# Predictive analysis of naval incidents in the USA, 2002 - 2015:

## Annex 5.1. Data Model VesselBalancedSample


> Author: Oscar Anton


> Date: 2024



> Version: 0.9


# 0. Loadings

## Libraries

In [3]:
```
# System environment
import os

# Data general management
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import graphviz

# Model training
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, KFold, cross_val_score, ShuffleSpl
from sklearn.ensemble import BaggingClassifier, GradientBoostingClassifier, RandomForest
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier

# Model metrics
from sklearn.metrics import accuracy_score, mean_squared_error, r2_score, f1_score, mean
from scipy.stats import ttest_rel

# Model Export
import joblib

# Model explainers
import dalex as dx
```

## General variables

In [4]:
```
# Main data folders
merged_activity_folder = '../3.DataPreprocess/DataMergedActivity'
datasets_folder = 'Datasets'
models_folder = 'Models'
```

```
# Toggle for export data to external file
file_export_enabled = False
# Toggle for train model
train_model_enabled = False

# Available CPU cores for multiprocessing (training models)
n_jobs = os.cpu_count() - 1
# Random seed for reproducibility
seed = 42
```

### Base dataframe

In [5]:
```
# Load dataframe
VesselBalancedSample = pd.read_feather(merged_activity_folder + '/' + 'VesselBalancedSam

# Check dataframe
VesselBalancedSample.head(5)
```

Out[5]:

| | index | vessel_id | imo_number | vessel_name | vessel_class | build_year | gross_ton | length | flag_ |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 5820 | | ISABELLA MARIE | Recreational | 1999 | 14 | 32.8 | |
| **1** | 1 | 170582 | | TERMINATOR | Fishing Vessel | 1979 | 17 | 38.1 | |
| **2** | 2 | 257931 | | SUMMER ISLE | Recreational | 1984 | 8 | 29.9 | |
| **3** | 3 | 151752 | | NORJERNAN | Passenger Ship | 1976 | 18 | 35.2 | |
| **4** | 4 | 308953 | | NONSENSE | Recreational | 1987 | 8 | 27.0 | |

# 1. Dataframe creation

## 1.1. Variable transformation

> Note: Missing and NA values previously processed. Data balanced by design

### 1.1.1. Category variables: Reduce excessive variability

In [6]:
```
# Function: cutting values
def cut_years(column):
    return pd.cut(column,
                  bins=[-float('inf'), 1940, 1960, 1980, 2000, float('inf')],
                  labels=['very Old', 'old', 'average', 'new', 'very new'],
                  include_lowest=True)

# Function: group minority values
def lump_factorials(column, prop=0.008, other_level="other value"):
    counts = column.value_counts(normalize=True)
    mask = column.isin(counts[counts < prop].index)
```

```
    column[mask] = other_level
    return column

# First transformations: assign 'y': No event = 0, otherwise = 1
# Cutting, renaming, dropping not relevant variables
data_general = (
    VesselBalancedSample
    .assign(y=lambda x: pd.factorize(x['event_type'] != "No event", sort=True)[0])
    .assign(build_year=cut_years(VesselBalancedSample['build_year']))
    .rename(columns={'length': 'vessel_length'})
    .drop(columns=['vessel_id', 'imo_number', 'vessel_name', 'event_type', 'damage_statu
)

# Group minority values
data_general[['vessel_class', 'flag_abbr', 'classification_society', 'solas_desc']] = (d
    .apply(lump_factorials)
)

# Check structure
data_general.dtypes
```

Out[6]:
```
index                      int64
vessel_class              object
build_year              category
gross_ton                  int32
vessel_length            float64
flag_abbr                 object
classification_society    object
solas_desc                object
y                          int64
dtype: object
```

### 1.1.2. Category variables: one-hot-encoding

In [7]:
```
# One hot encoding for not numeric variables
data_ohe = (pd.get_dummies(data_general
                .select_dtypes(exclude=['number']))
                .astype(int))
```

### 1.1.3. Numeric variables: Scale

In [8]:
```
# Apply standard scaler
data_scaled = pd.DataFrame(StandardScaler()
               .fit_transform(data_general[['gross_ton', 'vessel_length']])
)

# Rename column names as strings
data_scaled.columns = ['gross_ton', 'vessel_length']
```

## 1.2. Join data

In [9]:
```
# Dataset join: encoded, scaled and 'y'
data_num = pd.concat([data_ohe,
                      data_scaled,
                      data_general['y']],
                      axis=1)

# Rename column names as strings
data_num.columns = data_num.columns.astype(str)
```

```
# Verify variables
for column in data_num.columns:
    print(f"Name: {column} | Type: {data_num[column].dtype} | Levels: {data_num[column].
```

```
Name: vessel_class_Barge | Type: int32 | Levels: 2
Name: vessel_class_Bulk Carrier | Type: int32 | Levels: 2
Name: vessel_class_Fishing Vessel | Type: int32 | Levels: 2
Name: vessel_class_General Dry Cargo Ship | Type: int32 | Levels: 2
Name: vessel_class_Miscellaneous Vessel | Type: int32 | Levels: 2
Name: vessel_class_Offshore | Type: int32 | Levels: 2
Name: vessel_class_Passenger Ship | Type: int32 | Levels: 2
Name: vessel_class_Recreational | Type: int32 | Levels: 2
Name: vessel_class_Tank Ship | Type: int32 | Levels: 2
Name: vessel_class_Towing Vessel | Type: int32 | Levels: 2
Name: vessel_class_other value | Type: int32 | Levels: 2
Name: build_year_very Old | Type: int32 | Levels: 2
Name: build_year_old | Type: int32 | Levels: 2
Name: build_year_average | Type: int32 | Levels: 2
Name: build_year_new | Type: int32 | Levels: 2
Name: build_year_very new | Type: int32 | Levels: 2
Name: flag_abbr_CA | Type: int32 | Levels: 2
Name: flag_abbr_LR | Type: int32 | Levels: 2
Name: flag_abbr_PA | Type: int32 | Levels: 2
Name: flag_abbr_US | Type: int32 | Levels: 2
Name: flag_abbr_other value | Type: int32 | Levels: 2
Name: classification_society_AMERICAN BUREAU OF SHIPPING | Type: int32 | Levels: 2
Name: classification_society_DET NORSKE VERITAS | Type: int32 | Levels: 2
Name: classification_society_LLOYD'S REGISTER OF SHIPPING | Type: int32 | Levels: 2
Name: classification_society_NIPPON KAIJI KYOKAI | Type: int32 | Levels: 2
Name: classification_society_UNSPECIFIED | Type: int32 | Levels: 2
Name: classification_society_other value | Type: int32 | Levels: 2
Name: solas_desc_Active SOLAS | Type: int32 | Levels: 2
Name: solas_desc_Historical SOLAS | Type: int32 | Levels: 2
Name: solas_desc_Non SOLAS | Type: int32 | Levels: 2
Name: gross_ton | Type: float64 | Levels: 6622
Name: vessel_length | Type: float64 | Levels: 4597
Name: y | Type: int64 | Levels: 2
```

In [10]:
```
# Save dataset
if file_export_enabled :
    data_num.reset_index().to_feather(datasets_folder + '/' + 'data_num.feather')
    print(f'data_num {data_num.shape} exported to {datasets_folder}')
else:
    data_num = pd.read_feather(datasets_folder + '/' + 'data_num.feather')
    print(f'data_num {data_num.shape} imported from {datasets_folder}')
```

```
data_num (109836, 34) imported from Datasets
```

## 1.3. Train / Test Split

In [11]:
```
# Data split
X_train, X_test, y_train, y_test = train_test_split(
                                    data_num.drop(columns=['y']),
                                    data_num['y'],
                                    test_size=0.2,
                                    random_state=42)
```

In [12]:
```
# Save / Load dataframes in one file (h5 format for multiple data)
if file_export_enabled :
    dfs = {'X_train':X_train, 'X_test':X_test, 'y_train':y_train, 'y_test':y_test}
    for key, df in dfs.items():
        df.to_hdf(datasets_folder + '/' + 'datasets_splited.h5', key=key, format='table'
```

```
        print(f'{key} {eval(key).shape} exported to {datasets_folder} folder')
else:
    dfs = ['X_train', 'X_test', 'y_train', 'y_test']
    for df in dfs:
        globals()[df] = pd.read_hdf(datasets_folder + '/' + 'datasets_splited.h5', key =
        print(f'{df} {eval(df).shape} imported from {datasets_folder} folder')
```

```
X_train (87868, 32) imported from Datasets folder
X_test (21968, 32) imported from Datasets folder
y_train (87868,) imported from Datasets folder
y_test (21968,) imported from Datasets folder
```

# 2. Model Training

## Performance displaying functions

In [13]:
```
# Function: Table with main metrics data
def model_metrics(model, X, y):
    # Predictions (absolute)
    y_pred = model.predict(X)

    # Calculate main metrics
    roc_auc = round(roc_auc_score(y, y_pred), 4)
    accuracy = round(accuracy_score(y, y_pred), 4)
    kappa = round(cohen_kappa_score(y, y_pred), 4)
    rmse = round(mean_squared_error(y, y_pred), 4)
    mae = round(mean_absolute_error(y, y_pred), 4)
    r2 = round(r2_score(y, y_pred), 4)
    f1 = round(f1_score(y, y_pred), 4)

    # Sensitivity And Specificity
    tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()
    sensitivity =  round(tp / (tp + fn), 4)
    specificity = round(tn / (tn + fp), 4)

    # Build multiindex table
    metrics_df = pd.DataFrame([['ROC AUC:', roc_auc],['Accuracy:', accuracy], ['Kappa:',
                               ['RMSE:', rmse], ['MAE:', mae], ['R2:', r2], ['F1:', f1],['
                               ['Sensitivity:', sensitivity], ['Specificity:', specificity]
                               columns=pd.MultiIndex.from_product([[model.__class__.__name_

    return metrics_df.style.hide()
```

In [14]:
```
# Function: Table with Confusion Matrix data
def confusion_matrix_table(model, X, y):
    # Predictions (absolute)
    y_pred = model.predict(X)

    # Confusion matrix
    tn, fp, fn, tp = confusion_matrix(y, y_pred).ravel()

    # Dataframe creation
    df = pd.DataFrame([[tp, fn],[fp, tn]],
                  index=pd.Index(['1', '0'], name='Actual Label:'),
                  columns=pd.MultiIndex.from_product([[model.__class__.__name__],['1', '

    # Dataframe style
    styled_df = df.style.set_table_styles([
        {'selector': 'th.col_heading', 'props': 'text-align: center;'},
        {'selector': 'td', 'props': 'text-align: center;'},
```

```
    ], overwrite=False)

    return styled_df
```

In [15]:
```
# Function: Plot ROC Curve
def plot_roc_curve(model, X, y):
    # Predicted probabilities
    y_score = model.predict_proba(X)

    # Calculate ROC for each class
    fpr, tpr, _ = roc_curve(y, y_score[:, 1])

    # Calculate AUC (Area Under Curve)
    roc_auc = auc(fpr, tpr)

    # Plot ROC Curve
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC = %0.2f)' % roc_a
    plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='dotted')
    plt.xlim([0, 1])
    plt.ylim([0, 1.01])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {model.__class__.__name__}')
    plt.legend(loc="lower right")
    plt.show()
```

# 2.1. Bayesian networks

## 2.1.1. Naïve Bayes -Gaussian- (NB)

### Model Training

In [16]:
```
if train_model_enabled :
    # Define model parameters
    params = {}                          # No parameters for this model

    # Create and train model
    model = GaussianNB(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'nb_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'nb_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: GaussianNB
Parameters:
 priors: None
 var_smoothing: 1e-09
```

### Model Main Metrics

In [17]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[17]:

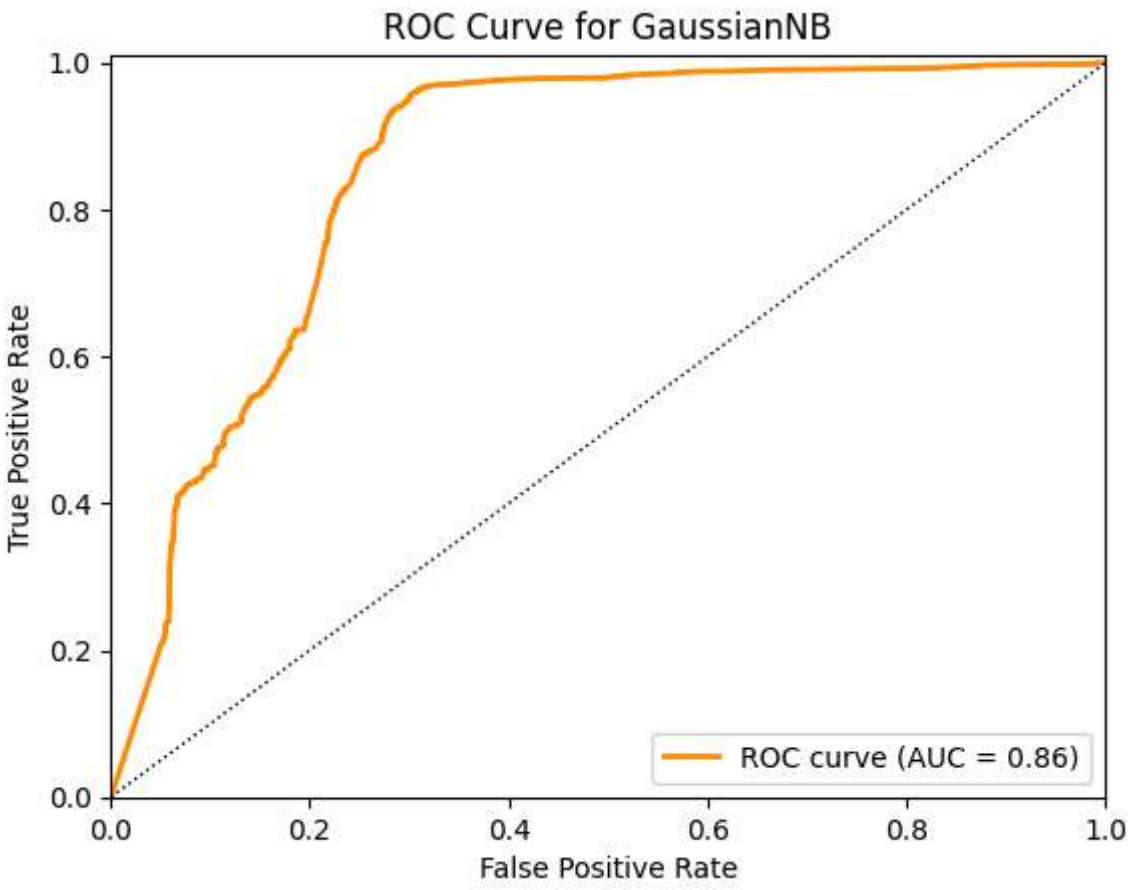
| | GaussianNB |
|---|---|
| **Metric** | **Value** |
| ROC AUC: | 0.673700 |
| Accuracy: | 0.674900 |
| Kappa: | 0.348200 |
| RMSE: | 0.325100 |
| MAE: | 0.325100 |
| R2: | -0.300300 |
| F1: | 0.580000 |
| | |
| Sensitivity: | 0.451400 |
| Specificity: | 0.895900 |

In [18]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[18]:

| **Model:** | **GaussianNB** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 4930 | 5991 |
| **0** | 1150 | 9897 |

In [19]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




## 2.1.2. Bagged Naïve Bayes -Gaussian- (BNB)

### Model Training

In [20]:
```
if train_model_enabled :
    # Define model parameters
    params = {}                          # No parameters for this model

    # Create and train model.
    model = BaggingClassifier(GaussianNB(**params), n_estimators=5).fit(X_train, y_train

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'bnb_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'bnb_train.pkl')

# Bagging (Bootstrap Aggregating) is a technique used to improve the stability and accur
# n_estimators specifies how many individual Naive Bayes classifiers are trained indepen
# It approximates the behavior of AODE

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: BaggingClassifier
Parameters:
 bootstrap: True
 bootstrap_features: False
 estimator__priors: None
 estimator__var_smoothing: 1e-09
 estimator: GaussianNB()
 max_features: 1.0
 max_samples: 1.0
 n_estimators: 5
 n_jobs: None
 oob_score: False
 random_state: None
 verbose: 0
 warm_start: False
```

## Model Main Metrics

In [21]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[21]:

**BaggingClassifier**

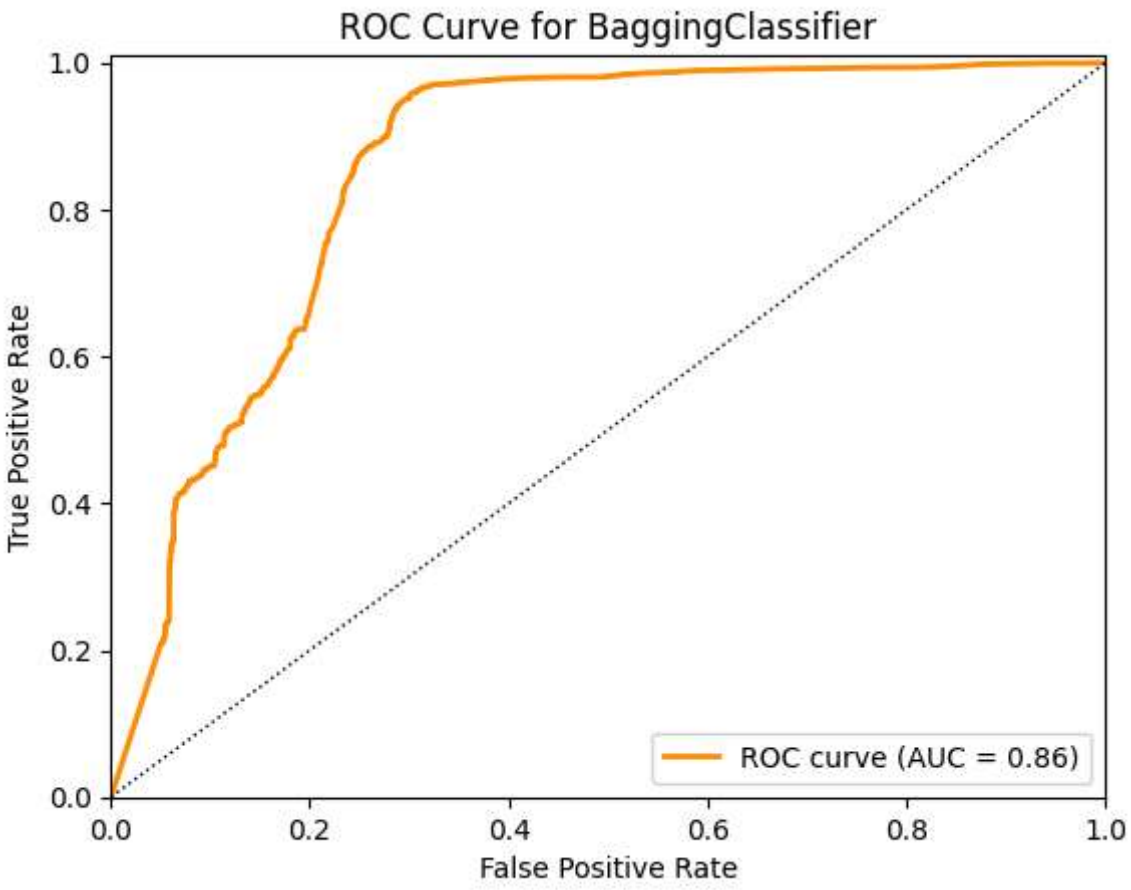
| Metric | Value |
|---|---|
| ROC AUC: | 0.673800 |
| Accuracy: | 0.675100 |
| Kappa: | 0.348600 |
| RMSE: | 0.324900 |
| MAE: | 0.324900 |
| R2: | -0.299600 |
| F1: | 0.579900 |
| | |
| Sensitivity: | 0.451000 |
| Specificity: | 0.896700 |

In [22]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[22]:

| **Model:** | **BaggingClassifier** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 4925 | 5996 |
| **0** | 1141 | 9906 |

In [23]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




# 2.2. Gradient Boosting Models

## 2.2.1. Gradient Boosting Machine (GBM)

### Model Training

In [24]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'n_estimators': 500,            # Number of trees
        'learning_rate': 0.1,           # Contribution of each tree to the model
        'max_depth': 3,                 # Maximum levels of each tree
        'random_state': seed,           # Random seed for reproducibility
    }

    # Create and train model
    model = GradientBoostingClassifier(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'GBM_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'GBM_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: GradientBoostingClassifier
Parameters:
 ccp_alpha: 0.0
 criterion: friedman_mse
 init: None
 learning_rate: 0.1
 loss: log_loss
 max_depth: 3
 max_features: None
 max_leaf_nodes: None
 min_impurity_decrease: 0.0
 min_samples_leaf: 1
 min_samples_split: 2
 min_weight_fraction_leaf: 0.0
 n_estimators: 500
 n_iter_no_change: None
 random_state: 42
 subsample: 1.0
 tol: 0.0001
 validation_fraction: 0.1
 verbose: 0
 warm_start: False
```

## Model Main Metrics

In [25]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[25]:

**GradientBoostingClassifier**

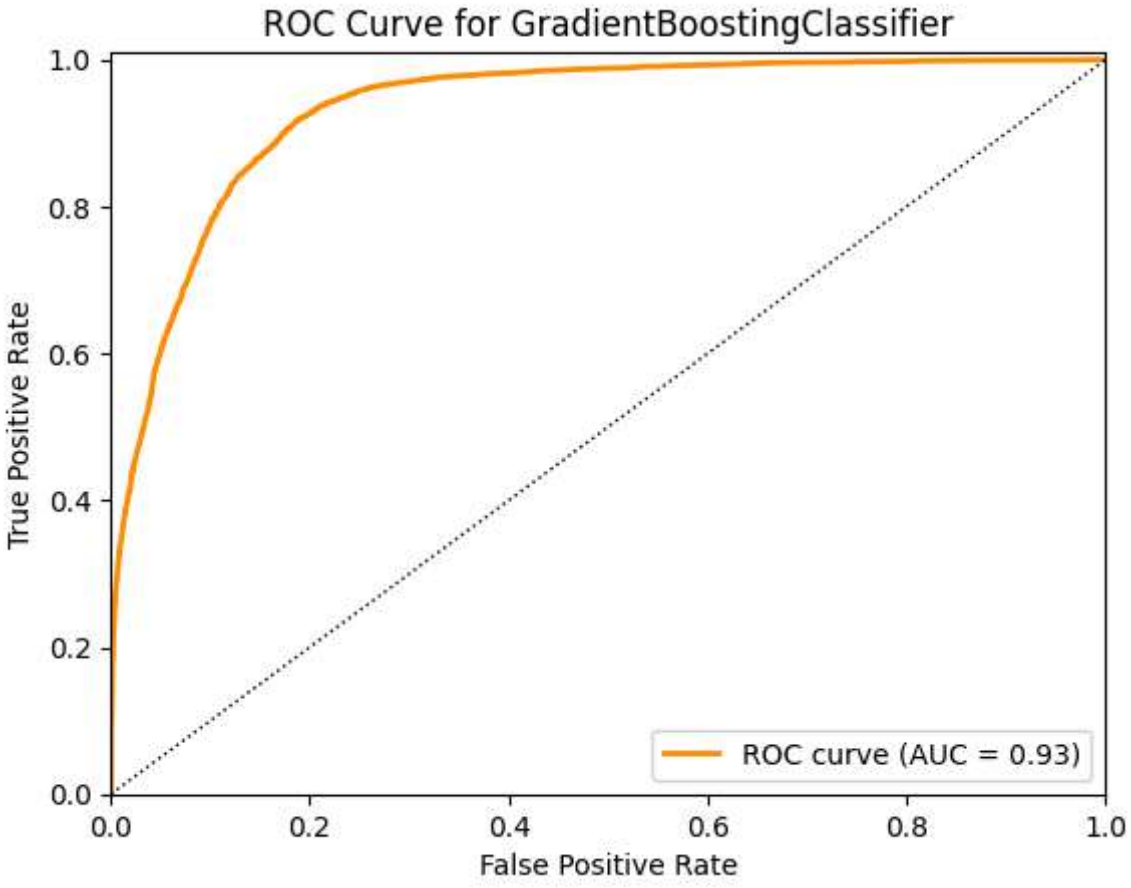
| Metric | Value |
|---|---|
| ROC AUC: | 0.865100 |
| Accuracy: | 0.864800 |
| Kappa: | 0.729900 |
| RMSE: | 0.135200 |
| MAE: | 0.135200 |
| R2: | 0.459400 |
| F1: | 0.871000 |
| | |
| Sensitivity: | 0.917600 |
| Specificity: | 0.812700 |

In [26]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[26]:

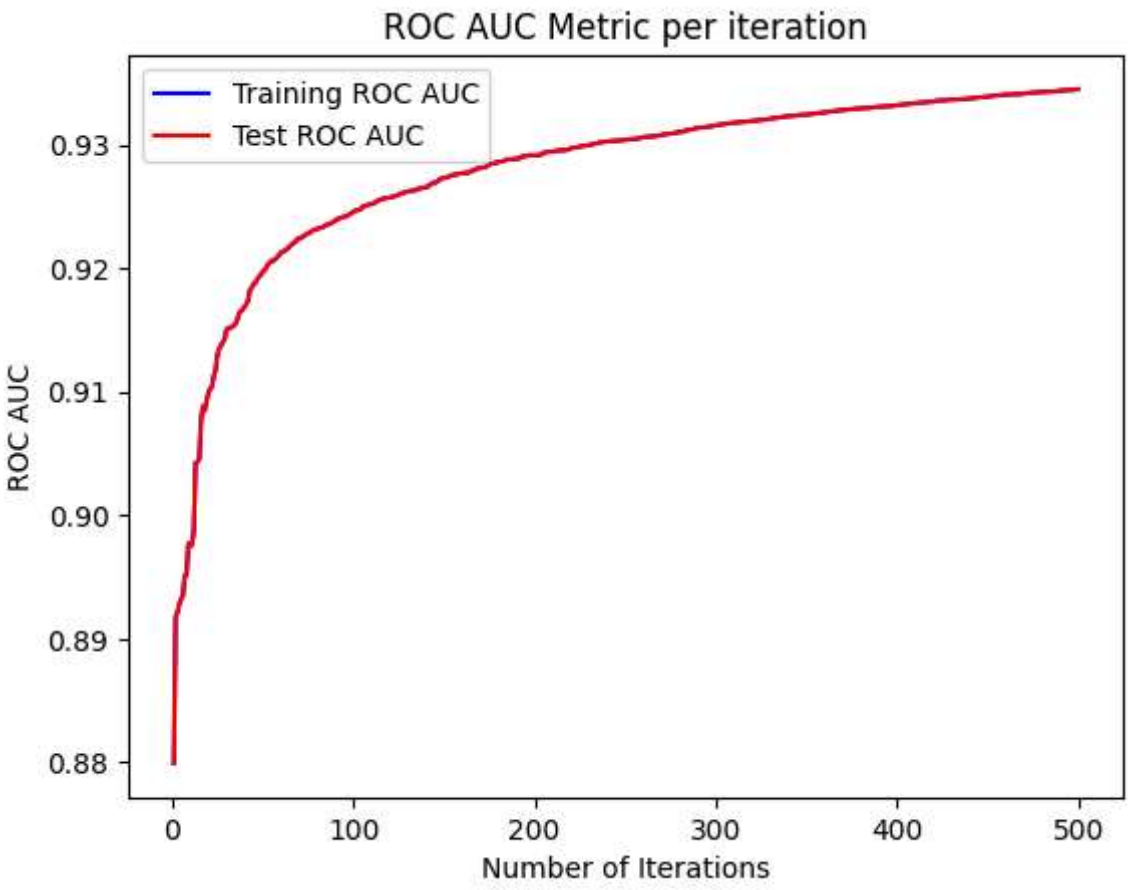
| **Model:** | **GradientBoostingClassifier** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 10021 | 900 |
| **0** | 2069 | 8978 |

In [27]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```

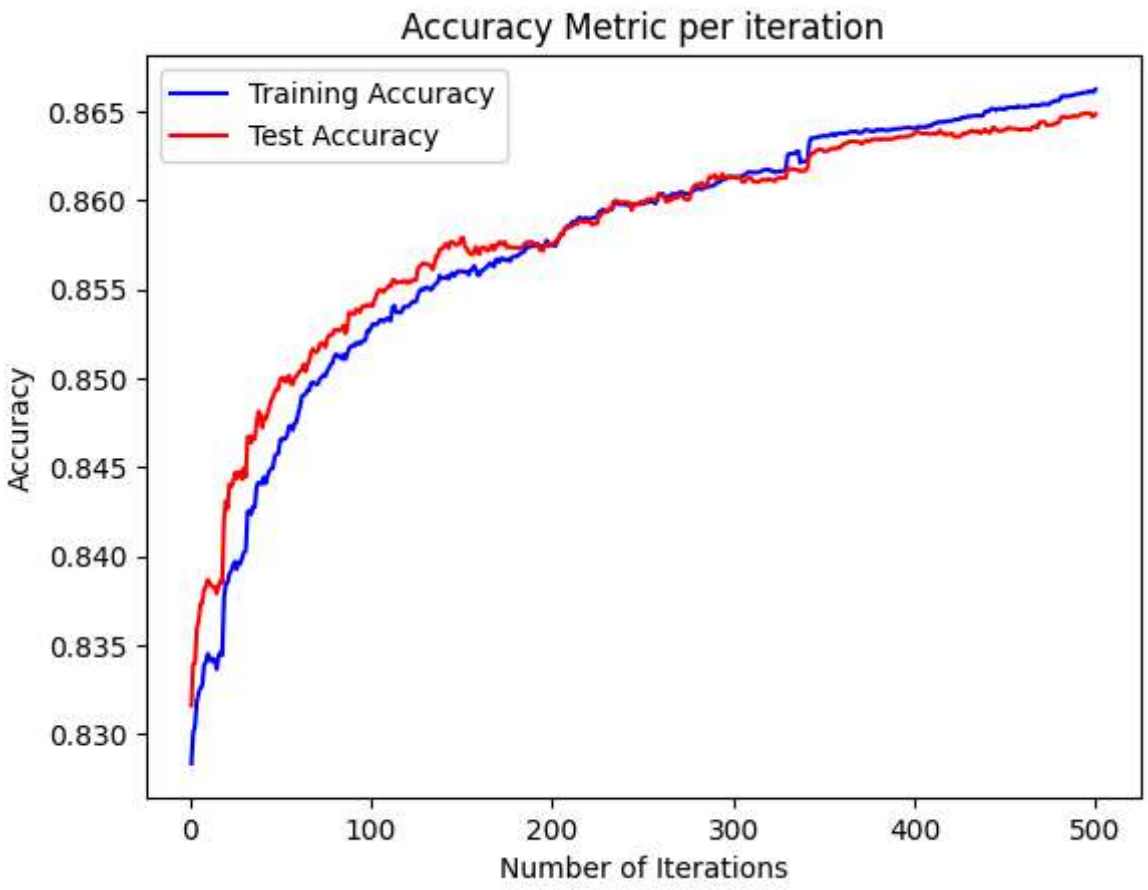



## Extra: Main metrics per iteration (n_estimators=500)

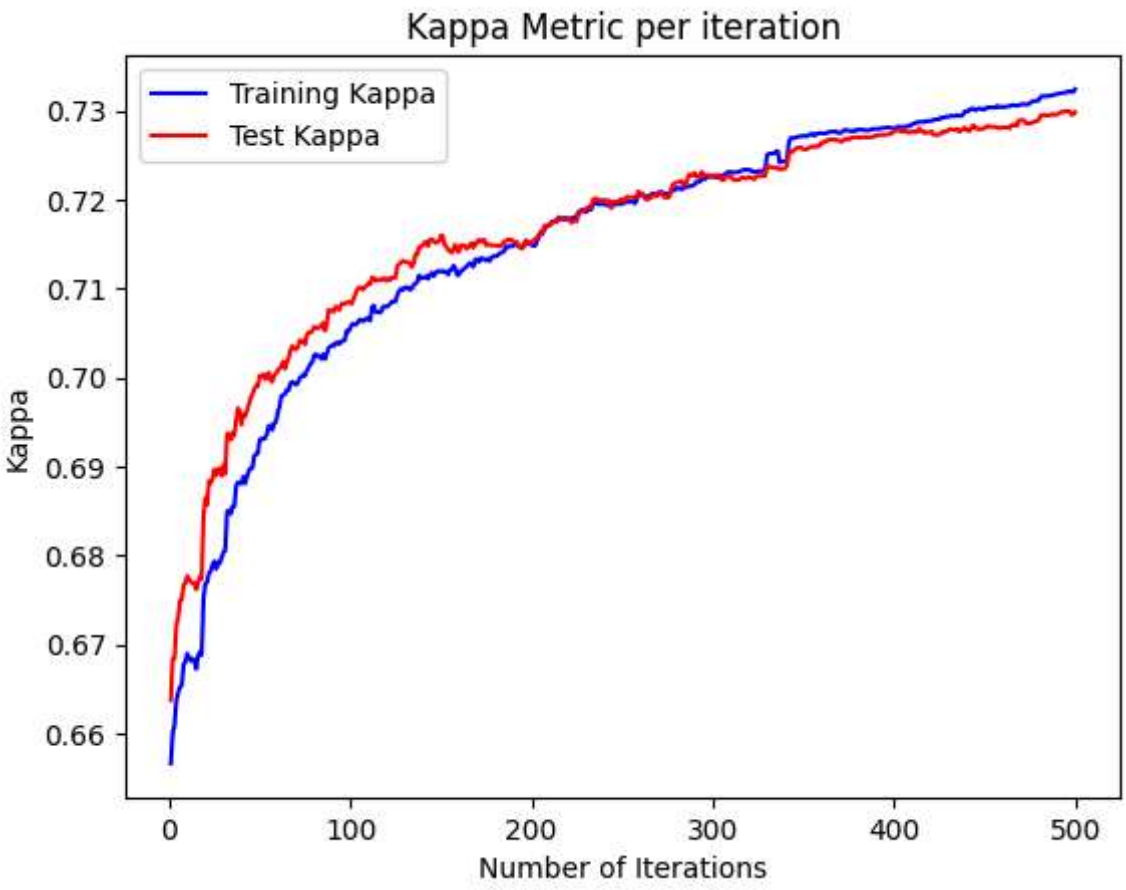
In [28]:
```
# Number of iterations during training
iterations = int(model.get_params().get("n_estimators") + 1)

# Obtain prediction probabilities at each stage of training
train_probs = np.array([preds[:, 1] for preds in model.staged_predict_proba(X_train)])
test_probs = np.array([preds[:, 1] for preds in model.staged_predict_proba(X_test)])

# Calculate ROC metric at each stage of training
train_roc_auc = [roc_auc_score(y_train, prob) for prob in train_probs]
test_roc_auc = [roc_auc_score(y_train, prob) for prob in train_probs]

# Plot the ROC metric as a function of iterations
plt.plot(range(1, iterations), train_roc_auc, label="Training ROC AUC", color="blue")
plt.plot(range(1, iterations), test_roc_auc, label="Test ROC AUC", color="red")
plt.xlabel("Number of Iterations")
plt.ylabel("ROC AUC")
plt.title("ROC AUC Metric per iteration")
plt.legend()
plt.show()
```



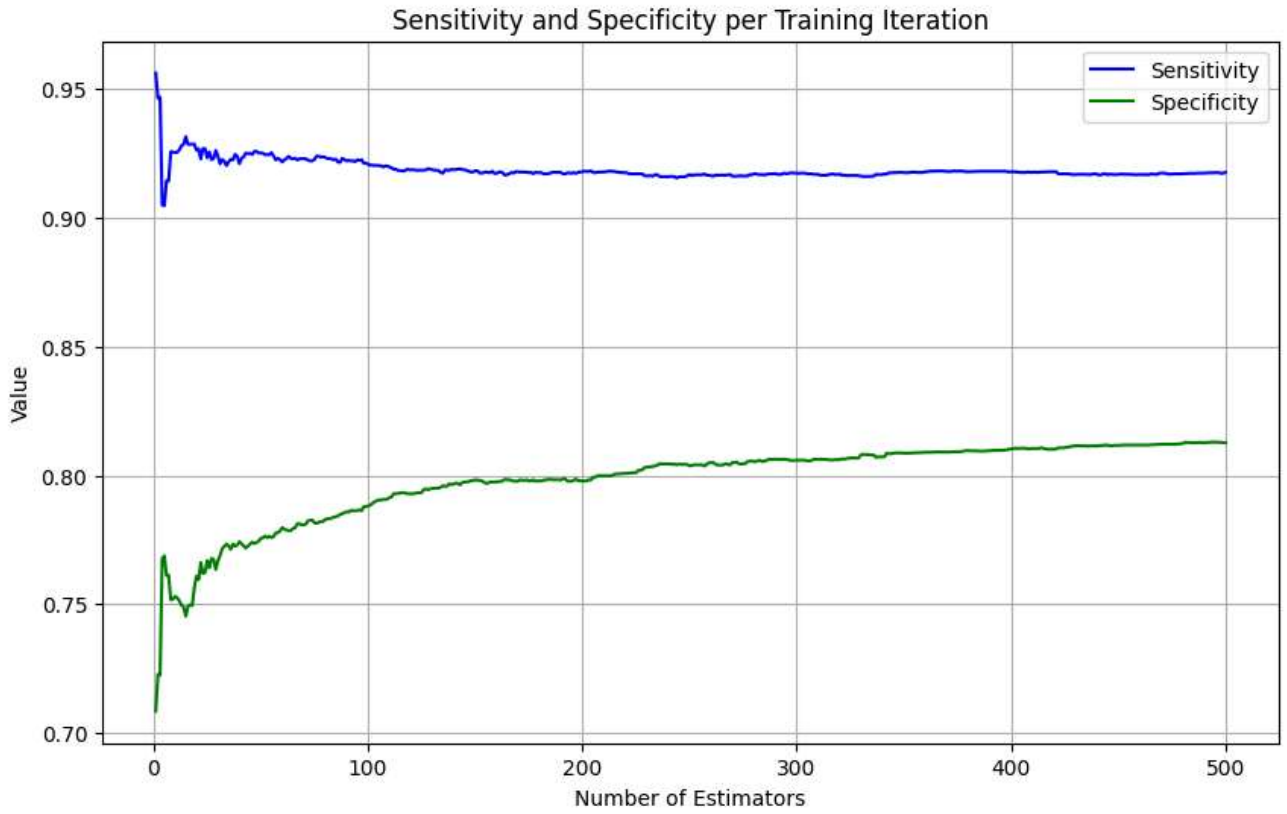

In [29]:
```
# Get predictions at each stage of training
train_errors = [accuracy_score(y_train, y_pred) for y_pred in model.staged_predict(X_tra
test_errors = [accuracy_score(y_test, y_pred) for y_pred in model.staged_predict(X_test)

# Plot the Accuracy metric as a function of iterations
plt.plot(np.arange(1, iterations), train_errors, label="Training Accuracy", color="blue"
plt.plot(np.arange(1, iterations), test_errors, label="Test Accuracy", color="red")
plt.xlabel("Number of Iterations")
plt.ylabel("Accuracy")
plt.title("Accuracy Metric per iteration")
plt.legend()
plt.show()
```




In [30]:
```
# Get predictions at each stage of training
train_kappa = [cohen_kappa_score(y_train, y_pred) for y_pred in model.staged_predict(X_t
test_kappa = [cohen_kappa_score(y_test, y_pred) for y_pred in model.staged_predict(X_tes

# Plot the Kappa metric as a function of iterations
plt.plot(np.arange(1, iterations), train_kappa, label="Training Kappa", color="blue")
plt.plot(np.arange(1, iterations), test_kappa, label="Test Kappa", color="red")
plt.xlabel("Number of Iterations")
plt.ylabel("Kappa")
plt.title("Kappa Metric per iteration")
plt.legend()
plt.show()
```




In [31]:
```
sensitivities = []
specificities = []

# Get values per each iteration
for i, y_pred in enumerate(model.staged_predict(X_test)):
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
    sensitivity = tp / (tp + fn)
    specificity = tn / (tn + fp)
    sensitivities.append(sensitivity)
    specificities.append(specificity)

# Plot sensitivity and specificity for each iteration of training
plt.figure(figsize=(10, 6))
plt.plot(range(1, iterations), sensitivities, label='Sensitivity', color='blue')
plt.plot(range(1, iterations), specificities, label='Specificity', color='green')
plt.xlabel('Number of Estimators')
plt.ylabel('Value')
plt.title('Sensitivity and Specificity per Training Iteration')
plt.legend()
plt.grid(True)
plt.show()
```




## 2.2.2. Extreme Gradient Boosting (XGB)

### Model Training

In [32]:
```
if train_model_enabled :
    # Define model parameters
    params = {
    'objective': 'binary:logistic',     # Binary classification problem
    'eval_metric': 'error',             # Evaluation metric: error rate
    'eta': 0.1,                         # Learning rate
    'max_depth': 3,                     # Maximum tree depth
    'colsample_bytree': 0.3,            # Subsample ratio of columns when constructing e
    'random_state': seed                # Random seed for reproducibility
    }

    # Create and train model
    model = XGBClassifier(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'XGB_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'XGB_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: XGBClassifier
Parameters:
 objective: binary:logistic
 base_score: None
 booster: None
 callbacks: None
 colsample_bylevel: None
 colsample_bynode: None
 colsample_bytree: 0.3
 device: None
 early_stopping_rounds: None
 enable_categorical: False
 eval_metric: error
 feature_types: None
 gamma: None
 grow_policy: None
 importance_type: None
 interaction_constraints: None
 learning_rate: None
 max_bin: None
 max_cat_threshold: None
 max_cat_to_onehot: None
 max_delta_step: None
 max_depth: 3
 max_leaves: None
 min_child_weight: None
 missing: nan
 monotone_constraints: None
 multi_strategy: None
 n_estimators: None
 n_jobs: None
 num_parallel_tree: None
 random_state: 42
 reg_alpha: None
 reg_lambda: None
 sampling_method: None
 scale_pos_weight: None
 subsample: None
 tree_method: None
 validate_parameters: None
 verbosity: None
 eta: 0.1
```

## Model Main Metrics

In [33]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[33]:

**XGBClassifier**

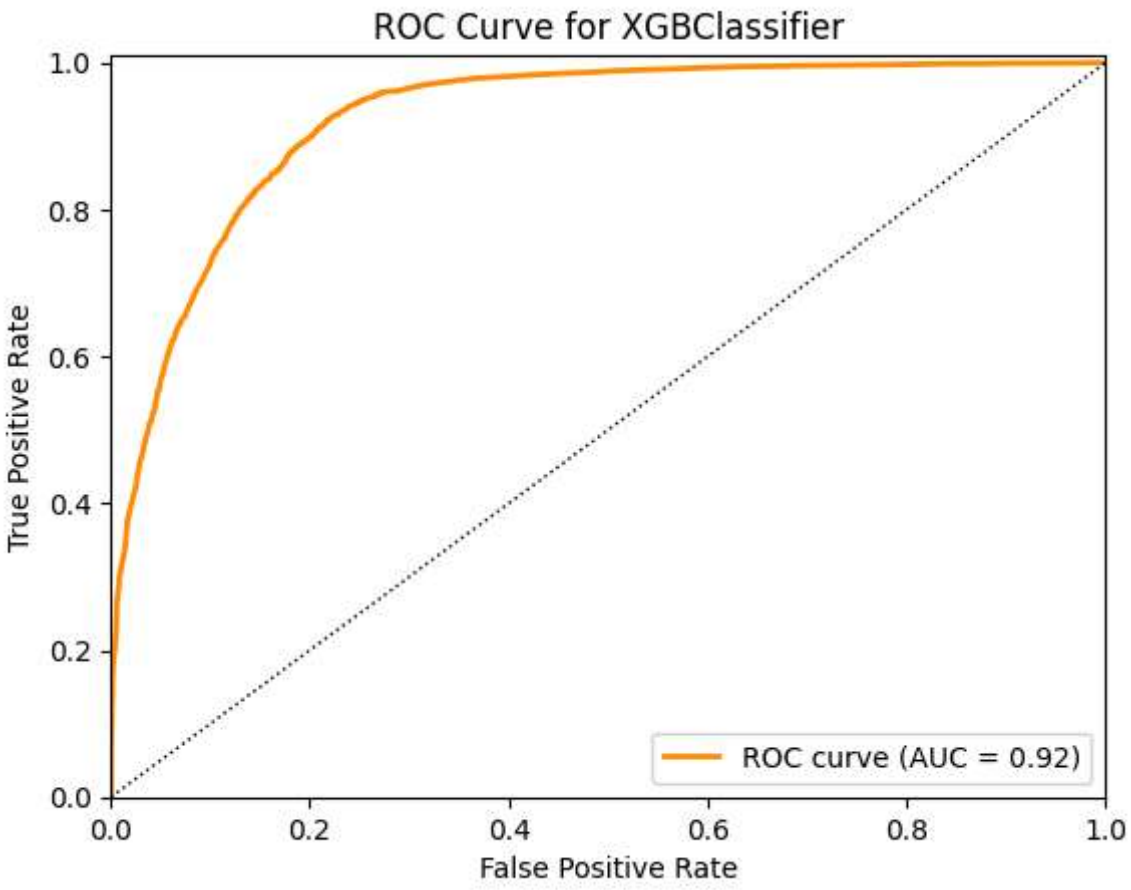
| Metric | Value |
|---|---|
| ROC AUC: | 0.851500 |
| Accuracy: | 0.851100 |
| Kappa: | 0.702400 |
| RMSE: | 0.148900 |
| MAE: | 0.148900 |
| R2: | 0.404400 |
| F1: | 0.859700 |
| | |
| Sensitivity: | 0.917500 |
| Specificity: | 0.785500 |

In [34]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[34]:

| **Model:** | **XGBClassifier** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 10020 | 901 |
| **0** | 2370 | 8677 |

In [35]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




# 2.3. More models

## 2.3.1. Random Forest (RF)

### Model Training

In [36]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'n_estimators': 100,            # Number of trees in the forest
        'max_depth': None,              # Maximum depth of the trees (no restrictions)
        'random_state': seed            # Random seed for reproducibility
    }

    # Create and train model
    model = RandomForestClassifier(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'rf_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'rf_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: RandomForestClassifier
Parameters:
 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: None
 oob_score: False
 random_state: 42
 verbose: 0
 warm_start: False
```

## Model Main Metrics

In [37]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[37]:

**RandomForestClassifier**

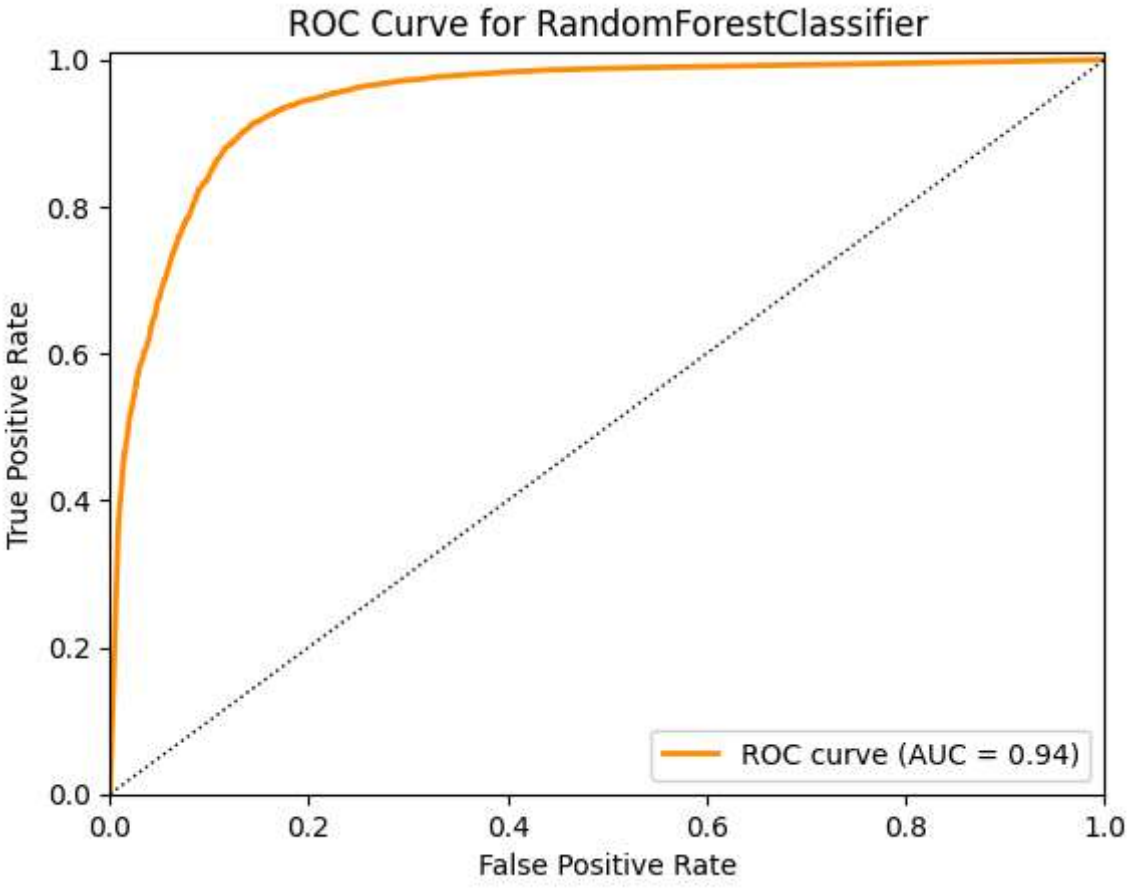
| Metric | Value |
|---|---|
| ROC AUC: | 0.883000 |
| Accuracy: | 0.882800 |
| Kappa: | 0.765700 |
| RMSE: | 0.117200 |
| MAE: | 0.117200 |
| R2: | 0.531100 |
| F1: | 0.886600 |
| | |
| Sensitivity: | 0.921300 |
| Specificity: | 0.844700 |

In [38]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[38]:

| **Model:** | **RandomForestClassifier** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 10062 | 859 |
| **0** | 1716 | 9331 |

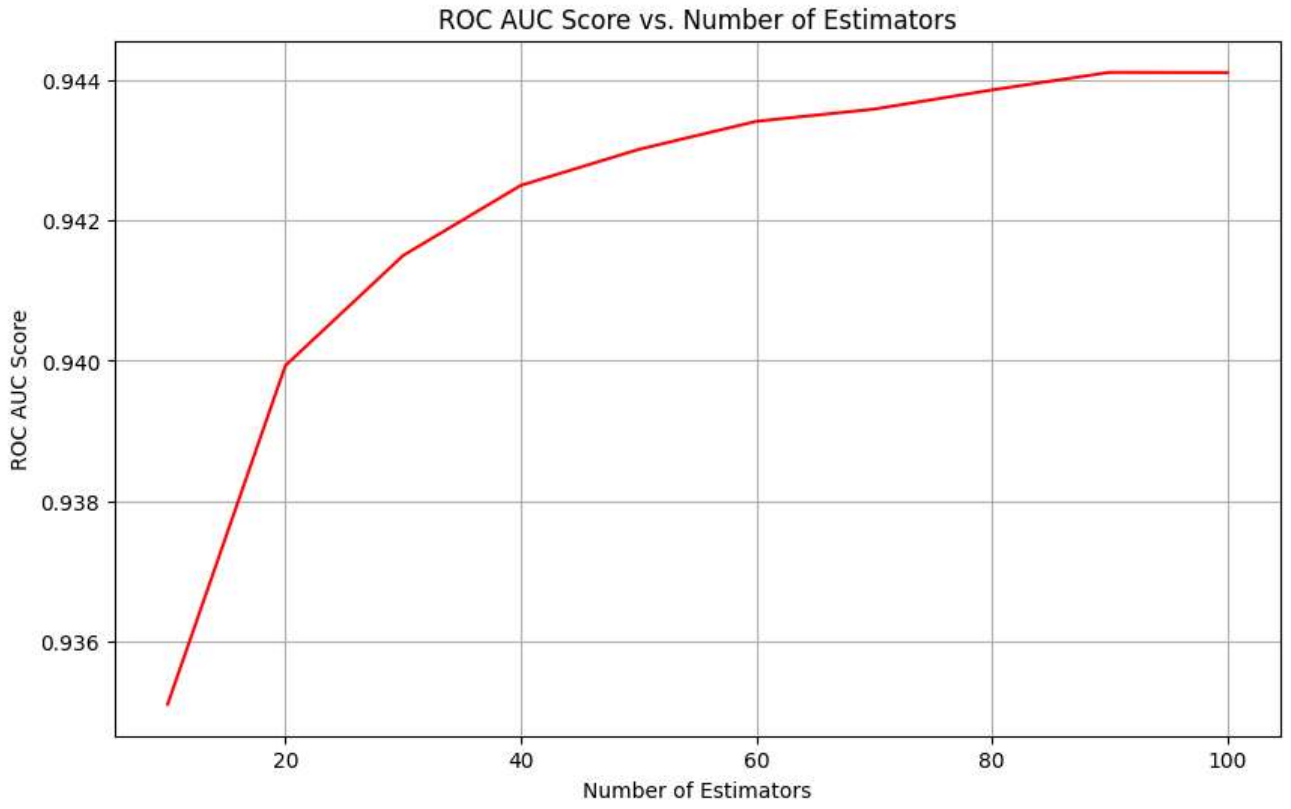
In [39]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```



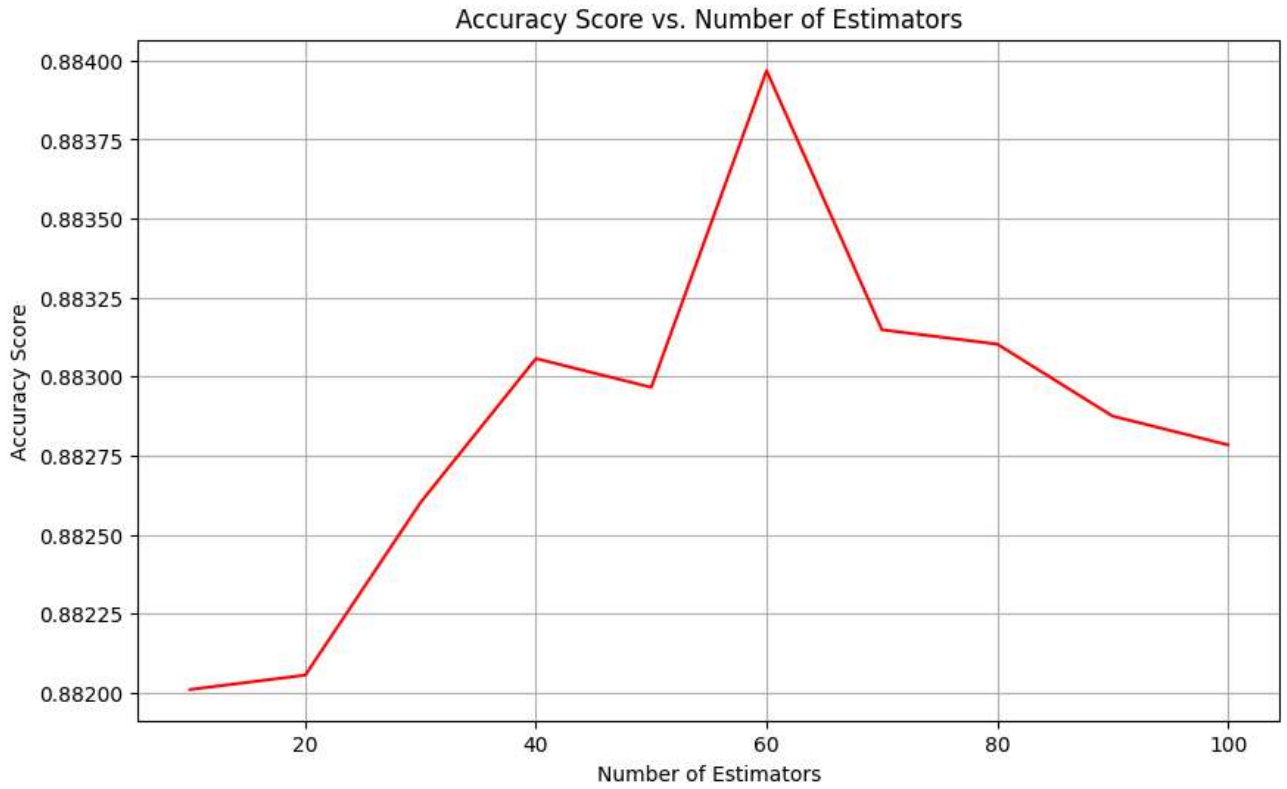

## Extra: Main metrics per iteration (n_estimators=100)

In [40]:
```
# Define the number of iterations, initial number of estimators, and step size
n_iterations = 10
initial_n_estimators = 10
step_size = 10

# Create empty lists to store the number of estimators and ROC AUC scores
num_estimators_list = []
roc_auc_scores = []

# Iterate over the specified number of iterations
for i in range(n_iterations):
    # Create the Random Forest model with the current number of estimators
    model = RandomForestClassifier(n_estimators=initial_n_estimators + i * step_size, ra

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions and calculate ROC AUC score
    y_pred_proba = model.predict_proba(X_test)[:, 1]
    roc_auc = roc_auc_score(y_test, y_pred_proba)

    # Store the number of estimators and ROC AUC score
    num_estimators_list.append(initial_n_estimators + i * step_size)
    roc_auc_scores.append(roc_auc)

# Plot the ROC AUC scores over the number of estimators
plt.figure(figsize=(10, 6))
plt.plot(num_estimators_list, roc_auc_scores, label="ROC AUC", color="red")
plt.title('ROC AUC Score vs. Number of Estimators')
plt.xlabel('Number of Estimators')
```

