In [1]:
```
import pandas as pd
import pycaret
```

In [2]:
```
from pycaret.classification import *
```

In [3]:
```
def load_data(file_path):
    """
    Function to load data from file_path.
    """
    # Implement code to load data using pandas
    data = pd.read_csv(file_path)
    return data
```

In [4]:
```
data = load_data('heart.csv')
```

In [5]:
```
def read_csv_file(file_path):
    return pd.read_csv(file_path)
```

In [7]:
```
file_path = 'heart.csv'  # Replace 'train.csv' with the path to your CSV file
df = read_csv_file(file_path)
print(df)
```

```
      age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0      52    1   0       125   212    0        1      168      0      1.0
1      53    1   0       140   203    1        0      155      1      3.1
2      70    1   0       145   174    0        1      125      1      2.6
3      61    1   0       148   203    0        1      161      0      0.0
4      62    0   0       138   294    1        1      106      0      1.9
...   ...  ...  ..       ...   ...  ...      ...      ...      ...      ...
1020   59    1   1       140   221    0        1      164      1      0.0
1021   60    1   0       125   258    0        0      141      1      2.8
1022   47    1   0       110   275    0        0      118      1      1.0
1023   50    0   0       110   254    0        0      159      0      0.0
1024   54    1   0       120   188    0        1      113      0      1.4

      slope  ca  thal  target
0         2   2     3       0
1         0   0     3       0
2         0   0     3       0
3         2   1     3       0
4         1   3     2       0
...     ...  ..   ...     ...
1020      2   0     2       1
1021      1   1     3       0
1022      1   1     2       0
1023      2   0     2       1
1024      1   1     3       0

[1025 rows x 14 columns]
```

In [8]:
```
df.head()
```

Out[8]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	52	1	0	125	212	0	1	168	0	1.0	2	2	3	0
1	53	1	0	140	203	1	0	155	1	3.1	0	0	3	0
2	70	1	0	145	174	0	1	125	1	2.6	0	0	3	0
3	61	1	0	148	203	0	1	161	0	0.0	2	1	3	0
4	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0

In [9]:
```
df.dtypes
```

Out[9]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

Handling Missing Values:

drop columns

In [11]:
```
def handle_missing_values(df, column_name, method='assign', value=None):
    """
    Function to handle missing values in a specific column of a DataFrame.

    Parameters:
    column_name (str): Input DataFrame
    column_name (str): Name of the column containing missing values.
    method (str): Method to handle missing values. Options: 'assign', 'drop'.
    value: The value used for imputation if method='assign'.

    Returns:
    df_handled (DataFrame): DataFrame with missing values handled according to the specified method.
    """
    df_handled = df.copy()  # Create a copy of the original DataFrame

    if method == 'assign':
        # Assign a specific value to missing values in the specified column
        df_handled[column_name].fillna(value, inplace=True)
    elif method == 'drop':
        # Drop rows with missing values in the specified column
        df_handled.dropna(subset=[column_name], inplace=True)
    else:
        raise ValueError("Invalid method. Choose between 'assign' and 'drop'.")

    return df_handled
```

In [12]:
```
df_assigned = handle_missing_values(df, 'exang', method='assign', value=0)

df_dropped = handle_missing_values(df, 'trestbps', method='drop')
```

In [13]:
```
print(df.columns)
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

In [14]:
```
def check_null_values(df):
    return df.isnull()
```

In [15]:
```
null_values = check_null_values(df)
print(null_values)
```

```
        age    sex     cp  trestbps   chol    fbs  restecg  thalach  exang  \
0     False  False  False     False  False  False    False    False  False
1     False  False  False     False  False  False    False    False  False
2     False  False  False     False  False  False    False    False  False
3     False  False  False     False  False  False    False    False  False
4     False  False  False     False  False  False    False    False  False
...     ...    ...    ...       ...    ...    ...      ...      ...    ...
1020  False  False  False     False  False  False    False    False  False
1021  False  False  False     False  False  False    False    False  False
1022  False  False  False     False  False  False    False    False  False
1023  False  False  False     False  False  False    False    False  False
1024  False  False  False     False  False  False    False    False  False

      oldpeak  slope     ca   thal  target
0       False  False  False  False   False
1       False  False  False  False   False
2       False  False  False  False   False
3       False  False  False  False   False
4       False  False  False  False   False
...       ...    ...    ...    ...     ...
1020    False  False  False  False   False
1021    False  False  False  False   False
1022    False  False  False  False   False
1023    False  False  False  False   False
1024    False  False  False  False   False

[1025 rows x 14 columns]
```

In [16]:
```
def check_missing_values(df):
    """
    Function to check missing values in a DataFrame.

    Parameters:
    df (DataFrame): Input DataFrame.

    Returns:
    missing_values (Series): Series containing the count of missing values for each column.
    """
    missing_values = df.isnull().sum()
    return missing_values
```

In [17]:
```
# Assuming df is your DataFrame
missing_values_counts = check_missing_values(df)
print(missing_values_counts)
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Categorical Data Encoding

In [18]:
```
from sklearn.preprocessing import LabelEncoder
```

In [19]:
```
def create_label_encoder():
    return LabelEncoder()
```

In [20]:
```
label_encoder = create_label_encoder()
```

In [22]:
```
df['sex']
```

Out[22]:
```
0       1
1       1
2       1
3       1
4       0
       ..
1020    1
1021    1
1022    1
1023    1
1024    1
Name: sex, Length: 1025, dtype: int64
```

In [23]:
```
label_encoder.fit_transform(df['sex'])
```

Out[23]:
```
array([1, 1, 1, ..., 1, 0, 1], dtype=int64)
```

PyCaret

Train and Evaluate Model

In [23]:
```
cat_features=['sex','cp','fbs','restecg','exang','thal']
```

In [24]:
```
experiment=setup(df,target='target',categorical_features=cat_features)
```

	Description	Value
0	Session id	567
1	Target	target
2	Target type	Binary
3	Original data shape	(1025, 14)
4	Transformed data shape	(1025, 22)
5	Transformed train set shape	(717, 22)
6	Transformed test set shape	(308, 22)
7	Numeric features	7
8	Categorical features	6
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Maximum one-hot encoding	25
14	Encoding method	None
15	Fold Generator	StratifiedKFold
16	Fold Number	10
17	CPU Jobs	-1
18	Use GPU	False
19	Log Experiment	False
20	Experiment Name	clf-default-name
21	USI	4b03

In [25]:
```
beast_model=compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
rf	Random Forest Classifier	0.9847	0.9999	0.9784	0.9925	0.9856	0.9694	0.9708	0.2180
et	Extra Trees Classifier	0.9847	0.9999	0.9784	0.9925	0.9848	0.9695	0.9707	0.1950
lightgbm	Light Gradient Boosting Machine	0.9833	0.9999	0.9757	0.9923	0.9834	0.9666	0.9677	0.2530
dt	Decision Tree Classifier	0.9791	0.9899	0.9783	0.9819	0.9796	0.9581	0.9591	0.0970
gbc	Gradient Boosting Classifier	0.9665	0.9921	0.9649	0.9712	0.9672	0.9330	0.9346	0.1900
ada	Ada Boost Classifier	0.9066	0.9677	0.9212	0.9007	0.9086	0.8128	0.8152	0.1350
ridge	Ridge Classifier	0.8550	0.9237	0.8995	0.8337	0.8641	0.7093	0.7145	0.0690
lda	Linear Discriminant Analysis	0.8522	0.9237	0.8941	0.8332	0.8611	0.7037	0.7095	0.1160
lr	Logistic Regression	0.8465	0.9233	0.8650	0.8307	0.8548	0.6924	0.6967	1.2250
nb	Naive Bayes	0.8451	0.9090	0.8940	0.8225	0.8553	0.6893	0.6953	0.0710
knn	K Neighbors Classifier	0.6973	0.8000	0.7094	0.7064	0.7058	0.3939	0.3964	0.1610
svm	SVM - Linear Kernel	0.6442	0.8425	0.5733	0.7040	0.5299	0.2904	0.3503	0.0990
qda	Quadratic Discriminant Analysis	0.5188	0.5285	0.5080	0.4895	0.4139	0.0358	0.0649	0.0980
dummy	Dummy Classifier	0.5132	0.0000	1.0000	0.5132	0.6783	0.0000	0.0000	0.0780

