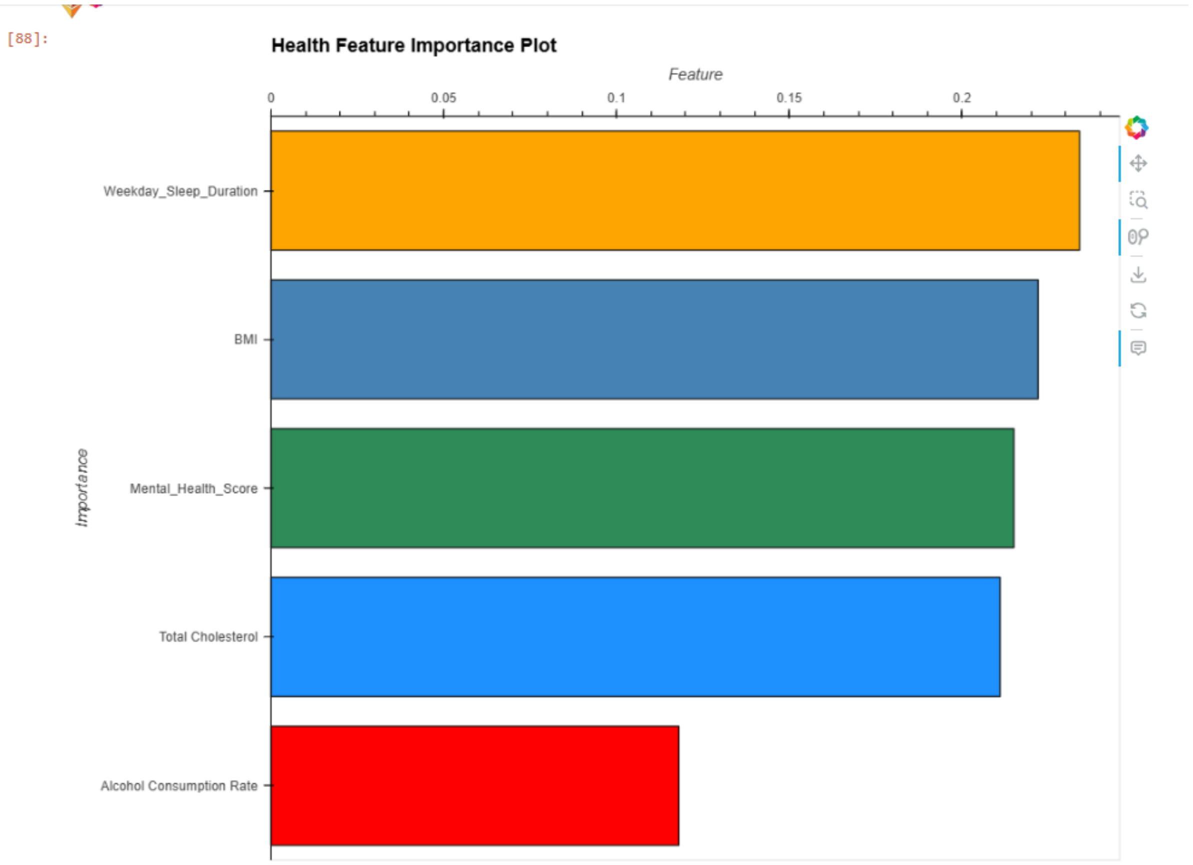


## Visualization 2:

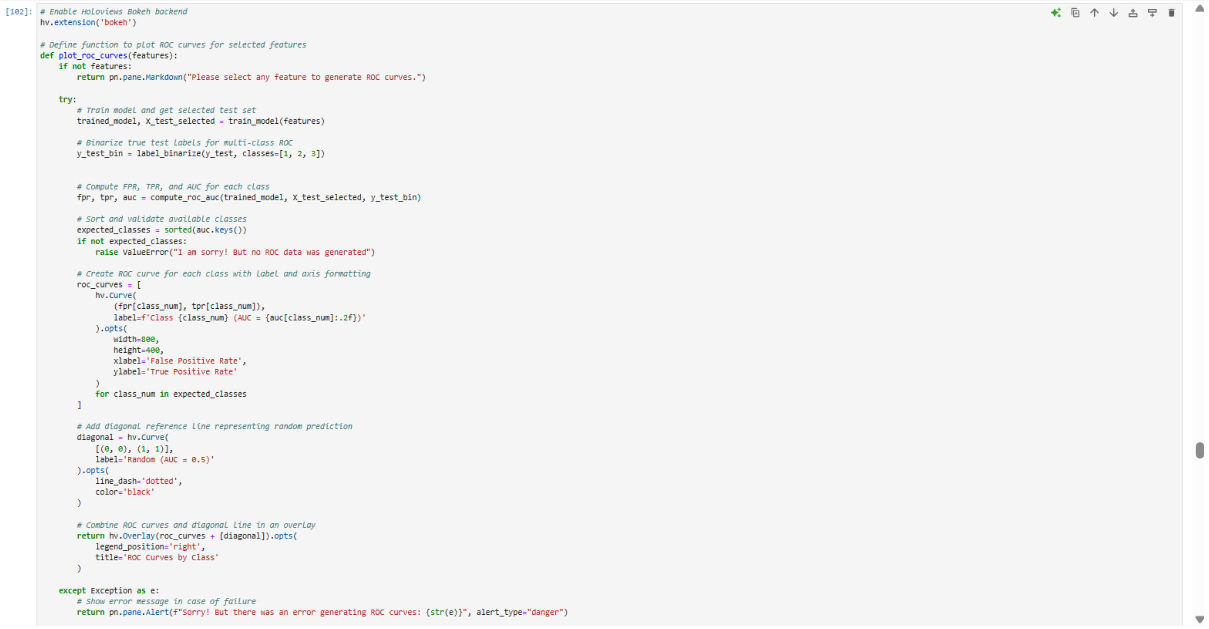


```
hv.extension('bokeh')

# Step 1: Map custom colors for features
feature_colors = {
    'Weekday_Sleep_Duration': 'orange',
    'BMI': 'steelblue',
    'Mental_Health_Score': 'seagreen',
    'Total Cholesterol': 'dodgerblue',
    'Alcohol Consumption Rate': 'red'
}

# Step 2: Add color to dataframe
health_feature_importance_df['Color'] = health_feature_importance_df['Feature'].map(feature_colors)

# Step 3: Sort and build Bars object
bars = hv.Bars(
    health_feature_importance_df.sort_values(by='Importance'),
