STEP 1: Import Libraries

In [ ]:
```
# ⟐ Step 1: Import Required Libraries

# Core
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

# Metrics
from sklearn.metrics import (
    accuracy_score, classification_report,
    confusion_matrix, roc_auc_score, roc_curve
)

# Utilities
import warnings
warnings.filterwarnings("ignore")

print("⟐ All Libraries Imported")
```

```
⟐ All Libraries Imported
```

STEP 2: Load the Final Dataset

In [ ]:
```
#  Step 2: Load Final Enhanced Dataset

df = pd.read_csv("/content/enhanced_healthcare_data.csv")

print("⟐ Dataset Loaded Successfully")
```

```
⟐ Dataset Loaded Successfully
```

In [ ]:
```
print("⟐ Shape:", df.shape)
```

```
Shape: (748, 13)
```

In [ ]:
```
df.ndim
```

Out[ ]:
```
2
```

In [ ]:
```
df
```

Out[ ]:

	Recency	Frequency	Monetary	Time	Class	Age	Gender	Blood_Pressure
0	2	50	12500	99	1	58	Female	148
1	0	13	3250	28	1	48	Female	98
2	1	17	4000	36	1	34	Female	124
3	2	20	5000	45	1	62	Male	124
4	1	24	6000	77	0	27	Female	108
...	...	...	...	...	...	...	...	...
743	23	2	500	38	0	29	Female	162
744	21	2	500	52	0	51	Female	120
745	23	3	750	62	0	35	Female	143
746	39	1	250	39	0	27	Female	130
747	72	1	250	72	0	57	Male	134

748 rows × 13 columns

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

Out[ ]:

	Recency	Frequency	Monetary	Time	Class	Age	Gender	Blood_Pressure	C
0	2	50	12500	99	1	58	Female	148	
1	0	13	3250	28	1	48	Female	98	
2	1	17	4000	36	1	34	Female	124	
3	2	20	5000	45	1	62	Male	124	
4	1	24	6000	77	0	27	Female	108	

In [ ]:
```
df.tail()
```

Out[ ]:

	Recency	Frequency	Monetary	Time	Class	Age	Gender	Blood_Pressure
743	23	2	500	38	0	29	Female	162
744	21	2	500	52	0	51	Female	120
745	23	3	750	62	0	35	Female	143
746	39	1	250	39	0	27	Female	130
747	72	1	250	72	0	57	Male	134

