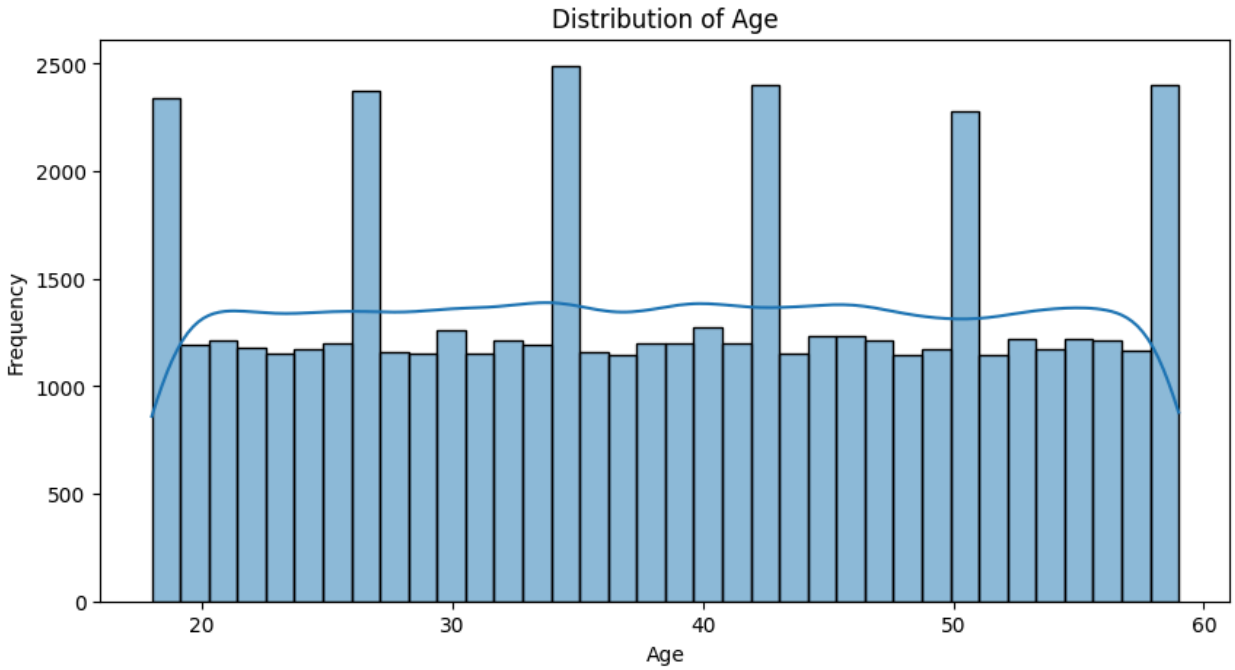
In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

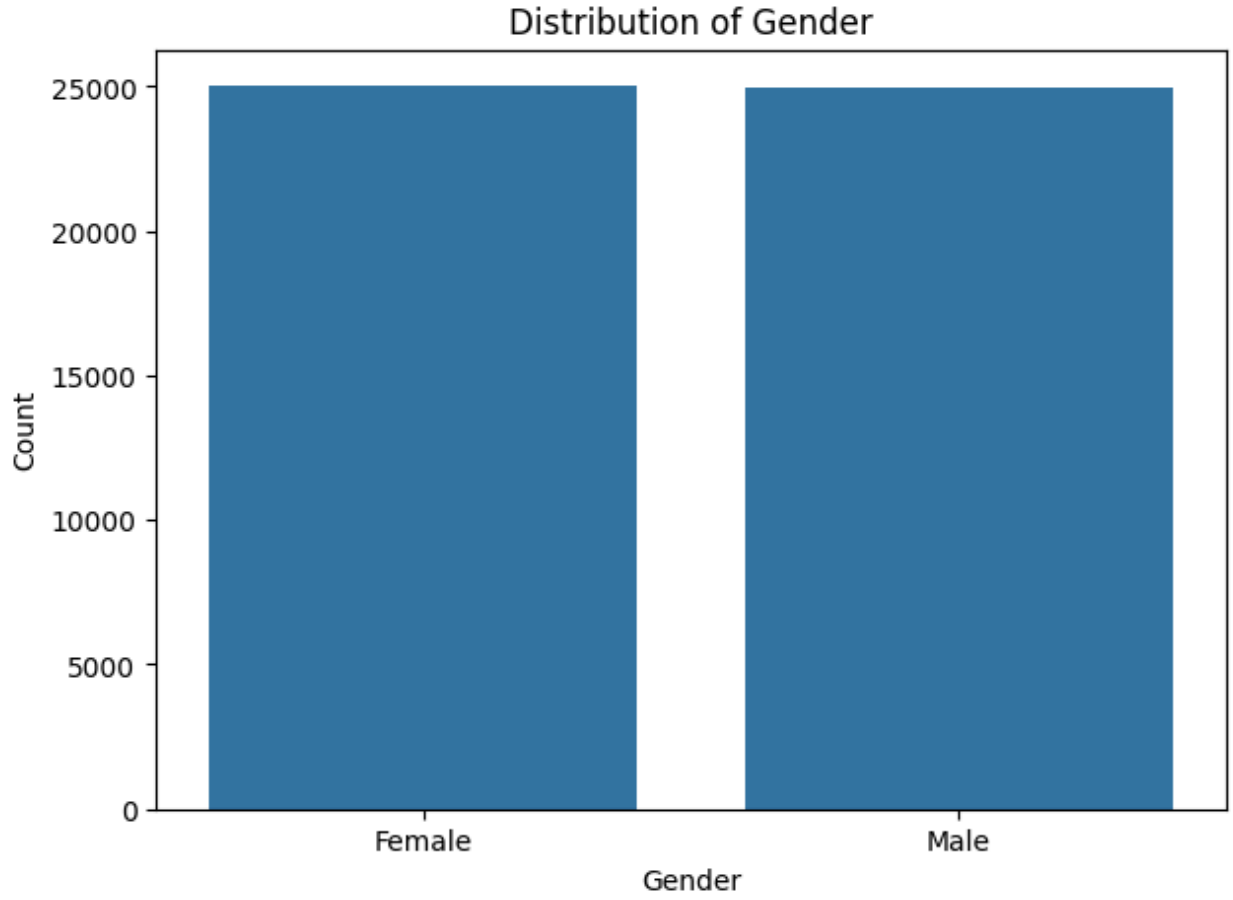
In [2]:
```
import pandas as pd

df = pd.read_excel("/content/SmartSizeAI_ClothingSizeDataset_.xlsx")
df.head()
```

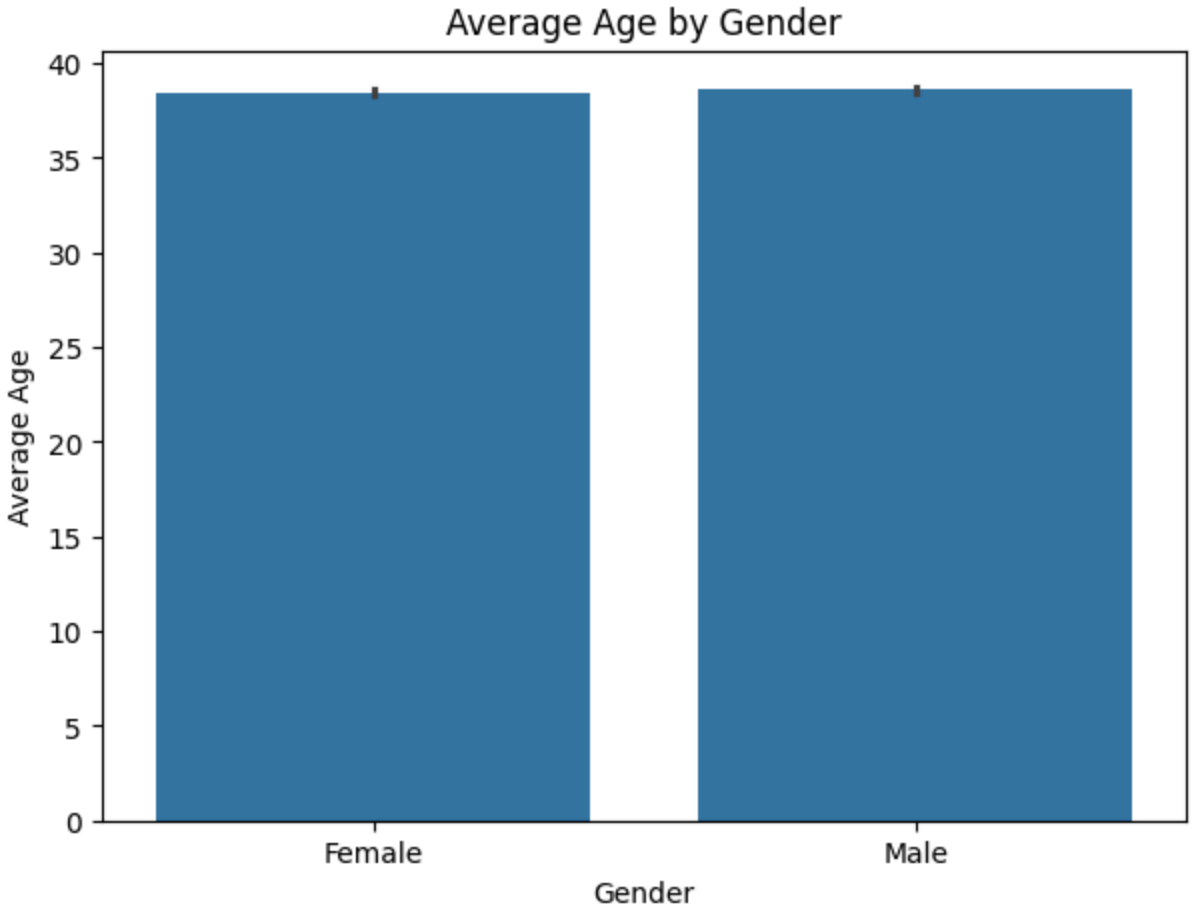
Out[2]:

	User_ID	Age	Gender	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hip_cm
0	1	56	Female	173.1	53.9	91.5	70.6	95.1
1	2	46	Male	162.3	36.8	80.6	69.8	94.6
2	3	32	Female	166.8	69.0	88.8	71.8	84.1
3	4	25	Male	171.3	64.9	85.4	73.2	96.5
4	5	38	Male	167.2	35.8	90.3	66.2	91.0

In [3]:
```
plt.figure(figsize=(10, 5))
sns.histplot(data=df, x='Age', kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(7, 5))
sns.countplot(data=df, x='Gender')
plt.title('Distribution of Gender')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.show()

plt.figure(figsize=(7, 5))
sns.barplot(data=df, x='Gender', y='Age', estimator=np.mean)
plt.title('Average Age by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Age')
plt.show()
```

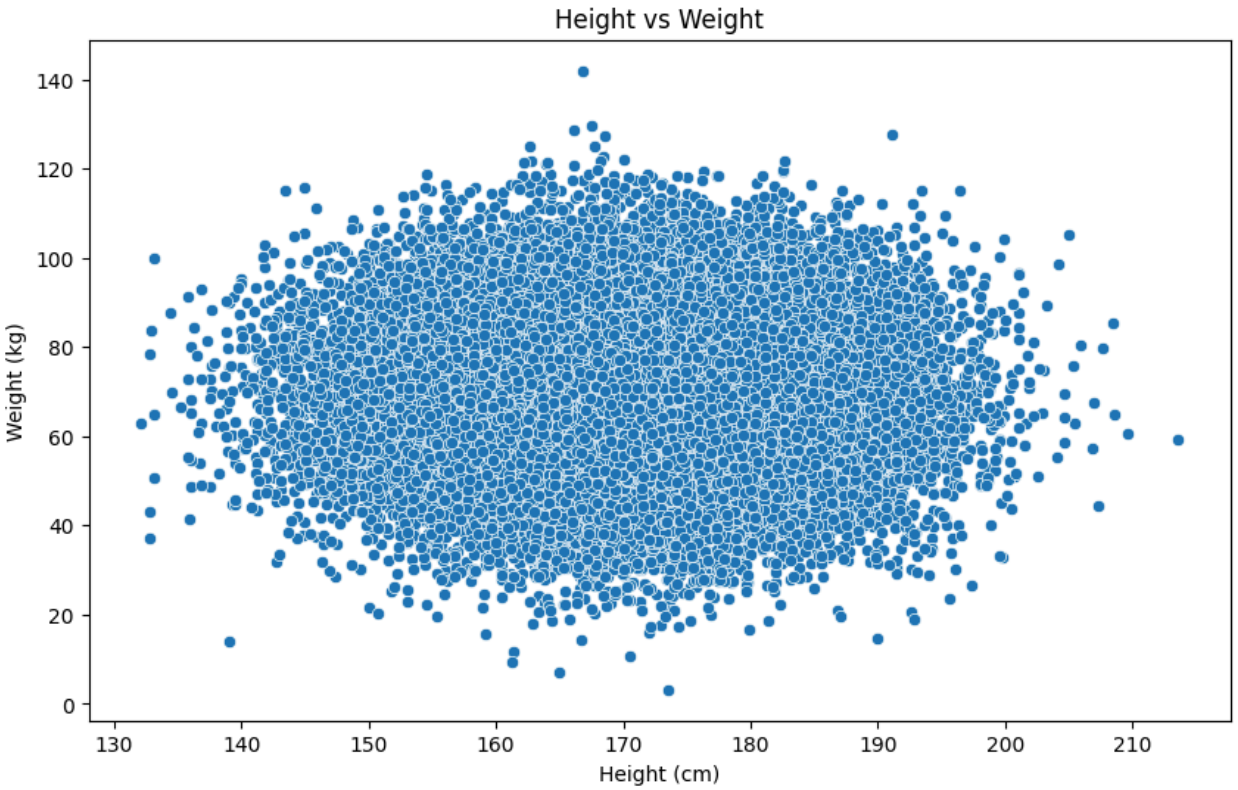

Distribution of Age
Frequency
Age
0
500
1000
1500
2000
2500
20
30
40
50
60

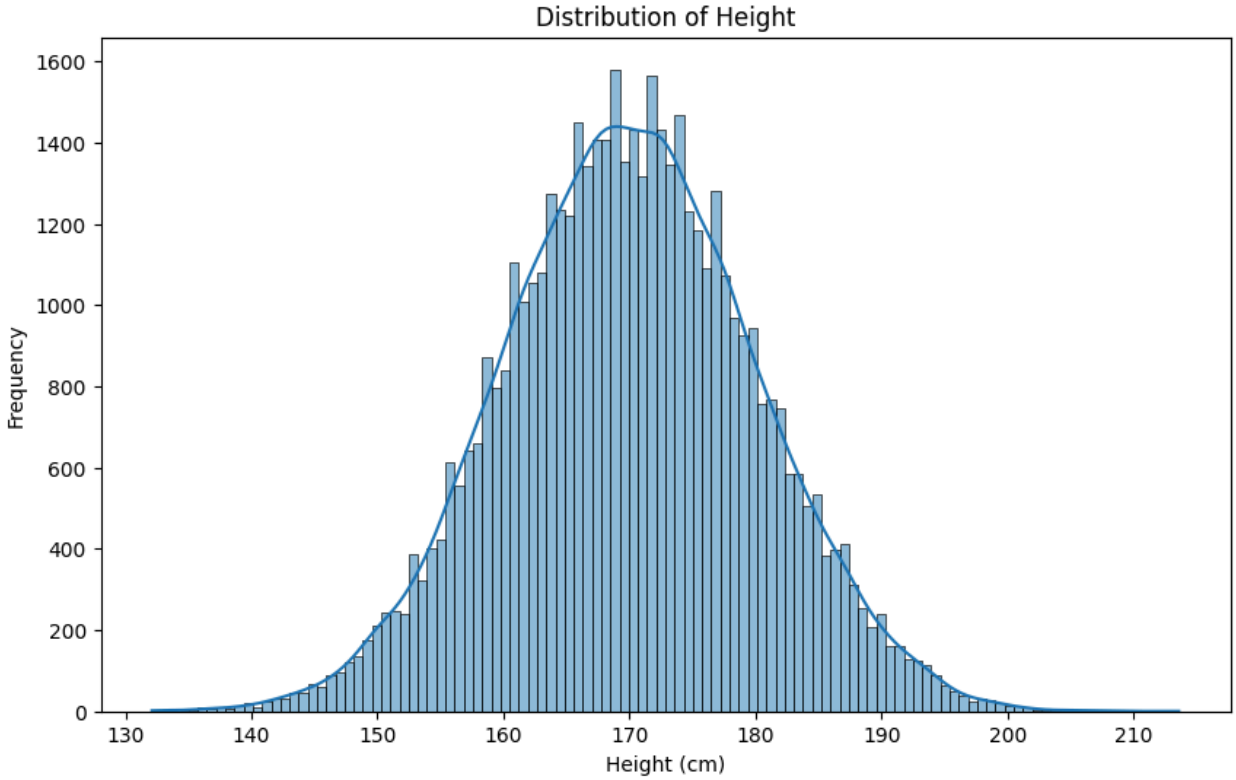

Distribution of Gender
Count
Gender
0
5000
10000
15000
20000
25000
Female
Male

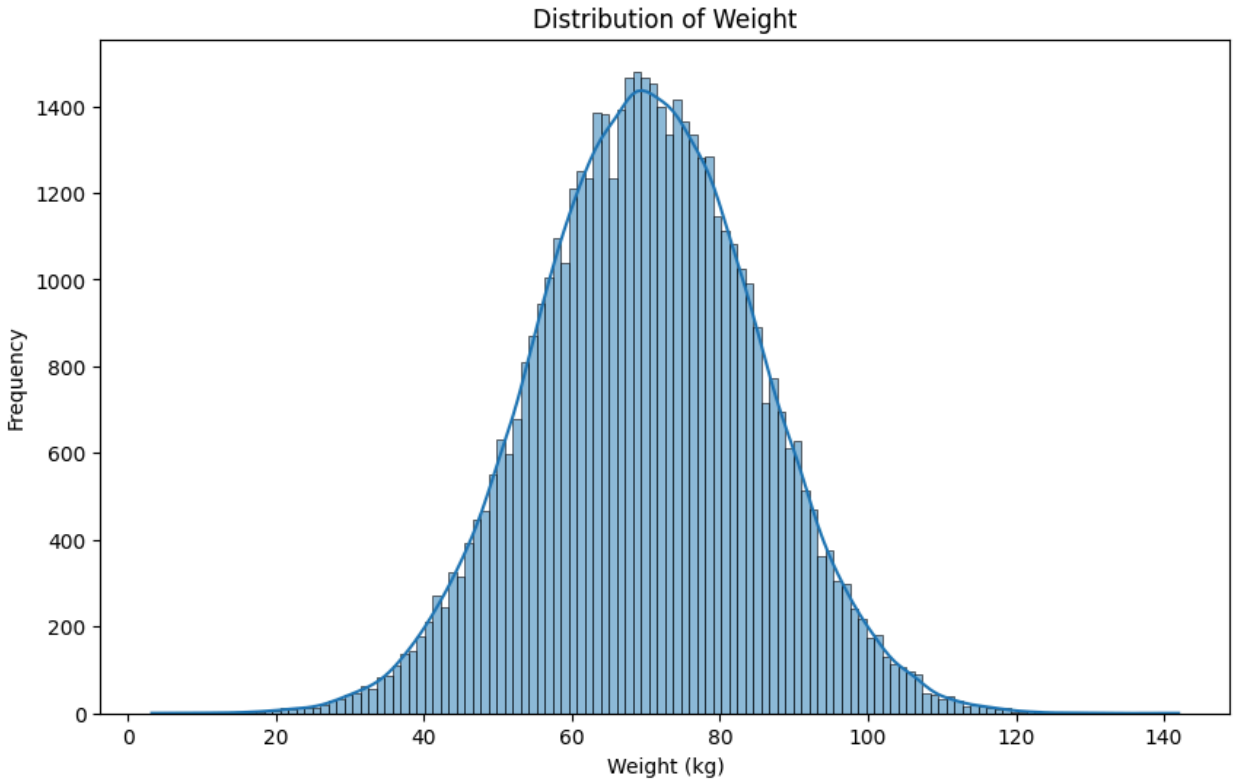


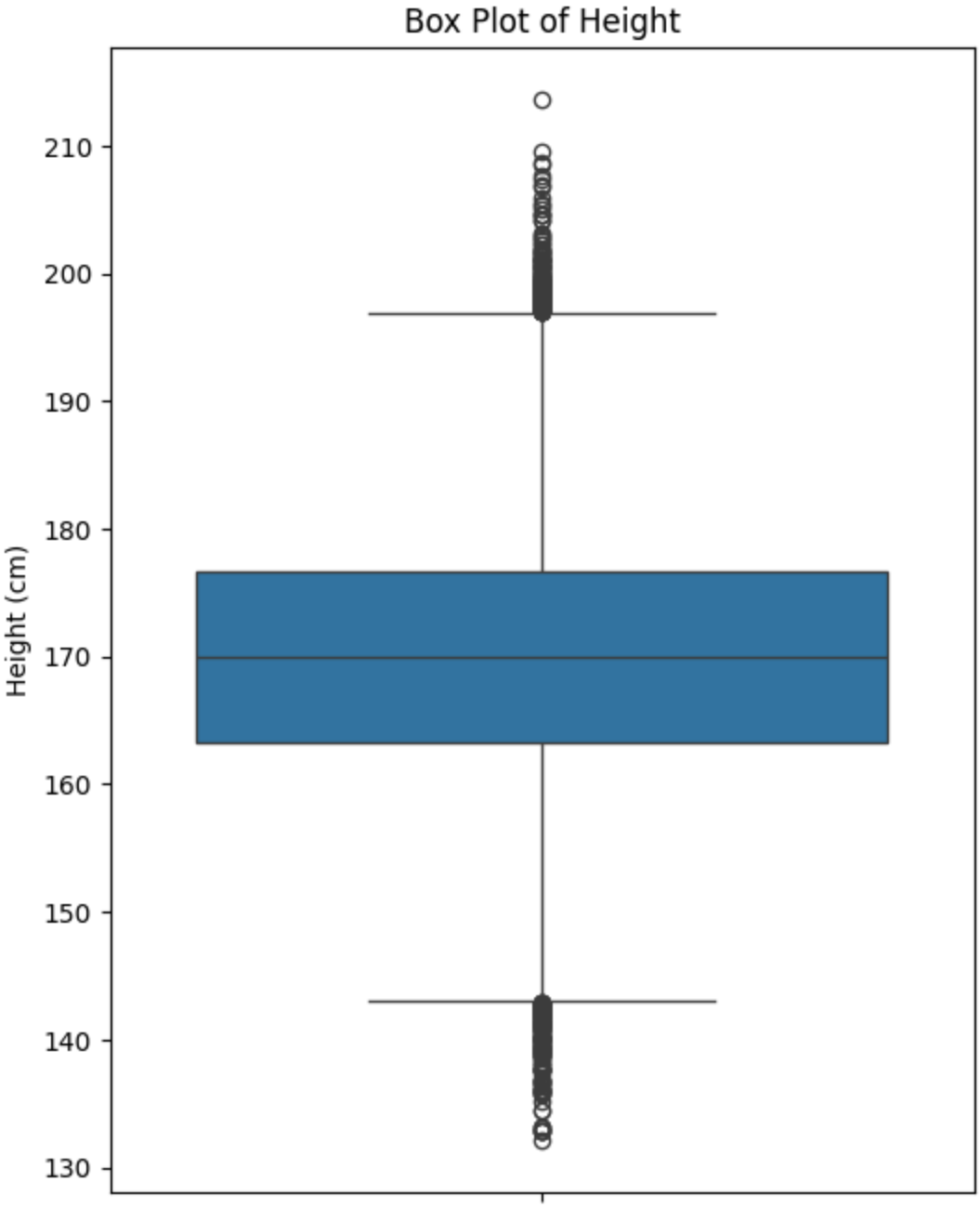
In [4]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Height_cm', y='Weight_kg')
plt.title('Height vs Weight')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Height_cm', kde=True)
plt.title('Distribution of Height')
plt.xlabel('Height (cm)')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Weight_kg', kde=True)
plt.title('Distribution of Weight')
plt.xlabel('Weight (kg)')
plt.ylabel('Frequency')
plt.show()

plt.figure(figsize=(6, 8))
sns.boxplot(data=df, y='Height_cm')
plt.title('Box Plot of Height')
plt.ylabel('Height (cm)')
```

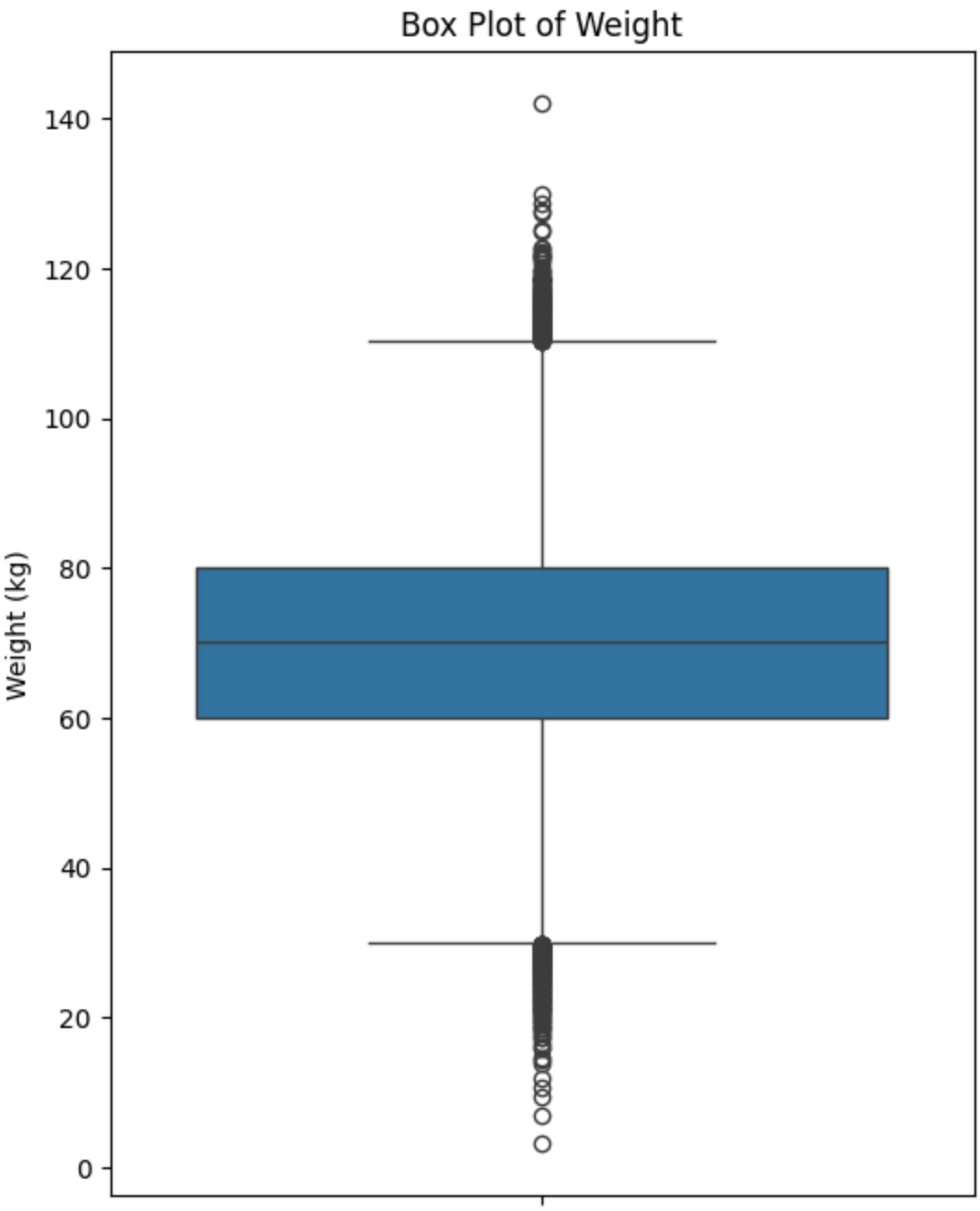
```
plt.show()

plt.figure(figsize=(6, 8))
sns.boxplot(data=df, y='Weight_kg')
plt.title('Box Plot of Weight')
plt.ylabel('Weight (kg)')
plt.show()
```




Distribution of Height
Frequency
Height (cm)
0
200
400
600
800
1000
1200
1400
1600
130
140
150
160
170
180
190
200
210

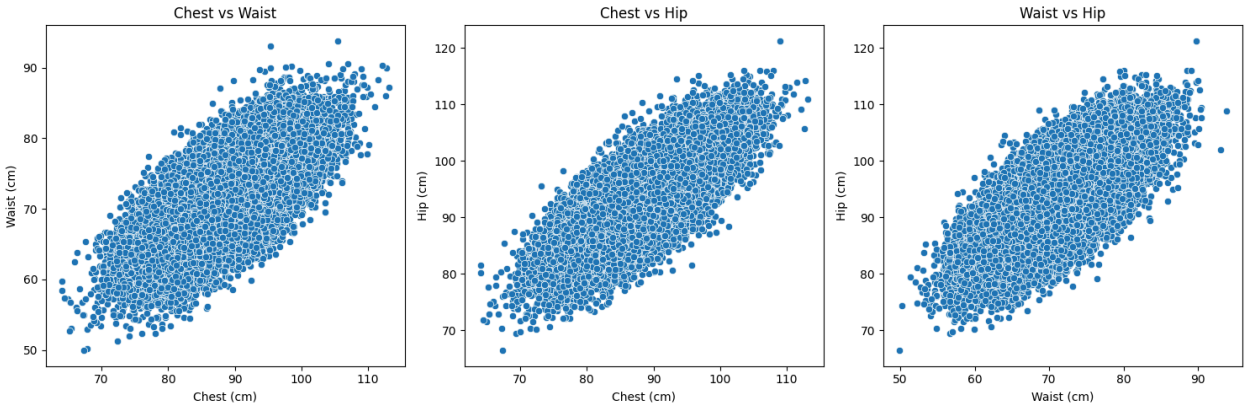

Distribution of Weight
Frequency
Weight (kg)
0
200
400
600
800
1000
1200
1400
0
20
40
60
80
100
120
140


Box Plot of Height
Height (cm)
210
200
190
180
170
160
150
140
130

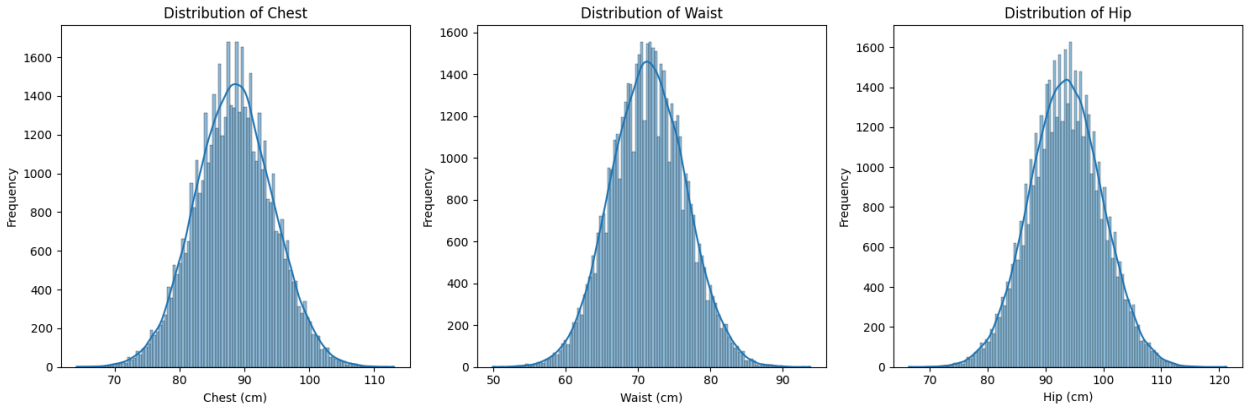


In [5]:
```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.scatterplot(data=df, x='Chest_cm', y='Waist_cm')
plt.title('Chest vs Waist')
plt.xlabel('Chest (cm)')
plt.ylabel('Waist (cm)')

plt.subplot(1, 3, 2)
sns.scatterplot(data=df, x='Chest_cm', y='Hip_cm')
plt.title('Chest vs Hip')
plt.xlabel('Chest (cm)')
plt.ylabel('Hip (cm)')
```

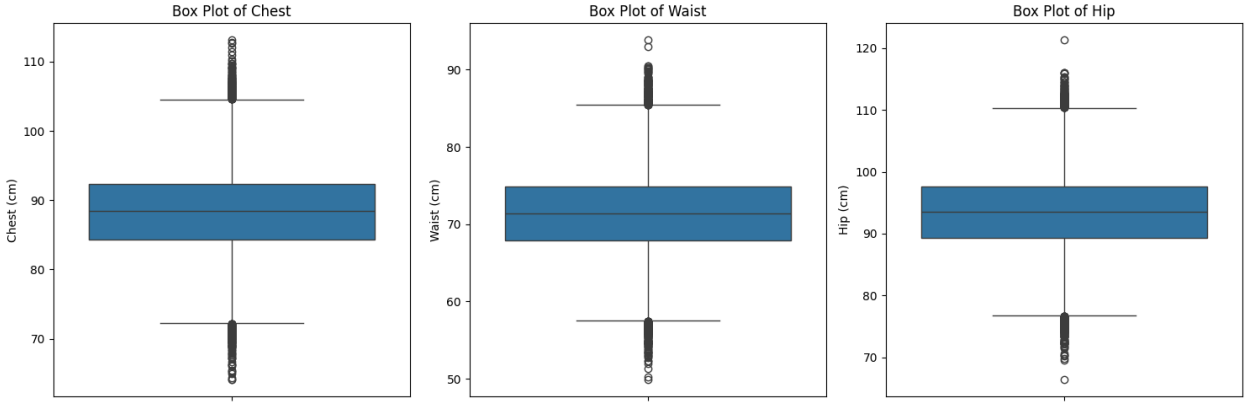
```
plt.subplot(1, 3, 3)
sns.scatterplot(data=df, x='Waist_cm', y='Hip_cm')
plt.title('Waist vs Hip')
plt.xlabel('Waist (cm)')
plt.ylabel('Hip (cm)')

plt.tight_layout()
plt.show()
```



In [6]:
```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.histplot(data=df, x='Chest_cm', kde=True)
plt.title('Distribution of Chest')
plt.xlabel('Chest (cm)')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
sns.histplot(data=df, x='Waist_cm', kde=True)
plt.title('Distribution of Waist')
plt.xlabel('Waist (cm)')
plt.ylabel('Frequency')

plt.subplot(1, 3, 3)
sns.histplot(data=df, x='Hip_cm', kde=True)
plt.title('Distribution of Hip')
plt.xlabel('Hip (cm)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

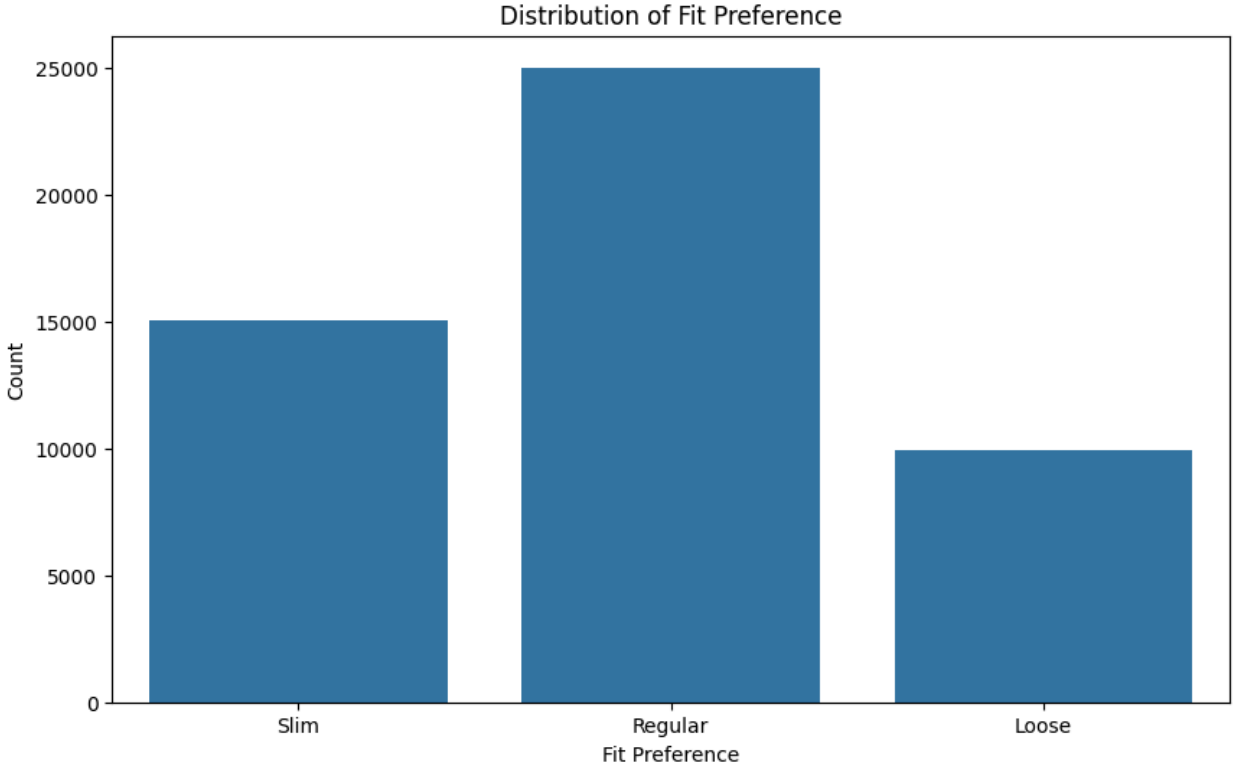


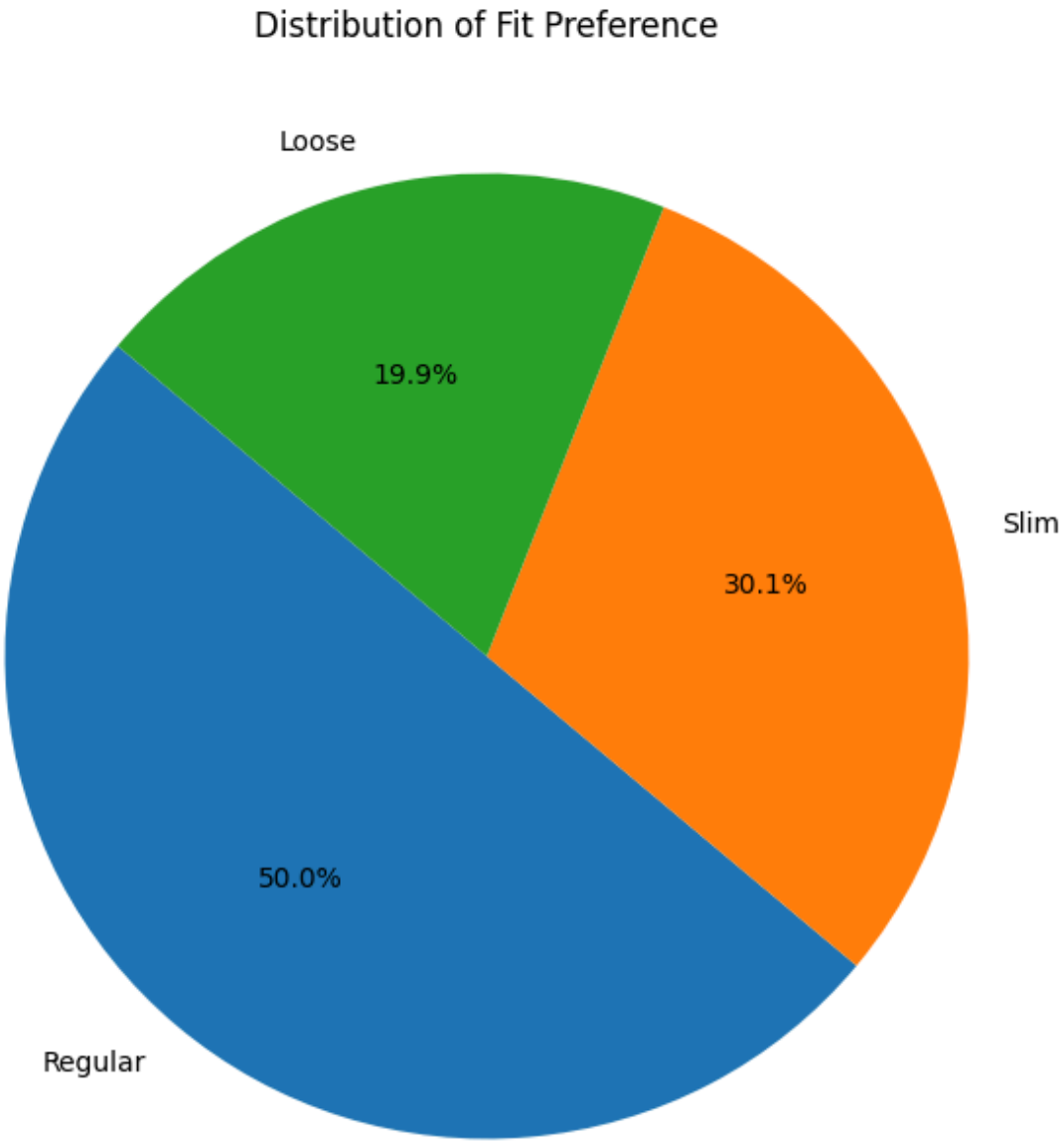
In [7]:
```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.boxplot(data=df, y='Chest_cm')
plt.title('Box Plot of Chest')
plt.ylabel('Chest (cm)')

plt.subplot(1, 3, 2)
sns.boxplot(data=df, y='Waist_cm')
plt.title('Box Plot of Waist')
plt.ylabel('Waist (cm)')

plt.subplot(1, 3, 3)
sns.boxplot(data=df, y='Hip_cm')
plt.title('Box Plot of Hip')
plt.ylabel('Hip (cm)')

plt.tight_layout()
plt.show()
```



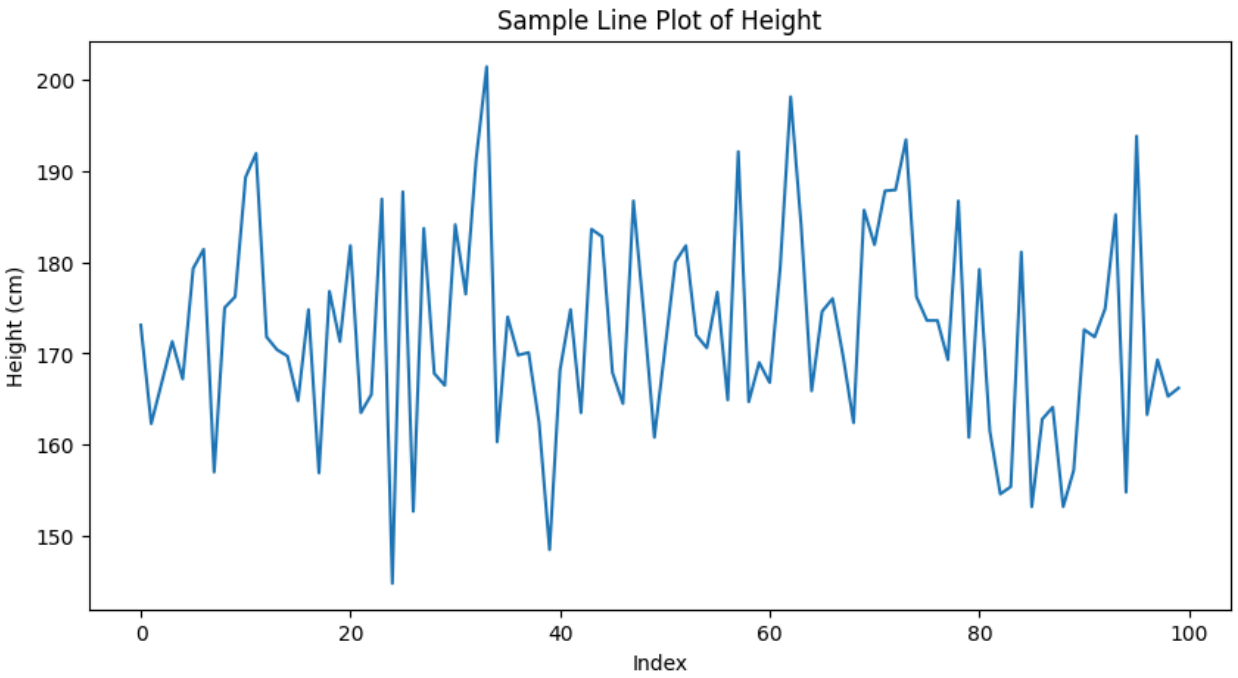
In [8]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='Fit_Preference')
plt.title('Distribution of Fit Preference')
plt.xlabel('Fit Preference')
plt.ylabel('Count')
plt.show()
```

```
fit_preference_counts = df['Fit_Preference'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(fit_preference_counts, labels=fit_preference_counts.index, autopct='%1
plt.title('Distribution of Fit Preference')
plt.show()
```

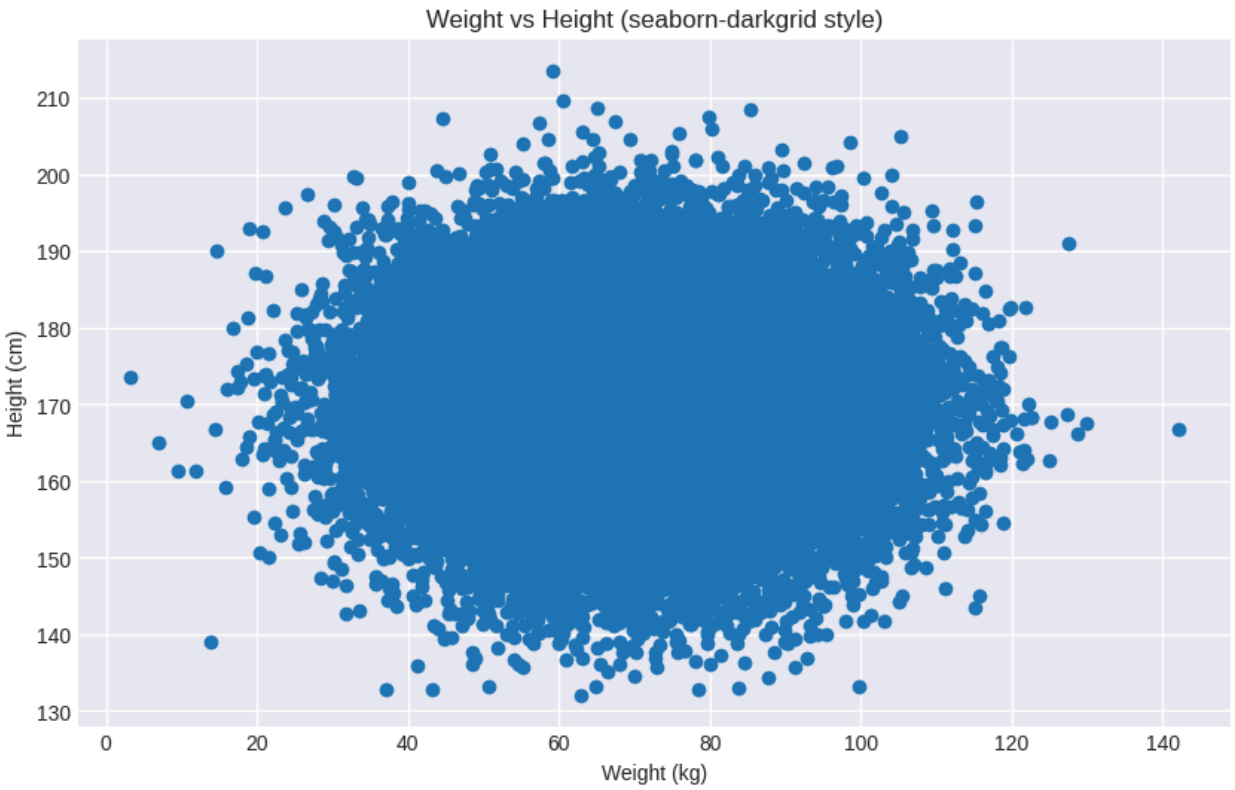




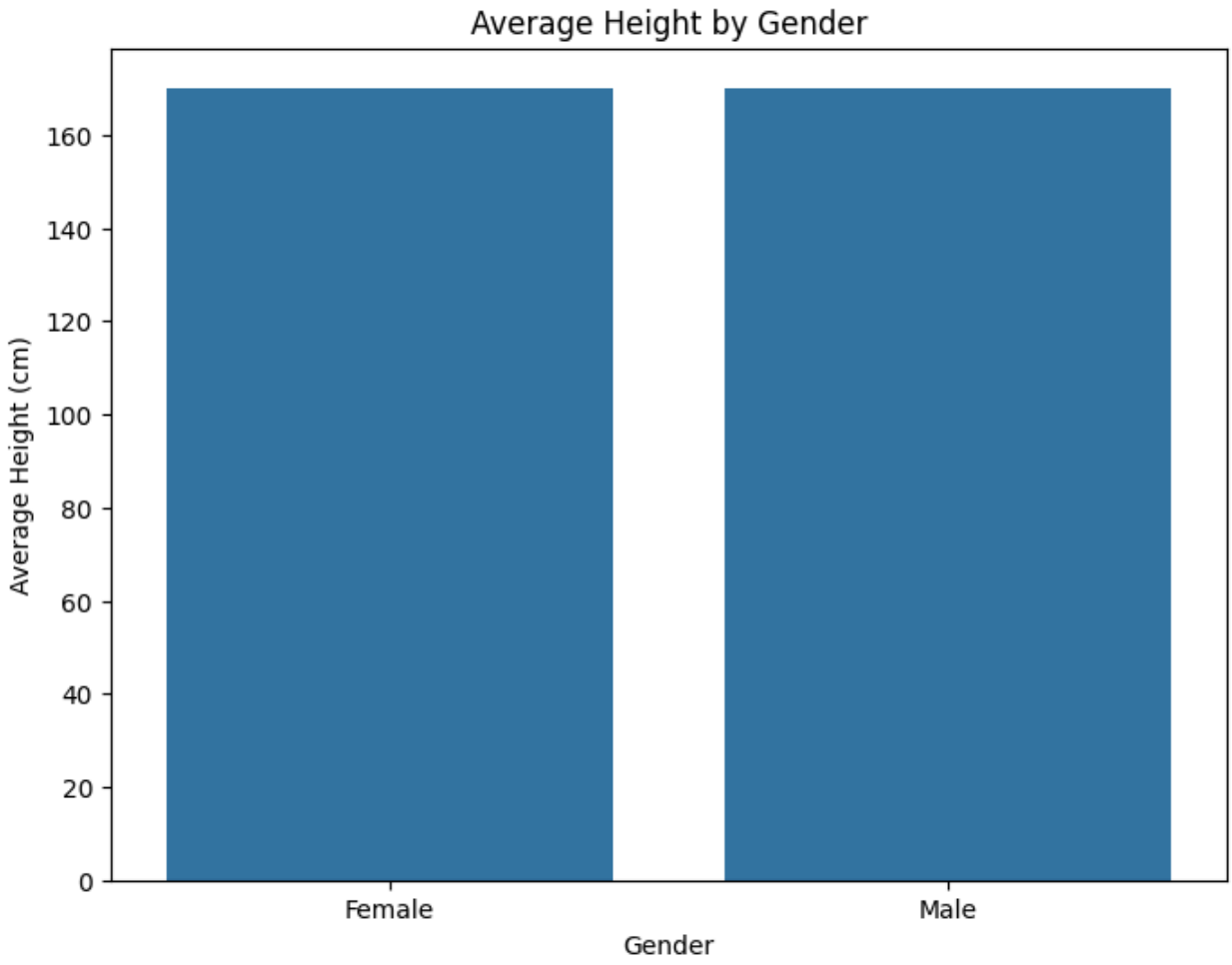
In [11]:
```
# 1. Sample plot (Line plot of a numerical column over its index)
plt.figure(figsize=(10, 5))
df['Height_cm'].head(100).plot(title='Sample Line Plot of Height')
plt.xlabel('Index')
plt.ylabel('Height (cm)')
plt.show()
```



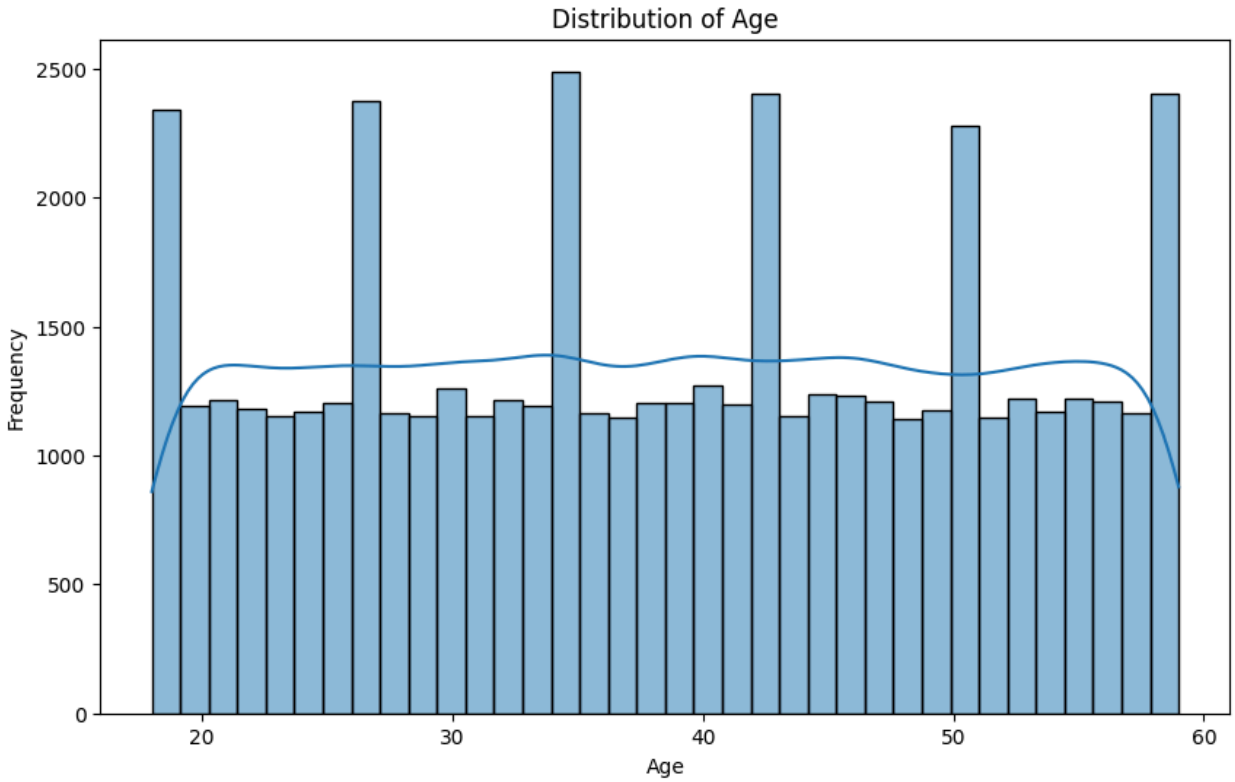
In [12]:
```
# 2. Simple plot with title and style (Scatter plot with custom style)
plt.style.use('seaborn-v0_8-darkgrid')
plt.figure(figsize=(10, 6))
plt.scatter(df['Weight_kg'], df['Height_cm'])
plt.title('Weight vs Height (seaborn-darkgrid style)')
plt.xlabel('Weight (kg)')
plt.ylabel('Height (cm)')
plt.show()
plt.style.use('default') # Reset style
```



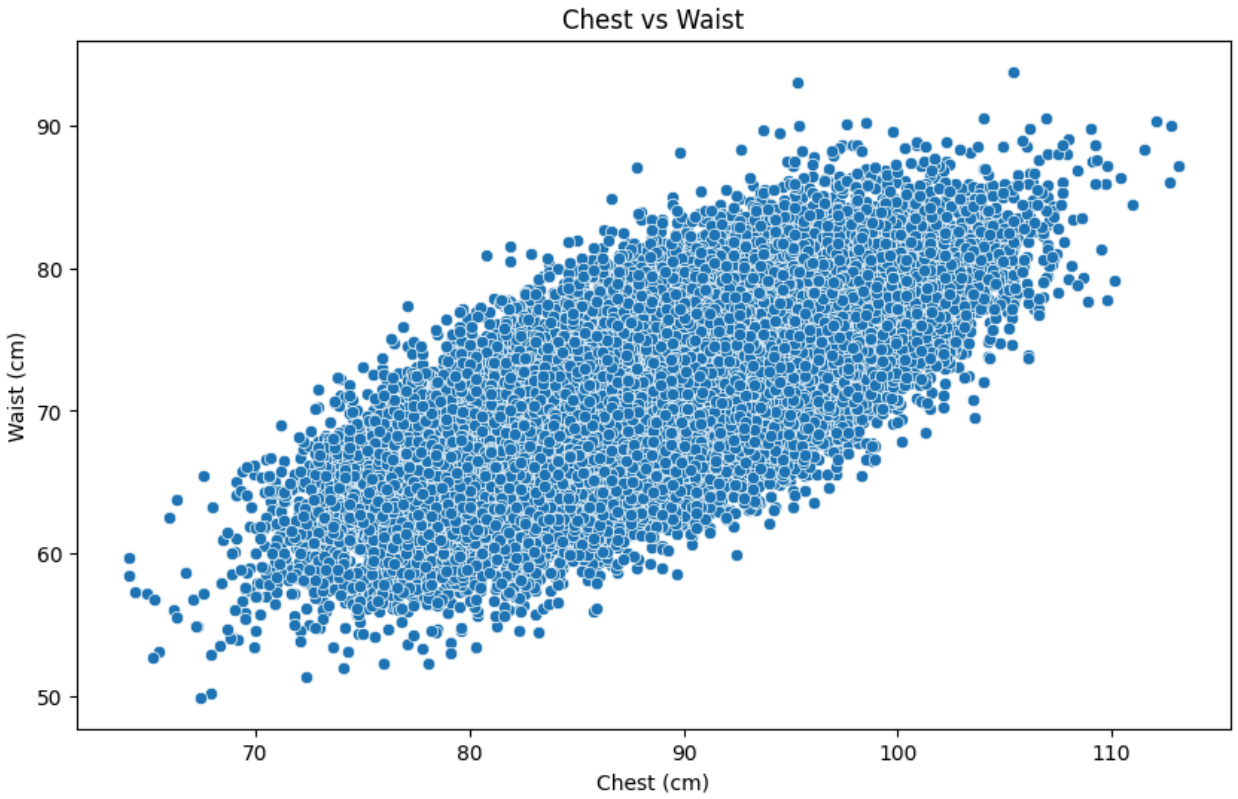
In [13]:
```
# 3. Bar plot (Average Height by Gender)
plt.figure(figsize=(8, 6))
sns.barplot(data=df, x='Gender', y='Height_cm', estimator=np.mean)
plt.title('Average Height by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Height (cm)')
plt.show()
```



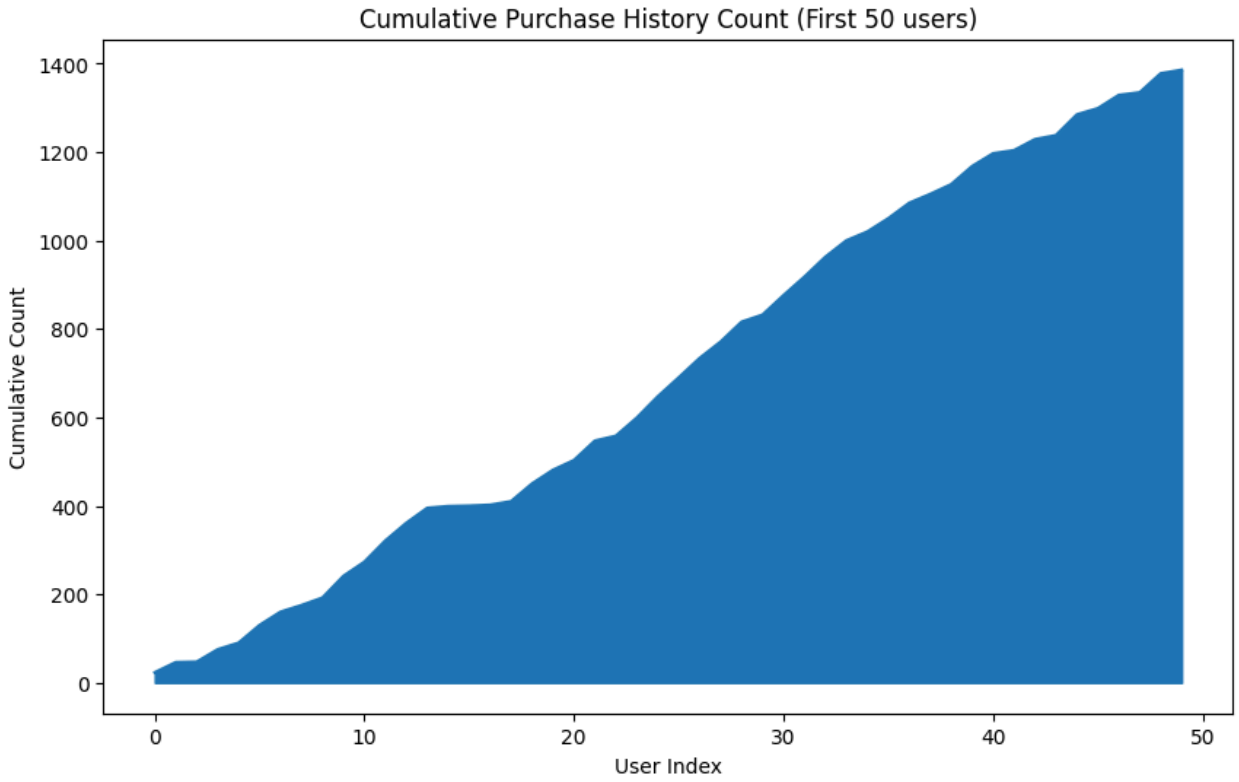
In [14]:
```
# 4. Histogram (Distribution of Age)
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Age', kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```



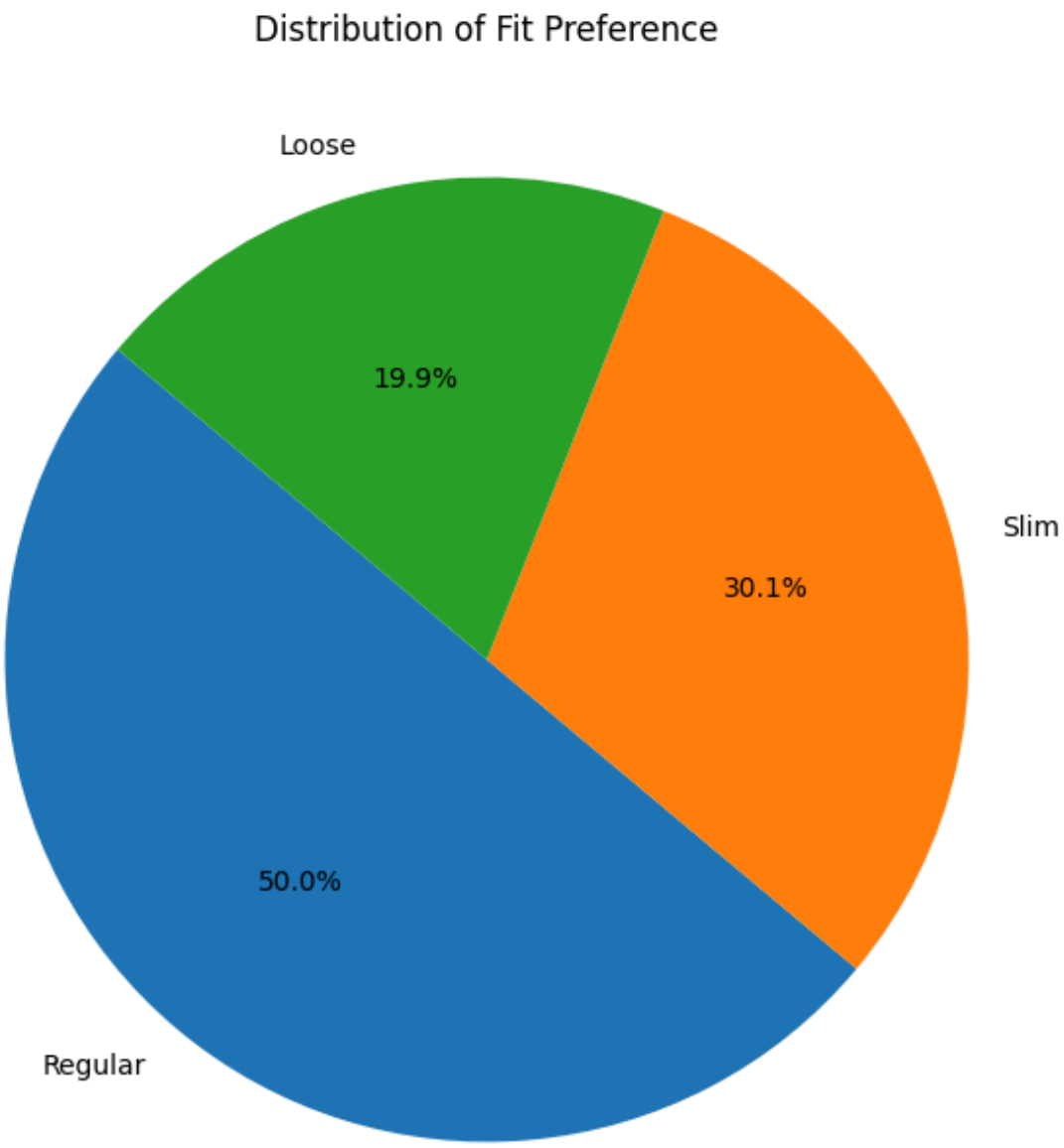
In [15]:
```
# 5. Scatter plot (Chest vs Waist)
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Chest_cm', y='Waist_cm')
plt.title('Chest vs Waist')
plt.xlabel('Chest (cm)')
plt.ylabel('Waist (cm)')
plt.show()
```



In [16]:
```
# 6. Area plot (Cumulative sum of Purchase History Count)
plt.figure(figsize=(10, 6))
df['Purchase_History_Count'].head(50).cumsum().plot(kind='area')
plt.title('Cumulative Purchase History Count (First 50 users)')
plt.xlabel('User Index')
plt.ylabel('Cumulative Count')
plt.show()
```

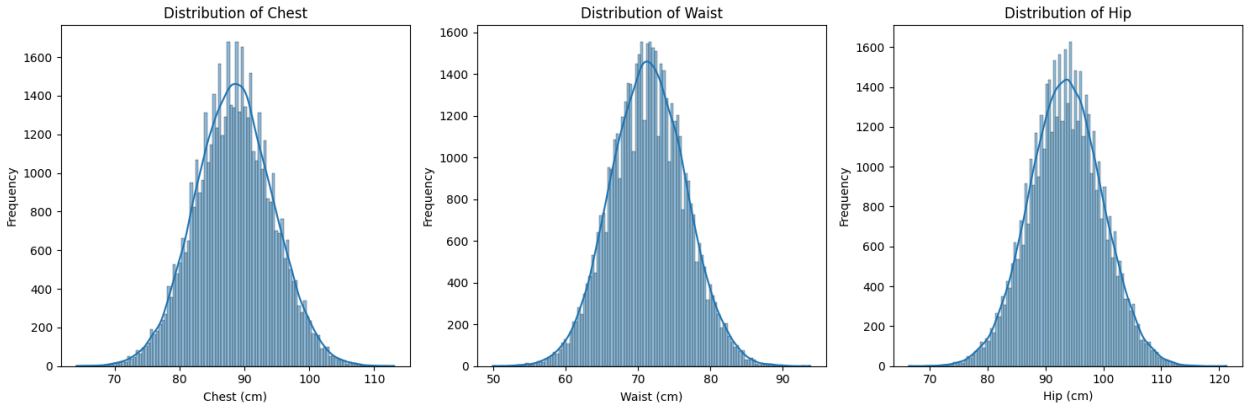


In [17]:
```
# 7. Pie chart (Distribution of Fit Preference)
fit_preference_counts = df['Fit_Preference'].value_counts()
plt.figure(figsize=(8, 8))
plt.pie(fit_preference_counts, labels=fit_preference_counts.index, autopct='%1
plt.title('Distribution of Fit Preference')
plt.show()
```

