StratThink_HDipData_CA_2023

Students:

Otáwio 2022200

Ruben,

Barbara,

Katia,

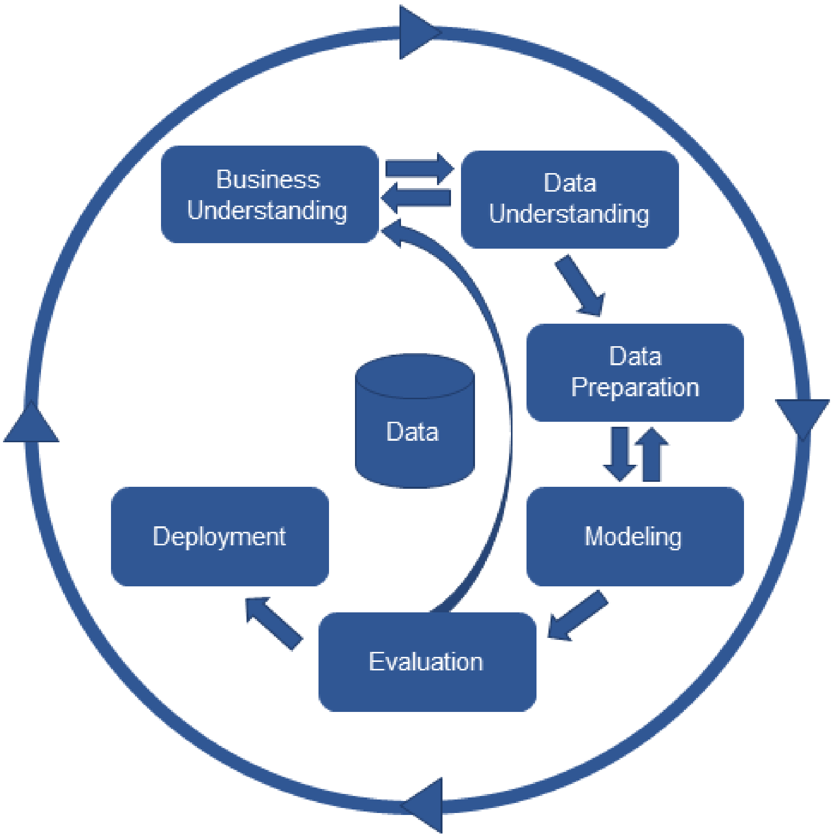
Eliene.

Lecturer: James Garza

Module: Strategic Thinking

Business understanding

The implementation of data preparation and machine learning (ML) techniques will be the way to determine which of these objects are hazardous and which are not. The knowledge discovery process (KDD - Knowledge Discovery in Databases) was created using the CRISP-DM (CRoss Industry Standard Process for Data Mining) methodology (Chapman et al., 2000). Thus, domain understanding, data understanding, data preparation, modeling, evaluation, and distribution are the six phases that make up a project's life cycle.



NASA - Nearest Earth Objects

https://jpl.nasa.gov, n.d

Libraries

In [1]:
```
# Import warnings.
import warnings
warnings.filterwarnings('ignore')
```

```
# Import the numpy and pandas package.
import numpy as np
import pandas as pd

# Import sklearn preprocessing.
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustSca

# Import imblearn SMOTEENN.
from imblearn.combine import SMOTEENN
from collections import Counter

# Import train_test_split function.
from sklearn.model_selection import train_test_split

# Import time.
import time

# Import math.
import math

# Scipy package.
from scipy import sparse

# Import cross-validation libraries.
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict

# Import sklearn model selection.
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# Import machine learning Classifier.
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier

# Import sklearn.metrics.
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, classification_report, roc_
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import cohen_kappa_score

# Import the mglearn package.
import mglearn

# Import data visualisation package.
from yellowbrick.classifier import ClassificationReport
from matplotlib import pyplot
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Setting data visualisation style.
sns.set_palette("bright")
sns.set_style('whitegrid')
sns.color_palette("bright")
```

Out[1]:

Libraries (description and usefulness of the main libraries used in the project)

Python libraries	Libraries Description	Use of the library in the project
Pandas	Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. (Pandas, 2018)	Reading and manipulating data, making use of EDA, and other functions.
Numpy	NumPy (Numerical Python) is an open source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python, and it's at the core of the scientific Python and PyData ecosystems. (Numpy, n.d.)	Numerical manipulation, numerical functions, converting data into an array and others.
Scikit-learn	Simple and efficient tools for predictive data analysis .Accessible to everybody, and reusable in various contexts, built on NumPy, SciPy, and matplotlib and open source. (scikit-learn, 2019)	Used for data preparation, cross-validation of sets, training and testing in the split part of the dataset, importing machine learning models for classification, and selecting the best model parameters.
XGBoost	XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. (xgboost.readthedocs.io, n.d.)	Used to import XGBoost model for classification.
Scipy	SciPy provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems. (SciPy, 2020)	Apply to check the sparsity of the data.
Matplotlib	Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible. (Matplotlib, 2012)	It was used for visualization throughout the project's progress.
Seaborn	Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics. (seaborn, 2012)	It was used for visualization throughout the project's progress.
imblearn	Imbalanced-learn (imported as imblearn) is an open source, MIT-licensed library relying on scikit-learn (imported as sklearn) and provides tools when dealing with classification with imbalanced classes. (Imbalanced-learn.org, 2021)	It was used for data imbalance adjustment.
Collections	This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, dict, list, set, and tuple. (docs.python.org, n.d.)	It was used to apply element counts before and after balancing the data.
Time	This module provides various time-related functions. For related functionality, see also the datetime and calendar modules. (Python.org, 2000)	Timed the duration of each classification model's application on the training data during the evaluation mode.
Warnings	The warnings module was introduced in PEP 230 as a way to warn programmers about changes in language or library features in anticipation of backwards incompatible changes coming with Python 3.0. Since warnings are not fatal, a program may encounter the same warn-able situation many times in the course of running. (pymotw.com, n.d.)	Used to avoid unnecessary warnings

Data Dictionary

VARIABLE	DATA TYPE	DESCRIPTION	UNIT
id	int64	Unique Identifier for each Asteroid	-
name	object	Name given by NASA	-
est_diameter_min	float64	Minimum Estimated Diameter in Kilometres	km
est_diameter_max	float64	Maximum Estimated Diameter in Kilometres	km
relative_velocity	float64	Velocity Relative to Earth	Meter per second (m/s).
miss_distance	float64	Distance in Kilometres missed	km
orbiting_body	object	Planet that the asteroid orbits	-
sentry_object	bool	Included in sentry - an automated collision monitoring system	-
absolute_magnitude	float64	Describes intrinsic luminosity	Joules per second, or watts
hazardous	bool	Boolean feature that shows whether asteroid is harmful or not	Astronomical units

https://jpl.nasa.gov (n.d.)

Data Understanding

Exploratory Data Analysis (EDA)

In [2]:
```
# Checking missing value formats and read data file.
missing_value_formats = ["na", "n.a.", "?", "NA", "n/a", "--"]
df = pd.read_csv("neo.csv", na_values=missing_value_formats)

# Checking for any missing values.
if df.isna().sum().sum() == 0:
    print("No missing values.")
else:
    print("There are missing values in the data.")

# Display first and last 5 records.
display(df.head())
display(df.tail())

# Display shape and info.
print("Shape of the dataset:", df.shape)
print("\nInformation about the dataset:\n")
df.info()
```

```
No missing values.
```

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_distan
0	2162635	162635 (2000 SS164)	1.198271	2.679415	13569.249224	5.483974e+
1	2277475	277475 (2005 WK4)	0.265800	0.594347	73588.726663	6.143813e+
2	2512244	512244 (2015 YE18)	0.722030	1.614507	114258.692129	4.979872e+
3	3596030	(2012 BV13)	0.096506	0.215794	24764.303138	2.543497e+
4	3667127	(2014 GE35)	0.255009	0.570217	42737.733765	4.627557e+

	id	name	est_diameter_min	est_diameter_max	relative_velocity	miss_d
90831	3763337	(2016 VX1)	0.026580	0.059435	52078.886692	1.230(
90832	3837603	(2019 AD3)	0.016771	0.037501	46114.605073	5.432
90833	54017201	(2020 JP3)	0.031956	0.071456	7566.807732	2.840(
90834	54115824	(2021 CN5)	0.007321	0.016370	69199.154484	6.8692
90835	54205447	(2021 TW7)	0.039862	0.089133	27024.455553	5.977:

```
Shape of the dataset: (90836, 10)

Information about the dataset:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 90836 entries, 0 to 90835
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   id                  90836 non-null  int64  
 1   name                90836 non-null  object 
 2   est_diameter_min    90836 non-null  float64
 3   est_diameter_max    90836 non-null  float64
 4   relative_velocity   90836 non-null  float64
 5   miss_distance       90836 non-null  float64
 6   orbiting_body       90836 non-null  object 
 7   sentry_object       90836 non-null  bool   
 8   absolute_magnitude  90836 non-null  float64
 9   hazardous           90836 non-null  bool   
dtypes: bool(2), float64(5), int64(1), object(2)
memory usage: 5.7+ MB
```

In [3]:
```
# Checking if dataset has duplicate rows.
print("Duplicated rows before: ", df.duplicated().sum())
df = df.drop_duplicates()
print("Duplicated rows after: ", df.duplicated().sum())
```

```
Duplicated rows before:  0
Duplicated rows after:  0
```

In [4]:
```
# Printing columns to copy and paste
print(df.columns)
```

```
Index(['id', 'name', 'est_diameter_min', 'est_diameter_max',
       'relative_velocity', 'miss_distance', 'orbiting_body', 'sentry_ob
ject',
       'absolute_magnitude', 'hazardous'],
      dtype='object')
```

In [5]:
```
# Rename the columns.
df = df.rename(columns={
    'id': 'Asteroid ID',
    'name': 'Asteroid Name',
    'est_diameter_min': 'Estimated Diameter (Min)',
    'est_diameter_max': 'Estimated Diameter (Max)',
    'relative_velocity': 'Relative Velocity (km/s)',
    'miss_distance': 'Miss Distance (km)',
    'orbiting_body': 'Orbiting Body',
    'sentry_object': 'Sentry Object',
    'absolute_magnitude': 'Absolute Magnitude',
    'hazardous': 'Potentially Hazardous'
})
print(df.columns)
```

```
Index(['Asteroid ID', 'Asteroid Name', 'Estimated Diameter (Min)',
       'Estimated Diameter (Max)', 'Relative Velocity (km/s)',
       'Miss Distance (km)', 'Orbiting Body', 'Sentry Object',
       'Absolute Magnitude', 'Potentially Hazardous'],
      dtype='object')
```

Checking for missing values

The info( ) function had already shown that the data set had no missing values, but the isnull( ) function can be used as a backup to make sure there weren't any.

In [6]:
```
# Check for null values and calculate the percentage of null values.
null_df = df.isnull().sum().sort_values().apply(lambda x: '{:.2%}'.format
           .to_frame(name='Null Percentage').rename_axis('Column Name').r

# Print the null value information.
print("Null values per column:")
print(null_df)
```

```
Null values per column:
                Column Name Null Percentage
0               Asteroid ID           0.00%
1             Asteroid Name           0.00%
2  Estimated Diameter (Min)           0.00%
3  Estimated Diameter (Max)           0.00%
4  Relative Velocity (km/s)           0.00%
5         Miss Distance (km)          0.00%
6             Orbiting Body           0.00%
7             Sentry Object           0.00%
8        Absolute Magnitude           0.00%
9     Potentially Hazardous           0.00%
```

Descriptive Statistics

In [7]:
```
# Applying the describe() to view some statistical information.
df.describe().T
```

Out[7]:

	count	mean	std	min	25%	
Asteroid ID	90836.0	1.438288e+07	2.087202e+07	2.000433e+06	3.448110e+06	3.74836
Estimated Diameter (Min)	90836.0	1.274321e-01	2.985112e-01	6.089126e-04	1.925551e-02	4.83676
Estimated Diameter (Max)	90836.0	2.849469e-01	6.674914e-01	1.361570e-03	4.305662e-02	1.0815:
Relative Velocity (km/s)	90836.0	4.806692e+04	2.529330e+04	2.033464e+02	2.861902e+04	4.41901
Miss Distance (km)	90836.0	3.706655e+07	2.235204e+07	6.745533e+03	1.721082e+07	3.78465
Absolute Magnitude	90836.0	2.352710e+01	2.894086e+00	9.230000e+00	2.134000e+01	2.3700(

In [8]:
```
# Applying the describe() to view some statistical information.
df.describe(include='object').T
```

Out[8]:

	count	unique	top	freq
Asteroid Name	90836	27423	469219 Kamo`oalewa (2016 HO3)	43
Orbiting Body	90836	1	Earth	90836

In [9]:

```
# Applying the describe() to view some statistical information.
df.describe(include='bool')
```

Out[9]:

	Sentry Object	Potentially Hazardous
count	90836	90836
unique	1	2
top	False	False
freq	90836	81996

Index Setting

In [10]:

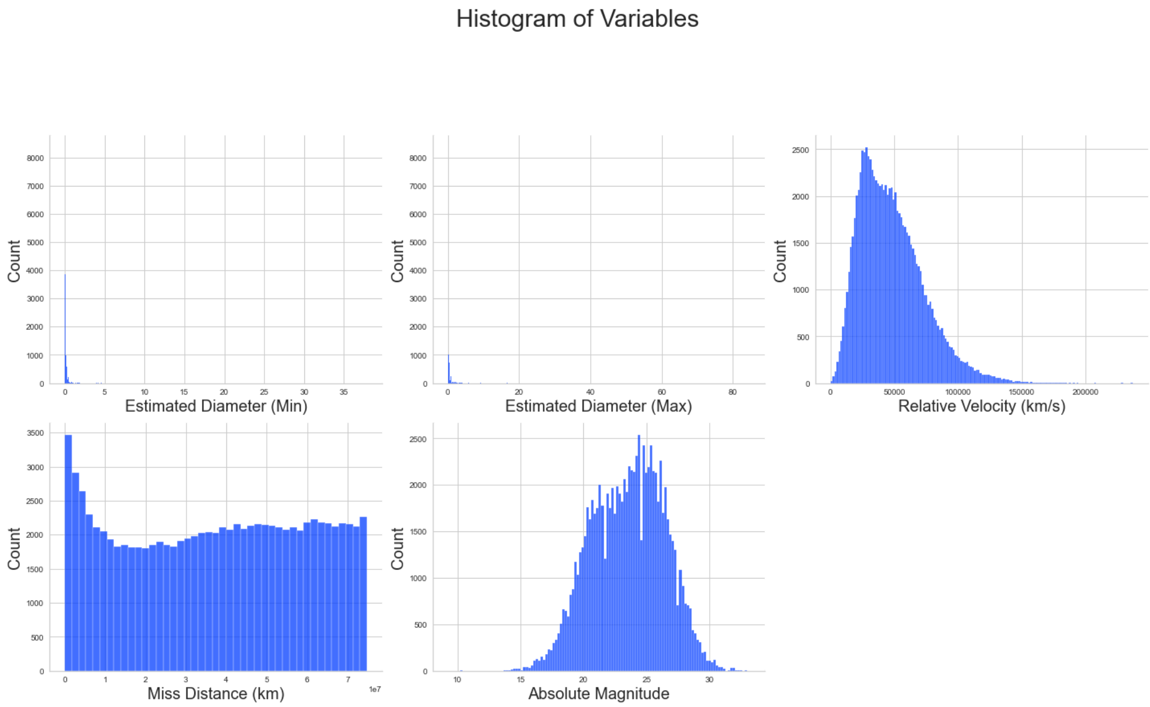
```
# Set the "ID" as the index.
df = df.set_index('Asteroid ID')

# Display first and last 5 records.
display(df.head())
display(df.tail())
```