    kdims='Feature',
    vdims=['Importance', 'Color']
)

# Step 4: Apply color via hooks and show legend using legend_labels
bars = bars.opts(
    width=900,
    height=700,
    xaxis='top',
    xlabel='Importance',
    ylabel='Feature',
    title='Health Feature Importance Plot',
    invert_axes=True,
    show_legend=False,
    tools=['hover'],
    color='Feature',  # this activates legend per Feature
    legend_labels={feature: feature for feature in feature_colors},  # control label mapping
    cmap=feature_colors  # assign colors explicitly
)

# Step 5: Display
pn.Row(bars)
```

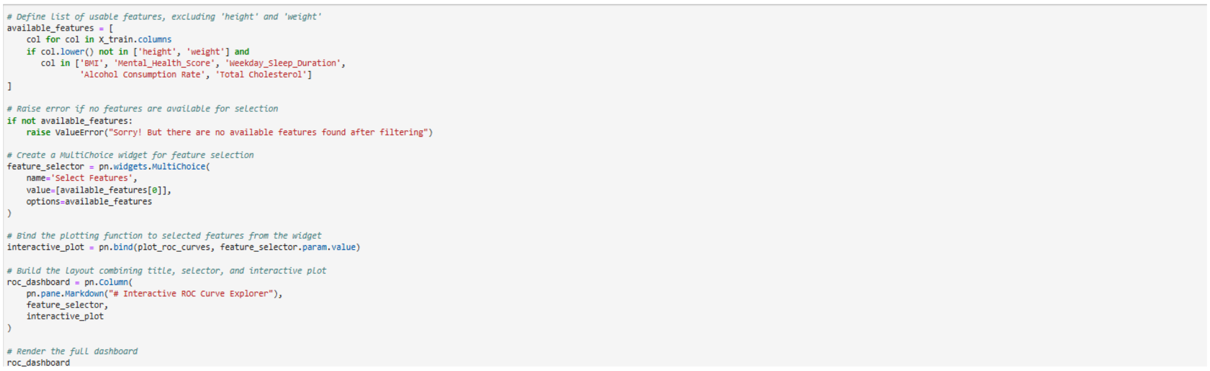
## Output:



## Visualization 3:

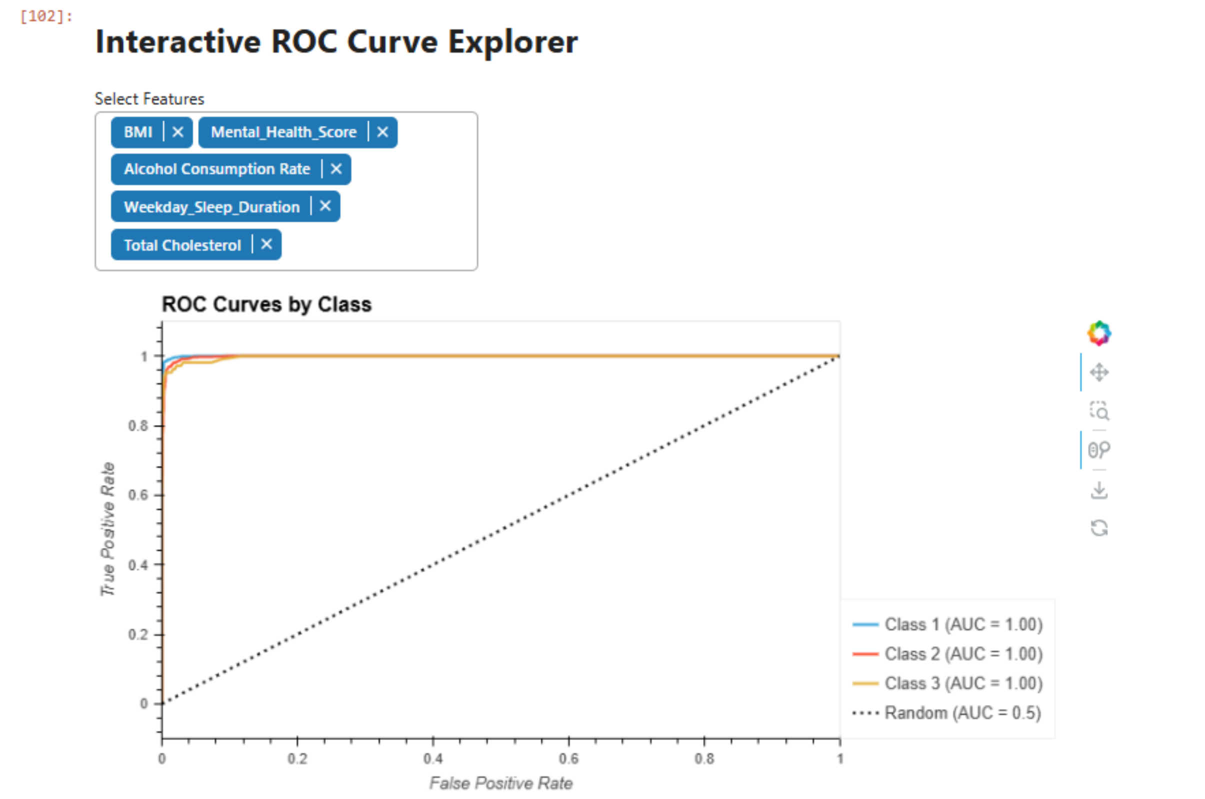


```
[102]: # Enable Holoviews Bokeh backend
hv.extension('bokeh')

# Define function to plot ROC curves for selected features
def plot_roc_curves(features):
    if not features:
        return pn.pane.Markdown("Please select any feature to generate ROC curves.")

    try:
        # Train model and get selected test set
        trained_model, X_test_selected = train_model(features)

        # Binarize true test labels for multi-class ROC
        y_test_bin = label_binarize(y_test, classes=[1, 2, 3])


        # Compute FPR, TPR, and AUC for each class
        fpr, tpr, auc = compute_roc_auc(trained_model, X_test_selected, y_test_bin)

        # Sort and validate available classes
        expected_classes = sorted(auc.keys())
        if not expected_classes:
            raise ValueError("I am sorry! But no ROC data was generated")

        # Create ROC curve for each class with label and axis formatting
        roc_curves = [
            hv.Curve(
                (fpr[class_num], tpr[class_num]),
                label=f'Class {class_num} (AUC = {auc[class_num]:.2f})'
            ).opts(
                width=800,
                height=400,
                xlabel='False Positive Rate',
                ylabel='True Positive Rate'
            )
            for class_num in expected_classes
        ]

        # Add diagonal reference line representing random prediction
        diagonal = hv.Curve(
            [(0, 0), (1, 1)],
            label='Random (AUC = 0.5)'
        ).opts(
            line_dash='dotted',
            color='black'
        )

        # Combine ROC curves and diagonal line in an overlay
        return hv.Overlay(roc_curves + [diagonal]).opts(
            legend_position='right',
            title='ROC Curves by Class'
        )

    except Exception as e:
        # Show error message in case of failure
        return pn.pane.Alert(f"Sorry! But there was an error generating ROC curves: {str(e)}", alert_type="danger")
```



```
# Define list of usable features, excluding 'height' and 'weight'
available_features = [
    col for col in X_train.columns
    if col.lower() not in ['height', 'weight'] and
       col in ['BMI', 'Mental_Health_Score', 'Weekday_Sleep_Duration',
               'Alcohol Consumption Rate', 'Total Cholesterol']
]

# Raise error if no features are available for selection
if not available_features:
    raise ValueError("Sorry! But there are no available features found after filtering")

# Create a MultiChoice widget for feature selection
feature_selector = pn.widgets.MultiChoice(
    name='Select Features',
    value=[available_features[0]],
    options=available_features
)

# Bind the plotting function to selected features from the widget
interactive_plot = pn.bind(plot_roc_curves, feature_selector.param.value)

# Build the layout combining title, selector, and interactive plot
roc_dashboard = pn.Column(
    pn.pane.Markdown("# Interactive ROC Curve Explorer"),
    feature_selector,
    interactive_plot
)

# Render the full dashboard
roc_dashboard
```

Output:



[102]:

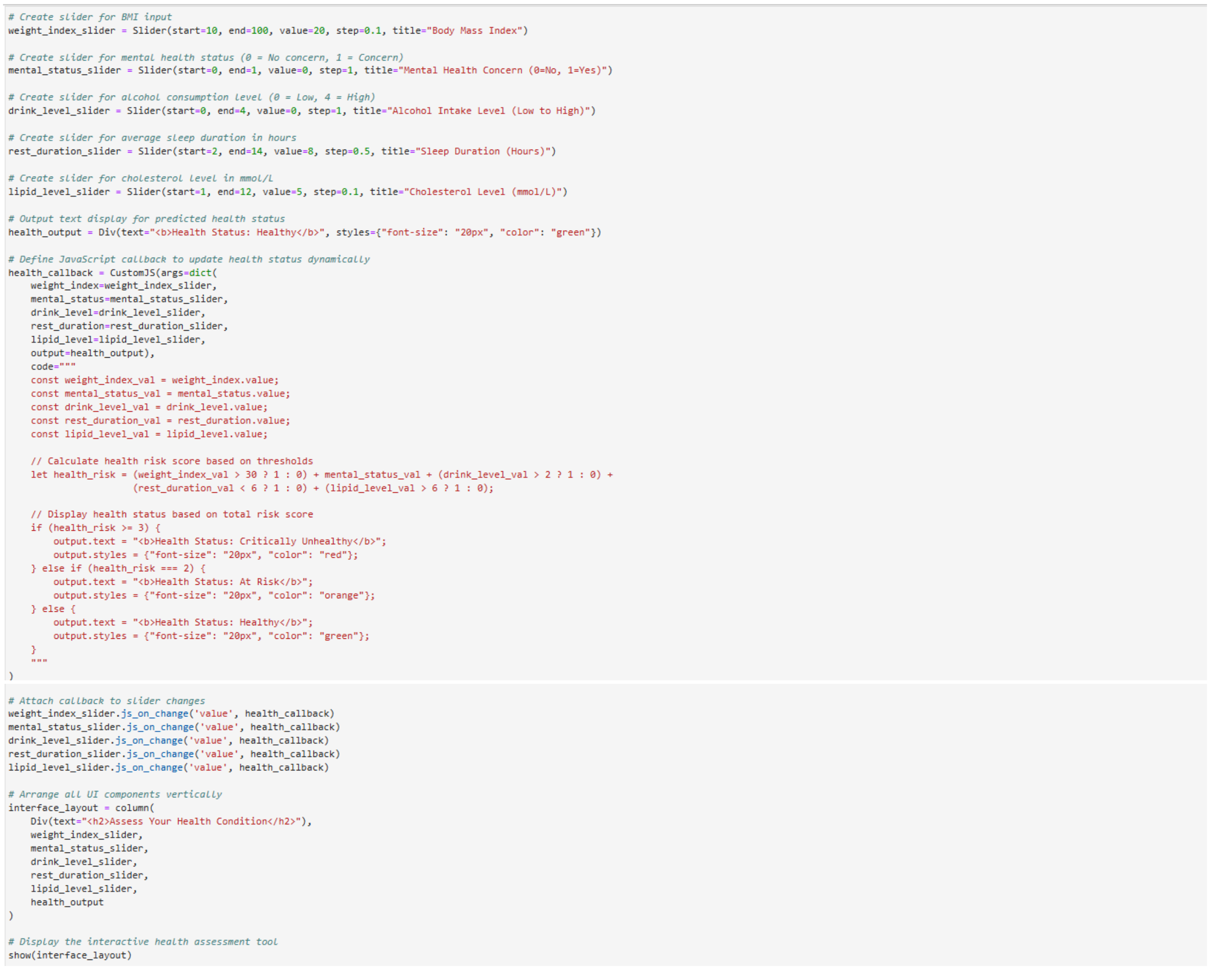
# Interactive ROC Curve Explorer

Select Features

BMI ✕ | Mental_Health_Score ✕ | Alcohol Consumption Rate ✕ | Weekday_Sleep_Duration ✕ | Total Cholesterol ✕

ROC Curves by Class

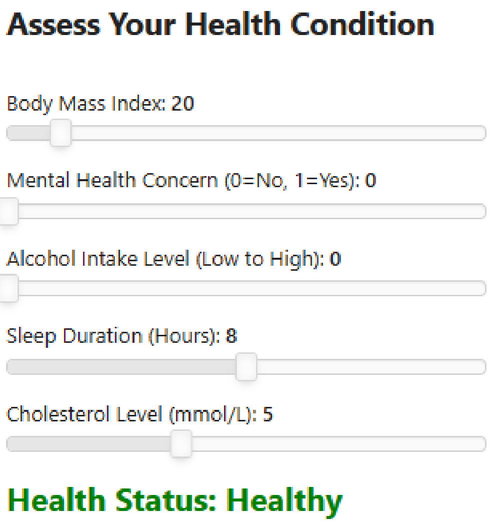
Visualization 4:



```
# Create slider for BMI input
weight_index_slider = Slider(start=10, end=100, value=20, step=0.1, title="Body Mass Index")

# Create slider for mental health status (0 = No concern, 1 = Concern)
mental_status_slider = Slider(start=0, end=1, value=0, step=1, title="Mental Health Concern (0=No, 1=Yes)")

# Create slider for alcohol consumption level (0 = Low, 4 = High)
drink_level_slider = Slider(start=0, end=4, value=0, step=1, title="Alcohol Intake Level (Low to High)")

# Create slider for average sleep duration in hours
rest_duration_slider = Slider(start=2, end=14, value=8, step=0.5, title="Sleep Duration (Hours)")

# Create slider for cholesterol level in mmol/L
lipid_level_slider = Slider(start=1, end=12, value=5, step=0.1, title="Cholesterol Level (mmol/L)")

# Output text display for predicted health status
health_output = Div(text="<b>Health Status: Healthy</b>", styles={"font-size": "20px", "color": "green"})

# Define JavaScript callback to update health status dynamically
health_callback = CustomJS(args=dict(
    weight_index=weight_index_slider,
    mental_status=mental_status_slider,
    drink_level=drink_level_slider,
    rest_duration=rest_duration_slider,
    lipid_level=lipid_level_slider,
    output=health_output),
    code="""
    const weight_index_val = weight_index.value;
    const mental_status_val = mental_status.value;
    const drink_level_val = drink_level.value;
    const rest_duration_val = rest_duration.value;
    const lipid_level_val = lipid_level.value;

    // Calculate health risk score based on thresholds
    let health_risk = (weight_index_val > 30 ? 1 : 0) + mental_status_val + (drink_level_val > 2 ? 1 : 0) +
                      (rest_duration_val < 6 ? 1 : 0) + (lipid_level_val > 6 ? 1 : 0);

    // Display health status based on total risk score
    if (health_risk >= 3) {
        output.text = "<b>Health Status: Critically Unhealthy</b>";
        output.styles = {"font-size": "20px", "color": "red"};
    } else if (health_risk === 2) {
        output.text = "<b>Health Status: At Risk</b>";
        output.styles = {"font-size": "20px", "color": "orange"};
    } else {
        output.text = "<b>Health Status: Healthy</b>";
        output.styles = {"font-size": "20px", "color": "green"};
    }
    """
)
```

```
# Attach callback to slider changes
weight_index_slider.js_on_change('value', health_callback)
mental_status_slider.js_on_change('value', health_callback)
drink_level_slider.js_on_change('value', health_callback)
rest_duration_slider.js_on_change('value', health_callback)
lipid_level_slider.js_on_change('value', health_callback)

# Arrange all UI components vertically
interface_layout = column(
    Div(text="<h2>Assess Your Health Condition</h2>"),
    weight_index_slider,
    mental_status_slider,
    drink_level_slider,
    rest_duration_slider,
    lipid_level_slider,
    health_output
)

# Display the interactive health assessment tool
show(interface_layout)
```

# Output:



## Assess Your Health Condition

Body Mass Index: 20

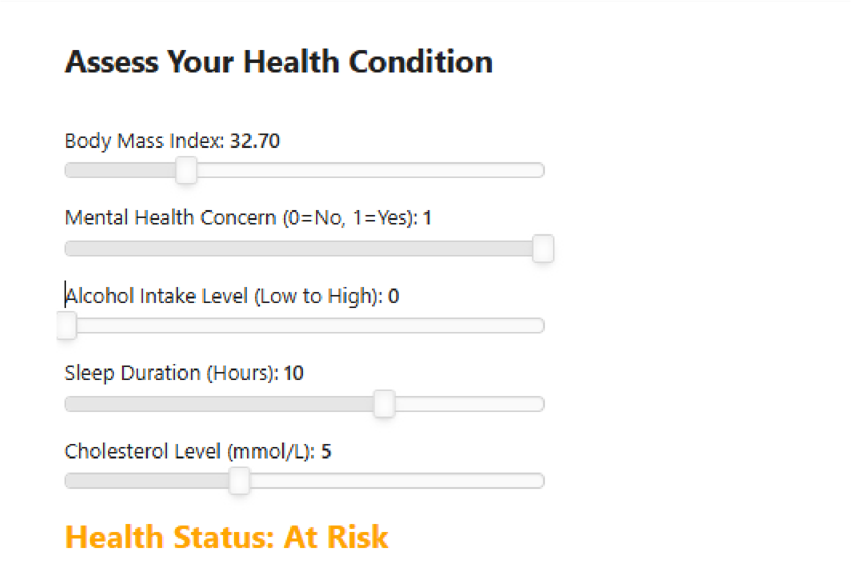
Mental Health Concern (0=No, 1=Yes): 0

Alcohol Intake Level (Low to High): 0

Sleep Duration (Hours): 8

Cholesterol Level (mmol/L): 5

**Health Status: Healthy**



Assess Your Health Condition

Body Mass Index: 32.70

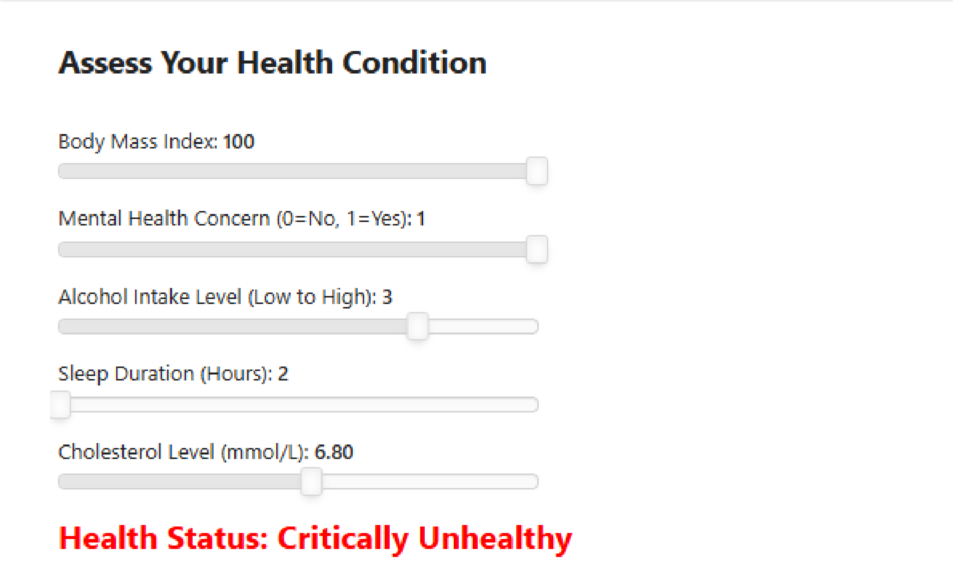
Mental Health Concern (0=No, 1=Yes): 1

Alcohol Intake Level (Low to High): 0

Sleep Duration (Hours): 10

Cholesterol Level (mmol/L): 5

Health Status: At Risk



Assess Your Health Condition

Body Mass Index: 100

Mental Health Concern (0=No, 1=Yes): 1

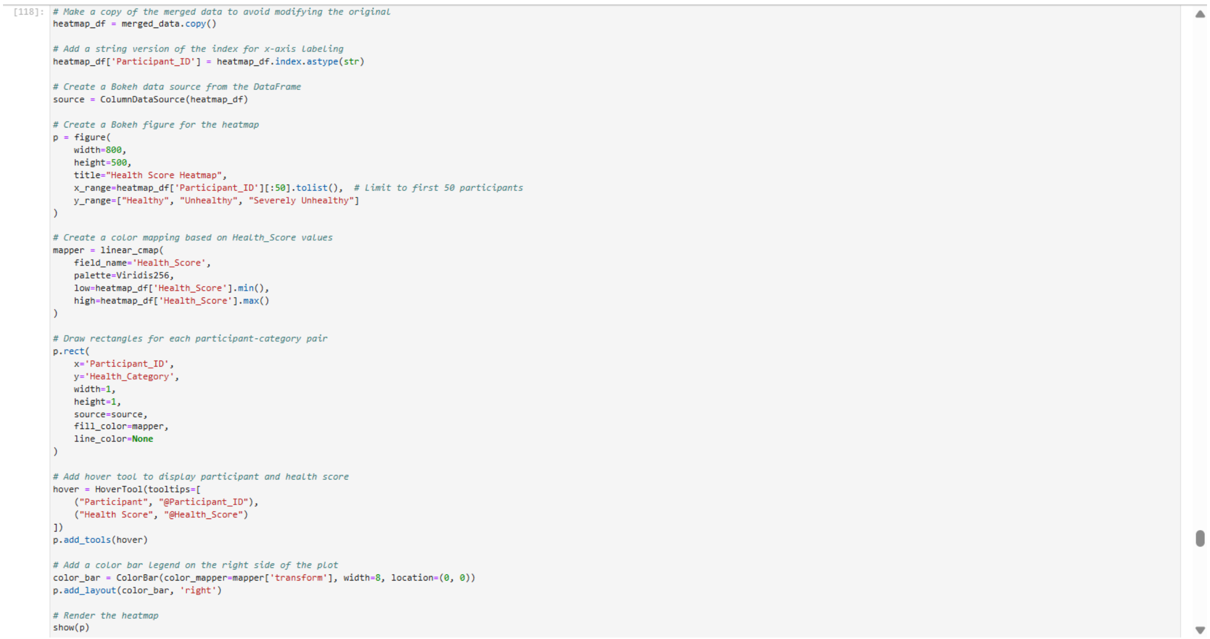
Alcohol Intake Level (Low to High): 3

Sleep Duration (Hours): 2

Cholesterol Level (mmol/L): 6.80

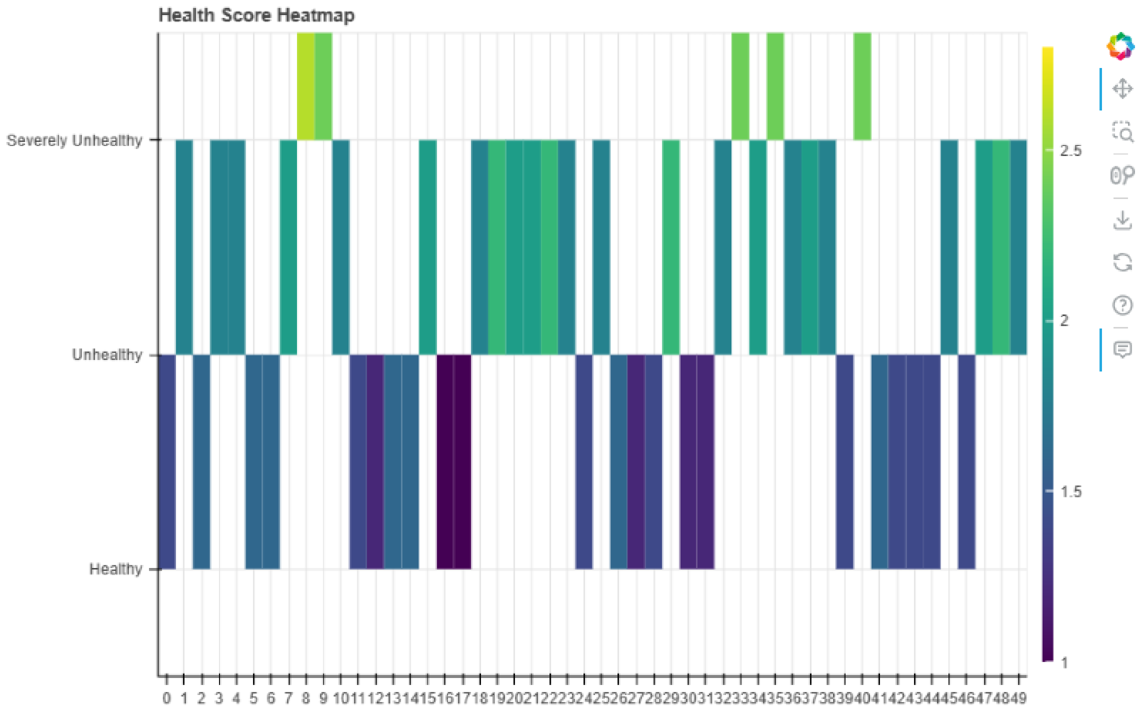
Health Status: Critically Unhealthy

Visualization 5:



```
[118]: # Make a copy of the merged data to avoid modifying the original
heatmap_df = merged_data.copy()

# Add a string version of the index for x-axis labeling
heatmap_df['Participant_ID'] = heatmap_df.index.astype(str)

# Create a Bokeh data source from the DataFrame
source = ColumnDataSource(heatmap_df)

# Create a Bokeh figure for the heatmap
p = figure(
    width=800,
    height=500,
    title="Health Score Heatmap",
    x_range=heatmap_df['Participant_ID'][:50].tolist(),  # Limit to first 50 participants
    y_range=["Healthy", "Unhealthy", "Severely Unhealthy"]
)

# Create a color mapping based on Health_Score values
mapper = linear_cmap(
    field_name='Health_Score',
    palette=Viridis256,
    low=heatmap_df['Health_Score'].min(),
    high=heatmap_df['Health_Score'].max()
)

# Draw rectangles for each participant-category pair
p.rect(
    x='Participant_ID',
    y='Health_Category',
    width=1,
    height=1,
    source=source,
    fill_color=mapper,
    line_color=None
)

# Add hover tool to display participant and health score
hover = HoverTool(tooltips=[
    ("Participant", "@Participant_ID"),
    ("Health Score", "@Health_Score")
])
p.add_tools(hover)

# Add a color bar legend on the right side of the plot
color_bar = ColorBar(color_mapper=mapper['transform'], width=8, location=(0, 0))
p.add_layout(color_bar, 'right')

# Render the heatmap
show(p)
```

Output:


Health Score Heatmap
Severely Unhealthy
Unhealthy
Healthy