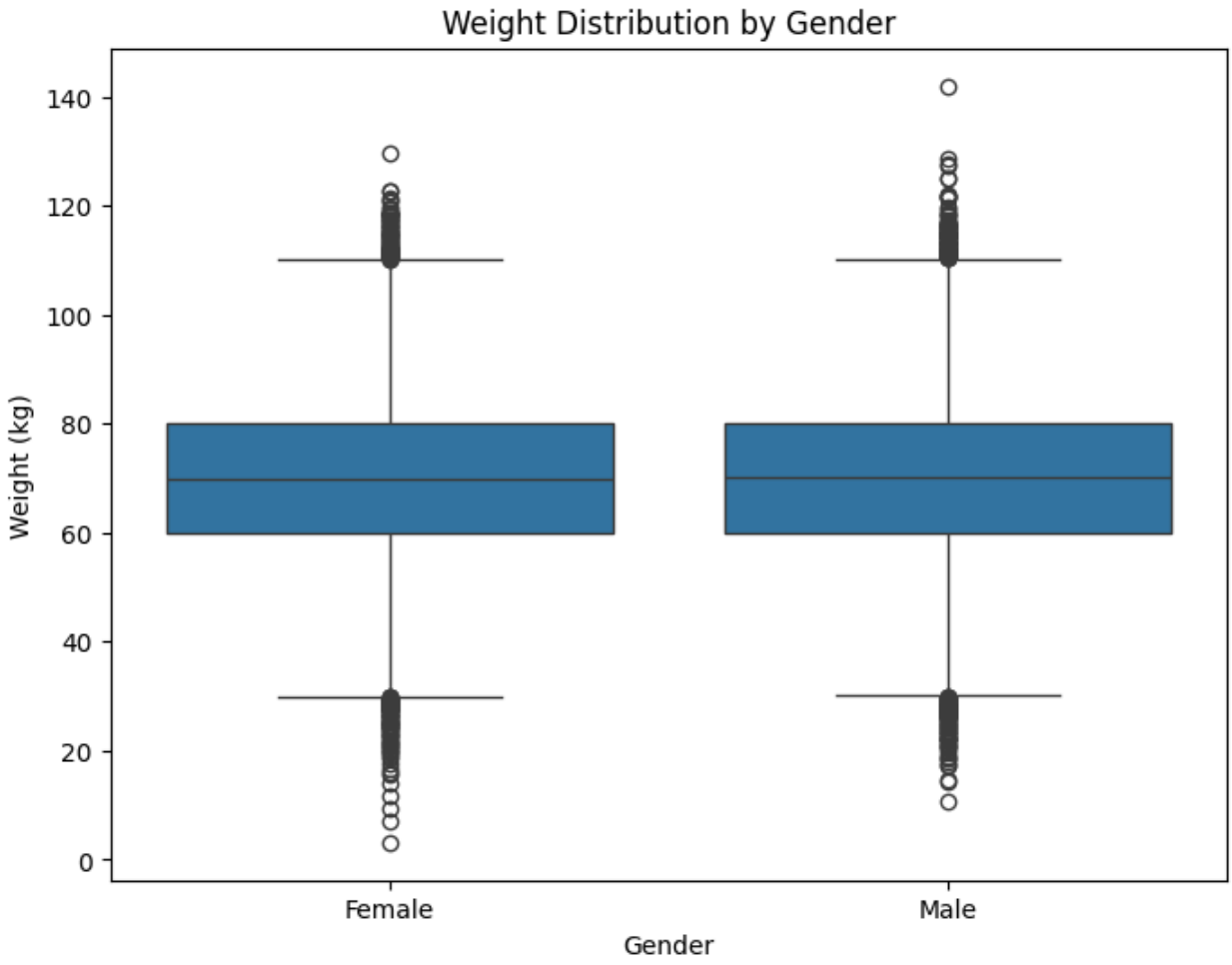


In [18]:
```
# 8. Subplots (Histograms of Chest, Waist, Hip)
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.histplot(data=df, x='Chest_cm', kde=True)
plt.title('Distribution of Chest')
plt.xlabel('Chest (cm)')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
sns.histplot(data=df, x='Waist_cm', kde=True)
plt.title('Distribution of Waist')
plt.xlabel('Waist (cm)')
plt.ylabel('Frequency')
```

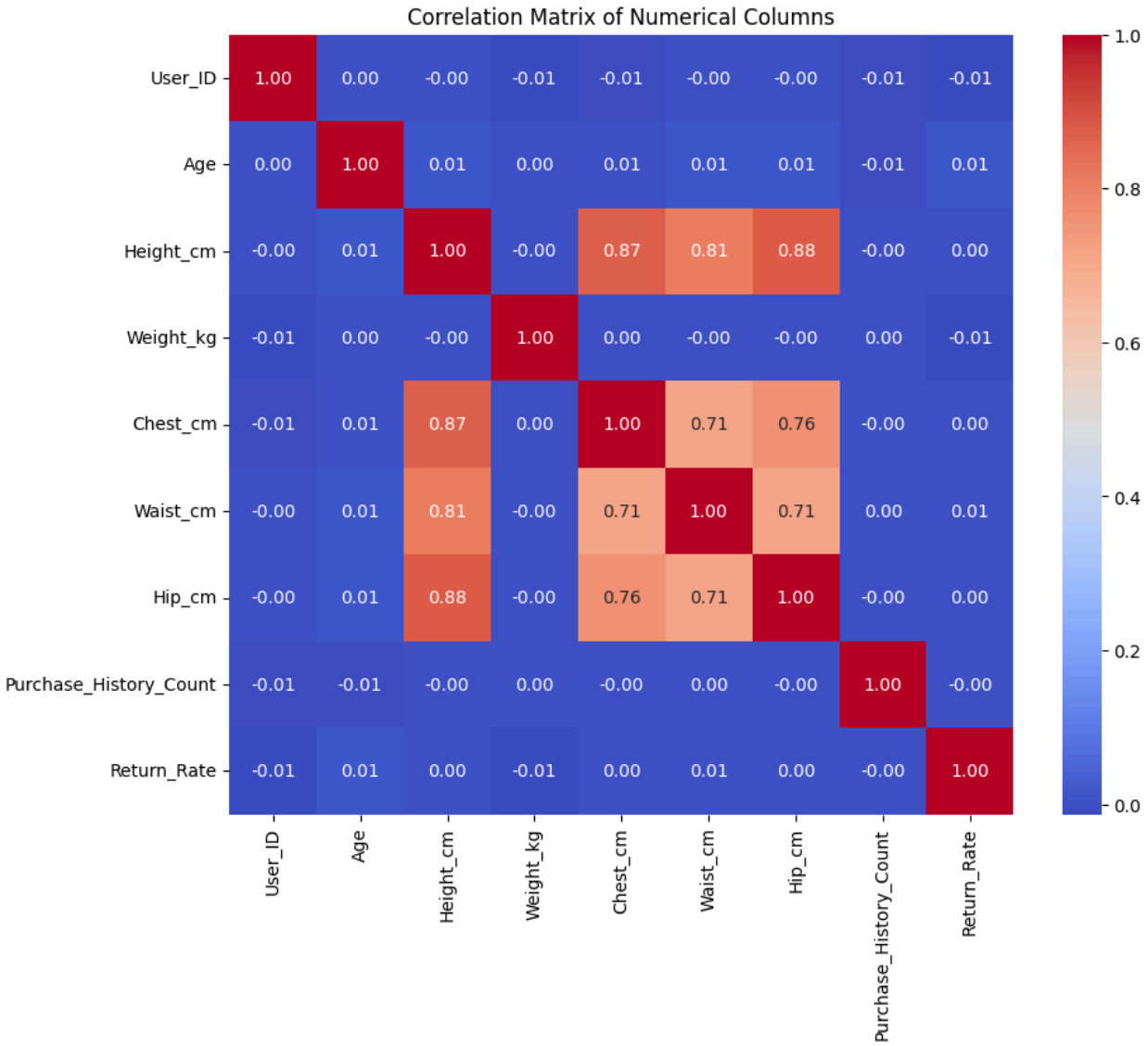
```
plt.subplot(1, 3, 3)
sns.histplot(data=df, x='Hip_cm', kde=True)
plt.title('Distribution of Hip')
plt.xlabel('Hip (cm)')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



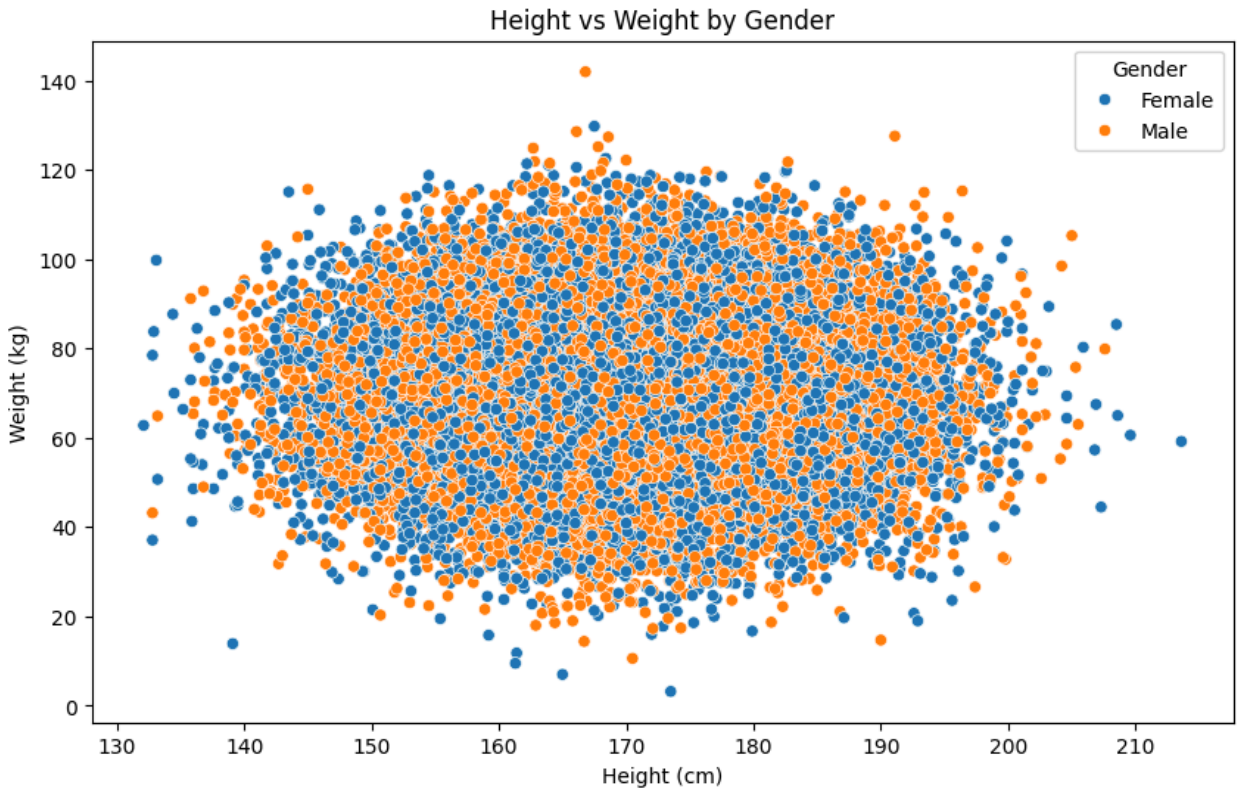
In [19]:
```
# 9. Box plot (Weight by Gender)
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x='Gender', y='Weight_kg')
plt.title('Weight Distribution by Gender')
plt.xlabel('Gender')
plt.ylabel('Weight (kg)')
plt.show()
```



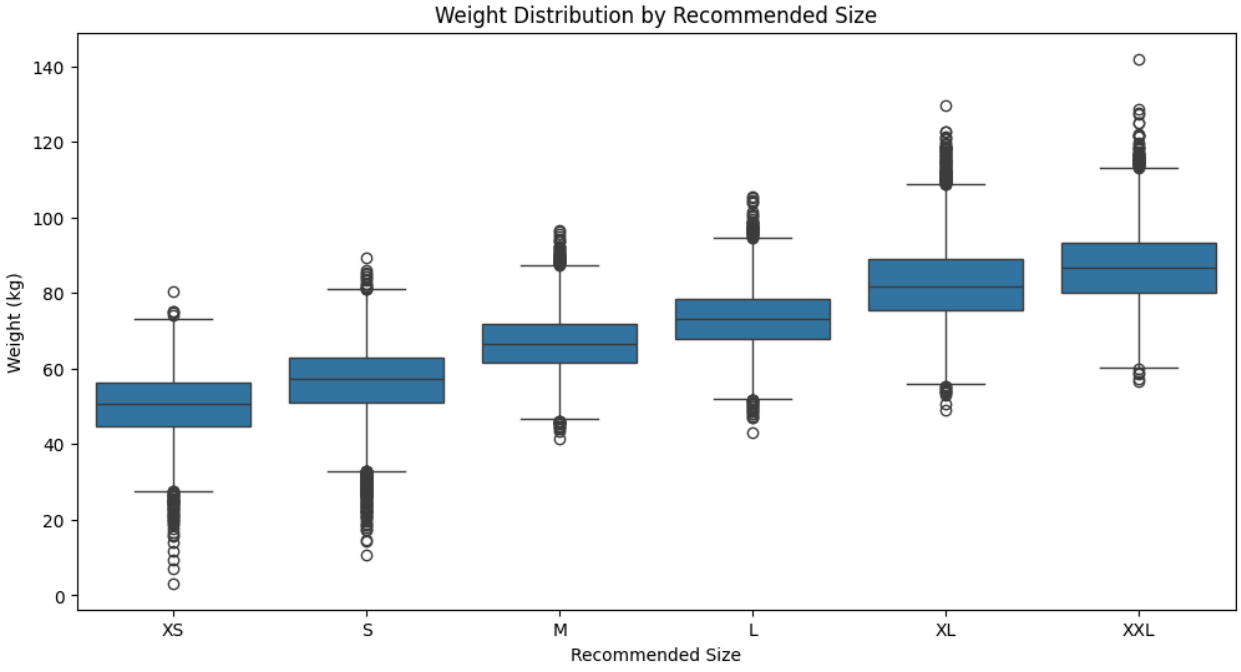
In [20]:
```
# 10. Heatmap (Correlation matrix of numerical columns)
numerical_cols = df.select_dtypes(include=np.number).columns
correlation_matrix = df[numerical_cols].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Columns')
plt.show()
```



In [21]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='Height_cm', y='Weight_kg', hue='Gender')
plt.title('Height vs Weight by Gender')
plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.show()
```

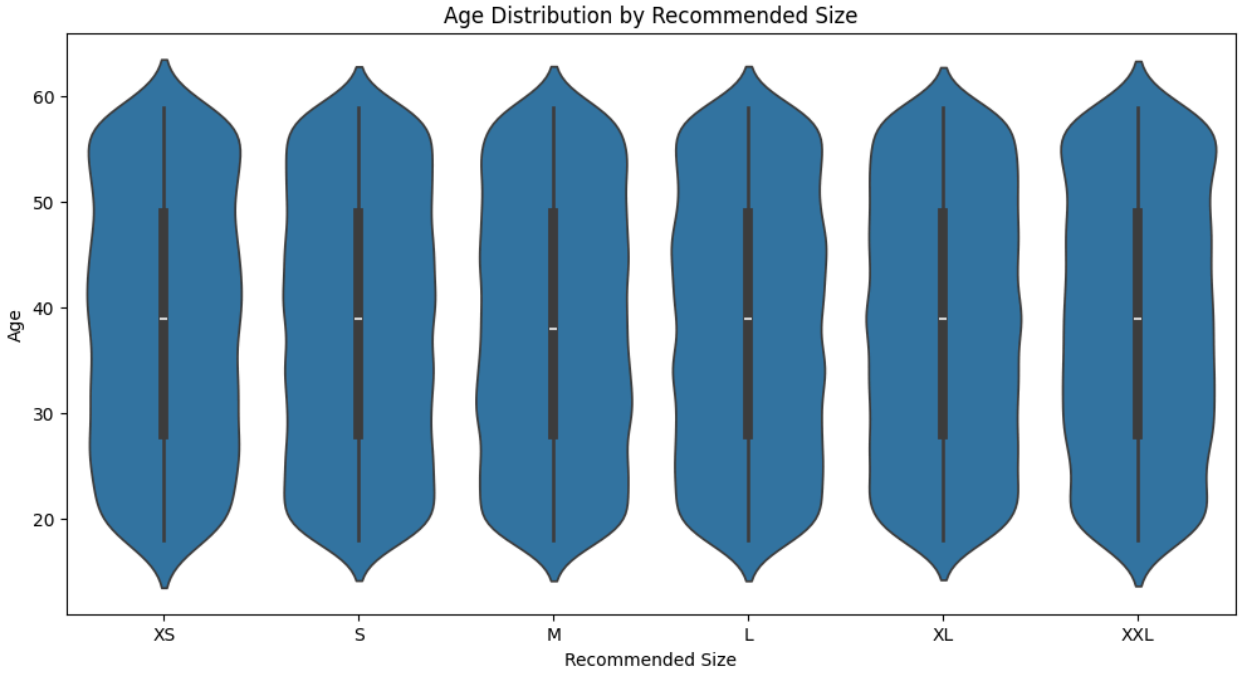


In [22]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x='Recommended_Size', y='Weight_kg', order=['XS', 'S', 'M
plt.title('Weight Distribution by Recommended Size')
plt.xlabel('Recommended Size')
plt.ylabel('Weight (kg)')
plt.show()
```

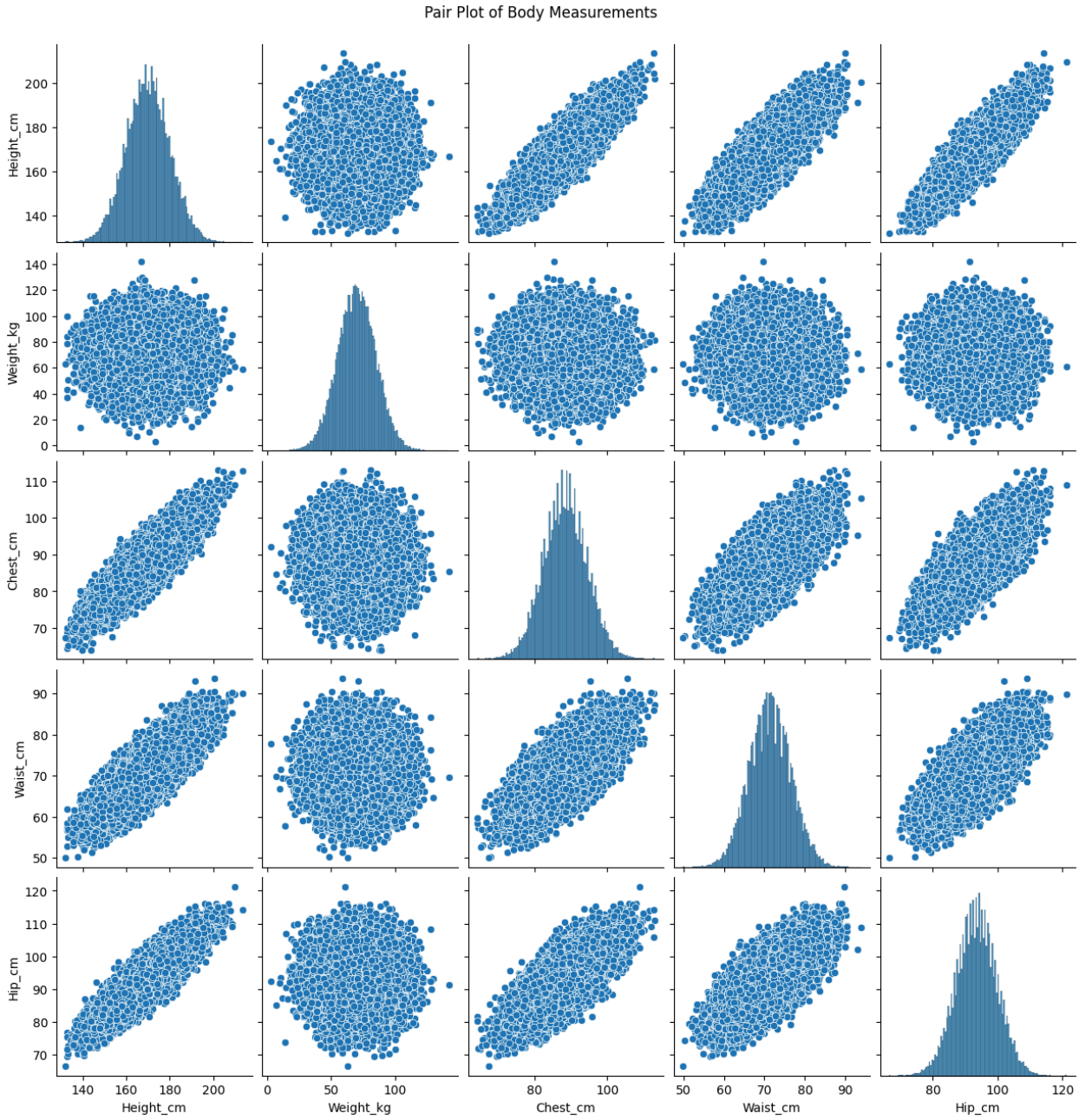


In [23]:
```
plt.figure(figsize=(12, 6))
```

```
sns.violinplot(data=df, x='Recommended_Size', y='Age', order=['XS', 'S', 'M',
plt.title('Age Distribution by Recommended Size')
plt.xlabel('Recommended Size')
plt.ylabel('Age')
plt.show()
```



In [24]:
```
numerical_cols = ['Height_cm', 'Weight_kg', 'Chest_cm', 'Waist_cm', 'Hip_cm']
sns.pairplot(df[numerical_cols])
plt.suptitle('Pair Plot of Body Measurements', y=1.02)
plt.show()
```



In [25]:
```
import pandas as pd

df = pd.read_excel("/content/SmartSizeAI_ClothingSizeDataset_.xlsx")
df.head()
```

Out[25]:

	User_ID	Age	Gender	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hip_cm
0	1	56	Female	173.1	53.9	91.5	70.6	95.1
1	2	46	Male	162.3	36.8	80.6	69.8	94.6
2	3	32	Female	166.8	69.0	88.8	71.8	84.1
3	4	25	Male	171.3	64.9	85.4	73.2	96.5
4	5	38	Male	167.2	35.8	90.3	66.2	91.0

In [26]:
```
# Check for missing values in the dataset
print(df.isnull().sum())
```

```
User_ID                   0
Age                       0
Gender                    0
Height_cm                 0
Weight_kg                 0
Chest_cm                  0
Waist_cm                  0
Hip_cm                    0
Brand                     0
Fit_Preference            0
Purchase_History_Count    0
Return_Rate               0
Recommended_Size          0
dtype: int64
```

In [27]:
```
# One-hot encode categorical columns (Gender and Fit_Preference)
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)
encoded_features = encoder.fit_transform(df[['Gender', 'Fit_Preference']])
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_
display(encoded_df.head())
```

	Gender_Female	Gender_Male	Fit_Preference_Loose	Fit_Preference_Regular	Fi
0	1.0	0.0	0.0	0.0	
1	0.0	1.0	0.0	0.0	
2	1.0	0.0	0.0	1.0	
3	0.0	1.0	0.0	1.0	
4	0.0	1.0	0.0	1.0	

In [28]:
```
# Normalize numeric features using StandardScaler (excluding ID and count colu
from sklearn.preprocessing import StandardScaler
numeric_cols = df.select_dtypes(include=['number']).columns.tolist()
cols_to_exclude = ['User_ID', 'Purchase_History_Count', 'Return_Rate']
cols_to_scale = [col for col in numeric_cols if col not in cols_to_exclude]
```

```
scaler = StandardScaler()
scaled_numeric_features = scaler.fit_transform(df[cols_to_scale])
scaled_numeric_df = pd.DataFrame(scaled_numeric_features, columns=cols_to_scal
display(scaled_numeric_df.head())
```

	Age	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hip_cm
0	1.445821	0.309618	-1.085653	0.515269	-0.154937	0.257933
1	0.619022	-0.769673	-2.231203	-1.303103	-0.309889	0.178012
2	-0.538498	-0.319969	-0.074085	0.064847	0.077492	-1.500344
3	-1.117258	0.129736	-0.348749	-0.502352	0.348659	0.481714
4	-0.042418	-0.279995	-2.298195	0.315082	-1.007176	-0.397425

In [29]:
```
# Combine scaled numerical features and one-hot encoded categorical features
processed_df = pd.concat([scaled_numeric_df, encoded_df], axis=1)
display(processed_df.head())
```

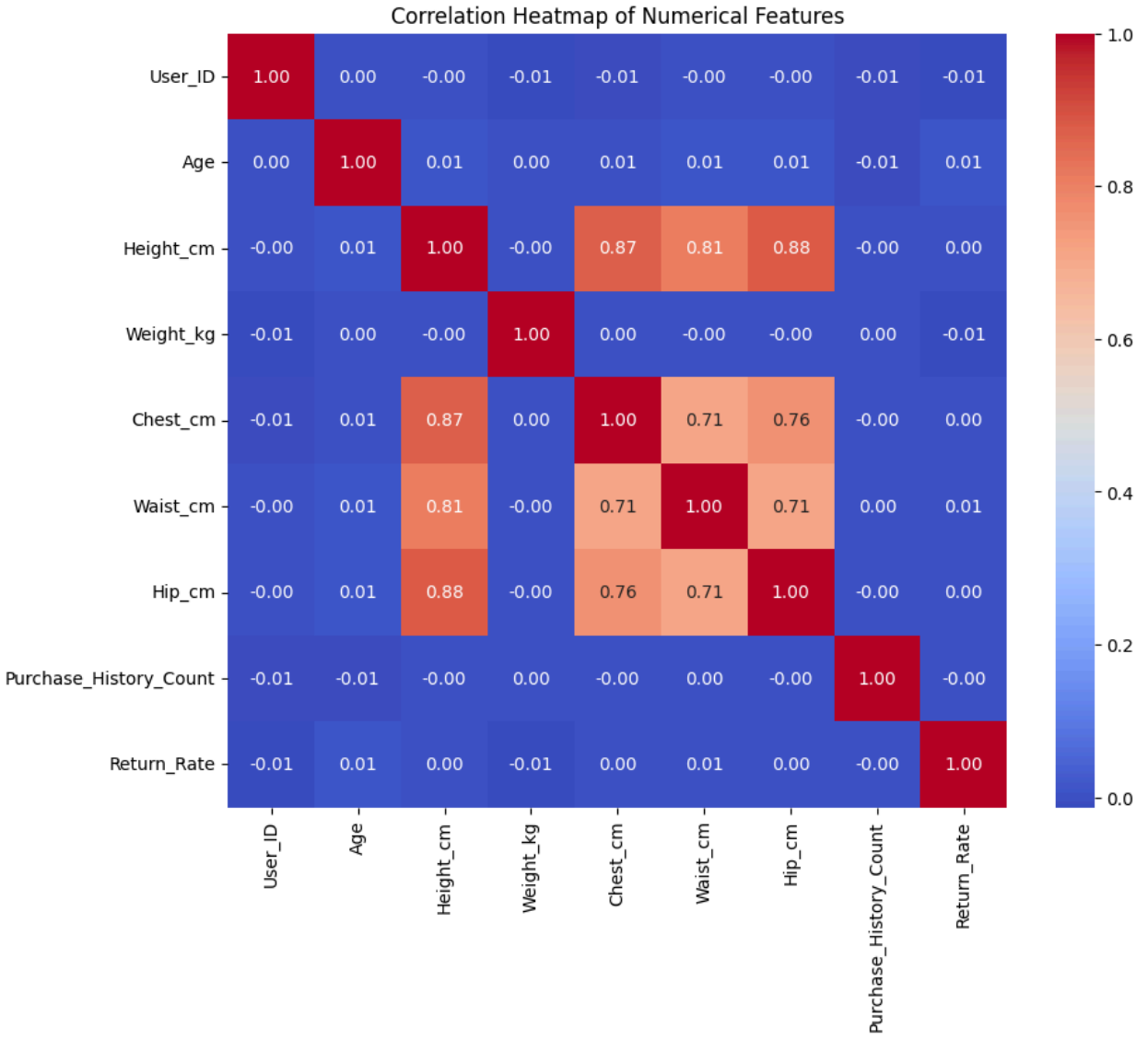
	Age	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hip_cm	Gender_Fema
0	1.445821	0.309618	-1.085653	0.515269	-0.154937	0.257933	1
1	0.619022	-0.769673	-2.231203	-1.303103	-0.309889	0.178012	0
2	-0.538498	-0.319969	-0.074085	0.064847	0.077492	-1.500344	1
3	-1.117258	0.129736	-0.348749	-0.502352	0.348659	0.481714	0
4	-0.042418	-0.279995	-2.298195	0.315082	-1.007176	-0.397425	0

In [30]:
```
# Select only numerical columns from the original DataFrame
numerical_df = df.select_dtypes(include=['number'])
display(numerical_df.head())
```

	User_ID	Age	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hip_cm	Purchase_H
0	1	56	173.1	53.9	91.5	70.6	95.1	
1	2	46	162.3	36.8	80.6	69.8	94.6	
2	3	32	166.8	69.0	88.8	71.8	84.1	
3	4	25	171.3	64.9	85.4	73.2	96.5	
4	5	38	167.2	35.8	90.3	66.2	91.0	

In [31]:
```
# Calculate correlation matrix and create heatmap visualization
import seaborn as sns
import matplotlib.pyplot as plt
```

```
correlation_matrix = numerical_df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



In [32]:
```
# Group by Recommended_Size and calculate mean of numerical features to analyz
average_numerical_features = df.groupby('Recommended_Size')[cols_to_scale].mea
display(average_numerical_features)
```

Recommended_Size	Age	Height_cm	Weight_kg	Chest_cm	Waist_cm	Hi
L	38.544858	169.749271	73.406578	88.286542	71.282723	93.3
M	38.292473	172.055803	66.938979	89.494906	72.281768	94.6
S	38.439202	174.695459	56.564946	90.804394	73.368466	96.0
XL	38.509024	165.121357	82.575578	85.896644	69.331058	90.8
XS	38.782276	176.085560	50.123684	91.560976	73.979105	96.8
XXL	38.772644	162.643736	87.055606	84.600822	68.318042	89.4

In [33]:
```
# Split the processed data into training and testing sets
from sklearn.model_selection import train_test_split

X = processed_df
y = df['Recommended_Size']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randc

print("Training set shape (features):", X_train.shape)
print("Testing set shape (features):", X_test.shape)
print("Training set shape (target):", y_train.shape)
print("Testing set shape (target):", y_test.shape)
```

```
Training set shape (features): (35000, 11)
Testing set shape (features): (15000, 11)
Training set shape (target): (35000,)
Testing set shape (target): (15000,)
```

In [ ]:
```
```

In [34]:
```
# Train Logistic Regression
from sklearn.linear_model import LogisticRegression

print("Training Logistic Regression...")
log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train, y_train)
print("Logistic Regression trained.")
```

```
Training Logistic Regression...
Logistic Regression trained.
```

In [35]:
```
# Train K-Nearest Neighbors (KNN)
from sklearn.neighbors import KNeighborsClassifier

print("Training KNN...")
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print("KNN trained.")
```

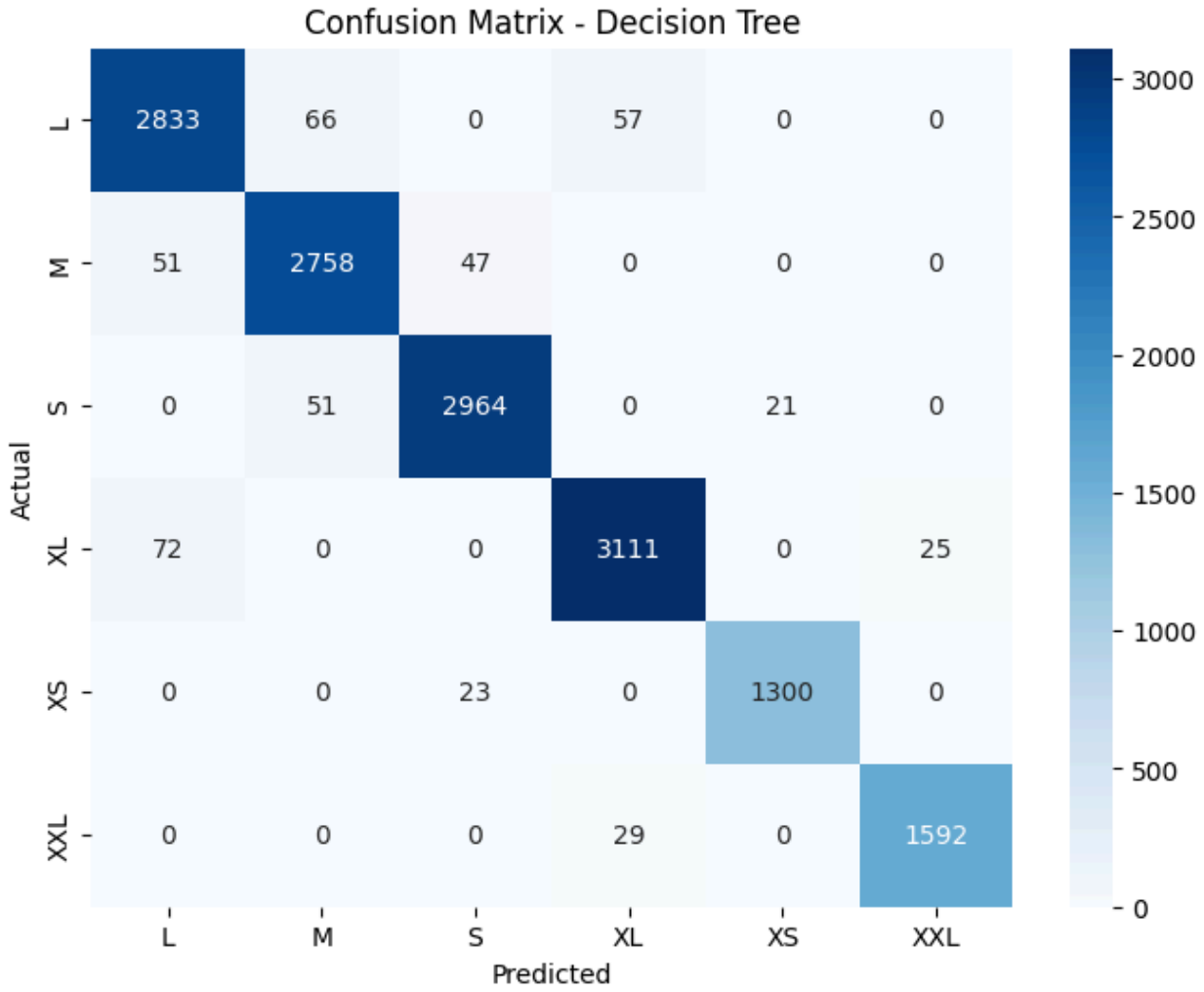
```
Training KNN...
KNN trained.
```

In [36]:

```python
# Train Decision Tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

print("Training Decision Tree...")
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
print("Decision Tree trained.")

# Evaluate Decision Tree
y_pred_dt = decision_tree.predict(X_test)

print("\n--- Evaluation for Decision Tree ---")
print("Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Precision:", precision_score(y_test, y_pred_dt, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_dt, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_dt, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_dt))

# Confusion Matrix as heatmap
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues', xticklabels=decision_tre
plt.title('Confusion Matrix - Decision Tree')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

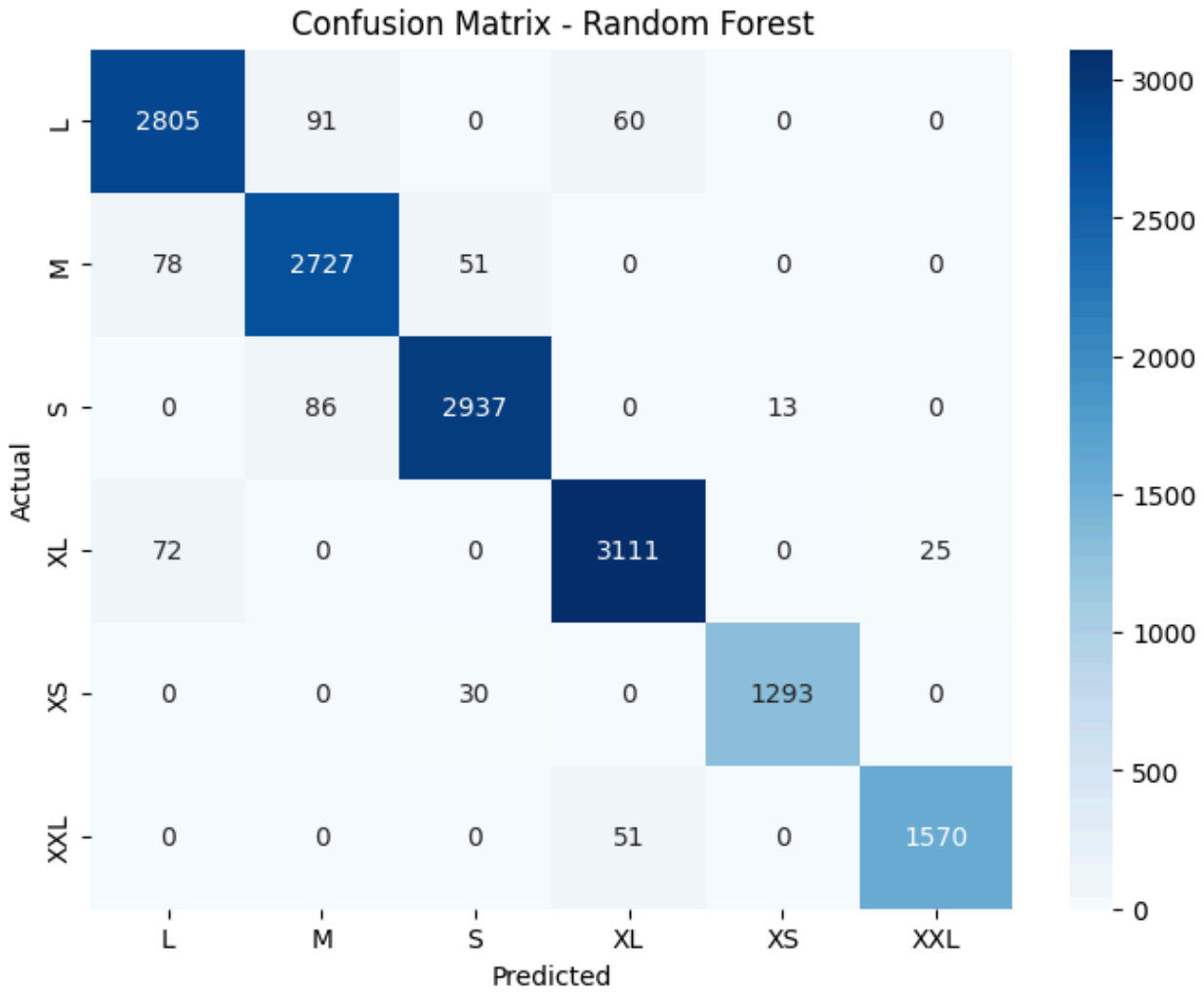
```
Training Decision Tree...
Decision Tree trained.

--- Evaluation for Decision Tree ---
Accuracy: 0.9705333333333334
Precision: 0.9705558356175832
Recall: 0.9705333333333334
F1-score: 0.9705417242048755

Classification Report:
```

	precision	recall	f1-score	support
L	0.96	0.96	0.96	2956
M	0.96	0.97	0.96	2856
S	0.98	0.98	0.98	3036
XL	0.97	0.97	0.97	3208
XS	0.98	0.98	0.98	1323
XXL	0.98	0.98	0.98	1621
accuracy			0.97	15000
macro avg	0.97	0.97	0.97	15000
weighted avg	0.97	0.97	0.97	15000



In [37]:
```
# Train Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

print("Training Random Forest...")
random_forest = RandomForestClassifier()
random_forest.fit(X_train, y_train)
print("Random Forest trained.")

# Evaluate Random Forest
y_pred_rf = random_forest.predict(X_test)

print("\n--- Evaluation for Random Forest ---")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_rf, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_rf, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))

# Confusion Matrix as heatmap
cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues', xticklabels=random_fores
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
Training Random Forest...
Random Forest trained.

--- Evaluation for Random Forest ---
Accuracy: 0.9628666666666666
Precision: 0.9630198215764171
Recall: 0.9628666666666666
F1-score: 0.9629174763808872

Classification Report:
               precision    recall  f1-score   support

           L       0.95      0.95      0.95      2956
           M       0.94      0.95      0.95      2856
           S       0.97      0.97      0.97      3036
          XL       0.97      0.97      0.97      3208
          XS       0.99      0.98      0.98      1323
         XXL       0.98      0.97      0.98      1621

    accuracy                           0.96     15000
   macro avg       0.97      0.96      0.97     15000
weighted avg       0.96      0.96      0.96     15000
```



In [38]:
```python
# Train Naive Bayes
from sklearn.naive_bayes import GaussianNB

print("Training Naive Bayes...")
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
print("Naive Bayes trained.")
```

```
Training Naive Bayes...
Naive Bayes trained.
```

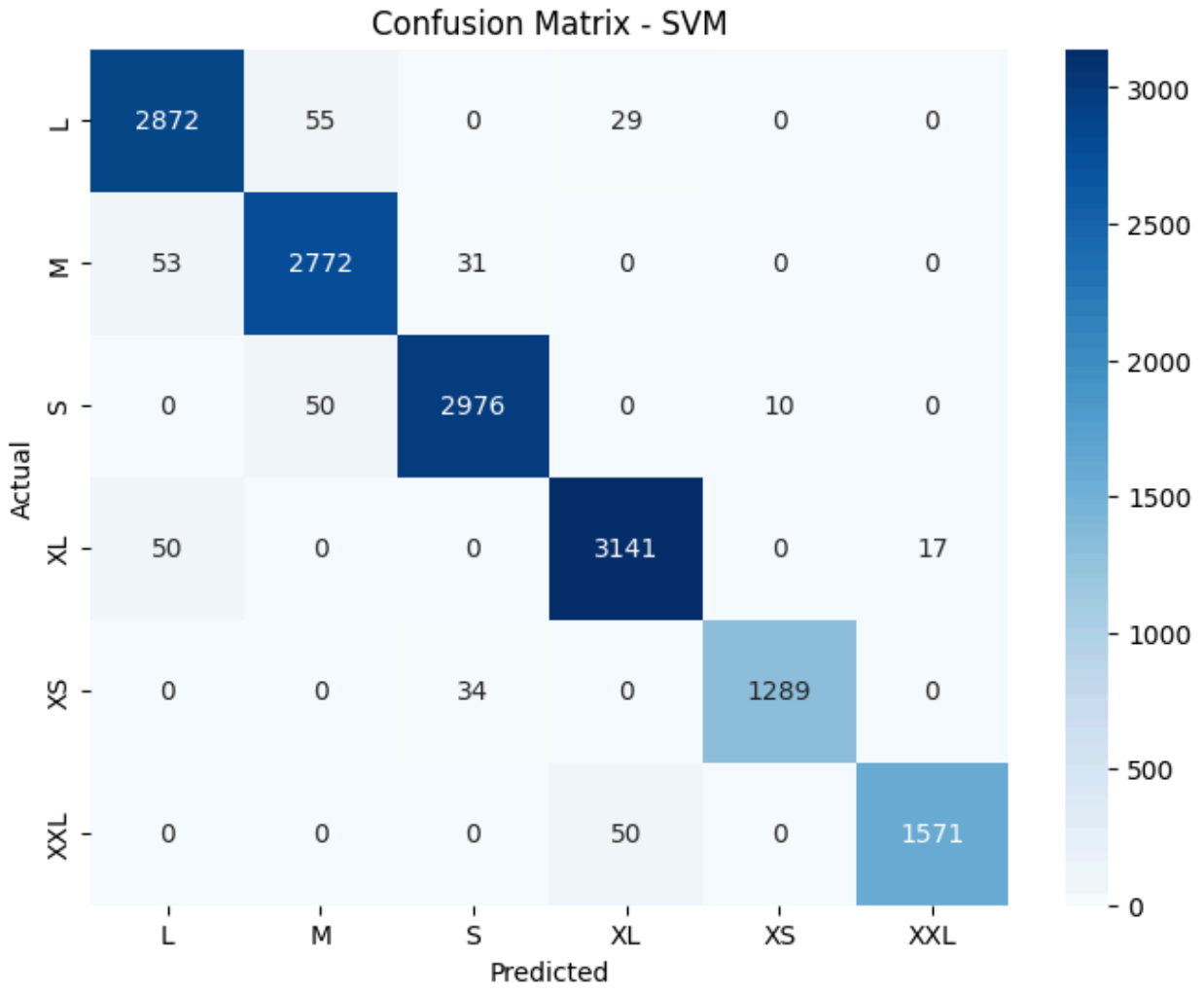
In [39]:
```python
# Train AdaBoost
from sklearn.ensemble import AdaBoostClassifier

print("Training AdaBoost...")
adaboost = AdaBoostClassifier()
adaboost.fit(X_train, y_train)
print("AdaBoost trained.")
```

```
Training AdaBoost...
AdaBoost trained.
```

In [40]:
```python
# Train SVM
```

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

print("Training SVM...")
svm = SVC()
svm.fit(X_train, y_train)
print("SVM trained.")

# Evaluate SVM
y_pred_svm = svm.predict(X_test)

print("\n--- Evaluation for SVM ---")
print("Accuracy:", accuracy_score(y_test, y_pred_svm))
print("Precision:", precision_score(y_test, y_pred_svm, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_svm, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_svm, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_svm))

# Confusion Matrix as heatmap
cm_svm = confusion_matrix(y_test, y_pred_svm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=svm.classes
plt.title('Confusion Matrix - SVM')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

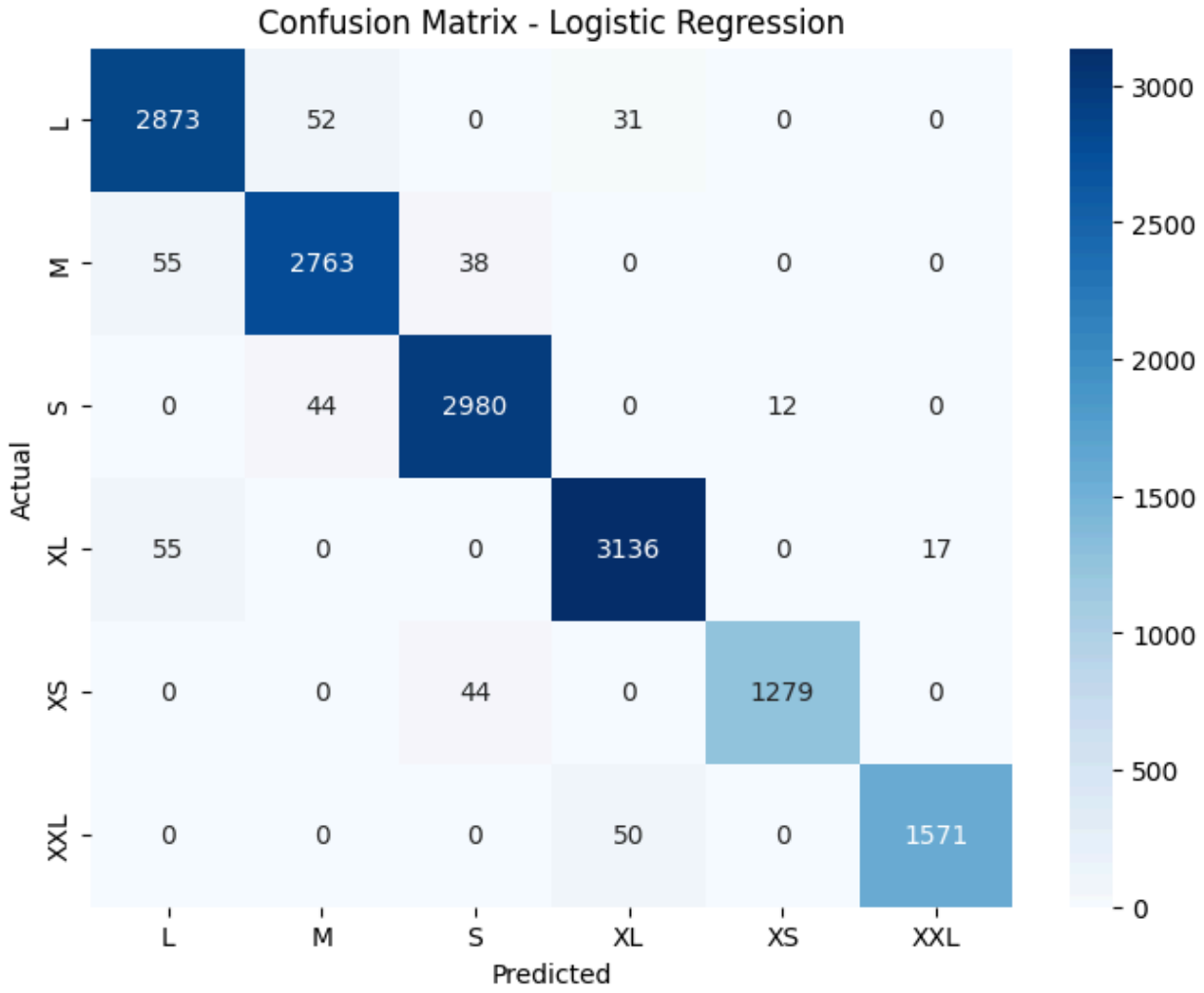
```
Training SVM...
SVM trained.

--- Evaluation for SVM ---
Accuracy: 0.9747333333333333
Precision: 0.9748191659397244
Recall: 0.9747333333333333
F1-score: 0.974752500739235

Classification Report:
               precision    recall  f1-score   support

           L       0.97      0.97      0.97      2956
           M       0.96      0.97      0.97      2856
           S       0.98      0.98      0.98      3036
          XL       0.98      0.98      0.98      3208
          XS       0.99      0.97      0.98      1323
         XXL       0.99      0.97      0.98      1621

    accuracy                           0.97     15000
   macro avg       0.98      0.97      0.98     15000
weighted avg       0.97      0.97      0.97     15000
```



In [41]:

```
# Evaluate Logistic Regression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

y_pred_lr = log_reg.predict(X_test)

print("\n--- Evaluation for Logistic Regression ---")
print("Accuracy:", accuracy_score(y_test, y_pred_lr))
print("Precision:", precision_score(y_test, y_pred_lr, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_lr, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_lr, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_lr))

# Confusion Matrix as heatmap
cm_lr = confusion_matrix(y_test, y_pred_lr)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues', xticklabels=log_reg.clas
plt.title('Confusion Matrix - Logistic Regression')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

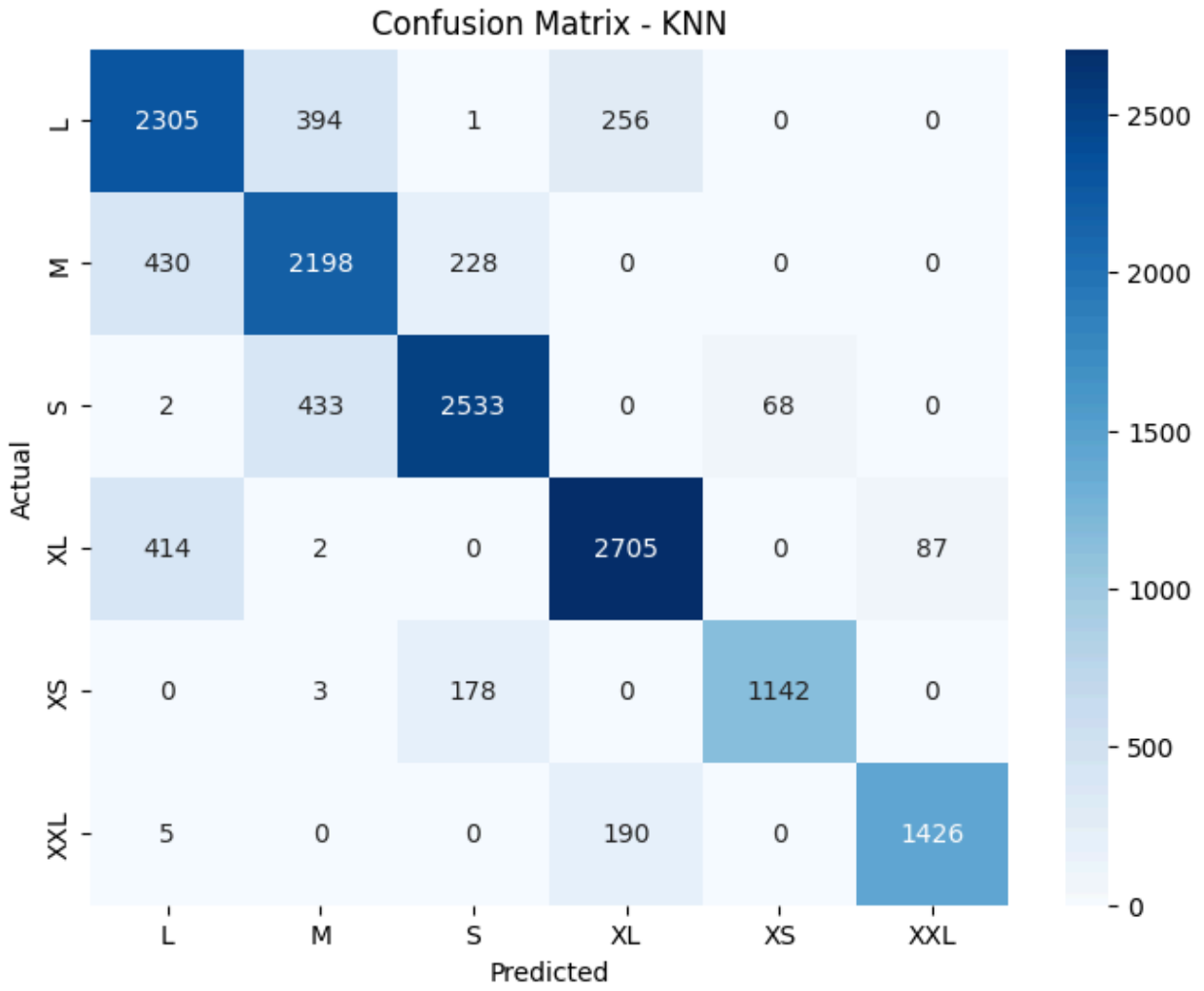
```
--- Evaluation for Logistic Regression ---
Accuracy: 0.9734666666666667
Precision: 0.9735579029118213
Recall: 0.9734666666666667
F1-score: 0.973480160088336

Classification Report:
              precision    recall  f1-score   support

           L       0.96      0.97      0.97      2956
           M       0.97      0.97      0.97      2856
           S       0.97      0.98      0.98      3036
          XL       0.97      0.98      0.98      3208
          XS       0.99      0.97      0.98      1323
         XXL       0.99      0.97      0.98      1621

    accuracy                           0.97     15000
   macro avg       0.98      0.97      0.97     15000
weighted avg       0.97      0.97      0.97     15000
```



In [42]:
```
# Evaluate KNN
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
```

```python
import seaborn as sns
import matplotlib.pyplot as plt

y_pred_knn = knn.predict(X_test)

print("\n--- Evaluation for KNN ---")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_knn, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_knn, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_knn))

# Confusion Matrix as heatmap
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues', xticklabels=knn.classes
plt.title('Confusion Matrix - KNN')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
--- Evaluation for KNN ---
Accuracy: 0.8206
Precision: 0.8251192626240232
Recall: 0.8206
F1-score: 0.8222415000341878

Classification Report:
               precision    recall  f1-score   support

           L       0.73      0.78      0.75      2956
           M       0.73      0.77      0.75      2856
           S       0.86      0.83      0.85      3036
          XL       0.86      0.84      0.85      3208
          XS       0.94      0.86      0.90      1323
         XXL       0.94      0.88      0.91      1621

    accuracy                           0.82     15000
   macro avg       0.84      0.83      0.84     15000
weighted avg       0.83      0.82      0.82     15000
```

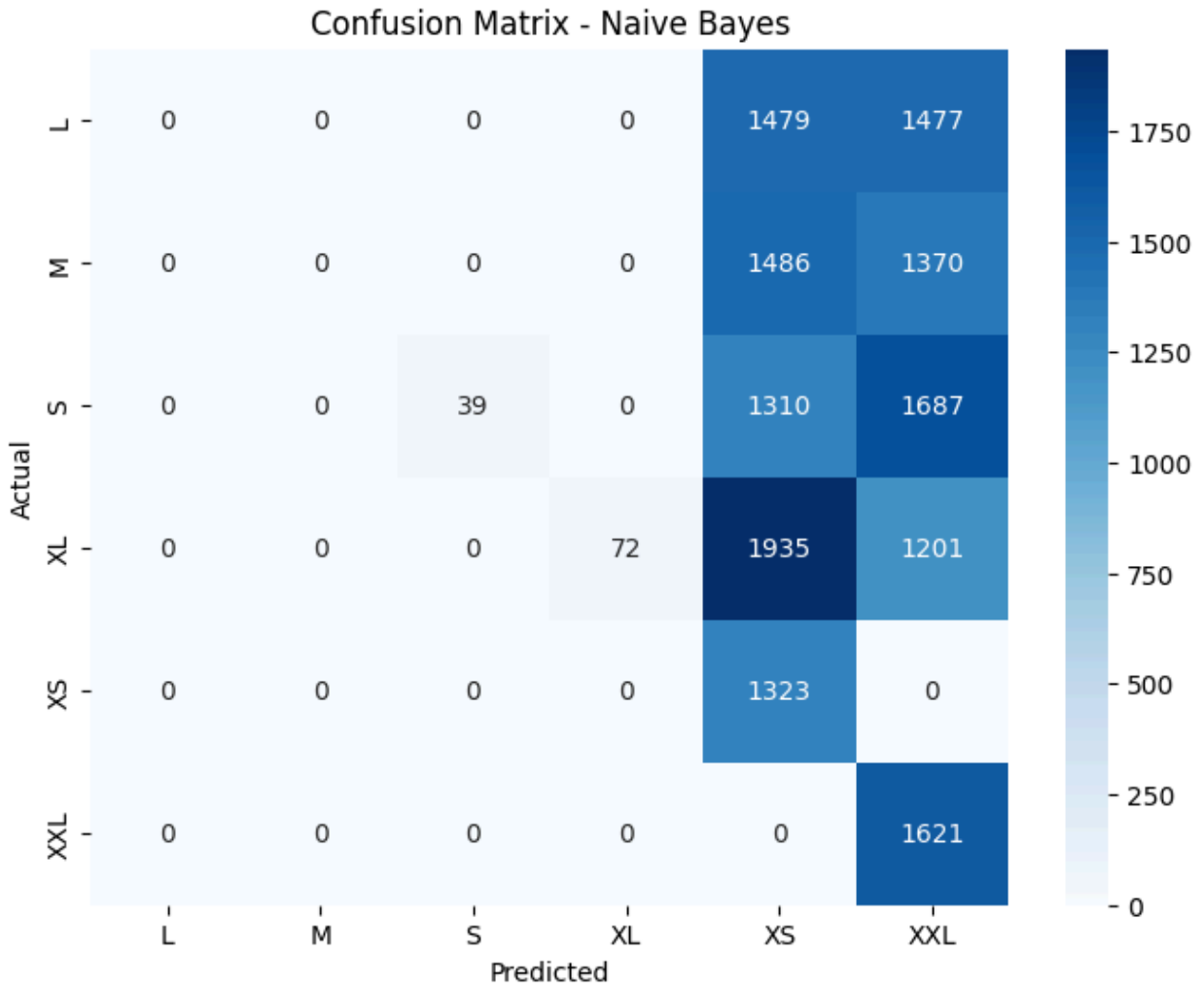


In [43]:
```python
# Evaluate Naive Bayes
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

y_pred_nb = naive_bayes.predict(X_test)

print("\n--- Evaluation for Naive Bayes ---")
print("Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Precision:", precision_score(y_test, y_pred_nb, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_nb, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_nb, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))

# Confusion Matrix as heatmap
cm_nb = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues', xticklabels=naive_bayes.
plt.title('Confusion Matrix - Naive Bayes')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
--- Evaluation for Naive Bayes ---
Accuracy: 0.20366666666666666
Precision: 0.45557102767399293
Recall: 0.20366666666666666
F1-score: 0.07990350850185343
```

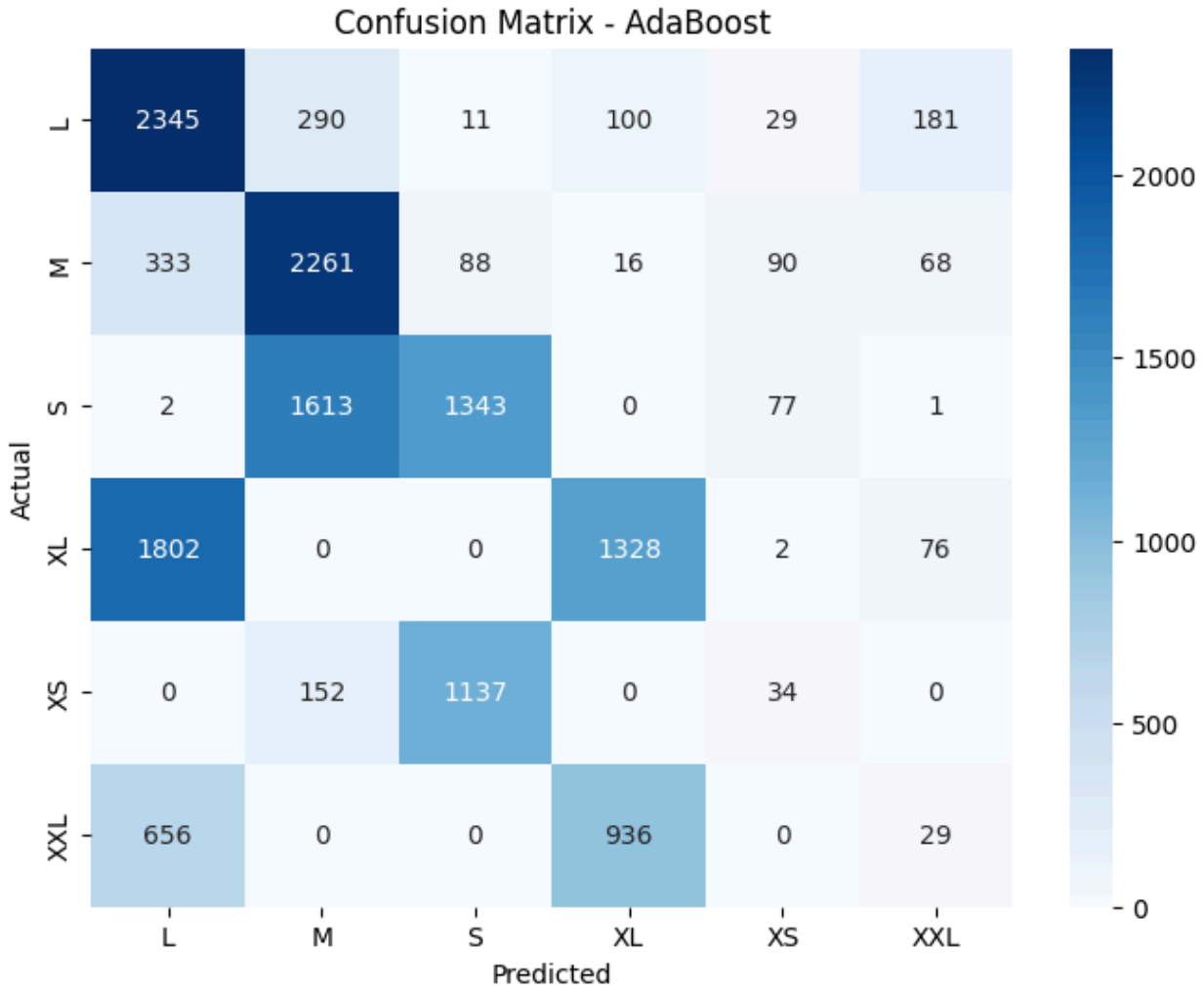
```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:156
5: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in lab
els with no predicted samples. Use `zero_division` parameter to control this be
havior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
Classification Report:
               precision    recall  f1-score   support

           L       0.00      0.00      0.00      2956
           M       0.00      0.00      0.00      2856
           S       1.00      0.01      0.03      3036
          XL       1.00      0.02      0.04      3208
          XS       0.18      1.00      0.30      1323
         XXL       0.22      1.00      0.36      1621

    accuracy                           0.20     15000
   macro avg       0.40      0.34      0.12     15000
weighted avg       0.46      0.20      0.08     15000
```



In [44]:
```
# Evaluate AdaBoost
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
import seaborn as sns
import matplotlib.pyplot as plt

y_pred_ab = adaboost.predict(X_test)

print("\n--- Evaluation for AdaBoost ---")
print("Accuracy:", accuracy_score(y_test, y_pred_ab))
print("Precision:", precision_score(y_test, y_pred_ab, average='weighted'))
print("Recall:", recall_score(y_test, y_pred_ab, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred_ab, average='weighted'))
print("\nClassification Report:\n", classification_report(y_test, y_pred_ab))

# Confusion Matrix as heatmap
cm_ab = confusion_matrix(y_test, y_pred_ab)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_ab, annot=True, fmt='d', cmap='Blues', xticklabels=adaboost.cla
plt.title('Confusion Matrix - AdaBoost')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
--- Evaluation for AdaBoost ---
Accuracy: 0.48933333333333334
Precision: 0.4361722384637669
Recall: 0.48933333333333334
F1-score: 0.43973832934615087

Classification Report:
              precision    recall  f1-score   support

           L       0.46      0.79      0.58      2956
           M       0.52      0.79      0.63      2856
           S       0.52      0.44      0.48      3036
          XL       0.56      0.41      0.48      3208
          XS       0.15      0.03      0.04      1323
         XXL       0.08      0.02      0.03      1621

    accuracy                           0.49     15000
   macro avg       0.38      0.41      0.37     15000
weighted avg       0.44      0.49      0.44     15000
```



In [46]:
```
import pandas as pd
```

```
# Assuming the model variables (log_reg, knn, decision_tree, random_forest, na

model_accuracies = {
    "Logistic Regression": log_reg.score(X_test, y_test),
    "KNN": knn.score(X_test, y_test),
    "Decision Tree": decision_tree.score(X_test, y_test),
    "Random Forest": random_forest.score(X_test, y_test),
    "Naive Bayes": naive_bayes.score(X_test, y_test),
    "AdaBoost": adaboost.score(X_test, y_test),
    "SVM": svm.score(X_test, y_test)
}

accuracy_df = pd.DataFrame(list(model_accuracies.items()), columns=['Model', '
accuracy_df = accuracy_df.sort_values(by='Accuracy', ascending=False).reset_ir

print("Model Accuracy Summary:")
display(accuracy_df)
```

Model Accuracy Summary:

	Model	Accuracy
0	SVM	0.974733
1	Logistic Regression	0.973467
2	Decision Tree	0.970533
3	Random Forest	0.962867
4	KNN	0.820600
5	AdaBoost	0.489333
6	Naive Bayes	0.203667

In [59]:

```
import pandas as pd
import numpy as np

def predict_size(age, height, weight, chest, waist, hip, gender, fit_preferenc
    # Create a DataFrame for the new input
    new_data = pd.DataFrame([[age, height, weight, chest, waist, hip, gender,
                              columns=['Age', 'Height_cm', 'Weight_kg', 'Chest_c

    # Preprocess the new data (scaling and one-hot encoding)
    # Use the same scaler and encoder fitted on the training data
    # Access the scaler and encoder from the environment
    global scaler, encoder

    # Scale numerical features
    numeric_cols_for_prediction = ['Age', 'Height_cm', 'Weight_kg', 'Chest_cm'
    scaled_numeric_features_new = scaler.transform(new_data[numeric_cols_for_p
    scaled_numeric_df_new = pd.DataFrame(scaled_numeric_features_new, columns=

    # One-hot encode categorical features
```

```
    encoded_features_new = encoder.transform(new_data[['Gender', 'Fit_Preferer
    encoded_df_new = pd.DataFrame(encoded_features_new, columns=encoder.get_fe

    # Combine processed features - ensure columns are in the same order as pro
    # We need to ensure the columns in the new data match the order of columns
    # Let's get the column order from X_train
    global X_train
    processed_df_new = pd.concat([scaled_numeric_df_new, encoded_df_new], axis

    # Reindex to ensure the same column order as X_train
    processed_df_new = processed_df_new.reindex(columns=X_train.columns, fill_


    # Make prediction using the loaded model
    global loaded_model
    predicted_size = loaded_model.predict(processed_df_new)

    return predicted_size[0]
```

In [61]:
```
# Example user inputs
example_age = 60
example_height = 179.0
example_weight = 90.0
example_chest = 95.0
example_waist = 75.0
example_hip = 99.0
example_gender = 'Female'
example_fit_preference = 'Slim'

# Get the recommended size using the predict_size function
recommended_size = predict_size(example_age, example_height, example_weight, e

print(f"For the given inputs, the recommended size is: {recommended_size}")
```

```
For the given inputs, the recommended size is: XL
```

In [48]:
```
import joblib

# Save the best-performing model (SVM)
best_model = svm  # Based on the accuracy summary

filename = 'SMARTSIZE_MODEL.pkl'
joblib.dump(best_model, filename)

print(f"Best model saved to {filename}")
```

```
Best model saved to SMARTSIZE_MODEL.pkl
```

In [49]:
```
import joblib

# Load the saved model
loaded_model = joblib.load('SMARTSIZE_MODEL.pkl')

print(f"Model loaded from SMARTSIZE_MODEL.pkl")
```

```
# You can now use loaded_model to make predictions
# For example:
# predictions = loaded_model.predict(X_test)
```

```
Model loaded from SMARTSIZE_MODEL.pkl
```