In [1]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score,
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
import pyswarms as ps
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier,
from sklearn.metrics import roc_curve, auc
```

```
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\computation\expression
s.py:21: UserWarning: Pandas requires version '2.8.4' or newer of 'numexp
r' (version '2.8.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\arrays\masked.py:60: U
serWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (vers
ion '1.3.4' currently installed).
  from pandas.core import (
C:\Users\HP\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarnin
g: A NumPy version >=1.16.5 and <1.23.0 is required for this version of Sc
iPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Load the dataset
df = pd.read_csv("Heart_disease_cleveland_new.csv")
print(df)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   0       145   233    1        2      150      0      2.3
1     67    1   3       160   286    0        2      108      1      1.5
2     67    1   3       120   229    0        2      129      1      2.6
3     37    1   2       130   250    0        0      187      0      3.5
4     41    0   1       130   204    0        2      172      0      1.4
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   45    1   0       110   264    0        0      132      0      1.2
299   68    1   3       144   193    1        0      141      0      3.4
300   57    1   3       130   131    0        0      115      1      1.2
301   57    0   1       130   236    0        2      174      0      0.0
302   38    1   2       138   175    0        0      173      0      0.0

     slope  ca  thal  target
0        2   0     2       0
1        1   3     1       1
2        1   2     3       1
3        2   0     1       0
4        0   0     1       0
..     ...  ..   ...     ...
298      1   0     3       1
299      1   2     3       1
300      1   1     3       1
301      1   1     1       1
302      0   0     1       0

[303 rows x 14 columns]
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [4]:
```
#managing missing values
missing_values=df.isnull().sum()
print(missing_values)
```

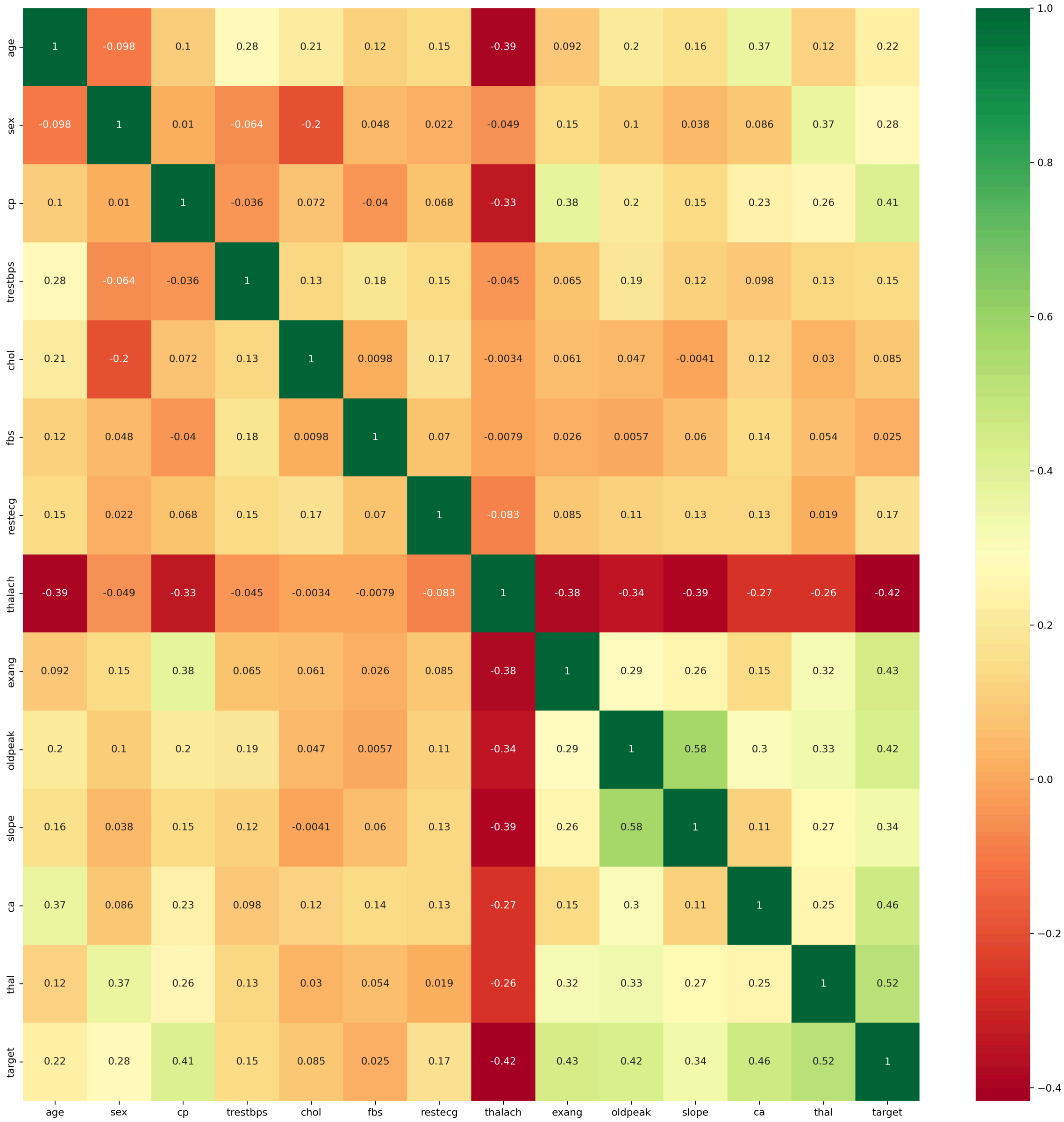
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

In [5]:
```
df.describe()
```

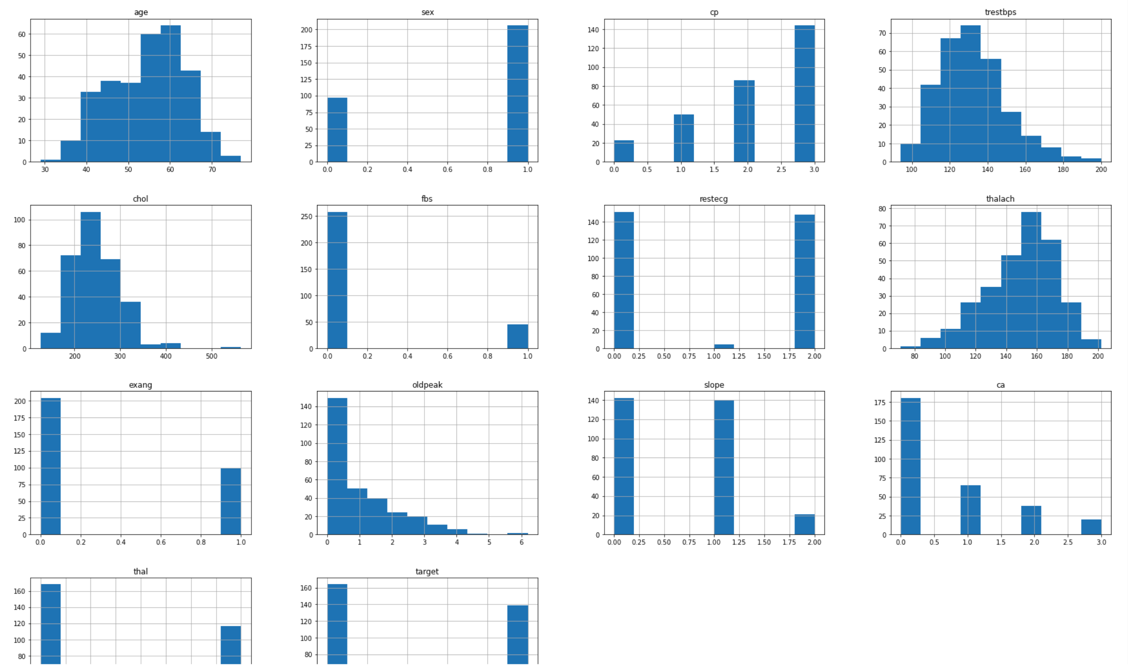
Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.438944	0.679868	2.158416	131.689769	246.693069	0.148515	0.990099
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	0.994971
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	48.000000	0.000000	2.000000	120.000000	211.000000	0.000000	0.000000
50%	56.000000	1.000000	2.000000	130.000000	241.000000	0.000000	1.000000
75%	61.000000	1.000000	3.000000	140.000000	275.000000	0.000000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

In [6]:
```
import seaborn as sns
#get correlations of each features in dataset
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20),dpi=300)
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

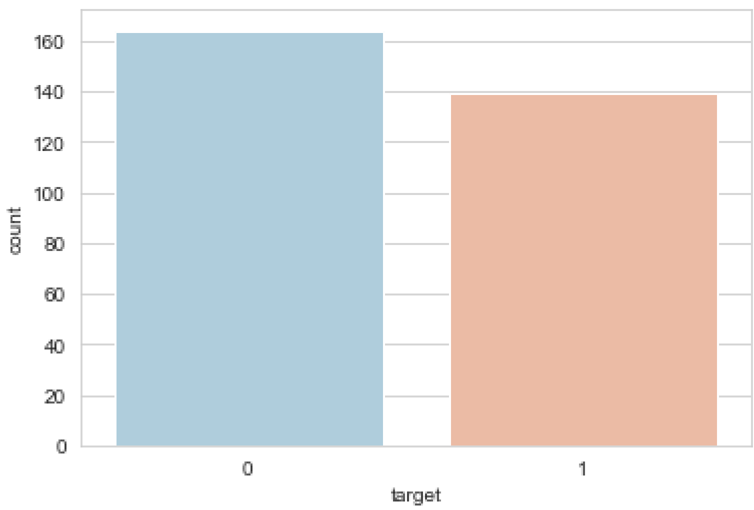


In [7]:
```
df.hist(figsize=(30, 20))
plt.savefig("histogram.png", dpi=300)
plt.show()
```



In [8]:
```
sns.set_style('whitegrid')
sns.countplot(x='target',data=df,palette='RdBu_r')
```

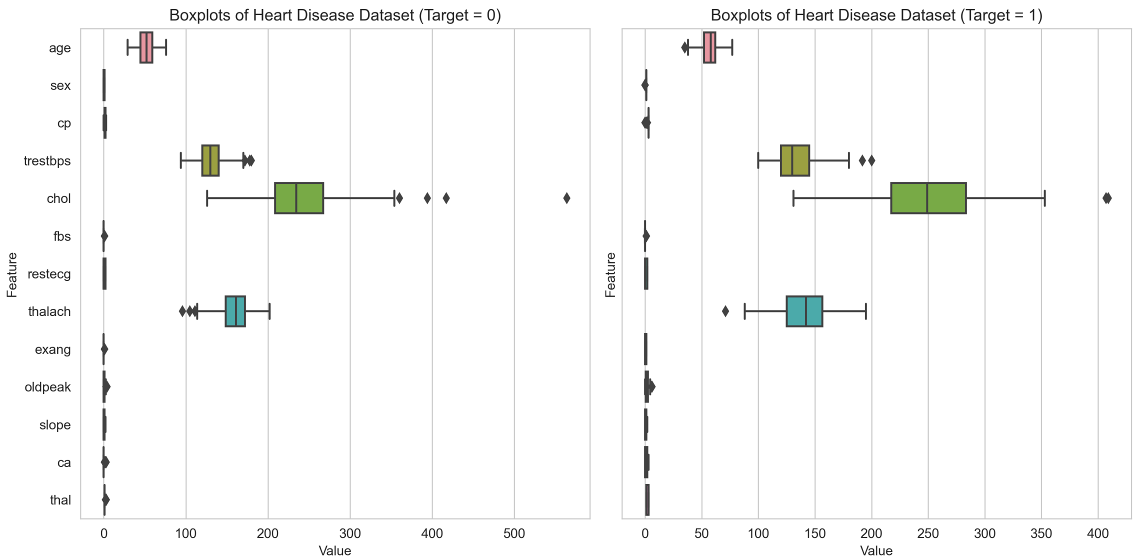
Out[8]: <AxesSubplot:xlabel='target', ylabel='count'>



In [9]:
```
# Define the target column
target_column='target'
```

In [10]:

