

START MY PROJECT 2 FOR (Practical_Machine_Learning_and_Data_Exploration)

Section 0 – Import Libraries

```
1 import warnings
2 warnings.filterwarnings("ignore")
3
4 import numpy as np
5 import pandas as pd
6
7 import matplotlib.pyplot as plt
8 import seaborn as sns
9 sns.set(style="whitegrid")
10
11 from sklearn.model_selection import train_test_split, GridSearchCV
12 from sklearn.compose import ColumnTransformer
13 from sklearn.pipeline import Pipeline
14 from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler
15 from sklearn.impute import SimpleImputer, KNNImputer
16
17 from sklearn.metrics import (
18     accuracy_score, precision_score, recall_score, f1_score,
19     classification_report, confusion_matrix
20 )
21
22 from sklearn.linear_model import LogisticRegression
23 from sklearn.tree import DecisionTreeClassifier
24 from sklearn.ensemble import RandomForestClassifier
25 from sklearn.svm import SVC
26 from sklearn.neighbors import KNeighborsClassifier
27
28 from sklearn.feature_selection import SelectKBest, chi2, mutual_info_classif, RFE
```

Section 1 – Project Introduction & Loading the Dataset

```
1 print("Dataset: bodyPerformance.csv")
2 print("Kaggle Link:")
3 print("https://www.kaggle.com/datasets/kukuroo3/body-performance-data")
4
5 df = pd.read_csv("bodyPerformance.csv")
6 print("\nProject: Body Performance Data Analysis")
7 display(df.head())
```

Dataset: bodyPerformance.csv
Kaggle Link:
<https://www.kaggle.com/datasets/kukuroo3/body-performance-data>

Project: Body Performance Data Analysis

	age	gender	height_cm	weight_kg	body_fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad_jump_cm	class
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4	60.0	217.0	C
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3	53.0	229.0	A
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0	49.0	181.0	C
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2	53.0	219.0	B
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1	45.0	217.0	B

Section 2 – Explore the Data

```
1 print("Dataset shape:", df.shape)
2 print("\nData info:")
3 print(df.info())
4 print("\nStatistics:")
5 display(df.describe())
6 print("\nMissing values per column:")
7 print(df.isnull().sum())
```

Dataset shape: (13393, 12)

Data info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13393 entries, 0 to 13392
Data columns (total 12 columns):
Column Non-Null Count Dtype
--- --
0 age 13393 non-null float64
1 gender 13393 non-null object
2 height_cm 13393 non-null float64
3 weight_kg 13393 non-null float64
4 body_fat_% 13393 non-null float64
5 diastolic 13393 non-null float64
6 systolic 13393 non-null float64
7 gripForce 13393 non-null float64
8 sit and bend forward_cm 13393 non-null float64
9 sit-ups counts 13393 non-null float64
10 broad_jump_cm 13393 non-null float64
11 class 13393 non-null object
dtypes: float64(10), object(2)
memory usage: 1.2+ MB
None

	age	height_cm	weight_kg	body_fat_%	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad_jump_cm
count	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000	13393.000000
mean	36.775106	168.559807	67.447316	23.240165	78.796842	130.234817	36.963877	15.209268	39.771224	190.129627
std	13.625639	8.426583	11.949666	7.256844	10.742033	14.713954	10.624864	8.456677	14.276698	39.868000
min	21.000000	125.000000	26.300000	3.000000	0.000000	0.000000	0.000000	-25.000000	0.000000	0.000000
25%	25.000000	162.400000	58.200000	18.000000	71.000000	120.000000	27.500000	10.900000	30.000000	162.000000
50%	32.000000	169.200000	67.400000	22.800000	79.000000	130.000000	37.900000	16.200000	41.000000	193.000000
75%	48.000000	174.800000	75.300000	28.000000	86.000000	141.000000	45.200000	20.700000	50.000000	221.000000
max	64.000000	193.800000	138.100000	78.400000	156.200000	201.000000	70.500000	213.000000	80.000000	303.000000

Missing values per column:
age 0
gender 0
height_cm 0
weight_kg 0
body_fat_% 0
diastolic 0
systolic 0
gripForce 0
sit and bend forward_cm 0
sit-ups counts 0
broad_jump_cm 0
class 0
dtype: int64

Section 3 – Fix Missing Values

```
1 missing_pct = df.isnull().mean() * 100
2 print("Missing percentage per column:")
3 print(missing_pct.sort_values(ascending=False))
4
5 cols_to_drop = missing_pct[missing_pct > 30].index
6 if len(cols_to_drop) > 0:
7     df = df.drop(columns=cols_to_drop)
8     print("\nDropped columns (>30% missing):", list(cols_to_drop))
9 else:
10    print("\nNo columns dropped.")
11
12 numeric_cols = df.select_dtypes(include=["float64", "int64"]).columns
13 categorical_cols = df.select_dtypes(include=["object", "category", "bool"]).columns
14
15 for col in numeric_cols:
16     df[col] = df[col].fillna(df[col].median())
17
18 for col in categorical_cols:
19     if df[col].isnull().any():
20         df[col] = df[col].fillna(df[col].mode()[0])
21
22 print("\nMissing values after filling:")
23 print(df.isnull().sum())
```

Missing percentage per column:
age 0.0
gender 0.0
height_cm 0.0
weight_kg 0.0
body_fat_% 0.0
diastolic 0.0
systolic 0.0
gripForce 0.0
sit and bend forward_cm 0.0
sit-ups counts 0.0
broad_jump_cm 0.0
class 0.0
dtype: float64

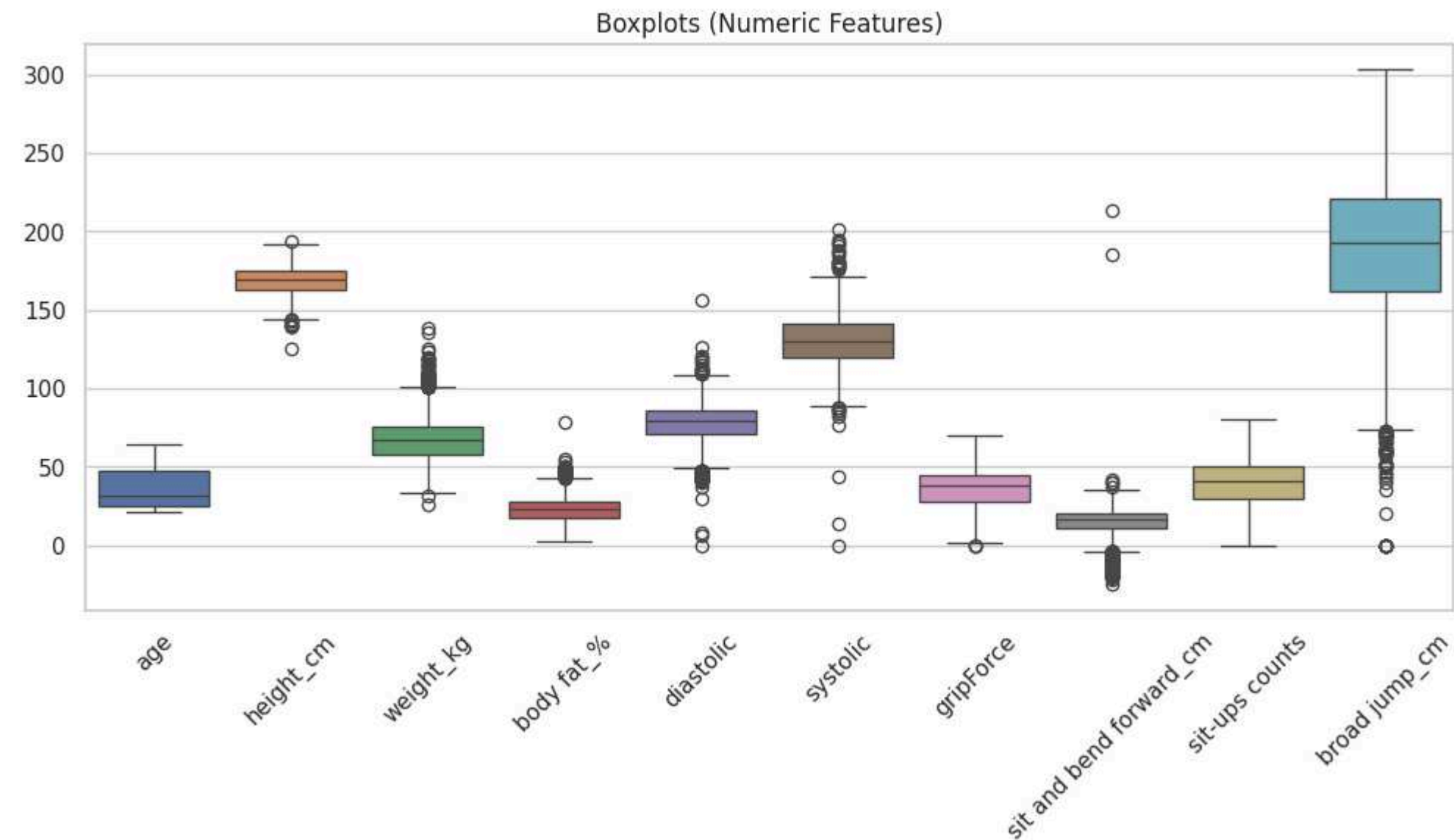
No columns dropped.

Missing values after filling:
age 0
gender 0
height_cm 0
weight_kg 0
body_fat_% 0
diastolic 0
systolic 0
gripForce 0
sit and bend forward_cm 0
sit-ups counts 0
broad_jump_cm 0
class 0
dtype: int64

Section 4 – Detect Outliers

```
1 target_col = "class"
2 numeric_feature_cols = [c for c in numeric_cols if c != target_col]
3
4 plt.figure(figsize=(12,5))
5 sns.boxplot(data=df[numeric_feature_cols])
```

```
6 plt.xticks(rotation=45)
7 plt.title("Boxplots (Numeric Features)")
8 plt.show()
9
10 def outliers_iqr(column):
11     Q1 = column.quantile(0.25)
12     Q3 = column.quantile(0.75)
13     IQR = Q3 - Q1
14     low = Q1 - 1.5 * IQR
15     high = Q3 + 1.5 * IQR
16     return (column < low) | (column > high)
17
18 data_iqr = df.copy()
19 for col in numeric_feature_cols:
20     data_iqr[col + "_iqr_outlier"] = outliers_iqr(data_iqr[col])
21
22 print("Sample rows with IQR outliers:")
23 display(data_iqr[data_iqr.filter(like="_iqr_outlier").any(axis=1)].head())
24
25 def outliers_z(col, threshold=3):
26     mean = col.mean()
27     std = col.std()
28     z = (col - mean) / std
29     return abs(z) > threshold
30
31 data_z = df.copy()
32 for col in numeric_feature_cols:
33     data_z[col + "_z_outlier"] = outliers_z(data_z[col])
34
35 print("Sample rows with Z-score outliers:")
36 display(data_z[data_z.filter(like="_z_outlier").any(axis=1)].head())
```



Sample rows with IQR outliers:

	age	gender	height_cm	weight_kg	body fat %	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	...	age_iqr_outlier	height_cm_iqr_outlier	weight_kg_iqr_outlier	body fat %_iqr_outlier	diastolic_iqr_outlier	systolic_iqr_outlier	gripForce_iqr_outlier	sit and bend forward_cm_iqr_outlier	sit-ups counts_iqr_outlier	broad jump_cm_iqr_outlier	
47	26.0	M	172.1	66.10	24.9	62.0	102.0	19.9	-9.9	44.0	...	False	False	False	False	False	False	False	True	False	False	
58	42.0	M	174.1	103.10	30.5	78.0	123.0	50.7	3.6	25.0	...	False	False	True	False	False	False	False	False	False	False	
75	22.0	M	171.9	78.46	22.9	64.0	116.0	43.6	-9.3	46.0	...	False	False	False	False	False	False	False	True	False	False	
136	61.0	F	153.2	62.20	37.0	71.0	111.0	25.7	3.0	0.0	...	False	False	False	False	False	False	False	False	False	True	
139	22.0	M	180.3	91.70	32.1	75.0	134.0	28.6	-6.9	36.0	...	False	False	False	False	False	False	False	True	False	False	

5 rows × 22 columns

Sample rows with Z-score outliers:

	age	gender	height_cm	weight_kg	body fat %	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	...	age_z_outlier	height_cm_z_outlier	weight_kg_z_outlier	body fat %_z_outlier	diastolic_z_outlier	systolic_z_outlier	gripForce_z_outlier	sit and bend forward_cm_z_outlier	sit-ups counts_z_outlier	broad jump_cm_z_outlier	
136	61.0	F	153.2	62.2	37.0	71.0	111.0	25.7	3.0	0.0	...	False	False	False	False	False	False	False	False	False	True	
231	26.0	M	180.2	103.4	24.7	70.0	128.0	67.6	16.2	46.0	...	False	False	True	False	False	False	False	False	False	False	
296	63.0	M	160.5	68.2	28.7	84.0	193.0	37.1	14.3	31.0	...	False	False	False	False	False	True	False	False	False	False	
303	30.0	M	185.2	108.9	23.1	98.0	143.0	54.0	27.7	52.0	...	False	False	True	False	False	False	False	False	False	False	
327	24.0	M	176.1	74.8	24.3	96.0	125.0	35.7	-12.0	1.0	...	False	False	False	False	False	False	False	True	False	False	

5 rows × 22 columns

Section 5 – Fix Outliers (Replace with NaN + KNNImputer for numeric)

```
1 def replace_outliers_iqr(column):
2     Q1 = column.quantile(0.25)
3     Q3 = column.quantile(0.75)
4     IQR = Q3 - Q1
5     low = Q1 - 1.5 * IQR
6     high = Q3 + 1.5 * IQR
7     return column.where((column >= low) & (column <= high), np.nan)
8
9 clean_df = df.copy()
10
11 for col in numeric_feature_cols:
12     clean_df[col] = replace_outliers_iqr(clean_df[col])
13
14 print("Missing values after replacing outliers with NaN:")
15 print(clean_df.isnull().sum())
16
17 imputer = KNNImputer(n_neighbors=3)
18 imputed_numeric = pd.DataFrame(
19     imputer.fit_transform(clean_df[numeric_feature_cols]),
20     columns=numeric_feature_cols
21 )
22
23 data_imputed = pd.concat(
24     [
25         imputed_numeric.reset_index(drop=True),
26         clean_df[categorical_cols].reset_index(drop=True)
27     ],
28     axis=1
29 )
30
31 data_imputed = data_imputed[df.columns]
32
33 print("\nCleaned dataset preview:")
34 display(data_imputed.head())
```

Missing values after replacing outliers with NaN:

age	0
gender	0
height_cm	10
weight_kg	83
body fat %	77
diastolic	54
systolic	29
gripForce	3
sit and bend forward_cm	409
sit-ups counts	0
broad jump_cm	57
class	0

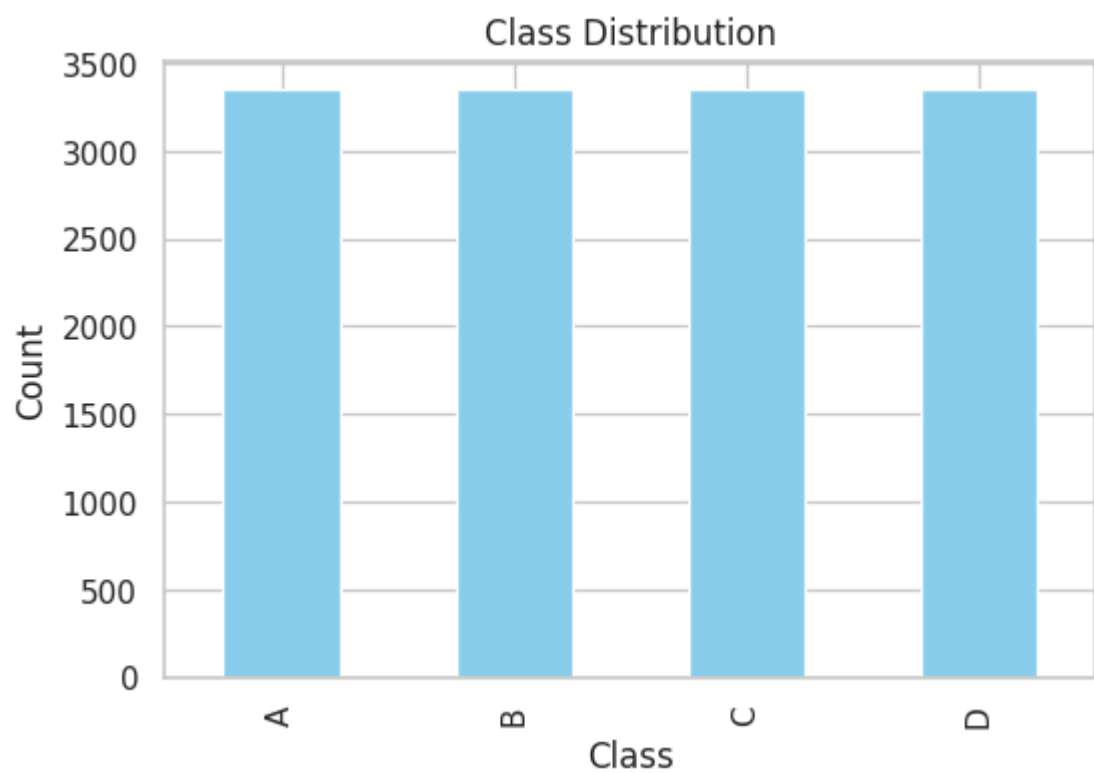
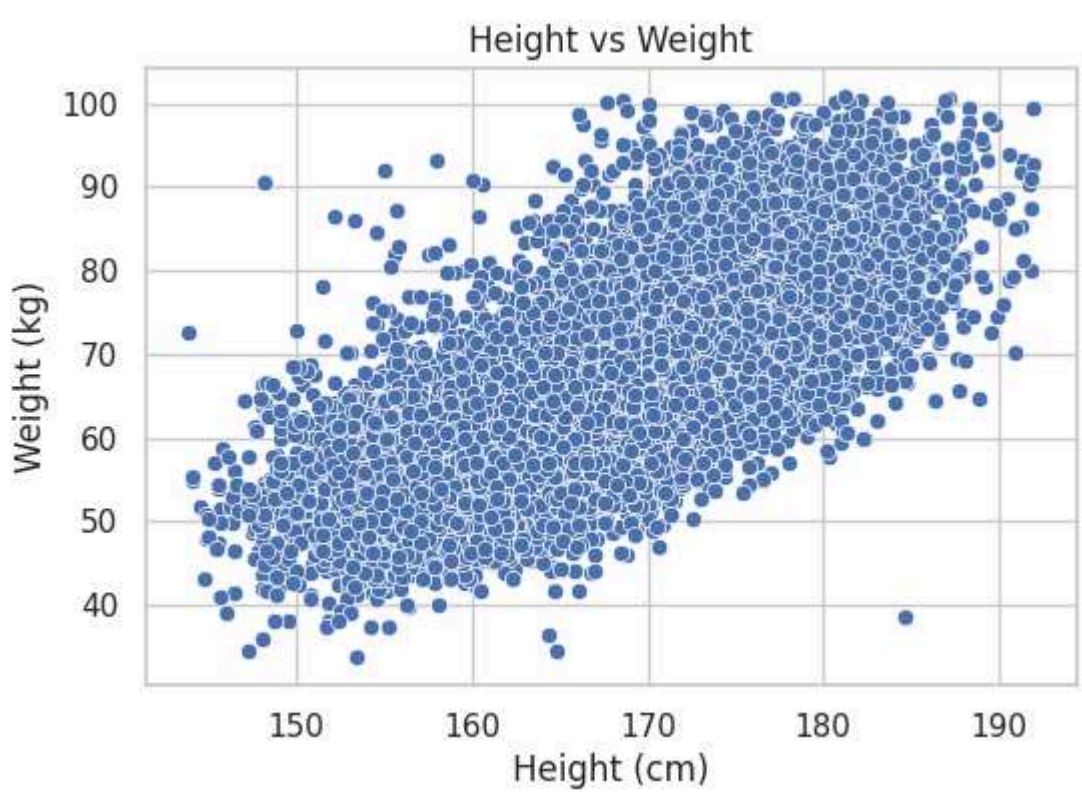
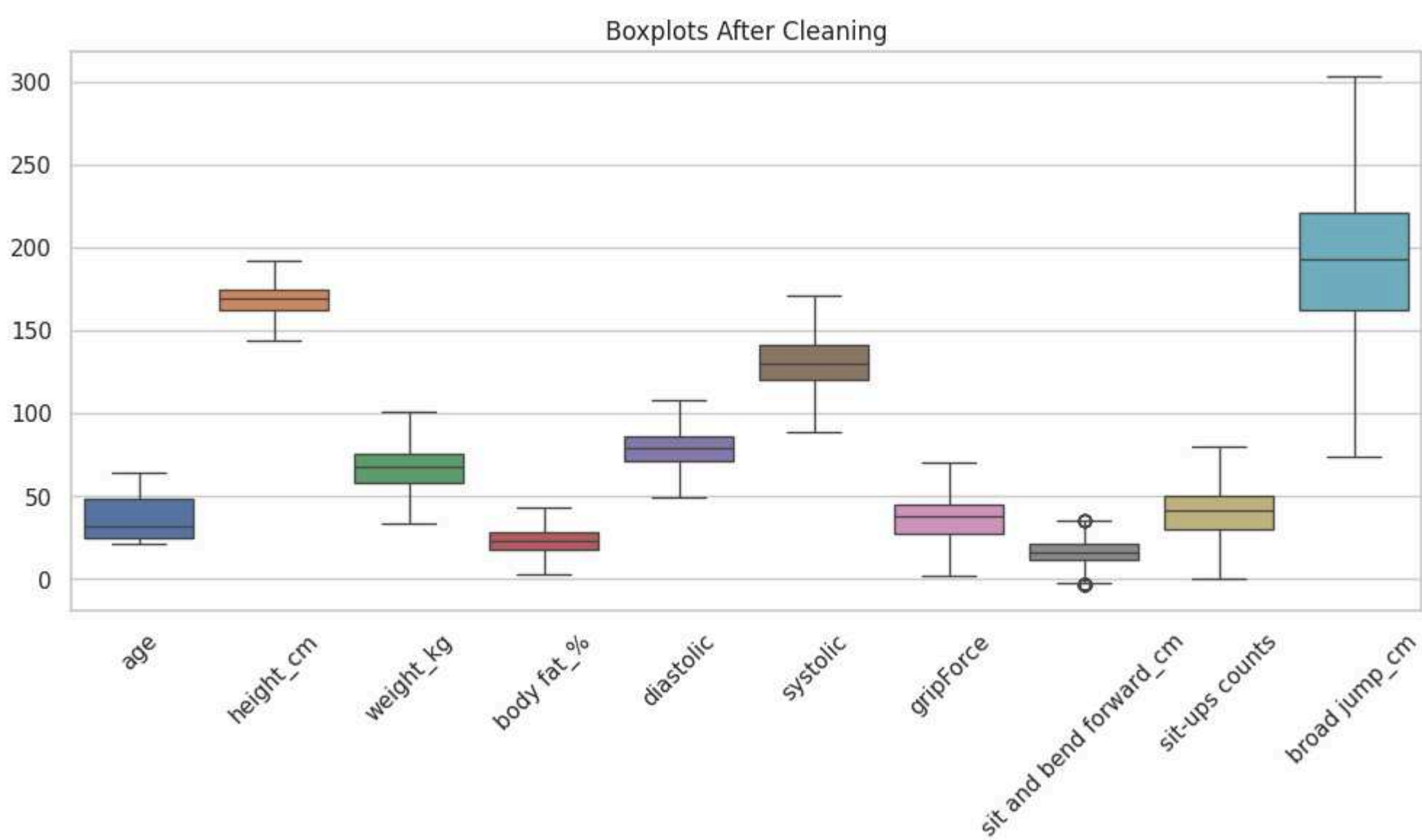
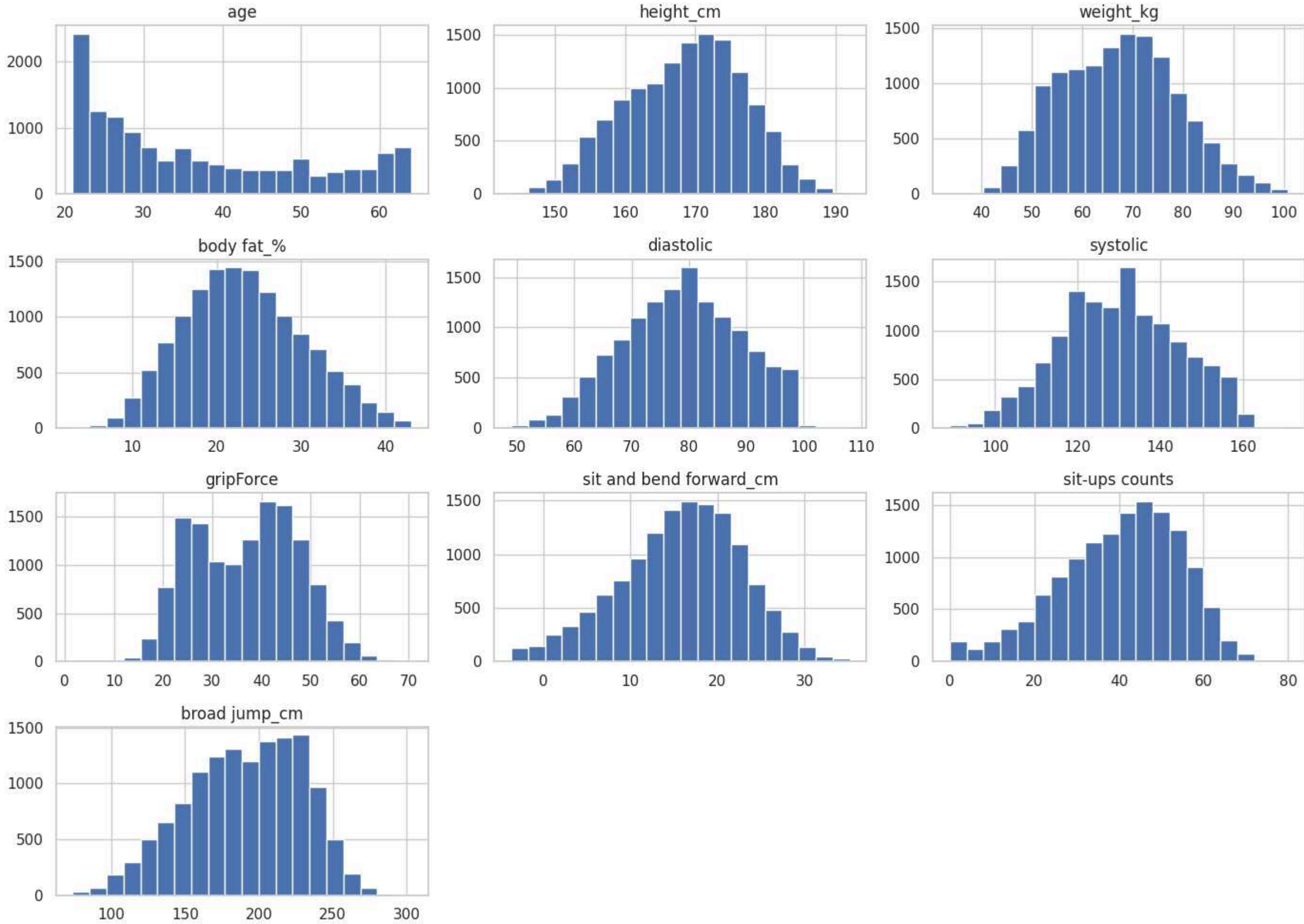
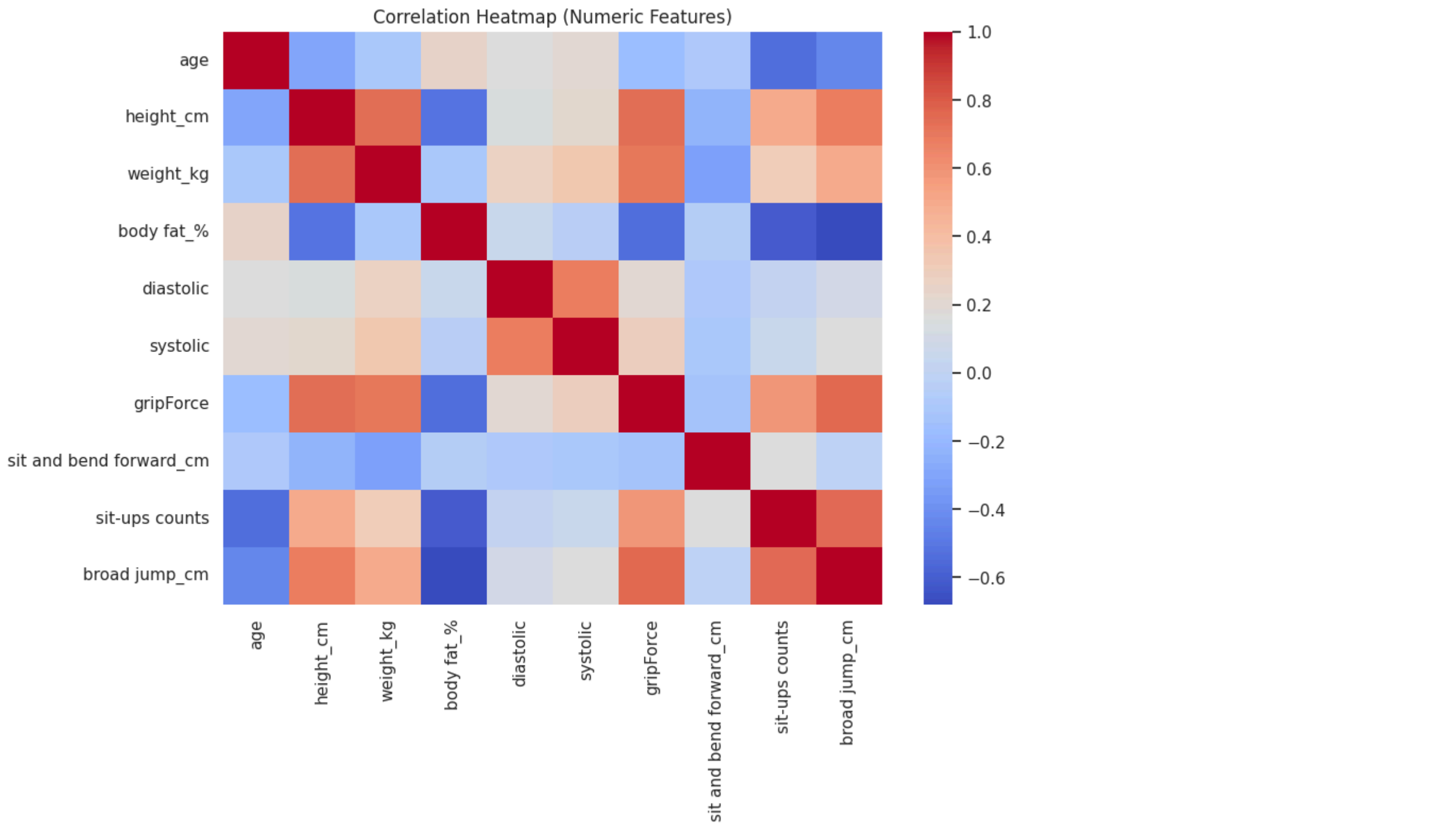
dtype: int64

Cleaned dataset preview:

	age	gender	height_cm	weight_kg	body fat %	diastolic	systolic	gripForce	sit and bend forward_cm	sit-ups counts	broad jump_cm	class
0	27.0	M	172.3	75.24	21.3	80.0	130.0	54.9	18.4	60.0	217.0	C
1	25.0	M	165.0	55.80	15.7	77.0	126.0	36.4	16.3	53.0	229.0	A
2	31.0	M	179.6	78.00	20.1	92.0	152.0	44.8	12.0	49.0	181.0	C
3	32.0	M	174.5	71.10	18.4	76.0	147.0	41.4	15.2	53.0	219.0	B
4	28.0	M	173.8	67.70	17.1	70.0	127.0	43.5	27.1	45.0	217.0	B

Section 6 – Data Visualization

```
1 plt.figure(figsize=(10,7))
2 sns.heatmap(imputed_numeric.corr(), annot=False, cmap="coolwarm")
3 plt.title("Correlation Heatmap (Numeric Features)")
4 plt.show()
5
6 data_imputed[numeric_feature_cols].hist(figsize=(14,10), bins=20)
7 plt.tight_layout()
8 plt.show()
9
10 plt.figure(figsize=(12,5))
11 sns.boxplot(data=data_imputed[numeric_feature_cols])
12 plt.xticks(rotation=45)
13 plt.title("Boxplots After Cleaning")
14 plt.show()
15
16 if "height_cm" in data_imputed.columns and "weight_kg" in data_imputed.columns:
17     plt.figure(figsize=(6,4))
18     sns.scatterplot(x=data_imputed["height_cm"], y=data_imputed["weight_kg"])
19     plt.title("Height vs Weight")
20     plt.xlabel("Height (cm)")
21     plt.ylabel("Weight (kg)")
22     plt.show()
23
24 data_imputed[target_col].value_counts().sort_index().plot(kind="bar", figsize=(6,4), color="skyblue")
25 plt.title("Class Distribution")
26 plt.xlabel("Class")
27 plt.ylabel("Count")
28 plt.show()
```



Section 7 – Prepare Data for Machine Learning (Split)

```
1 X = data_imputed.drop(columns=[target_col])
2 y = data_imputed[target_col]
3
4 X_train, X_test, y_train, y_test = train_test_split(
5     X, y, test_size=0.30, random_state=42, stratify=y
6 )
7
8 print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
9 print("\nTarget distribution (train):")
10 print(y_train.value_counts())
11 print("\nTarget distribution (test):")
12 print(y_test.value_counts())
```

```
Train shape: (9375, 11) Test shape: (4018, 11)

Target distribution (train):
class
C    2344
D    2344
A    2344
B    2343
Name: count, dtype: int64

Target distribution (test):
class
C    1005
D    1005
B    1004
A    1004
Name: count, dtype: int64
```

Section 8 – Grid Search Optimizer (All Features)

```
1 num_cols2 = X.select_dtypes(include=["int64", "float64"]).columns
2 cat_cols2 = X.select_dtypes(include=["object", "category", "bool"]).columns
3
4 preprocess_std = ColumnTransformer(
5     transformers=[
6         ("num", Pipeline(steps=[
7             ("imputer", SimpleImputer(strategy="median")),
8             ("scaler", StandardScaler())
9         ]), num_cols2),
10        ("cat", Pipeline(steps=[
11            ("imputer", SimpleImputer(strategy="most_frequent")),
12            ("onehot", OneHotEncoder(handle_unknown="ignore"))
13        ]), cat_cols2),
14    ]
15 )
16
```

```
17 def eval_cls(y_true, y_pred):
18     return {
19         "Accuracy": accuracy_score(y_true, y_pred),
20         "Precision": precision_score(y_true, y_pred, average="weighted", zero_division=0),
21         "Recall": recall_score(y_true, y_pred, average="weighted", zero_division=0),
22         "F1": f1_score(y_true, y_pred, average="weighted", zero_division=0),
23     }
24
25 models_and_grids = {
26     "LogisticRegression": (
27         LogisticRegression(max_iter=2000),
28         {'clf__C': [0.1, 1, 10]}
29     ),
30     "RandomForest": (
31         RandomForestClassifier(random_state=42, n_jobs=-1),
32         {
33             "clf__n_estimators": [100, 200],
34             "clf__max_depth": [None, 10, 20],
35             "clf__min_samples_split": [2, 5]
36         }
37     ),
38     "SVM": (
39         SVC(),
40         {"clf__kernel": ["rbf", "linear"], "clf__C": [0.1, 1, 10]}
41     ),
42     "KNN": (
43         KNeighborsClassifier(),
44         {"clf__n_neighbors": [3, 5, 7, 9]}
45     ),
46     "DecisionTree": (
47         DecisionTreeClassifier(random_state=42),
48         {"clf__max_depth": [None, 5, 10, 20], "clf__min_samples_split": [2, 5, 10]}
49     )
50 }
51
52 baseline_results = []
53 best_estimators = {}
54
55 for name, (model, grid) in models_and_grids.items():
56     pipe = Pipeline(steps=[("prep", preprocess_std), ("clf", model)])
57     gs = GridSearchCV(pipe, param_grid=grid, cv=5, scoring="accuracy", n_jobs=-1)
58     gs.fit(X_train, y_train)
59
60     best_model = gs.best_estimator_
61     best_estimators[name] = best_model
62
63     preds = best_model.predict(X_test)
64     m = eval_cls(y_test, preds)
65
66     baseline_results.append({
67         "Model": name,
68         *m,
69         "BestParams": gs.best_params_
70     })
71
72 baseline_df = pd.DataFrame(baseline_results).sort_values("F1", ascending=False)
73 print("GRID SEARCH RESULTS (ALL FEATURES):")
74 display(baseline_df)
75
76 top_model_name = baseline_df.iloc[0]["Model"]
77 print("\nTop model:", top_model_name)
78 print("Best Params:", baseline_df.iloc[0]["BestParams"])
79
80 top_preds = best_estimators[top_model_name].predict(X_test)
81 print("\nClassification Report (Top Model):\n")
82 print(classification_report(y_test, top_preds, zero_division=0))
83
84 print("\nConfusion Matrix (Top Model):\n")
85 print(confusion_matrix(y_test, top_preds))
```

GRID SEARCH RESULTS (ALL FEATURES):

	Model	Accuracy	Precision	Recall	F1	BestParams	
1	RandomForest	0.727476	0.728926	0.727476	0.725366	{'clf__max_depth': None, 'clf__min_samples_spl...	
2	SVM	0.707317	0.711794	0.707317	0.706487	{'clf__C': 10, 'clf__kernel': 'rbf'}	
4	DecisionTree	0.662519	0.669167	0.662519	0.662127	{'clf__max_depth': 10, 'clf__min_samples_spl...	
0	LogisticRegression	0.612494	0.606129	0.612494	0.608321	{'clf__C': 1}	
3	KNN	0.609009	0.628440	0.609009	0.607238	{'clf__n_neighbors': 9}	

Top model: RandomForest
Best Params: {'clf__max_depth': None, 'clf__min_samples_split': 2, 'clf__n_estimators': 200}

Classification Report (Top Model):

	precision	recall	f1-score	support
A	0.72	0.88	0.79	1004
B	0.62	0.58	0.60	1004
C	0.70	0.65	0.67	1005
D	0.88	0.81	0.84	1005
accuracy			0.73	4018
macro avg	0.73	0.73	0.73	4018
weighted avg	0.73	0.73	0.73	4018

Confusion Matrix (Top Model):

```
[[879 115  8  2]
 [234 581 146 43]
 [181 185 652  67]
 [ 15  54 125 811]]
```

Next steps: [Generate code with baseline_df](#) [New interactive sheet](#)

Section 9 – Feature Selection (Two Methods)

Method 1: Chi2 (SelectKBest chi2)

Method 2: Mutual Information (SelectKBest mutual_info_classif)

```
1 def get_feature_names(ct: ColumnTransformer):
2     output_features = []
3     for name, trans, cols in ct.transformers_:
4         if name == "remainder":
5             continue
6         if hasattr(trans, "named_steps"):
7             last = list(trans.named_steps.values())[-1]
8         else:
9             last = trans
10        if hasattr(last, "get_feature_names_out"):
11            try:
12                out = last.get_feature_names_out(cols)
13            except:
14                out = last.get_feature_names_out()
15            output_features.extend(list(out))
16        else:
17            output_features.extend(list(cols))
18    return np.array(output_features)
19
20 preprocess_mm = ColumnTransformer(
21     transformers=[
22         ("num", Pipeline(steps=[
23             ("imputer", SimpleImputer(strategy="median")),
24             ("scaler", MinMaxScaler())
25         ]), num_cols2),
26         ("cat", Pipeline(steps=[
27             ("imputer", SimpleImputer(strategy="most_frequent")),
28             ("onehot", OneHotEncoder(handle_unknown="ignore"))
29         ]), cat_cols2),
30     ]
31 )
32
33 Xtr_mm = preprocess_mm.fit_transform(X_train)
34 Xte_mm = preprocess_mm.transform(X_test)
35 feat_names_mm = get_feature_names(preprocess_mm)
36
37 Xtr_std = preprocess_std.fit_transform(X_train)
38 Xte_std = preprocess_std.transform(X_test)
39 feat_names_std = get_feature_names(preprocess_std)
40
41 k = 15
42
43 chi_selector = SelectKBest(score_func=chi2, k=min(k, Xtr_mm.shape[1]))
44 Xtr_chi = chi_selector.fit_transform(Xtr_mm, y_train)
45 Xte_chi = chi_selector.transform(Xte_mm)
46
47 chi_scores = chi_selector.scores_
48 chi_idx = np.argsort(chi_scores)[::-1][:chi_selector.k]
49 print("Top Chi2 Features:")
50 for i in chi_idx[:15]:
51     print(f"{feat_names_mm[i]} --> score: {chi_scores[i]:.4f}")
52
53 mi_selector = SelectKBest(score_func=mutual_info_classif, k=min(k, Xtr_std.shape[1]))
54 Xtr_mi = mi_selector.fit_transform(Xtr_std, y_train)
55 Xte_mi = mi_selector.transform(Xte_std)
56
57 mi_scores = mi_selector.scores_
58 mi_idx = np.argsort(mi_scores)[::-1][:mi_selector.k]
59 print("\nTop Mutual Information Features:")
60 for i in mi_idx[:15]:
61     print(f"{feat_names_std[i]} --> score: {mi_scores[i]:.4f}")
62
```

Top Chi2 Features:
sit and bend forward_cm --> score: 209.9947
sit-ups counts --> score: 127.7404
body fat_% --> score: 80.1380
gender_F --> score: 51.4933
broad jump_cm --> score: 36.4367
gender_M --> score: 31.0909
weight_kg --> score: 26.9397
age --> score: 10.9164
gripForce --> score: 8.6262
diastolic --> score: 2.5986
height_cm --> score: 1.9070
systolic --> score: 1.1355

Top Mutual Information Features:
sit and bend forward_cm --> score: 0.2745
sit-ups counts --> score: 0.1336
body fat_% --> score: 0.0758
broad jump_cm --> score: 0.0593
gripForce --> score: 0.0374
weight_kg --> score: 0.0148
gender_M --> score: 0.0127
gender_F --> score: 0.0127
age --> score: 0.0123
height_cm --> score: 0.0109
diastolic --> score: 0.0000
systolic --> score: 0.0000

Section 10 – Models on Selected Features (GridSearch + Evaluate)

```
1 def gridsearch_on_selected(Xtr, Xte, ytr, yte, tag):
2     out = []
3
4     chosen_models = {
5         "LogisticRegression": (
6             LogisticRegression(max_iter=2000),
7             {"C": [0.1, 1, 10]}
8         ),
```

```
9         "RandomForest": {
10             RandomForestClassifier(random_state=42, n_jobs=-1),
11             {"n_estimators": [100, 200], "max_depth": [None, 10, 20]}
12         )
13     }
14
15     for name, (model, grid) in chosen_models.items():
16         gs = GridSearchCV(model, param_grid=grid, cv=5, scoring="accuracy", n_jobs=-1)
17         gs.fit(Xtr, ytr)
18
19         preds = gs.best_estimator_.predict(Xte)
20         m = eval_cls(yte, preds)
21
22         out.append({"FS": tag, "Model": name, **m, "BestParams": gs.best_params_})
23
24         print(f"\n[{tag}] {name} BestParams:", gs.best_params_)
25         print("Metrics:", m)
26         print("Confusion Matrix:\n", confusion_matrix(yte, preds))
27
28     return pd.DataFrame(out)
29
30 chi_df = gridsearch_on_selected(Xtr_chi, Xte_chi, y_train, y_test, "Chi2")
31 mi_df = gridsearch_on_selected(Xtr_mi, Xte_mi, y_train, y_test, "MutualInfo")
32
33 final_df = pd.concat([chi_df, mi_df], ignore_index=True).sort_values(["F1", "Accuracy"], ascending=False)
34
35 print("\n===== FINAL RESULTS (Selected Features) =====")
36 display(final_df)
```

```
[Chi2] LogisticRegression BestParams: {'C': 10}
Metrics: {'Accuracy': 0.6090094574415131, 'Precision': 0.6022642257294133, 'Recall': 0.6090094574415131, 'F1': 0.6045430226815717}
Confusion Matrix:
[[758 232 14  0]
 [255 451 254 44]
 [ 99 203 477 220]
 [ 21 57 166 761]]

[Chi2] RandomForest BestParams: {'max_depth': None, 'n_estimators': 200}
Metrics: {'Accuracy': 0.7264808362369338, 'Precision': 0.7278233791804237, 'Recall': 0.7264808362369338, 'F1': 0.7243180728572042}
Confusion Matrix:
[[879 117  6  2]
 [234 580 146 44]
 [100 186 650 69]
 [ 16 55 124 810]]

[MutualInfo] LogisticRegression BestParams: {'C': 1}
Metrics: {'Accuracy': 0.6124937779990045, 'Precision': 0.606129017176413, 'Recall': 0.6124937779990045, 'F1': 0.6083206926976533}
Confusion Matrix:
[[759 231 14  0]
 [248 459 254 43]
 [ 99 203 480 223]
 [ 20 57 165 763]]

[MutualInfo] RandomForest BestParams: {'max_depth': None, 'n_estimators': 200}
Metrics: {'Accuracy': 0.727476356396217, 'Precision': 0.7289263572897089, 'Recall': 0.727476356396217, 'F1': 0.7253658852697776}
Confusion Matrix:
[[879 115  0  2]
 [234 581 146 43]
 [101 185 652 67]
 [ 15 54 125 811]]
```

===== FINAL RESULTS (Selected Features) =====							
	FS	Model	Accuracy	Precision	Recall	F1	BestParams
3	MutualInfo	RandomForest	0.727476	0.728926	0.727476	0.725366	['max_depth': None, 'n_estimators': 200]
1	Chi2	RandomForest	0.726481	0.727823	0.726481	0.724318	['max_depth': None, 'n_estimators': 200]
2	MutualInfo	LogisticRegression	0.612494	0.606129	0.612494	0.608321	['C': 1]
0	Chi2	LogisticRegression	0.609009	0.602264	0.609009	0.604544	['C': 10]

Next steps: [Generate code with final_df](#) [New interactive sheet](#)

_____DONE-PROJECT-FOR-Body-Performance-Data-Analysis