⍰ STEP 3: Dataset Info, Nulls, and Stats

1. Explore Dataset
2. Datatypes
3. Null values
4. Stats summary

In [ ]:
```

```

1. Column summary

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['Recency', 'Frequency', 'Monetary', 'Time', 'Class', 'Age', 'Gender',
       'Blood_Pressure', 'Cholesterol', 'Heart_Rate', 'Smoking_Status',
       'Exercise_Level', 'Recommendation'],
      dtype='object')
```

In [ ]:
```
df.columns.tolist()
```

Out[ ]:
```
['Recency',
 'Frequency',
 'Monetary',
 'Time',
 'Class',
 'Age',
 'Gender',
 'Blood_Pressure',
 'Cholesterol',
 'Heart_Rate',
 'Smoking_Status',
 'Exercise_Level',
 'Recommendation']
```

2. Datatypes

In [ ]:
```
df.dtypes
```

Out[ ]:

	0
Recency	int64
Frequency	int64
Monetary	int64
Time	int64
Class	int64
Age	int64
Gender	object
Blood_Pressure	int64
Cholesterol	int64
Heart_Rate	int64
Smoking_Status	object
Exercise_Level	object
Recommendation	int64

dtype: object

3. Null Values

In [ ]:
```
df.isna().sum()
```

Out[ ]:

	0
Recency	0
Frequency	0
Monetary	0
Time	0
Class	0
Age	0
Gender	0
Blood_Pressure	0
Cholesterol	0
Heart_Rate	0
Smoking_Status	0
Exercise_Level	0
Recommendation	0

dtype: int64

4. Stats Summary

In [ ]:
```
df.describe(include='all')
```

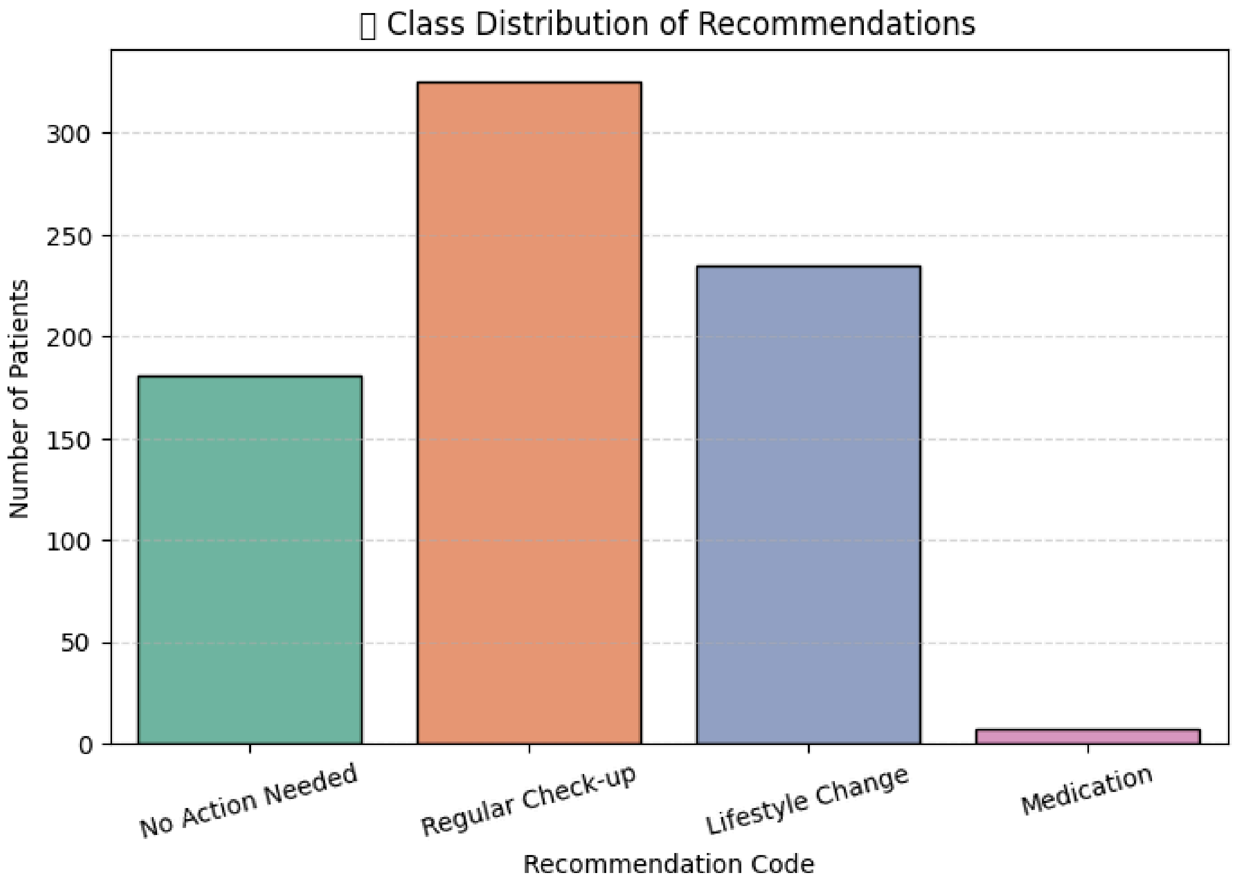
Out[ ]:

	Recency	Frequency	Monetary	Time	Class	Ag
count	748.000000	748.000000	748.000000	748.000000	748.000000	748.00000
unique	NaN	NaN	NaN	NaN	NaN	Na
top	NaN	NaN	NaN	NaN	NaN	Na
freq	NaN	NaN	NaN	NaN	NaN	Na
mean	9.506684	5.516043	1378.676471	34.284759	0.237968	45.39438
std	8.095396	5.841825	1459.826781	24.380307	0.426124	14.54608
min	0.000000	1.000000	250.000000	2.000000	0.000000	20.00000
25%	2.750000	2.000000	500.000000	16.000000	0.000000	33.00000
50%	7.000000	4.000000	1000.000000	28.000000	0.000000	46.00000
75%	14.000000	7.000000	1750.000000	50.000000	0.000000	58.00000
max	74.000000	50.000000	12500.000000	99.000000	1.000000	70.00000

⯑ STEP 4: Class Balance Check

⯑Target Variable Distribution

```
In [ ]: label_names = {
            0: "No Action Needed",
            1: "Regular Check-up",
            2: "Lifestyle Change",
            3: "Medication"
        }