	Asteroid Name	Estimated Diameter (Min)	Estimated Diameter (Max)	Relative Velocity (km/s)	Miss Distance (km)	Orbiting Body	Sentry Objec
Asteroid ID							
2162635	162635 (2000 SS164)	1.198271	2.679415	13569.249224	5.483974e+07	Earth	False
2277475	277475 (2005 WK4)	0.265800	0.594347	73588.726663	6.143813e+07	Earth	False
2512244	512244 (2015 YE18)	0.722030	1.614507	114258.692129	4.979872e+07	Earth	False
3596030	(2012 BV13)	0.096506	0.215794	24764.303138	2.543497e+07	Earth	False
3667127	(2014 GE35)	0.255009	0.570217	42737.733765	4.627557e+07	Earth	False

	Asteroid Name	Estimated Diameter (Min)	Estimated Diameter (Max)	Relative Velocity (km/s)	Miss Distance (km)	Orbiting Body	Sentı Objeı
Asteroid ID							
3763337	(2016 VX1)	0.026580	0.059435	52078.886692	1.230039e+07	Earth	Fals
3837603	(2019 AD3)	0.016771	0.037501	46114.605073	5.432121e+07	Earth	Fals
54017201	(2020 JP3)	0.031956	0.071456	7566.807732	2.840077e+07	Earth	Fals
54115824	(2021 CN5)	0.007321	0.016370	69199.154484	6.869206e+07	Earth	Fals
54205447	(2021 TW7)	0.039862	0.089133	27024.455553	5.977213e+07	Earth	Fals

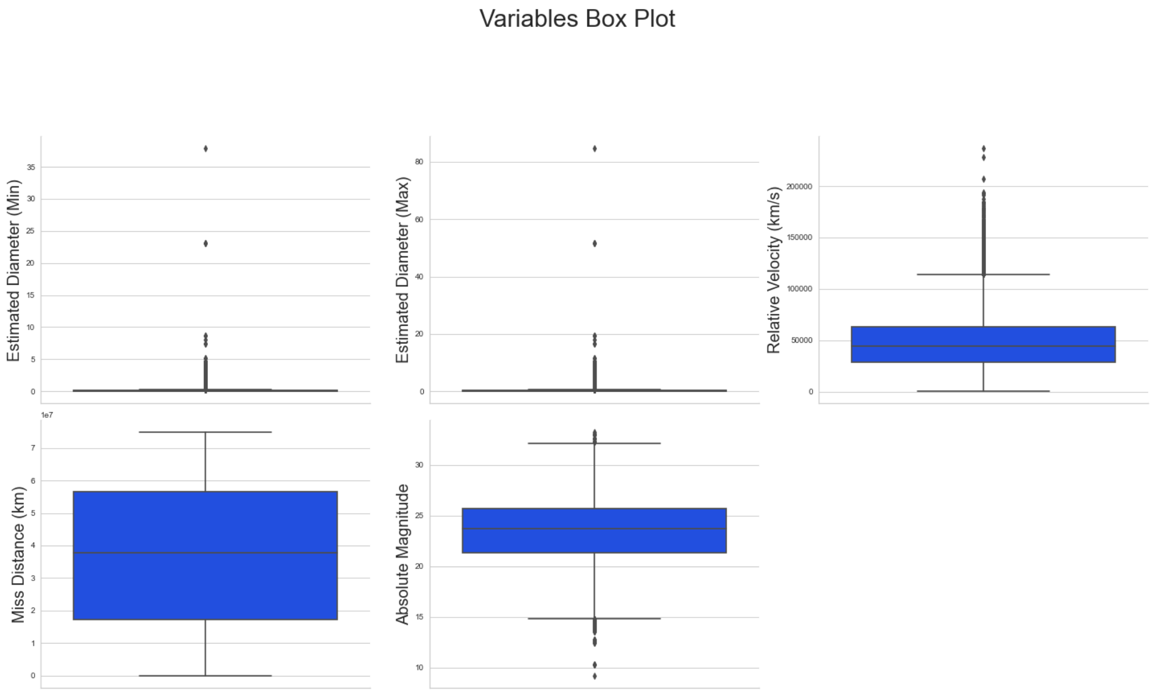
Data Visualization

In [11]:
```python
# Select variables for the histogram plot.
variables = ['Estimated Diameter (Min)','Estimated Diameter (Max)', 'Rela

# Create figure.
fig, axs = plt.subplots(nrows=6, ncols=3, figsize=(20, 30))
axs = axs.flatten() # Flatten axes.
plt.suptitle('Histogram of Variables', fontsize=30, y=1.05) #Title

# Loop variables and plot histogram.
for i, var in enumerate(variables):
    sns.histplot(df[var].dropna(), ax=axs[i], kde=False)
    axs[i].set_xlabel(var, fontsize=20)
    axs[i].set_ylabel('Count', fontsize=20)
    sns.despine()

# Check if there are any remaining axes and remove them.
if i < len(axs) - 1:
    for j in range(i+1, len(axs)):
        fig.delaxes(axs[j])

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

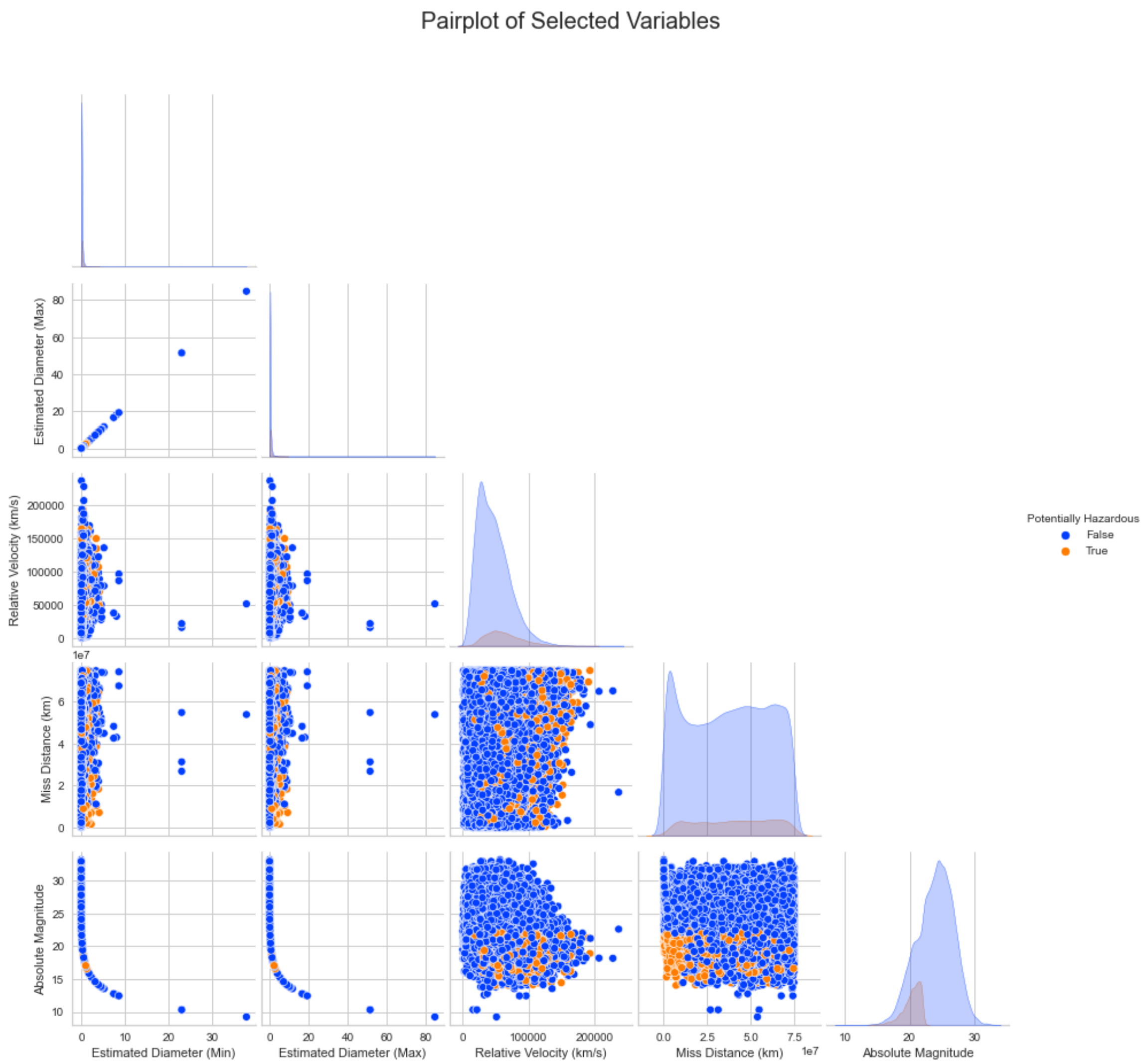


In [12]:

```
# Creating figure to the plot boxplots.
fig, axs = plt.subplots(nrows=6, ncols=3, figsize=(20, 30))
axs = axs.flatten() # Flatten axes.
plt.suptitle('Variables Box Plot', fontsize=30, y=1.05) #Title.

# Loop variables and plot boxplots.
for i, var in enumerate(variables):
    sns.boxplot(y=df[var].dropna(), ax=axs[i])
    axs[i].set_ylabel(var, fontsize=20)
    sns.despine()

# Checking if there are any axes and remove them.
if i < len(axs) - 1:
    for j in range(i+1, len(axs)):
        fig.delaxes(axs[j])

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

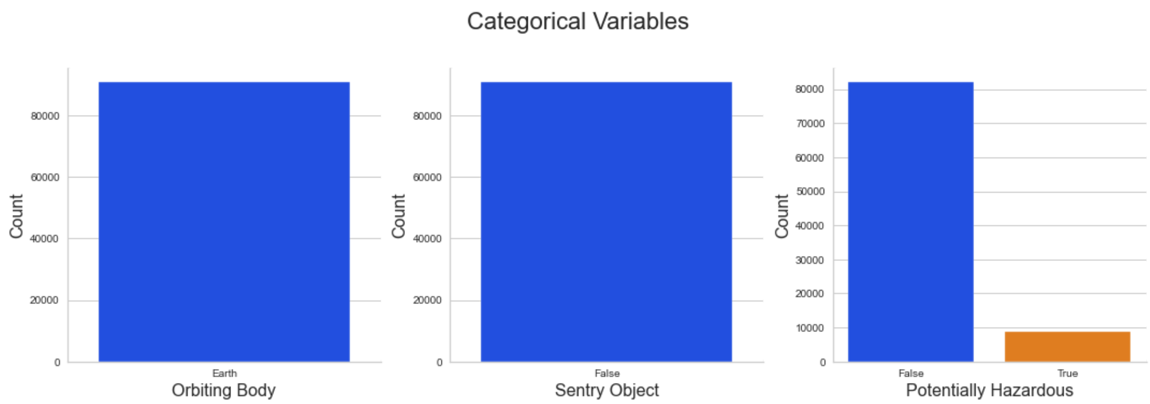


In [13]:
```
# Select variables for the pairplot.
variables = ["Potentially Hazardous",'Estimated Diameter (Min)', 'Estimat
df_selected = df[variables]

# Create pairplot with selected variables
g = sns.pairplot(df_selected.dropna(), hue="Potentially Hazardous", corne
g.fig.suptitle('Pairplot of Selected Variables', fontsize=20, y=1.03) #Ti
plt.subplots_adjust(top=0.95) # Adjust spacing between plots.
plt.show()
```