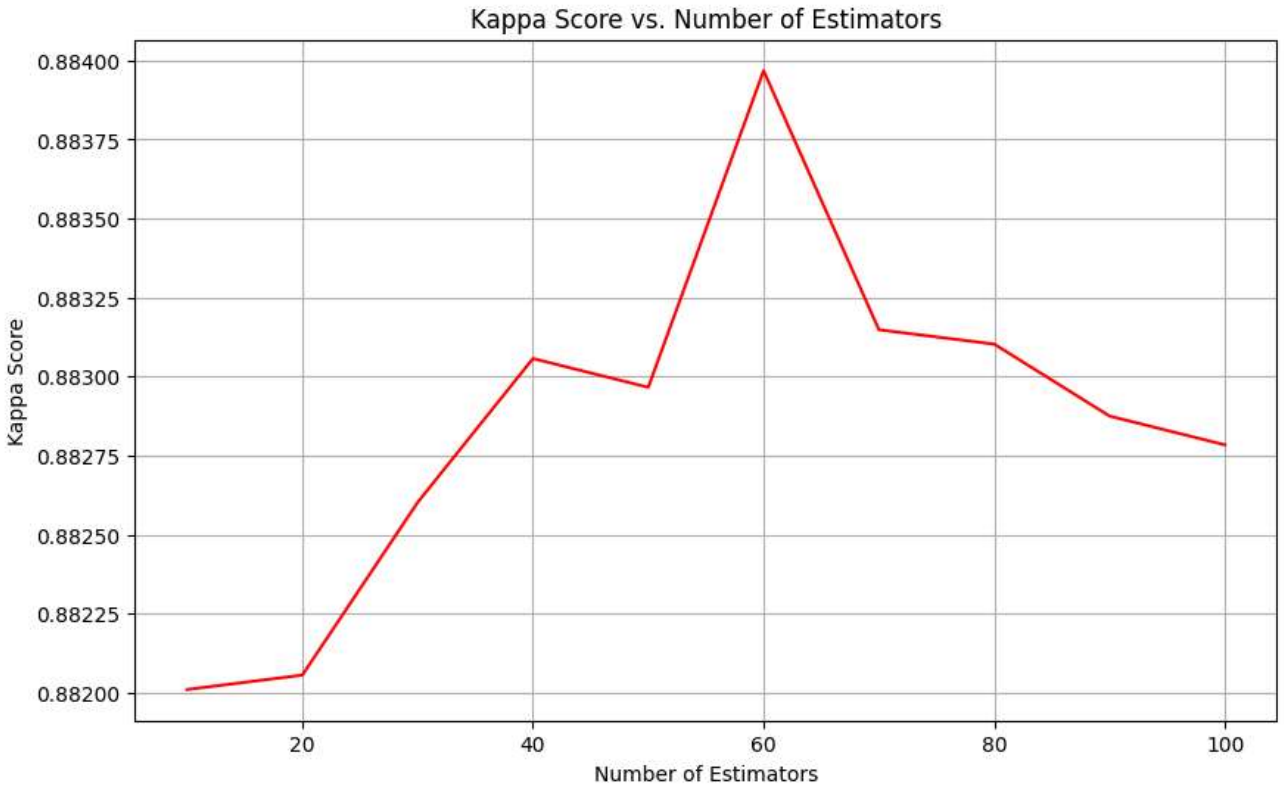
```
plt.ylabel('ROC AUC Score')
plt.grid(True)
```




In [41]:
```
# Create empty lists to store the number of estimators and ROC AUC scores
num_estimators_list = []
accuracy_auc_scores = []

# Iterate over the specified number of iterations
for i in range(n_iterations):
    # Create the Random Forest model with the current number of estimators
    model = RandomForestClassifier(n_estimators=initial_n_estimators + i * step_size, ra

    # Train the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Calculate Accuracy scores
    accuracy_auc = accuracy_score(y_test, y_pred)

    # Store the number of estimators and Accuracy score
    num_estimators_list.append(initial_n_estimators + i * step_size)
    accuracy_auc_scores.append(accuracy_auc)

# Plot the Accuracy scores over the number of estimators
plt.figure(figsize=(10, 6))
plt.plot(num_estimators_list, accuracy_auc_scores, label="Accuracy", color="red")
plt.title('Accuracy Score vs. Number of Estimators')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy Score')
plt.grid(True)
```




In [42]:
```python
# Create empty lists to store the number of estimators and ROC AUC scores
num_estimators_list = []
kappa_scores = []

# Iterate over the specified number of iterations
for i in range(n_iterations):
    # Create the Random Forest model with the current number of estimators
    model = RandomForestClassifier(n_estimators=initial_n_estimators + i * step_size, ra

    # Train the model
    model.fit(X_train, y_train)

    # Predictions
    y_pred = model.predict(X_test)

    # Calculate Kappa scores
    kappa = cohen_kappa_score(y_test, y_pred)

    # Store the number of estimators and Kappa scores
    num_estimators_list.append(initial_n_estimators + i * step_size)
    kappa_scores.append(kappa)

# Plot the Kappa scores over the number of estimators
plt.figure(figsize=(10, 6))
plt.plot(num_estimators_list, accuracy_auc_scores, label="Kappa", color="red")
plt.title('Kappa Score vs. Number of Estimators')
plt.xlabel('Number of Estimators')
plt.ylabel('Kappa Score')
plt.grid(True)
```




### Extra: Explore structure of a tree example

In [43]:
```
if file_export_enabled :
    # Extract a random model tree (for example, the first one)
    tree = model.estimators_[0]

    # Export the tree to Graphviz
    dot_data = export_graphviz(tree, out_file=None,
                               filled=True, rounded=True,
                               special_characters=True, precision=5)
    graph = graphviz.Source(dot_data)

    # Export to pdf file (lighter than svg)
    graph.render('rf_train_tree', format='pdf', cleanup=True)

# Note: File is not displayed because is extremely width
```

## 2.3.2. Support Vector Machine -Radial Kernel- (SVM)

### Model Training

In [44]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'kernel': 'rbf',                # Radial Kernel
        'probability': True,            # Enable probability estimates (uses 5-fold cros
        'random_state': seed            # Random seed for reproducibility
    }

    # Create and train model
    model = SVC(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'svm_train.pkl')

else:
```

```
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'svm_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: SVC
Parameters:
 C: 1.0
 break_ties: False
 cache_size: 200
 class_weight: None
 coef0: 0.0
 decision_function_shape: ovr
 degree: 3
 gamma: scale
 kernel: rbf
 max_iter: -1
 probability: True
 random_state: 42
 shrinking: True
 tol: 0.001
 verbose: False
```

## Model Main Metrics

In [45]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[45]:

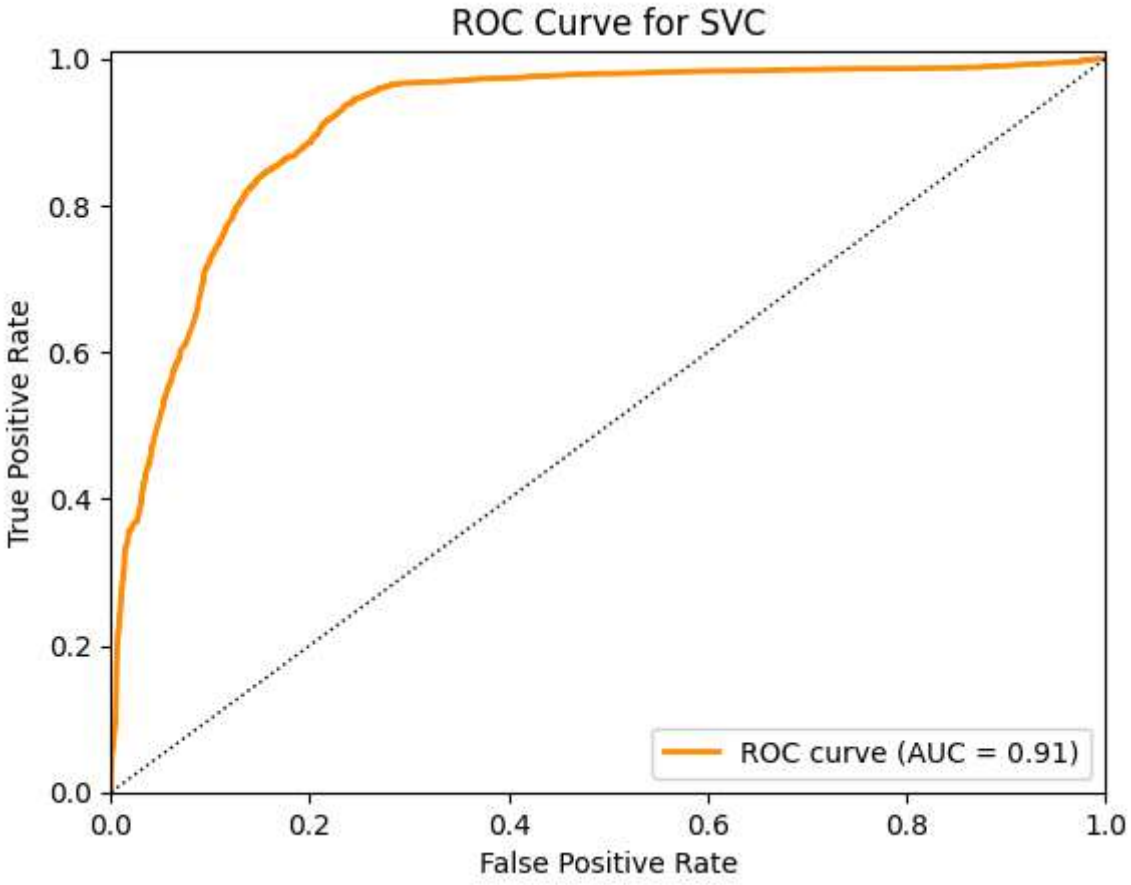
| | SVC |
|---|---|
| **Metric** | **Value** |
| ROC AUC: | 0.849200 |
| Accuracy: | 0.848700 |
| Kappa: | 0.697700 |
| RMSE: | 0.151300 |
| MAE: | 0.151300 |
| R2: | 0.394700 |
| F1: | 0.860900 |
| | |
| Sensitivity: | 0.941600 |
| Specificity: | 0.756900 |

In [46]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[46]:

| Model: | SVC | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 10283 | 638 |
| **0** | 2686 | 8361 |

In [47]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




## 2.3.3. Multilayer Perceptron (MLP)

### Model Training

In [48]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'hidden_layer_sizes': (100, 50),     # Two hidden layers with 100 and 50 neurons
        'activation': 'relu',                # Activation function for the hidden layers:
        'solver': 'adam',                    # Optimization algorithm
        'random_state': seed                 # Random seed for reproducibility
    }

    # Create and train model
    model = MLPClassifier(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'nnet_train.pkl')

else:
    # Load model from external file
```

```
    model = joblib.load(models_folder + '/' + 'nnet_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: MLPClassifier
Parameters:
 activation: relu
 alpha: 0.0001
 batch_size: auto
 beta_1: 0.9
 beta_2: 0.999
 early_stopping: False
 epsilon: 1e-08
 hidden_layer_sizes: (100, 50)
 learning_rate: constant
 learning_rate_init: 0.001
 max_fun: 15000
 max_iter: 200
 momentum: 0.9
 n_iter_no_change: 10
 nesterovs_momentum: True
 power_t: 0.5
 random_state: 42
 shuffle: True
 solver: adam
 tol: 0.0001
 validation_fraction: 0.1
 verbose: False
 warm_start: False
```

## Model Main Metrics

In [49]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[49]:

**MLPClassifier**

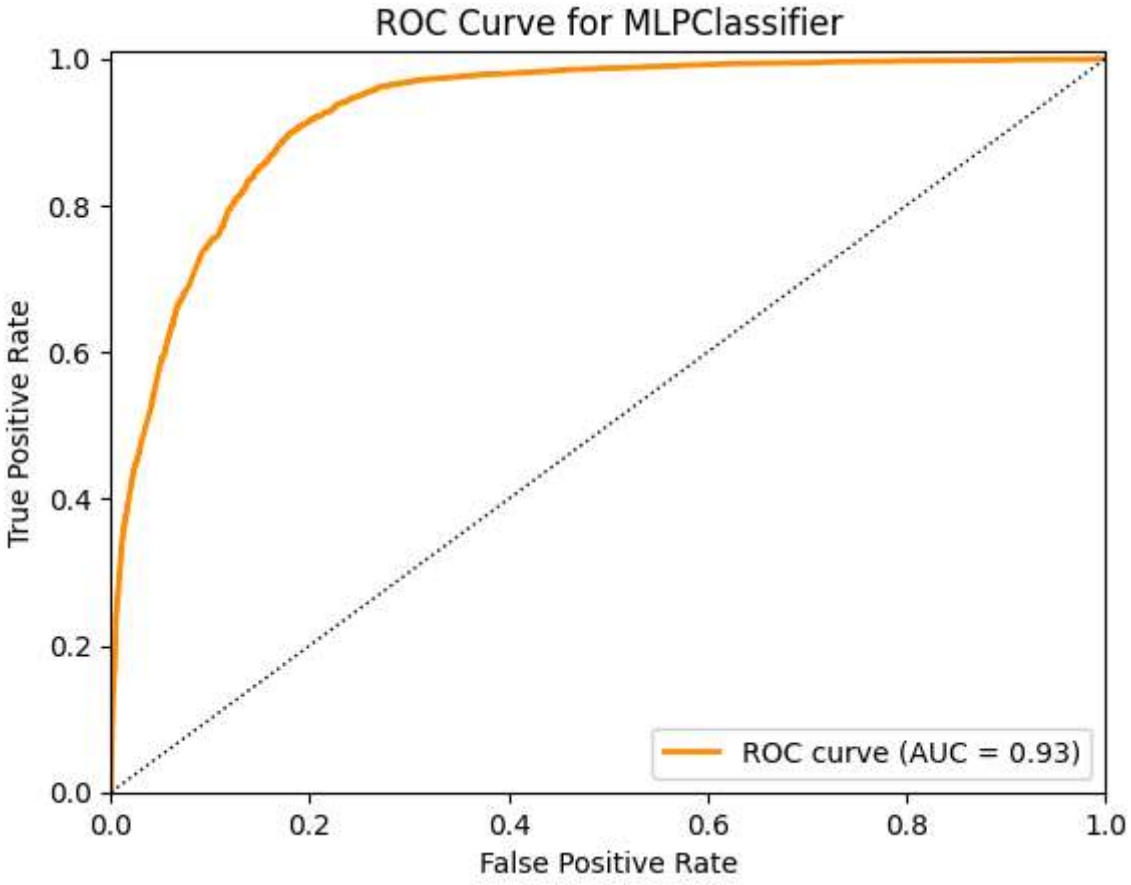
| Metric | Value |
|---|---|
| ROC AUC: | 0.857700 |
| Accuracy: | 0.857400 |
| Kappa: | 0.714900 |
| RMSE: | 0.142600 |
| MAE: | 0.142600 |
| R2: | 0.429500 |
| F1: | 0.864500 |
| | |
| Sensitivity: | 0.915300 |
| Specificity: | 0.800100 |

In [50]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[50]:

| Model: | MLPClassifier | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 9996 | 925 |
| **0** | 2208 | 8839 |

In [51]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




## 2.3.4. Classification and Regression Trees (CART)

### Model Training

In [52]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'criterion':'gini',                 # Criteria
        'random_state': seed                # Random seed for reproducibility
    }

    # Create and train model
    model = DecisionTreeClassifier(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'cart_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'cart_train.pkl')
```

```
# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: DecisionTreeClassifier
Parameters:
 ccp_alpha: 0.0
 class_weight: None
 criterion: gini
 max_depth: None
 max_features: None
 max_leaf_nodes: None
 min_impurity_decrease: 0.0
 min_samples_leaf: 1
 min_samples_split: 2
 min_weight_fraction_leaf: 0.0
 monotonic_cst: None
 random_state: 42
 splitter: best
```

## Model Main Metrics

In [53]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[53]:

**DecisionTreeClassifier**

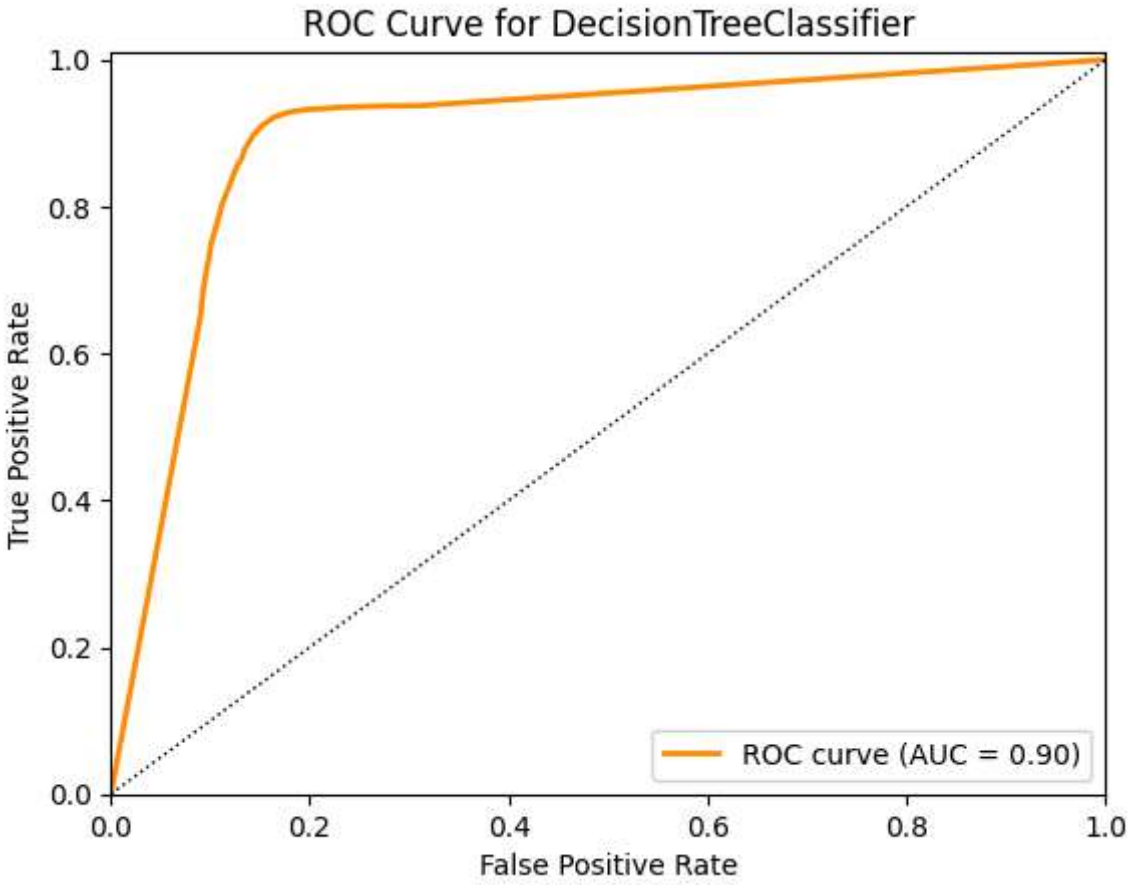
| Metric | Value |
|---|---|
| ROC AUC: | 0.879000 |
| Accuracy: | 0.878900 |
| Kappa: | 0.757800 |
| RMSE: | 0.121100 |
| MAE: | 0.121100 |
| R2: | 0.515500 |
| F1: | 0.881800 |
| | |
| Sensitivity: | 0.908800 |
| Specificity: | 0.849300 |

In [54]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[54]:

| **Model:** | **DecisionTreeClassifier** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 9925 | 996 |
| **0** | 1665 | 9382 |

In [55]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




## 2.3.5. Logistic Regression

### Model Training

In [56]:
```
if train_model_enabled :
    # Define model parameters
    params = {
        'random_state': seed                # Random seed for reproducibility
    }

    # Create and train model
    model = LogisticRegression(**params).fit(X_train, y_train)

    # Save to external file
    joblib.dump(model, models_folder + '/' + 'rl_train.pkl')

else:
    # Load model from external file
    model = joblib.load(models_folder + '/' + 'rl_train.pkl')

# Get model parameters and print as a one row list
print(f'Model name: {model.__class__.__name__} \nParameters:')
params = model.get_params()
for param_name, param_value in params.items():
    print(f" {param_name}: {param_value}")
```

```
Model name: LogisticRegression
Parameters:
 C: 1.0
 class_weight: None
 dual: False
 fit_intercept: True
 intercept_scaling: 1
 l1_ratio: None
 max_iter: 100
 multi_class: auto
 n_jobs: None
 penalty: l2
 random_state: 42
 solver: lbfgs
 tol: 0.0001
 verbose: 0
 warm_start: False
```

## Model Main Metrics

In [57]:
```
# Call function to show main metrics
model_metrics(model, X_test, y_test)
```

Out[57]:

**LogisticRegression**

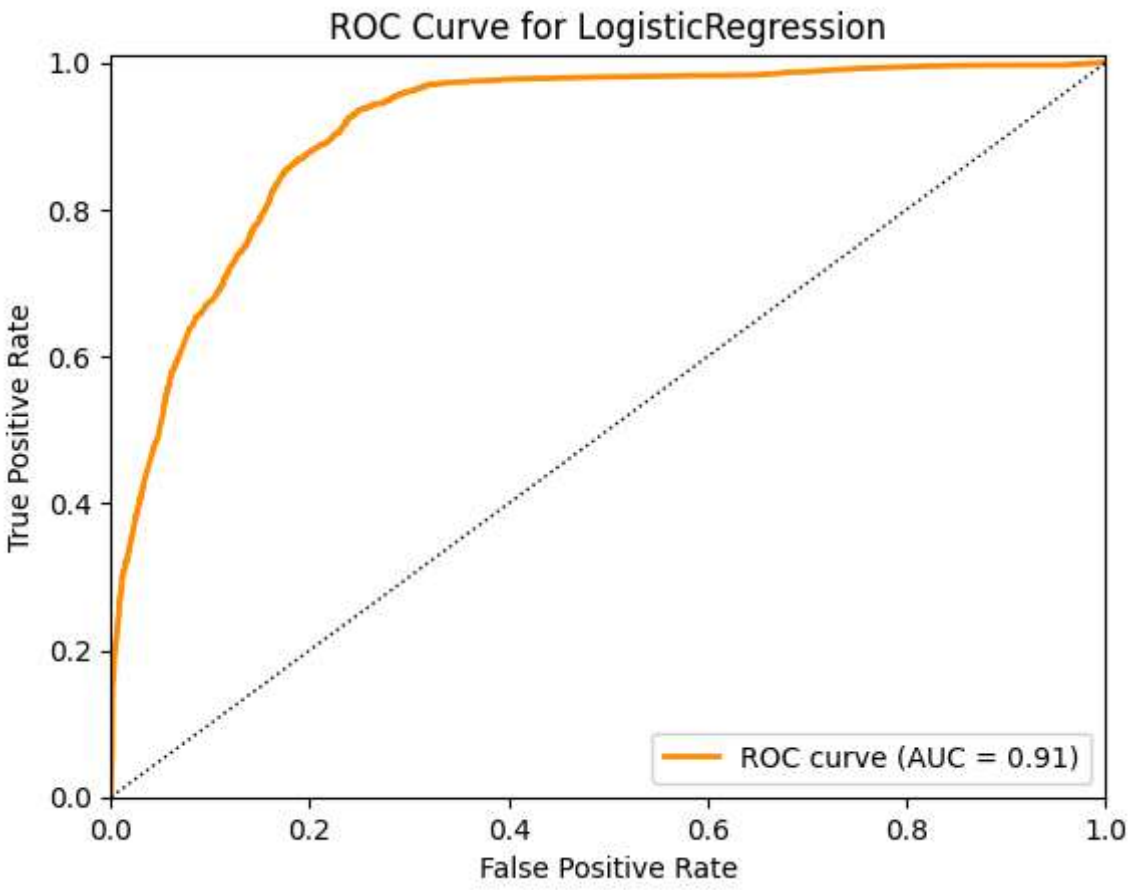
| Metric | Value |
|---|---|
| ROC AUC: | 0.842600 |
| Accuracy: | 0.842100 |
| Kappa: | 0.684500 |
| RMSE: | 0.157900 |
| MAE: | 0.157900 |
| R2: | 0.368300 |
| F1: | 0.854200 |
| | |
| Sensitivity: | 0.930100 |
| Specificity: | 0.755000 |

In [58]:
```
# Call function to show confusion matrix
confusion_matrix_table(model, X_test, y_test)
```

Out[58]:

| **Model:** | **LogisticRegression** | |
|---|---|---|
| **Predicted:** | **1** | **0** |
| **Actual Label:** | | |
| **1** | 10158 | 763 |
| **0** | 2706 | 8341 |

In [59]:
```
# Call function to plot Receiver Operating Characteristic (ROC) curve
plot_roc_curve(model, X_test, y_test)
```




# 3. Model Comparison

### Load saved models

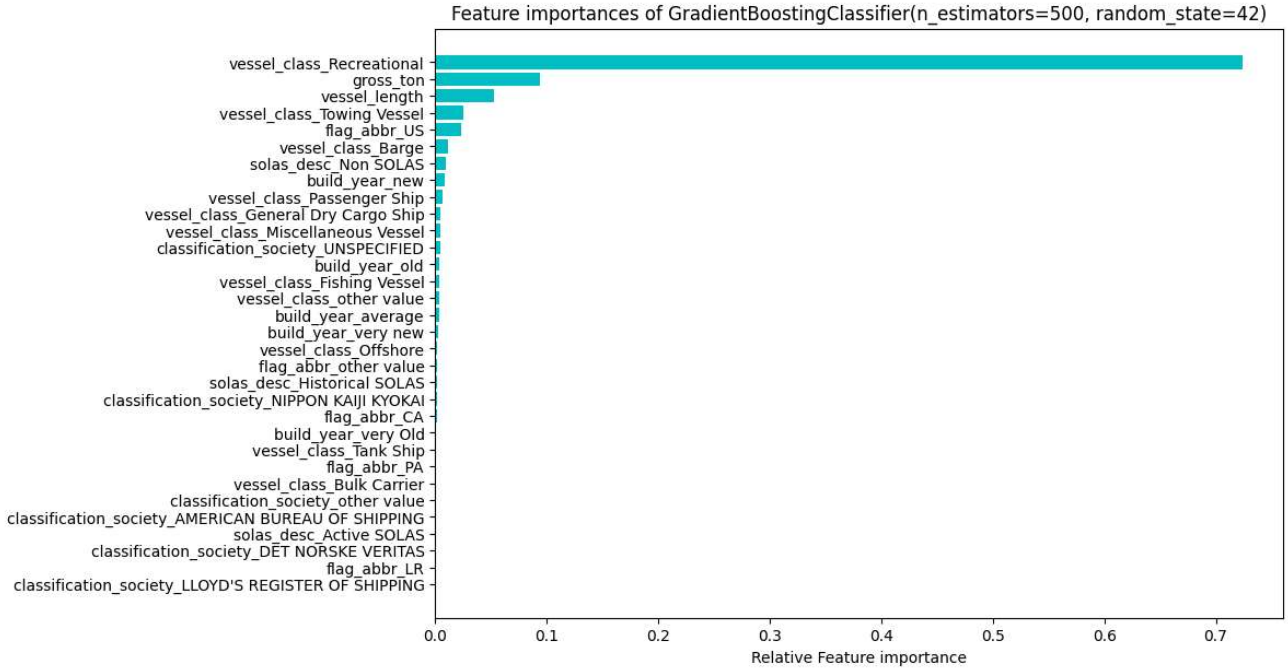
In [60]:
```
# All model list
models = ['nb_train', 'bnb_train', 'GBM_train',
          'XGB_train', 'rf_train', 'svm_train',
          'nnet_train', 'cart_train', 'rl_train']

# Load all models
for model in models:
    globals()[model] = joblib.load(models_folder + '/' + model + '.pkl')
    print(f'{model} loaded from {models_folder} folder')
```

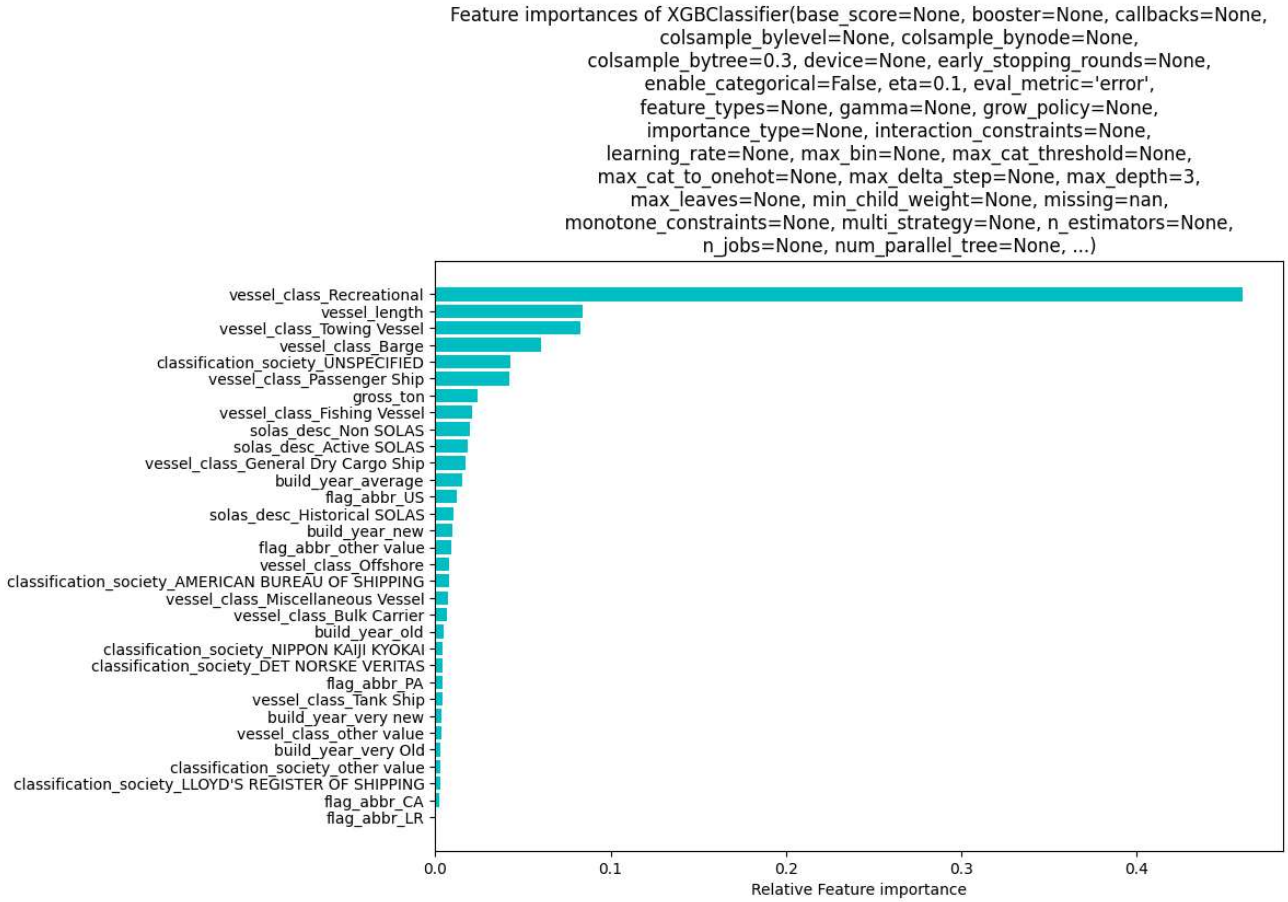
```
nb_train loaded from Models folder
bnb_train loaded from Models folder
GBM_train loaded from Models folder
XGB_train loaded from Models folder
rf_train loaded from Models folder
svm_train loaded from Models folder
nnet_train loaded from Models folder
cart_train loaded from Models folder
rl_train loaded from Models folder
```

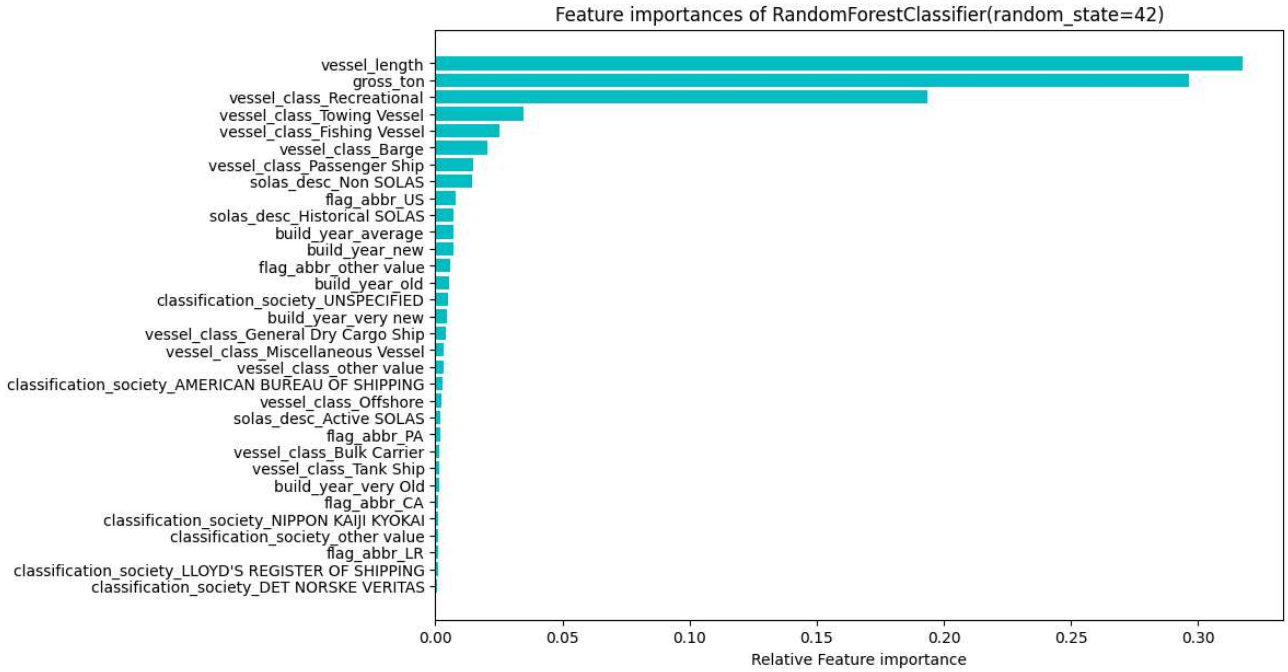
## 3.1. Feature importances

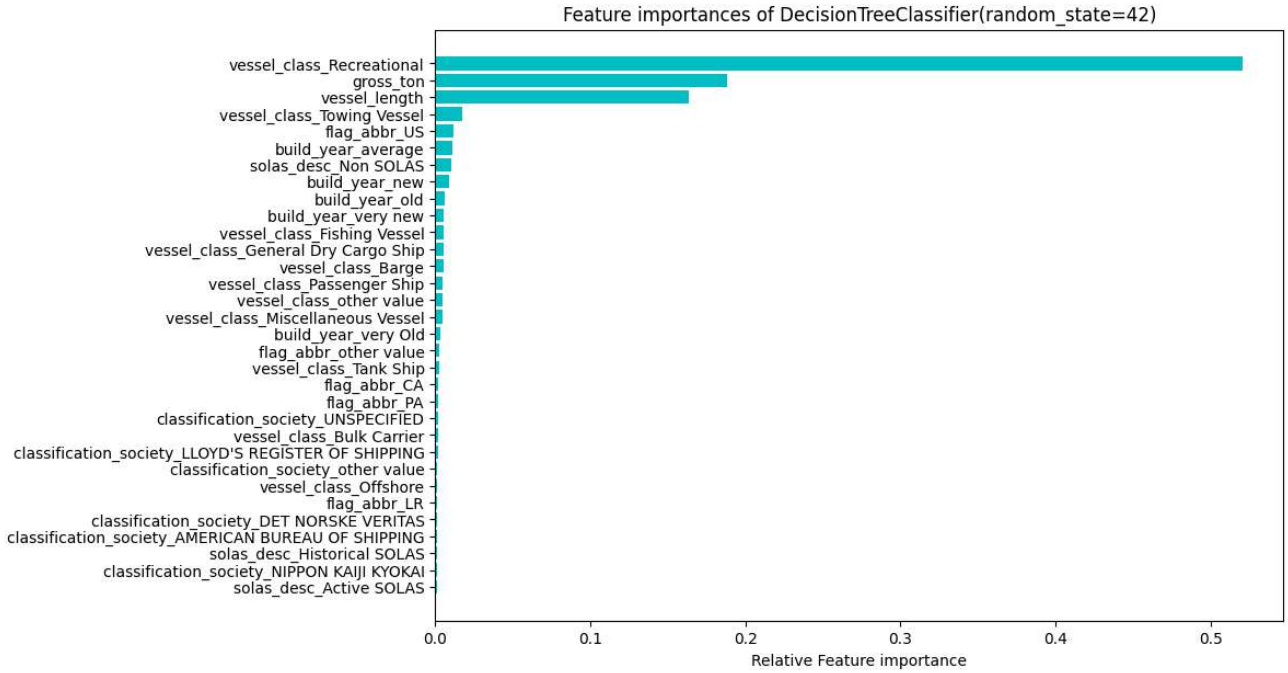
In [61]:
```
# Selected models list
selected_models = ['GBM_train', 'XGB_train', 'rf_train', 'cart_train']

for model in selected_models:
# Plot feature importances
    importances = pd.DataFrame({'value':eval(model).feature_importances_,
```

```
                                'variable_name':eval(model).feature_names_in_}).sort_val
    fig, ax = plt.subplots(figsize=(10, 7))
    plt.title(f"Feature importances of {eval(model)}")
    plt.barh(importances['variable_name'], importances['value'], color='#00bfc4', align=
    plt.xlabel('Relative Feature importance')
    plt.show()
```






# 3.2. Performance of the models

## Table of main metrics

In [62]:
```
# Calculate predictions for all models
y_preds = [eval(model).predict(X_test) for model in models]

# Sensitivity & Specificity
sensitivity = []
specificity = []

for i in range(len(models)):
    tn, fp, fn, tp = confusion_matrix(y_test, y_preds[i]).ravel()
    sensitivity.append(round(tp / (tp + fn), 4))
    specificity.append(round(tn / (tn + fp), 4))

# Dataframe for all metrics, sorted by ROC AUC
metrics_table = pd.DataFrame({
    'Model': [model.split('_')[0].upper() for model in models],
    'ROC AUC': [round(roc_auc_score(y_test, y_preds[i]), 4) for i in range(len(models))]
```

```
    'Accuracy': [round(accuracy_score(y_test, y_preds[i]), 4) for i in range(len(models)
    'Kappa': [round(cohen_kappa_score(y_test, y_preds[i]), 4) for i in range(len(models)
    'RMSE': [round(mean_squared_error(y_test, y_preds[i]), 4) for i in range(len(models)
    'MAE': [round(mean_absolute_error(y_test, y_preds[i]), 4) for i in range(len(models)
    'R2': [round(r2_score(y_test, y_preds[i]), 4) for i in range(len(models))],
    'F1': [round(f1_score(y_test, y_preds[i]), 4) for i in range(len(models))],
    'Sensitivity': sensitivity,
    'Specificity': specificity
}).sort_values(['ROC AUC'], ascending=[False]).style.hide()

# Show table
metrics_table
```

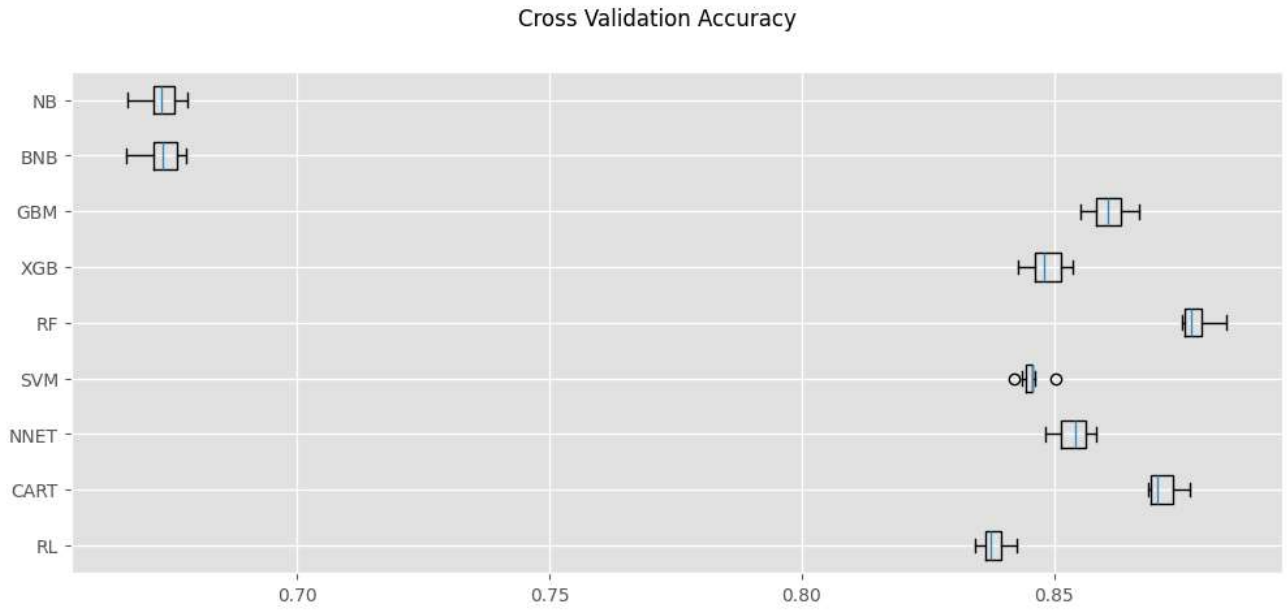
Out[62]:

| Model | ROC AUC | Accuracy | Kappa | RMSE | MAE | R2 | F1 | Sensitivity | Specif |
|---|---|---|---|---|---|---|---|---|---|
| RF | 0.883000 | 0.882800 | 0.765700 | 0.117200 | 0.117200 | 0.531100 | 0.886600 | 0.921300 | 0.84 |
| CART | 0.879000 | 0.878900 | 0.757800 | 0.121100 | 0.121100 | 0.515500 | 0.881800 | 0.908800 | 0.84 |
| GBM | 0.865100 | 0.864800 | 0.729900 | 0.135200 | 0.135200 | 0.459400 | 0.871000 | 0.917600 | 0.81 |
| NNET | 0.857700 | 0.857400 | 0.714900 | 0.142600 | 0.142600 | 0.429500 | 0.864500 | 0.915300 | 0.80 |
| XGB | 0.851500 | 0.851100 | 0.702400 | 0.148900 | 0.148900 | 0.404400 | 0.859700 | 0.917500 | 0.78 |
| SVM | 0.849200 | 0.848700 | 0.697700 | 0.151300 | 0.151300 | 0.394700 | 0.860900 | 0.941600 | 0.75 |
| RL | 0.842600 | 0.842100 | 0.684500 | 0.157900 | 0.157900 | 0.368300 | 0.854200 | 0.930100 | 0.75 |
| BNB | 0.673800 | 0.675100 | 0.348600 | 0.324900 | 0.324900 | -0.299600 | 0.579900 | 0.451000 | 0.89 |
| NB | 0.673700 | 0.674900 | 0.348200 | 0.325100 | 0.325100 | -0.300300 | 0.580000 | 0.451400 | 0.89 |

## Accuracy for Cross Validation (10 splits)

In [63]:
```
if file_export_enabled :
    # Calculate scores for accuracy for all models
    kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
    accuracy_scores = []
    for model in models:
        accuracy_scores.append(cross_val_score(eval(model), X_train, y_train, scoring='a

    # Store scores in a pandas dataframe and export to external file
    accuracy_scores = pd.DataFrame(accuracy_scores)
    accuracy_scores.columns = accuracy_scores.columns.astype(str)
    accuracy_scores.reset_index().to_feather(datasets_folder + '/' + 'accuracy_scores.fe

else:
    # Load this dataframe otherwise
    accuracy_scores = pd.read_feather(datasets_folder + '/' + 'accuracy_scores.feather')
```

In [64]:
```
# Boxplots of accuracy values in cross validation
plt.style.use('ggplot')
fig = plt.figure(figsize=(12,5))
fig.suptitle('Cross Validation Accuracy')
ax = fig.add_subplot(111)
plt.boxplot(accuracy_scores[::-1].T.iloc[1:].reset_index(drop=True), vert=False)
ax.set_yticklabels([model.split('_')[0].upper() for model in models][::-1])
plt.show()
```




## Statistical hypothesis test

In [65]:
```
# Convert score data to list
score_list = accuracy_scores.values.tolist()

# Initiate table
table=[]
for i in range(len(score_list)):
    table.append([])
for i in range(len(score_list)):
    for j in range(len(score_list)):
        table[i].append(0)

# Populate values
for i in range(len(score_list)):
    for j in range(i+1,len(score_list)):
        stat, p = ttest_rel(score_list[i], score_list[j])

        alpha = 0.05

        table[i][j]=np.round(p,3) # upper diagonal
        table[j][i]=np.round(stat,3) # lower diagonal
        table[i][i]=''

print('The following table shows the comparison of the statistics and the p-values of th
print('The value of the statistic on the lower diagonal and the p-value on the upper dia

# Show table, including model names
table = pd.DataFrame(table).set_index(pd.Index([model.split('_')[0].upper() for model in
table.columns = [model.split('_')[0].upper() for model in models]
table
```

```
The following table shows the comparison of the statistics and the p-values of the models
The value of the statistic on the lower diagonal and the p-value on the upper diagonal
```

Out[65]:

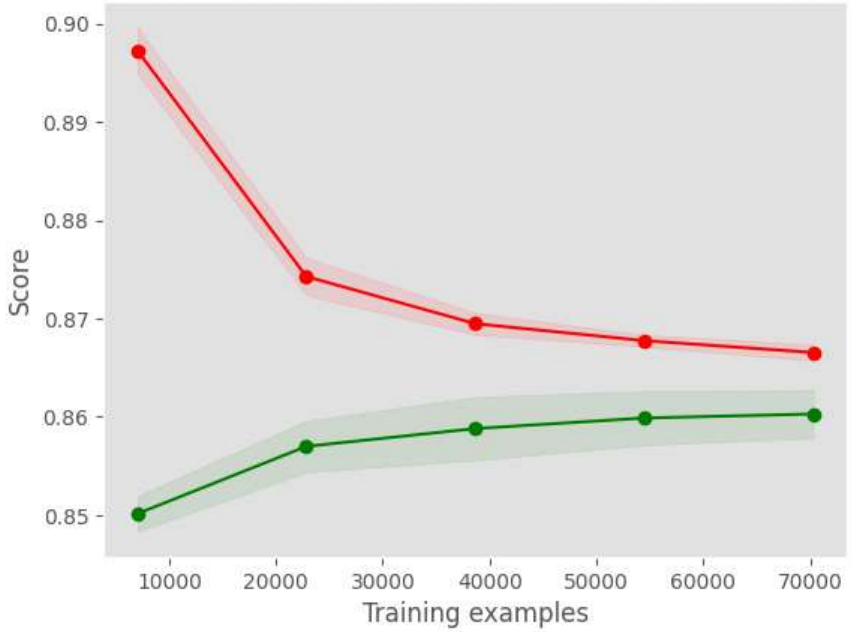
| | NB | BNB | GBM | XGB | RF | SVM | NNET | CART | RL |
|---|---|---|---|---|---|---|---|---|---|
| **NB** | | 0.34 | 0.058 | 0.123 | 0.142 | 0.194 | 0.21 | 0.216 | 0.246 |
| **BNB** | -1.001 | | 0.005 | 0.067 | 0.101 | 0.166 | 0.188 | 0.199 | 0.235 |
| **GBM** | -2.138 | -3.536 | | 0.407 | 0.298 | 0.367 | 0.35 | 0.33 | 0.361 |
| **XGB** | -1.682 | -2.055 | -0.865 | | 0.21 | 0.348 | 0.332 | 0.312 | 0.352 |
| **RF** | -1.594 | -1.807 | -1.097 | -1.339 | | 0.53 | 0.407 | 0.352 | 0.394 |
| **SVM** | -1.393 | -1.495 | -0.944 | -0.984 | -0.651 | | 0.301 | 0.279 | 0.354 |
| **NNET** | -1.341 | -1.412 | -0.98 | -1.019 | -0.866 | -1.09 | | 0.258 | 0.382 |
| **CART** | -1.321 | -1.376 | -1.024 | -1.064 | -0.975 | -1.145 | -1.2 | | 0.535 |
| **RL** | -1.231 | -1.265 | -0.958 | -0.977 | -0.89 | -0.972 | -0.914 | -0.642 | 0.000 |

## 3.3. Learning curves

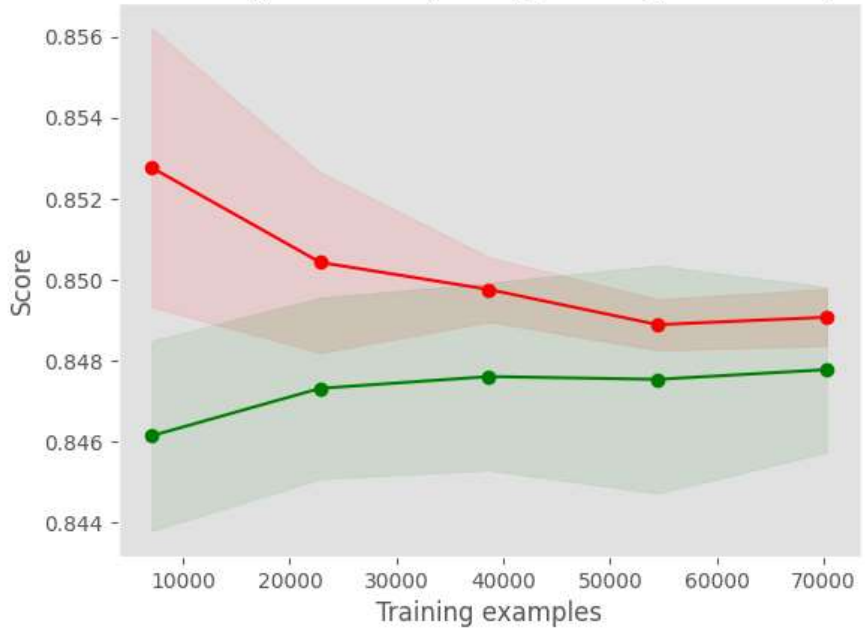
In [66]:

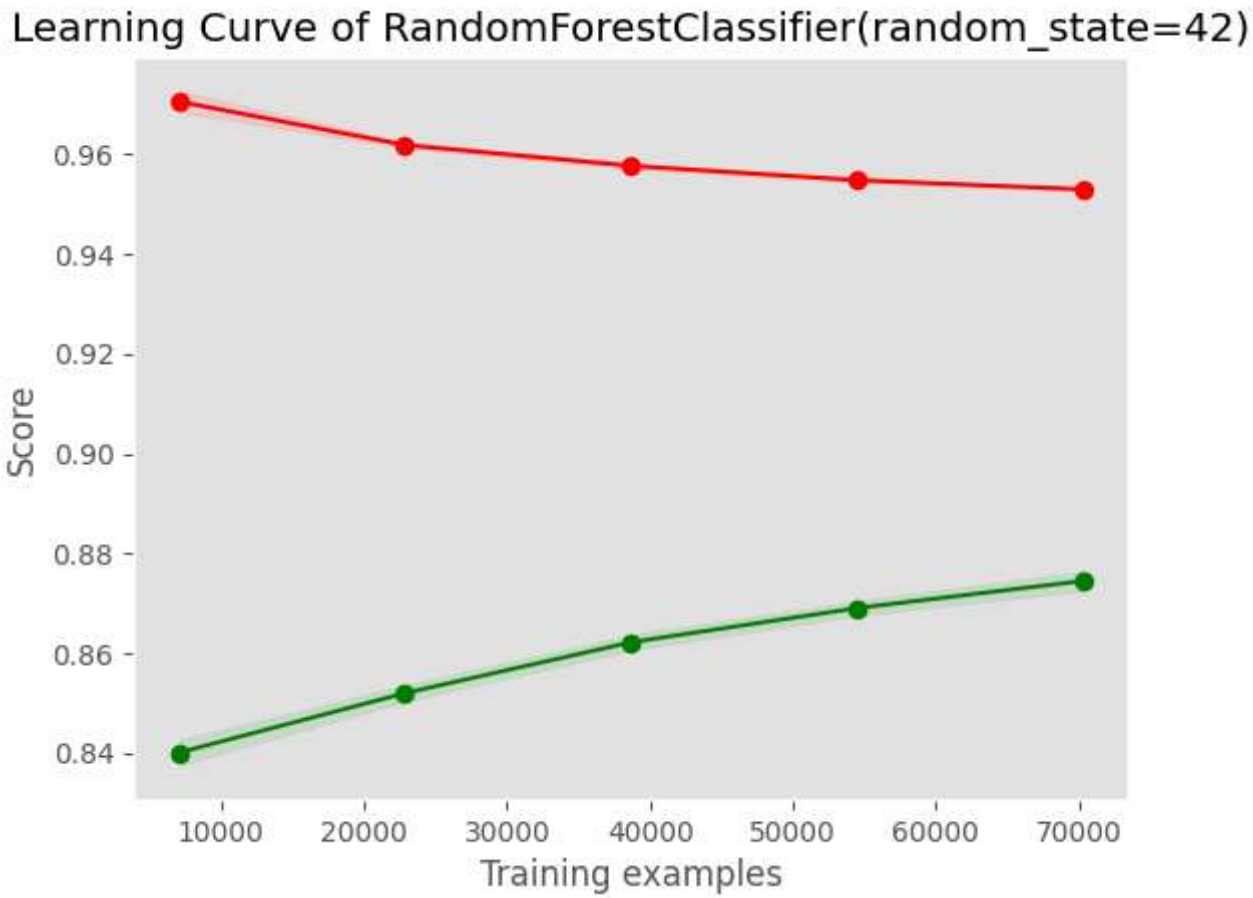
```
# Function to plot learning curves
def plot_learning_curve(model, X, y, cv, train_sizes=np.linspace(.1, 1.0, 5)):
    plt.figure()
    plt.title(f"Learning Curve of {model}")
    plt.xlabel("Training examples")
    plt.ylabel("Score")

    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=cv, train_sizes=train_sizes, n_jobs=n_jobs)
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1,color="r")
    plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

    return plt

# Set sample split
cv = ShuffleSplit(n_splits=10, test_size=0.2, random_state=seed)

for model in selected_models:
# Plot learning curves
    plot_learning_curve(eval(model), X_train, y_train, cv=cv)
```

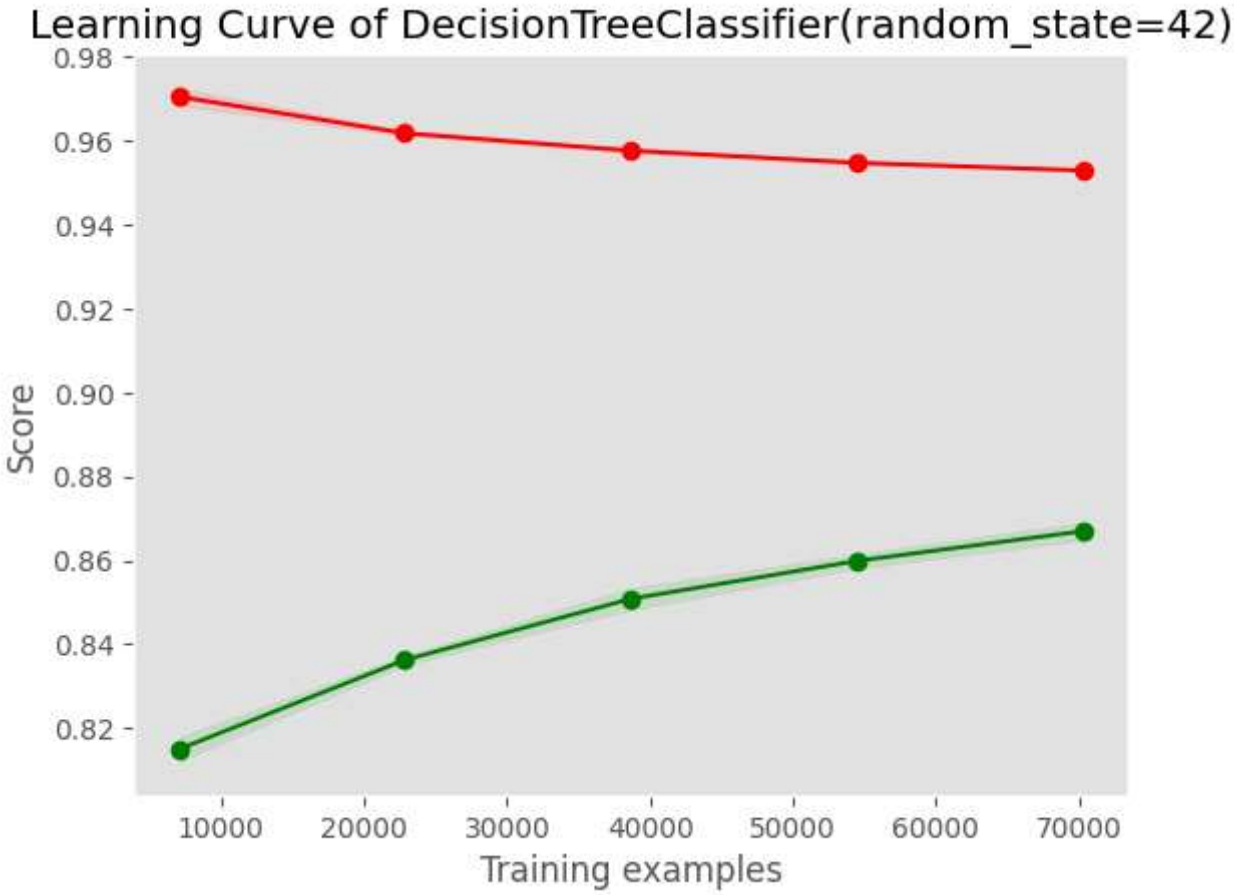
Learning Curve of GradientBoostingClassifier(n_estimators=500, random_state=42)




Learning Curve of XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=0.3, device=None, early_stopping_rounds=None, enable_categorical=False, eta=0.1, eval_metric='error', feature_types=None, gamma=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=None, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=3, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, multi_strategy=None, n_estimators=None, n_jobs=None, num_parallel_tree=None, ...)






# 4. Interpretability

In [67]:
```
# Create dalex explainers dict for selected models
explainers = {}
```

```
for model in selected_models:
    explainers[model] = dx.Explainer(eval(model), X_train, y_train, label=model.split('_
```

```
Preparation of a new explainer is initiated

  -> data              : 87868 rows 32 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 87868 values
  -> model_class       : sklearn.ensemble._gb.GradientBoostingClassifier (default)
  -> label             : GBM
  -> predict function  : <function yhat_proba_default at 0x0000021B3E76DC60> will be used
(default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 0.00103, mean = 0.501, max = 0.997
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.995, mean = 2.86e-06, max = 0.99
  -> model_info        : package sklearn

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 87868 rows 32 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 87868 values
  -> model_class       : xgboost.sklearn.XGBClassifier (default)
  -> label             : XGB
  -> predict function  : <function yhat_proba_default at 0x0000021B3E76DC60> will be used
(default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 0.0123, mean = 0.501, max = 0.988
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.986, mean = 2.31e-05, max = 0.986
  -> model_info        : package xgboost

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 87868 rows 32 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 87868 values
  -> model_class       : sklearn.ensemble._forest.RandomForestClassifier (default)
  -> label             : RF
  -> predict function  : <function yhat_proba_default at 0x0000021B3E76DC60> will be used
(default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 0.0, mean = 0.507, max = 1.0
  -> model type        : classification will be used (default)
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.987, mean = -0.00612, max = 0.993
  -> model_info        : package sklearn

A new explainer has been created!
Preparation of a new explainer is initiated

  -> data              : 87868 rows 32 cols
  -> target variable   : Parameter 'y' was a pandas.Series. Converted to a numpy.ndarray.
  -> target variable   : 87868 values
  -> model_class       : sklearn.tree._classes.DecisionTreeClassifier (default)
  -> label             : CART
  -> predict function  : <function yhat_proba_default at 0x0000021B3E76DC60> will be used
(default)
  -> predict function  : Accepts pandas.DataFrame and numpy.ndarray.
  -> predicted values  : min = 0.0, mean = 0.501, max = 1.0
  -> model type        : classification will be used (default)
```

```
  -> residual function : difference between y and yhat (default)
  -> residuals         : min = -0.985, mean = 3.84e-19, max = 0.992
  -> model_info        : package sklearn

A new explainer has been created!
```

## 4.1. Residual Diagnostics

In [68]:
```
# Plot Residual Diagnostics
for explainer in explainers.values():
    explainer.model_diagnostics().plot()
```


Residual Diagnostics
1
0.5
uals

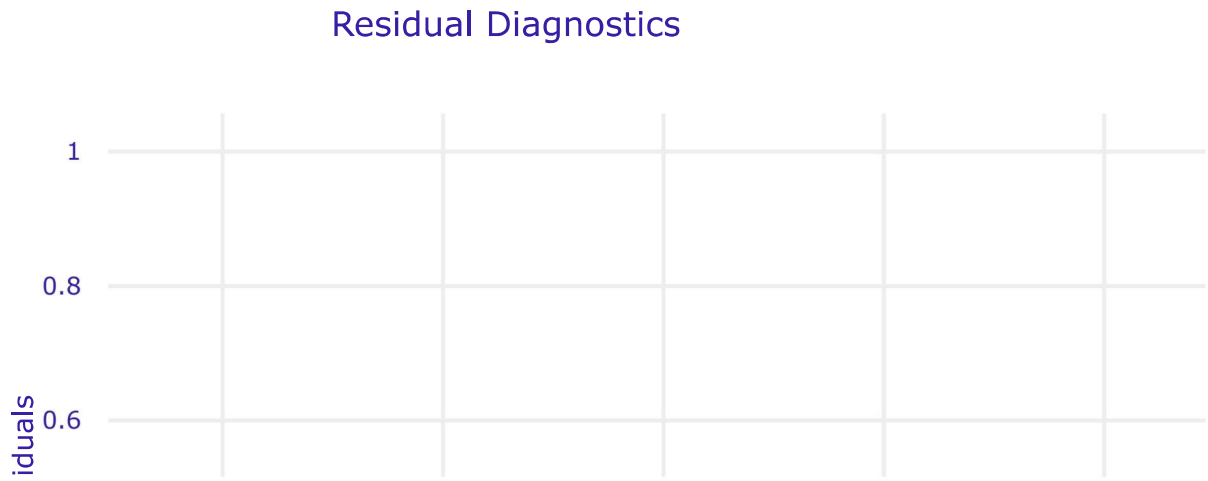

Residual Diagnostics
1
0.5
uals


In [69]:
```
# Plot Residual Diagnostics (Absolute)
for explainer in explainers.values():
    explainer.model_diagnostics().plot(variable = "ids", yvariable = "abs_residuals")
```


Residual Diagnostics
1
0.8
0.6
iduals

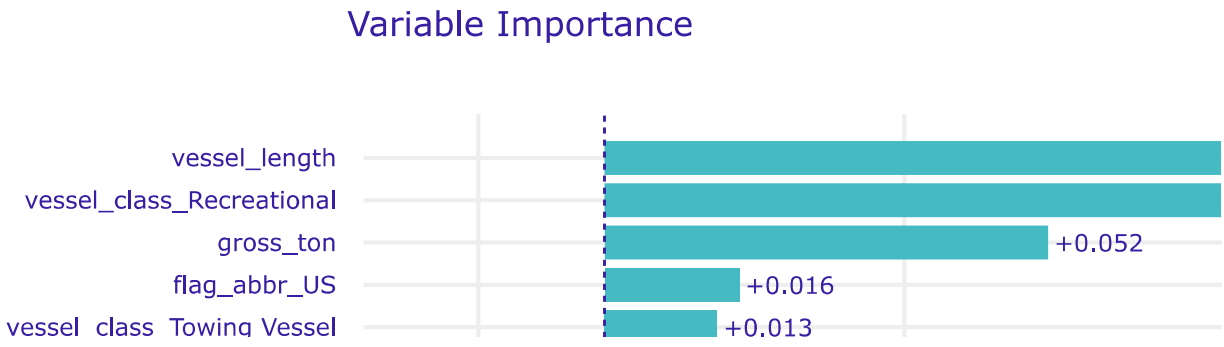

Residual Diagnostics
1
0.8
0.6
iduals

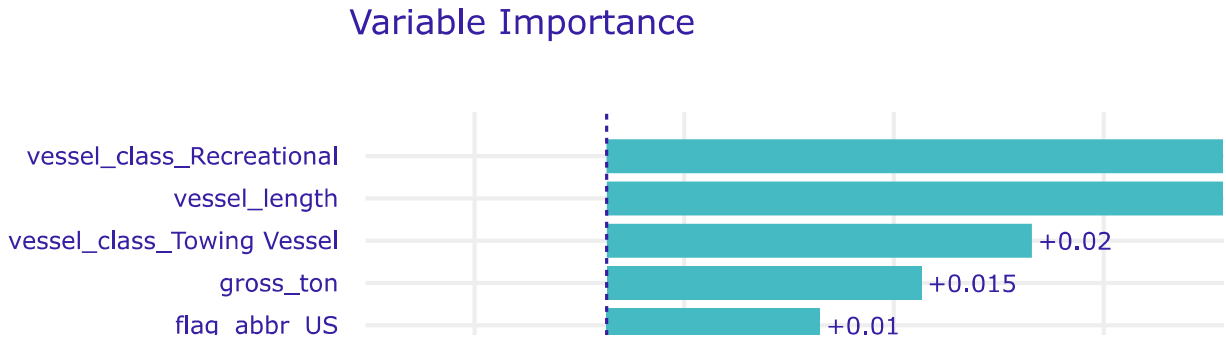

Residual Diagnostics
1
0.8
0.6
iduals

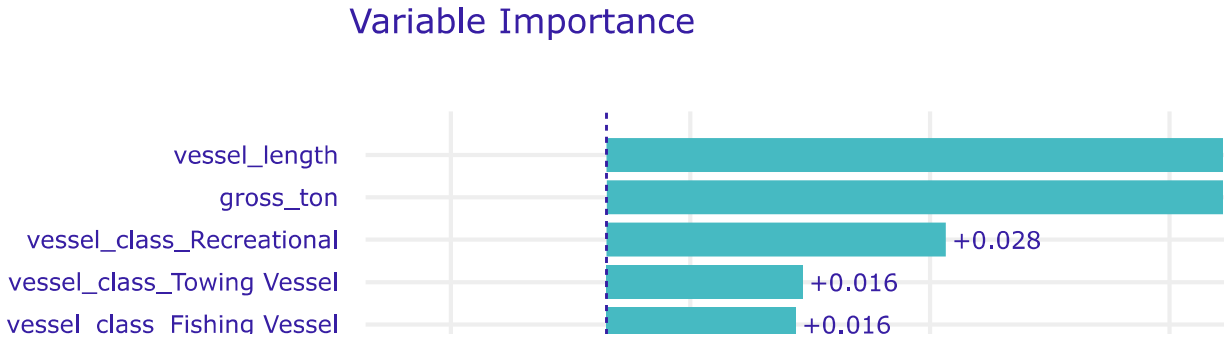


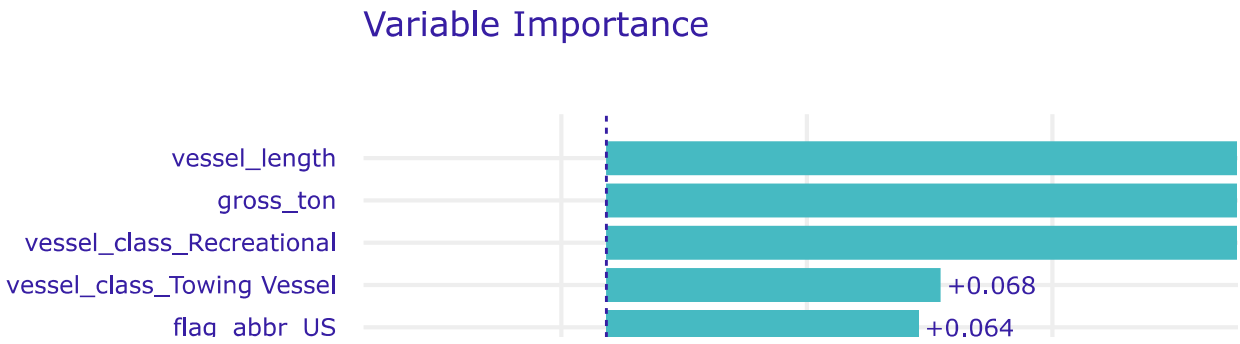

## 4.2. Global Explainability - Variable importances

In [70]:
```
# Plot variable importances for selected models
for explainer in explainers.values():
    explainer.model_parts().plot()
```



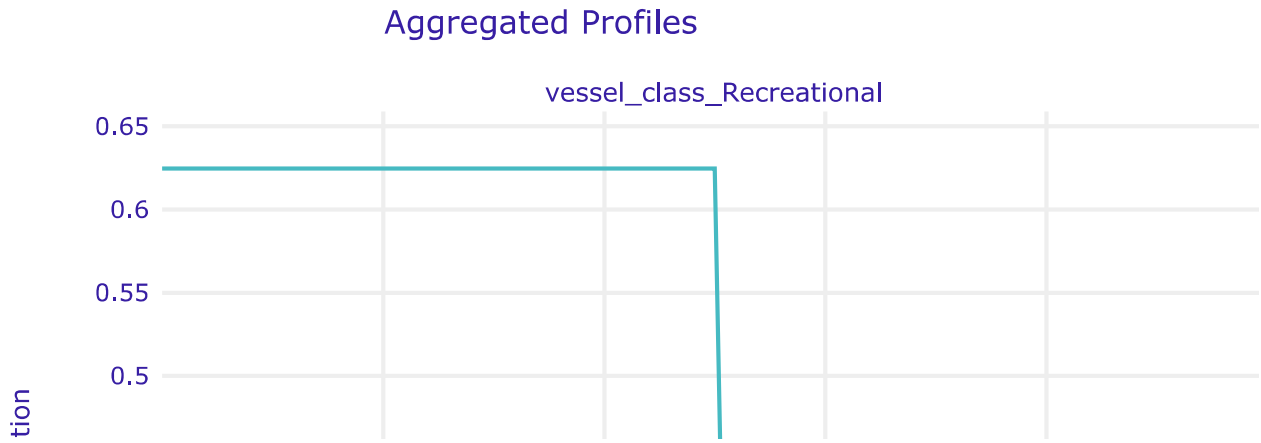

Variable Importance
vessel_class_Recreational
vessel_length
vessel_class_Towing Vessel
gross_ton
flag_abbr_US
+0.02
+0.015
+0.01

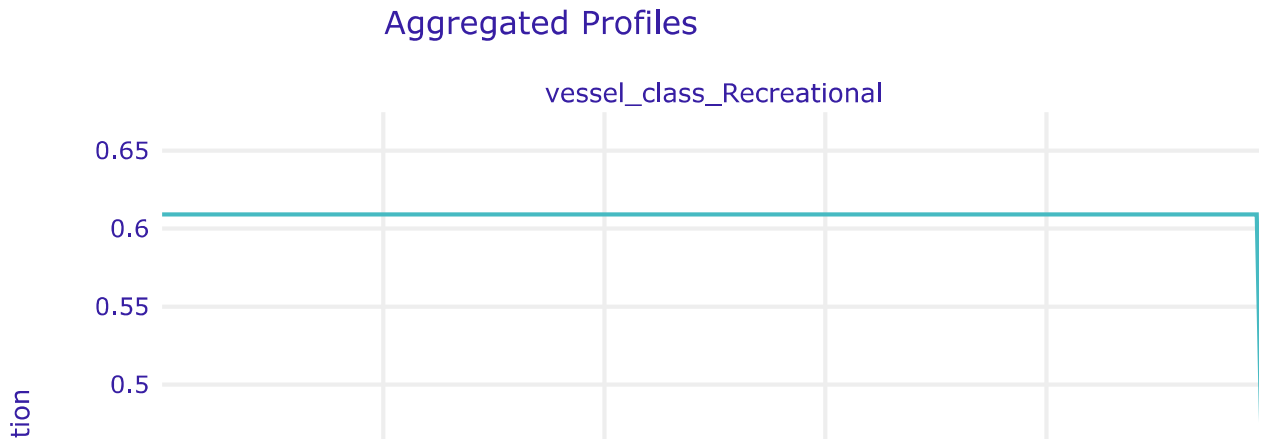

Variable Importance
vessel_length
gross_ton
vessel_class_Recreational
vessel_class_Towing Vessel
vessel_class_Fishing Vessel
+0.028
+0.016
+0.016

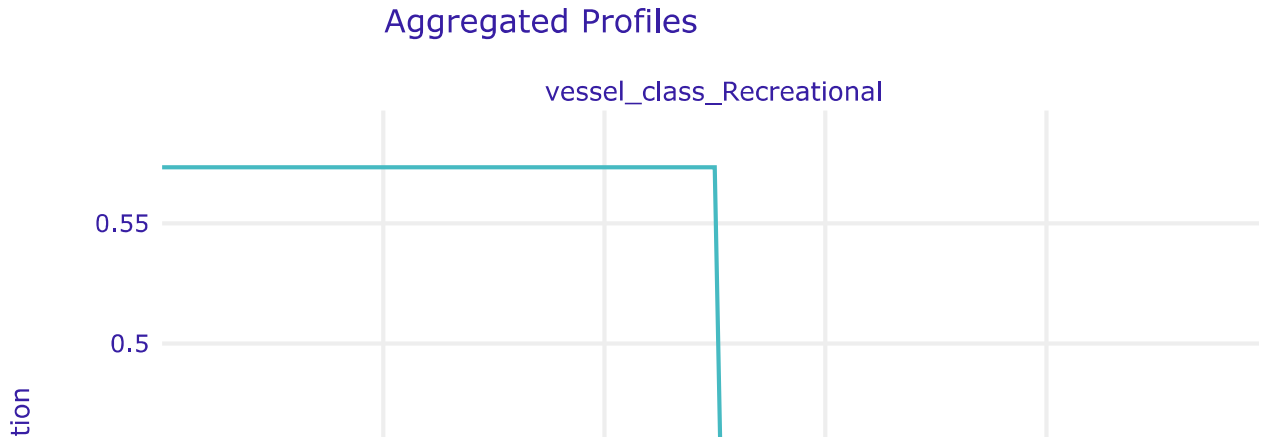


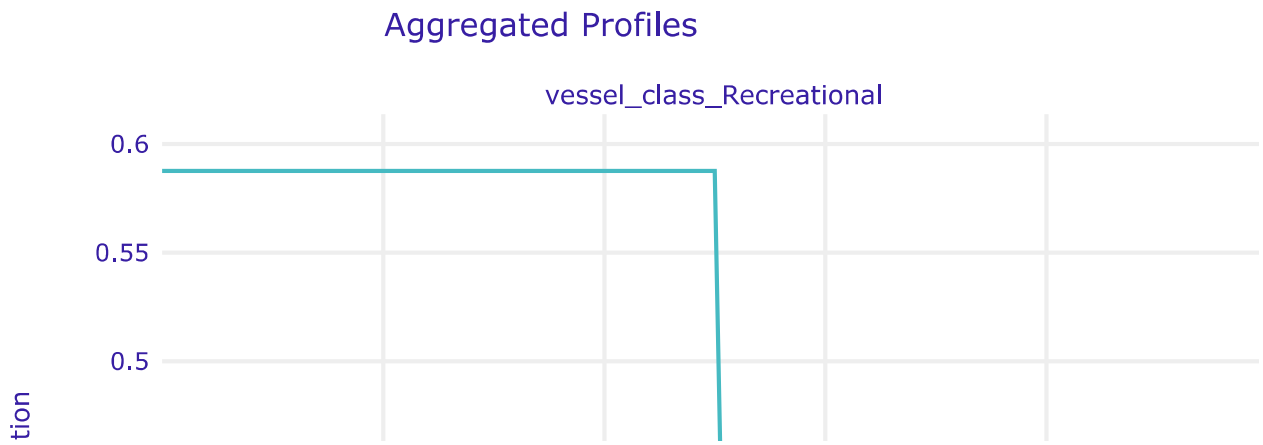

## 4.3. Partial Dependence Profile (PDP)

In [71]:
```
# Plot PDP for selected models
for explainer in explainers.values():
    explainer.model_profile(variables = ["vessel_class_Recreational", "vessel_class_Towi
```




Aggregated Profiles
vessel_class_Recreational
0.65
0.6
0.55
0.5
tion


Aggregated Profiles
vessel_class_Recreational
0.55
0.5
tion




## 4.4. Break Down profiles

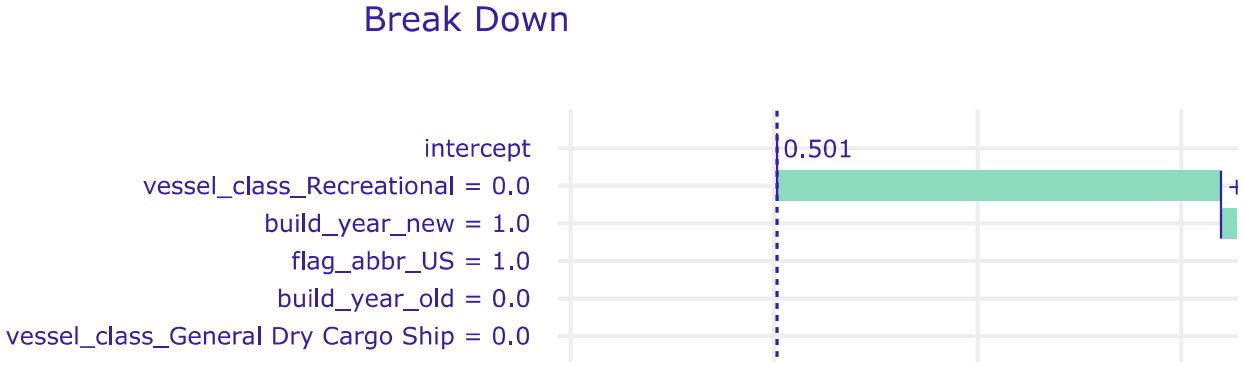
In [72]:
```
# Select a sample observation
vessel_sample = X_train.sample(n=1, random_state=seed)

# Show this observation
vessel_sample
```

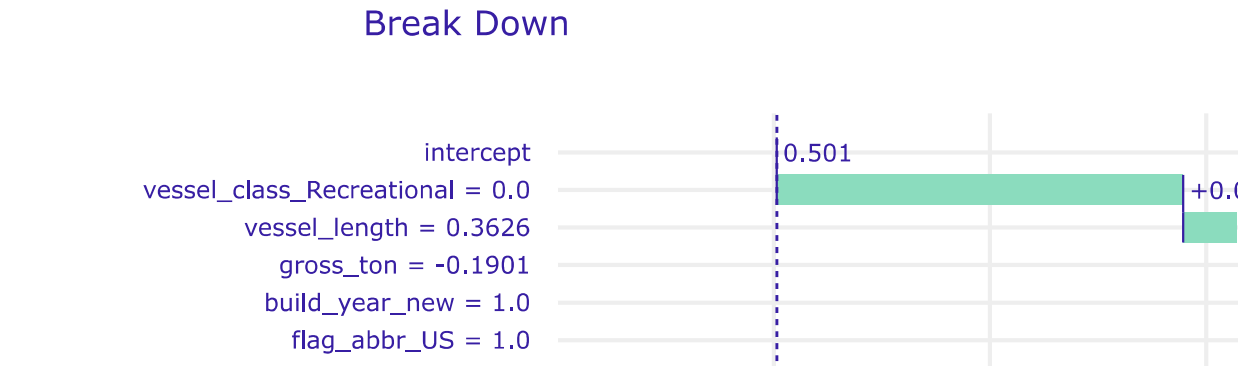
Out[72]:

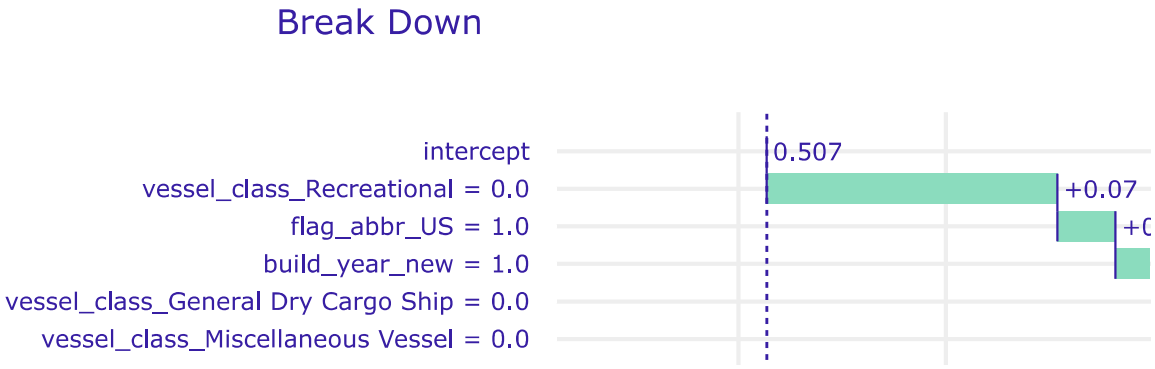
| | vessel_class_Barge | vessel_class_Bulk Carrier | vessel_class_Fishing Vessel | vessel_class_General Dry Cargo Ship | vessel_class_ |
|---|---|---|---|---|---|
| **88779** | 1 | 0 | 0 | 0 | |

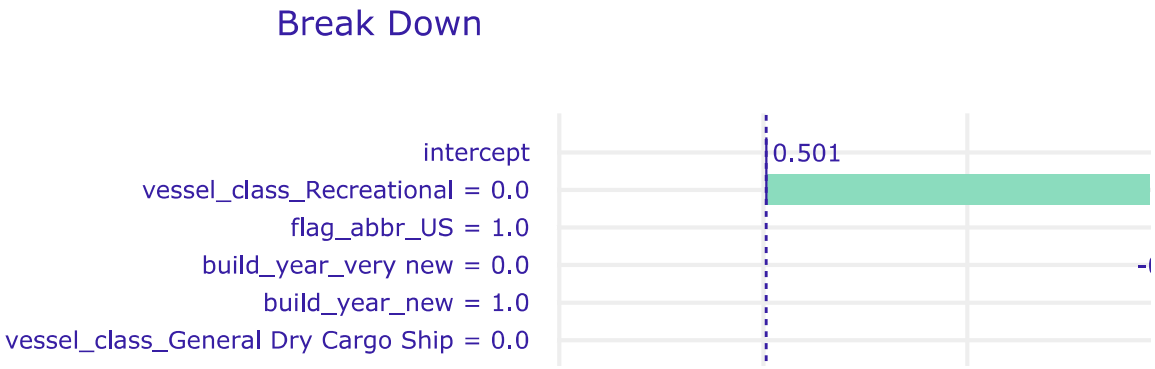
1 rows × 32 columns

In [73]:
```
# Plot shapley values for selected observation
for explainer in explainers.values():
    explainer.predict_parts(new_observation = vessel_sample, type = "break_down").plot()
```


Break Down
intercept
vessel_class_Recreational = 0.0
build_year_new = 1.0
flag_abbr_US = 1.0
build_year_old = 0.0
vessel_class_General Dry Cargo Ship = 0.0
0.501


Break Down
intercept
vessel_class_Recreational = 0.0
vessel_length = 0.3626
gross_ton = -0.1901
build_year_new = 1.0
flag_abbr_US = 1.0
0.501

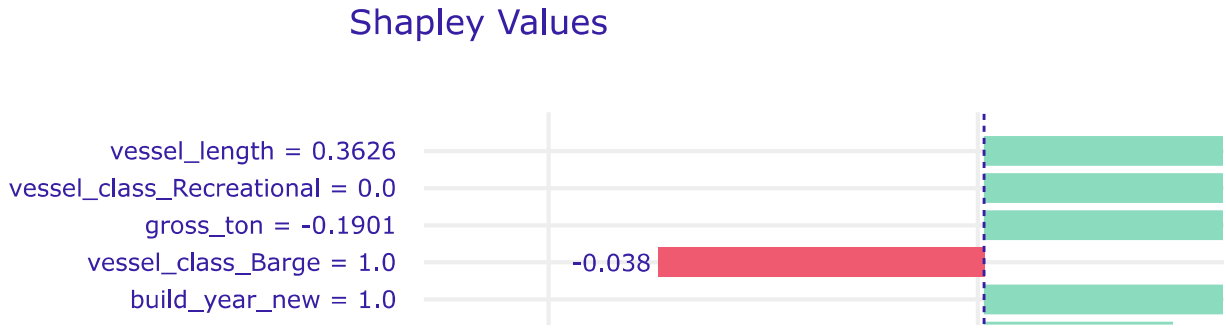


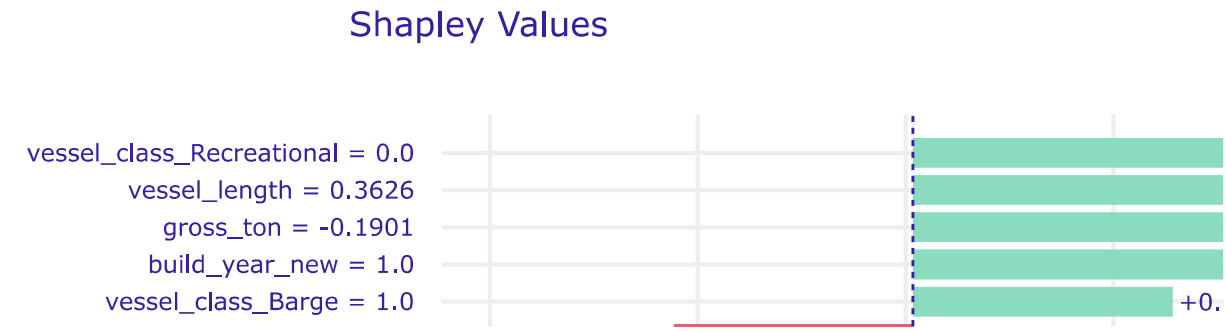



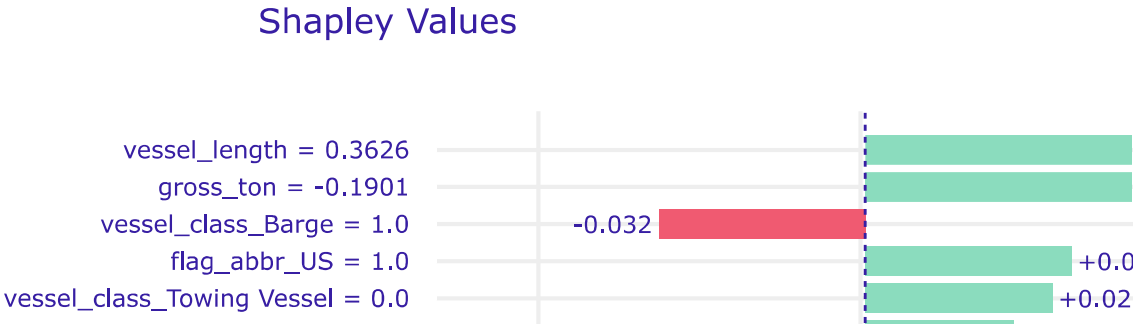

## 4.5. SHapley Additive exPlanations (SHAP)

In [74]:
```
# Calculate predictions (time-consuming process)
if train_model_enabled :
    shaps = {}
    for model in explainers:
        shaps[model] = explainers[model].predict_parts(new_observation = vessel_sample,
    joblib.dump(shaps, models_folder + '/' + 'shaps.pkl')
else:
    shaps = joblib.load(models_folder + '/' + 'shaps.pkl')

# Plot shapley values
```

```
for model in explainers:
    shaps[model].plot()
```

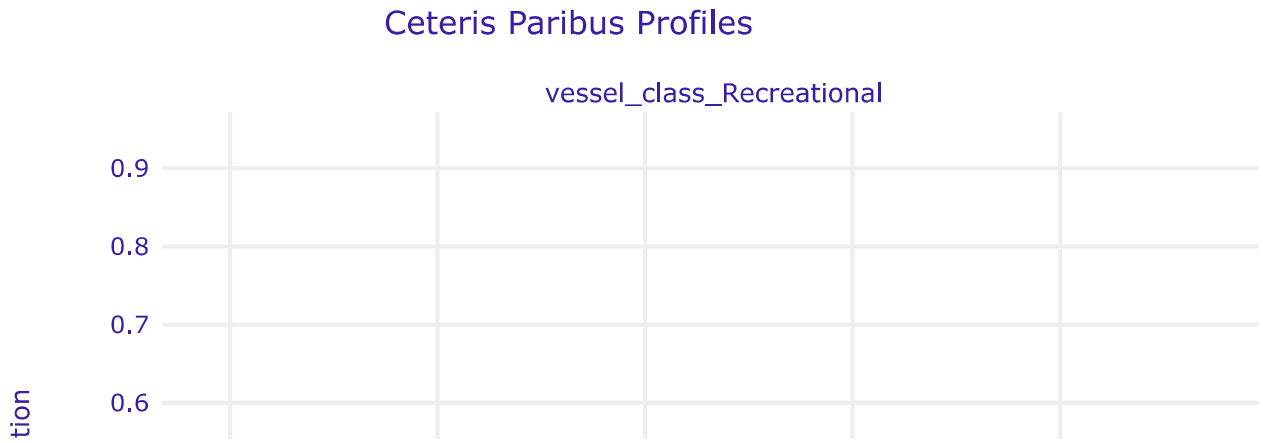


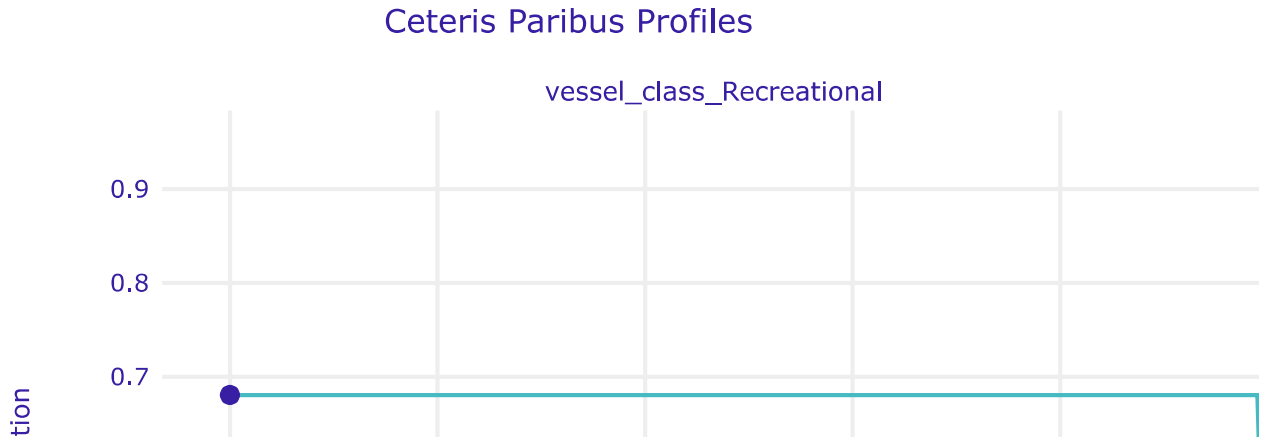



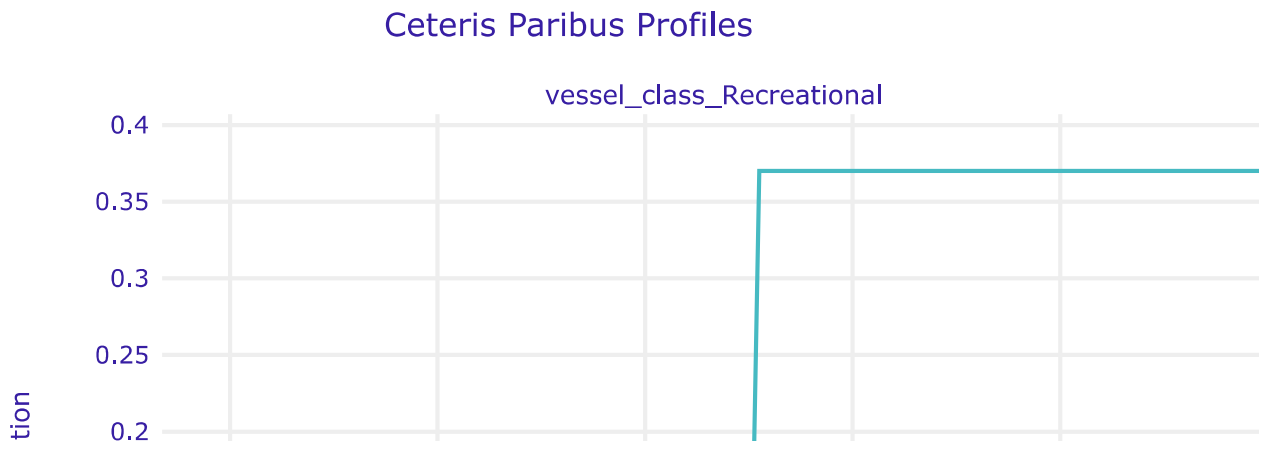



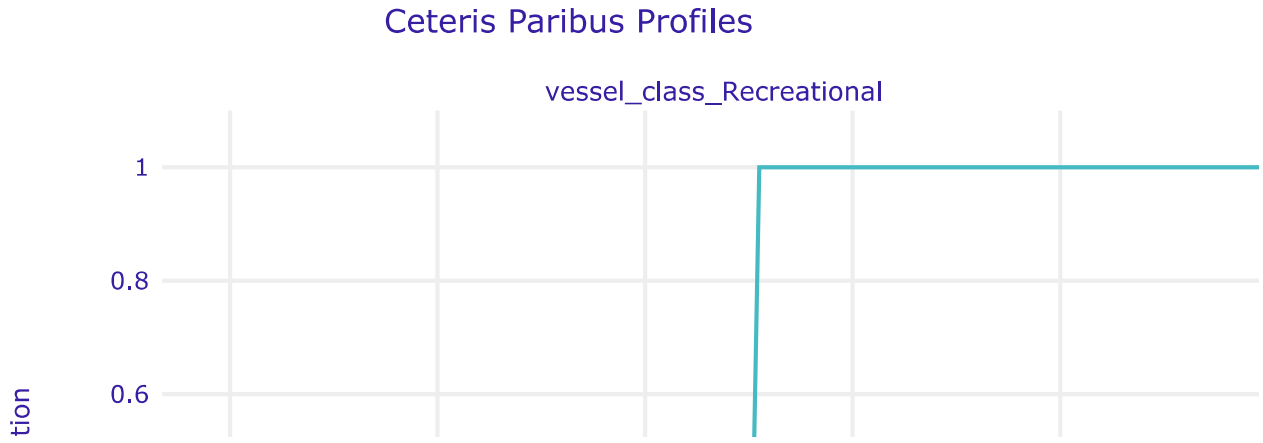

## 4.6. Ceteris Paribus profiles

In [75]:
```
# Select a sample observation
vessel_sample = X_train.sample()

# Plot shapley values for selected observation
for explainer in explainers.values():
    explainer.predict_profile(new_observation = vessel_sample).plot(variables = ["vessel
```


Ceteris Paribus Profiles
vessel_class_Recreational
0.9
0.8
0.7
0.6
tion


Ceteris Paribus Profiles
vessel_class_Recreational
0.9
0.8
0.7
tion




# Session info

In [76]:
```python
# Python version
import sys
print(f'Python version: {sys.version}')

# Initiate variables for package versions
pkgs_dict = {}
version = []

# Loop importing __version__ for each package (because no whole packages are imported)
for pkg in ['pandas', 'numpy', 'sklearn', 'dalex']:
    exec(f"from {pkg} import __version__ as version")
    pkgs_dict[pkg] = version

# Show as table
pd.DataFrame(pkgs_dict, index=["Package version"])
```

```
Python version: 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC
v.1916 64 bit (AMD64)]
```

Out[76]:

| | pandas | numpy | sklearn | dalex |
|---|---|---|---|---|
| **Package version** | 2.2.2 | 1.25.2 | 1.4.2 | 1.7.0 |