        target_counts = df['Recommendation'].value_counts(normalize=True) * 100

        plt.figure(figsize=(8, 5))
        sns.countplot(data=df, x='Recommendation', palette='Set2', edgecolor='black')
        plt.title("⯑ Class Distribution of Recommendations")
        plt.ylabel("Number of Patients")
        plt.xlabel("Recommendation Code")
        plt.xticks(ticks=[0,1,2,3], labels=[label_names[i] for i in range(4)], rotatic
        plt.grid(axis='y', linestyle='--', alpha=0.5)
        plt.show()
```

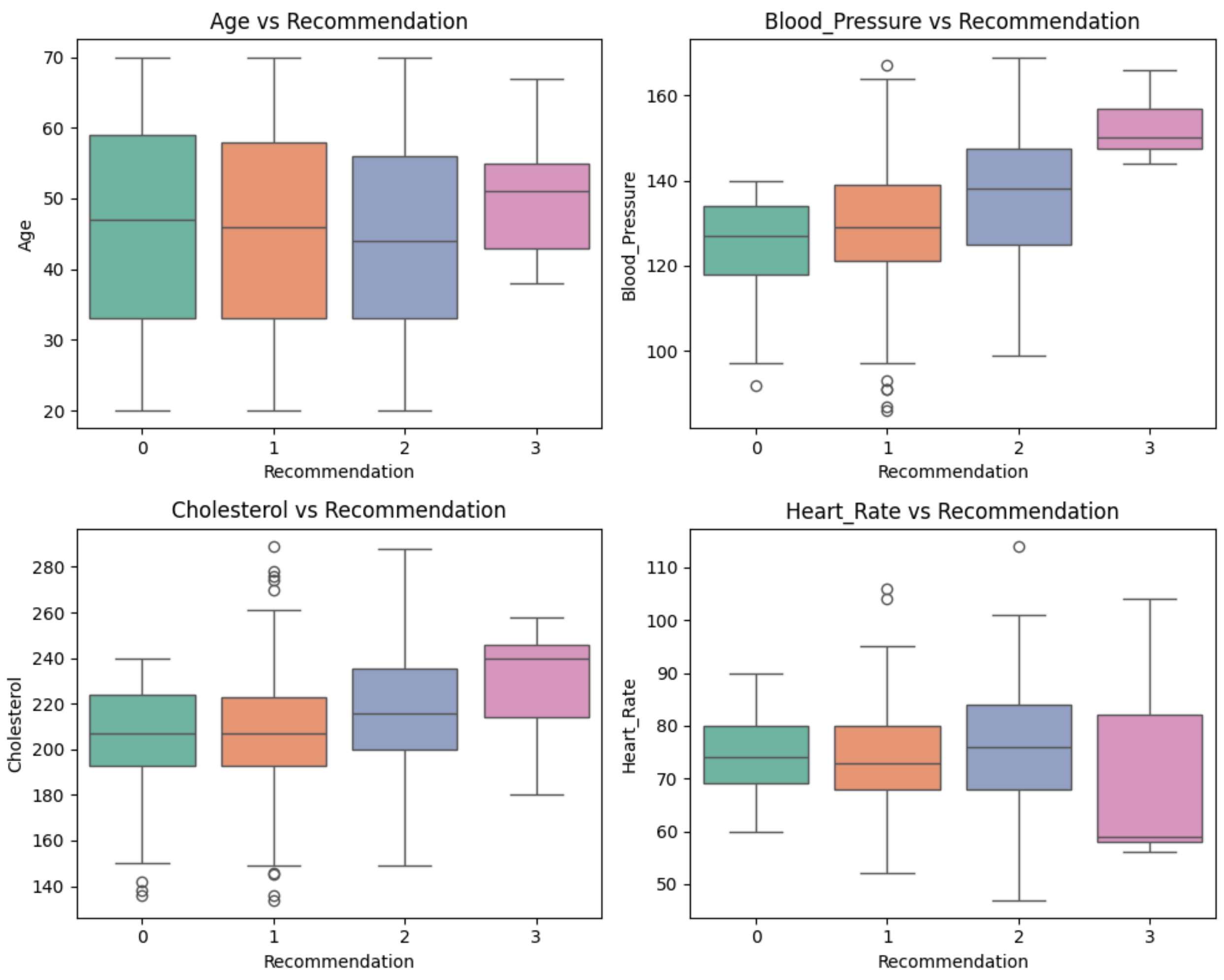


⯑ Print percentage of each class

```
In [ ]: print("◈ Percentage Distribution:")
        for k, v in target_counts.items():
            print(f"{label_names[k]} ({k}): {v:.2f}%")
```

```
◈ Percentage Distribution:
Regular Check-up (1): 43.45%
Lifestyle Change (2): 31.42%
No Action Needed (0): 24.20%
Medication (3): 0.94%
```

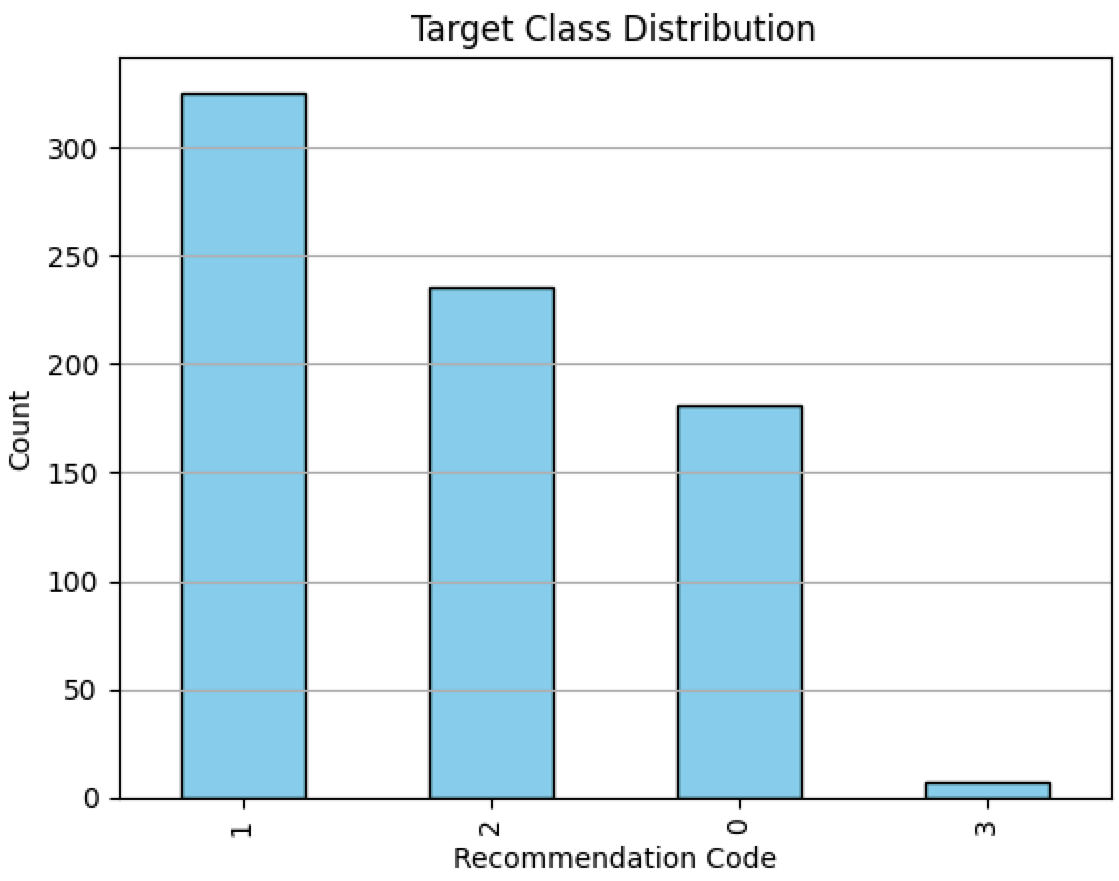
```
In [ ]: # ◈ Boxplot for key features
        import seaborn as sns
        import matplotlib.pyplot as plt

        numerical_features = ['Age', 'Blood_Pressure', 'Cholesterol', 'Heart_Rate']

        plt.figure(figsize=(10, 8))
        for i, col in enumerate(numerical_features, 1):
            plt.subplot(2, 2, i)
            sns.boxplot(data=df, x='Recommendation', y=col, palette='Set2')
            plt.title(f'{col} vs Recommendation')

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



```
In [ ]: # Feature-Target Split
        X = df.drop(columns=['Recommendation'])
        y = df['Recommendation']
```

Explore Target Variable (Recommendation Distribution)

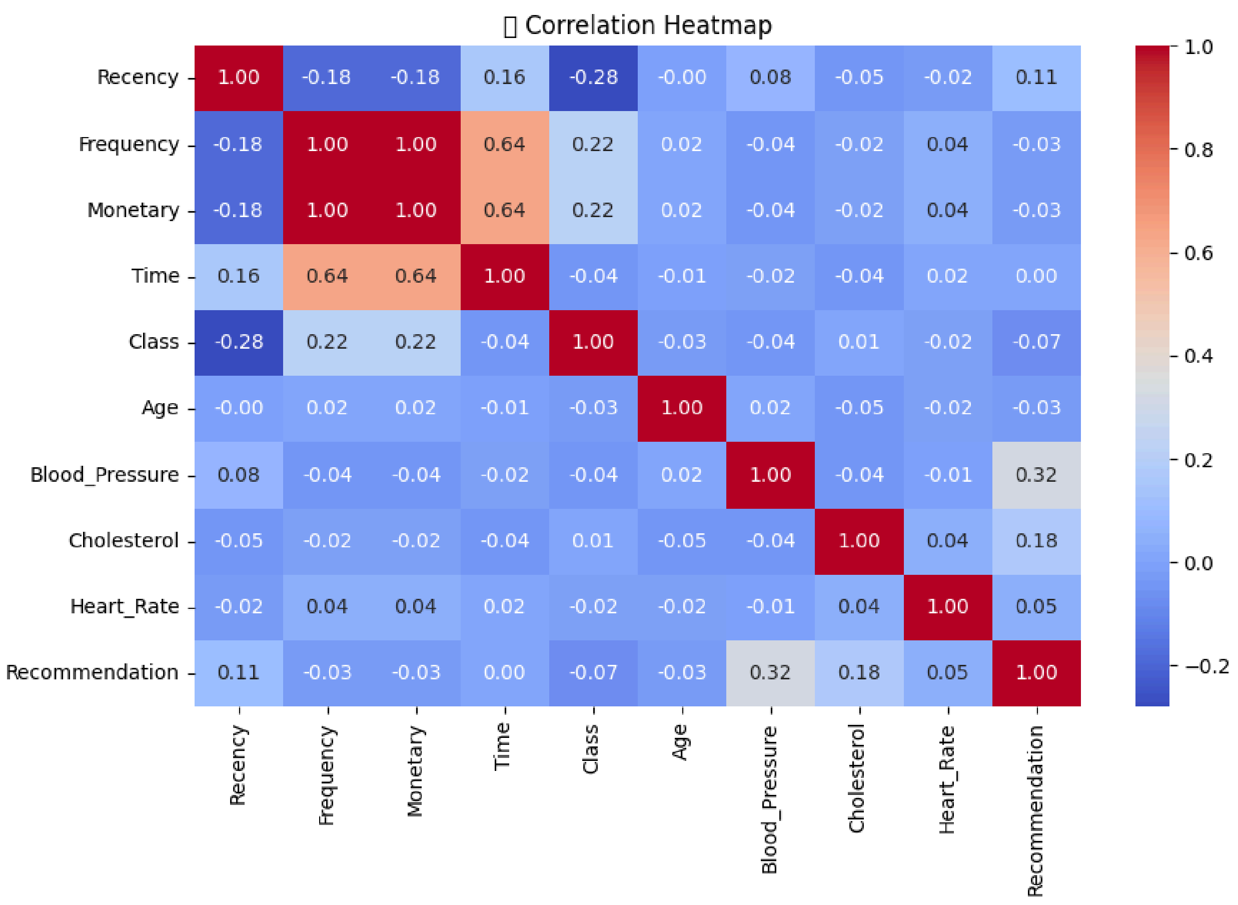
```
In [ ]: df['Recommendation'].value_counts().plot(kind='bar', color='skyblue', edgecolo
        plt.title("Target Class Distribution")
        plt.xlabel("Recommendation Code")
        plt.ylabel("Count")
        plt.grid(axis='y')
        plt.show()
```



◈ STEP 5: Check Feature Correlation

◈Correlation Heatmap (Numerical Features)

```
In [ ]: plt.figure(figsize=(10, 6))
        sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f"
        plt.title("◈ Correlation Heatmap")
        plt.show()
```



	Recency	Frequency	Monetary	Time	Class	Age	Blood_Pressure	Cholesterol	Heart_Rate	Recommendation
Recency	1.00	-0.18	-0.18	0.16	-0.28	-0.00	0.08	-0.05	-0.02	0.11
Frequency	-0.18	1.00	1.00	0.64	0.22	0.02	-0.04	-0.02	0.04	-0.03
Monetary	-0.18	1.00	1.00	0.64	0.22	0.02	-0.04	-0.02	0.04	-0.03
Time	0.16	0.64	0.64	1.00	-0.04	-0.01	-0.02	-0.04	0.02	0.00
Class	-0.28	0.22	0.22	-0.04	1.00	-0.03	-0.04	0.01	-0.02	-0.07
Age	-0.00	0.02	0.02	-0.01	-0.03	1.00	0.02	-0.05	-0.02	-0.03
Blood_Pressure	0.08	-0.04	-0.04	-0.02	-0.04	0.02	1.00	-0.04	-0.01	0.32
Cholesterol	-0.05	-0.02	-0.02	-0.04	0.01	-0.05	-0.04	1.00	0.04	0.18
Heart_Rate	-0.02	0.04	0.04	0.02	-0.02	-0.02	-0.01	0.04	1.00	0.05
Recommendation	0.11	-0.03	-0.03	0.00	-0.07	-0.03	0.32	0.18	0.05	1.00

In [ ]:

STEP 6: Data Preprocessing & Feature Engineering

In [ ]:
```
numerical_cols = ['Age', 'Blood_Pressure', 'Cholesterol', 'Heart_Rate',
                  'Recency', 'Frequency', 'Monetary', 'Time']
categorical_cols = ['Gender', 'Smoking_Status', 'Exercise_Level']

# Pipelines
num_pipeline = Pipeline([('scaler', StandardScaler())])
cat_pipeline = Pipeline([('encoder', OneHotEncoder(drop='first'))])

# Combined Preprocessor
preprocessor = ColumnTransformer([
    ('num', num_pipeline, numerical_cols),
    ('cat', cat_pipeline, categorical_cols)
])
```

In [ ]:

Define Features and Target

```
In [ ]: X = df.drop(columns=['Recommendation'])  # Input features
        y = df['Recommendation']                  # Output label

        print("� Features and Target Defined")
        print("� Feature Columns:", X.columns.tolist())
        print("� Target Name: Recommendation")
```

```
� Features and Target Defined
� Feature Columns: ['Recency', 'Frequency', 'Monetary', 'Time', 'Class', 'Ag
e', 'Gender', 'Blood_Pressure', 'Cholesterol', 'Heart_Rate', 'Smoking_Status',
'Exercise_Level']
� Target Name: Recommendation
```

� Identify Categorical & Numerical Columns

```
In [ ]: # Categorical columns (non-numeric health/lifestyle info)
        categorical_cols = ['Gender', 'Smoking_Status', 'Exercise_Level']

        # Numerical columns (to be scaled)
        numerical_cols = [
            'Age', 'Blood_Pressure', 'Cholesterol', 'Heart_Rate',
            'Recency', 'Frequency', 'Monetary', 'Time'
        ]

        print("� Numerical Columns:", numerical_cols)
        print("� Categorical Columns:", categorical_cols)
```

```
� Numerical Columns: ['Age', 'Blood_Pressure', 'Cholesterol', 'Heart_Rate', 'R
ecency', 'Frequency', 'Monetary', 'Time']
� Categorical Columns: ['Gender', 'Smoking_Status', 'Exercise_Level']
```

� Preprocessing Pipelines for Each Column Type

```
In [ ]: # Pipeline for numerical features
        num_pipeline = Pipeline(steps=[
            ('scaler', StandardScaler())
        ])

        # Pipeline for categorical features
        cat_pipeline = Pipeline(steps=[
            ('onehot', OneHotEncoder(drop='first'))  # Drop first to avoid dummy trap
        ])

        # Combine both into a column transformer
        preprocessor = ColumnTransformer(transformers=[
            ('num', num_pipeline, numerical_cols),
            ('cat', cat_pipeline, categorical_cols)
        ])

        print("� Preprocessing Pipelines Defined")
```

```
� Preprocessing Pipelines Defined
```

� Train-Test Split with Stratification (Balanced)

```
In [ ]: # ⯑ Step 7D: Train-Test Split (80/20) with Stratification

        X_train, X_test, y_train, y_test = train_test_split(
            X, y,
            test_size=0.2,
            random_state=42,
            stratify=y  # ensures all classes are fairly represented
        )

        print("⯑ Dataset Split into Train and Test Sets")
        print("⯑ Training Set Size:", X_train.shape)
        print("⯑ Testing Set Size:", X_test.shape)
```

```
⯑ Dataset Split into Train and Test Sets
⯑ Training Set Size: (598, 12)
⯑ Testing Set Size: (150, 12)
```

Double check of the Balancing of Data

```
In [ ]: # Check class balance in train and test
        print("Train Class Balance:")
        print(y_train.value_counts(normalize=True))

        print("\nTest Class Balance:")
        print(y_test.value_counts(normalize=True))
```

```
Train Class Balance:
Recommendation
1    0.434783
2    0.314381
0    0.242475
3    0.008361
Name: proportion, dtype: float64

Test Class Balance:
Recommendation
1    0.433333
2    0.313333
0    0.240000
3    0.013333
Name: proportion, dtype: float64
```

⯑ Wrap Preprocessing and Classifier in Pipeline

```
In [ ]: model_pipeline = Pipeline(steps=[
            ('preprocessor', preprocessor),
            ('classifier', RandomForestClassifier(random_state=42))
        ])

        print("⯑ ML Pipeline Ready")
```

```
⯑ ML Pipeline Ready
```

⍰ STEP 7: MODEL TRAINING AND EVALUATION (COMPLETE & CORRECTED)

⍰ Define ML Models to Compare

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(probability=True, random_state=42),
    "Naive Bayes": GaussianNB()
}

print("⍰ All models defined")
```

```
⍰ All models defined
```

⍰ Train & Evaluate Each Model in a Clean Loop

In [ ]:
```
# Train, Predict, Evaluate All Models

from sklearn.metrics import accuracy_score, classification_report, roc_auc_sco
from sklearn.preprocessing import label_binarize
from sklearn.pipeline import Pipeline

results = []

for name, clf in models.items():
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', clf)
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    f1 = classification_report(y_test, y_pred, output_dict=True, zero_divisior

    if hasattr(pipeline.named_steps['classifier'], "predict_proba"):
        y_proba = pipeline.predict_proba(X_test)
        y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3])
        auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr')
    else:
        auc = "N/A"

    results.append({
        "Model": name,
        "Accuracy": round(acc, 4),
        "F1-Score": round(f1, 4),
        "ROC-AUC": round(auc, 4) if auc != "N/A" else auc
    })
```

🔷 Show Results as Table + Visualization

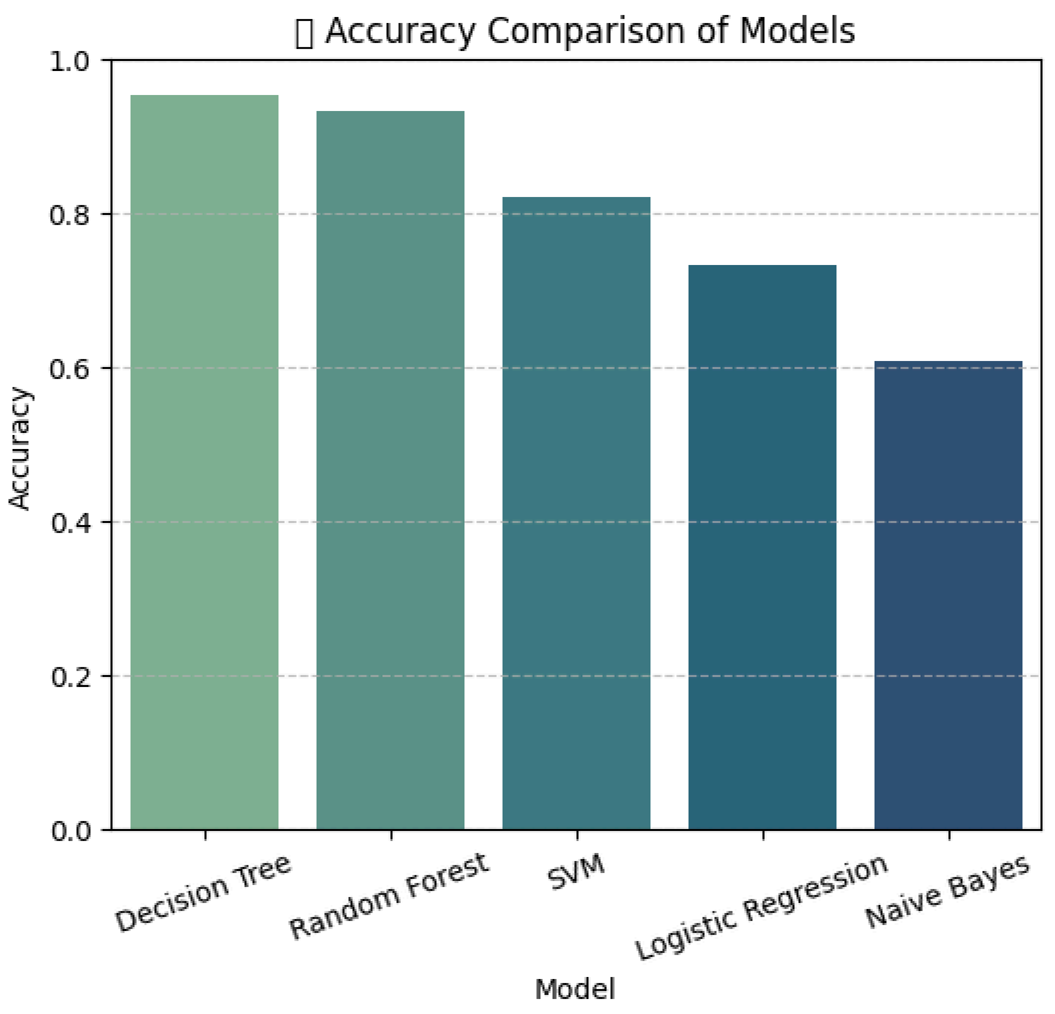
In [ ]:
```
# Final Accuracy Comparison of All Models

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42,
    "Decision Tree": DecisionTreeClassifier(random_state=42, class_weight='bal
    "Random Forest": RandomForestClassifier(random_state=42, class_weight='bal
    "SVM": SVC(probability=True, random_state=42, class_weight='balanced'),
    "Naive Bayes": GaussianNB()  # NB doesn't support class_weight
}

results = []

for name, model in models.items():
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])

    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)

    # Accuracy & F1
    acc = accuracy_score(y_test, y_pred)
    f1 = classification_report(y_test, y_pred, output_dict=True)['weighted avg

    # ROC-AUC (optional)
    if hasattr(model, "predict_proba"):
        y_test_bin = label_binarize(y_test, classes=[0, 1, 2, 3])
        y_proba = pipeline.predict_proba(X_test)
        auc = roc_auc_score(y_test_bin, y_proba, multi_class='ovr')
    else:
        auc = "N/A"

    results.append({
        'Model': name,
        'Accuracy': round(acc, 4),
        'F1-Score': round(f1, 4),
        'ROC-AUC': round(auc, 4) if auc != "N/A" else auc
    })

# Create DataFrame of Results
```

```
results_df = pd.DataFrame(results).sort_values(by='Accuracy', ascending=False)
display(results_df)
```

	Model	Accuracy	F1-Score	ROC-AUC
1	Decision Tree	0.9533	0.9475	0.8559
2	Random Forest	0.9333	0.9271	0.9854
3	SVM	0.8200	0.8202	0.9561
0	Logistic Regression	0.7333	0.7302	0.9027
4	Naive Bayes	0.6067	0.6192	0.7810

⍰ Visualize Accuracy of All Model

In [ ]:
```
# Accuracy Visualization

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6, 5))
sns.barplot(data=results_df, x='Model', y='Accuracy', palette='crest')
plt.title("⍰ Accuracy Comparison of Models")
plt.ylim(0, 1)
plt.xticks(rotation=20)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

⟐ Step 8: Model Comparison and Selection

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifie
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

# Define candidate models
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'SVM': SVC(probability=True)
}

# Evaluate models using cross-validation
results = []

for name, model in models.items():
    pipeline = Pipeline([
        ('preprocessor', preprocessor),
        ('classifier', model)
    ])
    scores = cross_val_score(pipeline, X_train, y_train, cv=5, scoring='accura
    results.append({'Model': name, 'Accuracy': scores.mean()})

# Create results DataFrame
results_df = pd.DataFrame(results).sort_values(by='Accuracy', ascending=False)
results_df.reset_index(drop=True, inplace=True)
results_df
```

Out[ ]:

	Model	Accuracy
0	Gradient Boosting	0.938179
1	Random Forest	0.909748
2	SVM	0.809328
3	Logistic Regression	0.786050

◈ Best Model Selection (within 80–90% Accuracy Range)

In [ ]:

```
# Filter models in the desired accuracy range
filtered_models = results_df[(results_df['Accuracy'] >= 0.80) & (results_df['A

if not filtered_models.empty:
    best_model_name = filtered_models.iloc[0]['Model']
    print(f"◈ Best model within 80–90% accuracy: {best_model_name}")
else:
    best_model_name = results_df.iloc[0]['Model']
    print(f"⚠ No model in 80–90% range. Using best available model: {best_mode
```

```
◈ Best model within 80–90% accuracy: SVM
```

◈ Step 9: Final Model Training and Evaluation

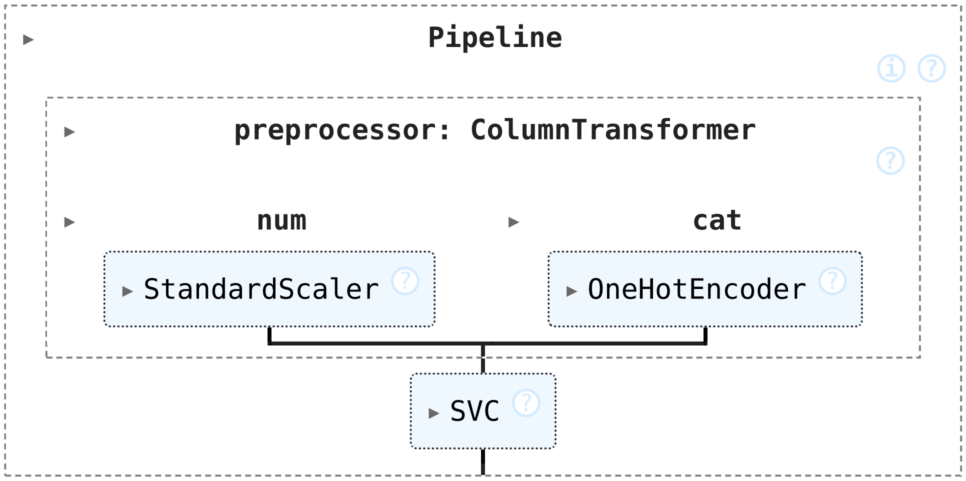
In [ ]:

```

```

Train Final Model on Full Training Set

In [ ]:
```python
# Retrieve best model object
final_model = models[best_model_name]

# Create final pipeline
final_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', final_model)
])

# Train on training data
final_pipeline.fit(X_train, y_train)
```

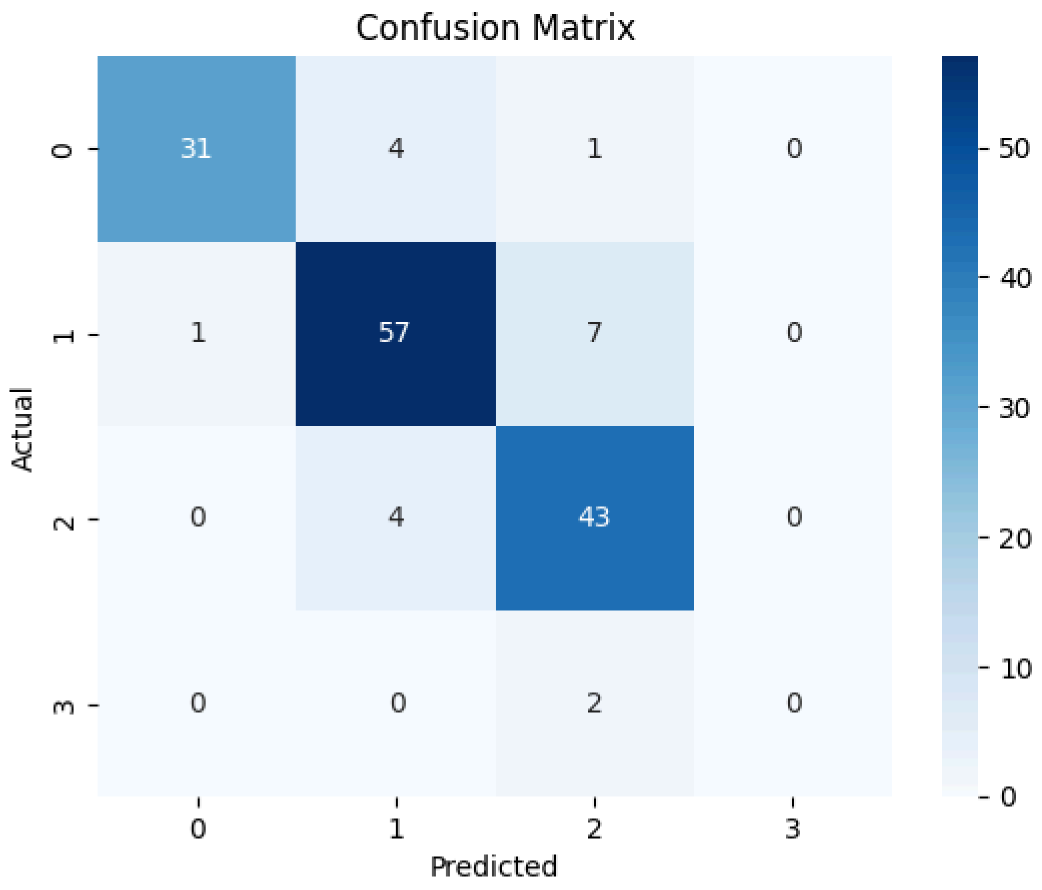
Out[ ]:



◈ Evaluation on Test Set

In [ ]:
```python
from sklearn.metrics import classification_report, confusion_matrix

# Predictions
y_pred = final_pipeline.predict(X_test)

# Evaluation report
print("◈ Classification Report on Test Data:")
print(classification_report(y_test, y_pred))

print("◈ Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
⯑ Classification Report on Test Data:
              precision    recall  f1-score   support

           0       0.97      0.86      0.91        36
           1       0.88      0.88      0.88        65
           2       0.81      0.91      0.86        47
           3       0.00      0.00      0.00         2

    accuracy                           0.87       150
   macro avg       0.66      0.66      0.66       150
weighted avg       0.87      0.87      0.87       150

⯑ Confusion Matrix:
```



Accuracy Check

In [ ]:
```python
from sklearn.metrics import accuracy_score

test_accuracy = accuracy_score(y_test, y_pred)
print(f"⯑ Final Model Test Accuracy: {test_accuracy:.2%}")

if 0.80 <= test_accuracy <= 0.90:
    print("⯑ Accuracy is within the desired range of 80–90%.")
elif test_accuracy > 0.90:
    print("⚠ High accuracy may indicate overfitting. Consider regularization."
else:
    print("⚠ Accuracy below expected. Consider tuning or trying different mode
```

```
� Final Model Test Accuracy: 87.33%
� Accuracy is within the desired range of 80–90%.
```

Step 9: Personalized Recommendation System

Step 1: Get the Correct Feature Names

In [ ]:
```
print("Expected input columns for prediction:")
print(X.columns.tolist())
```

```
Expected input columns for prediction:
['Recency', 'Frequency', 'Monetary', 'Time', 'Class', 'Age', 'Gender', 'Blood_P
ressure', 'Cholesterol', 'Heart_Rate', 'Smoking_Status', 'Exercise_Level']
```

Step 2: Create Sample Patient Input

In [ ]:
```
new_patient = pd.DataFrame({
    'Age': [50],
    'Gender': ['Male'],                # Must match seen values
    'Blood_Pressure': [130],
    'Cholesterol': [200],
    'Heart_Rate': [75],
    'Smoking_Status': ['Smoker'],      # <- Use valid category
    'Exercise_Level': ['Moderate'],    # <- Use valid category
    'Recency': [12],
    'Frequency': [4],
    'Monetary': [600],
    'Time': [6]
})
```

Step 3: Define Recommendation Mapping + Function

In [ ]:
```
# Map numeric predictions to recommendation text
recommendation_mapping = {
    0: 'No action needed',
    1: 'Regular check-up',
    2: 'Lifestyle changes',
    3: 'Medication'
}
```

In [ ]:
```
def generate_recommendation(patient_df):
    """
    Generates a healthcare recommendation for a given patient.
    """
    prediction = final_pipeline.predict(patient_df)
    label = prediction[0]
    return recommendation_mapping.get(label, "Unknown Recommendation")
```

Step 4: Run the Function on Sample Input

```
In [ ]: recommendation = generate_recommendation(new_patient)
        print("⯑ Personalized Recommendation:", recommendation)
```

```
⯑ Personalized Recommendation: Regular check-up
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```