```
import pandas as pd
```

```
df = pd.read_csv('/content/heart_disease_uci.csv')
```

```
df.isnull().sum()
```

	0
id	0
age	0
sex	0
dataset	0
cp	0
trestbps	59
chol	30
fbs	90
restecg	2
thalch	55
exang	55
oldpeak	62
slope	309
ca	611
thal	486
num	0

dtype: int64

```
# Impute numerical columns with median
for col in ['trestbps', 'chol', 'thalch', 'oldpeak']:
    if df[col].isnull().any():
        df[col] = df[col].fillna(df[col].median())

# Impute categorical columns with mode
for col in ['fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']:
    if df[col].isnull().any():
        df[col] = df[col].fillna(df[col].mode()[0])

print("Missing values after imputation:")
print(df.isnull().sum())
```

```
Missing values after imputation:
id          0
age         0
sex         0
dataset     0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalch      0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
num         0
dtype: int64
```

```
df['sex'] = df['sex'].map({'Male': 0, 'Female': 1})
print(df['sex'].unique())
```

```
[0 1]
```

```
df.head()
```

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal
0	1	63	0	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect
1	2	67	0	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal
2	3	67	0	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect
3	4	37	0	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal
4	5	41	1	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal

Next steps: Generate code with df New interactive sheet

```
# Identify categorical columns to encode (excluding 'sex' which is already handled)
categorical_cols = df.select_dtypes(include='object').columns.tolist()

# Apply one-hot encoding
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True, dtype=int)

print("DataFrame after one-hot encoding:")
display(df.head())
print("\nDataFrame Info after one-hot encoding:")
df.info()
```

```
hot encoding:
```

stbps	chol	fbs	thalch	exang	oldpeak	ca	...	dataset_VA Long Beach	cp_atypical angina	cp_non-anginal	cp_typical angina	restecg
145.0	233.0	True	150.0	False	2.3	0.0	...	0	0	0	1	
160.0	286.0	False	108.0	True	1.5	3.0	...	0	0	0	0	
120.0	229.0	False	129.0	True	2.6	2.0	...	0	0	0	0	
130.0	250.0	False	187.0	False	3.5	0.0	...	0	0	1	0	
130.0	204.0	False	172.0	False	1.4	0.0	...	0	1	0	0	

```
 one-hot encoding:
frame.DataFrame'>
ies, 0 to 919
23 columns):
           Non-Null Count  Dtype
           --------------  -----
           920 non-null    int64
           920 non-null    int64
           920 non-null    int64
           920 non-null    float64
           920 non-null    float64
           920 non-null    bool
           920 non-null    float64
           920 non-null    bool
           920 non-null    float64
           920 non-null    float64
           920 non-null    int64
           920 non-null    int64
land       920 non-null    int64
 Beach     920 non-null    int64
ina        920 non-null    int64
           920 non-null    int64
na         920 non-null    int64
           920 non-null    int64
normality  920 non-null    int64
           920 non-null    int64
           920 non-null    int64
           920 non-null    int64
 defect    920 non-null    int64
at64(5), int64(16)
KB
```

Task

To prepare the data for the SVM model, I'll first define the features (X) and the target variable (y). The target variable 'num' will be converted into a binary classification problem (0 for no heart disease, 1 for heart disease). Then, I'll split the dataset into training and testing sets using an 80/20 ratio.

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
# Drop 'id' column as it's just an identifier
X = df.drop(columns=['id', 'num'])

# Convert 'num' column into a binary target: 0 for no disease, 1 for disease
y = (df['num'] > 0).astype(int)

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"Shape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")

print("\nDistribution of target variable in training set:")
print(y_train.value_counts(normalize=True))
print("\nDistribution of target variable in test set:")
print(y_test.value_counts(normalize=True))
```

Prepare Data for SVM

Subtask:

Split the preprocessed DataFrame into features (X) and target (y). Then, divide the data into training and testing sets using an 80/20 ratio.

Reasoning: To prepare the data for model training, I will split the preprocessed DataFrame into features (X) and target (y), and then divide them into training and testing sets using an 80/20 ratio as per the instructions.

```
    from sklearn.model_selection import train_test_split

    # Define features (X) by dropping 'id' and 'num' columns
    X = df.drop(['id', 'num'], axis=1)

    # Create the target variable (y) by converting 'num' to a binary target
    y = (df['num'] > 0).astype(int)

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

    print("Shapes after splitting:")
    print(f"X_train shape: {X_train.shape}")
    print(f"X_test shape: {X_test.shape}")
    print(f"y_train shape: {y_train.shape}")
    print(f"y_test shape: {y_test.shape}")

    print("\nNormalized value counts for y_train:")
    print(y_train.value_counts(normalize=True))

    print("\nNormalized value counts for y_test:")
    print(y_test.value_counts(normalize=True))
```

Train SVM with Polynomial Kernel (Degree 2 & 3)

Subtask:

Initialize and train an SVM classifier using a polynomial kernel with degrees 2 and 3. Evaluate its performance on the test set using accuracy and F1-score.

Reasoning: I need to import the necessary modules, initialize and train two SVM models with polynomial kernels of degree 2 and 3, respectively, and then evaluate their performance using accuracy and F1-score on the test set.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score

# Initialize and train SVM with polynomial kernel, degree 2
svm_poly_degree2 = SVC(kernel='poly', degree=2, random_state=42)
svm_poly_degree2.fit(X_train, y_train)

# Make predictions and evaluate for degree 2
y_pred_degree2 = svm_poly_degree2.predict(X_test)
accuracy_degree2 = accuracy_score(y_test, y_pred_degree2)
f1_degree2 = f1_score(y_test, y_pred_degree2)

print(f"SVM with Polynomial Kernel (Degree 2) - Accuracy: {accuracy_degree2:.4f}, F1-score: {f1_degree2:.4f}")

# Initialize and train SVM with polynomial kernel, degree 3
svm_poly_degree3 = SVC(kernel='poly', degree=3, random_state=42)
svm_poly_degree3.fit(X_train, y_train)

# Make predictions and evaluate for degree 3
y_pred_degree3 = svm_poly_degree3.predict(X_test)
accuracy_degree3 = accuracy_score(y_test, y_pred_degree3)
f1_degree3 = f1_score(y_test, y_pred_degree3)

print(f"SVM with Polynomial Kernel (Degree 3) - Accuracy: {accuracy_degree3:.4f}, F1-score: {f1_degree3:.4f}")
```

```
SVM with Polynomial Kernel (Degree 2) - Accuracy: 0.7554, F1-score: 0.7907
SVM with Polynomial Kernel (Degree 3) - Accuracy: 0.7446, F1-score: 0.7873
```

Train SVM with RBF Kernel

Subtask:

Initialize and train an SVM classifier using an RBF (Radial Basis Function) kernel. Evaluate its performance on the test set using accuracy and F1-score.

Reasoning: I need to initialize an SVM classifier with an RBF kernel, train it on the training data, make predictions on the test data, and then evaluate its performance using accuracy and F1-score.

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score

# Initialize and train SVM with RBF kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_train, y_train)

# Make predictions
y_pred_rbf = svm_rbf.predict(X_test)

# Evaluate performance
accuracy_rbf = accuracy_score(y_test, y_pred_rbf)
f1_rbf = f1_score(y_test, y_pred_rbf)

print(f"SVM with RBF Kernel - Accuracy: {accuracy_rbf:.4f}, F1-score: {f1_rbf:.4f}")
```

```
SVM with RBF Kernel - Accuracy: 0.7391, F1-score: 0.7714
```

Compare Initial Model Performance

Subtask:

Present a comparison of the accuracy and F1-score for the SVM models trained with Polynomial (degree 2 & 3) and RBF kernels on the test set.

Reasoning: The previous steps have already computed the accuracy and F1-score for each SVM model. I will now present these metrics in a structured way as requested by the instructions.

```
print(f"SVM with Polynomial Kernel (Degree 2) - Accuracy: {accuracy_degree2:.4f}, F1-score: {f1_degree2:.4f}")
print(f"SVM with Polynomial Kernel (Degree 3) - Accuracy: {accuracy_degree3:.4f}, F1-score: {f1_degree3:.4f}")
print(f"SVM with RBF Kernel - Accuracy: {accuracy_rbf:.4f}, F1-score: {f1_rbf:.4f}")
```

```
SVM with Polynomial Kernel (Degree 2) - Accuracy: 0.7554, F1-score: 0.7907
SVM with Polynomial Kernel (Degree 3) - Accuracy: 0.7446, F1-score: 0.7873
SVM with RBF Kernel - Accuracy: 0.7391, F1-score: 0.7714
```

Model Performance Comparison

Model	Accuracy	F1-score
SVM with Polynomial Kernel (Degree 2)	0.7554	0.7907
SVM with Polynomial Kernel (Degree 3)	0.7446	0.7873
SVM with RBF Kernel	0.7391	0.7714

Based on these results, the **SVM with Polynomial Kernel (Degree 2)** performed best, achieving the highest accuracy (0.7554) and F1-score (0.7907) among the evaluated models.

Hyperparameter Tuning for Polynomial Kernel

Subtask:

Perform GridSearchCV to find the optimal 'C' and 'gamma' hyperparameters for the SVM with a polynomial kernel. Report the best parameters and corresponding accuracy.

Reasoning: I need to perform hyperparameter tuning for the SVM with a polynomial kernel using GridSearchCV. This involves importing the necessary module, defining a parameter grid for 'C' and 'gamma', initializing the SVC and GridSearchCV objects, fitting the grid search to the training data, and then reporting the best parameters and corresponding accuracy.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# Define the parameter grid for GridSearchCV
param_grid_poly = {
    'C': [0.1, 1, 10,1,1],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3] # Include degree as it's a poly kernel, using 2 and 3 as previously explored
}

# Initialize SVC with polynomial kernel
svm_poly = SVC(kernel='poly', random_state=42)

# Initialize GridSearchCV
grid_search_poly = GridSearchCV(estimator=svm_poly, param_grid=param_grid_poly, scoring='accuracy', cv=5, n_jobs=-1, verbose

# Fit GridSearchCV to the training data
grid_search_poly.fit(X_train, y_train)

# Print the best parameters and best score
print("Best parameters for Polynomial Kernel:", grid_search_poly.best_params_)
print("Best cross-validation accuracy for Polynomial Kernel:", grid_search_poly.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```