```
# Separate the dataset based on target values
df_healthy = df[df[target_column] == 0]  # No heart disease
df_disease = df[df[target_column] == 1]  # Heart disease

# Features to plot (excluding the target column)
features = [col for col in df.columns if col != target_column]

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 6),dpi=300, sharey=True)

# Convert DataFrame to long format for Seaborn
df_healthy_melted = df_healthy.melt(value_vars=features, var_name="Feature"
df_disease_melted = df_disease.melt(value_vars=features, var_name="Feature"

# Boxplot for target = 0 (No heart disease)
sns.boxplot(y="Feature", x="Value", data=df_healthy_melted, ax=axes[0])
axes[0].set_title("Boxplots of Heart Disease Dataset (Target = 0)")

# Boxplot for target = 1 (Heart disease)
sns.boxplot(y="Feature", x="Value", data=df_disease_melted, ax=axes[1])
axes[1].set_title("Boxplots of Heart Disease Dataset (Target = 1)")

# Adjust layout
plt.tight_layout()
plt.show()
```



In [11]:

```
# Extract features and target variable
X = df.drop(columns=["target"]).values
y = df["target"].values
```

In [12]:

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [13]:

```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [14]:
```
# Define the MLPGAN class
class MLPGAN:
    def __init__(self, n_inputs, n_hidden=64, n_outputs=1, population_size=
        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_outputs = n_outputs
        self.dim = (n_inputs * n_hidden) + (n_hidden * n_outputs) + n_hidde
        self.population_size = population_size
        self.generations = generations
        self.mutation_rate = mutation_rate
        self.crossover_rate = crossover_rate
        self.population = np.random.randn(self.population_size, self.dim) *

    def forward_prop(self, params, X):
        input_hidden_weights = params[:self.n_inputs * self.n_hidden].resha
        hidden_output_weights = params[self.n_inputs * self.n_hidden:self.n
        hidden_bias = params[self.n_inputs * self.n_hidden + self.n_hidden
        output_bias = params[-self.n_outputs:]

        hidden_layer = np.maximum(0.01 * (np.dot(X, input_hidden_weights) +
        output_layer = 1 / (1 + np.exp(-(np.dot(hidden_layer, hidden_output
        return output_layer

    def fitness_function(self, params, X, y):
        y_pred = self.forward_prop(params, X)
        accuracy = accuracy_score(y, (y_pred >= 0.5).astype(int))
        mse = np.mean((y_pred - y.reshape(-1, 1))**2)
        return accuracy - (0.4 * mse)

    def select_parents(self):
        fitness = np.array([self.fitness_function(ind, X_train_scaled, y_tr
        fitness = np.maximum(fitness - fitness.min(), 1e-10)
        probabilities = fitness / fitness.sum()
        selected_indices = np.random.choice(len(probabilities), self.popula
        return self.population[selected_indices]

    def crossover(self, parents):
        offspring = []
        for _ in range(self.population_size - len(parents)):
            if random.random() < self.crossover_rate:
                p1, p2 = random.sample(list(parents), 2)
                point = random.randint(1, self.dim - 1)
                child = np.concatenate((p1[:point], p2[point:]))
                offspring.append(child)
        return np.array(offspring)

    def mutate(self, offspring):
        for i in range(len(offspring)):
            if random.random() < self.mutation_rate:
                mutation_point = random.randint(0, self.dim - 1)
                offspring[i][mutation_point] += np.random.randn() * 0.01
        return offspring

    def train(self, X_train, y_train):
        for _ in range(self.generations):
            parents = self.select_parents()
            offspring = self.crossover(parents)
            offspring = self.mutate(offspring)
            self.population = np.vstack((parents, offspring))
        self.best_params = self.select_parents()[-1]
```

```
    def predict(self, X):
        y_pred = self.forward_prop(self.best_params, X)
        return (y_pred >= 0.5).astype(int)
```

In [15]:
```
# Train the genetic algorithm-based MLP model
mlp_ga = MLPGAN(n_inputs=X.shape[1])
mlp_ga.train(X_train_scaled, y_train)
```

In [16]:
```
# Generate predictions from MLPGAN
y_pred_mlp_ga = mlp_ga.predict(X_train_scaled)
y_pred_mlp_ga_test = mlp_ga.predict(X_test_scaled)
```

In [17]:
```
# Append MLP-GA predictions as additional features
X_train_combined = np.column_stack((X_train_scaled, y_pred_mlp_ga))
X_test_combined = np.column_stack((X_test_scaled, y_pred_mlp_ga_test))
```

In [18]:
```
# Train the Random Forest Classifier with tuned parameters
rf = RandomForestClassifier(n_estimators=200, max_depth=10, min_samples_spl
rf.fit(X_train_combined, y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split
=4,
                       n_estimators=200, random_state=42)
```

In [19]:
```
# Make final predictions
y_pred_rf = rf.predict(X_test_combined)
```

In [20]:
```
# Compute evaluation metrics
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
auc_rf = roc_auc_score(y_test, y_pred_rf)
```

In [21]:
```
models = {
    "Logistic Regression": LogisticRegression(C=1.5, penalty='l2'),
    "SVM": SVC(C=1, gamma=0.1, kernel='rbf'),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Decision Tree": DecisionTreeClassifier(criterion='gini'),
    "Random Forest": RandomForestClassifier(n_estimators=1000, criterion='g
    "Extra Trees": ExtraTreesClassifier(n_estimators=100),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100, max_d
    "GaussianNB": GaussianNB(),
    "XGBoost": XGBClassifier(n_estimators=300, max_depth=15),
    "MLP-BP": MLPClassifier(hidden_layer_sizes=(30,), activation='relu', so
}
```

In [22]:
```
results = []
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test_scaled)[:,1]
    else:
        y_proba = model.decision_function(X_test_scaled)

    results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'AUC': roc_auc_score(y_test, y_proba)
    })
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iter
ations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [23]:
```
results.append({
    'Model': 'MLP-GA-RF',
    'Accuracy': accuracy_rf,
    'Precision': precision_rf,
    'Recall': recall_rf,
    'F1 Score': f1_rf,
    'AUC': auc_rf
})
```

In [24]:
```
# Print the results
print("MLP+GA+RF Model Metrics:")
print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1 Score: {f1_rf:.4f}")
print(f"AUC: {auc_rf:.4f}")
```

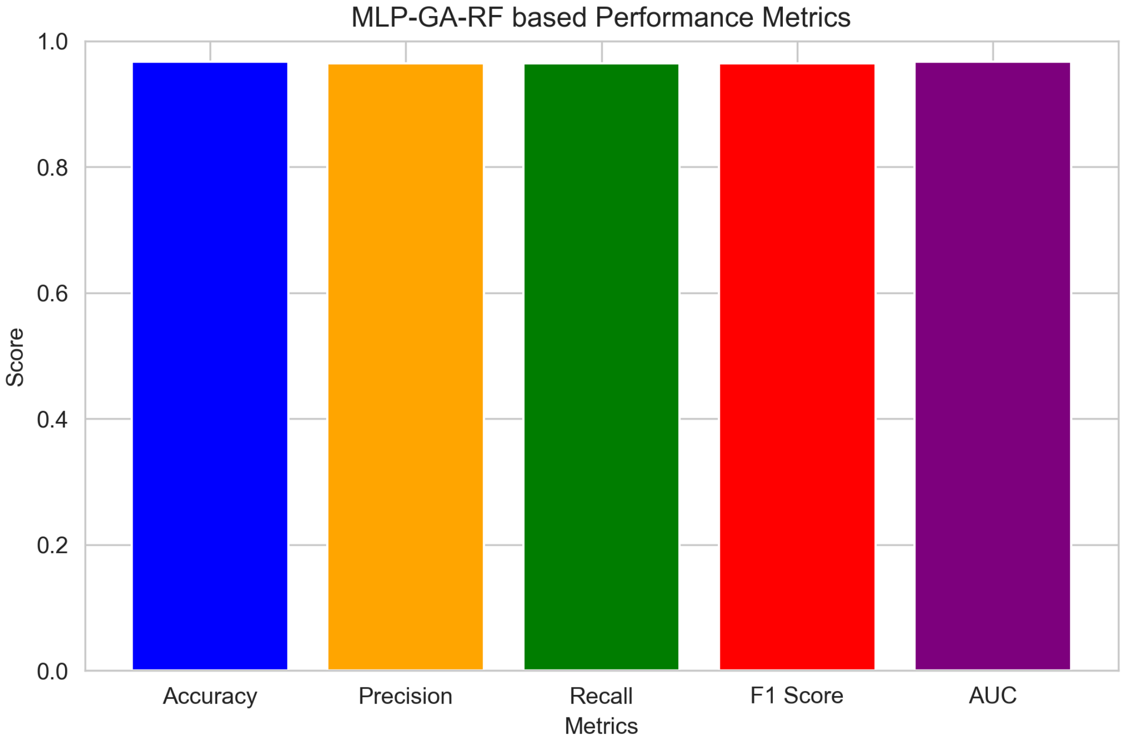
```
MLP+GA+RF Model Metrics:
Accuracy: 0.9672
Precision: 0.9643
Recall: 0.9643
F1 Score: 0.9643
AUC: 0.9670
```

In [25]:
```

metrics = {
    "Accuracy": accuracy_rf,
    "Precision": precision_rf,
    "Recall": recall_rf,
    "F1 Score": f1_rf,
    "AUC": auc_rf
}
```

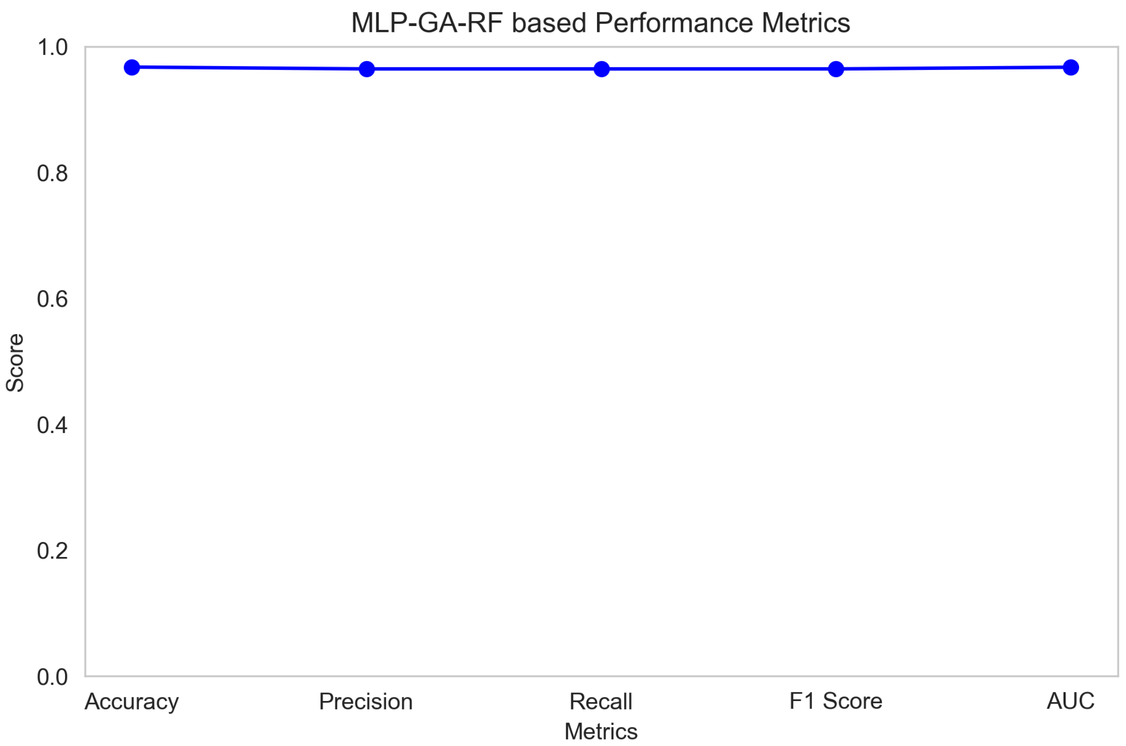
In [26]:
```
results_df = pd.DataFrame(results)
print(results_df.sort_values(by='Accuracy', ascending=False))
```

```
                  Model  Accuracy  Precision    Recall  F1 Score       AUC
10               MLP-GA-RF  0.967213   0.964286  0.964286  0.964286  0.966991
2                    KNN  0.901639   0.823529  1.000000  0.903226  0.924242
4          Random Forest  0.901639   0.843750  0.964286  0.900000  0.952381
0    Logistic Regression  0.868852   0.812500  0.928571  0.866667  0.952381
7             GaussianNB  0.868852   0.794118  0.964286  0.870968  0.949134
8                XGBoost  0.868852   0.812500  0.928571  0.866667  0.906926
9                 MLP-BP  0.868852   0.794118  0.964286  0.870968  0.957792
1                    SVM  0.852459   0.806452  0.892857  0.847458  0.944805
6      Gradient Boosting  0.852459   0.787879  0.928571  0.852459  0.945887
5            Extra Trees  0.836066   0.764706  0.928571  0.838710  0.935065
3          Decision Tree  0.786885   0.714286  0.892857  0.793651  0.794913
```

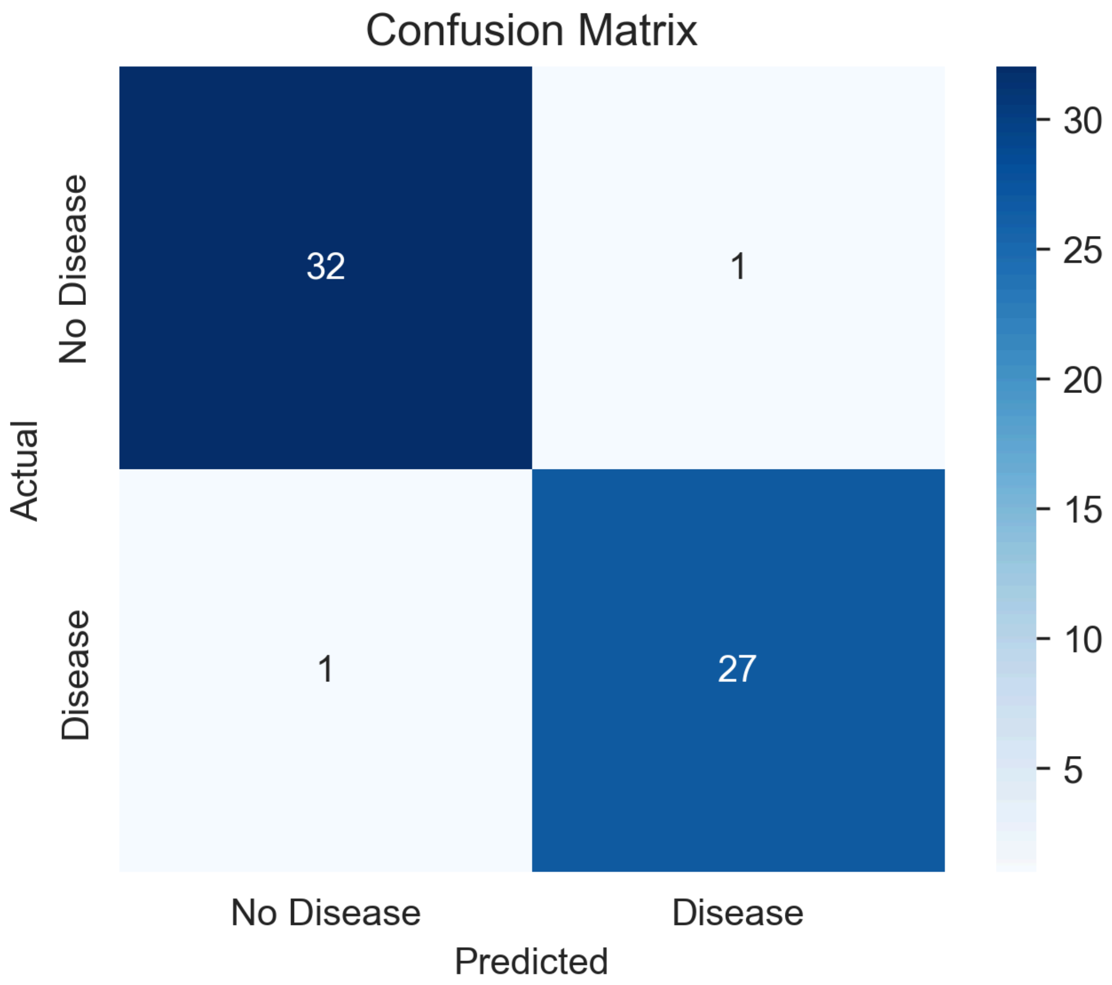
In [27]:
```
plt.figure(figsize=(8, 5),dpi=300)
plt.bar(metrics.keys(), metrics.values(), color=['blue', 'orange', 'green',
plt.xlabel("Metrics")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.title("MLP-GA-RF based Performance Metrics")
plt.show()
```



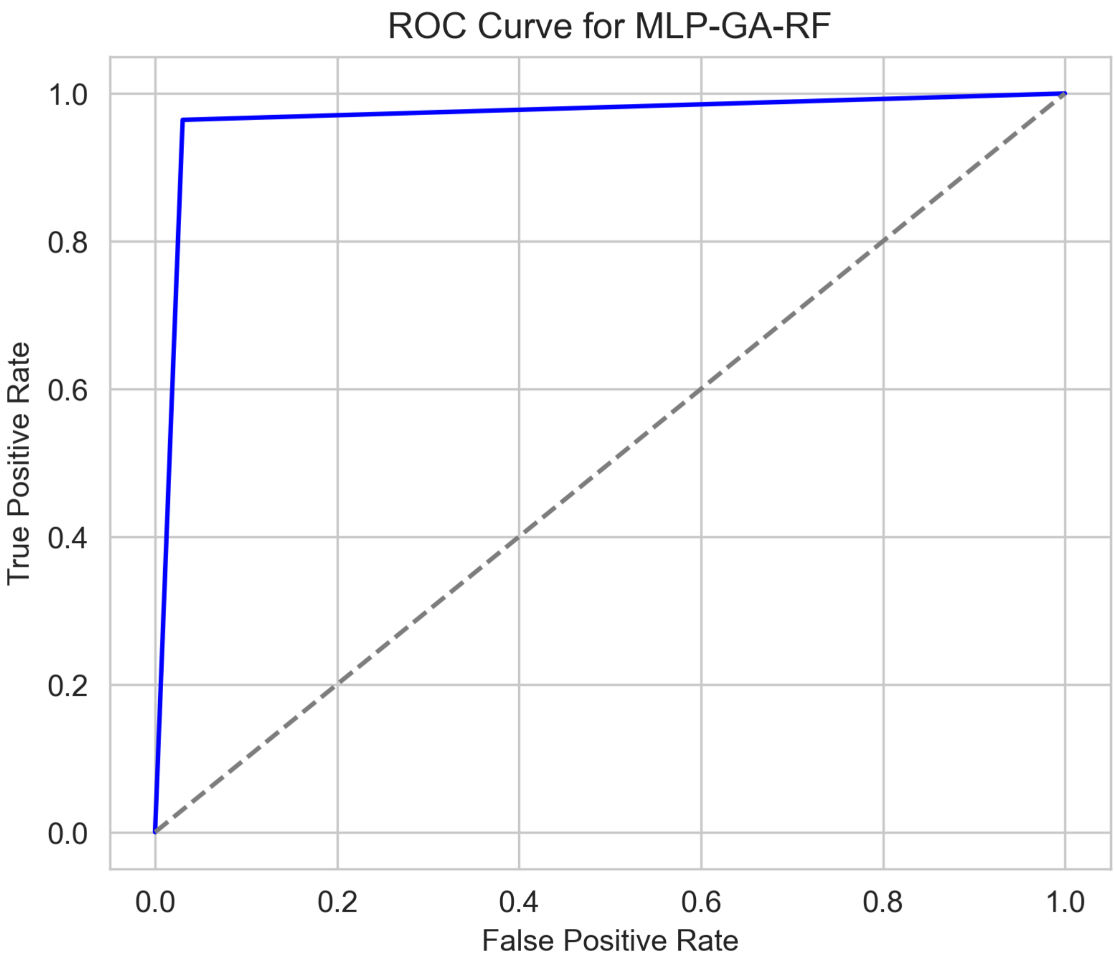
In [28]:
```
plt.figure(figsize=(8, 5),dpi=300)
plt.plot(list(metrics.keys()), list(metrics.values()), marker='o', linestyl
plt.xlabel("Metrics")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.title("MLP-GA-RF based Performance Metrics")
plt.grid()
plt.show()
```



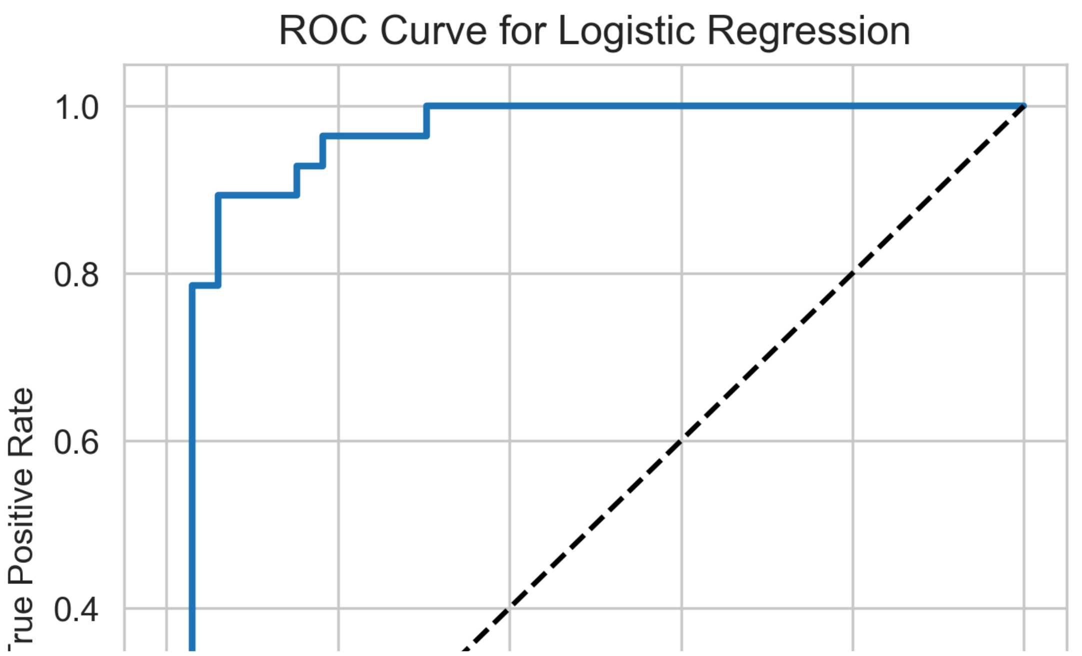
In [29]:
```
# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(5, 4),dpi=300)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



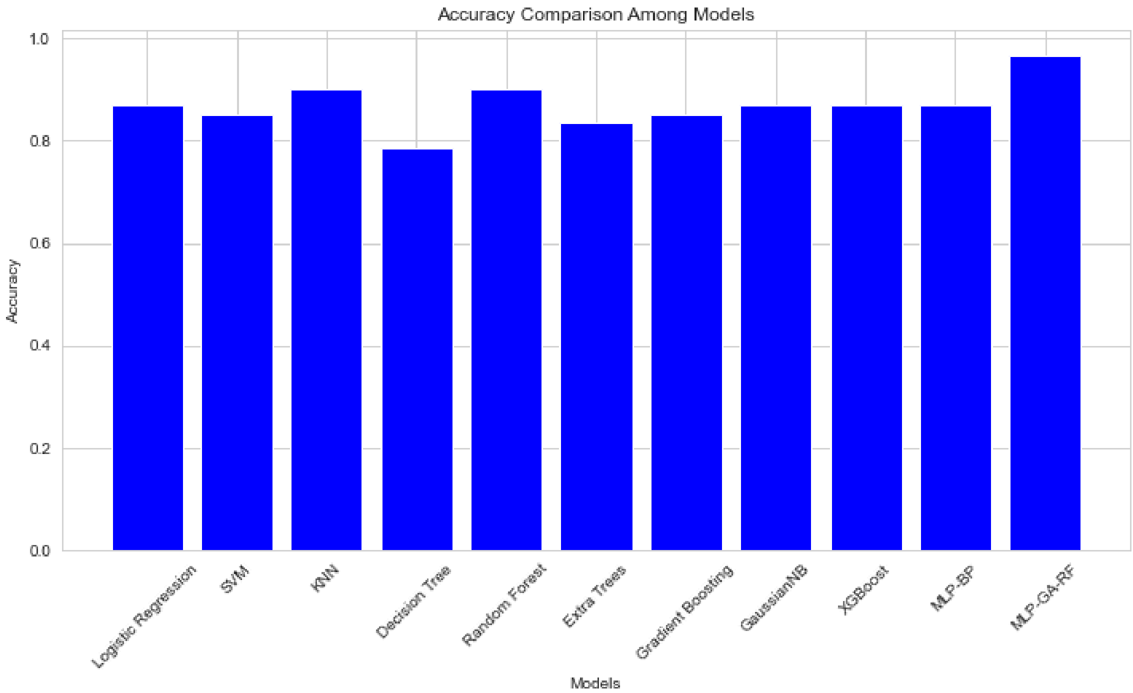
In [30]:
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_rf)
plt.figure(figsize=(6, 5),dpi=300)
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {auc_rf:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for MLP-GA-RF")
#plt.legend()
plt.show()
```



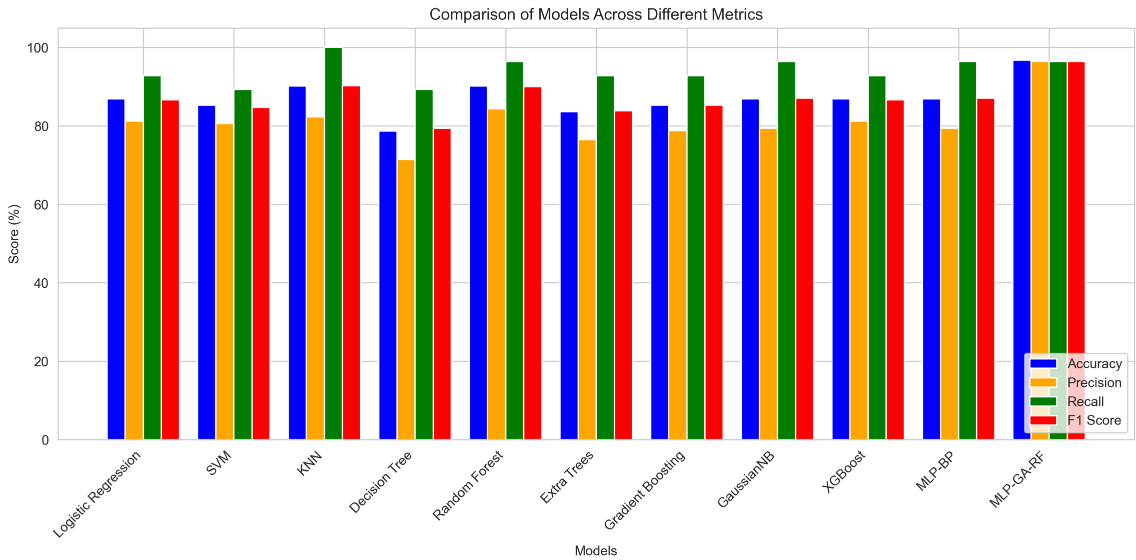
In [31]:
```
# Plot individual ROC curves for each model without AUC and with dpi=300
for name, model in models.items():
    plt.figure(figsize=(5, 5), dpi=300)  # High-resolution figure
    model.fit(X_train_scaled, y_train)
    if hasattr(model, "predict_proba"):
        y_probs = model.predict_proba(X_test_scaled)[:, 1]  # Get probabili
    else:
        y_probs = model.decision_function(X_test_scaled)  # Use decision_fu
    fpr, tpr, _ = roc_curve(y_test, y_probs)

    plt.plot(fpr, tpr, linewidth=2, label=f'{name}')
    plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {name}')
    plt.legend(loc='lower right')
    plt.show()
```



In [32]:
```
# Bar Plot for Accuracy Comparison
plt.figure(figsize=(12, 6))
plt.bar(results_df['Model'], results_df['Accuracy'], color='blue')
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison Among Models")
plt.xticks(rotation=45)
plt.show()
```



In [33]:
```
# Extracting performance metrics for visualization
models_list = results_df['Model'].values
accuracy = results_df['Accuracy'].values * 100
precision = results_df['Precision'].values * 100
recall = results_df['Recall'].values * 100
f1_score_values = results_df['F1 Score'].values * 100

# Set width and positions for bars
x = np.arange(len(models_list))
width = 0.2

# Create a grouped bar chart
fig, ax = plt.subplots(figsize=(12, 6),dpi=300)
ax.bar(x - 1.5 * width, accuracy, width, label="Accuracy", color='blue')
ax.bar(x - 0.5 * width, precision, width, label="Precision", color='orange'
ax.bar(x + 0.5 * width, recall, width, label="Recall", color='green')
ax.bar(x + 1.5 * width, f1_score_values, width, label="F1 Score", color='re

# Labels and formatting
ax.set_ylabel("Score (%)")
ax.set_xlabel("Models")
ax.set_xticks(x)
ax.set_xticklabels(models_list, rotation=45, ha='right')
ax.set_title("Comparison of Models Across Different Metrics")
ax.legend(loc="lower right")

# Display the plot
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [1]:
```
import numpy as np
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score,
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
import pyswarms as ps
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier,
from sklearn.metrics import roc_curve, auc
```

```
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\computation\expression
s.py:21: UserWarning: Pandas requires version '2.8.4' or newer of 'numexp
r' (version '2.8.1' currently installed).
  from pandas.core.computation.check import NUMEXPR_INSTALLED
C:\Users\HP\anaconda3\lib\site-packages\pandas\core\arrays\masked.py:60: U
serWarning: Pandas requires version '1.3.6' or newer of 'bottleneck' (vers
ion '1.3.4' currently installed).
  from pandas.core import (
C:\Users\HP\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarnin
g: A NumPy version >=1.16.5 and <1.23.0 is required for this version of Sc
iPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [2]:
```
# Load the dataset
df = pd.read_csv("Heart_disease_cleveland_new.csv")
print(df)
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   0       145   233    1        2      150      0      2.3   
1     67    1   3       160   286    0        2      108      1      1.5   
2     67    1   3       120   229    0        2      129      1      2.6   
3     37    1   2       130   250    0        0      187      0      3.5   
4     41    0   1       130   204    0        2      172      0      1.4   
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...   
298   45    1   0       110   264    0        0      132      0      1.2   
299   68    1   3       144   193    1        0      141      0      3.4   
300   57    1   3       130   131    0        0      115      1      1.2   
301   57    0   1       130   236    0        2      174      0      0.0   
302   38    1   2       138   175    0        0      173      0      0.0   

     slope  ca  thal  target  
0        2   0     2       0  
1        1   3     1       1  
2        1   2     3       1  
3        2   0     1       0  
4        0   0     1       0  
..     ...  ..   ...     ...  
298      1   0     3       1  
299      1   2     3       1  
300      1   1     3       1  
301      1   1     1       1  
302      0   0     1       0  

[303 rows x 14 columns]
```

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

In [4]:
```
#managing missing values
missing_values=df.isnull().sum()
print(missing_values)
```

```
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
```

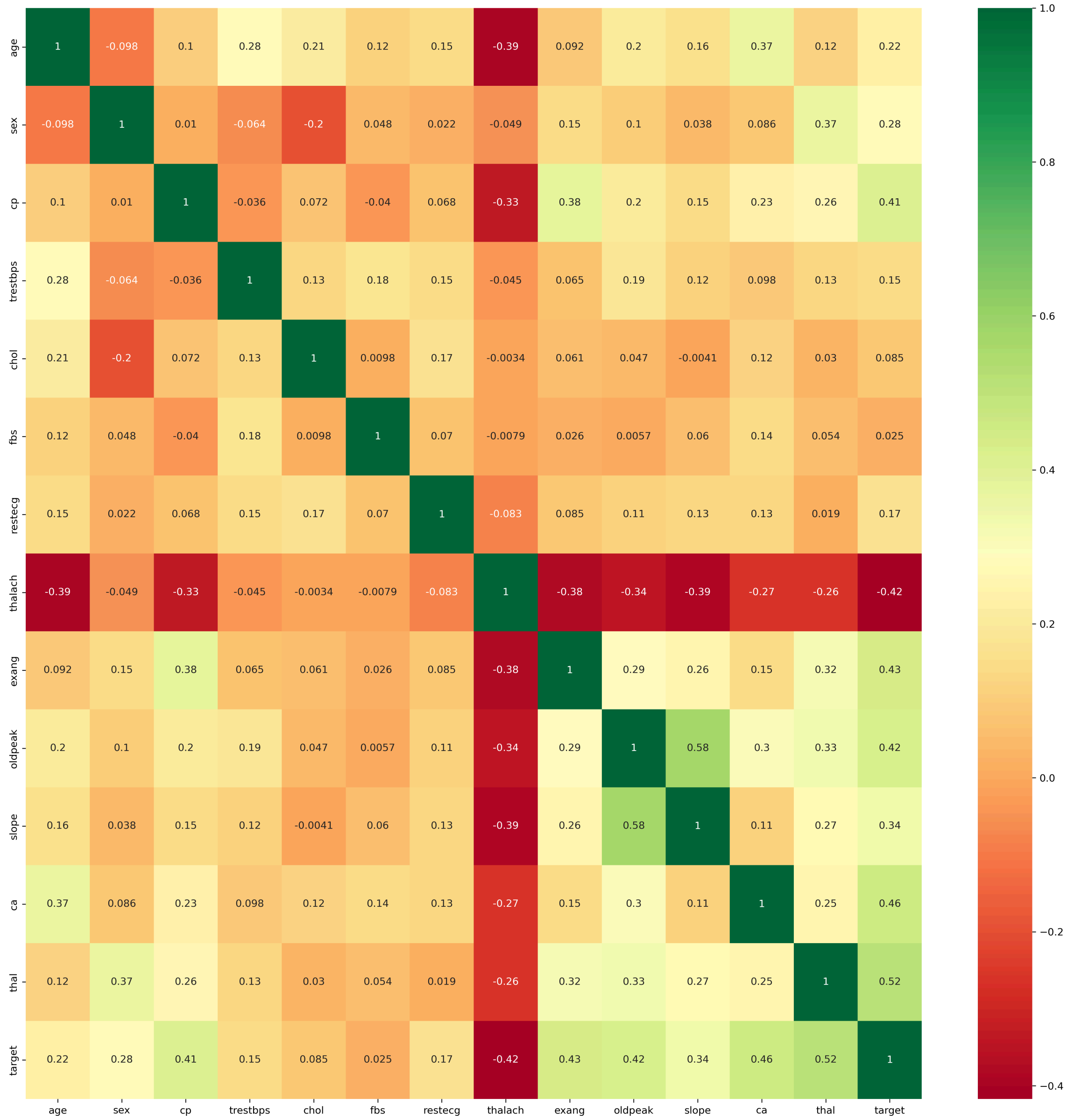
In [5]:
```
df.describe()
```

Out[5]:

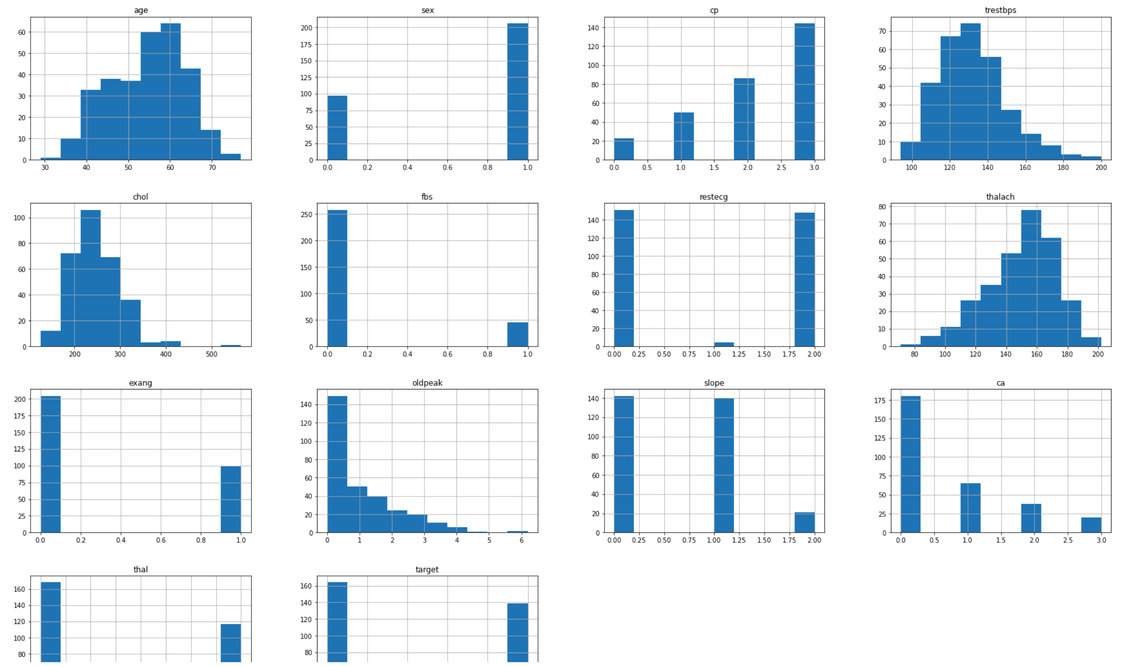
	age	sex	cp	trestbps	chol	fbs	restecg
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.438944	0.679868	2.158416	131.689769	246.693069	0.148515	0.990099
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	0.994971
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000
25%	48.000000	0.000000	2.000000	120.000000	211.000000	0.000000	0.000000
50%	56.000000	1.000000	2.000000	130.000000	241.000000	0.000000	1.000000
75%	61.000000	1.000000	3.000000	140.000000	275.000000	0.000000	2.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000

In [6]:

```
import seaborn as sns
#get correlations of each features in dataset
cormat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20),dpi=300)
#plot heat map
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

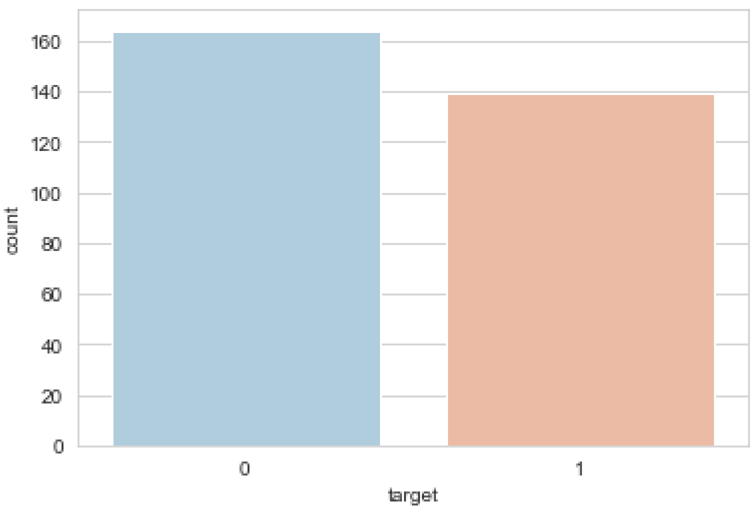


In [7]:
```
df.hist(figsize=(30, 20))
plt.savefig("histogram.png", dpi=300)
plt.show()
```



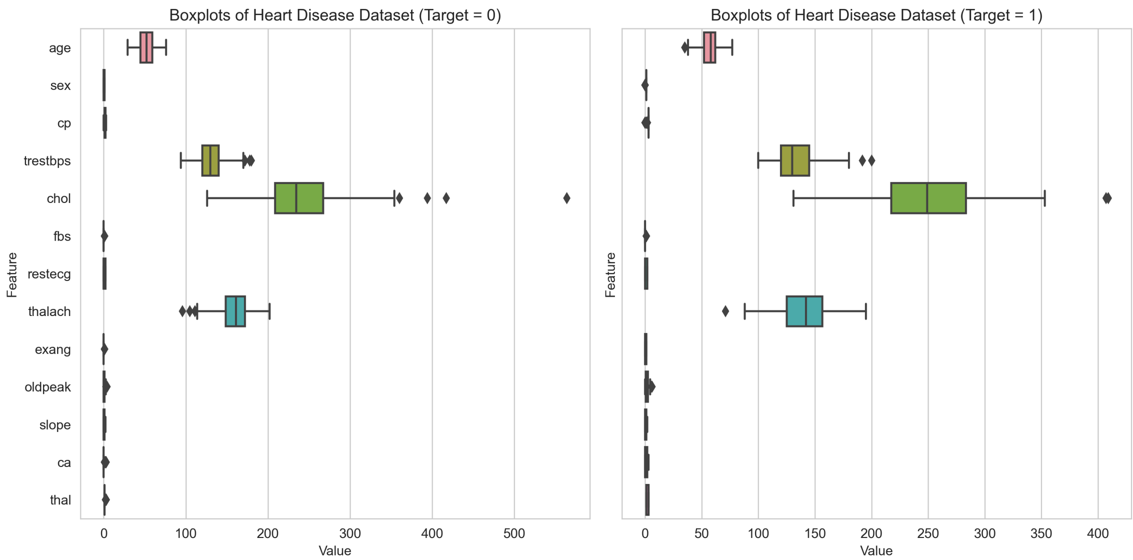
In [8]:
```
sns.set_style('whitegrid')
sns.countplot(x='target',data=df,palette='RdBu_r')
```

Out[8]: <AxesSubplot:xlabel='target', ylabel='count'>



In [9]:
```
# Define the target column
target_column='target'
```

In [10]:
```
# Separate the dataset based on target values
df_healthy = df[df[target_column] == 0]  # No heart disease
df_disease = df[df[target_column] == 1]  # Heart disease

# Features to plot (excluding the target column)
features = [col for col in df.columns if col != target_column]

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 6),dpi=300, sharey=True)

# Convert DataFrame to long format for Seaborn
df_healthy_melted = df_healthy.melt(value_vars=features, var_name="Feature"
df_disease_melted = df_disease.melt(value_vars=features, var_name="Feature"

# Boxplot for target = 0 (No heart disease)
sns.boxplot(y="Feature", x="Value", data=df_healthy_melted, ax=axes[0])
axes[0].set_title("Boxplots of Heart Disease Dataset (Target = 0)")

# Boxplot for target = 1 (Heart disease)
sns.boxplot(y="Feature", x="Value", data=df_disease_melted, ax=axes[1])
axes[1].set_title("Boxplots of Heart Disease Dataset (Target = 1)")

# Adjust layout
plt.tight_layout()
plt.show()
```



In [11]:
```
# Extract features and target variable
X = df.drop(columns=["target"]).values
y = df["target"].values
```

In [12]:
```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [13]:
```
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [14]:
```
# Define the MLPGAN class
class MLPGAN:
    def __init__(self, n_inputs, n_hidden=64, n_outputs=1, population_size=
        self.n_inputs = n_inputs
        self.n_hidden = n_hidden
        self.n_outputs = n_outputs
        self.dim = (n_inputs * n_hidden) + (n_hidden * n_outputs) + n_hidde
        self.population_size = population_size
        self.generations = generations
        self.mutation_rate = mutation_rate
        self.crossover_rate = crossover_rate
        self.population = np.random.randn(self.population_size, self.dim) *

    def forward_prop(self, params, X):
        input_hidden_weights = params[:self.n_inputs * self.n_hidden].resha
        hidden_output_weights = params[self.n_inputs * self.n_hidden:self.n
        hidden_bias = params[self.n_inputs * self.n_hidden + self.n_hidden
        output_bias = params[-self.n_outputs:]

        hidden_layer = np.maximum(0.01 * (np.dot(X, input_hidden_weights) +
        output_layer = 1 / (1 + np.exp(-(np.dot(hidden_layer, hidden_output
        return output_layer

    def fitness_function(self, params, X, y):
        y_pred = self.forward_prop(params, X)
        accuracy = accuracy_score(y, (y_pred >= 0.5).astype(int))
        mse = np.mean((y_pred - y.reshape(-1, 1))**2)
        return accuracy - (0.4 * mse)

    def select_parents(self):
        fitness = np.array([self.fitness_function(ind, X_train_scaled, y_tr
        fitness = np.maximum(fitness - fitness.min(), 1e-10)
        probabilities = fitness / fitness.sum()
        selected_indices = np.random.choice(len(probabilities), self.popula
        return self.population[selected_indices]

    def crossover(self, parents):
        offspring = []
        for _ in range(self.population_size - len(parents)):
            if random.random() < self.crossover_rate:
                p1, p2 = random.sample(list(parents), 2)
                point = random.randint(1, self.dim - 1)
                child = np.concatenate((p1[:point], p2[point:]))
                offspring.append(child)
        return np.array(offspring)

    def mutate(self, offspring):
        for i in range(len(offspring)):
            if random.random() < self.mutation_rate:
                mutation_point = random.randint(0, self.dim - 1)
                offspring[i][mutation_point] += np.random.randn() * 0.01
        return offspring

    def train(self, X_train, y_train):
        for _ in range(self.generations):
            parents = self.select_parents()
            offspring = self.crossover(parents)
            offspring = self.mutate(offspring)
            self.population = np.vstack((parents, offspring))
        self.best_params = self.select_parents()[-1]
```

```
    def predict(self, X):
        y_pred = self.forward_prop(self.best_params, X)
        return (y_pred >= 0.5).astype(int)
```

In [15]:
```
# Train the genetic algorithm-based MLP model
mlp_ga = MLPGAN(n_inputs=X.shape[1])
mlp_ga.train(X_train_scaled, y_train)
```

In [16]:
```
# Generate predictions from MLPGAN
y_pred_mlp_ga = mlp_ga.predict(X_train_scaled)
y_pred_mlp_ga_test = mlp_ga.predict(X_test_scaled)
```

In [17]:
```
# Append MLP-GA predictions as additional features
X_train_combined = np.column_stack((X_train_scaled, y_pred_mlp_ga))
X_test_combined = np.column_stack((X_test_scaled, y_pred_mlp_ga_test))
```

In [18]:
```
# Train the Random Forest Classifier with tuned parameters
rf = RandomForestClassifier(n_estimators=200, max_depth=10, min_samples_spl
rf.fit(X_train_combined, y_train)
```

Out[18]:
```
RandomForestClassifier(max_depth=10, min_samples_leaf=2, min_samples_split
=4,
                       n_estimators=200, random_state=42)
```

In [19]:
```
# Make final predictions
y_pred_rf = rf.predict(X_test_combined)
```

In [20]:
```
# Compute evaluation metrics
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)
auc_rf = roc_auc_score(y_test, y_pred_rf)
```

In [21]:
```
models = {
    "Logistic Regression": LogisticRegression(C=1.5, penalty='l2'),
    "SVM": SVC(C=1, gamma=0.1, kernel='rbf'),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Decision Tree": DecisionTreeClassifier(criterion='gini'),
    "Random Forest": RandomForestClassifier(n_estimators=1000, criterion='g
    "Extra Trees": ExtraTreesClassifier(n_estimators=100),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100, max_d
    "GaussianNB": GaussianNB(),
    "XGBoost": XGBClassifier(n_estimators=300, max_depth=15),
    "MLP-BP": MLPClassifier(hidden_layer_sizes=(30,), activation='relu', so
}
```

In [22]:
```
results = []
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    if hasattr(model, "predict_proba"):
        y_proba = model.predict_proba(X_test_scaled)[:,1]
    else:
        y_proba = model.decision_function(X_test_scaled)

    results.append({
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1 Score': f1_score(y_test, y_pred),
        'AUC': roc_auc_score(y_test, y_proba)
    })
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer
_perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iter
ations (200) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [23]:
```
results.append({
    'Model': 'MLP-GA-RF',
    'Accuracy': accuracy_rf,
    'Precision': precision_rf,
    'Recall': recall_rf,
    'F1 Score': f1_rf,
    'AUC': auc_rf
})
```

In [24]:
```
# Print the results
print("MLP+GA+RF Model Metrics:")
print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1 Score: {f1_rf:.4f}")
print(f"AUC: {auc_rf:.4f}")
```

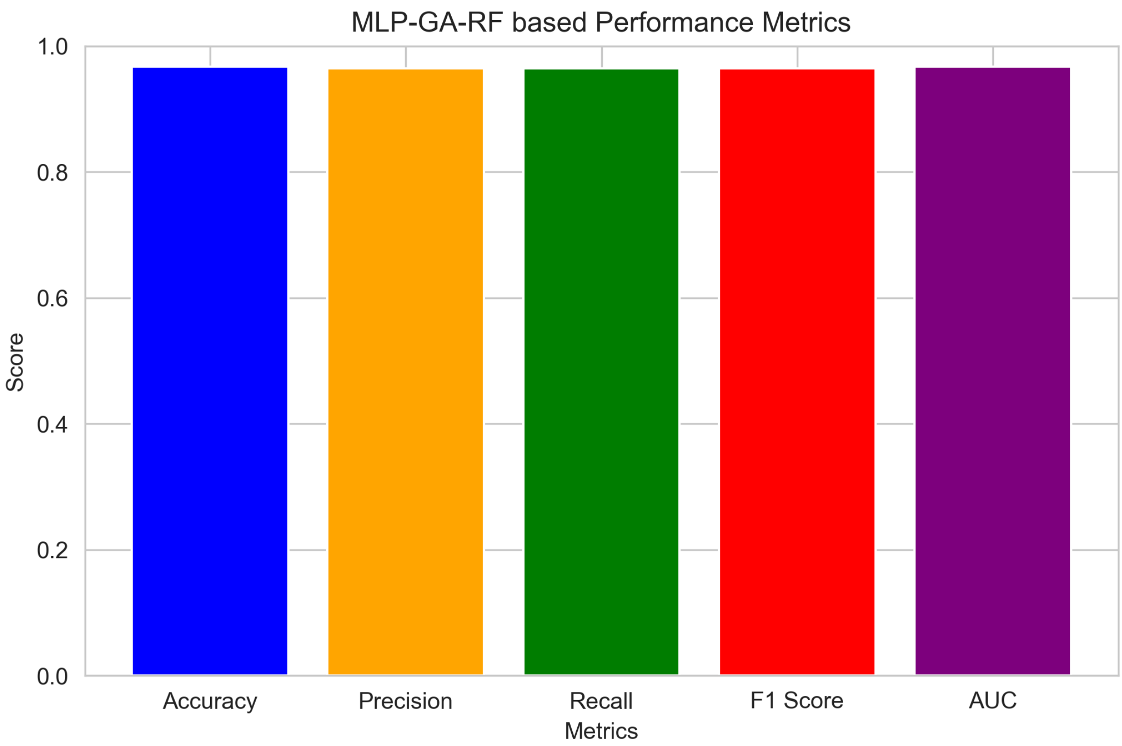
```
MLP+GA+RF Model Metrics:
Accuracy: 0.9672
Precision: 0.9643
Recall: 0.9643
F1 Score: 0.9643
AUC: 0.9670
```

In [25]:
```

metrics = {
    "Accuracy": accuracy_rf,
    "Precision": precision_rf,
    "Recall": recall_rf,
    "F1 Score": f1_rf,
    "AUC": auc_rf
}
```

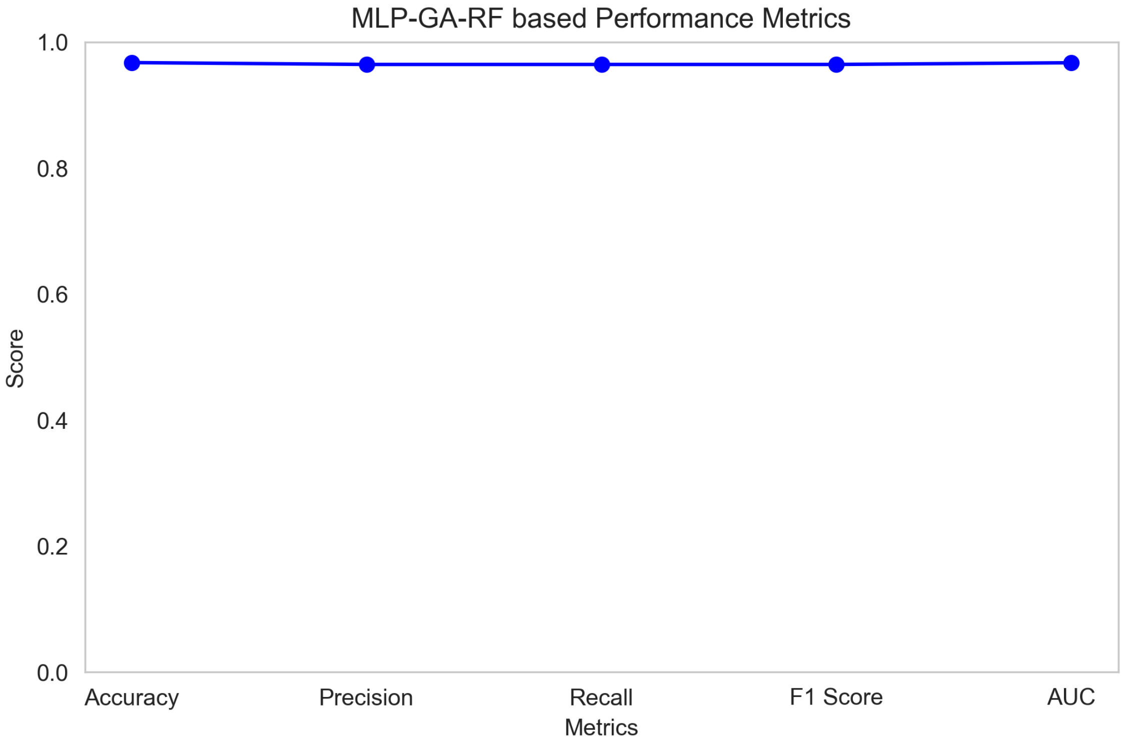
In [26]:
```
results_df = pd.DataFrame(results)
print(results_df.sort_values(by='Accuracy', ascending=False))
```

```
                  Model  Accuracy  Precision    Recall  F1 Score       AUC
10               MLP-GA-RF  0.967213   0.964286  0.964286  0.964286  0.966991
2                      KNN  0.901639   0.823529  1.000000  0.903226  0.924242
4            Random Forest  0.901639   0.843750  0.964286  0.900000  0.952381
0      Logistic Regression  0.868852   0.812500  0.928571  0.866667  0.952381
7               GaussianNB  0.868852   0.794118  0.964286  0.870968  0.949134
8                  XGBoost  0.868852   0.812500  0.928571  0.866667  0.906926
9                   MLP-BP  0.868852   0.794118  0.964286  0.870968  0.957792
1                      SVM  0.852459   0.806452  0.892857  0.847458  0.944805
6        Gradient Boosting  0.852459   0.787879  0.928571  0.852459  0.945887
5              Extra Trees  0.836066   0.764706  0.928571  0.838710  0.935065
3            Decision Tree  0.786885   0.714286  0.892857  0.793651  0.794913
```

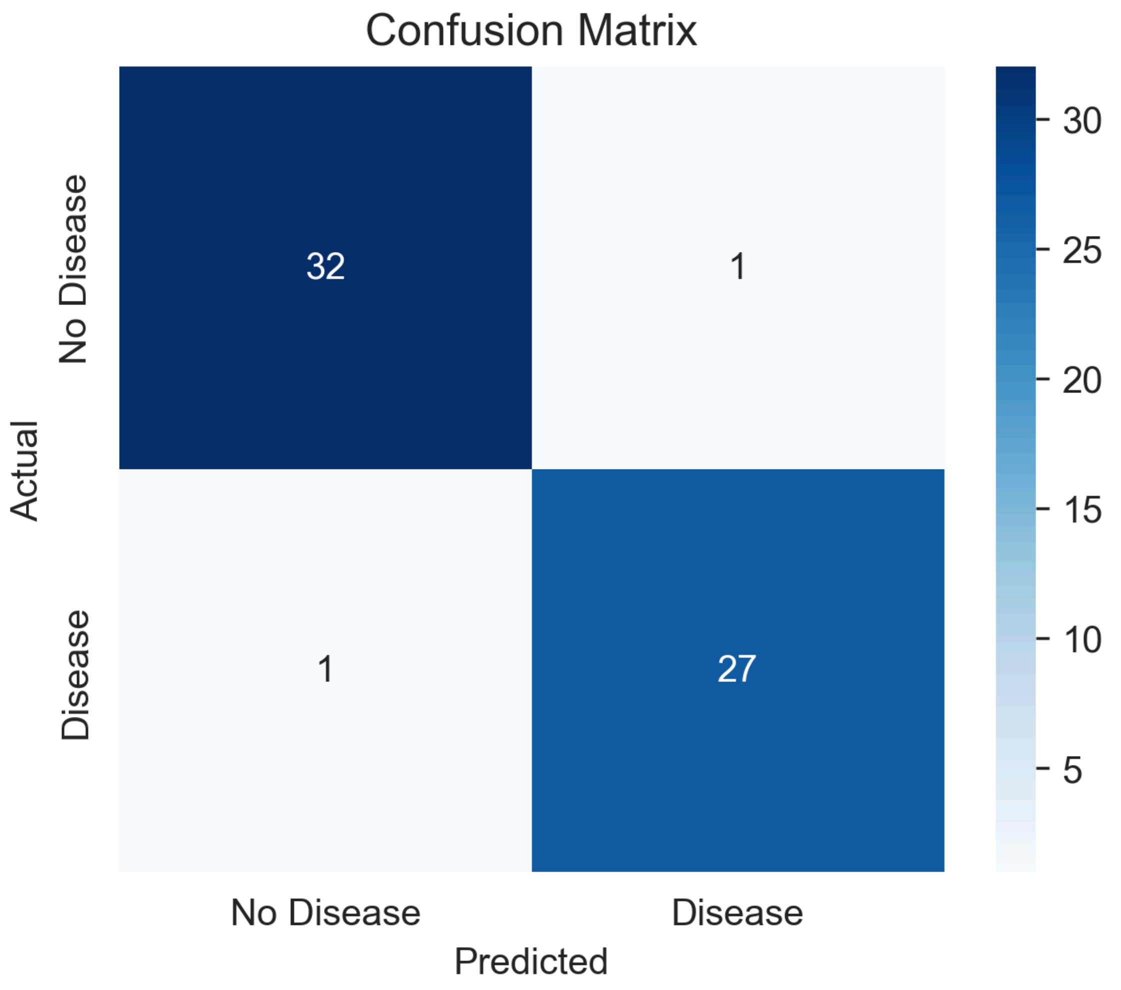
In [27]:
```
plt.figure(figsize=(8, 5),dpi=300)
plt.bar(metrics.keys(), metrics.values(), color=['blue', 'orange', 'green',
plt.xlabel("Metrics")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.title("MLP-GA-RF based Performance Metrics")
plt.show()
```



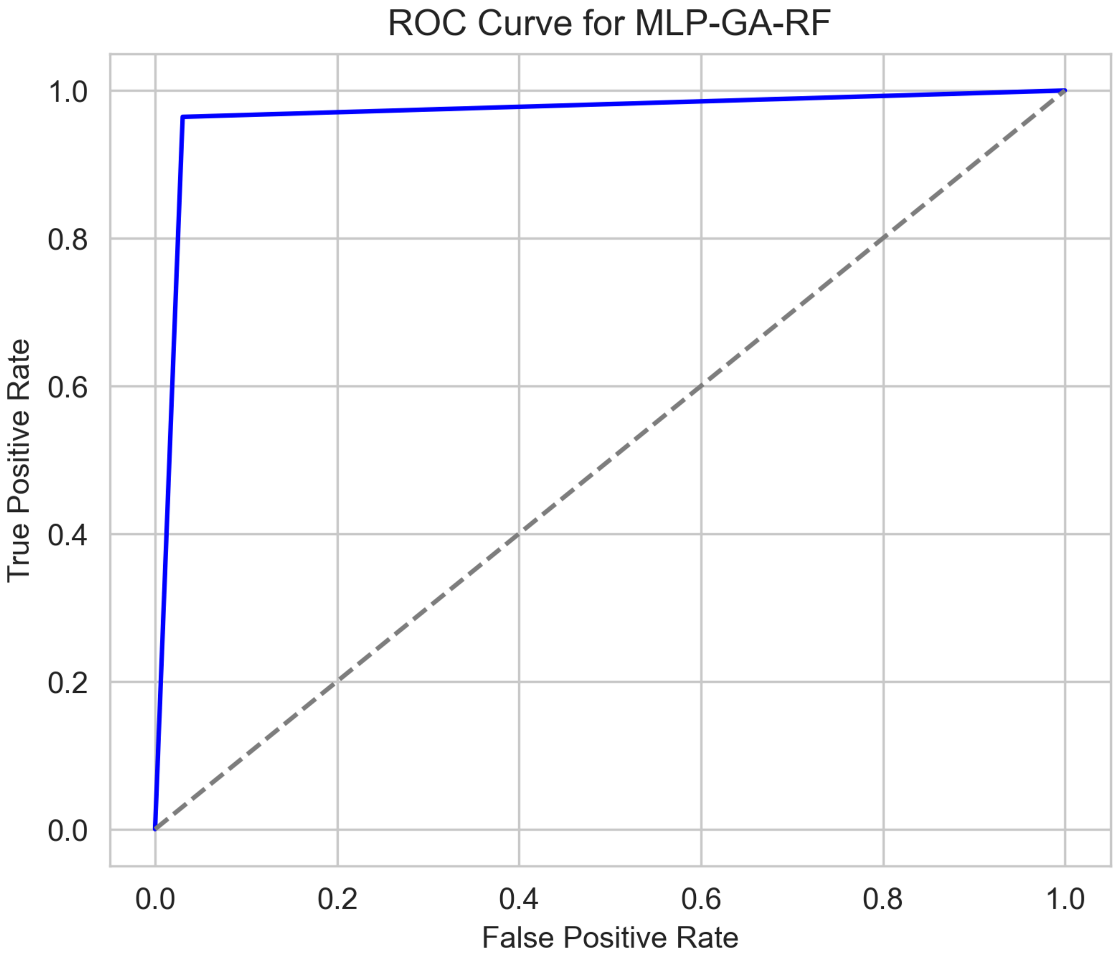
In [28]:
```
plt.figure(figsize=(8, 5),dpi=300)
plt.plot(list(metrics.keys()), list(metrics.values()), marker='o', linestyl
plt.xlabel("Metrics")
plt.ylabel("Score")
plt.ylim(0, 1)
plt.title("MLP-GA-RF based Performance Metrics")
plt.grid()
plt.show()
```



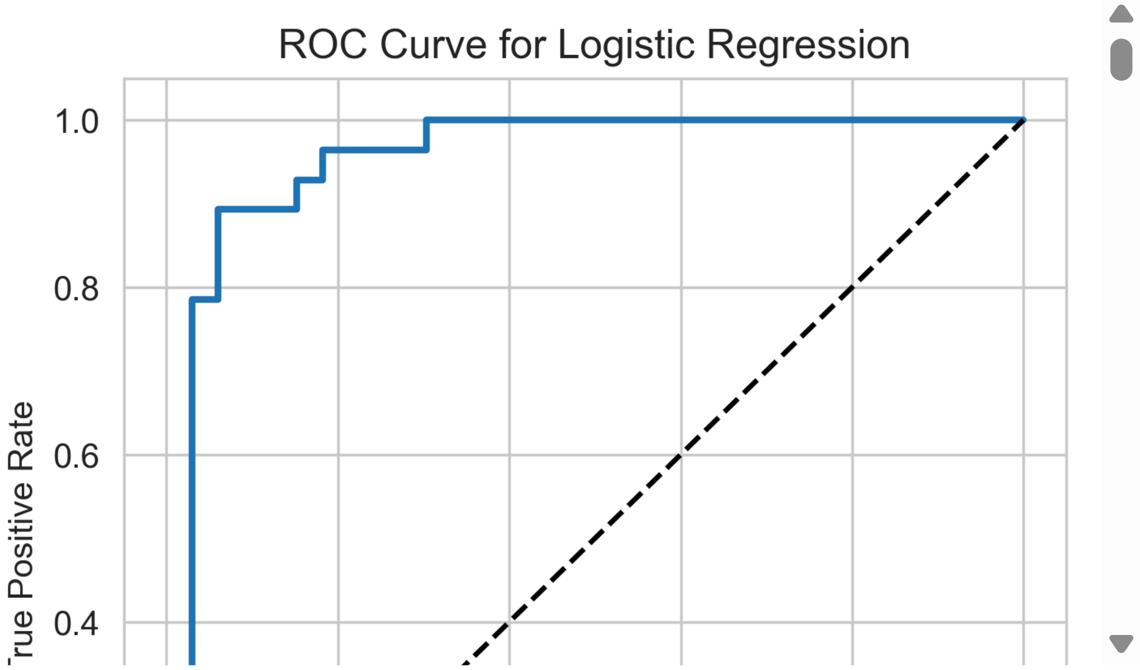
In [29]:
```
# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(5, 4),dpi=300)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["No Disease
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



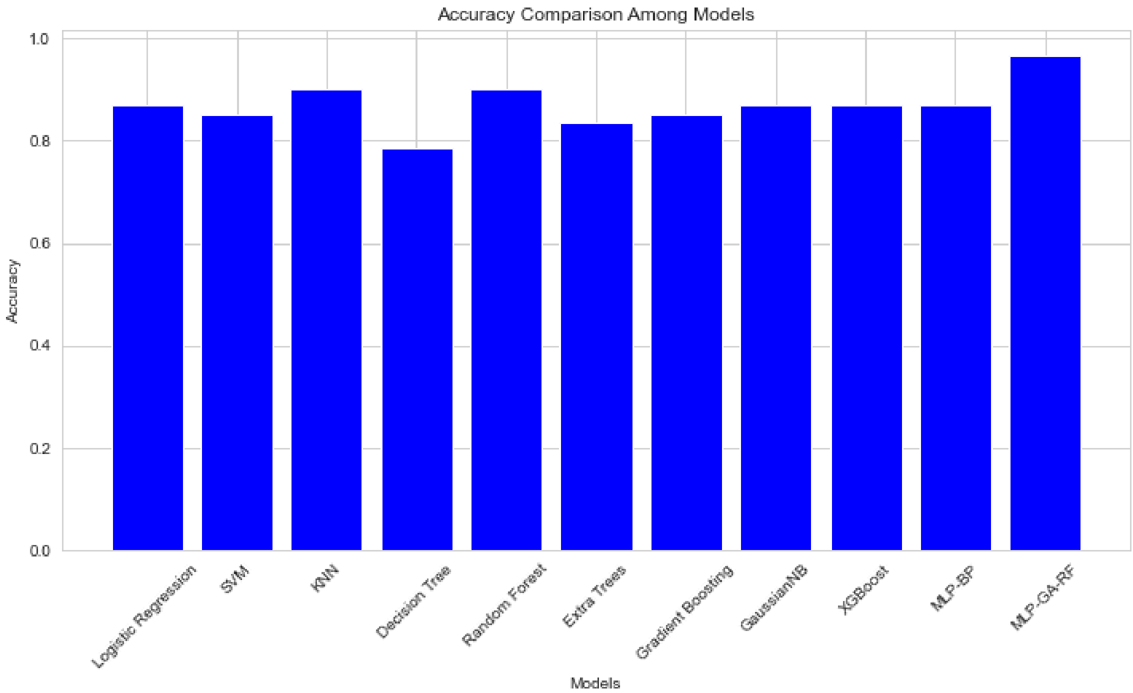
In [30]:
```
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_rf)
plt.figure(figsize=(6, 5),dpi=300)
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (AUC = {auc_rf:.2f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for MLP-GA-RF")
#plt.legend()
plt.show()
```



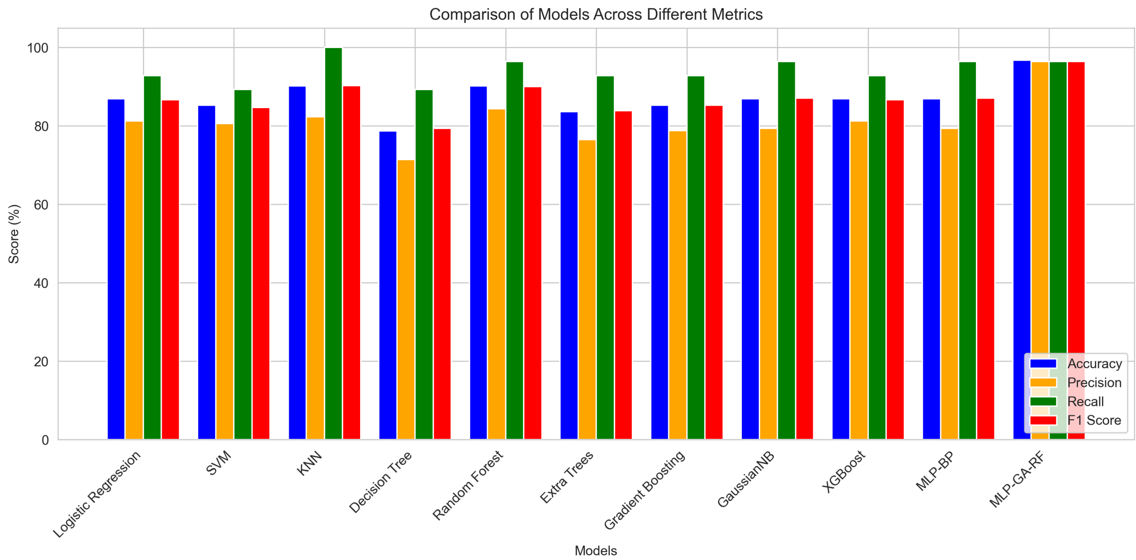
In [31]:
```
# Plot individual ROC curves for each model without AUC and with dpi=300
for name, model in models.items():
    plt.figure(figsize=(5, 5), dpi=300)  # High-resolution figure
    model.fit(X_train_scaled, y_train)
    if hasattr(model, "predict_proba"):
        y_probs = model.predict_proba(X_test_scaled)[:, 1]  # Get probabili
    else:
        y_probs = model.decision_function(X_test_scaled)  # Use decision_fu
    fpr, tpr, _ = roc_curve(y_test, y_probs)

    plt.plot(fpr, tpr, linewidth=2, label=f'{name}')
    plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {name}')
    plt.legend(loc='lower right')
    plt.show()
```



In [32]:
```
# Bar Plot for Accuracy Comparison
plt.figure(figsize=(12, 6))
plt.bar(results_df['Model'], results_df['Accuracy'], color='blue')
plt.xlabel("Models")
plt.ylabel("Accuracy")
plt.title("Accuracy Comparison Among Models")
plt.xticks(rotation=45)
plt.show()
```



In [33]:
```
# Extracting performance metrics for visualization
models_list = results_df['Model'].values
accuracy = results_df['Accuracy'].values * 100
precision = results_df['Precision'].values * 100
recall = results_df['Recall'].values * 100
f1_score_values = results_df['F1 Score'].values * 100

# Set width and positions for bars
x = np.arange(len(models_list))
width = 0.2

# Create a grouped bar chart
fig, ax = plt.subplots(figsize=(12, 6),dpi=300)
ax.bar(x - 1.5 * width, accuracy, width, label="Accuracy", color='blue')
ax.bar(x - 0.5 * width, precision, width, label="Precision", color='orange'
ax.bar(x + 0.5 * width, recall, width, label="Recall", color='green')
ax.bar(x + 1.5 * width, f1_score_values, width, label="F1 Score", color='re

# Labels and formatting
ax.set_ylabel("Score (%)")
ax.set_xlabel("Models")
ax.set_xticks(x)
ax.set_xticklabels(models_list, rotation=45, ha='right')
ax.set_title("Comparison of Models Across Different Metrics")
ax.legend(loc="lower right")

# Display the plot
plt.tight_layout()
plt.show()
```



In [ ]:

In [ ]:

In [ ]: