

In [4]: `import pandas as pd`

In [5]: `df = pd.read_csv( 'fitness_dataset.csv')`

In [6]: `df.head()`

Out[6]:

	age	height_cm	weight_kg	heart_rate	blood_pressure	sleep_hours	nutrition_quality	activity_index	smokes	gender	is_fit
0	56	152	65	69.6	117.0	NaN	2.37	3.97	0	F	1
1	69	186	95	60.8	114.8	7.5	8.77	3.19	0	F	1
2	46	192	103	61.4	116.4	NaN	8.20	2.03	0	F	0
3	32	189	83	60.2	130.1	7.0	6.18	3.68	0	M	1
4	60	175	99	58.1	115.8	8.0	9.95	4.83	1	F	1

In [7]: `len(df)`

Out[7]: 2000

In [8]: `df.isnull().sum()`

Out[8]:

age	0
height_cm	0
weight_kg	0
heart_rate	0
blood_pressure	0
sleep_hours	160
nutrition_quality	0
activity_index	0
smokes	0
gender	0
is_fit	0

dtype: int64

In [9]: `missing_percentage = df.isnull().mean() * 100`
`columns_with_missing_30 = missing_percentage[missing_percentage > 30]`
`print("Columns with more than 30% missing values:")`
`print(columns_with_missing_30)`

Columns with more than 30% missing values:
Series([], dtype: float64)

In [10]: `df.describe()`

Out[10]:

	age	height_cm	weight_kg	heart_rate	blood_pressure	sleep_hours	nutrition_quality	activity_index	smokes	is_fit
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	1840.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	49.114000	174.533300	83.540500	70.288600	119.908850	7.513315	5.035140	2.999040	0.450500	0.399500
std	17.926564	14.37175	25.852534	11.846339	14.578032	1.502031	2.864156	1.136383	0.497668	0.489918
min	18.000000	150.00000	30.000000	45.000000	90.000000	4.000000	0.000000	1.000000	0.000000	0.000000
25%	34.000000	162.00000	64.000000	62.100000	109.700000	6.500000	2.547500	2.037500	0.000000	0.000000
50%	49.000000	174.00000	83.000000	70.250000	120.000000	7.500000	5.065000	2.980000	0.000000	0.000000
75%	65.000000	187.00000	102.000000	78.425000	129.800000	8.600000	7.470000	3.950000	1.000000	1.000000
max	79.000000	199.00000	250.000000	118.600000	171.200000	12.000000	10.000000	4.990000	1.000000	1.000000

In [11]: `print(df['sleep_hours'].mean())`

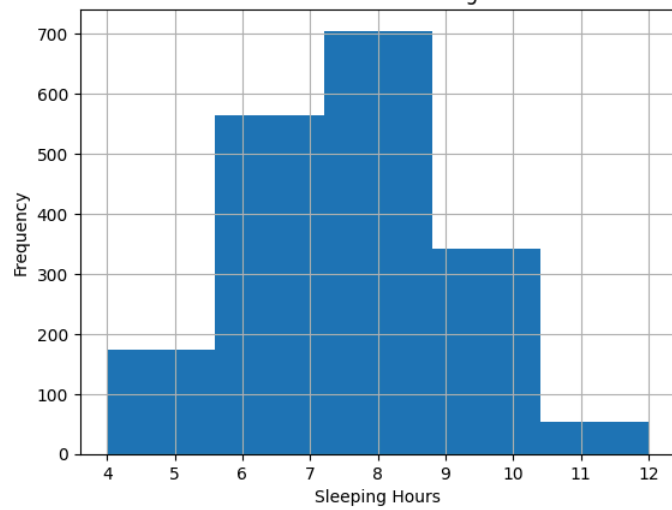
7.5133152173913045

In [12]: `print(df['sleep_hours'].median())`

7.5

In [13]: `import matplotlib.pyplot as plt`
`# Histogram for the column "sleep_hours"`
`df['sleep_hours'].hist(bins=5)`
`plt.title('Data Distribution Histogram')`
`plt.xlabel('Sleeping Hours')`
`plt.ylabel('Frequency')`
`plt.show()`

Data Distribution Histogram



```
In [14]: df['sleep_hours'] = df['sleep_hours'].fillna(df['sleep_hours'].mean())
```

```
In [15]: print(df['sleep_hours'].isna().sum())
```

```
0
```

```
In [16]: # Function to detect outliers using IQR
```

```
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (column < lower_bound) | (column > upper_bound)
    return outliers

# Apply the function to all numeric columns
numeric_cols = df.select_dtypes(include='number').columns
outlier_df = pd.DataFrame({'Column': numeric_cols, 'Has Outliers': ["Yes" if detect_outliers_iqr(df[col]).any() else "No" for col in numeric_cols]})
print(outlier_df)
```

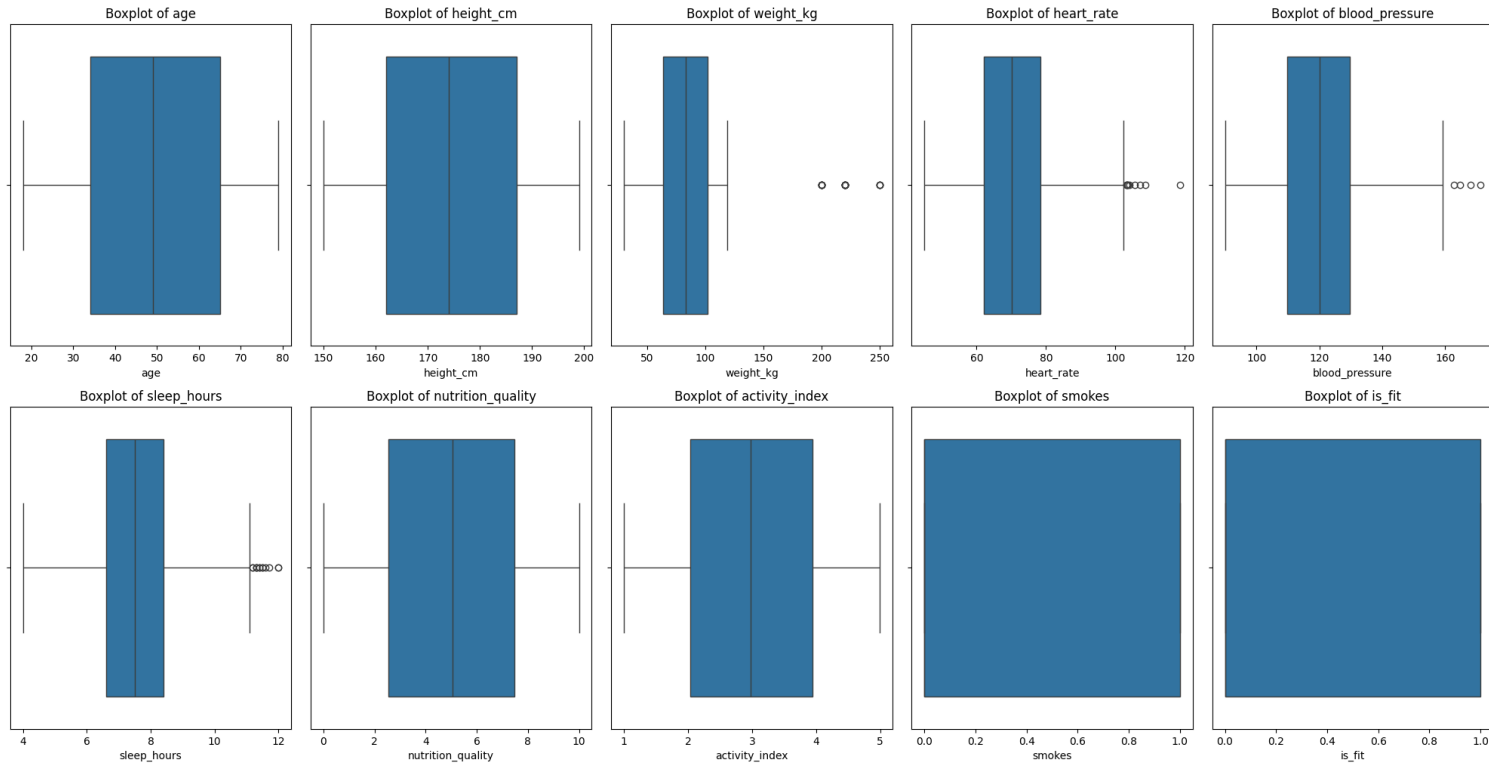
	Column	Has Outliers
0	age	No
1	height_cm	No
2	weight_kg	Yes
3	heart_rate	Yes
4	blood_pressure	Yes
5	sleep_hours	Yes
6	nutrition_quality	No
7	activity_index	No
8	smokes	No
9	is_fit	No

```
In [17]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math

numerical_df = df.select_dtypes(include=['number'])
num_cols = numerical_df.columns
n = len(num_cols)
rows = math.ceil(n / 3) # Number of rows needed
plt.figure(figsize=(20, 5 * rows))

for i, col in enumerate(num_cols):
    plt.subplot(rows, 5, i + 1) # plots per row
    sns.boxplot(x=numerical_df[col])
    plt.title(f"Boxplot of {col}")

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



```
In [18]: data = df.copy()

def remove_outliers_iqr(data):

    # Select only numeric columns
    numeric_data = data.select_dtypes(include='number')

    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each feature
    Q1 = numeric_data.quantile(0.25)
    Q3 = numeric_data.quantile(0.75)
    IQR = Q3 - Q1

    # Define Lower and upper bounds
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove rows with outliers in any numeric column
    data_cleaned = numeric_data[~((numeric_data < lower_bound) | (numeric_data > upper_bound)).any(axis=1)]

    return data_cleaned

cleaned_data = remove_outliers_iqr(data)

print("Original Data length:", len(data))
print("Cleaned Data length (without outliers):", len(cleaned_data))
```

Original Data length: 2000

Cleaned Data length (without outliers): 1953

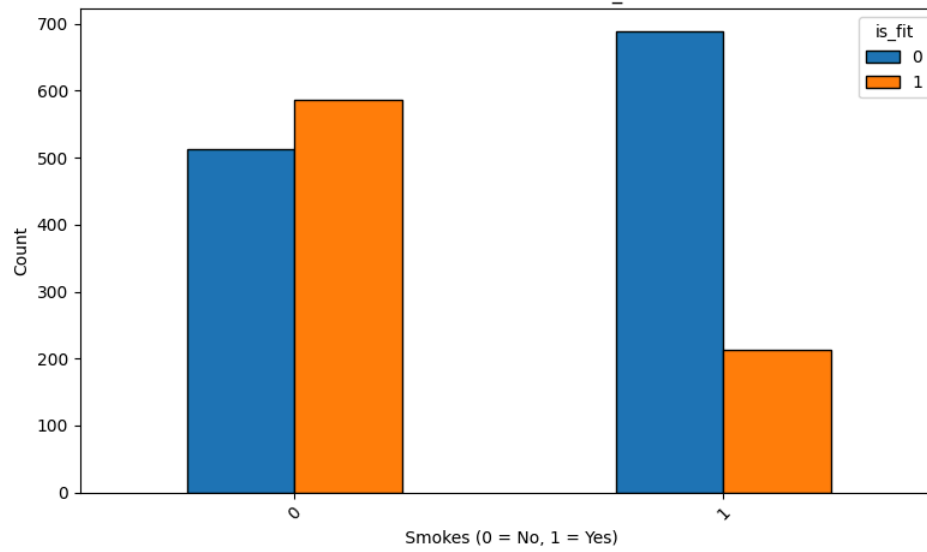
```
In [19]: print("\n" + "*60)
print("PART 2: VISUALIZATION")
print("*60 + "\n")
```

```
=====
PART 2: VISUALIZATION
=====
```

```
In [20]: # Count combinations of two columns
counts = df.groupby(['smokes', 'is_fit']).size().unstack(fill_value=0)

# Plot grouped bar chart
counts.plot(kind='bar', figsize=(8,5), edgecolor='black')
plt.title('Bar Chart: Smokes vs Is_fit')
plt.xlabel('Smokes (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Bar Chart: Smokes vs Is_fit

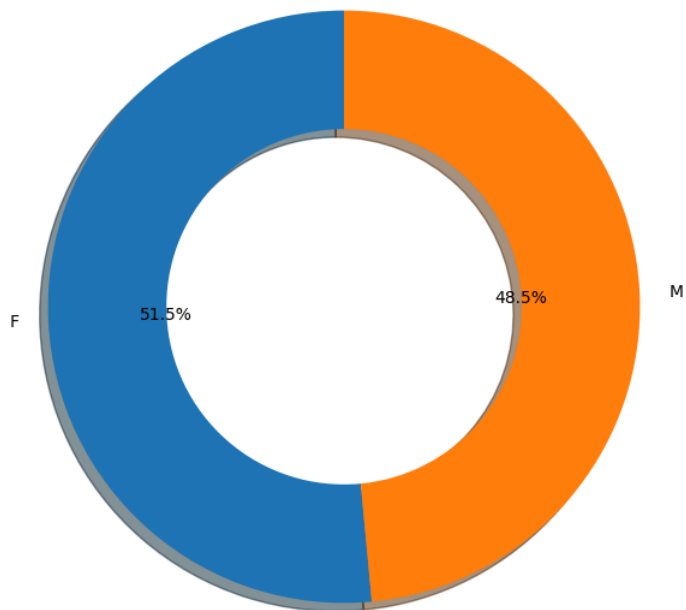


```
In [21]: category_counts = df['gender'].value_counts()

plt.figure(figsize=(8, 8)) # Optional: Adjust figure size

plt.pie(category_counts,
        labels=category_counts.index,
        autopct='%1.1f%%',
        startangle=90,
        shadow=True,
        wedgeprops={'width': 0.4}) # create the donut hole
plt.title('Distribution of Gender (Donut Chart)')
plt.show()
```

Distribution of Gender (Donut Chart)



```
In [22]: plt.hist(data['sleep_hours'], bins=5, color='skyblue', edgecolor='black')
plt.title('Basic Histogram of Sleep Hours')
plt.xlabel('sleep_hours')
plt.ylabel('Frequency')
plt.show()
```

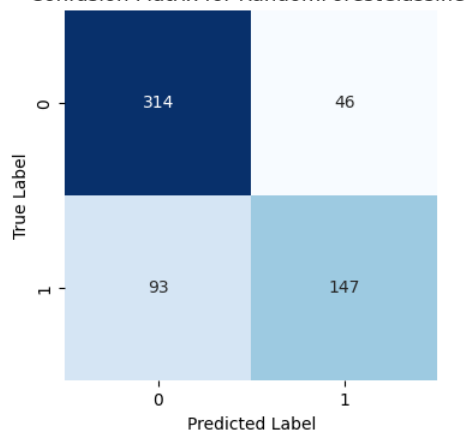


```

# Train the model
rf_classifier.fit(X_train, y_train)
# Make predictions
y_pred = rf_classifier.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('RandomForestClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value)
# Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

Confusion Matrix for RandomForestClassifier



	precision	recall	f1-score	support
0	0.77	0.87	0.82	360
1	0.76	0.61	0.68	240
accuracy			0.77	600
macro avg	0.77	0.74	0.75	600
weighted avg	0.77	0.77	0.76	600

Classification Report:				
	precision	recall	f1-score	support
0	0.77	0.87	0.82	360
1	0.76	0.61	0.68	240
accuracy			0.77	600
macro avg	0.77	0.74	0.75	600
weighted avg	0.77	0.77	0.76	600

LogisticRegression

```

In [30]: LR = LogisticRegression()
# Train the model
LR.fit(X_train, y_train)
# Make predictions
y_pred = LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\linear_model_logistic.py:473: ConvergenceWarning: lbfgs failed to converge after 100 iteration(s) (status=1): STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=100).

You might also want to scale the data as shown in:

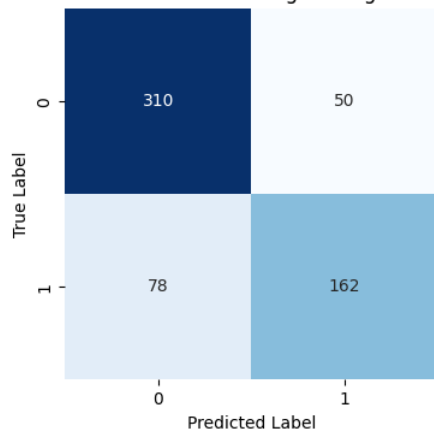
<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

n_iter_i = _check_optimize_result(

Confusion Matrix for LogisticRegression



Model Name:

LogisticRegression

Accuracy: 0.7867

Precision: 0.7850

Recall: 0.7867

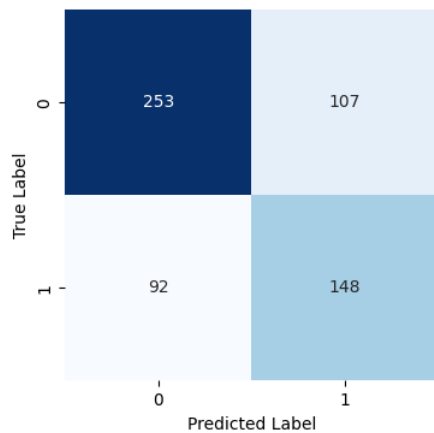
F1 Score: 0.7841

	precision	recall	f1-score	support
0	0.80	0.86	0.83	360
1	0.76	0.68	0.72	240
accuracy			0.79	600
macro avg	0.78	0.77	0.77	600
weighted avg	0.79	0.79	0.78	600

DecisionTreeClassifier

```
In [31]: DT = DecisionTreeClassifier()
# Train the model
DT.fit(X_train, y_train)
# Make predictions
y_pred = DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for DecisionTreeClassifier



Model Name:

DecisionTreeClassifier

Accuracy: 0.6683

Precision: 0.6722

Recall: 0.6683

F1 Score: 0.6698

	precision	recall	f1-score	support
0	0.73	0.70	0.72	360
1	0.58	0.62	0.60	240
accuracy			0.67	600
macro avg	0.66	0.66	0.66	600
weighted avg	0.67	0.67	0.67	600

Support Vector Machine

```
In [32]: SVM = SVC()
# Train the model
SVM.fit(X_train, y_train)
# Make predictions
```

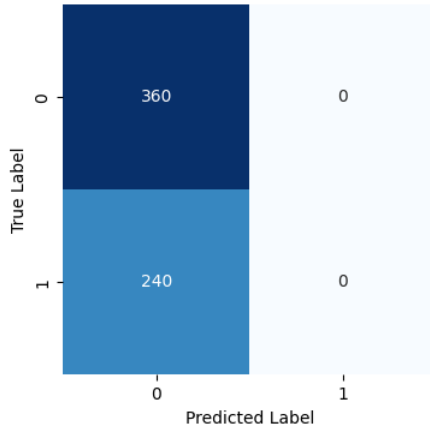
```

y_pred = SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label with no predicted samples. Use `zero\_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label with no predicted samples. Use `zero\_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label with no predicted samples. Use `zero\_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\metrics_classification.py:1731: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in label with no predicted samples. Use `zero\_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])

Confusion Matrix for Support Vector Machine



Model Name:
Support Vector Machine
Accuracy: 0.6000
Precision: 0.3600
Recall: 0.6000
F1 Score: 0.4500

	precision	recall	f1-score	support
0	0.60	1.00	0.75	360
1	0.00	0.00	0.00	240
accuracy			0.60	600
macro avg	0.30	0.50	0.38	600
weighted avg	0.36	0.60	0.45	600

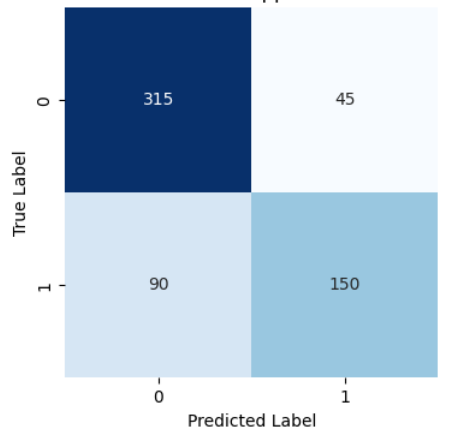
GaussianNB

```

In [33]: gnb = GaussianNB()
# Train the model
gnb.fit(X_train, y_train)
# Make predictions
y_pred = gnb.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")

```

Confusion Matrix for Support Vector Machine



	precision	recall	f1-score	support
0	0.78	0.88	0.82	360
1	0.77	0.62	0.69	240
accuracy			0.78	600
macro avg	0.77	0.75	0.76	600
weighted avg	0.77	0.78	0.77	600

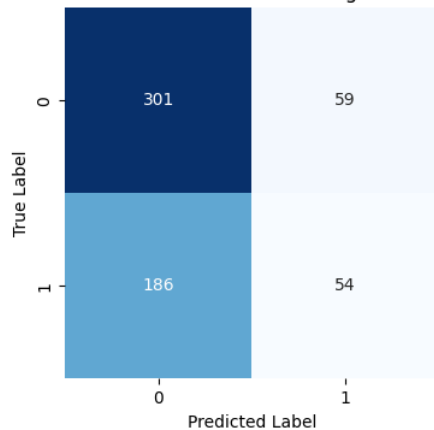
Classification Report:

	precision	recall	f1-score	support
0	0.78	0.88	0.82	360
1	0.77	0.62	0.69	240
accuracy			0.78	600
macro avg	0.77	0.75	0.76	600
weighted avg	0.77	0.78	0.77	600

KNeighborsClassifier

```
In [34]: knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('KNeighbors', y_test, y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
```

Confusion Matrix for KNeighbors



Model Name:

KNeighbors

Accuracy: 0.5917

Precision: 0.5620

Recall: 0.5917

F1 Score: 0.5488

	precision	recall	f1-score	support
0	0.62	0.84	0.71	360
1	0.48	0.23	0.31	240
accuracy			0.59	600
macro avg	0.55	0.53	0.51	600
weighted avg	0.56	0.59	0.55	600

Hyperparameter optimization

Random Forest Classifier with grid search

```
In [35]: from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the hyperparameters for grid search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 1, 3, 5],
    'min_samples_split': [2, 5, 7],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=rf_classifier,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best estimator from grid search
best_rf_classifier = grid_search.best_estimator_

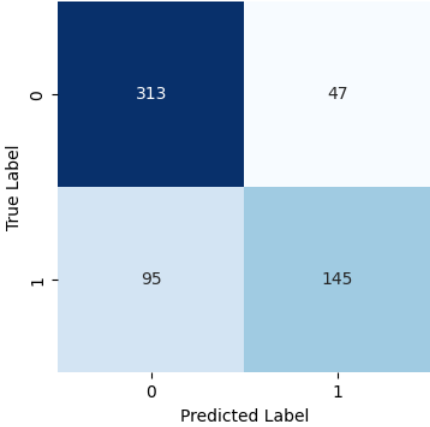
# Make predictions using the best model
y_pred = best_rf_classifier.predict(X_test)

# Evaluate the model
evaluation_results = evaluate_model(
    'RandomForestClassifier',
    y_test,
    y_pred
)

# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
              f"{key}: \n{value}")

# Print the best parameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)
```

Confusion Matrix for RandomForestClassifier



Model Name:
RandomForestClassifier
Accuracy: 0.7633
Precision: 0.7624
Recall: 0.7633
F1 Score: 0.7576

	precision	recall	f1-score	support
0	0.77	0.87	0.82	360
1	0.76	0.60	0.67	240
accuracy			0.76	600
macro avg	0.76	0.74	0.74	600
weighted avg	0.76	0.76	0.76	600

Best hyperparameters found by GridSearchCV:
{'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 300}

LogisticRegression with grid search

```
In [36]: # Define the Logistic Regression model
LR = LogisticRegression(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10], # Regularization strength
    'solver': ['liblinear', 'lbfgs'], # Solver type
    'max_iter': [100, 200, 300] # Maximum iterations for convergence
```

```

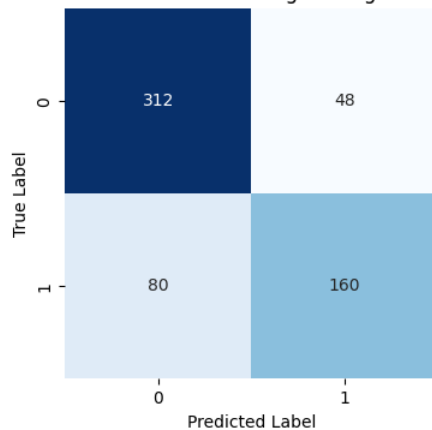
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_LR = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_LR.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('LogisticRegression', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

C:\Users\youse\anaconda3\envs\Coding\Lib\site-packages\sklearn\linear_model_logistic.py:473: ConvergenceWarning: lbfgs failed to converge after 200 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=200).
You might also want to scale the data as shown in:
<https://scikit-learn.org/stable/modules/preprocessing.html>
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
n_iter_i = _check_optimize_result(

Confusion Matrix for LogisticRegression



Model Name:
LogisticRegression
Accuracy: 0.7867
Precision: 0.7852
Recall: 0.7867
F1 Score: 0.7836

	precision	recall	f1-score	support
0	0.80	0.87	0.83	360
1	0.77	0.67	0.71	240
accuracy			0.79	600
macro avg	0.78	0.77	0.77	600
weighted avg	0.79	0.79	0.78	600

Best hyperparameters found by GridSearchCV:
{'C': 1, 'max_iter': 200, 'solver': 'lbfgs'}

DecisionTreeClassifier with grid search

```

In [37]: # Define the Decision Tree model
DT = DecisionTreeClassifier(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'criterion': ['gini', 'entropy'], # The function to measure the quality of a split
    'max_depth': [None, 10, 20, 30], # The maximum depth of the tree
    'min_samples_split': [2, 5, 10], # The minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4], # The minimum number of samples required to be at a leaf node
    'max_features': [None, 'sqrt', 'log2'] # The number of features to consider when looking for the best split
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=DT, param_grid=param_grid, cv=5,
scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_DT = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_DT.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('DecisionTreeClassifier', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':

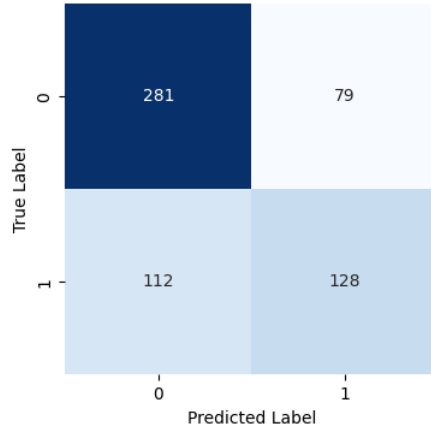
```

```

print(value) # Print report separately for better readability
else:
    print(f"{key}: {value:.4f}" if isinstance(value, float) else f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Confusion Matrix for DecisionTreeClassifier



Model Name:
DecisionTreeClassifier
Accuracy: 0.6817
Precision: 0.6764
Recall: 0.6817
F1 Score: 0.6769

	precision	recall	f1-score	support
0	0.72	0.78	0.75	360
1	0.62	0.53	0.57	240
accuracy			0.68	600
macro avg	0.67	0.66	0.66	600
weighted avg	0.68	0.68	0.68	600

Best hyperparameters found by GridSearchCV:
{'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2}

Support Vector Machine with grid search

```

In [ ]: # Define the Support Vector Machine model
SVM = SVC(random_state=42)
# Define the hyperparameters for grid search
param_grid = {
    'C': [0.01, 0.1, 1, 10], # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'], # Kernel type
    'gamma': ['scale', 'auto'], # Kernel coefficient for 'rbf', 'poly', 'sigmoid'
    'degree': [3, 4, 5], # Degree of the polynomial kernel function ('poly')
    'class_weight': [None, 'balanced'] # Weighing classes in the decision function
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=SVM, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_SVM = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_SVM.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('Support Vector Machine', y_test,
y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

GaussianNB with grid search

```

In [ ]: # Imports
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    classification_report
)

# -----
# Evaluation function (REQUIRED)
# -----
def evaluate_model(model_name, y_true, y_pred):
    return {

```

```

"Model": model_name,
"Accuracy": accuracy_score(y_true, y_pred),
"Precision": precision_score(y_true, y_pred, average='weighted', zero_division=0),
"Recall": recall_score(y_true, y_pred, average='weighted', zero_division=0),
"F1 Score": f1_score(y_true, y_pred, average='weighted', zero_division=0),
"Classification Report": classification_report(y_true, y_pred)
}

# -----
# Gaussian Naive Bayes
# -----
gnb = GaussianNB()

# Hyperparameters for GridSearch
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6]
}

# GridSearchCV
grid_search = GridSearchCV(
    estimator=gnb,
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Fit model
grid_search.fit(X_train, y_train)

# Best model
best_gnb = grid_search.best_estimator_

# Predictions
y_pred = best_gnb.predict(X_test)

# Evaluation
evaluation_results = evaluate_model(
    'Gaussian Naive Bayes',
    y_test,
    y_pred
)

# -----
# Print results
# -----
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print("\nClassification Report:\n")
        print(value)
    else:
        if isinstance(value, float):
            print(f"{key}: {value:.4f}")
        else:
            print(f"{key}: {value}")

# Best hyperparameters
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

KNeighborsClassifier with grid search

```

In [ ]: # Define the K-Nearest Neighbors model
knn = KNeighborsClassifier()
# Define the hyperparameters for grid search
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9], # Number of neighbors to use
    'weights': ['uniform', 'distance'], # Weight function used in prediction
    'metric': ['euclidean', 'manhattan'], # Distance metric
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'] #Algorithm to compute the nearest neighbors
}
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5,
    scoring='accuracy', n_jobs=-1)
# Fit the grid search to the data
grid_search.fit(X_train, y_train)
# Get the best estimator from grid search
best_knn = grid_search.best_estimator_
# Make predictions using the best model
y_pred = best_knn.predict(X_test)
# Evaluate the model
evaluation_results = evaluate_model('K-Nearest Neighbors', y_test,
    y_pred)
# Print the evaluation results
for key, value in evaluation_results.items():
    if key == 'Classification Report':
        print(value) # Print report separately for better readability
    else:
        print(f"{key}: {value:.4f}" if isinstance(value, float) else
            f"{key}: \n{value}")
# Print the best hyperparameters found by grid search
print("\nBest hyperparameters found by GridSearchCV:")
print(grid_search.best_params_)

```

Summary The Best model for prediction the fitness is LogisticRegression

Accuracy: 0.7867