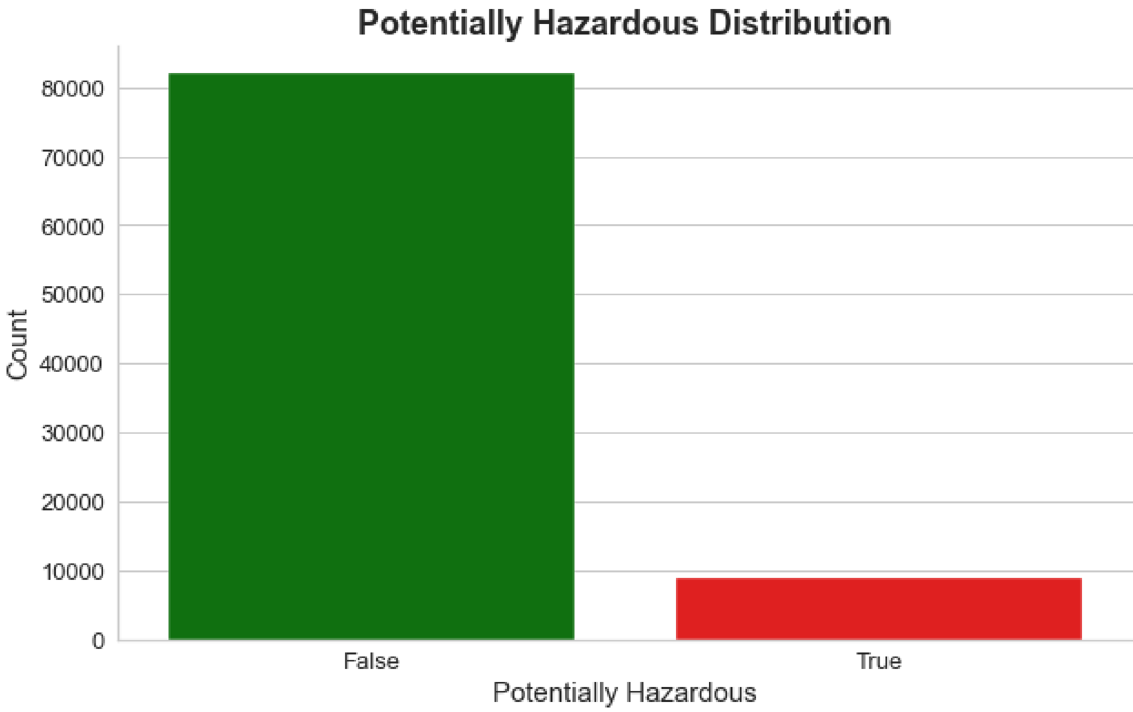


In [14]:

```
# Select the categorical variables
cat_vars = ['Orbiting Body', 'Sentry Object', 'Potentially Hazardous']
fig, axs = plt.subplots(nrows=1, ncols=len(cat_vars), figsize=(15,5))
fig.suptitle('Categorical Variables', fontsize=22, y=1.03) #Title.

# Loop through each categorical variable and create a count plot.
for i, var in enumerate(cat_vars):
    sns.countplot(data=df, x=var, ax=axs[i])  # Create count plot.
    axs[i].set_xlabel((var), fontsize=16)  # Set x-axis label
    axs[i].set_ylabel('Count',fontsize=16)  # Set y-axis label
    sns.despine()

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



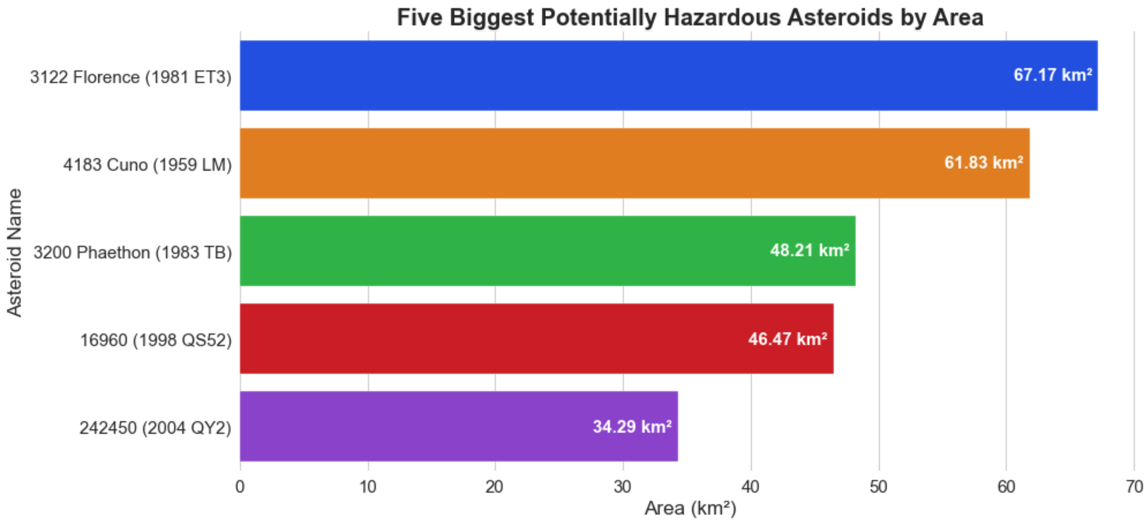
In [15]:
```
# Create a count plot of "Potentially Hazardous".
plt.figure(figsize=(10,6))
sns.countplot(data=df, x="Potentially Hazardous", palette={0: "green", 1:
plt.title("Potentially Hazardous Distribution", fontsize=18, fontweight="
plt.xlabel("Potentially Hazardous", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
sns.despine()
plt.show()

# Print the number of positive and negative classes
total_count = df.shape[0]
ph_count = df["Potentially Hazardous"].value_counts()[1]
non_ph_count = df["Potentially Hazardous"].value_counts()[0]
print(f"Number of Potentially Hazardous: {ph_count} ({ph_count/total_coun
print(f"Number of no Potentially Hazardous: {non_ph_count} ({non_ph_count
```




```
Number of Potentially Hazardous: 8840 (9.73%)
Number of no Potentially Hazardous: 81996 (90.27%)
```

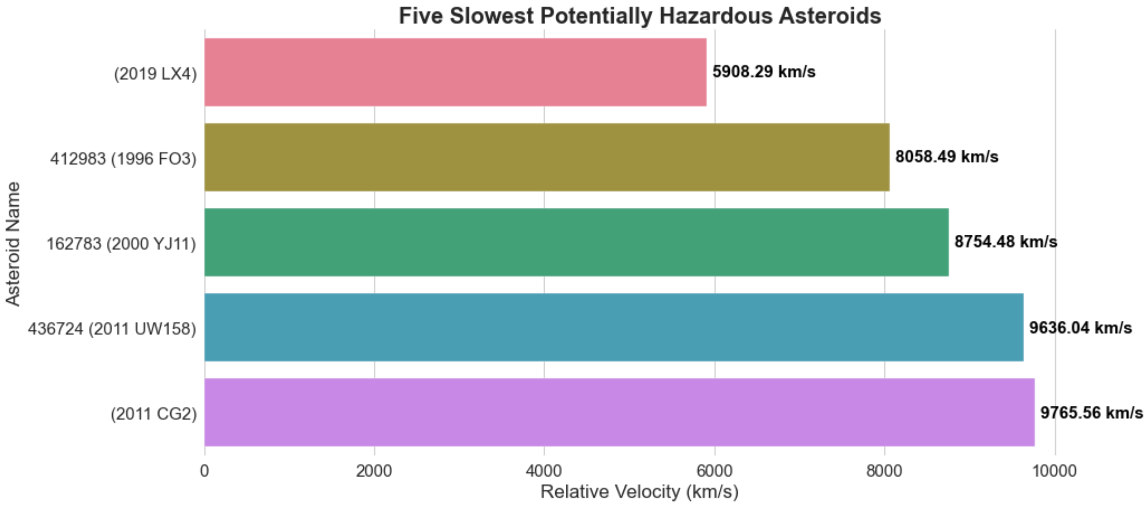
In [16]:
```
# Potentially Hazardous = 1
df_hazardous = df[df['Potentially Hazardous'] == 1]

# Estimated Diameter (Max) in descending order
df_biggers = df_hazardous.sort_values(by='Estimated Diameter (Max)', asce

# Create a new column with the diameter values in meters and and a new co
df_biggers['Diameter (km)'] = df_biggers['Estimated Diameter (Max)']
df_biggers['Area (km²)'] = (math.pi * ((df_biggers['Diameter (km)'] / 2)
df_biggers = df_biggers[['Asteroid Name', 'Estimated Diameter (Max)','Dia

# Seleciona as cinco maiores áreas.
df_biggers = df_biggers.sort_values(by='Area (km²)', ascending=False).hea

# Define graph width and height.
WIDTH = 1400
HEIGHT = 700
```

```
# Generate a barplot.
ax = sns.barplot(x='Area (km²)', y='Asteroid Name', data=df_biggers, orie
ax.set_title('Five Biggest Potentially Hazardous Asteroids by Area', font
ax.figure.set_size_inches(WIDTH/100, HEIGHT/100) # Set the width and heig
sns.despine(left=True, bottom=True) # Remove the spines.
ax.set_xlabel('Area (km²)', fontsize=17)
ax.set_ylabel('Asteroid Name', fontsize=17)
ax.tick_params(axis='x', which='both', labelsize=15)
ax.tick_params(axis='y', which='both', labelsize=15)

# Add the area value inside the bar.
for patch in ax.patches:
    area = patch.get_width()
    ax.annotate('{:.2f} km²'.format(area), xy=(patch.get_x() + patch.get_
        xytext=(-5, 0), textcoords="offset points", # Annotation position
        ha='right', va='center', fontsize=15, fontweight='bold', color='w

plt.show()
```



In [17]:
```
# Potentially Hazardous = 1
df_hazardous = df[df['Potentially Hazardous'] == 1]

# Relative Velocity (km/s) in ascending order
df_velocity = df_hazardous.sort_values(by='Relative Velocity (km/s)', asc

# Define graph width and height.
WIDTH = 1400
HEIGHT = 700

# Generate a barplot.
ax = sns.barplot(x='Relative Velocity (km/s)', y='Asteroid Name', data=df
ax.set_title('Five Slowest Potentially Hazardous Asteroids', fontsize=20,
ax.figure.set_size_inches(WIDTH/100, HEIGHT/100) # Set the width and heig
sns.despine(left=True, bottom=True) # Remove the spines.
ax.set_xlabel('Relative Velocity (km/s)', fontsize=17)
ax.set_ylabel('Asteroid Name', fontsize=17)
ax.tick_params(axis='x', which='both', labelsize=15)
ax.tick_params(axis='y', which='both', labelsize=15)

# Add the velocity value inside the bar.
for patch in ax.patches:
    velocity = patch.get_width()
    ax.annotate('{:.2f} km/s'.format(velocity), xy=(patch.get_x() + patch
```

```
        xytext=(5, 0), textcoords="offset points", # Annotation positions
        ha='left', va='center', fontsize=15, fontweight='bold', color='bl

plt.show()
```



Data preparation

Removing unnecessary columns

In [18]:
```
# Removing categorical variables from the dataset.
df = df.drop(['Asteroid Name','Orbiting Body', 'Sentry Object'], axis = 1
```

Checking the dependent variable and Encoder

In [19]:
```
# Creating a LabelEncoder object.
lab = LabelEncoder()

# Encoding the 'hazardous' variable using LabelEncoder.
df['Potentially Hazardous'] = lab.fit_transform(df['Potentially Hazardous
print("Unique values in the 'Potentially Hazardous' column:")
print(df['Potentially Hazardous'].unique())
```

```
Unique values in the 'Potentially Hazardous' column:
[0 1]
```

Checking the sparsity

In [20]:
```
def sparsity_density(df):
    non_zero = np.count_nonzero(df)
    total_val = np.product(df.shape)
    sparsity = (total_val - non_zero) / total_val
    density = non_zero / total_val

    print("Density:  ", round(density, 4))
    print("Sparsity: ", round(sparsity, 4))
```

```
# Calculating the percentage with the function.
sparsity_density(df) # Confirming if data is sparse.

# Checking if the data is sparse.
if sparse.issparse(df):
    print("This dataframe is sparse.")
else:
    print("This dataframe is not sparse.")
```

```
Density:   0.8496
Sparsity:  0.1504
This dataframe is not sparse.
```

Spliting the dataset into independent and dependent variables

In [21]:

```
# Spliting the dataset into independent and dependent variables.
X = df.drop(["Potentially Hazardous"], axis = 1) # Independent variables
y = df["Potentially Hazardous"]                  # Dependent variables.
```

Checking Outliers

In [22]:

```
def count_outliers(df):
    # Calculate the interquartile range (IQR).
    q1 = df.drop(["Potentially Hazardous"], axis = 1).quantile(0.25)
    q3 = df.drop(["Potentially Hazardous"], axis = 1).quantile(0.75)
    iqr = q3 - q1

    # Define the threshold for outliers.
    threshold = 1.5 * iqr

    # Count the number of outliers for each variable.
    outliers = pd.DataFrame()
    for col in df.drop(["Potentially Hazardous"], axis = 1).columns:
        lower = q1[col] - threshold[col]
        upper = q3[col] + threshold[col]
        count = len(df[(df[col] < lower) | (df[col] > upper)])
        percentage = round(100 * count / len(df), 2)
        outliers[col] = [count, percentage]

    # Sort the DataFrame by ascending outlier count.
    outliers = outliers.T.sort_values(by=0)
    outliers.columns = ['Outliers', 'Percentage']

    return outliers
```

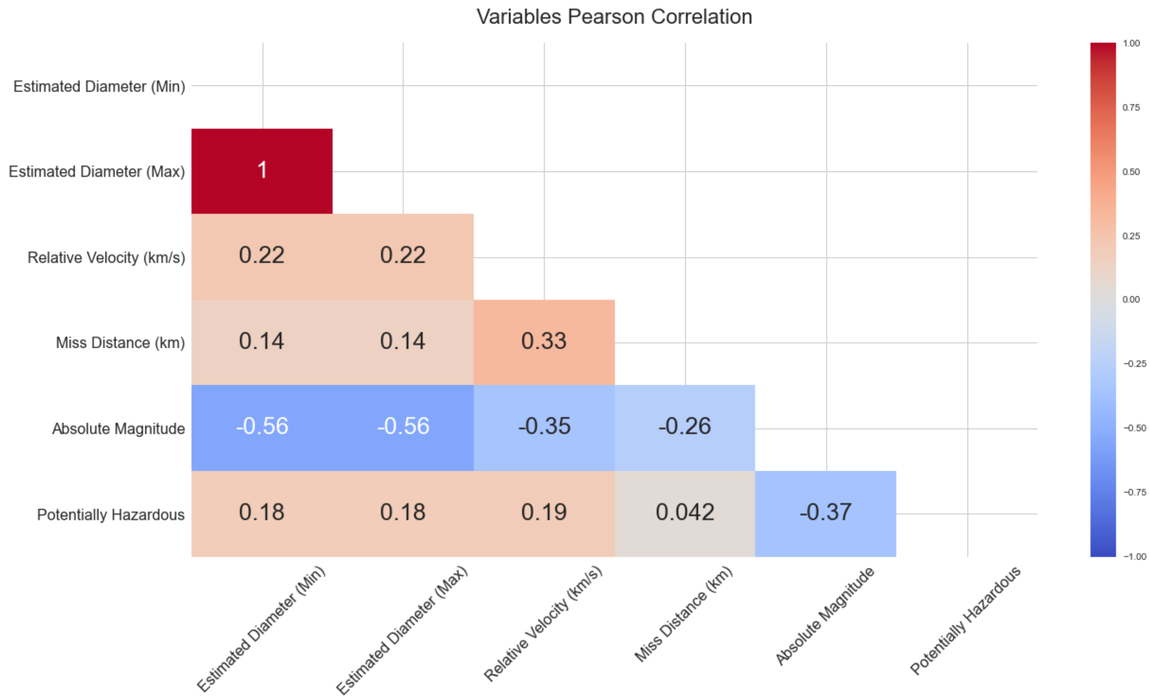
In [23]:

```
outliers = count_outliers(df)
print(outliers)
```

```
                           Outliers  Percentage
Miss Distance (km)              0.0        0.00
Absolute Magnitude            101.0        0.11
Relative Velocity (km/s)     1574.0        1.73
Estimated Diameter (Min)     8306.0        9.14
Estimated Diameter (Max)     8306.0        9.14
```

Correlation

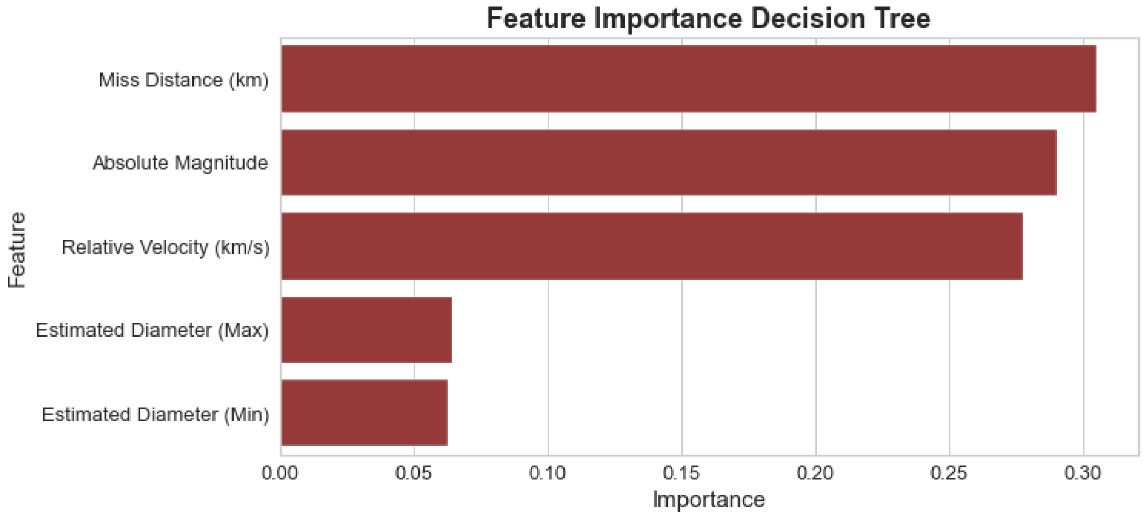
In [24]:
```
# Correlation of numerical variables
plt.figure(figsize=(20, 10))
mask = np.triu(np.ones_like(df.corr(method="pearson"), dtype=np.bool))
heatmap = sns.heatmap(df.corr(), mask=mask, vmin=-1, vmax=1, annot=True,
heatmap.set_title('Variables Pearson Correlation', fontdict={'fontsize':2
plt.xticks(rotation=45,fontsize=16)
plt.yticks(rotation=0,fontsize=16)
plt.show()
```



Decision Trees, Random Forest and Feature Importance

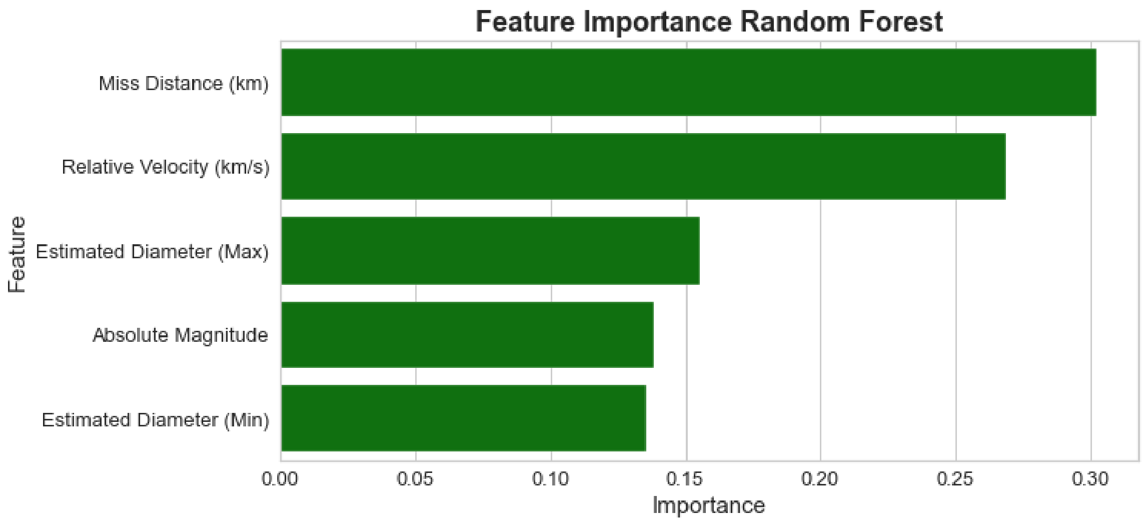
In [25]:
```
# Creating a Decision Tree Classifier.
decision_tree = DecisionTreeClassifier(random_state=0)
decision_tree.fit(X, y) # Fit the model.

# Get feature importances,creating a DataFrame to store the feature impor
importances = decision_tree.feature_importances_
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importa

# Plot a bar chart to visualize the feature importances.
plt.figure(figsize=(10,5))
sns.barplot(data=importance_df.head(10), x='Importance', y='Feature',colo
plt.title('Feature Importance Decision Tree', fontsize=18, fontweight='bo
plt.xlabel('Importance', fontsize=15)
```

```
plt.ylabel('Feature', fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.show()
```



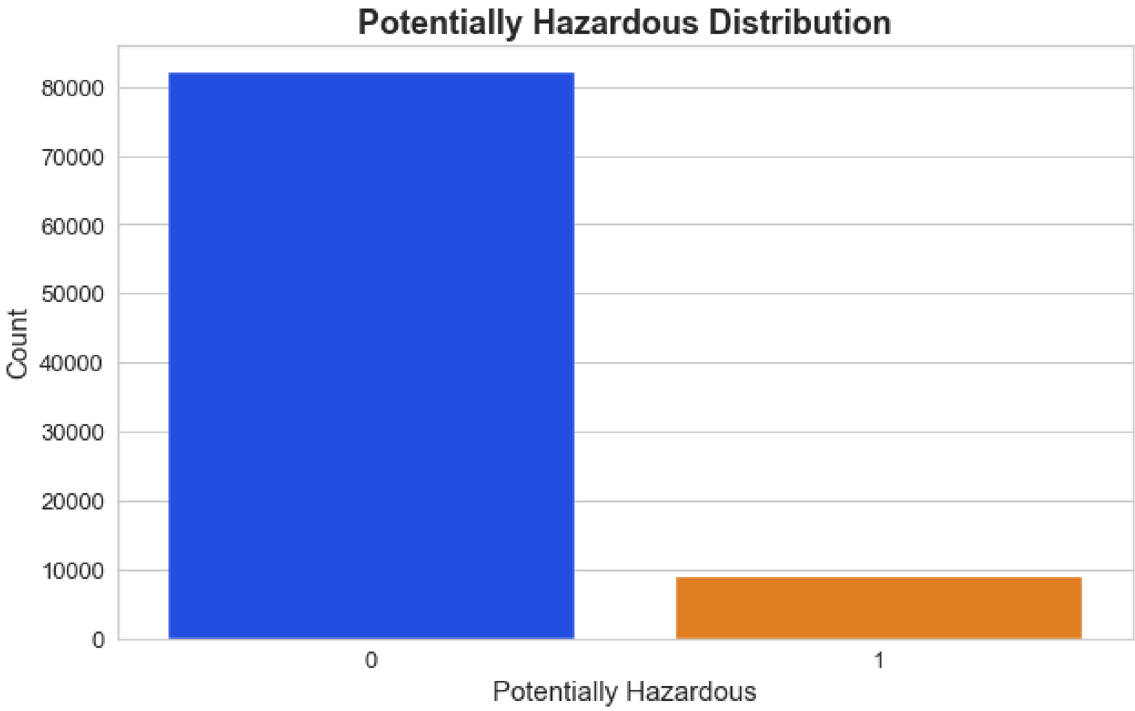
In [26]:

```
# Creating a Random Forest Classifier.
random_forest = RandomForestClassifier(random_state=0)
random_forest.fit(X, y) # Fit the model.

# Get feature importances, creating a DataFrame to store the feature impo
importances = random_forest.feature_importances_
importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': importa

# Plot a bar chart to visualize the feature importances.
plt.figure(figsize=(10,5))
sns.barplot(data=importance_df.head(), x='Importance', y='Feature', color
plt.title('Feature Importance Random Forest', fontsize=18, fontweight='bo
plt.xlabel('Importance', fontsize=15)
plt.ylabel('Feature', fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.show()
```



Adjusting the imbalance

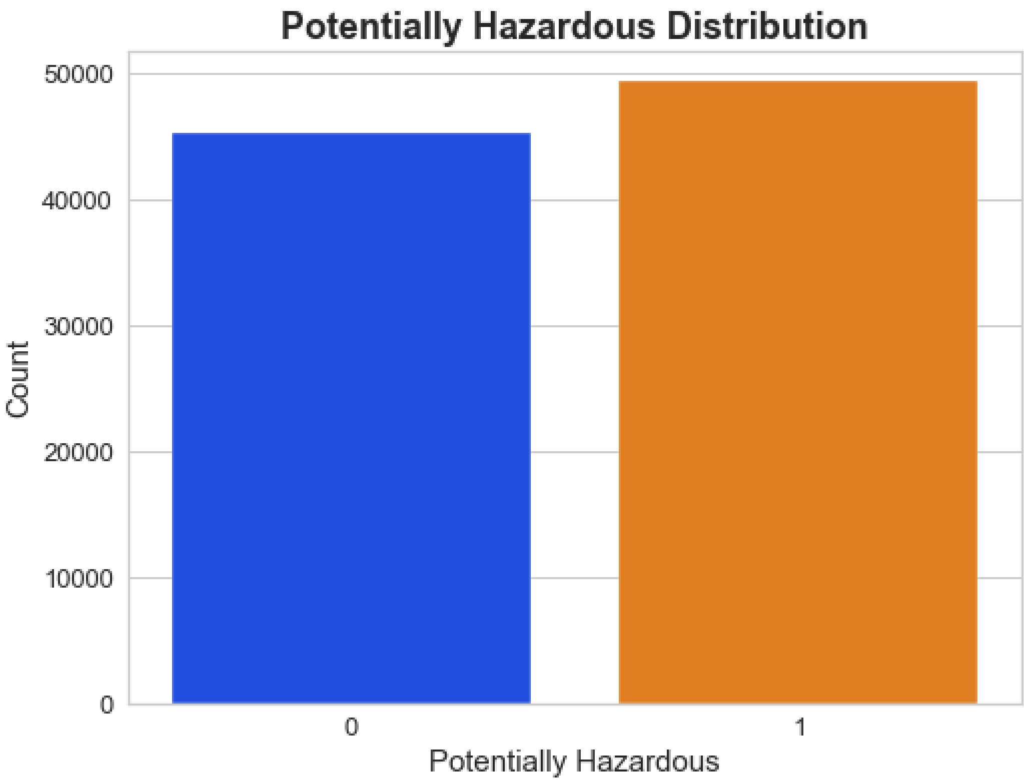
In [27]:
```
# Create a count plot of "Potentially Hazardous".
plt.figure(figsize=(10,6))
sns.countplot(data=df, x="Potentially Hazardous")
plt.title("Potentially Hazardous Distribution", fontsize=18, fontweight="
plt.xlabel("Potentially Hazardous", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.show()

# Print the number of positive and negative classes
print("Number of Potentially Hazardous:", df["Potentially Hazardous"].val
print("Number of no Potentially Hazardous:", df["Potentially Hazardous"].
plt.show()
```




```
Number of Potentially Hazardous: 8840
Number of no Potentially Hazardous: 81996
```

In [28]:
```
# Oversampling the train dataset using SMOTE + ENN.
smenn = SMOTEENN()
X_res, y_res = smenn.fit_resample(X, y)
print("Number of Potentially Hazardous after Smottter:", df["Potentially
print("Number of no Potentially Hazardous after Smottem", df["Potentially

# Plotting bar chart for y after SMOTEENN.
plt.figure(figsize=(8,6))
sns.countplot(y_res)
plt.title("\nPotentially Hazardous Distribution", fontsize=18, fontweight
plt.xlabel("Potentially Hazardous", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.xticks(fontsize=13)
plt.yticks(fontsize=13)
plt.show()
```

```
Number of Potentially Hazardous after Smottter: 8840
Number of no Potentially Hazardous after Smottem 81996
```

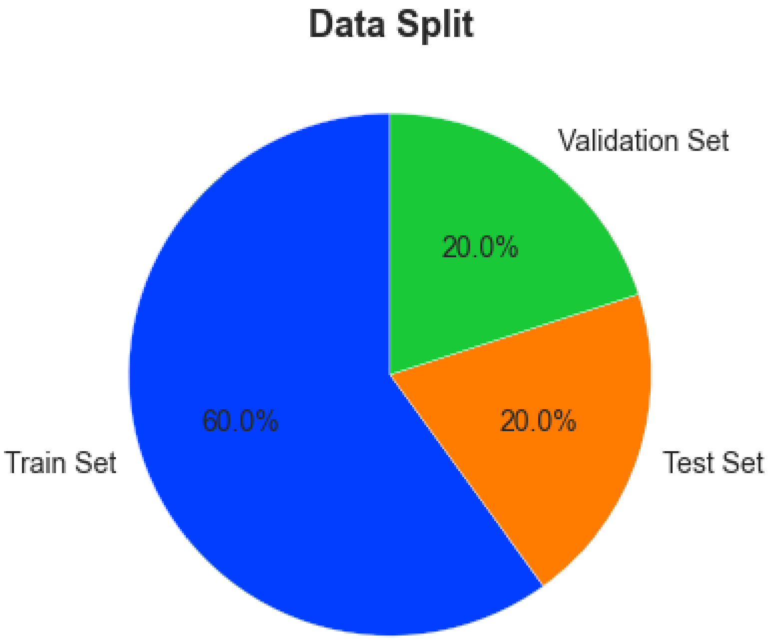


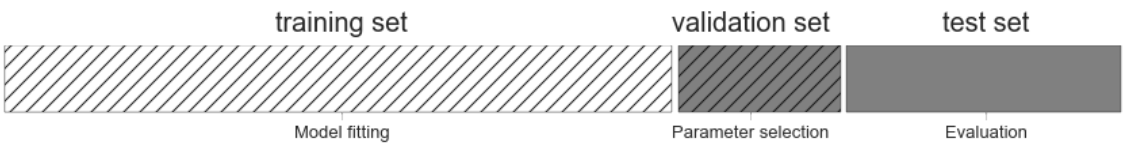
Splitting the Data-Set into Training Set and Test Set

In [29]:
```
X = X_res # Independent variables
y = y_res # Dependent variables.

# Split dataset into training and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
# Split training set into training and validation sets.
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_

# Print the shape of training, validation, and test.
print('Total number of rows and columns for the complete dataset: ', X.sh
print("X_train shape: {}".format(X_train.shape))
print("y_train shape: {}".format(y_train.shape))
print("X_val shape: {}".format(X_val.shape))
print("y_val shape: {}".format(y_val.shape))
print("X_test shape: {}".format(X_test.shape))
print("y_test shape: {}".format(y_test.shape))


# Define the sizes of the train, test, and validation sets.
train_size = len(X_train)
test_size = len(X_test)
val_size = len(X_val)
labels = ['Train Set', 'Test Set', 'Validation Set'] # Define the labels
sizes = [train_size, test_size, val_size]
plt.figure(figsize=(8,6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, textprops
plt.title('Data Split',fontsize=18, fontweight="bold") # Title.
plt.show()
```

```
# Illustration of train_test_split.
mglearn.plots.plot_threefold_split()
```

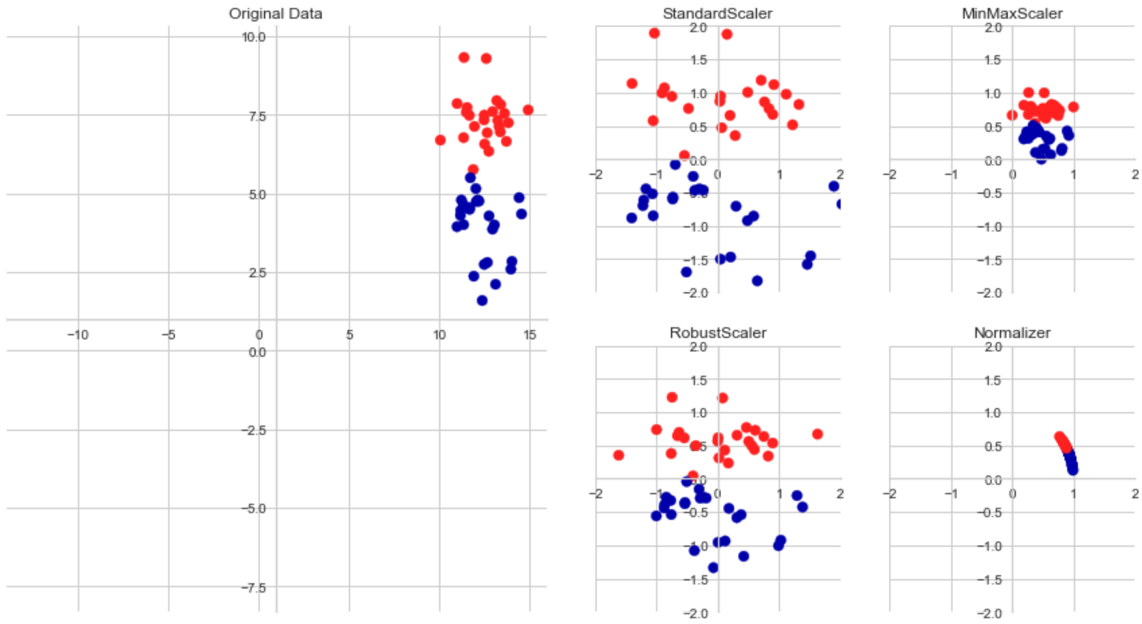
```
Total number of rows and columns for the complete dataset:  (94643, 5)
(94643,)
X_train shape: (56785, 5)
y_train shape: (56785,)
X_val shape: (18929, 5)
y_val shape: (18929,)
X_test shape: (18929, 5)
y_test shape: (18929,)
```





Normalization the data

In [30]:
```
# Creating a MinMaxScaler() object.
scaler = MinMaxScaler()

# Scale the features in the training set using fit_transform().
X_train = scaler.fit_transform(X_train.astype(np.float))

# Scale the features in the validation set and test set using transform()
X_val = scaler.transform(X_val.astype(np.float))
X_test= scaler.transform(X_test.astype(np.float))

# Graphical representation of different scaling processes.
mglearn.plots.plot_scaling()
```

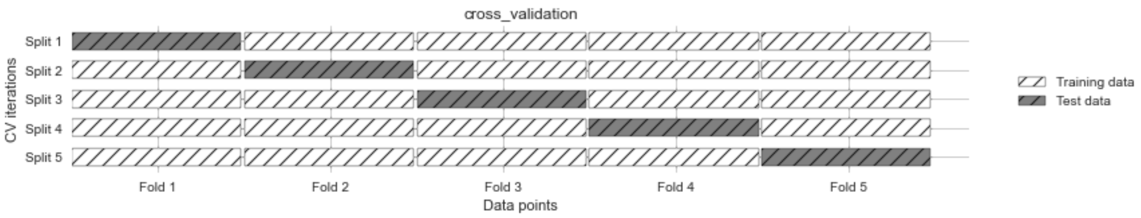


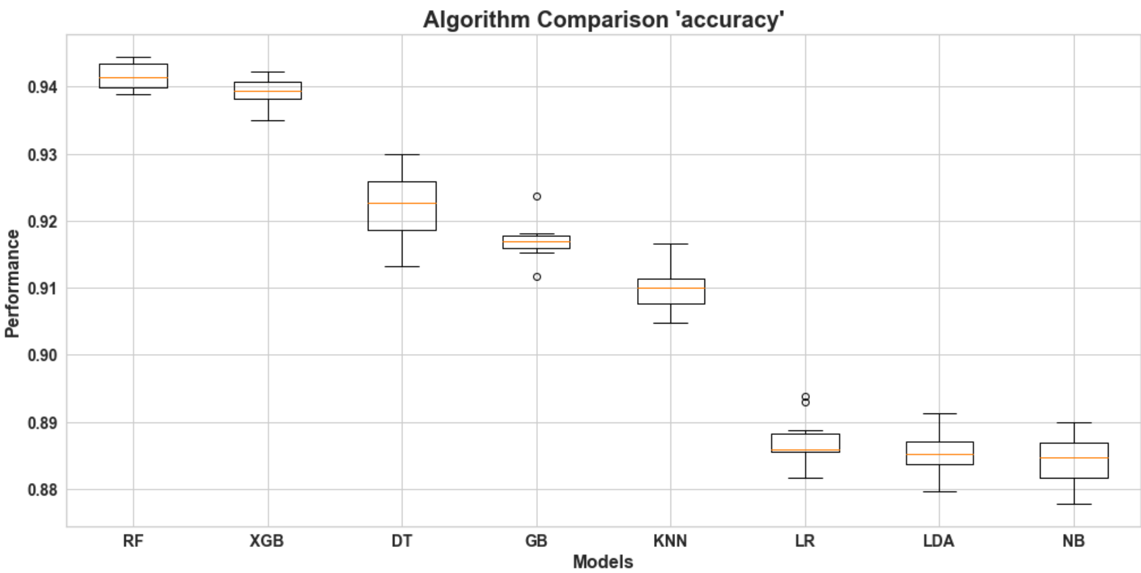
Machine Learning Models Comparison (X_train, y_train)

In [31]:
```
# Create a list of classifiers.
models = []
models.append(('RF', RandomForestClassifier(random_state=0)))
models.append(('DT', DecisionTreeClassifier(random_state=0)))
models.append(('LR', LogisticRegression(random_state=0)))
models.append(('KNN', KNeighborsClassifier()))
models.append(('NB', GaussianNB()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('GB', GradientBoostingClassifier(random_state=0)))
models.append(('XGB', XGBClassifier(random_state=0)))

# Informative of cross validation system.
mglearn.plots.plot_cross_validation()

# Running models with cross-validation on the training data.
results = []
names = []
times = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=0, shuffle=True)
    start_time = time.time()
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold, scorin
    end_time = time.time()
    results.append(cv_scores)
    names.append(name)
    times.append(end_time - start_time)
    print(f'{name}: {cv_scores.mean():.4f} (±{cv_scores.std():.4f}) - Tim

# Sort results by descending performance.
idx = np.argsort([-np.mean(result) for result in results])
results_sorted = [results[i] for i in idx]
names_sorted = [names[i] for i in idx]

# Boxplot comparing resulting from the accuracies of models.
plt.figure(figsize=(17, 8))
pyplot.boxplot(results_sorted, labels=names_sorted)
```

```
plt.title("Algorithm Comparison 'accuracy'", fontweight="bold", fontsize=
plt.xlabel("Models", fontweight="bold", fontsize=16)
plt.ylabel("Performance", fontweight="bold", fontsize=16)
plt.xticks(fontweight="bold", fontsize=14)
plt.yticks(fontweight="bold", fontsize=14)
plt.show()
```

```
RF: 0.9416 (±0.0020) - Time: 83.54s
DT: 0.9220 (±0.0049) - Time: 2.73s
LR: 0.8870 (±0.0037) - Time: 0.47s
KNN: 0.9098 (±0.0033) - Time: 1.46s
NB: 0.8842 (±0.0035) - Time: 0.09s
LDA: 0.8854 (±0.0031) - Time: 0.53s
GB: 0.9171 (±0.0028) - Time: 74.34s
XGB: 0.9393 (±0.0020) - Time: 14.89s
```

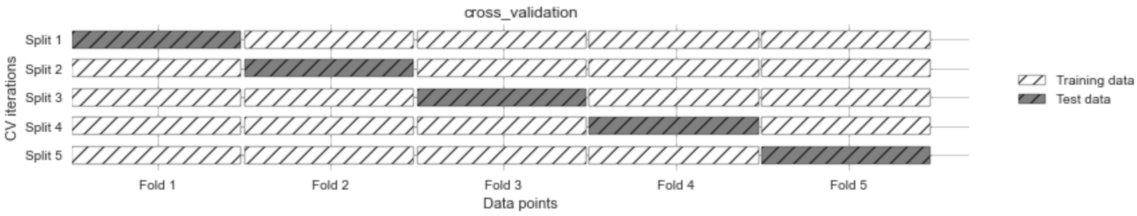


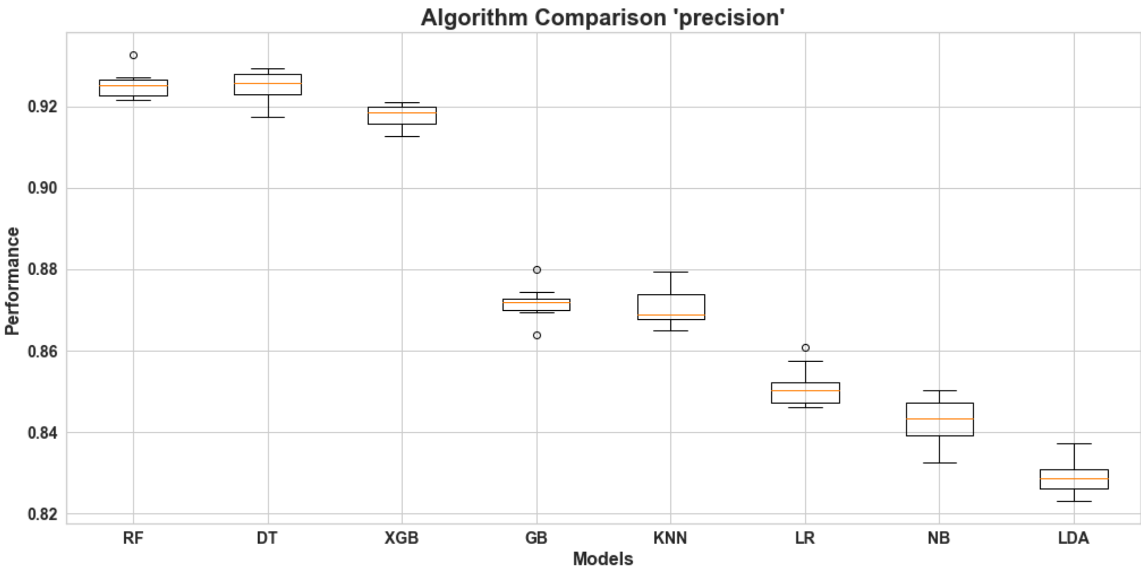


In [32]:
```
# Informative of cross validation system.
mglearn.plots.plot_cross_validation()

# Running models with cross-validation on the training data
results = []
names = []
times = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=0, shuffle=True)
    start_time = time.time()
    cv_scores = cross_val_score(model, X_train, y_train, cv=kfold, scorin
    end_time = time.time()
    results.append(cv_scores)
    names.append(name)
    times.append(end_time - start_time)
    print(f'{name}: {cv_scores.mean():.4f} (±{cv_scores.std():.4f}) - Tim

# Sort results by descending performance.
idx = np.argsort([-np.mean(result) for result in results])
results_sorted = [results[i] for i in idx]
```

```
names_sorted = [names[i] for i in idx]

# Boxplot comparing resulting from the accuracies of models
plt.figure(figsize=(17, 8))
pyplot.boxplot(results_sorted, labels=names_sorted)
plt.title("Algorithm Comparison 'precision'", fontweight="bold", fontsize
plt.xlabel("Models", fontweight="bold", fontsize=16)
plt.ylabel("Performance", fontweight="bold", fontsize=16)
plt.xticks(fontweight="bold", fontsize=14)
plt.yticks(fontweight="bold", fontsize=14)
plt.show()
```

```
RF: 0.9253 (±0.0031) - Time: 83.57s
DT: 0.9249 (±0.0037) - Time: 2.72s
LR: 0.8509 (±0.0047) - Time: 0.44s
KNN: 0.8707 (±0.0042) - Time: 1.44s
NB: 0.8430 (±0.0054) - Time: 0.10s
LDA: 0.8287 (±0.0042) - Time: 0.52s
GB: 0.8717 (±0.0039) - Time: 73.64s
XGB: 0.9177 (±0.0029) - Time: 15.00s
```





Hyperparameter tuning comparison (X_val_scaled, y_val)

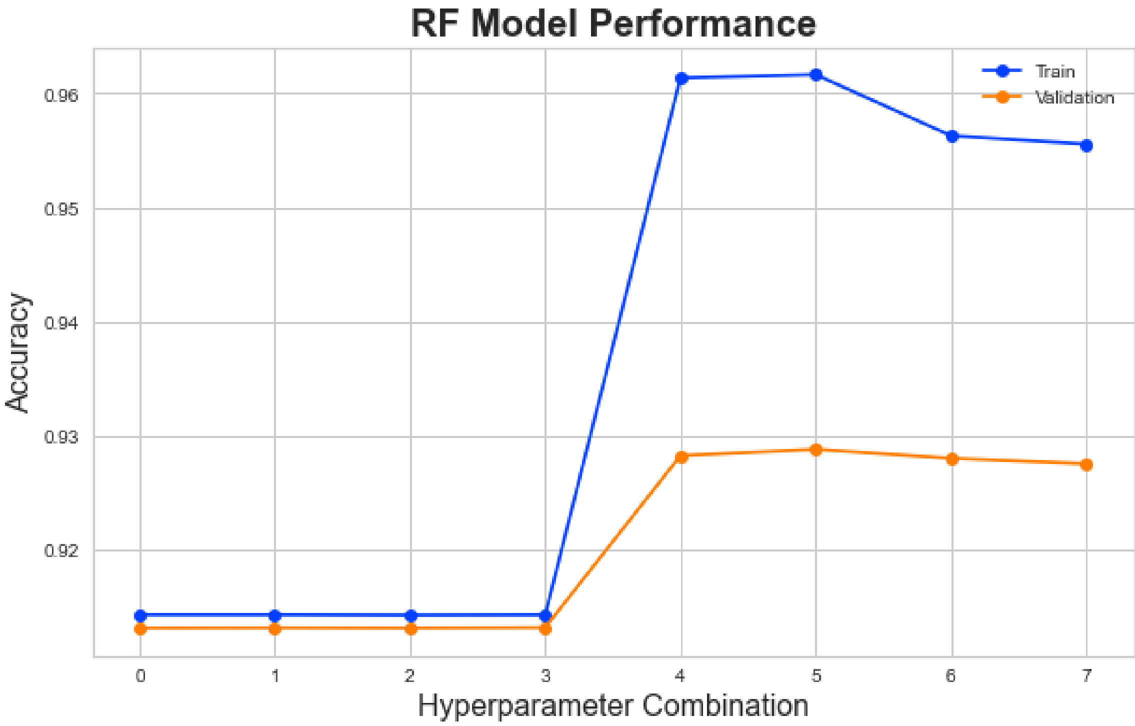
Model	Parameters Selected	Parameters Description
Random Forest Classifier	int, default=100	The number of trees in the forest.
	max_depthint, default=None	The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
	min_samples_splitint or float, default=2	The minimum number of samples required to split an internal node. If int, then consider min_samples_split as the minimum number. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
XGB Classifier	n_estimators, default value is 100	This parameter corresponds to the number of trees to be created in the gradient boosting model.
	learning_rate, default value is 0.1	This parameter is the step size used in updating the model weights.
	max_depth, default value is 3	This parameter is the maximum depth of each tree in the gradient boosting model.
Kneighbors Classifier	n_neighborsint, default=5	Number of neighbors to use by default for kneighbors queries.
	weights{'uniform', 'distance'}, callable or None, default='uniform'	Weight function used in prediction. Possible values: 'uniform' : uniform weights, all points in each neighborhood are weighted equally. 'distance': weight points by the inverse of their distance, in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
	algorithm{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'	Algorithm used to compute the nearest neighbors: 'ball_tree' will use BallTree 'kd_tree' will use KDTree 'brute' will use a brute-force search 'auto' will attempt to decide the most appropriate algorithm based on the values passed to fit method.

In [33]:

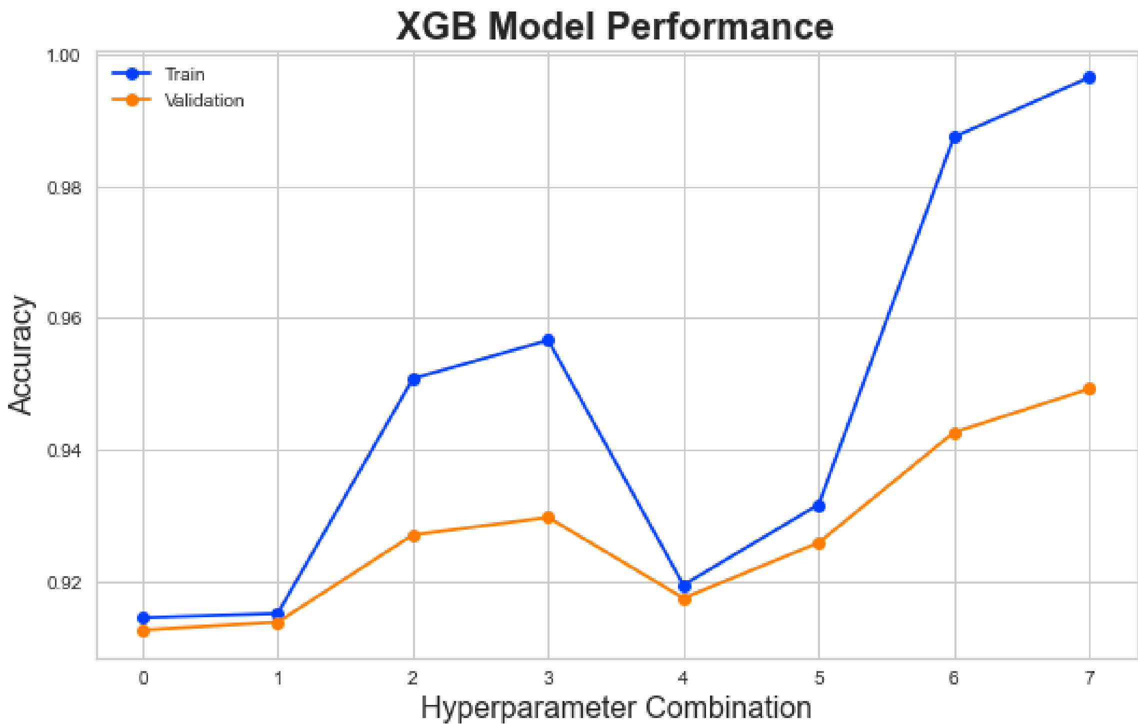
```
# Set up the parameter grids for each model.
param_grid_rf = {
    'n_estimators': [20, 80],
    'max_depth': [5, 20],
    'min_samples_split': [2, 5]
}
param_grid_xgb = {
    'n_estimators': [70, 150],
    'learning_rate': [0.01, 0.1],
    'max_depth': [5, 20]
}
param_grid_knn = {
    'n_neighbors': [3, 8],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

# Create a list of models and parameter grids.
models = []
models.append(('RF', RandomForestClassifier(random_state=0)))
models.append(('XGB', XGBClassifier(random_state=0)))
models.append(('KNN', KNeighborsClassifier()))
param_grids = []
param_grids.append(param_grid_rf)
param_grids.append(param_grid_xgb)
param_grids.append(param_grid_knn)

# Run grid search on each model.
for i, (name, model) in enumerate(models):
    print(f"Running GridSearchCV for {name}")
    inner_cv = KFold(n_splits=10, shuffle=True, random_state=0)
    grid_search = GridSearchCV(estimator=model, param_grid=param_grids[i]
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    val_acc = best_model.score(X_val, y_val)
    print(f"Best parameters for {name}: {grid_search.best_params_}") # Wh
    print(f"Validation accuracy for {name}: {val_acc:.3f}") # The accurac

    outer_cv = KFold(n_splits=10, shuffle=True, random_state=0)
```

```
    scores = cross_val_score(best_model, X_train, y_train, cv=outer_cv)
    print(f"Test accuracy for {name}: {scores.mean():.3f}") # The accurac

    # Create a list of scores for each hyperparameter combination.
    scores_train = list(grid_search.cv_results_['mean_train_score'])
    scores_val = list(grid_search.cv_results_['mean_test_score'])

    # Plot the training and validation scores
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(scores_train)), scores_train, '-o', label='Train')
    plt.plot(range(len(scores_val)), scores_val, '-o', label='Validation'
    plt.title(f'{name} Model Performance',fontweight="bold", fontsize=20)
    plt.xlabel('Hyperparameter Combination',fontsize=16)
    plt.ylabel('Accuracy',fontsize=16)
    plt.legend()
    plt.show()
```

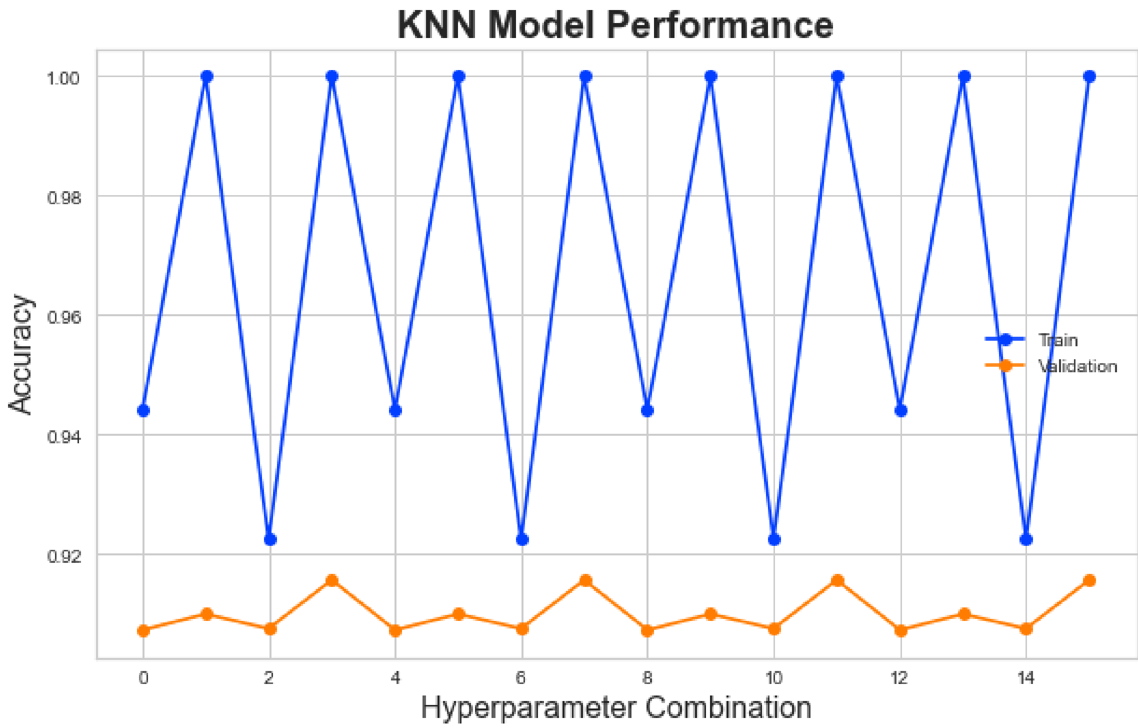
```
Running GridSearchCV for RF
Best parameters for RF: {'max_depth': 20, 'min_samples_split': 2, 'n_est
imators': 80}
Validation accuracy for RF: 0.932
Test accuracy for RF: 0.929
```




```
Running GridSearchCV for XGB
Best parameters for XGB: {'learning_rate': 0.1, 'max_depth': 20, 'n_esti
mators': 150}
Validation accuracy for XGB: 0.952
Test accuracy for XGB: 0.949
```



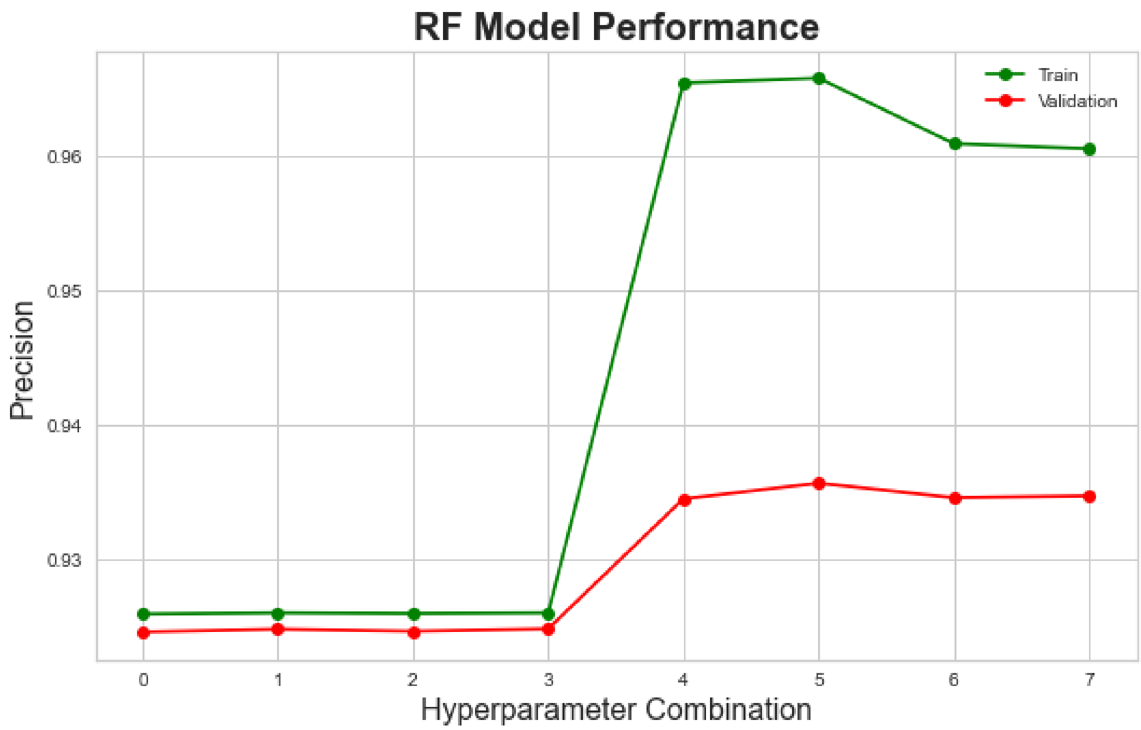

```
Running GridSearchCV for KNN
Best parameters for KNN: {'algorithm': 'auto', 'n_neighbors': 8, 'weight
s': 'distance'}
Validation accuracy for KNN: 0.920
Test accuracy for KNN: 0.916
```



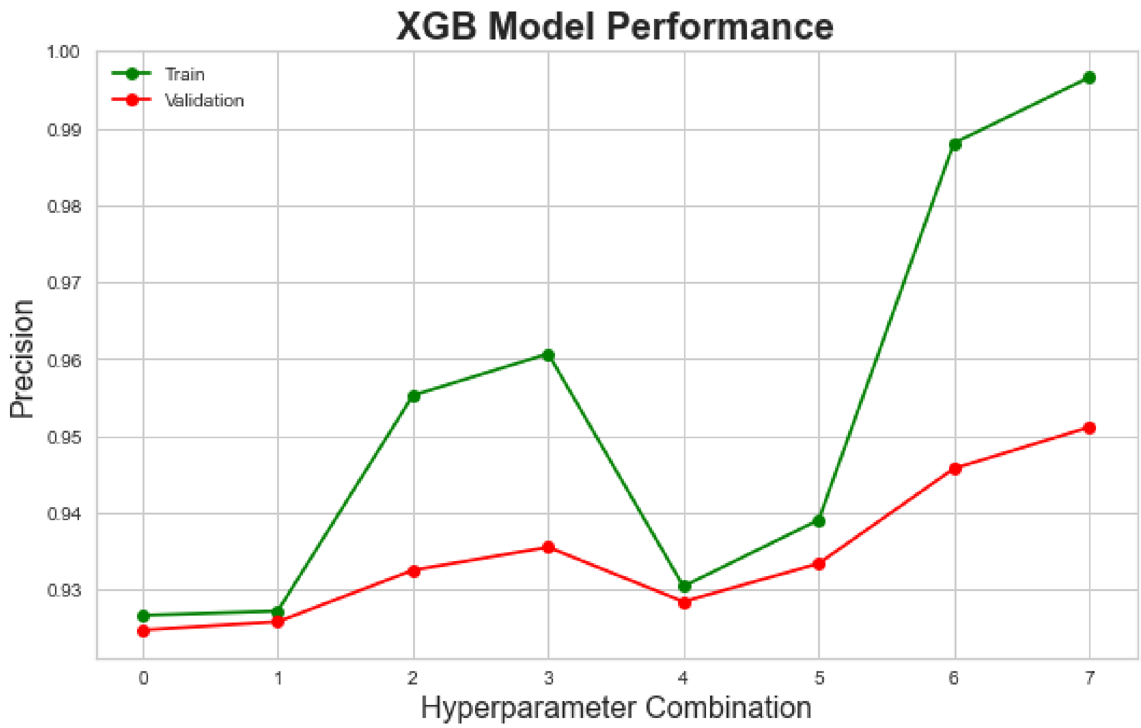
In [34]:
```
# Run grid search on each model.
for i, (name, model) in enumerate(models):
    print(f"Running GridSearchCV for {name}")
    inner_cv = KFold(n_splits=10, shuffle=True, random_state=0)
    grid_search = GridSearchCV(estimator=model, param_grid=param_grids[i]
    grid_search.fit(X_train, y_train)
    best_model = grid_search.best_estimator_
    val_acc = best_model.score(X_val, y_val)
    print(f"Best parameters for {name}: {grid_search.best_params_}")
    print(f"Validation precision for {name}: {val_acc:.3f}")
```

```
    outer_cv = KFold(n_splits=10, shuffle=True, random_state=0)
    scores = cross_val_score(best_model, X_train, y_train, cv=outer_cv, s
    print(f"Test precision for {name}: {scores.mean():.3f}")

    # Create a list of scores for each hyperparameter combination.
    scores_train = list(grid_search.cv_results_['mean_train_score'])
    scores_val = list(grid_search.cv_results_['mean_test_score'])

    # Plot the training and validation scores
    plt.figure(figsize=(10, 6))
    plt.plot(range(len(scores_train)), scores_train, '-o', label='Train',
    plt.plot(range(len(scores_val)), scores_val, '-o', label='Validation'
    plt.title(f'{name} Model Performance',fontweight="bold", fontsize=20)
    plt.xlabel('Hyperparameter Combination',fontsize=16)
    plt.ylabel('Precision',fontsize=16)
    plt.legend()
    plt.show()
```

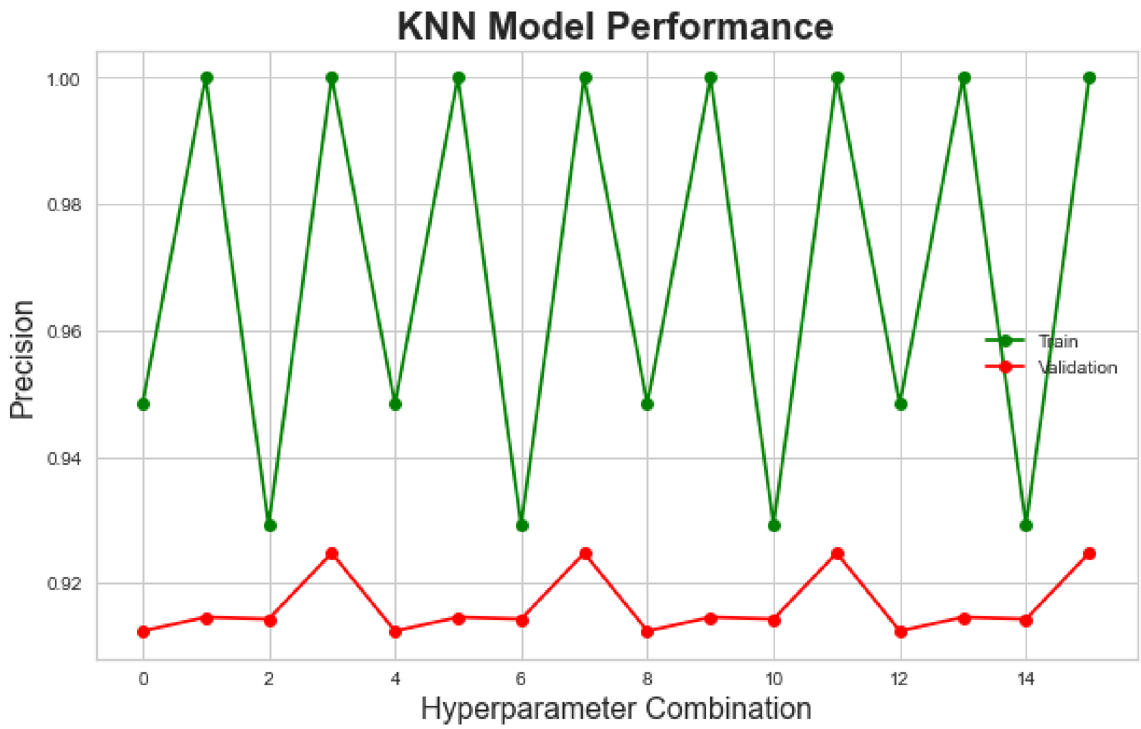
```
Running GridSearchCV for RF
Best parameters for RF: {'max_depth': 20, 'min_samples_split': 2, 'n_est
imators': 80}
Validation precision for RF: 0.932
Test precision for RF: 0.936
```




```
Running GridSearchCV for XGB
Best parameters for XGB: {'learning_rate': 0.1, 'max_depth': 20, 'n_esti
mators': 150}
Validation precision for XGB: 0.952
Test precision for XGB: 0.951
```




```
Running GridSearchCV for KNN
Best parameters for KNN: {'algorithm': 'auto', 'n_neighbors': 8, 'weight
s': 'distance'}
Validation precision for KNN: 0.920
Test precision for KNN: 0.925
```

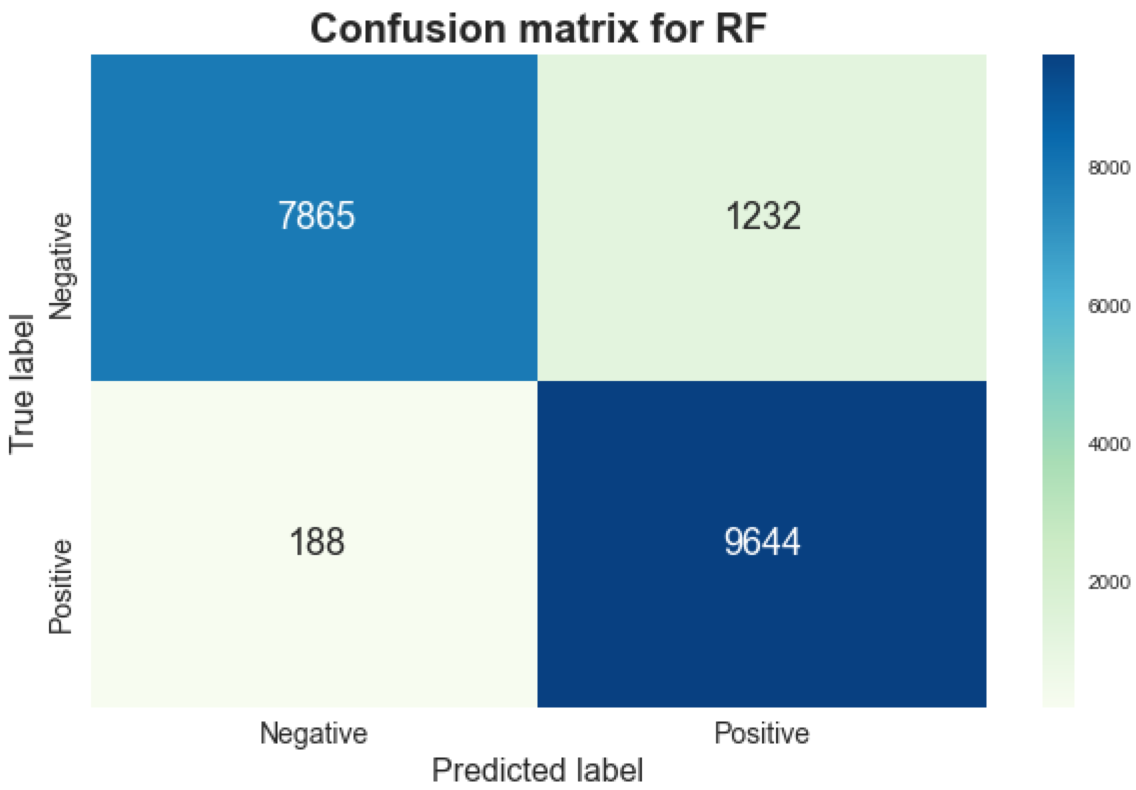


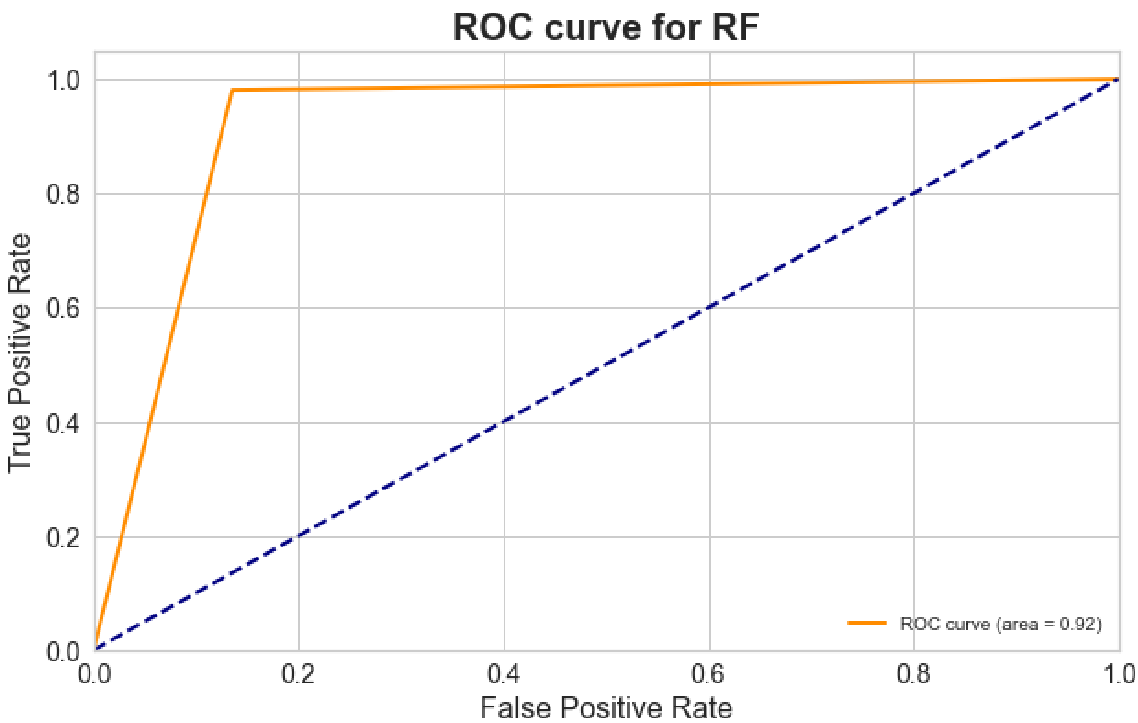
Model Evaluation (X_test, y_test)

In [36]:
```
# Define the models.
models = []
models.append(('RF', RandomForestClassifier(max_depth=20, min_samples_spl
models.append(('XGB', XGBClassifier(learning_rate=0.1, max_depth=20, n_es
models.append(('KNN', KNeighborsClassifier(weights='distance', n_neighbor
```

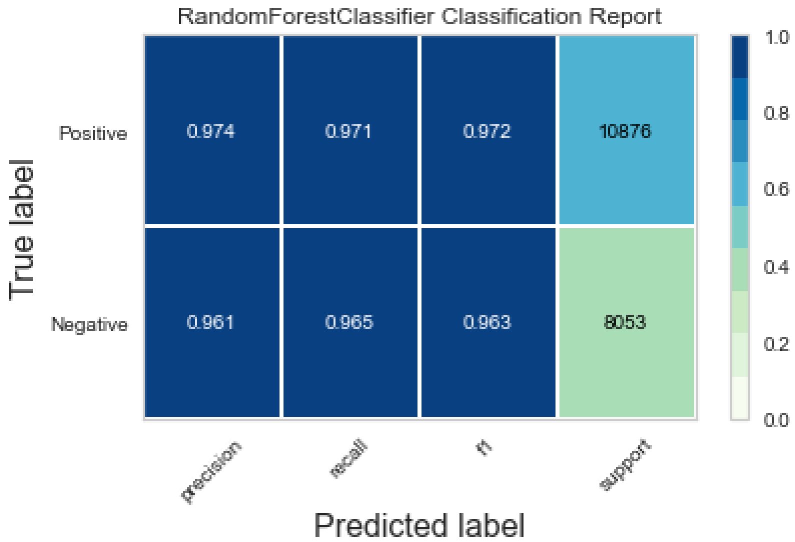
```
# Evaluate each model using cross-validation.
for name, model in models:
    print(f"\n\n\033[1mEvaluating {name}\033[0m")
    y_pred = cross_val_predict(model, X_test, y_test, cv=10)

    # Compute metrics
    acc = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, pos_label=1)
    recall = recall_score(y_test, y_pred, pos_label=1)
    f1 = f1_score(y_test, y_pred, pos_label=1)
    auc_score = roc_auc_score(y_test, y_pred)
    kappa = cohen_kappa_score(y_test, y_pred)

    # Display confusion matrix.
    cm = confusion_matrix(y_test, y_pred, labels=[0,1])
    plt.figure(figsize=(10,6))
    sns.heatmap(cm, annot=True, fmt='g', cmap='GnBu',xticklabels=['Negati
    plt.title(f"Confusion matrix for {name}",fontweight="bold", fontsize=
    plt.xlabel("Predicted label",fontsize=16)
    plt.ylabel("True label",fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.show()

    # Display ROC curve.
    fpr, tpr, thresholds = roc_curve(y_test, y_pred)
    roc_auc = auc(fpr, tpr)
    plt.figure(figsize=(10,6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area =
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate',fontsize=16)
    plt.ylabel('True Positive Rate',fontsize=16)
    plt.title(f"ROC curve for {name}",fontweight="bold", fontsize=20)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.legend(loc="lower right")
    plt.show()

    # Print AUC score.
    print(f"ROC curve (area = {roc_auc:.2f})")

    # Display classification report.
    cm_viz = ClassificationReport(model, support=True, cmap='GnBu', annot
    cm_viz.ax.set_title(f"Classification Report {name}", fontsize=20, for
    cm_viz.ax.set_xlabel("Predicted label", fontsize=16)
    cm_viz.ax.set_ylabel("True label", fontsize=16)
    cm_viz.fit(X_train, y_train)
    cm_viz.score(X_test, y_pred)
    cm_viz.show()

    # Print Kappa score.
    print(f"Kappa score for {name}: {kappa:.2f}")
```

Evaluating RF



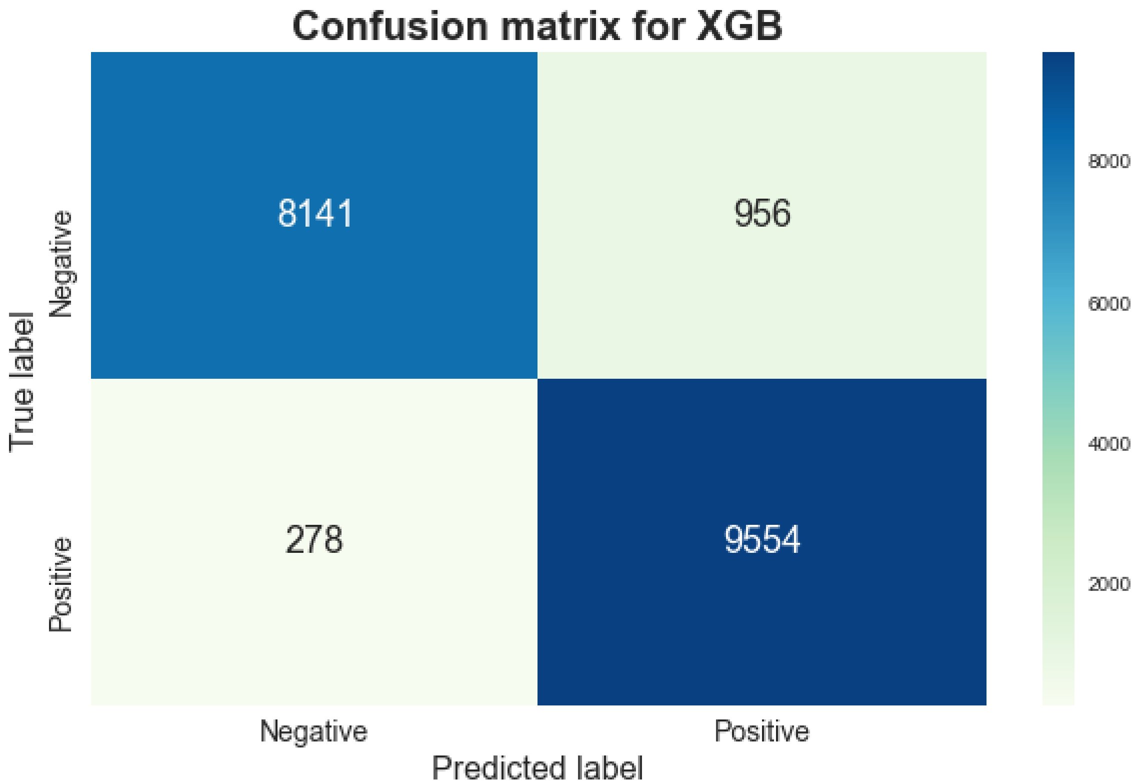


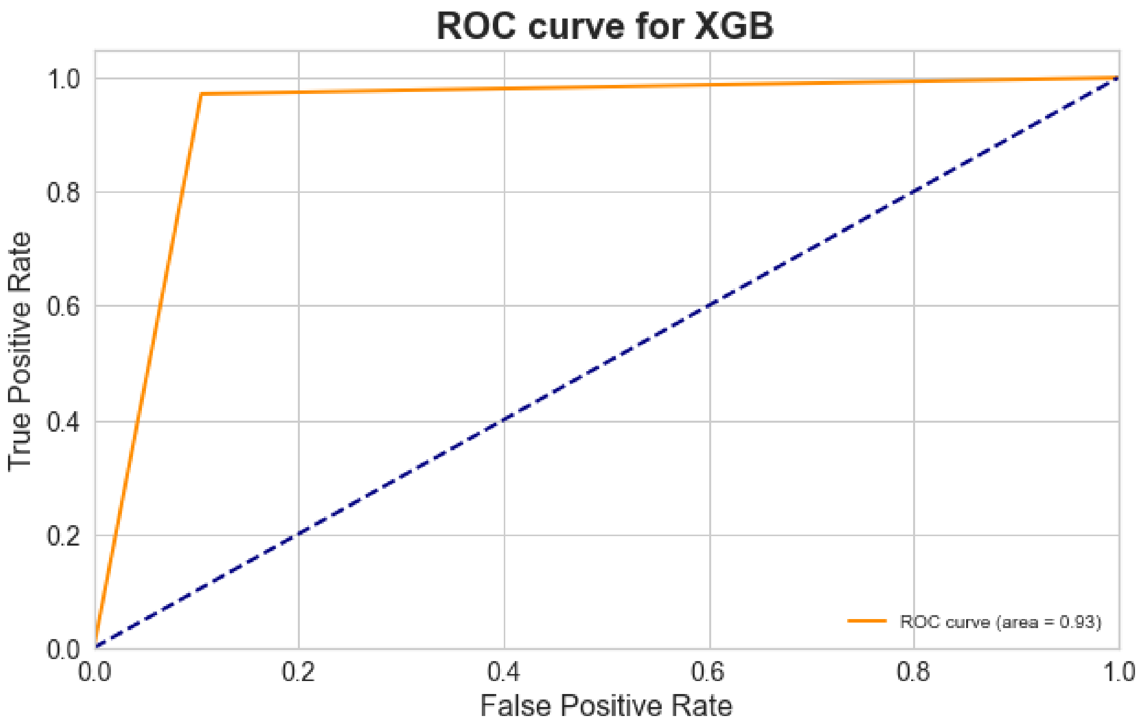

```
ROC curve (area = 0.92)
```



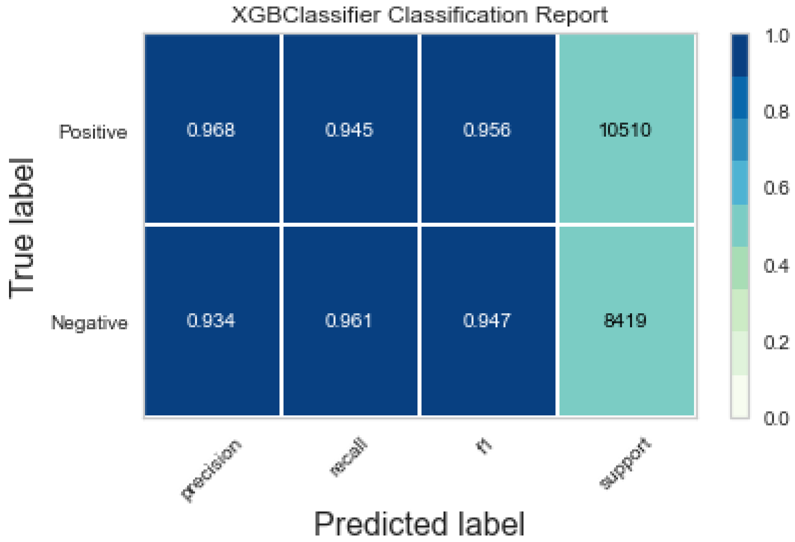

```
Kappa score for RF: 0.85
```

Evaluating XGB



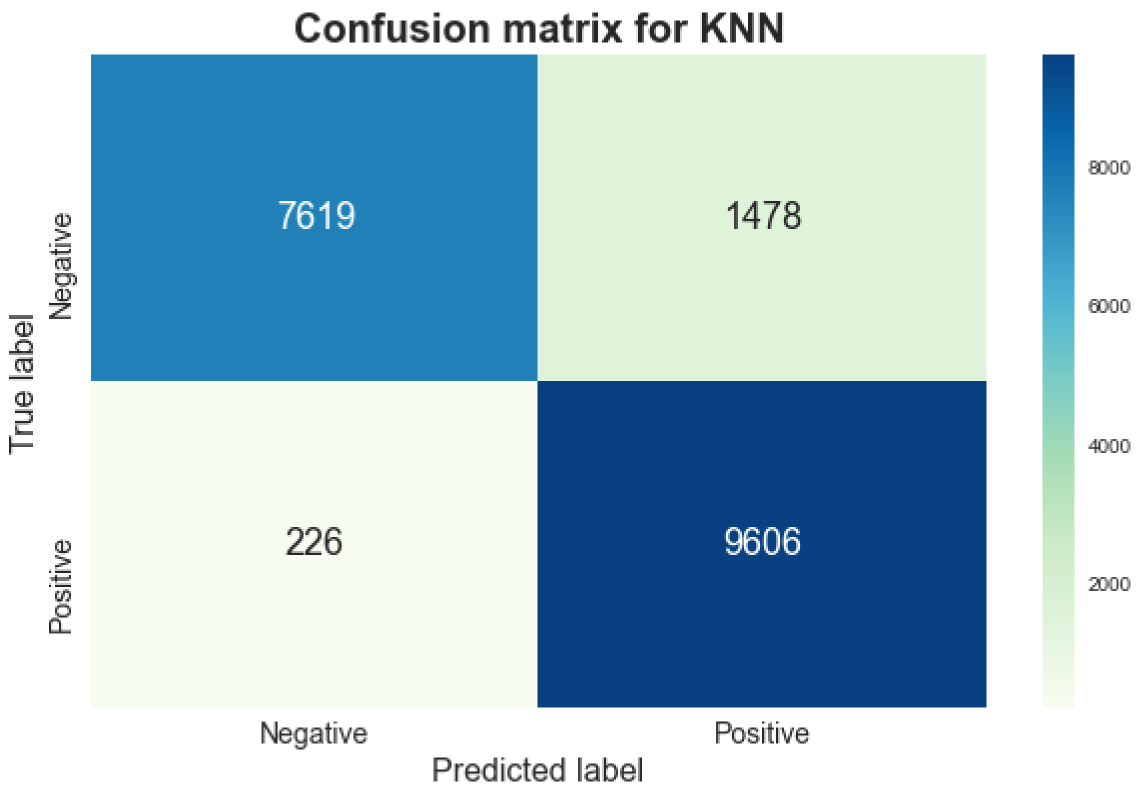


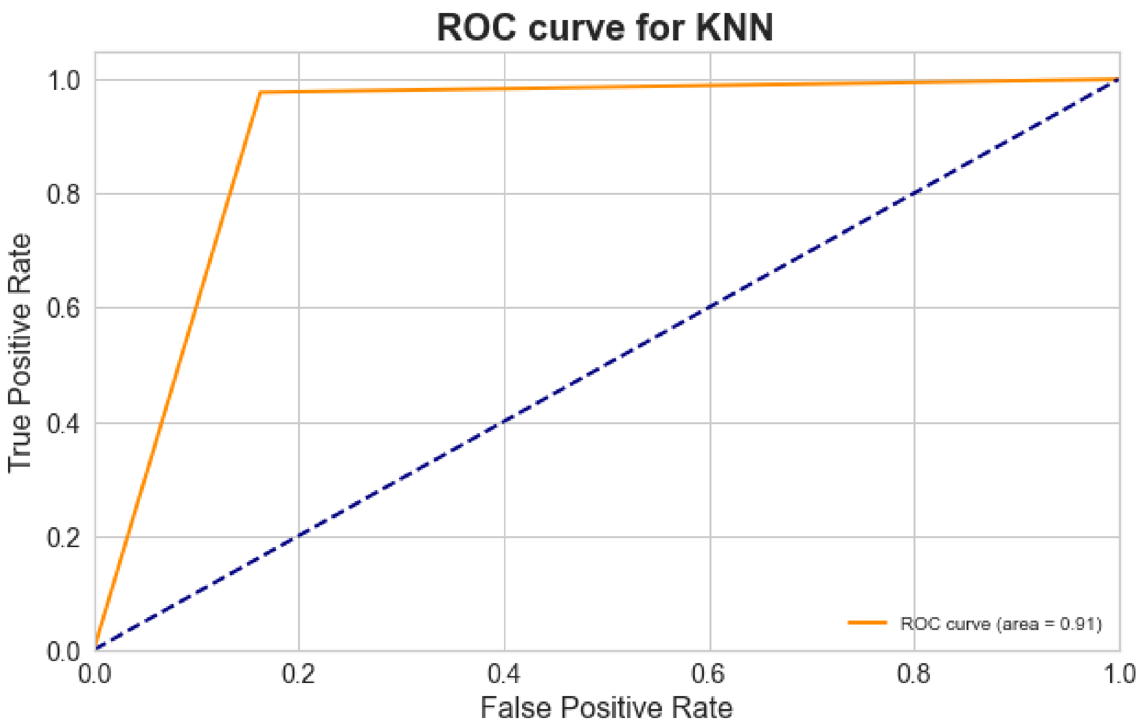

```
ROC curve (area = 0.93)
```



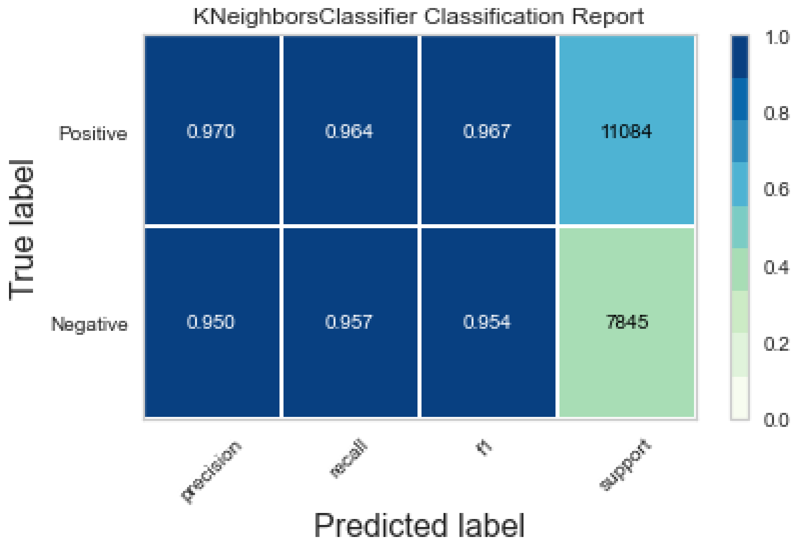

```
Kappa score for XGB: 0.87
```

Evaluating KNN






```
ROC curve (area = 0.91)
```




```
Kappa score for KNN: 0.82
```

In [ ]: