Business Understanding

Develop a predictive model to estimate calories burned during workouts based on member attributes and session details.

Key Stakeholders:

- Gym members (personalized fitness tracking)
- Trainers/coaches (optimizing workout plans)
- Healthcare providers (fitness recommendations)

Success Criteria:

- Model achieves $R^2 > 0.85$ (high explanatory power).
- MAE < 50 calories (practically useful precision).

```
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio
import seaborn as sns
pio.renderers.default = 'notebook'
```

Data Understanding

```
file_path = 'data/exercise_tracking.csv'
df = pd.read_csv(file_path)
numeric_cols = ['Age', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM',
                'Resting_BPM', 'Session_Duration (hours)', 'Calories_Burned']
```

Initial Checks

```
## Initial checks
print(df.info())
df.head()
print(df.info())  # Check missing values & data types
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 973 entries, 0 to 972
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Age                            973 non-null    int64
 1   Gender                         973 non-null    object
 2   Weight (kg)                    973 non-null    float64
 3   Height (m)                     973 non-null    float64
 4   Max_BPM                        973 non-null    int64
 5   Avg_BPM                        973 non-null    int64
 6   Resting_BPM                    973 non-null    int64
 7   Session_Duration (hours)       973 non-null    float64
 8   Calories_Burned                973 non-null    float64
 9   Workout_Type                   973 non-null    object
 10  Fat_Percentage                 973 non-null    float64
 11  Water_Intake (liters)          973 non-null    float64
 12  Workout_Frequency (days/week)  973 non-null    int64
 13  Experience_Level               973 non-null    int64
 14  BMI                            973 non-null    float64
dtypes: float64(7), int64(6), object(2)
memory usage: 114.2+ KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 973 entries, 0 to 972
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Age                            973 non-null    int64
```

```
 1   Gender                        973 non-null    object
 2   Weight (kg)                   973 non-null    float64
 3   Height (m)                    973 non-null    float64
 4   Max_BPM                       973 non-null    int64
 5   Avg_BPM                       973 non-null    int64
 6   Resting_BPM                   973 non-null    int64
 7   Session_Duration (hours)      973 non-null    float64
 8   Calories_Burned               973 non-null    float64
 9   Workout_Type                  973 non-null    object
 10  Fat_Percentage                973 non-null    float64
 11  Water_Intake (liters)         973 non-null    float64
 12  Workout_Frequency (days/week) 973 non-null    int64
 13  Experience_Level              973 non-null    int64
 14  BMI                           973 non-null    float64
dtypes: float64(7), int64(6), object(2)
memory usage: 114.2+ KB
None
```

```
unique_counts = df.nunique()
print(unique_counts)
```

```
Age                             42
Gender                           2
Weight (kg)                    532
Height (m)                      51
Max_BPM                         40
Avg_BPM                         50
Resting_BPM                     25
Session_Duration (hours)       147
Calories_Burned                621
Workout_Type                     4
Fat_Percentage                 239
Water_Intake (liters)           23
Workout_Frequency (days/week)    4
Experience_Level                 3
BMI                            771
dtype: int64
```

```
df.describe()
```

	Age	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Fat_Percentage	Wat
count	973.000000	973.000000	973.00000	973.000000	973.000000	973.000000	973.000000	973.000000	973.000000	
mean	38.683453	73.854676	1.72258	179.883864	143.766701	62.223022	1.256423	905.422405	24.976773	
std	12.180928	21.207500	0.12772	11.525686	14.345101	7.327060	0.343033	272.641516	6.259419	
min	18.000000	40.000000	1.50000	160.000000	120.000000	50.000000	0.500000	303.000000	10.000000	
25%	28.000000	58.100000	1.62000	170.000000	131.000000	56.000000	1.040000	720.000000	21.300000	
50%	40.000000	70.000000	1.71000	180.000000	143.000000	62.000000	1.260000	893.000000	26.200000	
75%	49.000000	86.000000	1.80000	190.000000	156.000000	68.000000	1.460000	1076.000000	29.300000	
max	59.000000	129.900000	2.00000	199.000000	169.000000	74.000000	2.000000	1783.000000	35.000000	

Check for missing data

```
# Check for missing data
print(df.isnull().sum())
```

```
Age                             0
Gender                          0
Weight (kg)                     0
Height (m)                      0
Max_BPM                         0
Avg_BPM                         0
Resting_BPM                     0
Session_Duration (hours)        0
Calories_Burned                 0
Workout_Type                    0
Fat_Percentage                  0
Water_Intake (liters)           0
Workout_Frequency (days/week)   0
Experience_Level                0
BMI                             0
dtype: int64
```

```
# Drop rows with missing target (Calories_Burned)
print(df.dropna(subset=['Calories_Burned'], inplace=True))
#No missing data found
```

```
None
```

Feature Engineering

```
df['BMI'] = df['Weight (kg)'] / (df['Height (m)']**2)
df['BPM_Difference'] = df['Max_BPM'] - df['Avg_BPM']
df['BPM_Increase_from_Rest'] = df['Avg_BPM'] - df['Resting_BPM']

if 'Exercise_Type' in df.columns:
    df = pd.get_dummies(df, columns=['Exercise_Type'], drop_first=True)
else:
    print("Warning: 'Exercise_Type' column not found. Skipping get_dummies for this column.")


# Example of creating a 'BMI_Category' feature
def bmi_category(bmi):
  if bmi < 18.5:
    return 'Underweight'
  elif 18.5 <= bmi < 25:
    return 'Normal weight'
  elif 25 <= bmi < 30:
    return 'Overweight'
  else:
    return 'Obese'

df['BMI_Category'] = df['BMI'].apply(bmi_category)

# Convert 'Gender' to numerical
# {0: 'Female', 1: 'Male'}

if 'Gender' in df.columns:
    df['Gender'] = df['Gender'].str.strip().str.capitalize()
    df['Gender'] = df['Gender'].astype('category').cat.codes
    print(dict(enumerate(df['Gender'].astype('category').cat.categories))) # {0: 'Female', 1: 'Male'}
else:
    print("Warning: 'Gender' column not found. Skipping category conversion.")


print(df.head())
print(df.info())
```

```
Warning: 'Exercise_Type' column not found. Skipping get_dummies for this column.
{0: 0, 1: 1}
   Age  Gender  Weight (kg)  Height (m)  Max_BPM  Avg_BPM  Resting_BPM  \
0   56       1         88.3        1.71      180      157           60
1   46       0         74.9        1.53      179      151           66
2   32       0         68.1        1.66      167      122           54
3   25       1         53.2        1.70      190      164           56
4   38       1         46.1        1.79      188      158           68

   Session_Duration (hours)  Calories_Burned Workout_Type  Fat_Percentage  \
0                      1.69           1313.0         Yoga            12.6
1                      1.30            883.0         HIIT            33.9
2                      1.11            677.0       Cardio            33.4
3                      0.59            532.0     Strength            28.8
4                      0.64            556.0     Strength            29.2

   Water_Intake (liters)  Workout_Frequency (days/week)  Experience_Level  \
0                    3.5                              4                 3
1                    2.1                              4                 2
2                    2.3                              4                 2
3                    2.1                              3                 1
4                    2.8                              3                 1

         BMI  BPM_Difference  BPM_Increase_from_Rest   BMI_Category
0  30.197326              23                      97          Obese
1  31.996241              28                      85          Obese
2  24.713311              45                      68  Normal weight
3  18.408304              26                     108    Underweight
4  14.387816              30                      90    Underweight
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 973 entries, 0 to 972
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Age                             973 non-null    int64
 1   Gender                          973 non-null    int8
 2   Weight (kg)                     973 non-null    float64
 3   Height (m)                      973 non-null    float64
 4   Max_BPM                         973 non-null    int64
 5   Avg_BPM                         973 non-null    int64
 6   Resting_BPM                     973 non-null    int64
 7   Session_Duration (hours)        973 non-null    float64
 8   Calories_Burned                 973 non-null    float64
 9   Workout_Type                    973 non-null    object
 10  Fat_Percentage                  973 non-null    float64
 11  Water_Intake (liters)           973 non-null    float64
 12  Workout_Frequency (days/week)   973 non-null    int64
 13  Experience_Level                973 non-null    int64
 14  BMI                             973 non-null    float64
 15  BPM_Difference                  973 non-null    int64
 16  BPM_Increase_from_Rest          973 non-null    int64
 17  BMI_Category                    973 non-null    object
dtypes: float64(7), int64(8), int8(1), object(2)
memory usage: 130.3+ KB
None
```

Encode Categorical Variables

- Our implmentation already includes one-hot encoding for the 'Exercise_Type' column using `pd.get_dummies`. which is necessary for learning models that require numerical input, as categorical features like 'Exercise_Type' cannot be directly used in their string format.
- The code also converts 'Gender' to numerical categories {0: 'Female', 1: 'Male'}
- No further one-hot encoding is immediately needed based on the provided code and the task description. All potentially relevant categorical columns ('Exercise_Type' and 'Gender') are already being handled.
- The current state of the dataframe, as shown by `df.info()`, confirms that Exercise_Type' dummy variables have been created (e.g., 'Exercise_Type_Cardio', 'Exercise_Type_Strength'), and 'Gender' has been converted to a numerical type (int8). 'BMI_Category' is a new categorical column created through feature engineering.

```
df.head()
```

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage
0	56	1	88.3	1.71	180	157	60	1.69	1313.0	Yoga	12.6
1	46	0	74.9	1.53	179	151	66	1.30	883.0	HIIT	33.9
2	32	0	68.1	1.66	167	122	54	1.11	677.0	Cardio	33.4
3	25	1	53.2	1.70	190	164	56	0.59	532.0	Strength	28.8
4	38	1	46.1	1.79	188	158	68	0.64	556.0	Strength	29.2

Exploratory Data Analysis (EDA) and Visualizations

Distribution of numerical data

We are checking for Skewness and Kurtosis of our numerical data.

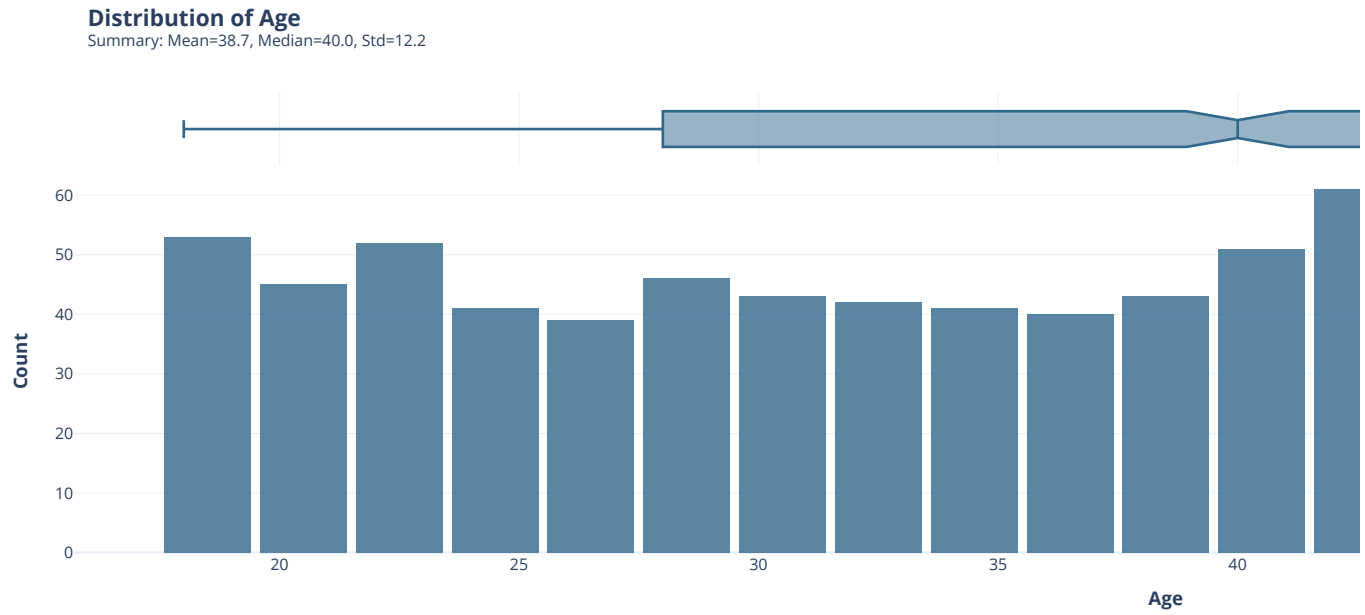
Rule of thumb for skewness:

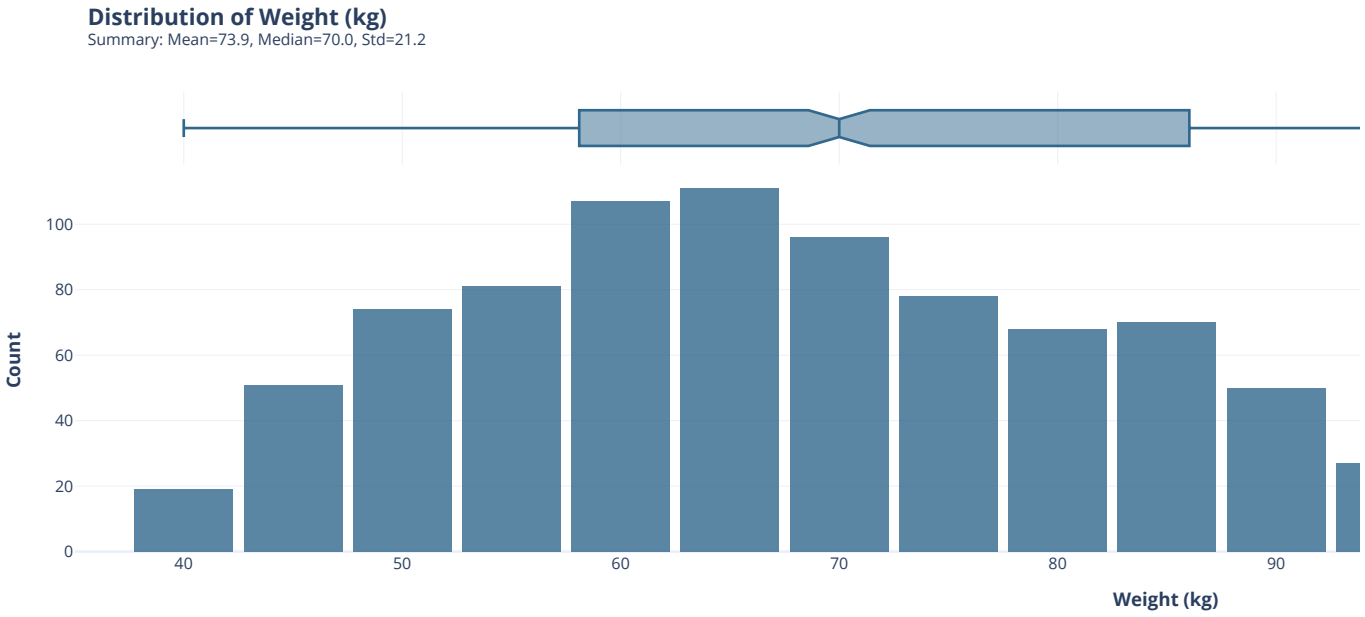
- |Skewness| < 0.5 → Fairly symmetrical
- 0.5 ≤ |Skewness| < 1 → Moderately skewed
- |Skewness| ≥ 1 → Highly skewed

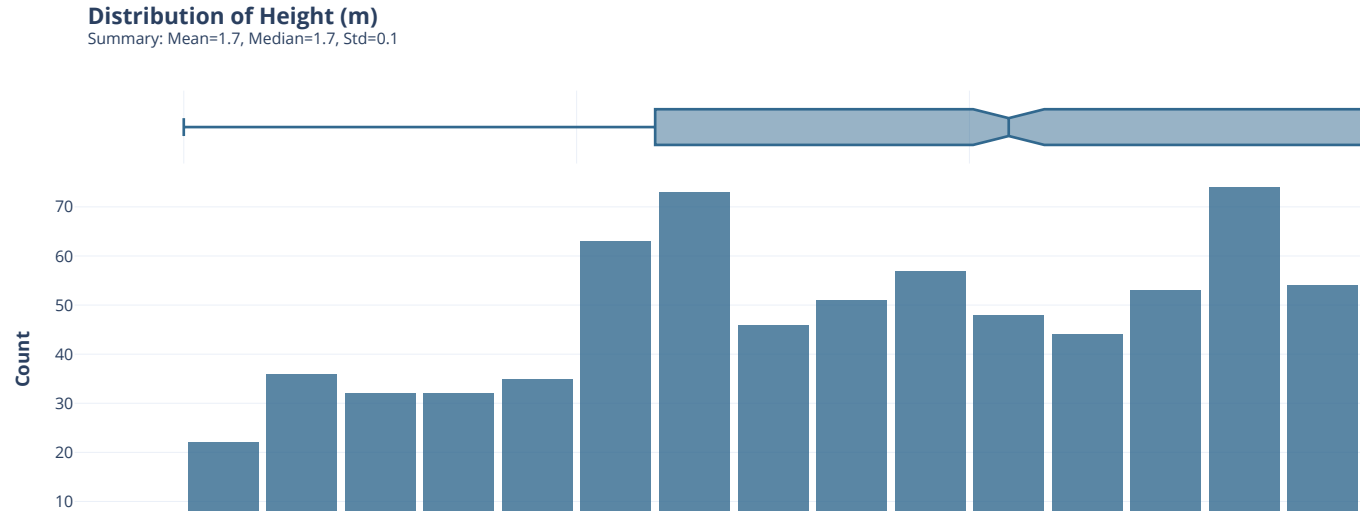
Rule of thumb for kurtosis:

- |Kurtosis| < 1 → Close to normal
- 1 ≤ |Kurtosis| < 2 → Moderately different
- |Kurtosis| ≥ 2 → Significantly different

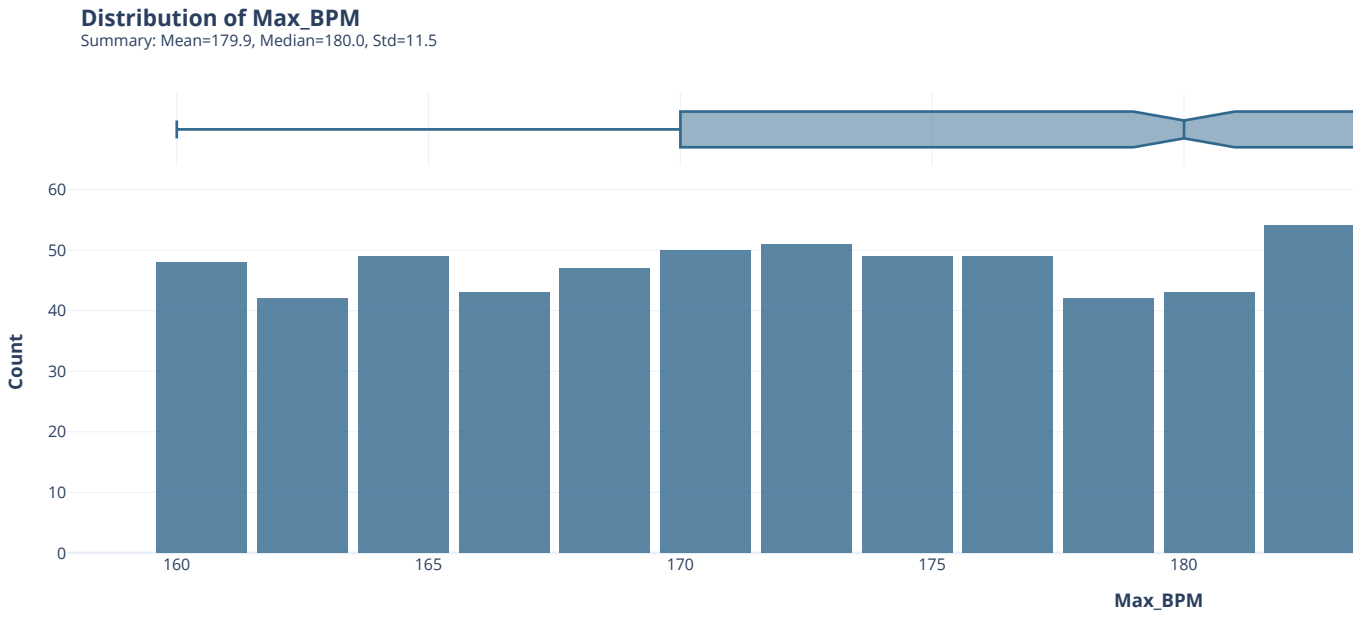
```
    # Create a subplot grid
    fig = make_subplots(rows=4, cols=2,
                        subplot_titles=numeric_cols,
                        horizontal_spacing=0.1,
                        vertical_spacing=0.1)

    for col in numeric_cols:
        fig = px.histogram(
            df,
            x=col,
            marginal='box',
            title=f'<b>Distribution of {col}</b><br><sup>Summary: Mean={df[col].mean():.1f}, Median={df[col].median():.1f}, Std={df[
            color_discrete_sequence=[px.colors.sequential.Viridis[3]],
            template='plotly_white',
            nbins=30,
            opacity=0.8,
            hover_data=df.columns
        )

        # Add summary statistics annotations
        fig.add_annotation(
            x=0.95, y=0.85,
            xref='paper', yref='paper',
            text=f"Skewness: {df[col].skew():.2f}<br>Kurtosis: {df[col].kurtosis():.2f}",
            showarrow=False,
            bgcolor="white",
            bordercolor="black",
            borderwidth=1
        )

        # Add interactive elements
        fig.update_layout(
            bargap=0.1,
            hovermode='x unified',
            xaxis_title=f"<b>{col}</b>",
            yaxis_title="<b>Count</b>"
        )

        fig.show()
```

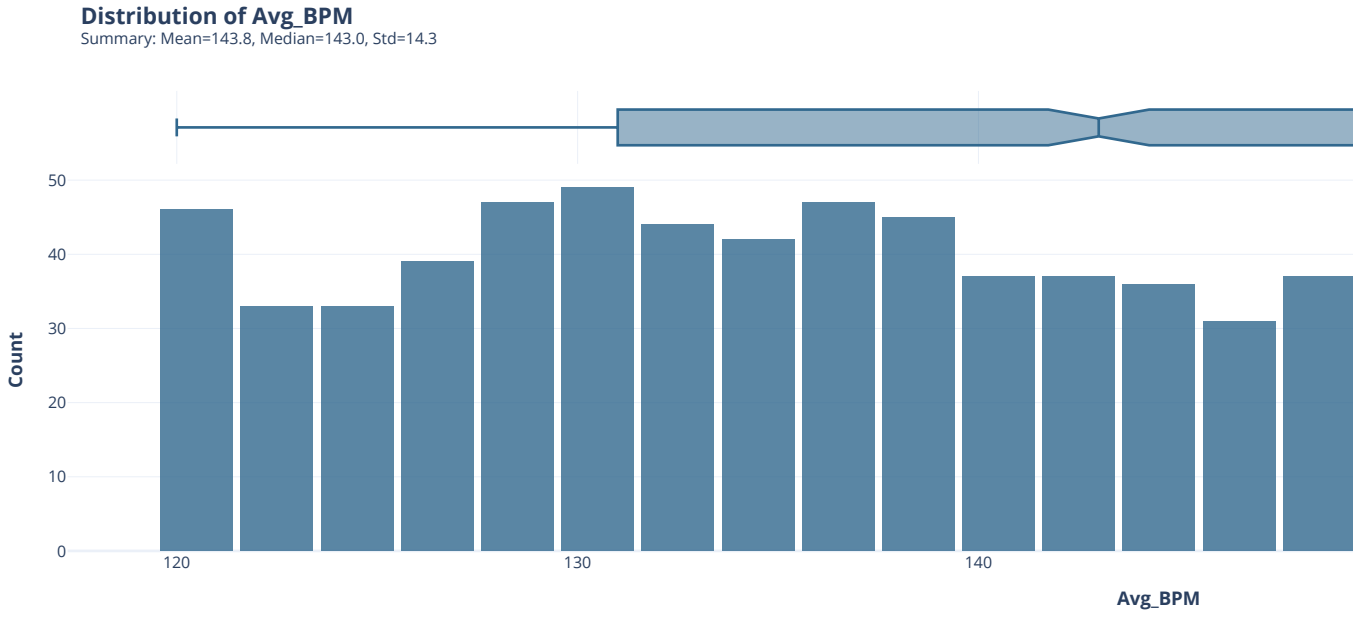

Distribution of Age
Summary: Mean=38.7, Median=40.0, Std=12.2
Count
Age
0
10
20
30
40
50
60
20
25
30
35
40

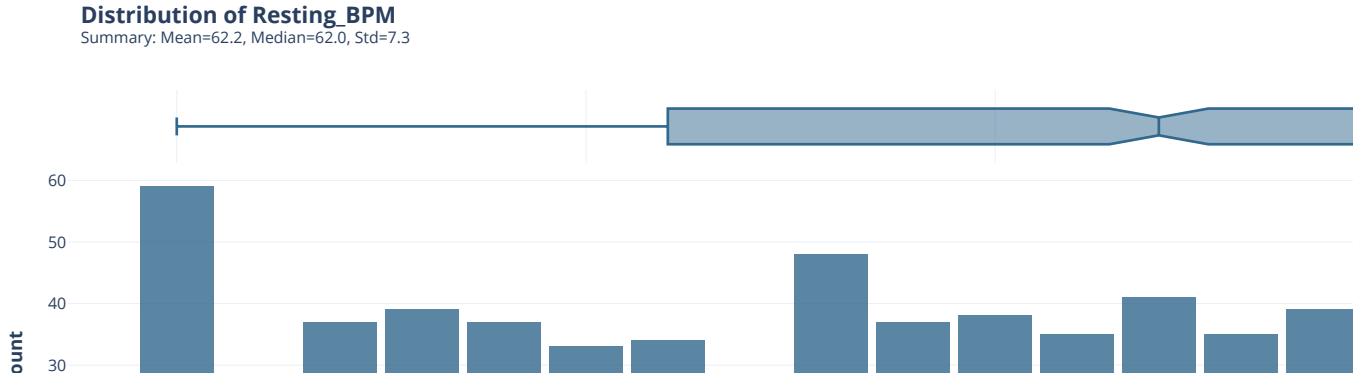

Distribution of Weight (kg)
Summary: Mean=73.9, Median=70.0, Std=21.2
Count
Weight (kg)
0
20
40
60
80
100
40
50
60
70
80
90

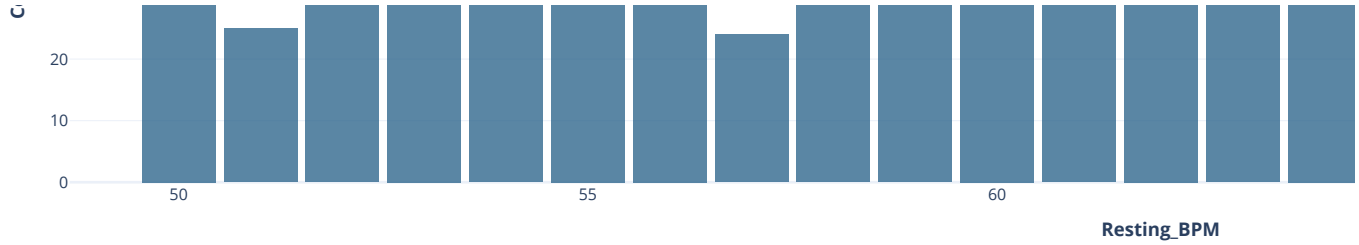

Distribution of Height (m)
Summary: Mean=1.7, Median=1.7, Std=0.1
Count
10
20
30
40
50
60
70


0
1.5
1.6
1.7
Height (m)


Distribution of Max_BPM
Summary: Mean=179.9, Median=180.0, Std=11.5
Count
60
50
40
30
20
10
0
160
165
170
175
180
Max_BPM

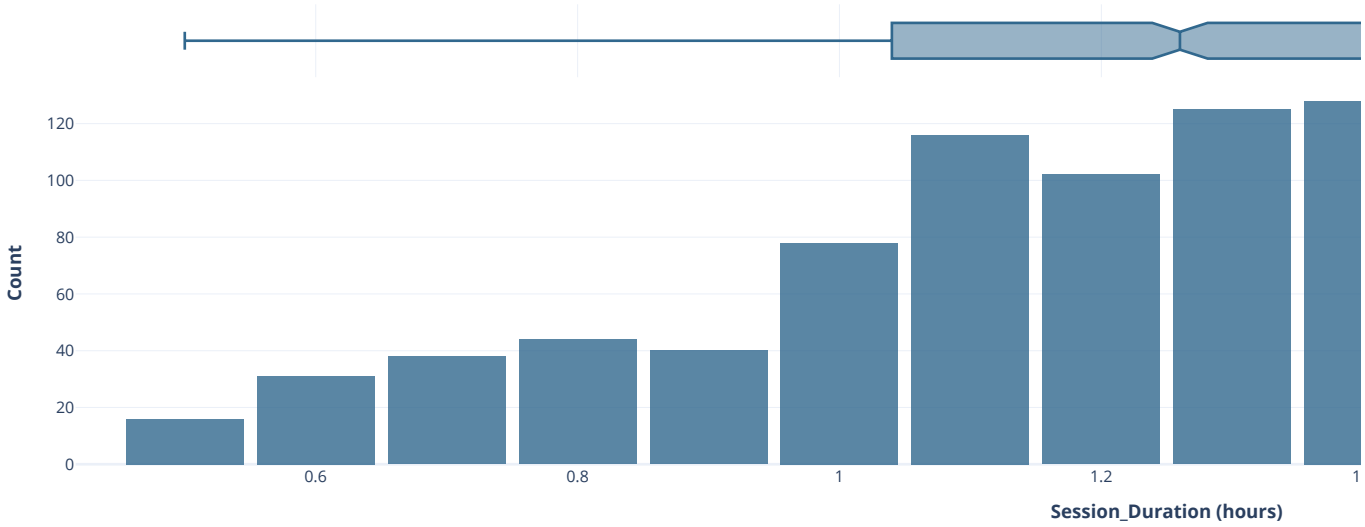

Distribution of Avg_BPM
Summary: Mean=143.8, Median=143.0, Std=14.3
Count
50
40
30
20
10
0
120
130
140
Avg_BPM


Distribution of Resting_BPM
Summary: Mean=62.2, Median=62.0, Std=7.3
ount
60
50
40
30



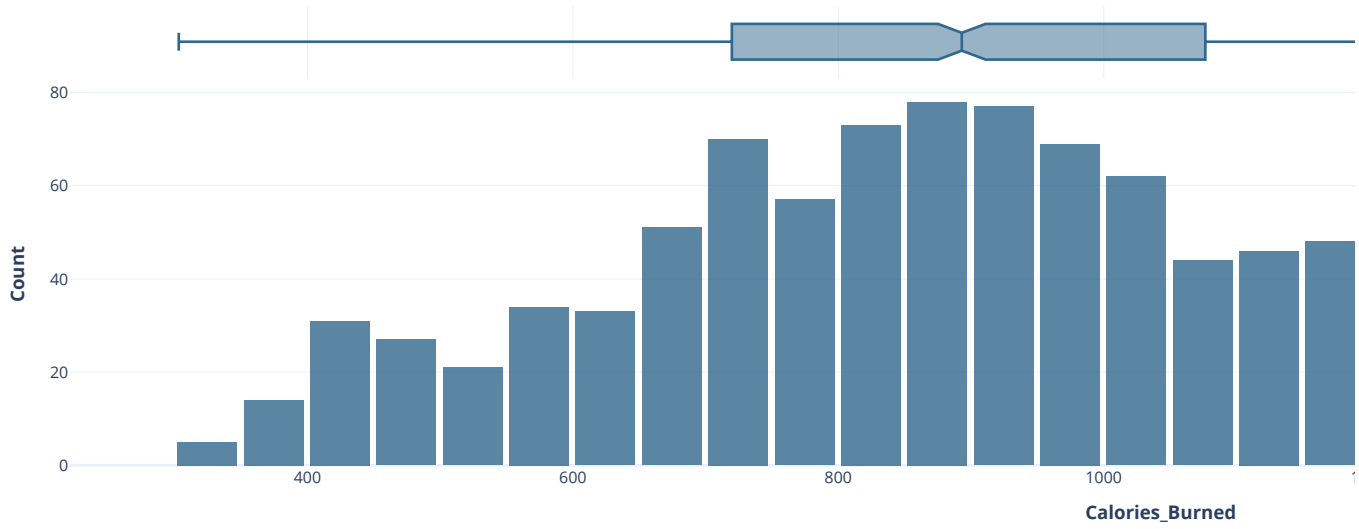
Distribution of Session_Duration (hours)

Summary: Mean=1.3, Median=1.3, Std=0.3



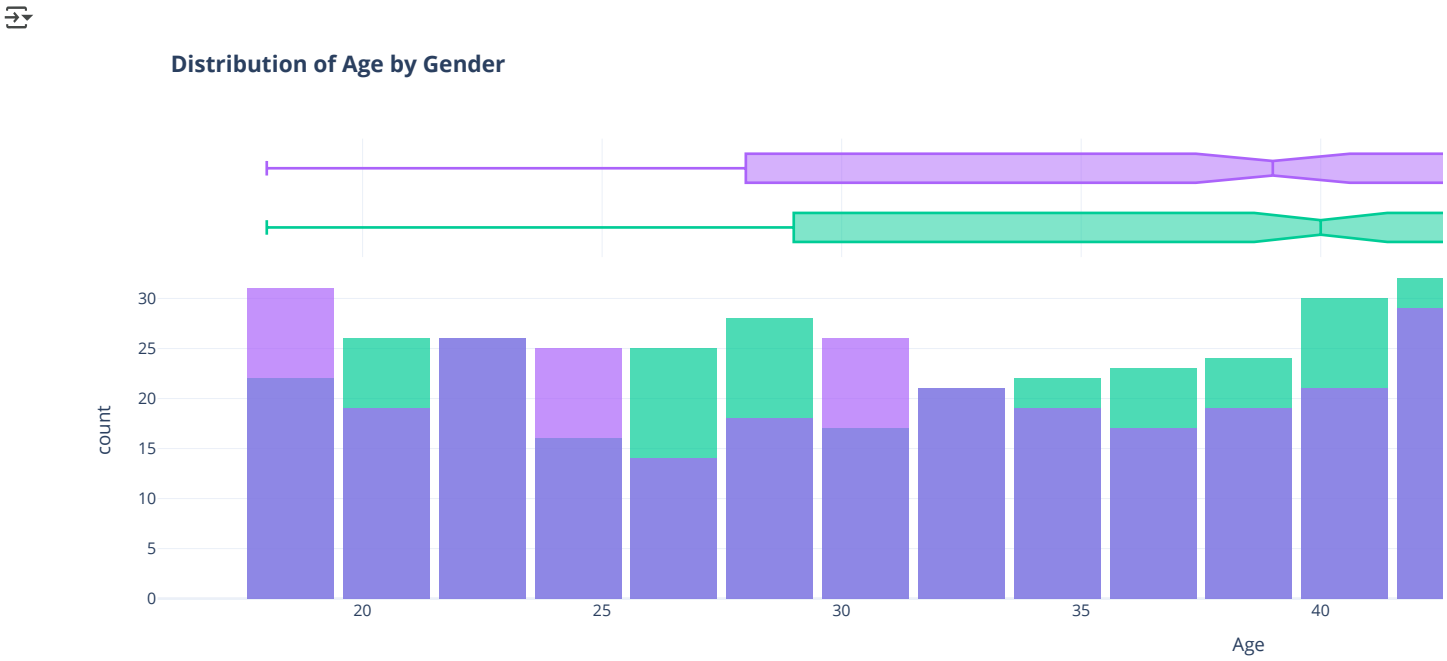
Distribution of Calories_Burned

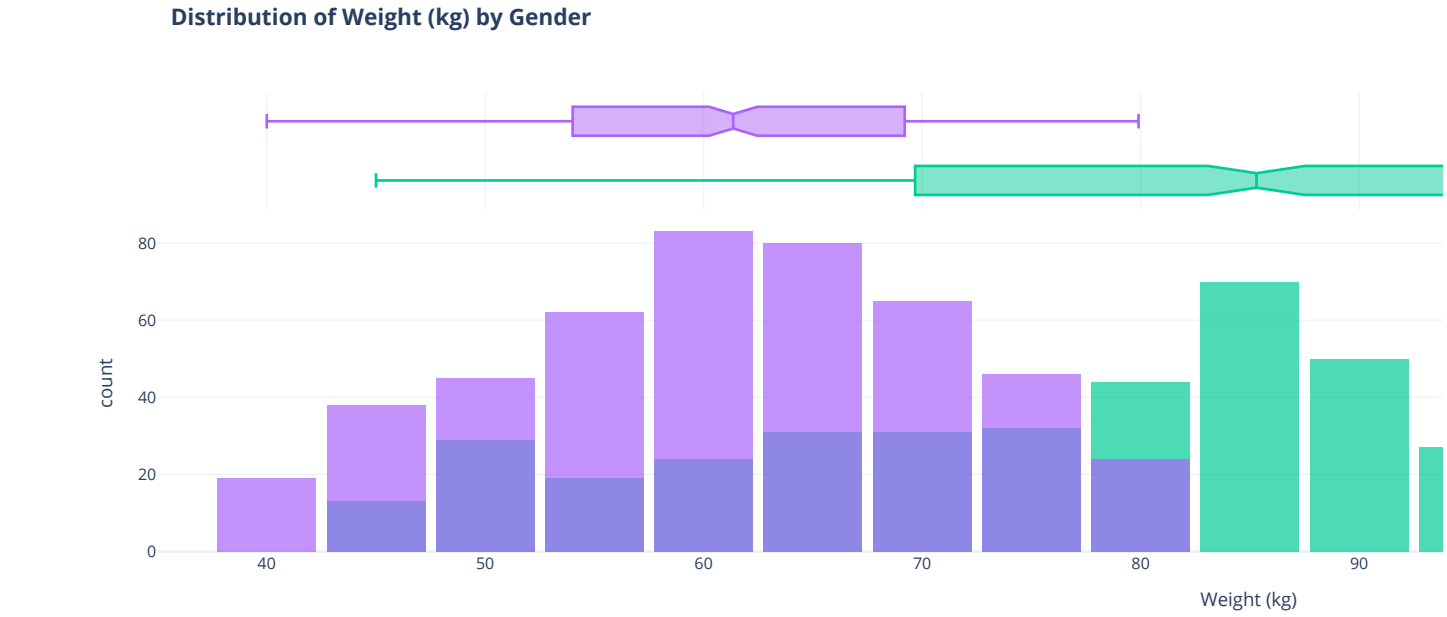
Summary: Mean=905.4, Median=893.0, Std=272.6

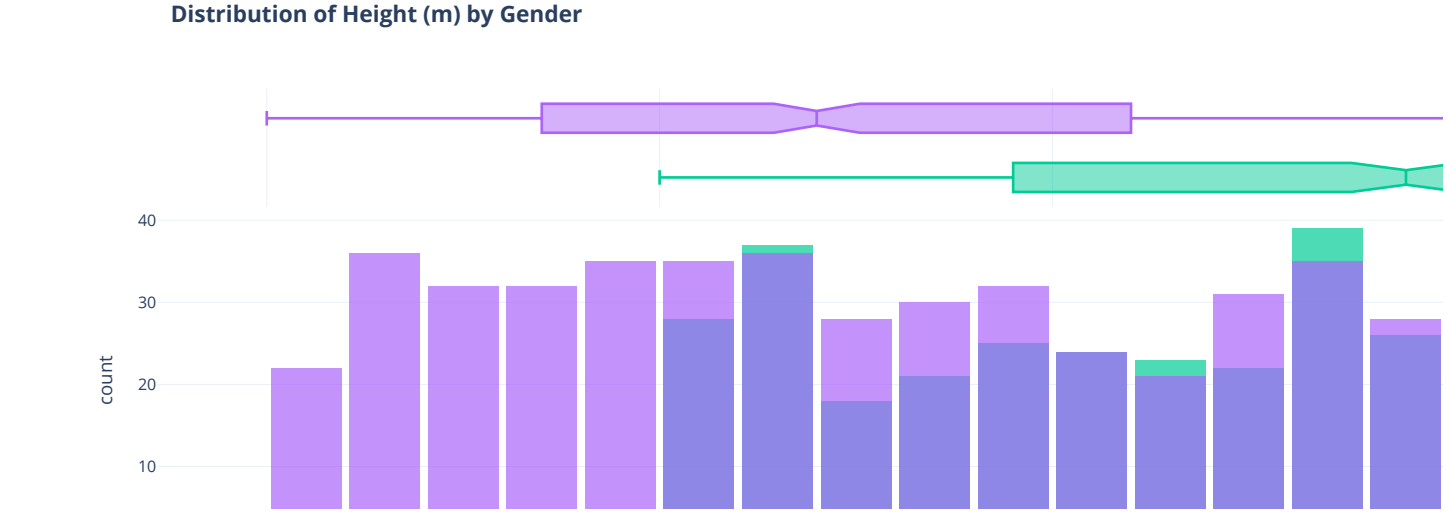


We already see some outliers in our data under **Calories_Burned** and **Weight** categories. But lets see if we can further distinguish based on **Gender**

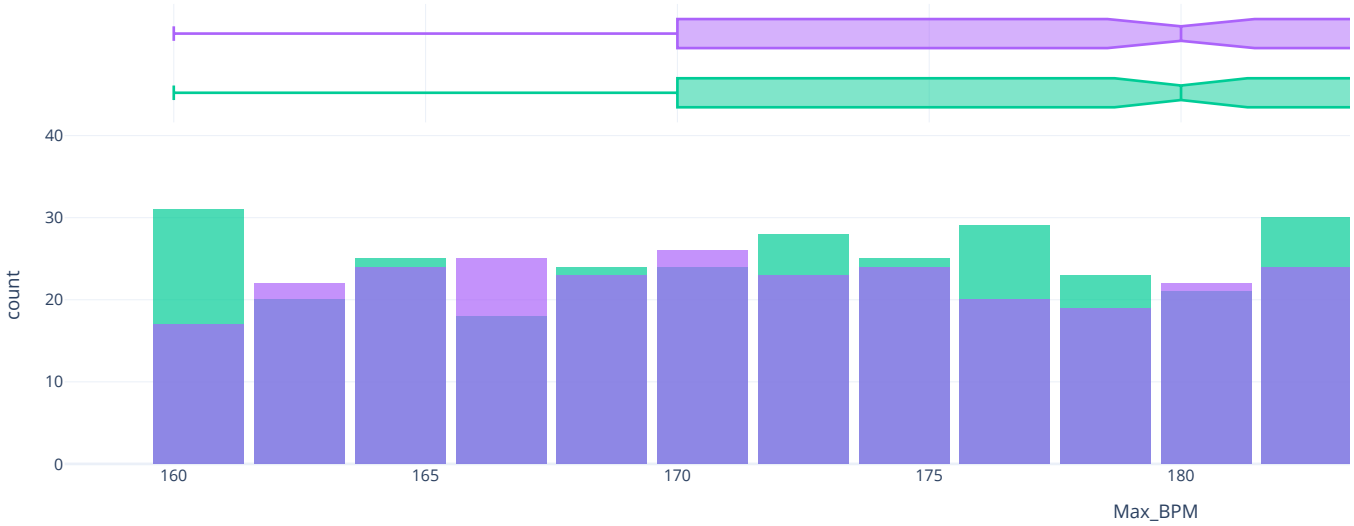
```
for col in numeric_cols:
    fig = px.histogram(
        df,
        x=col,
        color='Gender',
        marginal='box',
        title=f'<b>Distribution of {col} by Gender</b>',
        color_discrete_map={'Male': '#1F77B4', 'Female': '#FF7F0E'},
        template='plotly_white',
        nbins=30,
        opacity=0.7,
        barmode='overlay',
        hover_data=df.columns
    )

    fig.add_annotation(
        x=0.95, y=0.85,
        xref='paper', yref='paper',
        text=f"",
        showarrow=False,
        bgcolor="white",
        bordercolor="black",
        borderwidth=1
    )

    fig.update_layout(
        bargap=0.1,
        hovermode='x unified',
        legend=dict(
            orientation="h",
            yanchor="bottom",
            y=1.02,
            xanchor="right",
            x=1
        )
    )

    fig.show()
```


Distribution of Age by Gender
count
Age
0
5
10
15
20
25
30
20
25
30
35
40

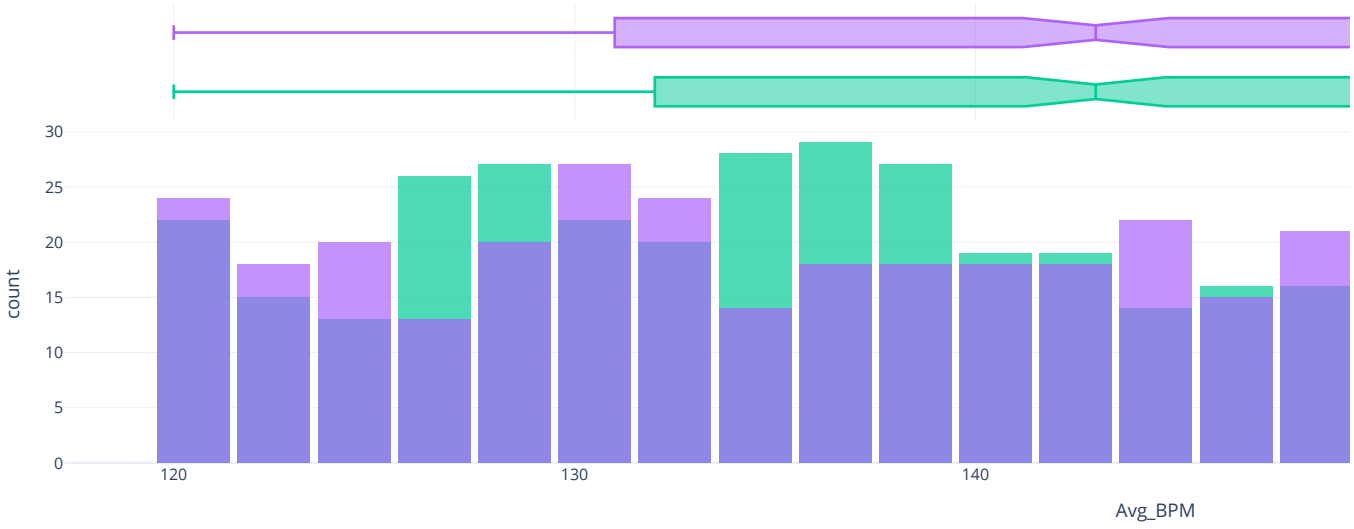

Distribution of Weight (kg) by Gender
count
Weight (kg)
0
20
40
60
80
40
50
60
70
80
90


Distribution of Height (m) by Gender
count
10
20
30
40

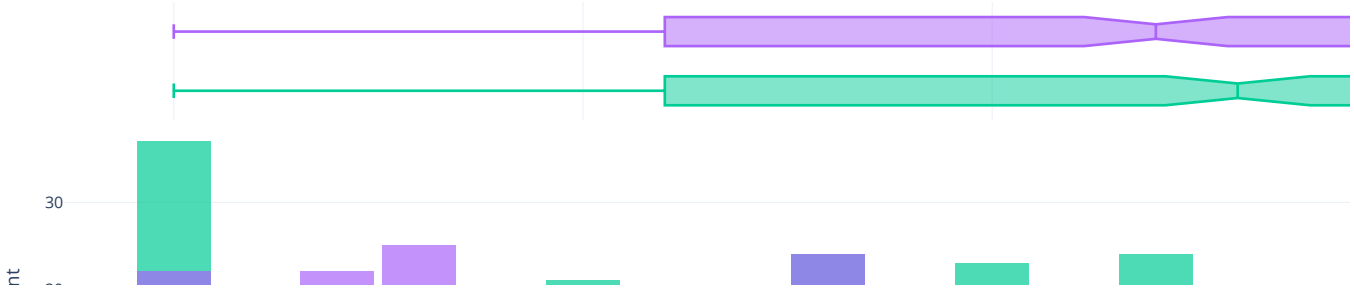
Distribution of Max_BPM by Gender

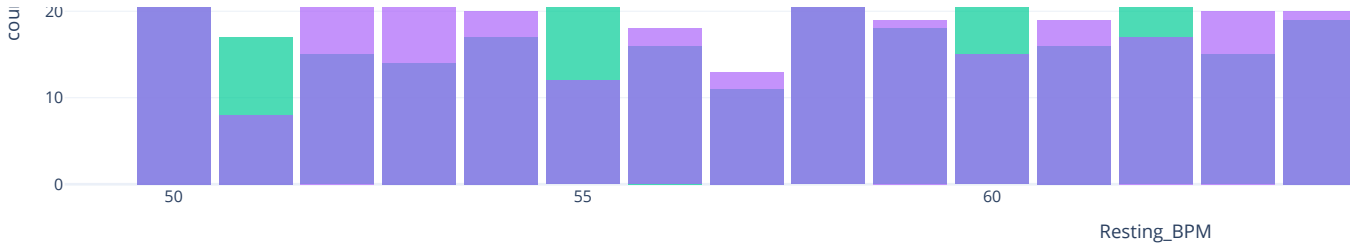


Distribution of Avg_BPM by Gender

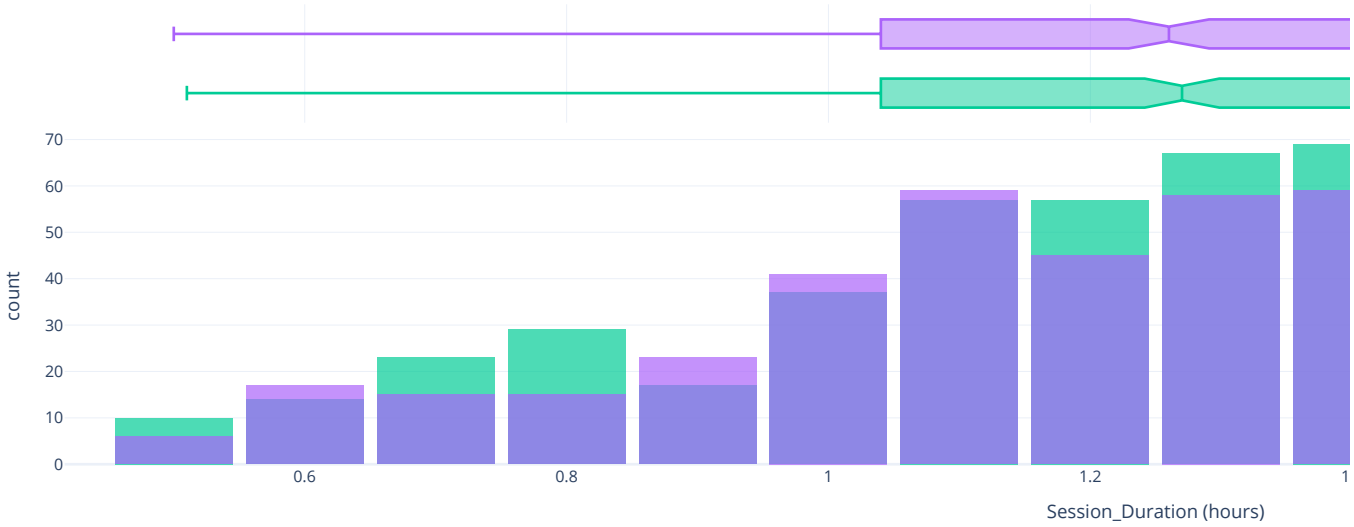


Distribution of Resting_BPM by Gender

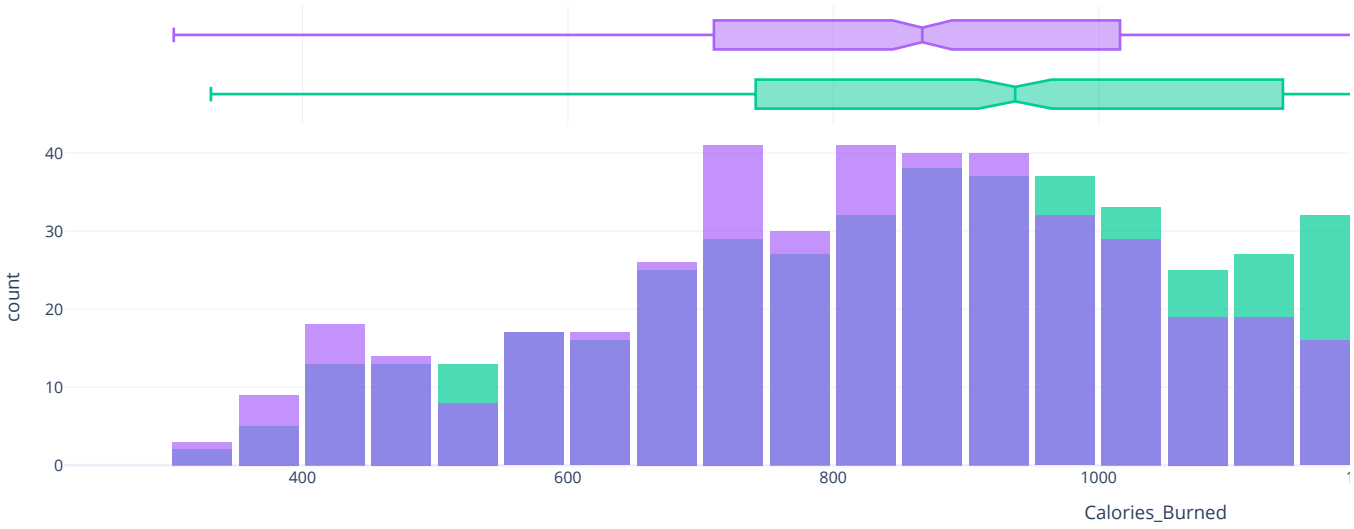




Distribution of Session_Duration (hours) by Gender



Distribution of Calories_Burned by Gender



```
Start coding or generate with AI.
```

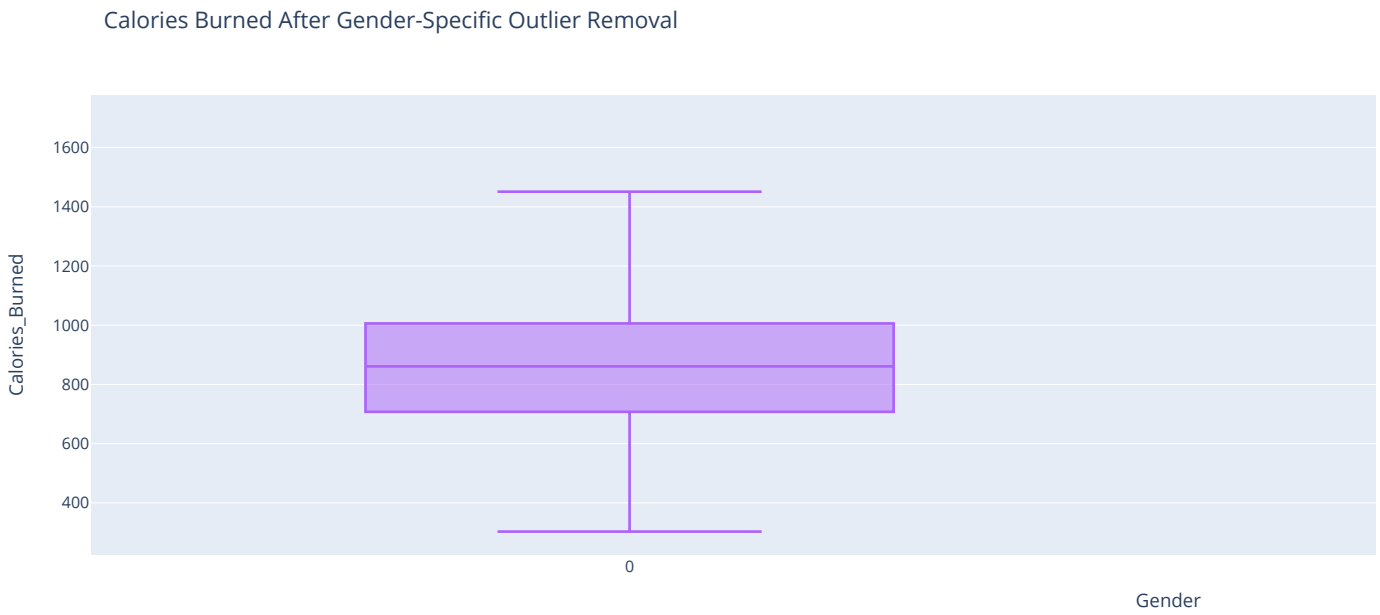
Distribution of numerical data based on Gender

We see that the values in age were not ture representation of the outliers but **Calories_Burned** do have outliers that we can safely remove.

Remove outliers from Calories_Burned

```
# Calculate quartiles and IQR for each gender
gender_stats = df.groupby('Gender')['Calories_Burned'].quantile([0.25, 0.75]).unstack()
gender_stats['IQR'] = gender_stats[0.75] - gender_stats[0.25]
gender_stats['Lower_Bound'] = gender_stats[0.25] - 1.45 * gender_stats['IQR']
gender_stats['Upper_Bound'] = gender_stats[0.75] + 1.45 * gender_stats['IQR']

print("Gender-specific bounds:")
print(gender_stats[['Lower_Bound', 'Upper_Bound']])

mask = pd.Series(True, index=df.index)

for gender in gender_stats.index:
    gender_mask = (df['Gender'] == gender)
    lower = gender_stats.loc[gender, 'Lower_Bound']
    upper = gender_stats.loc[gender, 'Upper_Bound']
    mask &= ~(gender_mask &
              ((df['Calories_Burned'] < lower) |
               (df['Calories_Burned'] > upper)))

# Apply the mask to filter outliers
df = df[mask].copy()

# Plot to verify outliers are removed
fig = px.box(
    df,
    x='Gender',
    y='Calories_Burned',
    color='Gender',
    color_discrete_map={'Male': '#1F77B4', 'Female': '#FF7F0E'},
    title='Calories Burned After Gender-Specific Outlier Removal'
)
fig.show()
```

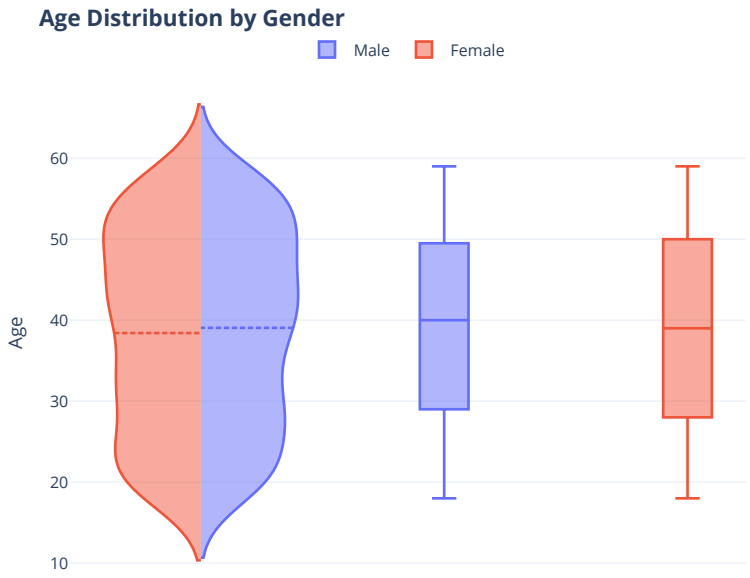
```
Gender-specific bounds:
        Lower_Bound  Upper_Bound
Gender
0          266.9125    1459.3375
1          167.0750    1713.4250
```

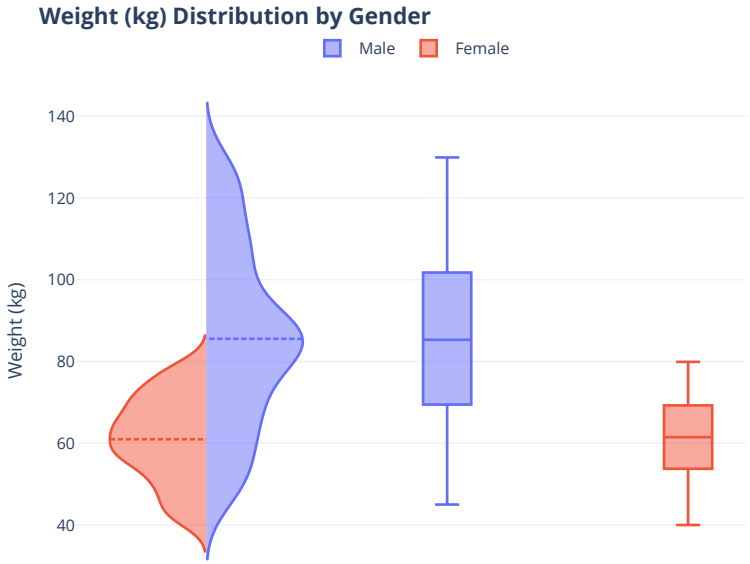


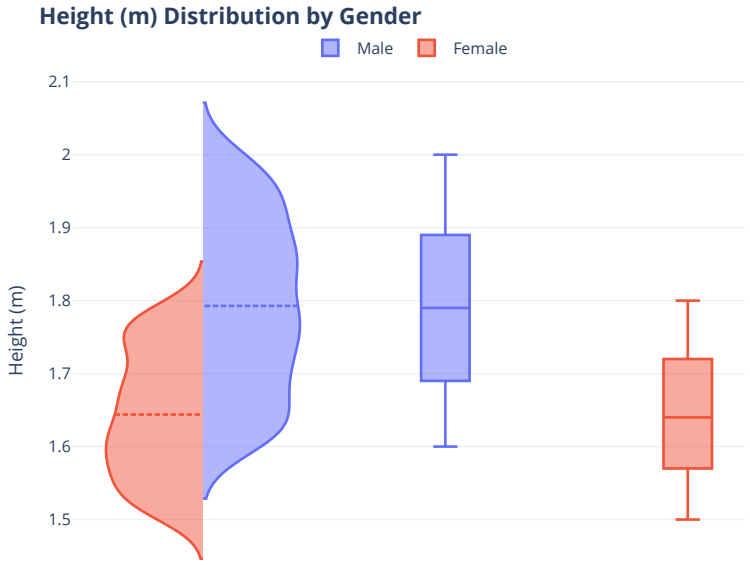
Visualizing data using Violin plots

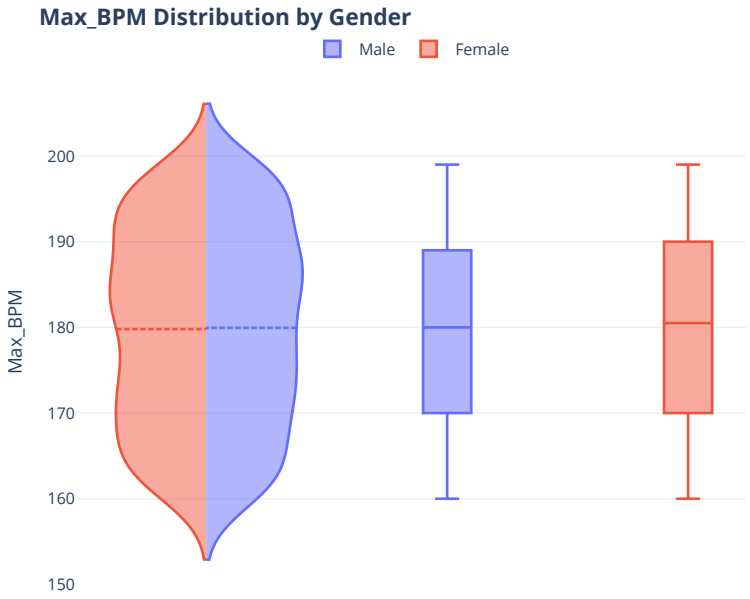
```
for col in numeric_cols:
    # Create figure
    fig = go.Figure()

    # Get data
    male_data = df[df['Gender'] == 1][col].dropna()
    female_data = df[df['Gender'] == 0][col].dropna()

    # print(df['Gender'].unique())
    # print(df['Gender'].dtype)

    # print(male_data)
    # print(female_data)

    # Create split violin
    fig.add_trace(go.Violin(
        x=np.repeat('Distribution', len(male_data)),  # Single category
        y=male_data,
        name='Male',
        side='positive',  # Right side
        line_color='#636EFA',
        width=0.8,
        hoverinfo='y+name',
        scalemode='width',
        meanline_visible=True,
        points=False
    ))

    fig.add_trace(go.Violin(
        x=np.repeat('Distribution', len(female_data)),
        y=female_data,
        name='Female',
        side='negative',  # Left side
        line_color='#EF553B',
        width=0.8,
        hoverinfo='y+name',
        scalemode='width',
        meanline_visible=True,
```

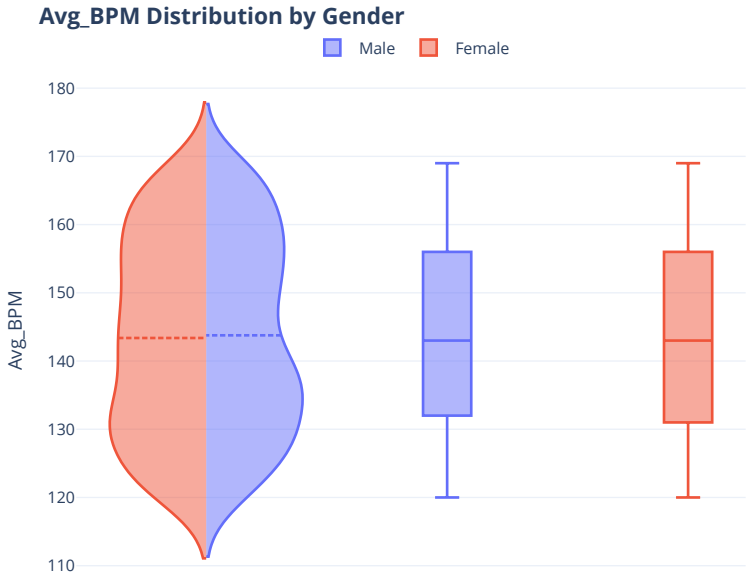
```
            points=False
        ))

        # Add box plots as lines (no subplot reference needed)
        fig.add_trace(go.Box(
            y=male_data,
            name='Male',
            marker_color='#636EFA',
            line_color='#636EFA',
            showlegend=False,
            boxpoints=False,
            width=0.2
        ))

        fig.add_trace(go.Box(
            y=female_data,
            name='Female',
            marker_color='#EF553B',
            line_color='#EF553B',
            showlegend=False,
            boxpoints=False,
            width=0.2
        ))

        # Update layout
        fig.update_layout(
            title=f'<b>{col} Distribution by Gender</b>',
            template='plotly_white',
            violinmode='overlay',
            hovermode='y unified',
            yaxis_title=col,
            xaxis=dict(showticklabels=False),
            showlegend=True,
            legend=dict(
                orientation="h",
                yanchor="bottom",
                y=1.02,
                xanchor="center",
                x=0.5
            ),
            margin=dict(l=40, r=40, t=80, b=40),
            width=600,
            height=500
        )

        fig.show()
```

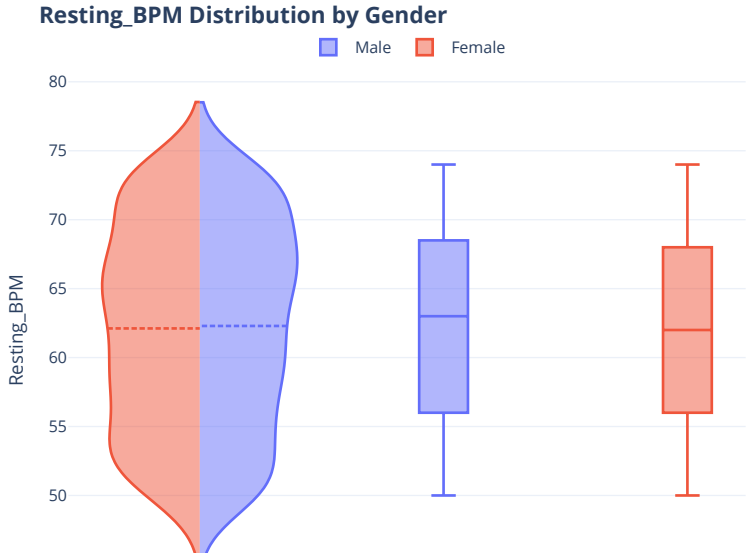

Age Distribution by Gender
Male
Female
Age
60
50
40
30
20
10

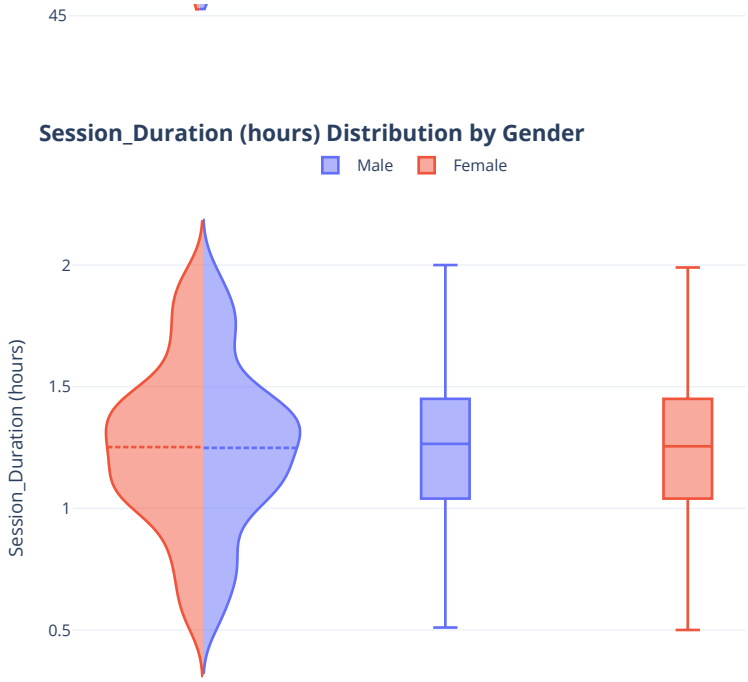

Weight (kg) Distribution by Gender
Male
Female
Weight (kg)
140
120
100
80
60
40

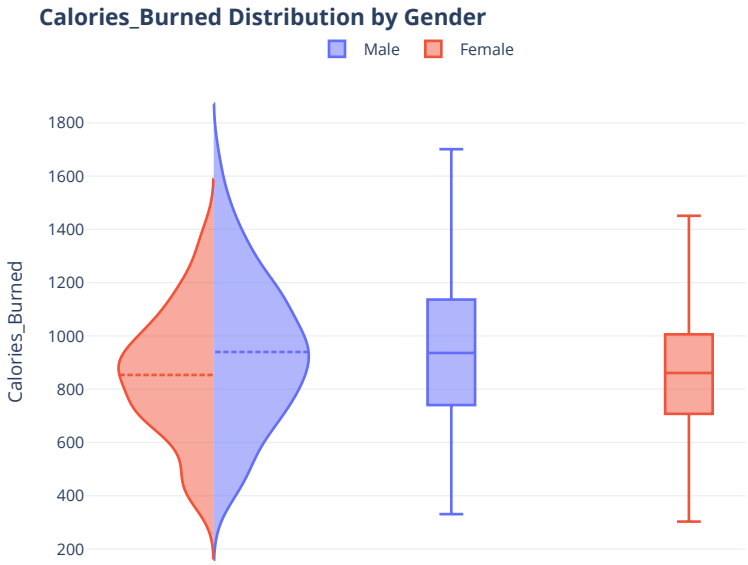

Height (m) Distribution by Gender
Male
Female
Height (m)
2.1
2
1.9
1.8
1.7
1.6
1.5


Max_BPM Distribution by Gender
Male
Female
Max_BPM
200
190
180
170
160
150


Avg_BPM Distribution by Gender
Male
Female
Avg_BPM
180
170
160
150
140
130
120
110

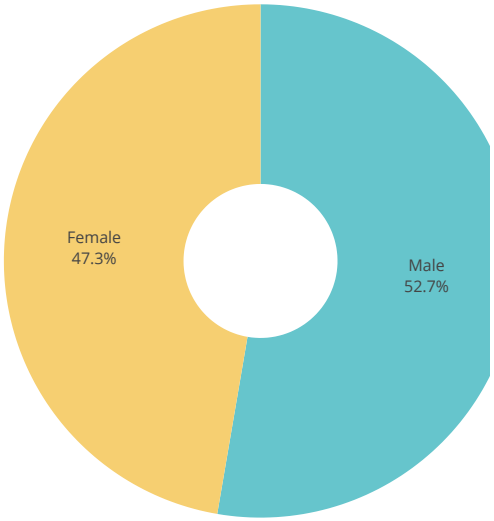

Resting_BPM Distribution by Gender
Male
Female
Resting_BPM
80
75
70
65
60
55
50


45
Session_Duration (hours) Distribution by Gender
Male
Female
Session_Duration (hours)
2
1.5
1
0.5

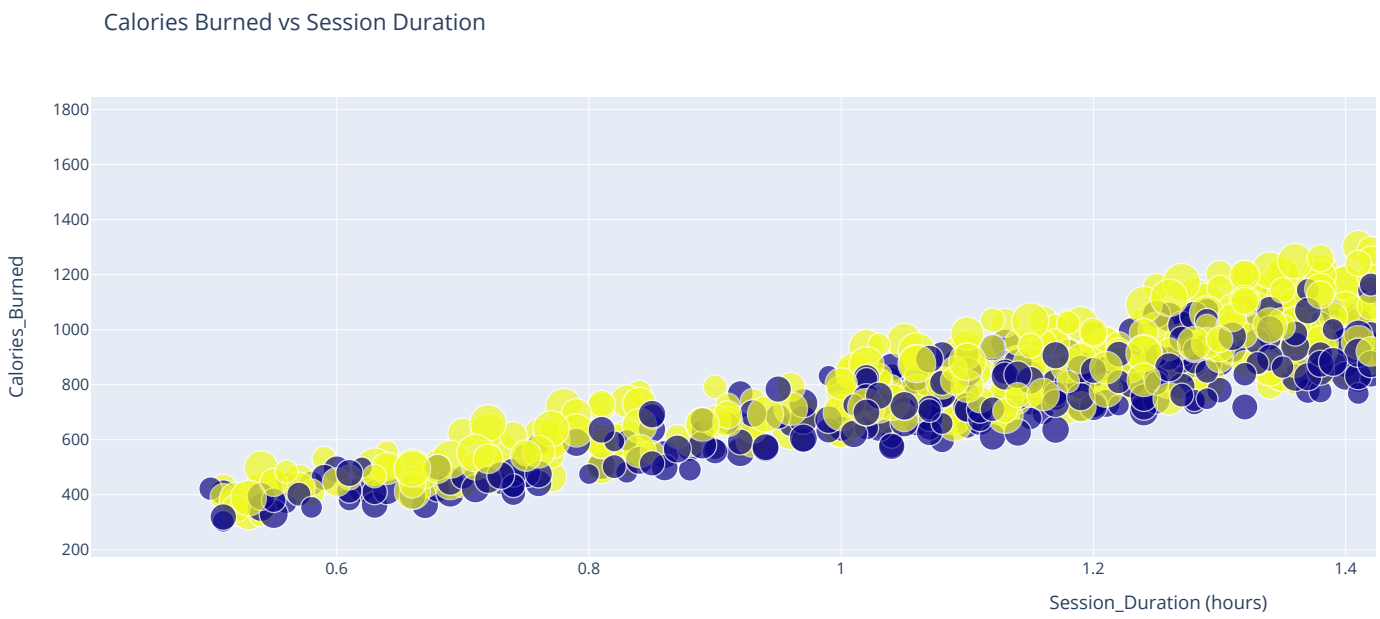

Calories_Burned Distribution by Gender
Male
Female
Calories_Burned
1800
1600
1400
1200
1000
800
600
400
200

Categorical Variables

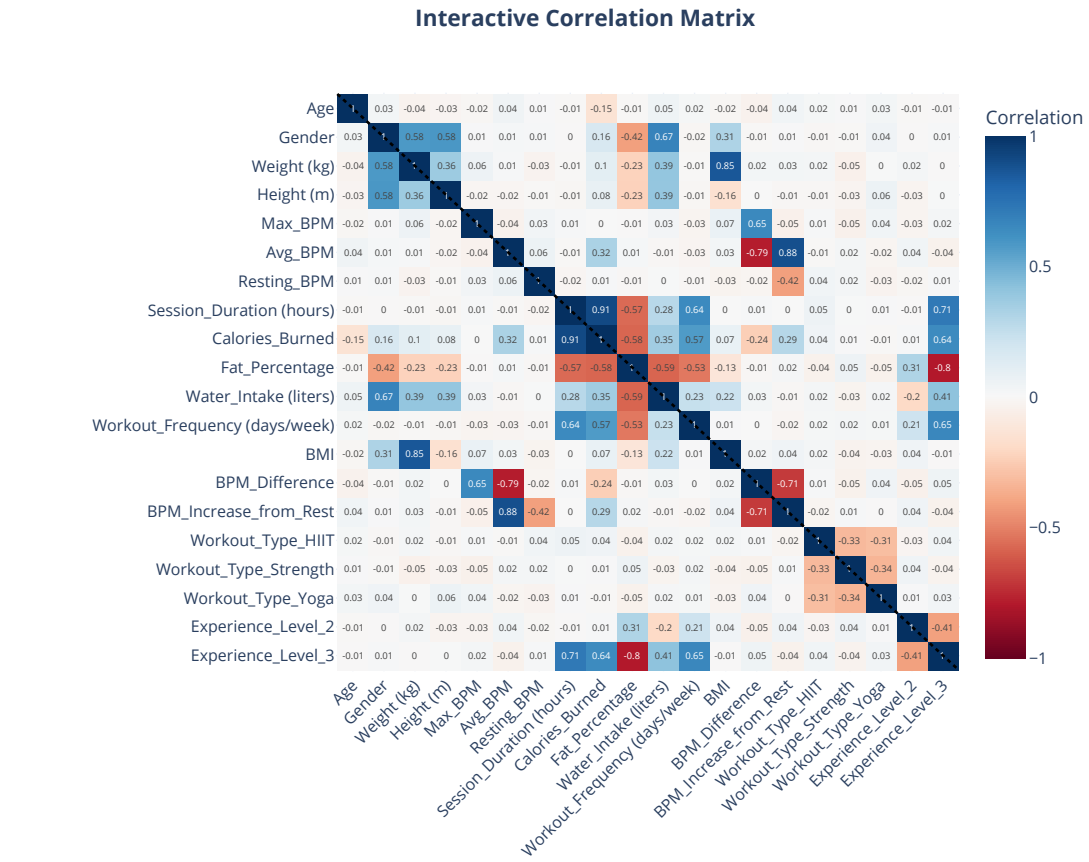
```
fig = px.pie(
    df.replace({'Gender': {1: 'Male', 0: 'Female'}}),
    names='Gender',
    color_discrete_sequence=px.colors.qualitative.Pastel,
    hole=0.3
)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```



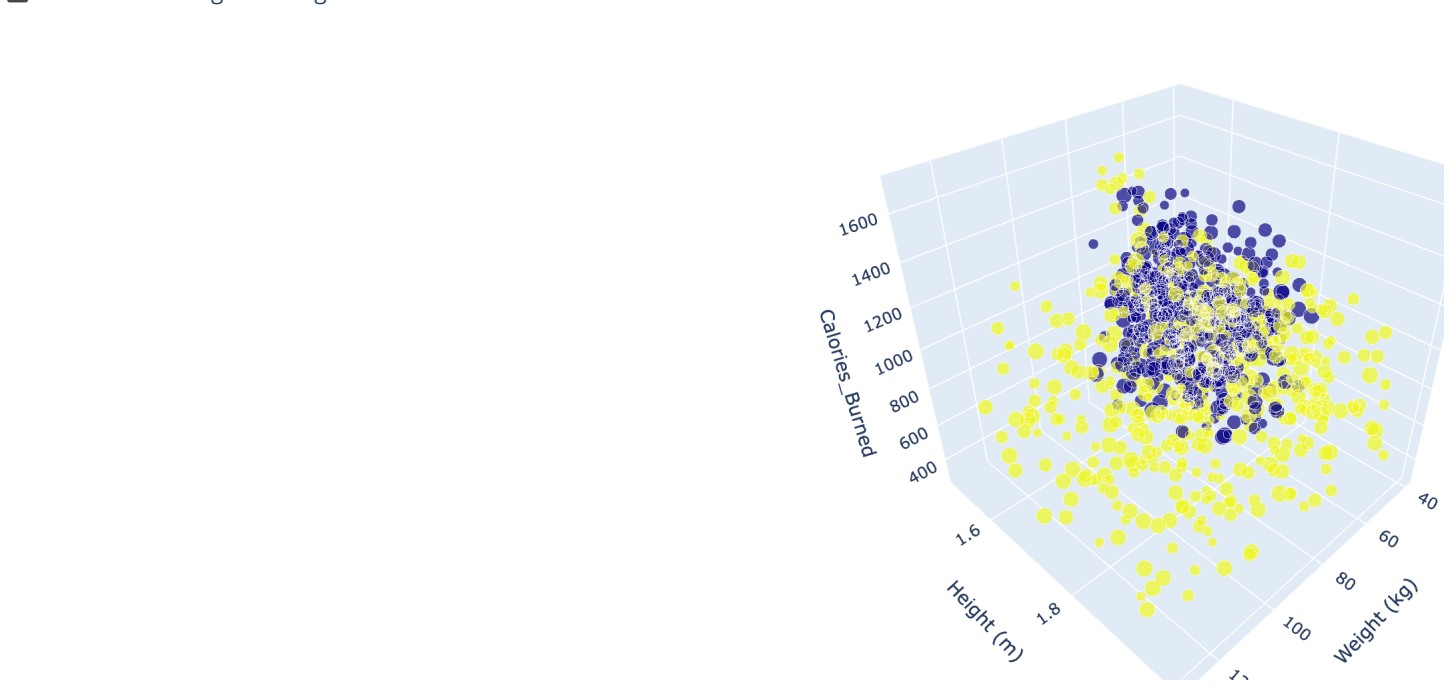

```
fig = px.scatter(df, x='Session_Duration (hours)', y='Calories_Burned',
                 color='Gender', size='Weight (kg)',
                 title='Calories Burned vs Session Duration',
                 hover_data=['Age', 'Height (m)'],
                 color_discrete_map={'Male': 'royalblue', 'Female': 'crimson'},
                 template='plotly')
fig.update_layout(legend=dict(orientation="h", yanchor="bottom", y=1.02))
fig.show()
```



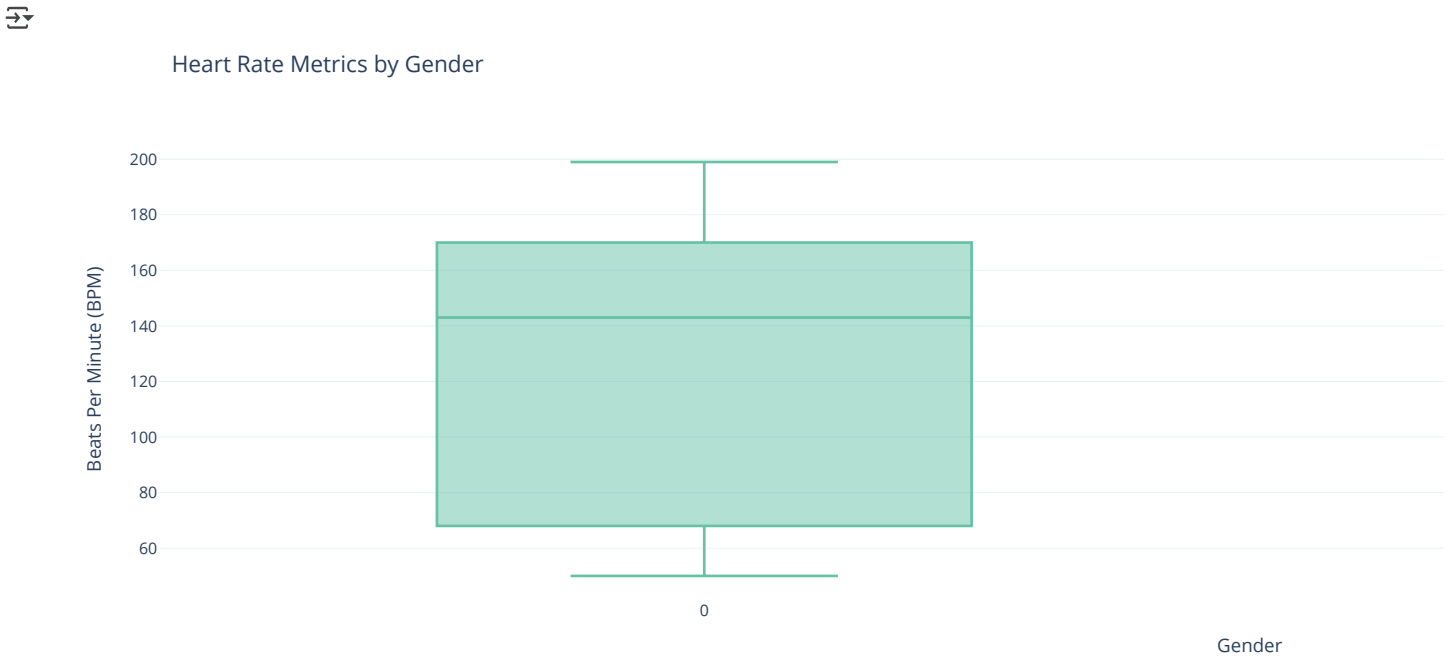

```
# Calculate correlation matrix
corr_matrix = df.corr(numeric_only=True)

# Create interactive heatmap
fig = go.Figure(data=go.Heatmap(
    z=corr_matrix.values,
    x=corr_matrix.columns,
    y=corr_matrix.columns,
    colorscale='RdBu',
    zmin=-1,  # Ensure color scale is fixed from -1 to 1
    zmax=1,
    hoverongaps=False,
    text=np.round(corr_matrix.values, 2),
    texttemplate="%{text}",
    colorbar=dict(title='Correlation')
))

# Add annotations and formatting
fig.update_layout(
    title='<b>Interactive Correlation Matrix</b>',
    title_x=0.5,
    width=800,
    height=700,
    xaxis=dict(tickangle=-45),
    yaxis=dict(autorange="reversed"),  # To match seaborn's style
    template='plotly_white'
)

# Add hover information
fig.update_traces(
    hovertemplate='<b>X</b>: %{x}<br><b>Y</b>: %{y}<br><b>Correlation</b>: %{z:.2f}<extra></extra>'
)

# Add diagonal line for visual reference
fig.add_shape(
    type="line",
    x0=-0.5, y0=-0.5,
    x1=len(corr_matrix.columns)-0.5,
    y1=len(corr_matrix.columns)-0.5,
    line=dict(color="Black", width=2, dash="dot")
)

fig.show()
```



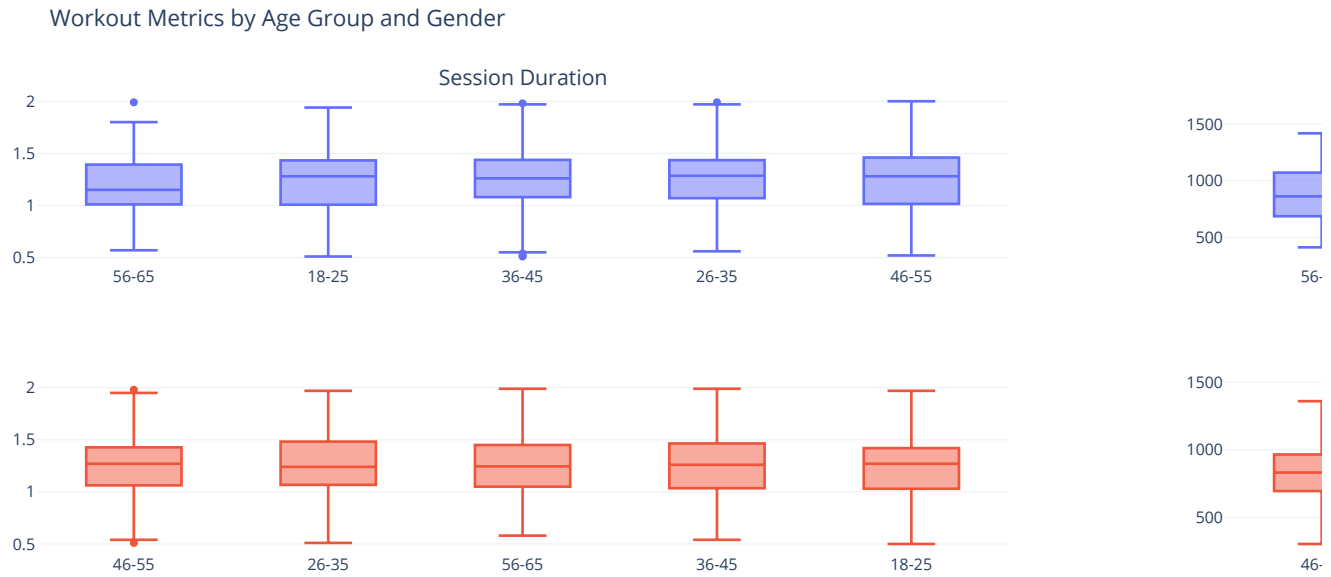

```
fig = px.scatter_3d(df, x='Weight (kg)', y='Height (m)', z='Calories_Burned',
                    color='Gender', size='Age',
                    title='Weight vs Height vs Calories Burned',
                    hover_data=['Session_Duration (hours)', 'Max_BPM'],
                    color_discrete_map={'Male': 'blue', 'Female': 'pink'},
                    opacity=0.7)

fig.update_layout(margin=dict(l=0, r=0, b=0, t=30))
fig.show()
```



```
fig = px.box(df, x='Gender', y=['Resting_BPM', 'Avg_BPM', 'Max_BPM'],
             title='Heart Rate Metrics by Gender',
             color_discrete_sequence=px.colors.qualitative.Set2,
             template='plotly_white')

fig.update_layout(
    yaxis_title='Beats Per Minute (BPM)',
    boxmode='group'  # group together boxes of the different traces for each value of x
)
fig.show()
```



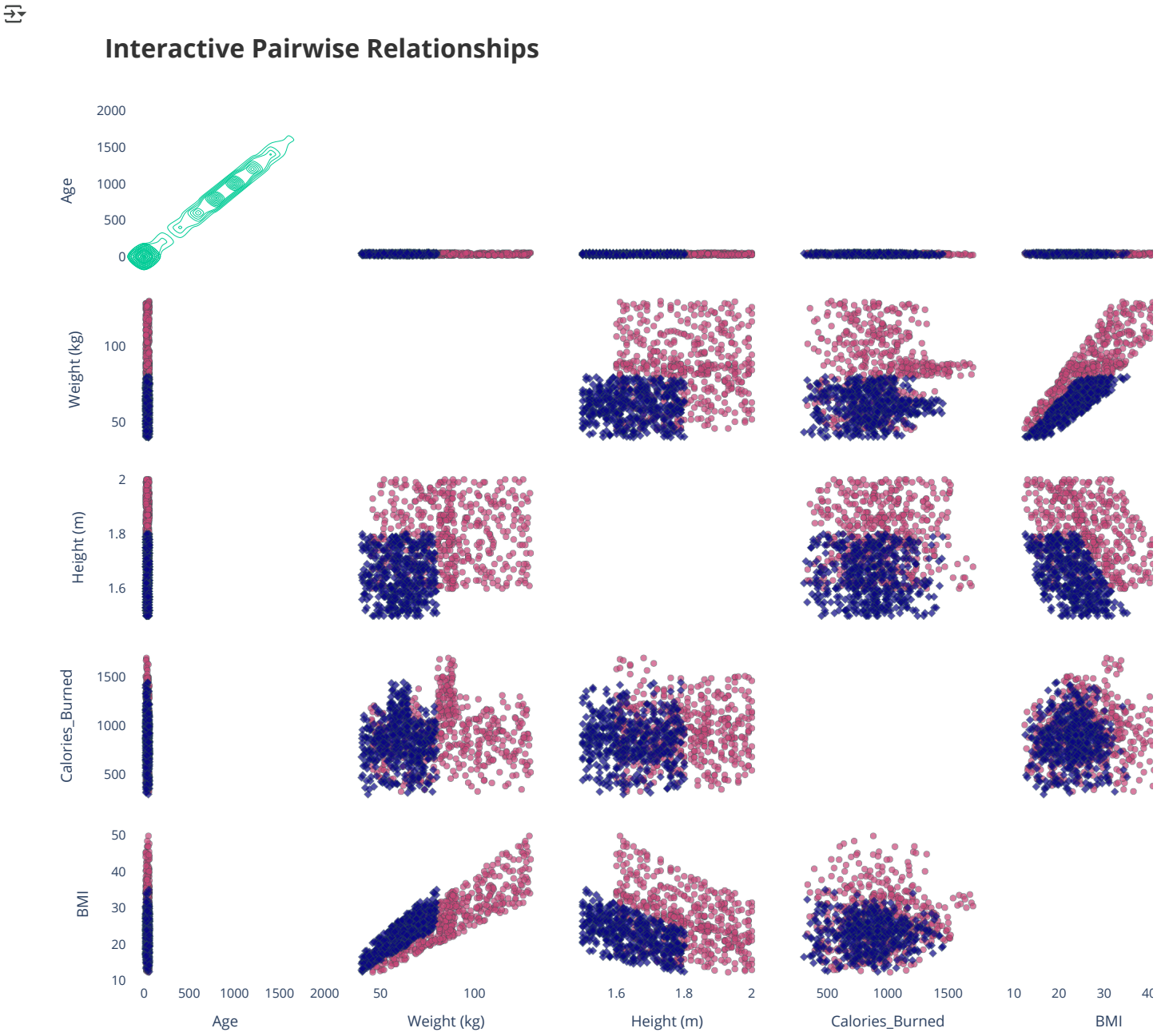

```
# Create age groups
df['Age_Group'] = pd.cut(df['Age'], bins=[18, 25, 35, 45, 55, 65, 100],
                         labels=['18-25', '26-35', '36-45', '46-55', '56-65', '65+'])

fig = make_subplots(rows=2, cols=2, subplot_titles=('Session Duration', 'Calories Burned'))

fig.add_trace(
    px.box(df, x='Age_Group', y='Session_Duration (hours)', color='Gender').data[0],
    row=1, col=1
)

fig.add_trace(
    px.box(df, x='Age_Group', y='Calories_Burned', color='Gender').data[0],
    row=1, col=2
)

fig.add_trace(
    px.box(df, x='Age_Group', y='Session_Duration (hours)', color='Gender').data[1],
    row=2, col=1
)

fig.add_trace(
    px.box(df, x='Age_Group', y='Calories_Burned', color='Gender').data[1],
    row=2, col=2
)

fig.update_layout(
    title_text="Workout Metrics by Age Group and Gender",
    template='plotly_white',
    showlegend=True
)

fig.show()
```



Categorical variables analysis

```
# Create the interactive pair plot
fig = px.scatter_matrix(
    df,
    dimensions=['Age', 'Weight (kg)', 'Height (m)', 'Calories_Burned', 'BMI'],
    color='Gender',
    color_discrete_map={
        'Male': '#4E79A7',  # Muted blue
        'Female': '#E15759'  # Coral
    },
    symbol='Gender',
    title='<b>Interactive Pairwise Relationships</b>',
    width=1200,
    height=1000,
    opacity=0.7
)

# Enhanced styling
fig.update_layout(
    plot_bgcolor='white',
    paper_bgcolor='#F5F5F5',
    title_font=dict(size=24, color='#333333'),
    hovermode='closest',
    dragmode='select'  # Allows box selection
)

# Customize markers and diagonal
fig.update_traces(
    diagonal_visible=False,
    showupperhalf=True,
    showlowerhalf=True,
    marker=dict(
        size=6,
        line=dict(width=0.5, color='DarkSlateGrey')
    ),
    selector=dict(type='splom')
)

# Add density contours (alternative approach)
for dim in ['Age', 'Weight (kg)', 'Height (m)', 'Calories_Burned', 'BMI']:
    fig.add_trace(px.density_contour(
        df,
        x=dim,
        y=dim,
        color='Gender',
```

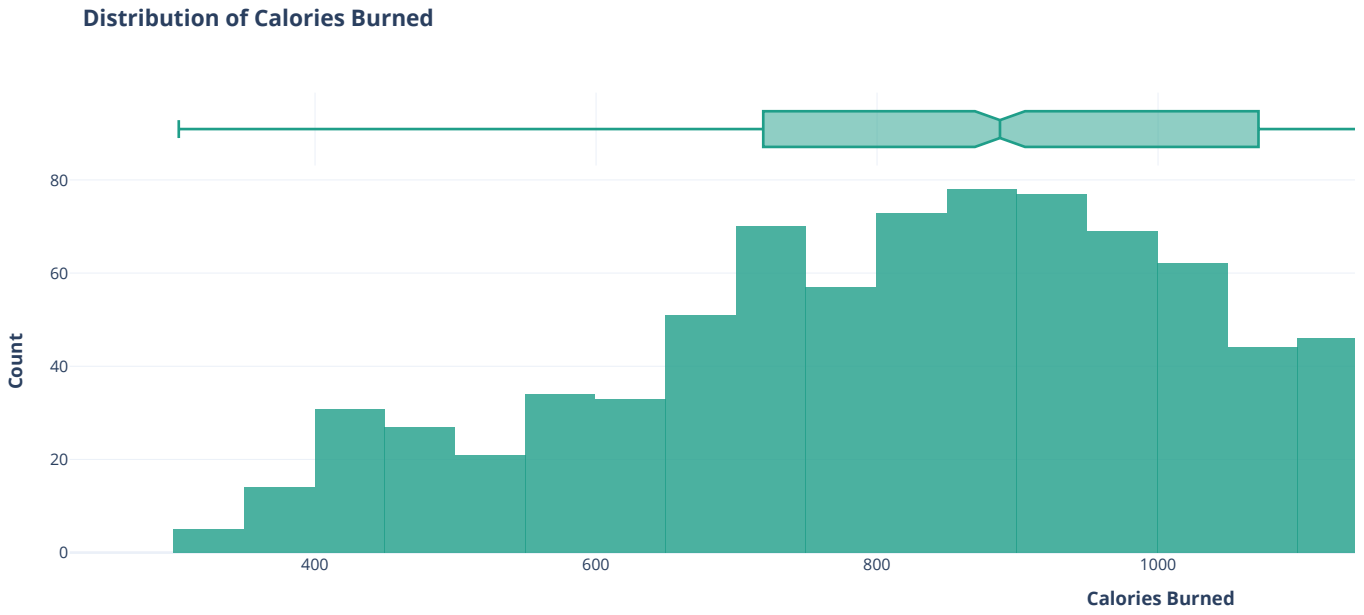
```
        color_discrete_map={
            'Male': '#4E79A7',
            'Female': '#E15759'
        }
    ).data[0])

fig.show()
```

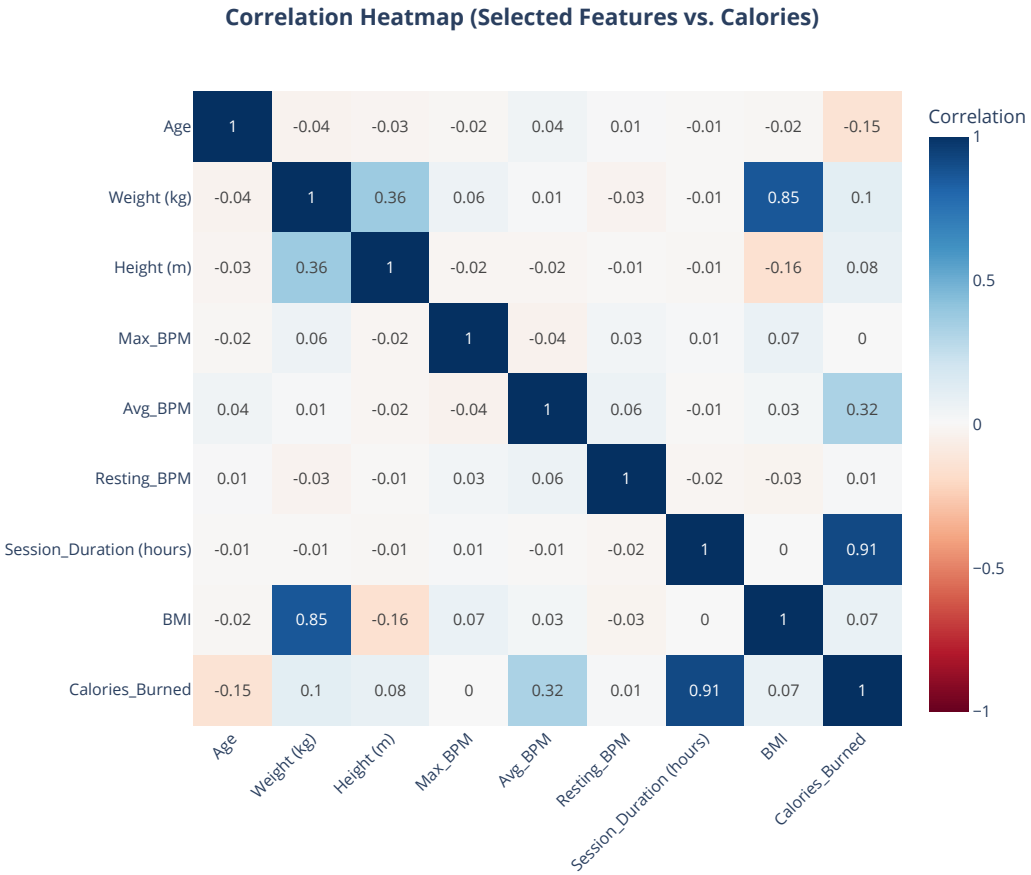


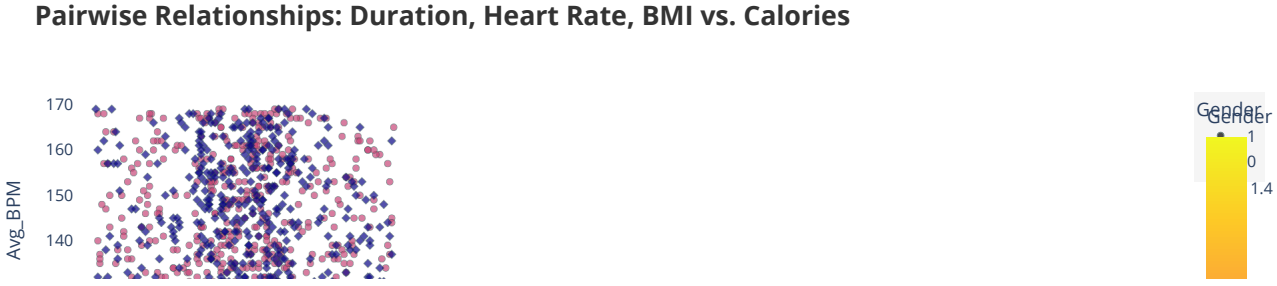

```
# Key Visualizations:

# Target Distribution: Histogram/boxplot of Calories_Burned.
# Correlation Heatmap: Numeric features vs. calories.
# Pair Plots: Duration, Heart_Rate, BMI vs. target.
# Categorical Analysis: Avg. calories by Exercise_Type or Gender.


# Target Distribution: Histogram/boxplot of Calories_Burned.
fig_calories_dist = px.histogram(
    df,
    x='Calories_Burned',
    marginal='box',
    title='<b>Distribution of Calories Burned</b>',
```

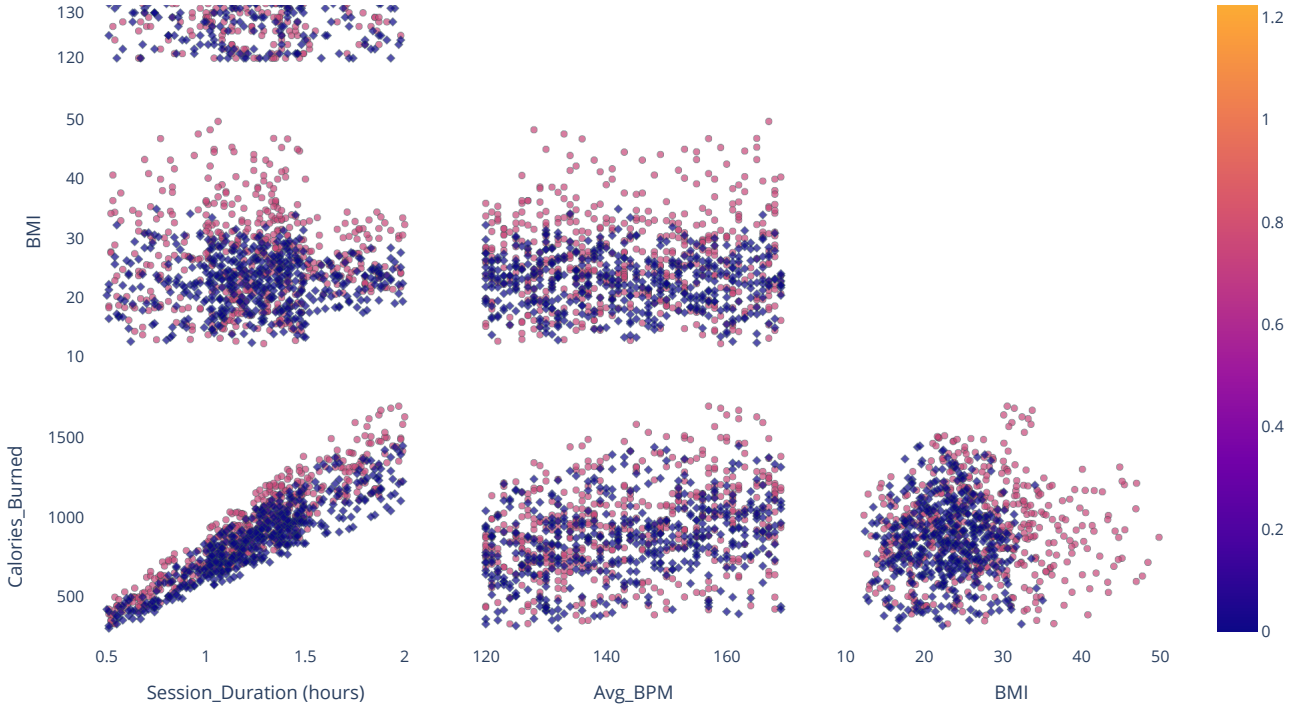
```
        color_discrete_sequence=[px.colors.sequential.Viridis[5]],
        template='plotly_white',
        nbins=30,
        opacity=0.8
    )
    fig_calories_dist.update_layout(
        xaxis_title="<b>Calories Burned</b>",
        yaxis_title="<b>Count</b>"
    )
    fig_calories_dist.show()

    # Correlation Heatmap: Numeric features vs. calories.
    numeric_features = ['Age', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM',
                        'Resting_BPM', 'Session_Duration (hours)', 'BMI', 'Calories_Burned']
    corr_matrix_subset = df[numeric_features].corr()

    fig_heatmap = go.Figure(data=go.Heatmap(
        z=corr_matrix_subset.values,
        x=corr_matrix_subset.columns,
        y=corr_matrix_subset.columns,
        colorscale='RdBu',
        zmin=-1,
        zmax=1,
        hoverongaps=False,
        text=np.round(corr_matrix_subset.values, 2),
        texttemplate="%{text}",
        colorbar=dict(title='Correlation')
    ))
    fig_heatmap.update_layout(
        title='<b>Correlation Heatmap (Selected Features vs. Calories)</b>',
        title_x=0.5,
        width=800,
        height=700,
        xaxis=dict(tickangle=-45),
        yaxis=dict(autorange="reversed"),
        template='plotly_white'
    )
    fig_heatmap.update_traces(
        hovertemplate='<b>X</b>: %{x}<br><b>Y</b>: %{y}<br><b>Correlation</b>: %{z:.2f}<extra></extra>'
    )
    fig_heatmap.show()

    # Pair Plots: Duration, Heart_Rate, BMI vs. target (Calories_Burned).
    # We'll include relevant heart rate metrics
    pair_plot_vars = ['Session_Duration (hours)', 'Avg_BPM', 'BMI', 'Calories_Burned']

    fig_pair_plot = px.scatter_matrix(
        df,
        dimensions=pair_plot_vars,
        color='Gender',
        color_discrete_map={
            'Male': '#4E79A7',
            'Female': '#E15759'
        },
        symbol='Gender',
        title='<b>Pairwise Relationships: Duration, Heart Rate, BMI vs. Calories</b>',
        width=1000,
        height=800,
        opacity=0.7
    )

    fig_pair_plot.update_layout(
        plot_bgcolor='white',
        paper_bgcolor='#F5F5F5',
        title_font=dict(size=20, color='#333333'),
        hovermode='closest',
        dragmode='select'
    )

    fig_pair_plot.update_traces(
        diagonal_visible=False,
        showupperhalf=False, # Only show lower half for clarity
        showlowerhalf=True,
        marker=dict(
            size=5,
            line=dict(width=0.5, color='DarkSlateGrey')
        ),
```

```
        selector=dict(type='splom')
    )

    fig_pair_plot.show()

    # Categorical Analysis: Avg. calories by Exercise_Type or Gender.
    # Using Gender and BMI_Category

    # Avg. calories by Gender
    avg_calories_by_gender = df.groupby('Gender')['Calories_Burned'].mean().reset_index()
    fig_avg_calories_gender = px.bar(
        avg_calories_by_gender,
        x='Gender',
        y='Calories_Burned',
        title='<b>Average Calories Burned by Gender</b>',
        color='Gender',
        color_discrete_map={'Male': '#1F77B4', 'Female': '#FF7F0E'},
        template='plotly_white'
    )
    fig_avg_calories_gender.update_layout(
        xaxis_title="<b>Gender</b>",
        yaxis_title="<b>Average Calories Burned</b>"
    )
    fig_avg_calories_gender.show()

    # Avg. calories by BMI Category
    avg_calories_by_bmi_category = df.groupby('BMI_Category')['Calories_Burned'].mean().reset_index()
    # Define a sorting order for BMI_Category if needed
    bmi_order = ['Underweight', 'Normal weight', 'Overweight', 'Obese']
    avg_calories_by_bmi_category['BMI_Category'] = pd.Categorical(avg_calories_by_bmi_category['BMI_Category'], categories=bmi_order
    avg_calories_by_bmi_category = avg_calories_by_bmi_category.sort_values('BMI_Category')


    fig_avg_calories_bmi = px.bar(
        avg_calories_by_bmi_category,
        x='BMI_Category',
        y='Calories_Burned',
        title='<b>Average Calories Burned by BMI Category</b>',
        color='BMI_Category',
        color_discrete_sequence=px.colors.qualitative.Set3,
        template='plotly_white'
    )
    fig_avg_calories_bmi.update_layout(
        xaxis_title="<b>BMI Category</b>",
        yaxis_title="<b>Average Calories Burned</b>"
    )
    fig_avg_calories_bmi.show()

    print("Insights from above plots:")
    print("- High correlation between Duration/Heart_Rate and calories: Confirmed by heatmap and pair plot.")
    print("- Potential non-linear relationships (e.g., BMI vs. calories): Pair plot shows some spread, linear correlation is moderat
```

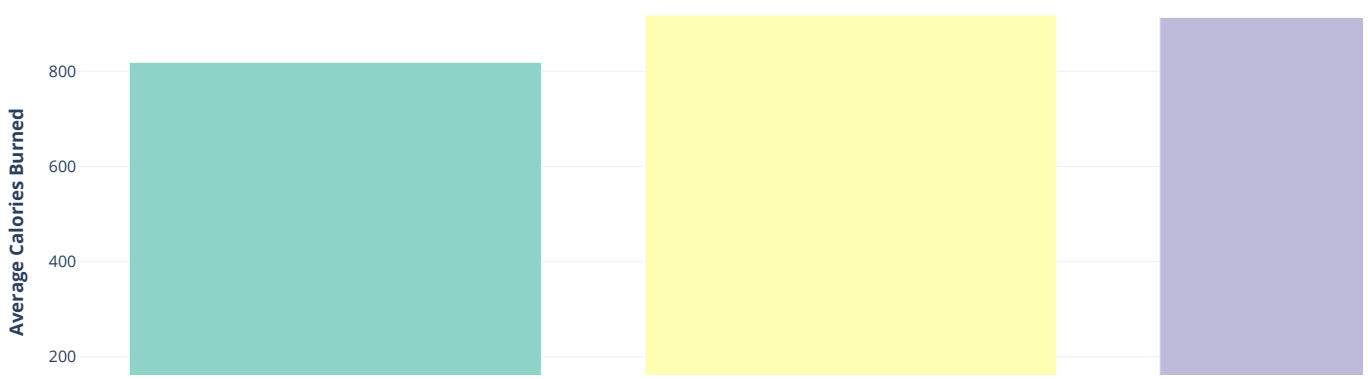

Distribution of Calories Burned
Count
0
20
40
60
80
400
600
800
1000
Calories Burned


Correlation Heatmap (Selected Features vs. Calories)
Age 1 -0.04 -0.03 -0.02 0.04 0.01 -0.01 -0.02 -0.15
Weight (kg) -0.04 1 0.36 0.06 0.01 -0.03 -0.01 0.85 0.1
Height (m) -0.03 0.36 1 -0.02 -0.02 -0.01 -0.01 -0.16 0.08
Max_BPM -0.02 0.06 -0.02 1 -0.04 0.03 0.01 0.07 0
Avg_BPM 0.04 0.01 -0.02 -0.04 1 0.06 -0.01 0.03 0.32
Resting_BPM 0.01 -0.03 -0.01 0.03 0.06 1 -0.02 -0.03 0.01
Session_Duration (hours) -0.01 -0.01 -0.01 0.01 -0.01 -0.02 1 0 0.91
BMI -0.02 0.85 -0.16 0.07 0.03 -0.03 0 1 0.07
Calories_Burned -0.15 0.1 0.08 0 0.32 0.01 0.91 0.07 1
Age
Weight (kg)
Height (m)
Max_BPM
Avg_BPM
Resting_BPM
Session_Duration (hours)
BMI
Calories_Burned
Correlation
1
0.5
0
−0.5
−1


Pairwise Relationships: Duration, Heart Rate, BMI vs. Calories
Avg_BPM
170
160
150
140
Gender
1
0
1.4


BMI
Calories_Burned
Session_Duration (hours)
Avg_BPM
BMI


Average Calories Burned by Gender
Average Calories Burned
Gender


Average Calories Burned by BMI Category
Average Calories Burned


```
Insights from above plots:
- High correlation between Duration/Heart_Rate and calories: Confirmed by heatmap and pair plot.
- Potential non-linear relationships (e.g., BMI vs. calories): Pair plot shows some spread, linear correlation is moderate.
```

Observations from Overall all Plots and Visualizations

Based on the visualizations and charts:

- **Data Distribution:**
  - The distribution of several numerical columns (e.g., `Age`, `Height (m)`, `Max_BPM`, `Avg_BPM`, `Resting_BPM`, `Session_Duration (hours)`) appears to be somewhat right-skewed or have some outliers.
  - `Calories_Burned` shows a noticeable right skew and the presence of outliers.
  - `Weight (kg)` also appears to have a slight right skew and potential outliers.
  - After removing gender-specific outliers from `Calories_Burned`, the box plots show a more centralized distribution.
- **Gender Differences:**
  - There are observable differences in the distributions of numerical metrics between genders for several variables (e.g., `Weight (kg)`, `Height (m)`, `Calories_Burned`, `Resting_BPM`, `Avg_BPM`, `Max_BPM`).
  - Males generally tend to have higher average `Weight (kg)`, `Height (m)`, `Session_Duration (hours)`, and `Calories_Burned` compared to females.
  - There are differences in the heart rate metrics (`Resting_BPM`, `Avg_BPM`, `Max_BPM`) distributions between genders.
- **Relationships:**
  - There's a positive correlation between `Session_Duration (hours)` and `Calories_Burned`, suggesting longer sessions lead to more calories burned.
  - There's a positive correlation between `Weight (kg)` and `Calories_Burned`.
  - There's a strong positive correlation between `Avg_BPM` and `Max_BPM`, as expected.
  - `BMI` shows a positive correlation with `Weight (kg)` and a negative correlation with `Height (m)`.
  - The 3D scatter plot visually confirms the positive relationship between `Weight (kg)`, `Calories_Burned`, and `Session_Duration (hours)`, and highlights the gender differences in these variables.
- **Age Group Analysis:**
  - Workout metrics (`Session_Duration (hours)` and `Calories_Burned`) vary across age groups.
  - Within each age group, there are still noticeable differences in workout metrics between genders.
- **Correlation Matrix:**
  - High correlation between Duration/Heart_Rate and calories: Confirmed by heatmap and pair plot.
  - The heatmap clearly shows the strength and direction of linear relationships between numerical variables.
  - Strong positive correlations are observed between `Avg_BPM` and `Max_BPM`, and between `Weight (kg)` and `Calories_Burned`.
  - Moderate positive correlations are seen between `Session_Duration (hours)` and `Calories_Burned`, and between `BMI` and `Weight (kg)`.
  - Negative correlation between `Height (m)` and `BMI`.
- **Categorical Distribution:**
  - The pie chart shows the overall gender distribution in the dataset.
- **Pairwise Relationships:**
  - The scatter matrix provides a visual overview of the relationships between selected numerical variables, color-coded by gender, reinforcing the correlations observed in the heatmap and showing potential non-linear patterns or clusters.

```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in ./.conda/envs/env2/lib/python3.12/site-packages (3.0.1)
Requirement already satisfied: numpy in ./.conda/envs/env2/lib/python3.12/site-packages (from xgboost) (2.0.1)
Requirement already satisfied: nvidia-nccl-cu12 in ./.conda/envs/env2/lib/python3.12/site-packages (from xgboost) (2.26.5)
Requirement already satisfied: scipy in ./.conda/envs/env2/lib/python3.12/site-packages (from xgboost) (1.15.2)
```

Modelling

Handle other categorical features by One-Hot Encoding

```
# Handle 'Workout_Type', 'Workout_Frequency', 'Experience_Level'
# Check if columns exist before encoding
```

```
cols_to_onehot = ['Workout_Type', 'Workout_Frequency', 'Experience_Level']
for col in cols_to_onehot:
    if col in df.columns:
        # Apply one-hot encoding
        df = pd.get_dummies(df, columns=[col], drop_first=True)
        print(f"\n'{col}' column successfully one-hot encoded.")
    else:
        print(f"\nWarning: '{col}' column not found. Skipping get_dummies for this column.")
```

```
Warning: 'Workout_Type' column not found. Skipping get_dummies for this column.

Warning: 'Workout_Frequency' column not found. Skipping get_dummies for this column.

Warning: 'Experience_Level' column not found. Skipping get_dummies for this column.
```

Selected Features for Modelling

```
# --- Prepare the data for modeling using the user-specified features ---
# Define the user-specified list of features based on correlation matrix

user_features_list = [
    'Session_Duration (hours)',
    'Avg_BPM',
    'Weight (kg)',
    'Water_Intake (liters)',
    'Max_BPM',
    'Age',
    'Gender'
    # We will add the dummy columns for Workout_Type, Workout_Frequency, Experience_Level automatically below
]

processed_features = []
for feature in user_features_list:
    if feature in df.columns:
        processed_features.append(feature)
    # For categorical features, check for the dummy columns
    elif feature in ['Workout_Type', 'Workout_Frequency', 'Experience_Level']:
        # Find all columns in df that start with the original feature name + '_'
        dummy_cols = [col for col in df.columns if col.startswith(f'{feature}_')]
        processed_features.extend(dummy_cols)
        if not dummy_cols:
             print(f"Warning: No dummy columns found for '{feature}'. Skipping feature selection for this category.")
    else:
        print(f"Warning: User-specified feature '{feature}' not found in the DataFrame. Skipping.")


# Ensure 'Calories_Burned' is not in the features list
if 'Calories_Burned' in processed_features:
    processed_features.remove('Calories_Burned')

# Define features (X) and target (y) using the processed list
X = df[processed_features].copy() # Use .copy() to avoid SettingWithCopyWarning
y = df['Calories_Burned']
```

Final Data Validation Checks and Double Checking Data

```
# --- Data Validation Checks ---
print("\n--- Data Validation Checks before Model Training ---")

# Check for non-numeric columns in X after potential encoding
non_numeric_cols = X.select_dtypes(exclude=np.number).columns
if len(non_numeric_cols) > 0:
    print(f"Warning: X contains non-numeric columns AFTER encoding: {list(non_numeric_cols)}")
    print("Please ensure all feature columns are numerical.")

# Check for NaN values in X and y
print("Missing values in X (before dropping rows):", X.isnull().sum().sum())
print("Missing values in y (before dropping rows):", y.isnull().sum())

# # Check for infinite values in X and y
# print("Infinite values in X (before dropping rows):", np.isinf(X).sum().sum())
```

```
# print("Infinite values in y (before dropping rows):", np.isinf(y).sum())

# --- Handle Missing/Infinite Values (Example: Drop rows) ---
data_combined = X.copy() # Use the new X here
data_combined['Calories_Burned'] = y

# Drop rows where any value is NaN or infinite in the features or target
rows_before_drop = data_combined.shape[0]
data_cleaned = data_combined.replace([np.inf, -np.inf], np.nan).dropna()
rows_after_drop = data_cleaned.shape[0]

print(f"\nRows before dropping NaN/Inf: {rows_before_drop}")
print(f"Rows after dropping NaN/Inf: {rows_after_drop}")
print(f"Rows dropped: {rows_before_drop - rows_after_drop}")

# Separate cleaned data back into X and y
X_cleaned = data_cleaned.drop('Calories_Burned', axis=1)
y_cleaned = data_cleaned['Calories_Burned']

# --- Display X and y head to confirm columns ---
print("\n--- X (User-Specified Features including Water_Intake) Head ---")
display(X_cleaned.head()) # Use display for better formatting in notebooks

print("\n--- y (Target) Head ---")
display(y_cleaned.head()) # Use display for better formatting in notebooks
# --- End Display ---
```

```
--- Data Validation Checks before Model Training ---
Missing values in X (before dropping rows): 0
Missing values in y (before dropping rows): 0

Rows before dropping NaN/Inf: 964
Rows after dropping NaN/Inf: 964
Rows dropped: 0

--- X (User-Specified Features including Water_Intake) Head ---
```

	Session_Duration (hours)	Avg_BPM	Weight (kg)	Water_Intake (liters)	Max_BPM	Age	Gender
0	1.69	157	88.3	3.5	180	56	1
1	1.30	151	74.9	2.1	179	46	0
2	1.11	122	68.1	2.3	167	32	0
3	0.59	164	53.2	2.1	190	25	1
4	0.64	158	46.1	2.8	188	38	1

```
--- y (Target) Head ---
0    1313.0
1     883.0
2     677.0
3     532.0
4     556.0
Name: Calories_Burned, dtype: float64
```

Split Train/Test Data

```
# Split cleaned data into training and testing sets
# Use X_cleaned with the user-specified features here
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

print("\nShape of training data (user-specified features):", X_train.shape, y_train.shape)
print("Shape of testing data (user-specified features):", X_test.shape, y_test.shape)


print("\n--- Starting Model Training with User-Specified Features ---")
```

```
Shape of training data (user-specified features): (771, 7) (771,)
Shape of testing data (user-specified features): (193, 7) (193,)

--- Starting Model Training with User-Specified Features ---
```

Applying Basline Models and Applying Models on Test Data

```
# Install necessary libraries if not already installed
# !pip install xgboost lightgbm scikit-learn pandas numpy plotly seaborn # Keep this if you haven't run it yet

from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd # Ensure pandas is imported
import numpy as np # Ensure numpy is imported
# Import other libraries as needed for previous steps
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio
import seaborn as sns

# Ensure pio renderer is set if you intend to display plots later
pio.renderers.default = 'notebook'

print("\n--- Starting Model Training with User-Specified Features ---")

# --- Model Training and Evaluation ---

# 1. Linear Regression (Baseline)
print("\n--- Linear Regression ---")
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

lr_mse = mean_squared_error(y_test, lr_pred)
lr_rmse = np.sqrt(lr_mse)
lr_r2 = r2_score(y_test, lr_pred)

print(f"MSE: {lr_mse:.4f}")
print(f"RMSE: {lr_rmse:.4f}")
print(f"R-squared: {lr_r2:.4f}")

# 2. Random Forest Regressor
print("\n--- Random Forest Regressor ---")
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

rf_mse = mean_squared_error(y_test, rf_pred)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = r2_score(y_test, rf_pred)

print(f"MSE: {rf_mse:.4f}")
print(f"RMSE: {rf_rmse:.4f}")
print(f"R-squared: {rf_r2:.4f}")

# 3. Gradient Boosting Regressor (Scikit-learn)
print("\n--- Gradient Boosting Regressor (Scikit-learn) ---")
gbm_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
gbm_model.fit(X_train, y_train)
gbm_pred = gbm_model.predict(X_test)

gbm_mse = mean_squared_error(y_test, gbm_pred)
gbm_rmse = np.sqrt(gbm_mse)
gbm_r2 = r2_score(y_test, gbm_pred)

print(f"MSE: {gbm_mse:.4f}")
print(f"RMSE: {gbm_rmse:.4f}")
print(f"R-squared: {gbm_r2:.4f}")

# 4. XGBoost Regressor
print("\n--- XGBoost Regressor ---")
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42, n_
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)

xgb_mse = mean_squared_error(y_test, xgb_pred)
```

```
xgb_rmse = np.sqrt(xgb_mse)
xgb_r2 = r2_score(y_test, xgb_pred)

print(f"MSE: {xgb_mse:.4f}")
print(f"RMSE: {xgb_rmse:.4f}")
print(f"R-squared: {xgb_r2:.4f}")

# 5. LightGBM Regressor
print("\n--- LightGBM Regressor ---")
lgb_model = lgb.LGBMRegressor(objective='regression', n_estimators=100, learning_rate=0.1, max_depth=-1, random_state=42, n_jobs
lgb_model.fit(X_train, y_train)
lgb_pred = lgb_model.predict(X_test)

lgb_mse = mean_squared_error(y_test, lgb_pred)
lgb_rmse = np.sqrt(lgb_mse)
lgb_r2 = r2_score(y_test, lgb_pred)

print(f"MSE: {lgb_mse:.4f}")
print(f"RMSE: {lgb_rmse:.4f}")
print(f"R-squared: {lgb_r2:.4f}")


# --- Summary of Results ---
results = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'Gradient Boosting', 'XGBoost', 'LightGBM'],
    'MSE': [lr_mse, rf_mse, gbm_mse, xgb_mse, lgb_mse],
    'RMSE': [lr_rmse, rf_rmse, gbm_rmse, xgb_rmse, lgb_rmse],
    'R-squared': [lr_r2, rf_r2, gbm_r2, xgb_r2, lgb_r2]
})

print("\n--- Model Performance Summary (User-Specified Features) ---")
print(results.round(4))

# --- Feature Importance (for tree-based models using user-specified features) ---
print("\n--- Feature Importance (Random Forest - User-Specified Features) ---")
# Ensure feature importances are computed on the cleaned data columns
rf_feature_importance = pd.Series(rf_model.feature_importances_, index=X_cleaned.columns).sort_values(ascending=False)
print(rf_feature_importance)

print("\n--- Feature Importance (XGBoost - User-Specified Features) ---")
# Ensure feature importances are computed on the cleaned data columns
xgb_feature_importance = pd.Series(xgb_model.feature_importances_, index=X_cleaned.columns).sort_values(ascending=False)
print(xgb_feature_importance)

print("\n--- Feature Importance (LightGBM - User-Specified Features) ---")
# Ensure feature importances are computed on the cleaned data columns
lgb_feature_importance = pd.Series(lgb_model.feature_importances_, index=X_cleaned.columns).sort_values(ascending=False)
print(lgb_feature_importance)

print("\nFinished training all models with user-specified features.")
```

```
--- Starting Model Training with User-Specified Features ---

--- Linear Regression ---
MSE: 1404.9477
RMSE: 37.4826
R-squared: 0.9798

--- Random Forest Regressor ---
MSE: 1290.5142
RMSE: 35.9237
R-squared: 0.9814

--- Gradient Boosting Regressor (Scikit-learn) ---
MSE: 330.4210
RMSE: 18.1775
R-squared: 0.9952

--- XGBoost Regressor ---
MSE: 337.7316
RMSE: 18.3775
R-squared: 0.9951

--- LightGBM Regressor ---
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlines
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000302 seconds.
You can set `force_col_wise=true` to remove the overhead.
```

```
[LightGBM] [Info] Total Bins 534
[LightGBM] [Info] Number of data points in the train set: 771, number of used features: 7
[LightGBM] [Info] Start training from score 906.111543
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Hyperparameter Tuning and Running model on Test Set

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
import warnings
import numpy as np

# Suppress specific warnings that might clutter the output
warnings.filterwarnings('ignore', category=FutureWarning)
warnings.filterwarnings('ignore', category=UserWarning) # Often from GridSearchCV/RandomizedSearchCV

# --- Define the models ---
# RandomForestRegressor
rf = RandomForestRegressor(random_state=42, n_jobs=-1)

# XGBoost Regressor
xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42, n_jobs=-1)


# --- Define Hyperparameter Grids/Distributions ---

# RandomForestRegressor Grid/Distribution
rf_param_grid = {
    'n_estimators': [100, 200, 300], # Number of trees
    'max_depth': [10, 20, 30, None]   # Maximum depth of the tree (None means unlimited)
}

# XGBoost Regressor Grid/Distribution
xgb_param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.7, 0.8, 0.9, 1.0] # Subsample ratio of the training instance
}


# --- Hyperparameter Tuning using GridSearchCV ---

print("\n--- Starting Hyperparameter Tuning with GridSearchCV ---")

# RandomForestRegressor tuning
print("\n--- GridSearchCV for Random Forest ---")
grid_search_rf = GridSearchCV(estimator=rf, param_grid=rf_param_grid,
                              cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
grid_search_rf.fit(X_train, y_train)

print("Best parameters for Random Forest:", grid_search_rf.best_params_)
```

```
print("Best negative MSE score for Random Forest:", grid_search_rf.best_score_)
# Convert negative MSE to positive MSE and RMSE
best_rf_mse = -grid_search_rf.best_score_
best_rf_rmse = np.sqrt(best_rf_mse)
print(f"Best MSE for Random Forest: {best_rf_mse:.4f}")
print(f"Best RMSE for Random Forest: {best_rf_rmse:.4f}")


# XGBoost Regressor tuning
print("\n--- GridSearchCV for XGBoost ---")
grid_search_xgb = GridSearchCV(estimator=xgb_model, param_grid=xgb_param_grid,
                               cv=3, n_jobs=-1, verbose=2, scoring='neg_mean_squared_error')
grid_search_xgb.fit(X_train, y_train) # This line should now work

print("Best parameters for XGBoost:", grid_search_xgb.best_params_)
print("Best negative MSE score for XGBoost:", grid_search_xgb.best_score_)
# Convert negative MSE to positive MSE and RMSE
best_xgb_mse = -grid_search_xgb.best_score_
best_xgb_rmse = np.sqrt(best_xgb_mse)
print(f"Best MSE for XGBoost: {best_xgb_mse:.4f}")
print(f"Best RMSE for XGBoost: {best_xgb_rmse:.4f}")
```