```
Processing:   0%|          | 0/61 [00:00<?, ?it/s]
```

Test model

In [27]:
```
predict_model(beast_model)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	0.9773	0.9958	0.9810	0.9748	0.9779	0.9545	0.9545

Out[27]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	prediction_label	prediction_score
985	62	1	2	130	231	0	1	146	0	1.8	1	3	3	1	1	0.73
568	54	0	2	160	201	0	1	163	0	0.0	2	1	2	1	1	0.95
535	76	0	2	140	197	0	2	116	0	1.1	1	0	2	1	1	0.99
339	71	0	1	130	253	0	1	144	1	1.4	2	1	3	0	0	1.00
822	62	0	0	138	294	1	1	106	0	1.9	1	3	2	0	0	0.93
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
468	61	1	2	150	243	1	1	137	1	1.0	1	0	2	1	1	0.94
600	62	0	2	130	263	0	1	97	0	1.2	1	1	3	0	0	0.97
278	55	1	0	160	289	0	0	145	1	0.8	1	1	3	0	0	0.99
972	52	1	3	118	186	0	0	190	0	0.0	1	0	1	1	1	0.97
106	51	1	0	140	299	0	1	173	1	1.6	2	0	3	0	0	1.00

308 rows × 16 columns

In [28]:
```
predict_model(beast_model,df.tail())
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Random Forest Classifier	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Out[28]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target	prediction_label	prediction_score
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1	1	1.00
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0	0	1.00
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0	0	0.96
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1	1	0.99
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0	0	1.00

In [29]:
```
predict_model(beast_model,df.drop('target',axis=1).tail())
```

Out[29]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	prediction_label	prediction_score
1020	59	1	1	140	221	0	1	164	1	0.0	2	0	2	1	1.00
1021	60	1	0	125	258	0	0	141	1	2.8	1	1	3	0	1.00
1022	47	1	0	110	275	0	0	118	1	1.0	1	1	2	0	0.96
1023	50	0	0	110	254	0	0	159	0	0.0	2	0	2	1	0.99
1024	54	1	0	120	188	0	1	113	0	1.4	1	1	3	0	1.00

save model

In [30]:
```
save_model(beast_model,model_name='ridge_model')
```

```
Transformation Pipeline and Model Successfully Saved
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['age', 'trestbps', 'chol',
                                              'thalach', 'oldpeak', 'slope',
                                              'ca'],
                                     transformer=SimpleImputer(add_indicator=False,
                                                               copy=True,
                                                               fill_value=None,
                                                               keep_empty_features=False,
                                                               missing_values=nan,
                                                               strategy='mean'))),
                 ('categorical_imputer',
                  TransformerWrapper(exclude=...
                  RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                         class_weight=None, criterion='gini',
                                         max_depth=None, max_features='sqrt',
                                         max_leaf_nodes=None, max_samples=None,
                                         min_impurity_decrease=0.0,
                                         min_samples_leaf=1, min_samples_split=2,
                                         min_weight_fraction_leaf=0.0,
                                         monotonic_cst=None, n_estimators=100,
                                         n_jobs=-1, oob_score=False,
                                         random_state=567, verbose=0,
                                         warm_start=False))]),
```

```
            verbose=false),
        "ridge.model.pkl"]
```

In [ ]: