Second IS assignment

Gasper Spagnolo

January 6, 2023

1 Seminar 2: Predicting Biodegradability of Chemical

1.1 1. Introduction

Chemicals are all around us. Studying their properties by the means of machine learning is an active research field; matching molecular patterns with their behavior can be a decisive factor in the creation of new materials, drugs, and more. In this seminar assignment, your task is to explore the data and build machine-learning models that predict the biodegradability of chemicals.

1.2 2. Task

You will work with the data set compiled by Mansouri et al. data. There are 41 features and one target feature (biodegradability). The target variable is encoded as ready biodegradable (1) and not ready biodegradable (2). The data set consists of 1055 instances. Features can be either symbolic or numeric. IMPORTANT: Use the dataset provided on uˇcilnica and NOT the one posted on the link above. It is minimally modified and split into train in test sets.

1.2.1 2.1 Exploration

Inspect the dataset. How balanced is the target variable? Are there any missing values present? If there are, choose a strategy that takes this into account. Most of your data is of the numeric type. Can you identify, by adopting exploratory analysis, whether some features are directly related to the target? What about feature pairs? Produce at least three types of visualizations of the feature space and be prepared to argue why these visualizations were useful for your subsequent analysis.

[1]:
```
# Needed imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
import scikitplot as skplt
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
df_train = pd.read_csv('train.csv')
df_test = pd.read_csv('test.csv')
```

Lets inspect training and test data

[3]:
```
df_test.head()
```

[3]:
```
       V1      V2  V3  V4  V5  V6  V7    V8  V9  V10  …  V33  V34  V35  \
1   3.919  2.6909   0   0   0   0   0  31.4   2    0  …    0    0    0
2   4.170  2.1144   0   0   0   0   0  30.8   1    1  …    0    0    0
4   3.000  2.7098   0   0   0   0   0  20.0   0    2  …    0    0    1
13  4.214  2.6272   0   0   0   0   0  30.0   3    0  …    0    0    0
16  3.942  2.7719   1   0   0   0   0  31.6   2    0  …    0    0    0

      V36    V37  V38    V39  V40  V41  Class
1   2.949  1.591    0  7.253    0    0      2
2   3.315  1.967    0  7.257    0    0      2
4   3.046  5.000    0  6.690    0    0      2
13  2.998  1.722    0  6.770    0    0      2
16  3.542  1.739    0  8.127    0    1      2

[5 rows x 42 columns]
```

[4]:
```
df_train.describe()
```

[4]:
```
               V1          V2          V3          V4          V5          V6  \
count  846.000000  846.000000  846.000000  821.000000  846.000000  846.000000
mean     4.790476    3.054551    0.739953    0.030451    0.946809    0.277778
std      0.531991    0.813983    1.504545    0.198281    2.318081    1.045544
min      2.000000    0.803900    0.000000    0.000000    0.000000    0.000000
25%      4.499000    2.510175    0.000000    0.000000    0.000000    0.000000
50%      4.840000    3.052400    0.000000    0.000000    0.000000    0.000000
75%      5.119000    3.415725    1.000000    0.000000    1.000000    0.000000
max      6.496000    7.918400   12.000000    2.000000   36.000000   13.000000

               V7          V8          V9         V10  …         V33  \
count  846.000000  846.000000  846.000000  846.000000  …  846.000000
mean     1.669031   37.422813    1.342790    1.784870  …    0.903073
std      2.220221    9.030008    2.018433    1.773856  …    1.526124
min      0.000000    9.100000    0.000000    0.000000  …    0.000000
25%      0.000000   30.800000    0.000000    0.000000  …    0.000000
50%      1.000000   37.850000    1.000000    1.500000  …    0.000000
75%      3.000000   43.800000    2.000000    3.000000  …    1.000000
max     18.000000   60.700000   24.000000   12.000000  …   12.000000

              V34         V35         V36         V37         V38         V39  \
count  846.000000  846.000000  846.000000  821.000000  846.000000  846.000000
mean     1.241135    0.926714    3.922100    2.549406    0.671395    8.643191
std      2.248684    1.239133    0.992636    0.625021    1.093633    1.223700
min      0.000000    0.000000    2.279000    1.467000    0.000000    4.948000
25%      0.000000    0.000000    3.497000    2.101000    0.000000    8.009500
50%      0.000000    1.000000    3.732500    2.461000    0.000000    8.508000
```

```
75%       2.000000    1.000000    3.980000    2.861000    1.000000    9.019750
max      18.000000    7.000000   10.695000    5.750000    8.000000   14.700000

              V40         V41       Class
count  846.000000  846.000000  846.000000
mean     0.059102    0.706856    1.333333
std      0.342364    2.145396    0.471683
min      0.000000    0.000000    1.000000
25%      0.000000    0.000000    1.000000
50%      0.000000    0.000000    1.000000
75%      0.000000    0.000000    2.000000
max      4.000000   27.000000    2.000000

[8 rows x 42 columns]
```

[5]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 846 entries, 3 to 1055
Data columns (total 42 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      846 non-null    float64
 1   V2      846 non-null    float64
 2   V3      846 non-null    int64
 3   V4      821 non-null    float64
 4   V5      846 non-null    int64
 5   V6      846 non-null    int64
 6   V7      846 non-null    int64
 7   V8      846 non-null    float64
 8   V9      846 non-null    int64
 9   V10     846 non-null    int64
 10  V11     846 non-null    int64
 11  V12     846 non-null    float64
 12  V13     846 non-null    float64
 13  V14     846 non-null    float64
 14  V15     846 non-null    float64
 15  V16     846 non-null    int64
 16  V17     846 non-null    float64
 17  V18     846 non-null    float64
 18  V19     846 non-null    int64
 19  V20     846 non-null    int64
 20  V21     846 non-null    int64
 21  V22     830 non-null    float64
 22  V23     846 non-null    int64
 23  V24     846 non-null    int64
 24  V25     846 non-null    int64
 25  V26     846 non-null    int64
```

```
 26  V27     838 non-null    float64
 27  V28     846 non-null    float64
 28  V29     838 non-null    float64
 29  V30     846 non-null    float64
 30  V31     846 non-null    float64
 31  V32     846 non-null    int64
 32  V33     846 non-null    int64
 33  V34     846 non-null    int64
 34  V35     846 non-null    int64
 35  V36     846 non-null    float64
 36  V37     821 non-null    float64
 37  V38     846 non-null    int64
 38  V39     846 non-null    float64
 39  V40     846 non-null    int64
 40  V41     846 non-null    int64
 41  Class   846 non-null    int64
dtypes: float64(19), int64(23)
memory usage: 284.2 KB
```

[6]:
```
df_test.head()
```

[6]:
```
       V1      V2  V3  V4  V5  V6  V7    V8  V9  V10  …  V33  V34  V35  \
1   3.919  2.6909   0   0   0   0   0  31.4   2    0  …    0    0    0
2   4.170  2.1144   0   0   0   0   0  30.8   1    1  …    0    0    0
4   3.000  2.7098   0   0   0   0   0  20.0   0    2  …    0    0    1
13  4.214  2.6272   0   0   0   0   0  30.0   3    0  …    0    0    0
16  3.942  2.7719   1   0   0   0   0  31.6   2    0  …    0    0    0

      V36    V37  V38    V39  V40  V41  Class
1   2.949  1.591    0  7.253    0    0      2
2   3.315  1.967    0  7.257    0    0      2
4   3.046  5.000    0  6.690    0    0      2
13  2.998  1.722    0  6.770    0    0      2
16  3.542  1.739    0  8.127    0    1      2

[5 rows x 42 columns]
```

[7]:
```
df_test.describe()
```

[7]:
```
               V1          V2         V3          V4          V5          V6  \
count  209.000000  209.000000  209.00000  209.000000  209.000000  209.000000
mean     4.750938    3.130050    0.62201    0.086124    1.114833    0.339713
std      0.603914    0.897556    1.27690    0.406969    2.393143    1.182566
min      2.000000    1.134900    0.00000    0.000000    0.000000    0.000000
25%      4.414000    2.494500    0.00000    0.000000    0.000000    0.000000
50%      4.807000    3.039300    0.00000    0.000000    0.000000    0.000000
75%      5.188000    3.555400    1.00000    0.000000    1.000000    0.000000
max      6.253000    9.177500    8.00000    3.000000   16.000000   12.000000
```

```
              V7          V8          V9         V10  …         V33  \
count  209.000000  209.000000  209.000000  209.000000  …  209.000000
mean     1.555024   35.569378    1.511962    1.880383  …    0.803828
std      2.246383    9.471334    1.721220    1.784023  …    1.498327
min      0.000000    0.000000    0.000000    0.000000  …    0.000000
25%      0.000000   29.400000    0.000000    0.000000  …    0.000000
50%      0.000000   34.200000    1.000000    2.000000  …    0.000000
75%      3.000000   41.200000    2.000000    3.000000  …    1.000000
max     14.000000   60.000000    9.000000   11.000000  …   12.000000

              V34         V35         V36         V37         V38         V39  \
count  209.000000  209.000000  209.000000  209.000000  209.000000  209.000000
mean     1.411483    1.100478    3.902612    2.629201    0.746411    8.574038
std      2.374355    1.320857    1.029605    0.714285    1.077657    1.315016
min      0.000000    0.000000    2.267000    1.576000    0.000000    4.917000
25%      0.000000    0.000000    3.401000    2.146000    0.000000    7.872000
50%      0.000000    1.000000    3.694000    2.469000    0.000000    8.464000
75%      2.000000    2.000000    3.991000    2.967000    1.000000    9.017000
max     18.000000    6.000000   10.355000    5.825000    6.000000   14.030000

              V40         V41       Class
count  209.000000  209.000000  209.000000
mean     0.019139    0.789474    1.354067
std      0.195176    2.589491    0.479378
min      0.000000    0.000000    1.000000
25%      0.000000    0.000000    1.000000
50%      0.000000    0.000000    1.000000
75%      0.000000    0.000000    2.000000
max      2.000000   27.000000    2.000000

[8 rows x 42 columns]
```

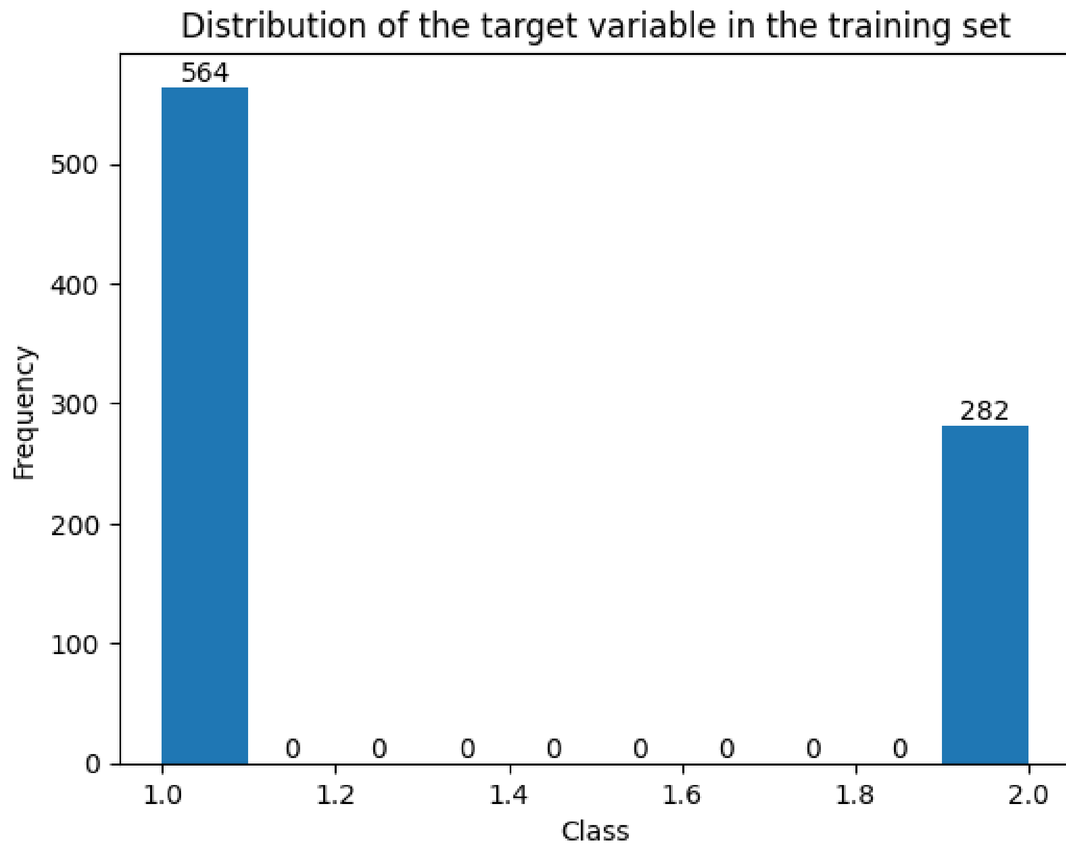
[8]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 209 entries, 1 to 1051
Data columns (total 42 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   V1      209 non-null    float64
 1   V2      209 non-null    float64
 2   V3      209 non-null    int64
 3   V4      209 non-null    int64
 4   V5      209 non-null    int64
 5   V6      209 non-null    int64
 6   V7      209 non-null    int64
 7   V8      209 non-null    float64
```

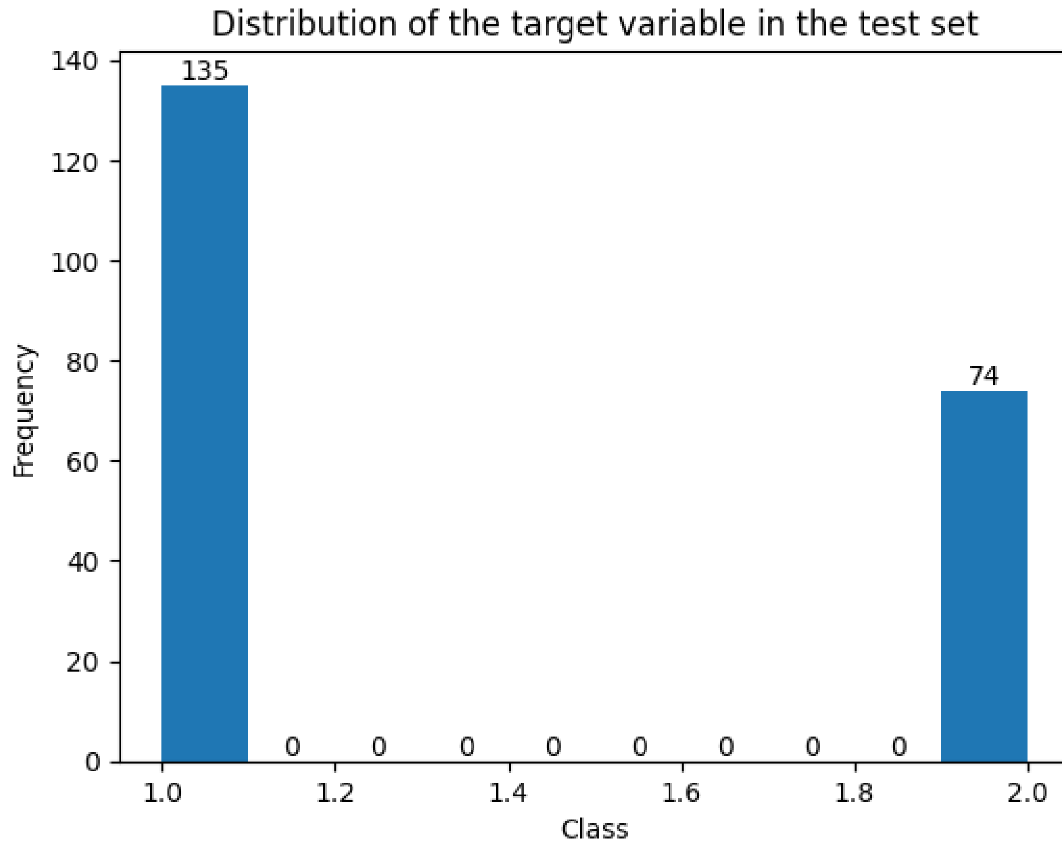
```
 8   V9      209 non-null    int64  
 9   V10     209 non-null    int64  
 10  V11     209 non-null    int64  
 11  V12     209 non-null    float64
 12  V13     209 non-null    float64
 13  V14     209 non-null    float64
 14  V15     209 non-null    float64
 15  V16     209 non-null    int64  
 16  V17     209 non-null    float64
 17  V18     209 non-null    float64
 18  V19     209 non-null    int64  
 19  V20     209 non-null    int64  
 20  V21     209 non-null    int64  
 21  V22     209 non-null    float64
 22  V23     209 non-null    int64  
 23  V24     209 non-null    int64  
 24  V25     209 non-null    int64  
 25  V26     209 non-null    int64  
 26  V27     209 non-null    float64
 27  V28     209 non-null    float64
 28  V29     209 non-null    int64  
 29  V30     209 non-null    float64
 30  V31     209 non-null    float64
 31  V32     209 non-null    int64  
 32  V33     209 non-null    int64  
 33  V34     209 non-null    int64  
 34  V35     209 non-null    int64  
 35  V36     209 non-null    float64
 36  V37     209 non-null    float64
 37  V38     209 non-null    int64  
 38  V39     209 non-null    float64
 39  V40     209 non-null    int64  
 40  V41     209 non-null    int64  
 41  Class   209 non-null    int64  
dtypes: float64(17), int64(25)
memory usage: 70.2 KB
```

Display distributions of target variable Class in training and validation set.

[9]:
```
_, _, bars = plt.hist(df_train['Class'], bins=10)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Distribution of the target variable in the training set')
plt.bar_label(bars, fmt='%1.0f')
plt.show()
```



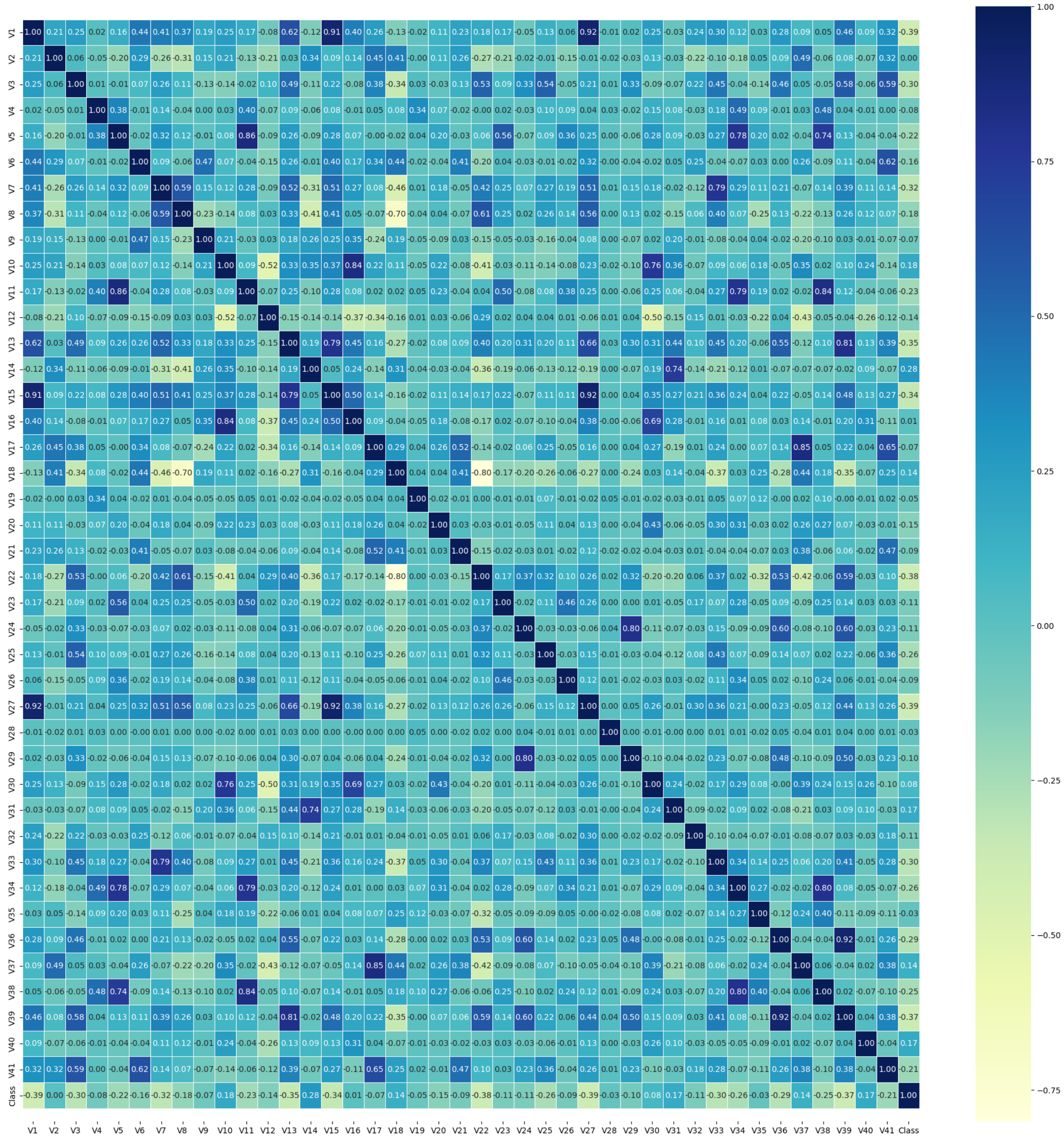
[10]:
```
_, _, bars = plt.hist(df_test['Class'], bins=10)
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.title('Distribution of the target variable in the test set')
plt.bar_label(bars, fmt='%1.0f')
plt.show()
```



Display relationship between features in the training set using the correlation matrix

[11]:
```
correlation_matrix = df_train.corr()
fig, ax = plt.subplots(figsize=(25, 25))

ax = sns.heatmap(
    correlation_matrix,
    annot=True,
    linewidths=0.5,
    fmt=".2f",
    cmap="YlGnBu"
)

# Jupyter notebook specific
bottom_side, top_side = ax.get_ylim()
ax.set_ylim(bottom_side + 0.5, top_side - 0.5)
```

[11]:
```
(42.5, -0.5)
```



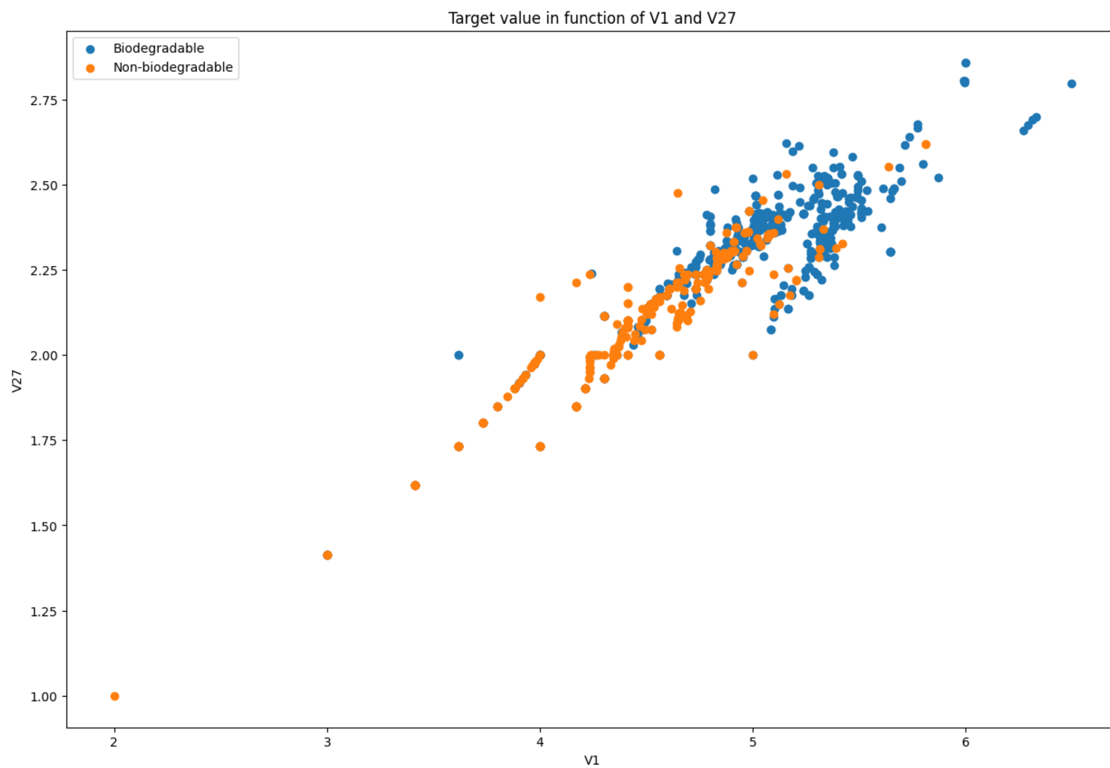
We can see that there is the highest positive correlation in **V14** atribute and the highest negative value in the attributes **V1, V27** So lets see the distribution of those values in comparrison to class.

V1 vs V27

[12]:
```
plt.figure(figsize=(15, 10))

# Scatter with 1 values of target class
plt.scatter(
    df_train['V1'][df_train['Class'] == 1],
    df_train['V27'][df_train['Class'] == 1],
```

```
)

# Scatter with  2 values of target class
plt.scatter(
    df_train['V1'][df_train['Class'] == 2],
    df_train['V27'][df_train['Class'] == 2],
)

plt.title('Target value in function of V1 and V27')

plt.xlabel('V1')
plt.ylabel('V27')
plt.legend(['Biodegradable', 'Non-biodegradable'])
```

[12]: <matplotlib.legend.Legend at 0x7f0852a7dcd0>

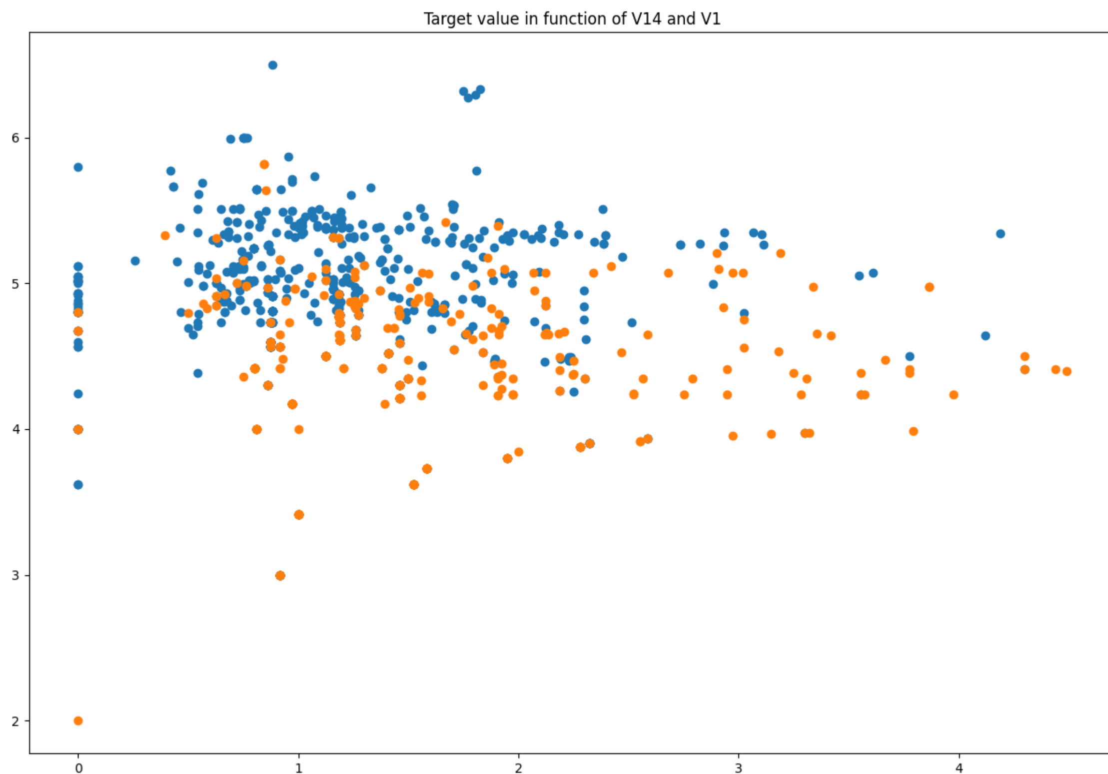


V14 vs V1

[13]:
```
# V14 vs V1
plt.figure(figsize=(15, 10))

# Scatter with 1 values of target class
plt.scatter(
```

```
    df_train['V14'][df_train['Class'] == 1],
    df_train['V1'][df_train['Class'] == 1],
)

# Scatter with  2 values of target class
plt.scatter(
    df_train['V14'][df_train['Class'] == 2],
    df_train['V1'][df_train['Class'] == 2],
)


plt.title('Target value in function of V14 and V1')
```

[13]: Text(0.5, 1.0, 'Target value in function of V14 and V1')

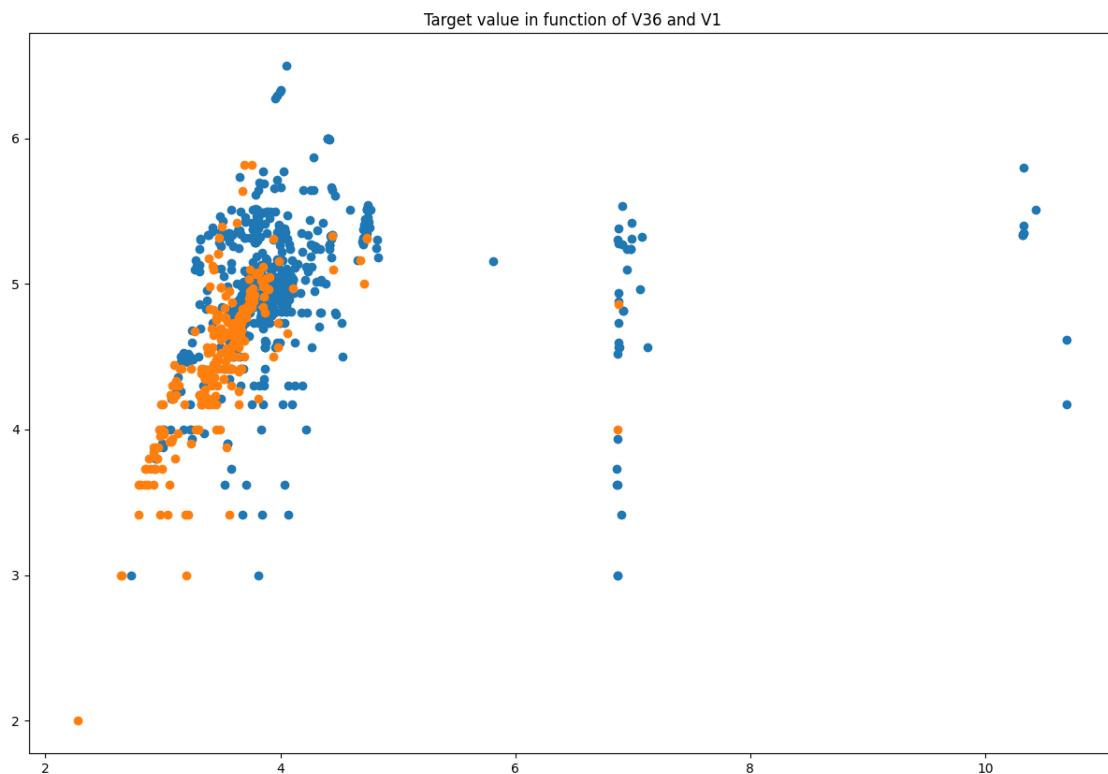


V36 vs V1

[14]:
```
# V36 vs v1
plt.figure(figsize=(15, 10))

# Scatter with 1 values of target class
plt.scatter(
    df_train['V36'][df_train['Class'] == 1],
```

```
    df_train['V1'][df_train['Class'] == 1],
)

# Scatter with  2 values of target class
plt.scatter(
    df_train['V36'][df_train['Class'] == 2],
    df_train['V1'][df_train['Class'] == 2],
)

plt.title('Target value in function of V36 and V1')
```

[14]: Text(0.5, 1.0, 'Target value in function of V36 and V1')



[15]:
```
# Spliting the data into features and labels
X_train = df_train.drop('Class', axis=1)
y_train = df_train['Class']
X_test = df_test.drop('Class', axis=1)
y_test = df_test['Class']
```

[16]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

```python
# Put models in a dictionary
models = {
    "Logistic Regression": LogisticRegression(),
    "KNN": KNeighborsClassifier(),
    "Random Forest": RandomForestClassifier()
}

# Create a function to fit and score models
def fit_and_score(models, X_train, X_test, y_train, y_test):
    """
    Fits and evaluates given machine learning models.
    models: dict of different Scikit-Learn machine learning models
    X_train: training data (no labels)
    x_test: testing data (no labels)
    y_train: training labels
    y_test: trest labels
    """

    # Set random seed
    np.random.seed(42)

    # Make a dictioanry to keep model scores
    model_scores = {}

    # Loop through models
    for name, model in models.items():
        # Fit the model to the data
        model.fit(X_train, y_train)
        # Evaluate the model and append its score to model_scores
        model_scores[name] = model.score(X_test, y_test)

    return model_scores
```

Check if there are any missing values

[17]:
```python
na_counts = df_train.isna().sum()
na_counts[na_counts > 0]
```

[17]:
```
V4     25
V22    16
V27     8
V29     8
V37    25
dtype: int64
```

We can see that there are five atributes that have missing values. Lets inspect them.

V4

[18]:
```
df_train['V4'].describe()
```

[18]:
```
count    821.000000
mean       0.030451
std        0.198281
min        0.000000
25%        0.000000
50%        0.000000
75%        0.000000
max        2.000000
Name: V4, dtype: float64
```

[19]:
```
df_train['V4'].value_counts()
```

[19]:
```
0.0    800
1.0     17
2.0      4
Name: V4, dtype: int64
```

We can see that the majority of entires in that particular atribute are zeros. So I think that it would be best if I set all the `Nan` values to zeros.

[20]:
```
df_train['V4'].fillna(0, inplace=True)
df_test['V4'].fillna(0, inplace=True)
```

V22

[21]:
```
df_train['V22'].describe()
```

[21]:
```
count    830.000000
mean       1.243898
std        0.094109
min        0.898000
25%        1.187500
50%        1.248500
75%        1.298750
max        1.641000
Name: V22, dtype: float64
```

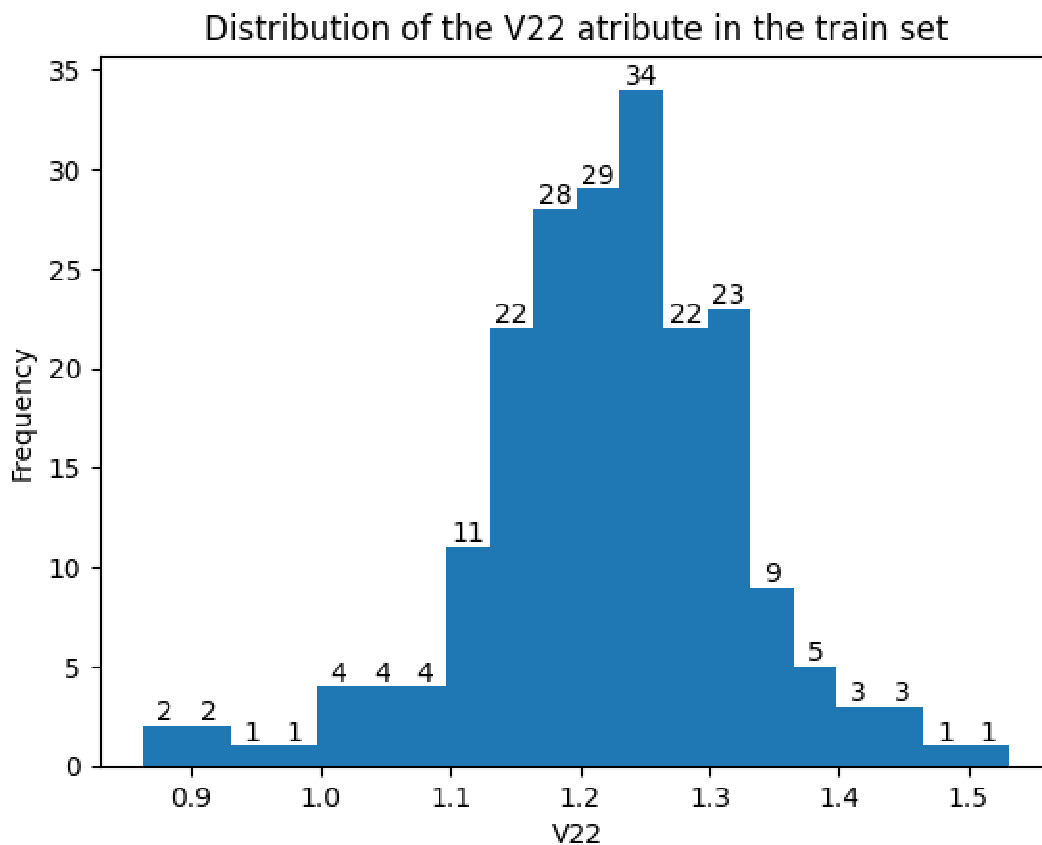
[22]:
```
df_train['V22'].value_counts()
```

[22]:
```
1.299    9
1.280    9
1.296    8
1.254    8
1.264    8
        ..
1.449    1
```

```
1.159    1
1.363    1
1.331    1
1.410    1
Name: V22, Length: 321, dtype: int64
```

[23]:

```
_, _, bars = plt.hist(df_test['V22'], bins=20)
plt.xlabel('V22')
plt.ylabel('Frequency')
plt.title('Distribution of the V22 atribute in the train set')
plt.bar_label(bars, fmt='%1.0f')
plt.show()
```



The distribution of the target variable **V22** is normal, so i could try to fill the missing values with `mean()`.

[24]:

```
df_train['V22'].fillna(df_train['V22'].mean(), inplace=True)
df_test['V22'].fillna(df_test['V22'].mean(), inplace=True)
```

V27

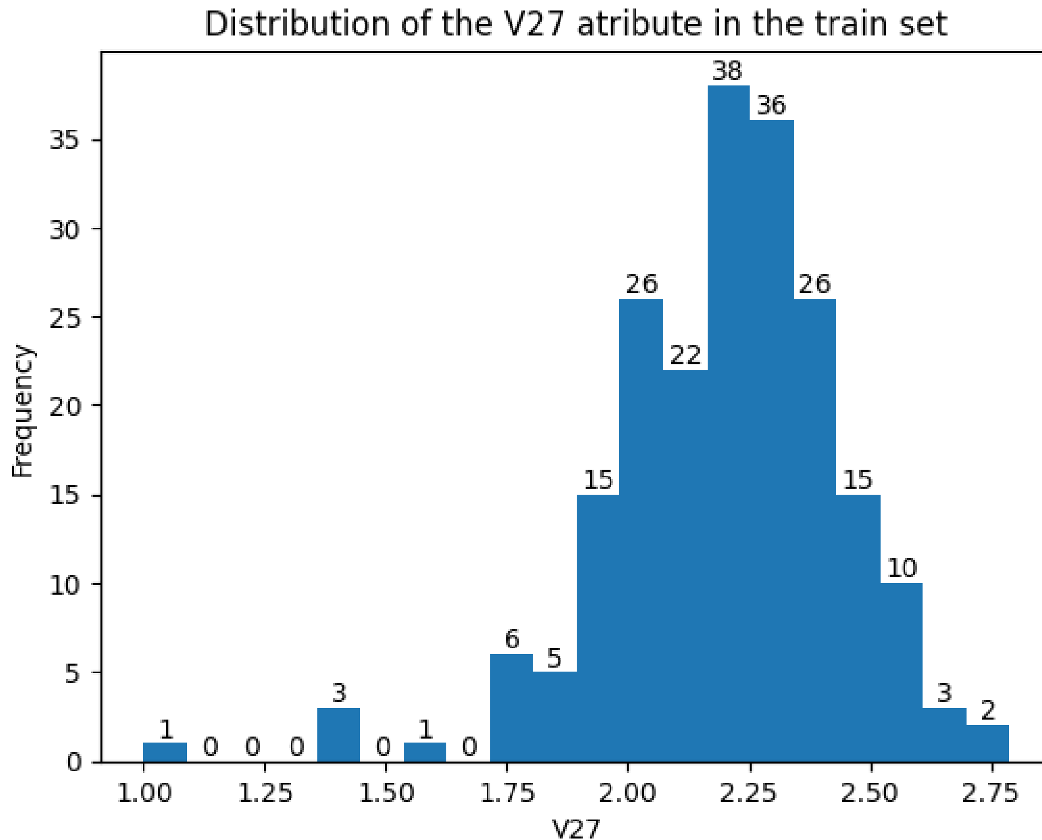
[25]:
```
df_train['V27'].describe()
```

[25]:
```
count    838.000000
mean       2.218153
std        0.221545
min        1.000000
25%        2.107000
50%        2.251000
75%        2.359750
max        2.859000
Name: V27, dtype: float64
```

[26]:
```
df_train['V27'].value_counts()
```

[26]:
```
2.000    36
2.236    31
2.194    24
1.848    22
2.175    21
         ..
2.294     1
2.466     1
2.488     1
2.372     1
2.622     1
Name: V27, Length: 290, dtype: int64
```

[27]:
```
_, _, bars = plt.hist(df_test['V27'], bins=20)
plt.xlabel('V27')
plt.ylabel('Frequency')
plt.title('Distribution of the V27 atribute in the train set')
plt.bar_label(bars, fmt='%1.0f')
plt.show()
```



The distribution of the target variable **V27** is normal, so i could try to fill the missing values with `mean()`.

[28]:
```
# Set the nan values to the mean of the column
df_train['V27'].fillna(df_train['V27'].mean(), inplace=True)
df_test['V27'].fillna(df_test['V27'].mean(), inplace=True)
```

V29

[29]:
```
df_train['V29'].describe()
```

[29]:
```
count    838.00000
mean       0.02506
std        0.15640
min        0.00000
25%        0.00000
50%        0.00000
75%        0.00000
max        1.00000
Name: V29, dtype: float64
```

[30]:
```
df_train['V29'].value_counts()
```

[30]:
```
0.0    817
1.0     21
Name: V29, dtype: int64
```

We can see that the majority of entires in that particular atribute are zeros. So I think that it would be best if I set all the Nan values to zeros.

[31]:
```
# Set nan values to 0
df_train['V29'].fillna(0, inplace=True)
df_test['V29'].fillna(0, inplace=True)
```

V37

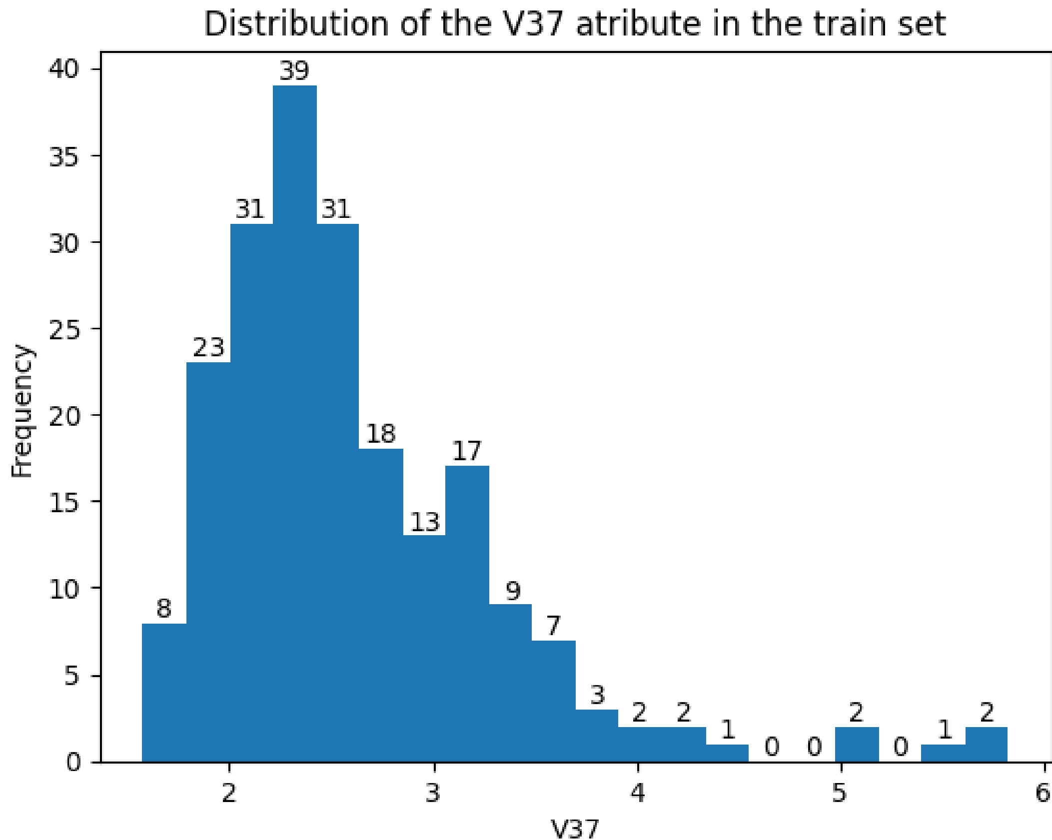
[32]:
```
df_train['V37'].describe()
```

[32]:
```
count    821.000000
mean       2.549406
std        0.625021
min        1.467000
25%        2.101000
50%        2.461000
75%        2.861000
max        5.750000
Name: V37, dtype: float64
```

[33]:
```
df_train['V37'].value_counts()
```

[33]:
```
2.167    9
2.500    9
2.833    8
2.667    8
1.833    7
        ..
2.029    1
1.886    1
2.089    1
2.197    1
2.206    1
Name: V37, Length: 535, dtype: int64
```

[34]:
```
_, _, bars = plt.hist(df_test['V37'], bins=20)
plt.xlabel('V37')
plt.ylabel('Frequency')
plt.title('Distribution of the V37 atribute in the train set')
plt.bar_label(bars, fmt='%1.0f')
plt.show()
```



The distribution of the target variable **V37** is normal, so i could try to fill the missing values with `mean()`.

[35]:
```
df_train['V37'].fillna(df_train['V37'].mean(), inplace=True)
df_test['V37'].fillna(df_test['V37'].mean(), inplace=True)
```

1.2.2 2.2 Modeling

Besides the baselines (majority classifier, random classifier), use at least three machine learning algorithms to model the target class. Be ready to argue why did you select specific algorithms and how did you find the best hyperparameters for them. Consider the following points when creating your models: - Create your models using all features and subsets of them using various feature selection techniques. - Certain models assume that data follows a particular distribution or may work better with other types of variables (e.g., categorical instead of numeric). Explore whether you can come up with feature transformations that are more appropriate for your models. Try to construct new features from existing ones. Try to explain the results and performance of different models.

[36]:
```
# Spliting the data into features and labels
X_train = df_train.drop('Class', axis=1).reset_index(drop=True)
```

```
y_train = df_train['Class'].reset_index(drop=True)
X_test = df_test.drop('Class', axis=1).reset_index(drop=True)
y_test = df_test['Class'].reset_index(drop=True)
```

Using majority classifier and random classifier

Majority classifier

[37]:
```
# Get the reatio between thhe class we are trying to predict
y_train.value_counts(normalize=True)
```

[37]:
```
1    0.666667
2    0.333333
Name: Class, dtype: float64
```

If we were to predict using the majority classifier then we would always predict Ready non-biodegradable.

[38]:
```
y_test[y_test == 1].shape[0] / y_test.shape[0]
```

[38]:
```
0.645933014354067
```

We would get the accuracy of 0.645933014354067 if we predicted all the values to be 1.

Random classifier We have two classes to predict, so probability of predicting the right class is 50%.

Lets firstly write a simple function that will score all our generated models

[39]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import KFold, RepeatedKFold
all_scores = []

def score_the_model(model, model_name, random_seed, X_train, X_test, y_train,␣
 ↪y_test, plot=False):
    """
    Fits and evaluates given machine learning models.
    models: dict of different Scikit-Learn machine learning models
    X_train: training data (no labels)
    x_test: testing data (no labels)
    y_train: training labels
    y_test: trest labels
    """

    # Set random seed
```

```
    np.random.seed(random_seed)

    # Fit the model to the data
    model.fit(X_train, y_train)

    model_score = model.score(X_test, y_test) # Mean accuracy of ``self.
↪predict(X)`` wrt. `y`.
    # Predict the labels
    y_pred = model.predict(X_test)

    # Compute scores
    f1 = f1_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred)
    # Plot scores
    normal_scores = {
        'Accuracy': model_score,
        'F1': f1,
        'Precision': precision,
        'Recall': recall,
        'AUC': auc
    }

    def normal_cv(model, X_train, y_train, random_seed):
        # Perform normal cross-validation
        X_train = X_train.copy()
        y_train = y_train.copy()
        kfold = KFold(n_splits=5, shuffle=True, random_state=random_seed)
        scores = []

        for train_ix, test_ix in kfold.split(X_train):
            # Split the data
            X_train_cv, X_test_cv = X_train.iloc[train_ix], X_train.
↪iloc[test_ix]
            y_train_cv, y_test_cv = y_train.iloc[train_ix], y_train.
↪iloc[test_ix]

            # Fit the model
            model.fit(X_train_cv, y_train_cv)

            # Evaluate the model
            y_pred = model.predict(X_test_cv)
            scrs = {
                'Accuracy': model.score(X_test_cv, y_test_cv),
                'F1': f1_score(y_test_cv, y_pred),
                'Precision': precision_score(y_test_cv, y_pred),
```

```
                'Recall': recall_score(y_test_cv, y_pred),
                'AUC': roc_auc_score(y_test_cv, y_pred)
            }
            scores.append(scrs)

        # Plot all the scores
        scores = pd.DataFrame(scores)
        scores.plot(kind='bar', figsize=(10, 8))
        # Plot also the values at the top of the bars
        plt.title(f'Cross-validated scores for {model_name}')
        plt.xlabel('Fold')
        plt.ylabel('Score')
        plt.legend(loc='lower right')
        plt.show()
        return scores
    if type(X_train) == pd.core.frame.DataFrame:
        scores_cv = normal_cv(model, X_train, y_train, random_seed)

    def repeated_cv(model, X_train, y_train, random_seed):
        # Perform another cv with 10 folds
        scores_k_fold = []
        rkf = RepeatedKFold(n_splits=10, n_repeats=10, random_state=random_seed)
        for train_index, test_index in rkf.split(X_train):
            model.fit(X_train.iloc[train_index], y_train.iloc[train_index])
            y_pred = model.predict(X_train.iloc[test_index])
            scrs = {
                'Accuracy': model.score(X_train.iloc[test_index], y_train.
↪iloc[test_index]),
                'F1': f1_score(y_train.iloc[test_index], y_pred),
                'Precision': precision_score(y_train.iloc[test_index], y_pred),
                'Recall': recall_score(y_train.iloc[test_index], y_pred),
                'AUC': roc_auc_score(y_train.iloc[test_index], y_pred)
            }

            scores_k_fold.append(scrs)
        return scores_k_fold

    k_fold_scores_mean = {}
    k_fold_scores_std = {}

    if type(X_train) == pd.core.frame.DataFrame:
        scores_k_fold = repeated_cv(model, X_train, y_train, random_seed)
        k_fold_scores_mean['acccuracy_mean'] = np.mean([score['Accuracy'] for␣
↪score in scores_k_fold])
        k_fold_scores_std['accuracy_std'] = np.std([score['Accuracy'] for score␣
↪in scores_k_fold])
```

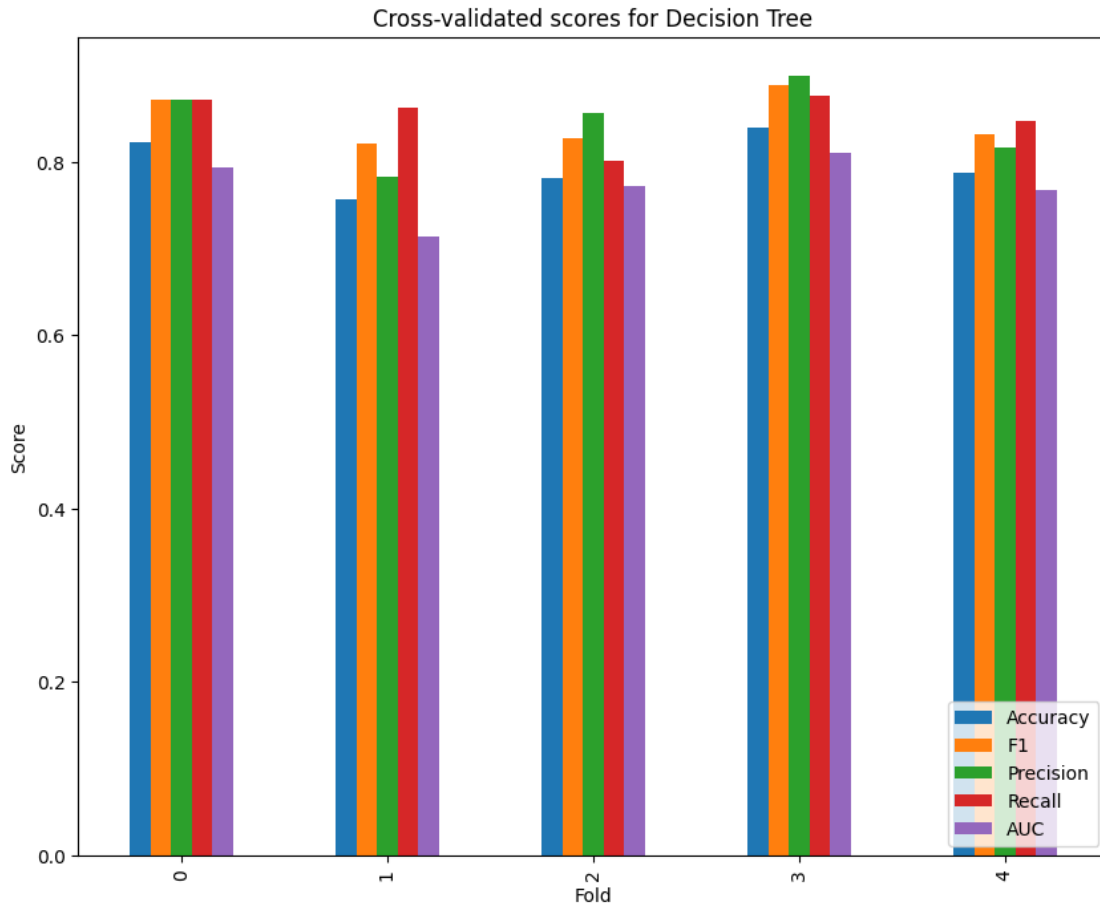
```
        k_fold_scores_mean['f1_mean'] = np.mean([score['F1'] for score in␣
↪scores_k_fold])
        k_fold_scores_std['f1_std'] = np.std([score['F1'] for score in␣
↪scores_k_fold])
        k_fold_scores_mean['precision_mean'] = np.mean([score['Precision'] for␣
↪score in scores_k_fold])
        k_fold_scores_std['precision_std'] = np.std([score['Precision'] for␣
↪score in scores_k_fold])
        k_fold_scores_mean['recall_mean'] = np.mean([score['Recall'] for score␣
↪in scores_k_fold])
        k_fold_scores_std['recall_std'] = np.std([score['Recall'] for score in␣
↪scores_k_fold])
        k_fold_scores_mean['auc_mean'] = np.mean([score['AUC'] for score in␣
↪scores_k_fold])
        k_fold_scores_std['auc_std'] = np.std([score['AUC'] for score in␣
↪scores_k_fold])

    if plot:
        # Plot scores
        fig, ax = plt.subplots(nrows=3, ncols=2, figsize=(15,15))

        # Plot the bar chart of Normal cv scores in the first subplot
        ax[0, 0].bar(normal_scores.keys(), normal_scores.values())
        # Display values of the bars
        for i, v in enumerate(normal_scores.values()):
            ax[0, 0].text(i-0.1, v+0.01, str(round(v, 2)))
        ax[0, 0].set_title(f'Default scoring of {model_name}')
        ax[0, 0].set_ylabel('Score')

        # Plot the k-fold cv scores in the third subplot
        ax[0, 1].bar(k_fold_scores_mean.keys(), k_fold_scores_mean.values())
        # Display values of the bars
        for i, v in enumerate(k_fold_scores_mean.values()):
            ax[0, 1].text(i-0.1, v+0.01, str(round(v, 2)))
        ax[0, 1].set_title(f'10-fold cross-validated scoring of {model_name}␣
↪(mean)')

        # Plot the k-fold cv scores in the third subplot
        ax[1, 0].bar(k_fold_scores_std.keys(), k_fold_scores_std.values())
        ax[1, 0].set_title(f'10-fold cross-validated scoring of {model_name}␣
↪(std)')


        # Plot the ROC curve in the second subplot
        f = RocCurveDisplay.from_estimator(model, X_test, y_test).plot(ax=ax[1,␣
↪1])
```

```
        # Plot the confusion matrix in the third subplot
        cm = confusion_matrix(y_test, y_pred, labels=model.classes_)
        cm_plt = ConfusionMatrixDisplay(cm, display_labels=model.classes_).
↪plot(ax=ax[2, 0])

        most_important_features = []
        if hasattr(model, 'feature_importances_'):
            # Plot feature importance in the fourth subplot
            feature_dict = dict(zip(X_train.columns, model.
↪feature_importances_))

            # Sort the features by their importance
            feature_dict = {k: v for k, v in sorted(feature_dict.items(),␣
↪key=lambda item: item[1], reverse=True)}

            # Plot the feature importance
            ax[2, 1].bar(feature_dict.keys(), feature_dict.values())
            ax[2, 1].set_title(f'Feature importance of {model_name}')
            ax[2, 1].set_ylabel('Importance')
            ax[2, 1].set_xticklabels(feature_dict.keys(), rotation=90)

            most_important_features = [k for k, v in feature_dict.items() if v␣
↪> 0.01]
            print(f'Most important features: {most_important_features}')
        else:
            ax[2, 1].set_visible(False)

    if not hasattr(model, 'feature_importances_'):
        most_important_features = None


    scores = []
    scores.append(normal_scores)
    normal_scores['model_name'] = model_name
    all_scores.append(normal_scores)
    return scores, model, most_important_features
```

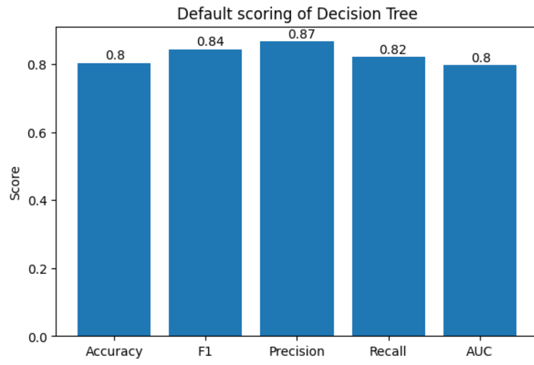
1.2.3 Decision tree model

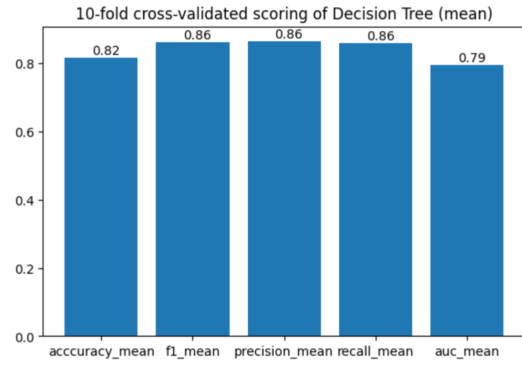
[40]:
```
from sklearn.tree import DecisionTreeClassifier

# Score the model with default parameters
score_dec_tree, model_dec_tree, most_importatn_features_dec_tree =␣
↪score_the_model(
    model=DecisionTreeClassifier(),
    model_name='Decision Tree',
```

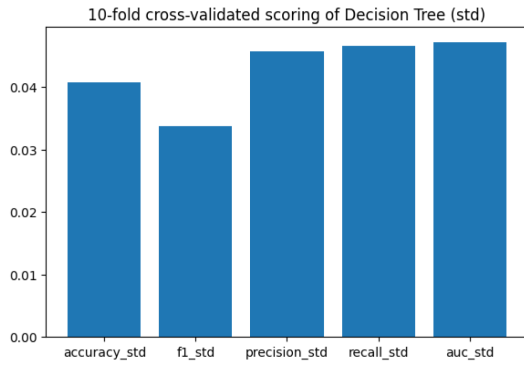
```
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)

print(score_dec_tree)
```

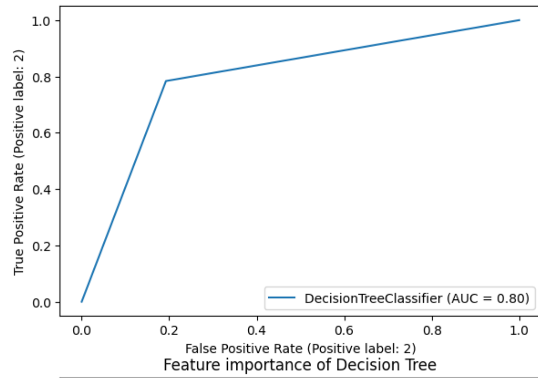


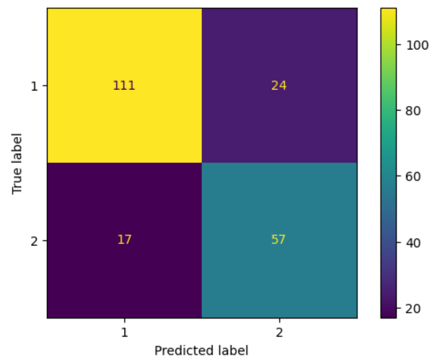

```
Most important features: ['V36', 'V1', 'V14', 'V27', 'V34', 'V12', 'V30', 'V38',
'V17', 'V31', 'V16', 'V8', 'V2', 'V37', 'V18', 'V39', 'V28', 'V3', 'V13', 'V15']
[{'Accuracy': 0.8038277511961722, 'F1': 0.844106463878327, 'Precision':
0.8671875, 'Recall': 0.8222222222222222, 'AUC': 0.7962462462462462,
'model_name': 'Decision Tree'}]
```

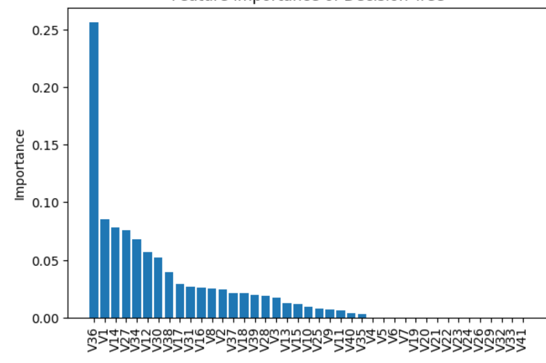

Default scoring of Decision Tree
Score
0.8
0.84
0.87
0.82
0.8
Accuracy
F1
Precision
Recall
AUC

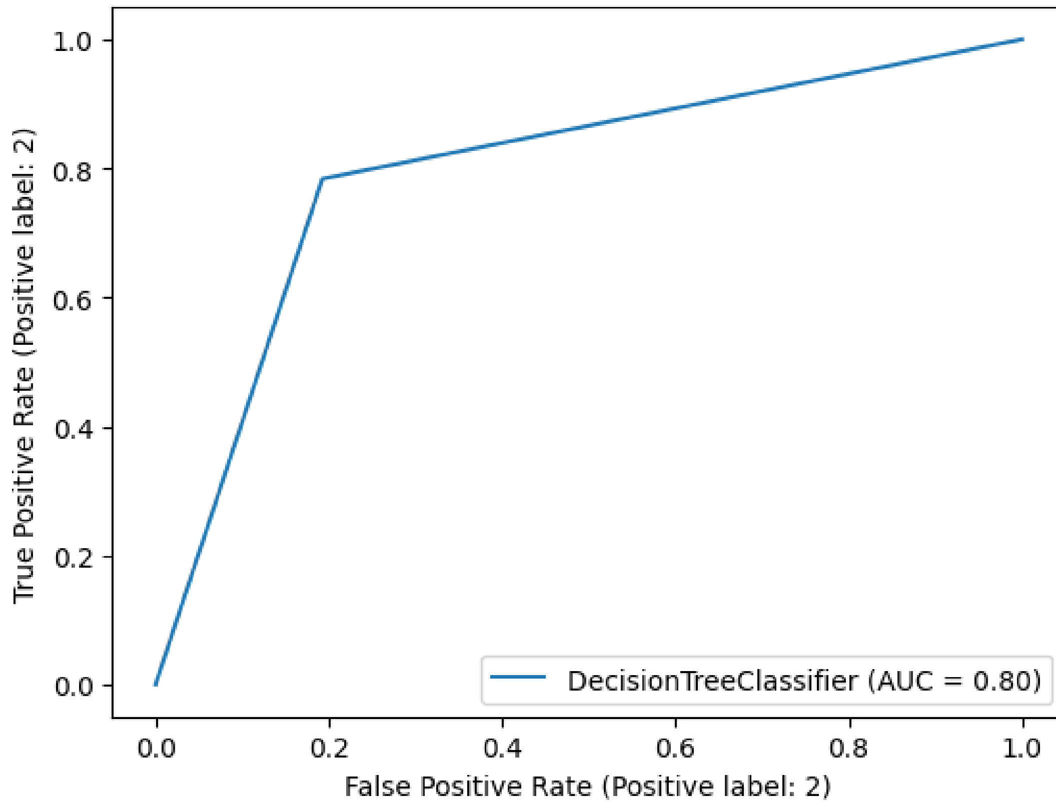

10-fold cross-validated scoring of Decision Tree (mean)
0.82
0.86
0.86
0.86
0.79
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Decision Tree (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std

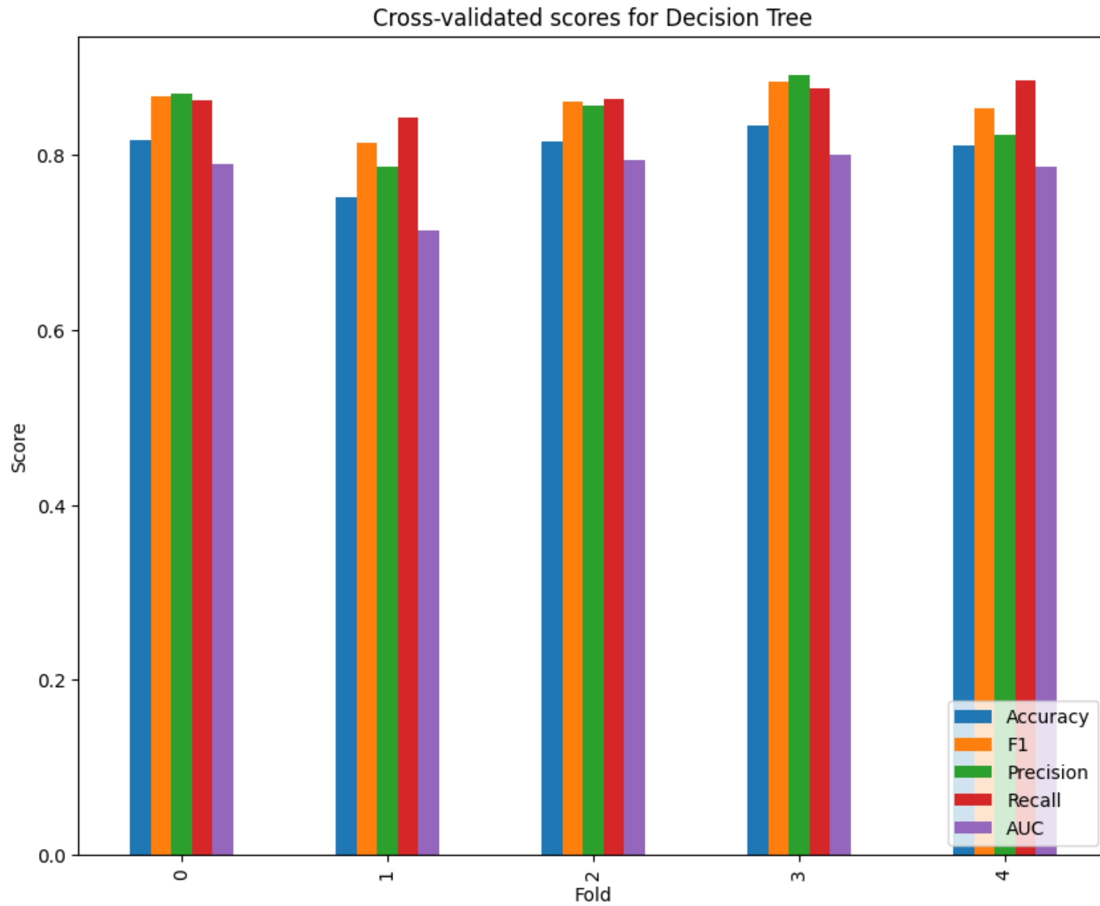

True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
DecisionTreeClassifier (AUC = 0.80)


111
24
17
57
True label
Predicted label

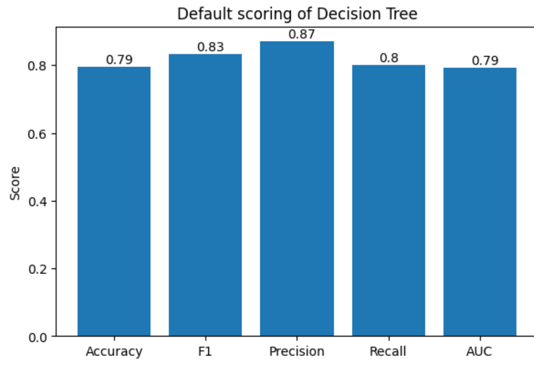

Feature importance of Decision Tree
Importance
V36 V1 V14 V27 V34 V12 V30 V38 V17 V31 V16 V8 V2 V37 V18 V39 V28 V3 V13 V15 V10 V25 V9 V11 V40 V35 V4 V5 V6 V7 V19 V20 V21 V22 V23 V24 V26 V29 V32 V33 V41

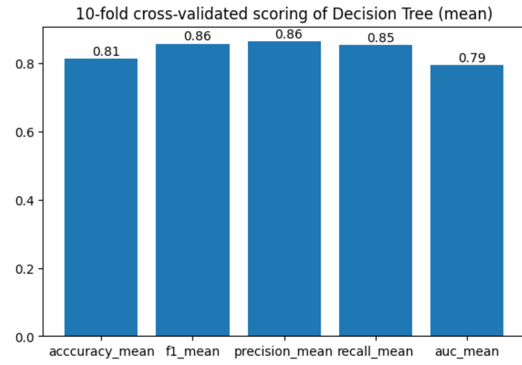


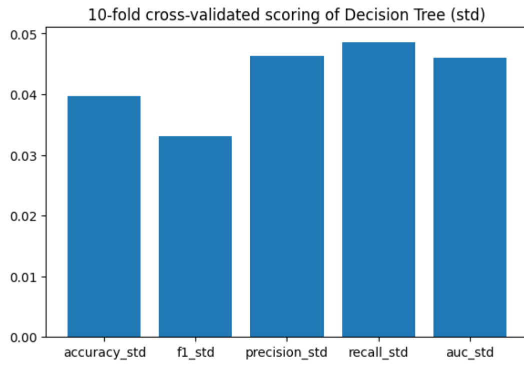
[41]:
```
# How is the model performing using subset of best features?
score_dec_tree, model_dec_tree, most_importatn_features_dec_tree =␣
 ↪score_the_model(
    model=DecisionTreeClassifier(),
    model_name='Decision Tree',
    random_seed=42,
    X_train=X_train[most_importatn_features_dec_tree],
    X_test=X_test[most_importatn_features_dec_tree],
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

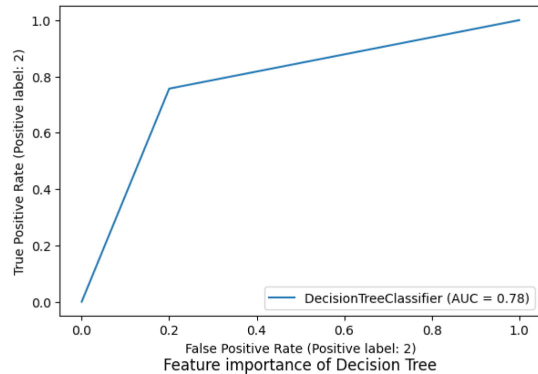


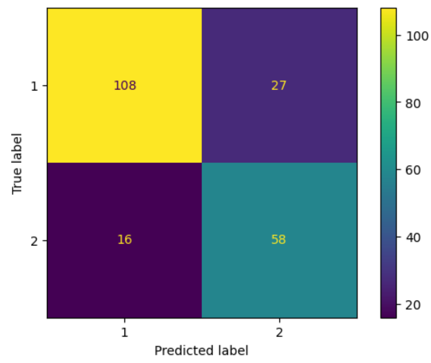

```
Most important features: ['V36', 'V1', 'V34', 'V27', 'V12', 'V14', 'V30', 'V37',
'V18', 'V38', 'V31', 'V16', 'V8', 'V17', 'V15', 'V3', 'V2', 'V39']
```

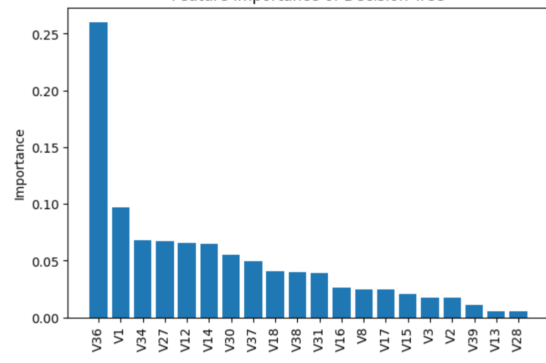

Default scoring of Decision Tree
Score
0.79
0.83
0.87
0.8
0.79
Accuracy
F1
Precision
Recall
AUC

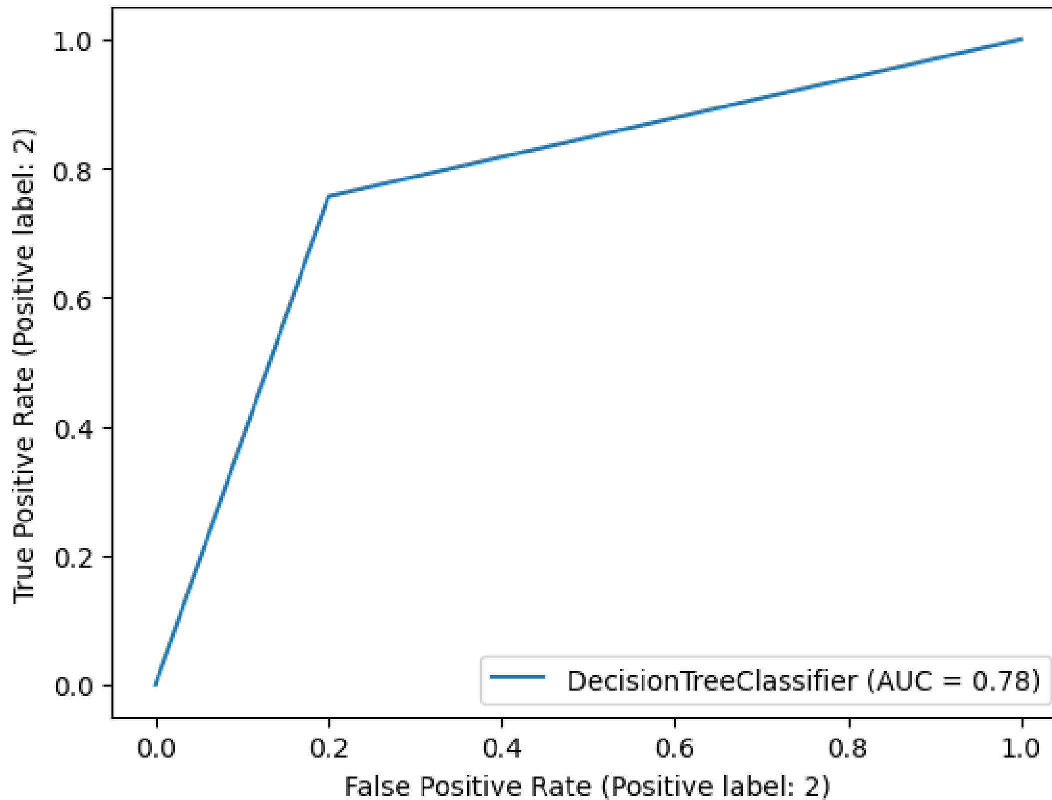

10-fold cross-validated scoring of Decision Tree (mean)
0.81
0.86
0.86
0.85
0.79
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Decision Tree (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
DecisionTreeClassifier (AUC = 0.78)


True label
Predicted label
108
27
16
58


Feature importance of Decision Tree
Importance
V36
V1
V34
V27
V12
V14
V30
V37
V18
V38
V31
V16
V8
V17
V15
V3
V2
V39
V13
V28



Now lets plot the decision tree

[42]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(40, 40), dpi=150)
plot_tree(model_dec_tree, filled=True, rounded=True, class_names=['Ready␣
↪biodegradable', 'Reday non-biodegradable'], feature_names=X_train.columns)
```

[42]:
```
[Text(0.5526960784313726, 0.9666666666666667, 'V1 <= 3.678\ngini =
0.443\nsamples = 762\nvalue = [510, 252]\nclass = Ready biodegradable'),
 Text(0.3424688057040998, 0.9, 'V2 <= 4.984\ngini = 0.485\nsamples = 325\nvalue
= [134, 191]\nclass = Reday non-biodegradable'),
 Text(0.23039215686274508, 0.8333333333333334, 'V5 <= 1.5\ngini = 0.451\nsamples
= 279\nvalue = [96, 183]\nclass = Reday non-biodegradable'),
 Text(0.14171122994652408, 0.7666666666666667, 'V3 <= 0.401\ngini =
0.347\nsamples = 206\nvalue = [46, 160]\nclass = Reday non-biodegradable'),
 Text(0.11319073083778966, 0.7, 'V15 <= 1.158\ngini = 0.231\nsamples = 15\nvalue
= [13, 2]\nclass = Ready biodegradable'),
 Text(0.09893048128342247, 0.6333333333333333, 'V17 <= 0.121\ngini =
0.133\nsamples = 14\nvalue = [13, 1]\nclass = Ready biodegradable'),
 Text(0.08467023172905526, 0.5666666666666667, 'gini = 0.0\nsamples = 13\nvalue
```

```
= [13, 0]\nclass = Ready biodegradable'),
 Text(0.11319073083778966, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.12745098039215685, 0.6333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.17023172905525846, 0.7, 'V8 <= 1.5\ngini = 0.286\nsamples = 191\nvalue =
[33, 158]\nclass = Reday non-biodegradable'),
 Text(0.15597147950089127, 0.6333333333333333, 'V18 <= 1.5\ngini =
0.262\nsamples = 187\nvalue = [29, 158]\nclass = Reday non-biodegradable'),
 Text(0.14171122994652408, 0.5666666666666667, 'V14 <= 1.986\ngini =
0.236\nsamples = 183\nvalue = [25, 158]\nclass = Reday non-biodegradable'),
 Text(0.07130124777183601, 0.5, 'V2 <= 4.776\ngini = 0.461\nsamples = 36\nvalue
= [13, 23]\nclass = Reday non-biodegradable'),
 Text(0.0570409982174688, 0.43333333333333335, 'V20 <= 9.798\ngini =
0.404\nsamples = 32\nvalue = [9, 23]\nclass = Reday non-biodegradable'),
 Text(0.0285204991087344, 0.36666666666666664, 'V1 <= 2.995\ngini =
0.488\nsamples = 19\nvalue = [8, 11]\nclass = Reday non-biodegradable'),
 Text(0.0142602495543672, 0.3, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]\nclass =
Reday non-biodegradable'),
 Text(0.0427807486631016, 0.3, 'V15 <= 1.118\ngini = 0.444\nsamples = 12\nvalue
= [8, 4]\nclass = Ready biodegradable'),
 Text(0.0285204991087344, 0.23333333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.0570409982174688, 0.23333333333333334, 'V15 <= 1.139\ngini =
0.32\nsamples = 10\nvalue = [8, 2]\nclass = Ready biodegradable'),
 Text(0.0427807486631016, 0.16666666666666666, 'gini = 0.0\nsamples = 6\nvalue =
[6, 0]\nclass = Ready biodegradable'),
 Text(0.07130124777183601, 0.16666666666666666, 'V9 <= 0.972\ngini =
0.5\nsamples = 4\nvalue = [2, 2]\nclass = Ready biodegradable'),
 Text(0.0570409982174688, 0.1, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass =
Reday non-biodegradable'),
 Text(0.0855614973262032, 0.1, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass =
Ready biodegradable'),
 Text(0.0855614973262032, 0.36666666666666664, 'V15 <= 1.142\ngini =
0.142\nsamples = 13\nvalue = [1, 12]\nclass = Reday non-biodegradable'),
 Text(0.07130124777183601, 0.3, 'gini = 0.0\nsamples = 12\nvalue = [0,
12]\nclass = Reday non-biodegradable'),
 Text(0.09982174688057041, 0.3, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass
= Ready biodegradable'),
 Text(0.0855614973262032, 0.43333333333333335, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]\nclass = Ready biodegradable'),
 Text(0.21212121212121213, 0.5, 'V9 <= 0.968\ngini = 0.15\nsamples = 147\nvalue
= [12, 135]\nclass = Reday non-biodegradable'),
 Text(0.16399286987522282, 0.43333333333333335, 'V6 <= 1.247\ngini =
0.444\nsamples = 3\nvalue = [2, 1]\nclass = Ready biodegradable'),
 Text(0.1497326203208556, 0.36666666666666664, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Ready biodegradable'),
```

```
 Text(0.17825311942959002, 0.36666666666666664, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]\nclass = Reday non-biodegradable'),
 Text(0.26024955436720143, 0.43333333333333335, 'V4 <= 2.223\ngini =
0.129\nsamples = 144\nvalue = [10, 134]\nclass = Reday non-biodegradable'),
 Text(0.20677361853832443, 0.36666666666666664, 'V15 <= 1.161\ngini =
0.09\nsamples = 127\nvalue = [6, 121]\nclass = Reday non-biodegradable'),
 Text(0.14260249554367202, 0.3, 'V15 <= 1.113\ngini = 0.051\nsamples =
114\nvalue = [3, 111]\nclass = Reday non-biodegradable'),
 Text(0.1140819964349376, 0.23333333333333334, 'V14 <= 2.401\ngini =
0.32\nsamples = 10\nvalue = [2, 8]\nclass = Reday non-biodegradable'),
 Text(0.09982174688057041, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]\nclass = Ready biodegradable'),
 Text(0.12834224598930483, 0.16666666666666666, 'gini = 0.0\nsamples = 8\nvalue
= [0, 8]\nclass = Reday non-biodegradable'),
 Text(0.1711229946524064, 0.23333333333333334, 'V3 <= 2.29\ngini =
0.019\nsamples = 104\nvalue = [1, 103]\nclass = Reday non-biodegradable'),
 Text(0.1568627450980392, 0.16666666666666666, 'gini = 0.0\nsamples = 84\nvalue
= [0, 84]\nclass = Reday non-biodegradable'),
 Text(0.18538324420677363, 0.16666666666666666, 'V3 <= 2.311\ngini =
0.095\nsamples = 20\nvalue = [1, 19]\nclass = Reday non-biodegradable'),
 Text(0.1711229946524064, 0.1, 'V9 <= 0.992\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]\nclass = Ready biodegradable'),
 Text(0.1568627450980392, 0.03333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Ready biodegradable'),
 Text(0.18538324420677363, 0.03333333333333333, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]\nclass = Reday non-biodegradable'),
 Text(0.19964349376114082, 0.1, 'gini = 0.0\nsamples = 18\nvalue = [0,
18]\nclass = Reday non-biodegradable'),
 Text(0.2709447415329768, 0.3, 'V15 <= 1.165\ngini = 0.355\nsamples = 13\nvalue
= [3, 10]\nclass = Reday non-biodegradable'),
 Text(0.25668449197860965, 0.23333333333333334, 'V2 <= 4.192\ngini =
0.5\nsamples = 6\nvalue = [3, 3]\nclass = Ready biodegradable'),
 Text(0.24242424242424243, 0.16666666666666666, 'V13 <= 1.915\ngini =
0.375\nsamples = 4\nvalue = [3, 1]\nclass = Ready biodegradable'),
 Text(0.2281639928698752, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass =
Reday non-biodegradable'),
 Text(0.25668449197860965, 0.1, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]\nclass
= Ready biodegradable'),
 Text(0.2709447415329768, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.28520499108734404, 0.23333333333333334, 'gini = 0.0\nsamples = 7\nvalue
= [0, 7]\nclass = Reday non-biodegradable'),
 Text(0.3137254901960784, 0.36666666666666664, 'V9 <= 0.983\ngini =
0.36\nsamples = 17\nvalue = [4, 13]\nclass = Reday non-biodegradable'),
 Text(0.2994652406417112, 0.3, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]\nclass =
Ready biodegradable'),
 Text(0.32798573975044565, 0.3, 'V1 <= 3.43\ngini = 0.231\nsamples = 15\nvalue =
```

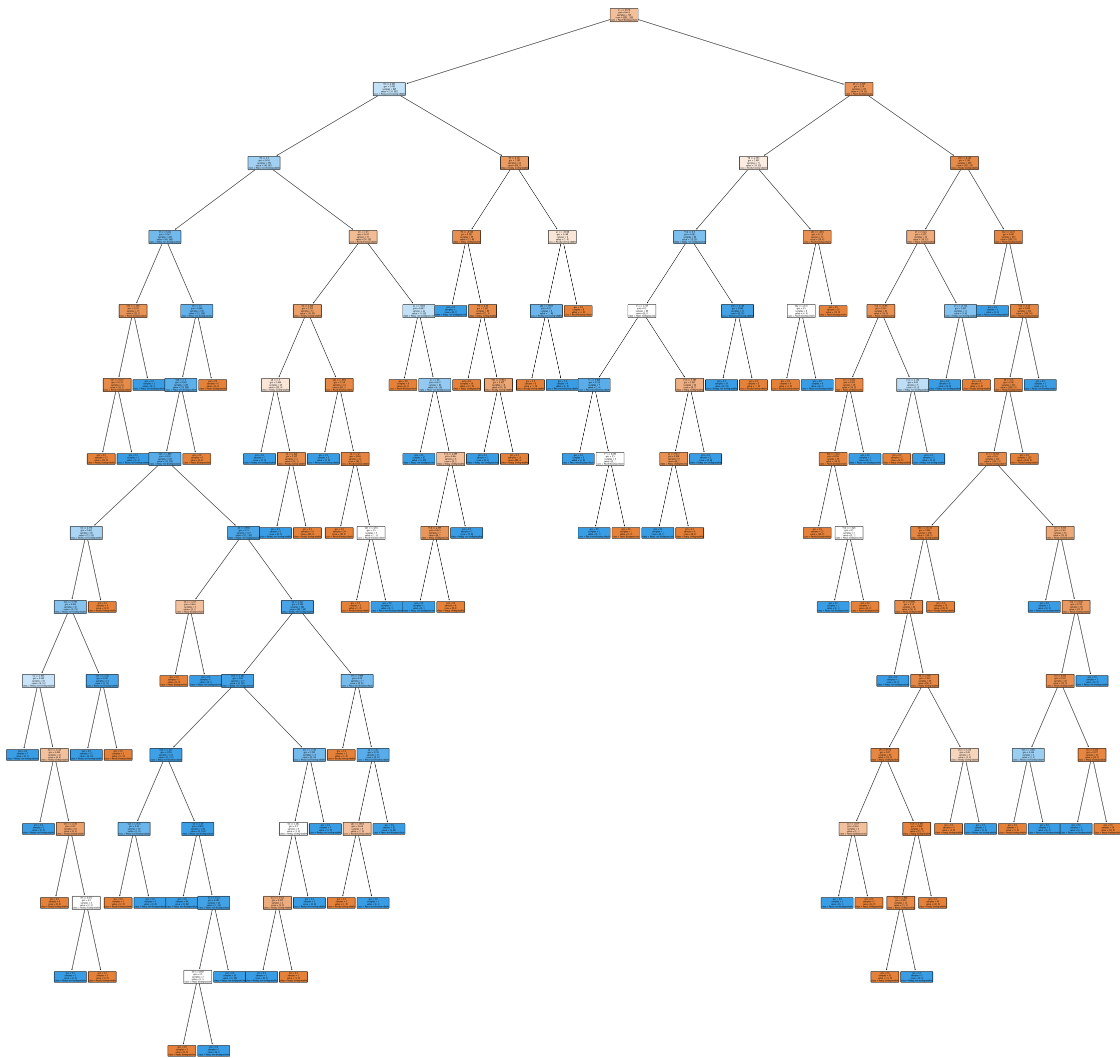
```
[2, 13]\nclass = Reday non-biodegradable'),
 Text(0.3137254901960784, 0.23333333333333334, 'V13 <= 3.663\ngini =
0.444\nsamples = 3\nvalue = [2, 1]\nclass = Ready biodegradable'),
 Text(0.2994652406417112, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Ready biodegradable'),
 Text(0.32798573975044565, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]\nclass = Reday non-biodegradable'),
 Text(0.3422459893048128, 0.23333333333333334, 'gini = 0.0\nsamples = 12\nvalue
= [0, 12]\nclass = Reday non-biodegradable'),
 Text(0.17023172905525846, 0.5666666666666667, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]\nclass = Ready biodegradable'),
 Text(0.18449197860962566, 0.6333333333333333, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]\nclass = Ready biodegradable'),
 Text(0.31907308377896615, 0.7666666666666667, 'V11 <= 0.5\ngini =
0.432\nsamples = 73\nvalue = [50, 23]\nclass = Ready biodegradable'),
 Text(0.26916221033868093, 0.7, 'V2 <= 4.426\ngini = 0.315\nsamples = 51\nvalue
= [41, 10]\nclass = Ready biodegradable'),
 Text(0.24064171122994651, 0.6333333333333333, 'V8 <= 1.5\ngini = 0.494\nsamples
= 18\nvalue = [10, 8]\nclass = Ready biodegradable'),
 Text(0.22638146167557932, 0.5666666666666667, 'gini = 0.0\nsamples = 7\nvalue =
[0, 7]\nclass = Reday non-biodegradable'),
 Text(0.2549019607843137, 0.5666666666666667, 'V1 <= 2.928\ngini =
0.165\nsamples = 11\nvalue = [10, 1]\nclass = Ready biodegradable'),
 Text(0.24064171122994651, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass
= Reday non-biodegradable'),
 Text(0.26916221033868093, 0.5, 'gini = 0.0\nsamples = 10\nvalue = [10,
0]\nclass = Ready biodegradable'),
 Text(0.2976827094474153, 0.6333333333333333, 'V14 <= 1.603\ngini =
0.114\nsamples = 33\nvalue = [31, 2]\nclass = Ready biodegradable'),
 Text(0.28342245989304815, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.31194295900178254, 0.5666666666666667, 'V4 <= 2.27\ngini =
0.061\nsamples = 32\nvalue = [31, 1]\nclass = Ready biodegradable'),
 Text(0.2976827094474153, 0.5, 'gini = 0.0\nsamples = 30\nvalue = [30, 0]\nclass
= Ready biodegradable'),
 Text(0.32620320855614976, 0.5, 'V13 <= 2.258\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]\nclass = Ready biodegradable'),
 Text(0.31194295900178254, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]\nclass = Ready biodegradable'),
 Text(0.3404634581105169, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.3689839572192513, 0.7, 'V4 <= 2.089\ngini = 0.483\nsamples = 22\nvalue =
[9, 13]\nclass = Reday non-biodegradable'),
 Text(0.35472370766488415, 0.6333333333333333, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]\nclass = Ready biodegradable'),
 Text(0.38324420677361853, 0.6333333333333333, 'V1 <= 3.6\ngini = 0.432\nsamples
= 19\nvalue = [6, 13]\nclass = Reday non-biodegradable'),
```

```
 Text(0.3689839572192513, 0.5666666666666667, 'gini = 0.0\nsamples = 10\nvalue =
[0, 10]\nclass = Reday non-biodegradable'),
 Text(0.39750445632798576, 0.5666666666666667, 'V10 <= 2.975\ngini =
0.444\nsamples = 9\nvalue = [6, 3]\nclass = Ready biodegradable'),
 Text(0.38324420677361853, 0.5, 'V14 <= 2.355\ngini = 0.245\nsamples = 7\nvalue
= [6, 1]\nclass = Ready biodegradable'),
 Text(0.3689839572192513, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.39750445632798576, 0.43333333333333335, 'gini = 0.0\nsamples = 6\nvalue
= [6, 0]\nclass = Ready biodegradable'),
 Text(0.4117647058823529, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]\nclass =
Reday non-biodegradable'),
 Text(0.45454545454545453, 0.8333333333333334, 'V6 <= 0.013\ngini =
0.287\nsamples = 46\nvalue = [38, 8]\nclass = Ready biodegradable'),
 Text(0.4117647058823529, 0.7666666666666667, 'V6 <= -3.595\ngini =
0.193\nsamples = 37\nvalue = [33, 4]\nclass = Ready biodegradable'),
 Text(0.39750445632798576, 0.7, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass
= Reday non-biodegradable'),
 Text(0.42602495543672014, 0.7, 'V3 <= 1.85\ngini = 0.153\nsamples = 36\nvalue =
[33, 3]\nclass = Ready biodegradable'),
 Text(0.4117647058823529, 0.6333333333333333, 'gini = 0.0\nsamples = 23\nvalue =
[23, 0]\nclass = Ready biodegradable'),
 Text(0.44028520499108736, 0.6333333333333333, 'V20 <= 10.664\ngini =
0.355\nsamples = 13\nvalue = [10, 3]\nclass = Ready biodegradable'),
 Text(0.42602495543672014, 0.5666666666666667, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3]\nclass = Reday non-biodegradable'),
 Text(0.45454545454545453, 0.5666666666666667, 'gini = 0.0\nsamples = 10\nvalue
= [10, 0]\nclass = Ready biodegradable'),
 Text(0.49732620320855614, 0.7666666666666667, 'V7 <= 10.694\ngini =
0.494\nsamples = 9\nvalue = [5, 4]\nclass = Ready biodegradable'),
 Text(0.483065953654189, 0.7, 'V10 <= 0.902\ngini = 0.32\nsamples = 5\nvalue =
[1, 4]\nclass = Reday non-biodegradable'),
 Text(0.46880570409982175, 0.6333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Ready biodegradable'),
 Text(0.49732620320855614, 0.6333333333333333, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]\nclass = Reday non-biodegradable'),
 Text(0.5115864527629234, 0.7, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]\nclass =
Ready biodegradable'),
 Text(0.7629233511586453, 0.9, 'V6 <= -0.595\ngini = 0.24\nsamples = 437\nvalue
= [376, 61]\nclass = Ready biodegradable'),
 Text(0.6684491978609626, 0.8333333333333334, 'V4 <= 2.363\ngini =
0.497\nsamples = 72\nvalue = [39, 33]\nclass = Ready biodegradable'),
 Text(0.6114081996434938, 0.7666666666666667, 'V11 <= 6.5\ngini = 0.381\nsamples
= 39\nvalue = [10, 29]\nclass = Reday non-biodegradable'),
 Text(0.5686274509803921, 0.7, 'V3 <= 1.207\ngini = 0.5\nsamples = 18\nvalue =
[9, 9]\nclass = Ready biodegradable'),
 Text(0.5258467023172906, 0.6333333333333333, 'V1 <= 3.916\ngini =
```

```
0.245\nsamples = 7\nvalue = [1, 6]\nclass = Reday non-biodegradable'),
 Text(0.5115864527629234, 0.5666666666666667, 'gini = 0.0\nsamples = 5\nvalue =
[0, 5]\nclass = Reday non-biodegradable'),
 Text(0.5401069518716578, 0.5666666666666667, 'V7 <= 9.886\ngini = 0.5\nsamples
= 2\nvalue = [1, 1]\nclass = Ready biodegradable'),
 Text(0.5258467023172906, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass =
Reday non-biodegradable'),
 Text(0.5543672014260249, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]\nclass =
Ready biodegradable'),
 Text(0.6114081996434938, 0.6333333333333333, 'V10 <= 3.283\ngini =
0.397\nsamples = 11\nvalue = [8, 3]\nclass = Ready biodegradable'),
 Text(0.5971479500891266, 0.5666666666666667, 'V7 <= 9.872\ngini =
0.198\nsamples = 9\nvalue = [8, 1]\nclass = Ready biodegradable'),
 Text(0.5828877005347594, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass =
Reday non-biodegradable'),
 Text(0.6114081996434938, 0.5, 'gini = 0.0\nsamples = 8\nvalue = [8, 0]\nclass =
Ready biodegradable'),
 Text(0.6256684491978609, 0.5666666666666667, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.6541889483065954, 0.7, 'V10 <= 9.256\ngini = 0.091\nsamples = 21\nvalue
= [1, 20]\nclass = Reday non-biodegradable'),
 Text(0.6399286987522281, 0.6333333333333333, 'gini = 0.0\nsamples = 20\nvalue =
[0, 20]\nclass = Reday non-biodegradable'),
 Text(0.6684491978609626, 0.6333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Ready biodegradable'),
 Text(0.7254901960784313, 0.7666666666666667, 'V10 <= 1.805\ngini =
0.213\nsamples = 33\nvalue = [29, 4]\nclass = Ready biodegradable'),
 Text(0.7112299465240641, 0.7, 'V12 <= 48.35\ngini = 0.5\nsamples = 8\nvalue =
[4, 4]\nclass = Ready biodegradable'),
 Text(0.696969696969697, 0.6333333333333333, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]\nclass = Ready biodegradable'),
 Text(0.7254901960784313, 0.6333333333333333, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]\nclass = Reday non-biodegradable'),
 Text(0.7397504456327986, 0.7, 'gini = 0.0\nsamples = 25\nvalue = [25, 0]\nclass
= Ready biodegradable'),
 Text(0.857397504456328, 0.8333333333333334, 'V16 <= 8.446\ngini =
0.142\nsamples = 365\nvalue = [337, 28]\nclass = Ready biodegradable'),
 Text(0.8181818181818182, 0.7666666666666667, 'V7 <= 5.124\ngini =
0.375\nsamples = 52\nvalue = [39, 13]\nclass = Ready biodegradable'),
 Text(0.7825311942959001, 0.7, 'V12 <= 45.85\ngini = 0.214\nsamples = 41\nvalue
= [36, 5]\nclass = Ready biodegradable'),
 Text(0.7540106951871658, 0.6333333333333333, 'V2 <= 5.601\ngini =
0.105\nsamples = 36\nvalue = [34, 2]\nclass = Ready biodegradable'),
 Text(0.7397504456327986, 0.5666666666666667, 'V19 <= 3.444\ngini =
0.056\nsamples = 35\nvalue = [34, 1]\nclass = Ready biodegradable'),
 Text(0.7254901960784313, 0.5, 'gini = 0.0\nsamples = 33\nvalue = [33, 0]\nclass
= Ready biodegradable'),
```

```
 Text(0.7540106951871658, 0.5, 'V19 <= 3.526\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]\nclass = Ready biodegradable'),
 Text(0.7397504456327986, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.768270944741533, 0.43333333333333335, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Ready biodegradable'),
 Text(0.768270944741533, 0.5666666666666667, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.8110516934046346, 0.6333333333333333, 'V15 <= 1.105\ngini =
0.48\nsamples = 5\nvalue = [2, 3]\nclass = Reday non-biodegradable'),
 Text(0.7967914438502673, 0.5666666666666667, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Ready biodegradable'),
 Text(0.8253119429590018, 0.5666666666666667, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3]\nclass = Reday non-biodegradable'),
 Text(0.8538324420677362, 0.7, 'V7 <= 15.793\ngini = 0.397\nsamples = 11\nvalue
= [3, 8]\nclass = Reday non-biodegradable'),
 Text(0.839572192513369, 0.6333333333333333, 'gini = 0.0\nsamples = 8\nvalue =
[0, 8]\nclass = Reday non-biodegradable'),
 Text(0.8680926916221033, 0.6333333333333333, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]\nclass = Ready biodegradable'),
 Text(0.8966131907308378, 0.7666666666666667, 'V12 <= 13.25\ngini =
0.091\nsamples = 313\nvalue = [298, 15]\nclass = Ready biodegradable'),
 Text(0.8823529411764706, 0.7, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass =
Reday non-biodegradable'),
 Text(0.910873440285205, 0.7, 'V10 <= 9.19\ngini = 0.086\nsamples = 312\nvalue =
[298, 14]\nclass = Ready biodegradable'),
 Text(0.8966131907308378, 0.6333333333333333, 'V8 <= 0.5\ngini = 0.08\nsamples =
311\nvalue = [298, 13]\nclass = Ready biodegradable'),
 Text(0.8823529411764706, 0.5666666666666667, 'V7 <= 10.253\ngini =
0.14\nsamples = 172\nvalue = [159, 13]\nclass = Ready biodegradable'),
 Text(0.82174688057041, 0.5, 'V20 <= 10.136\ngini = 0.069\nsamples = 139\nvalue
= [134, 5]\nclass = Ready biodegradable'),
 Text(0.8074866310160428, 0.43333333333333335, 'V3 <= 0.599\ngini =
0.15\nsamples = 61\nvalue = [56, 5]\nclass = Ready biodegradable'),
 Text(0.7932263814616756, 0.36666666666666664, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.82174688057041, 0.36666666666666664, 'V4 <= 2.305\ngini = 0.124\nsamples
= 60\nvalue = [56, 4]\nclass = Ready biodegradable'),
 Text(0.786096256684492, 0.3, 'V9 <= 0.979\ngini = 0.07\nsamples = 55\nvalue =
[53, 2]\nclass = Ready biodegradable'),
 Text(0.7575757575757576, 0.23333333333333334, 'V4 <= 1.837\ngini =
0.444\nsamples = 3\nvalue = [2, 1]\nclass = Ready biodegradable'),
 Text(0.7433155080213903, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.7718360071301248, 0.16666666666666666, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]\nclass = Ready biodegradable'),
 Text(0.8146167557932263, 0.23333333333333334, 'V13 <= 2.957\ngini =
```

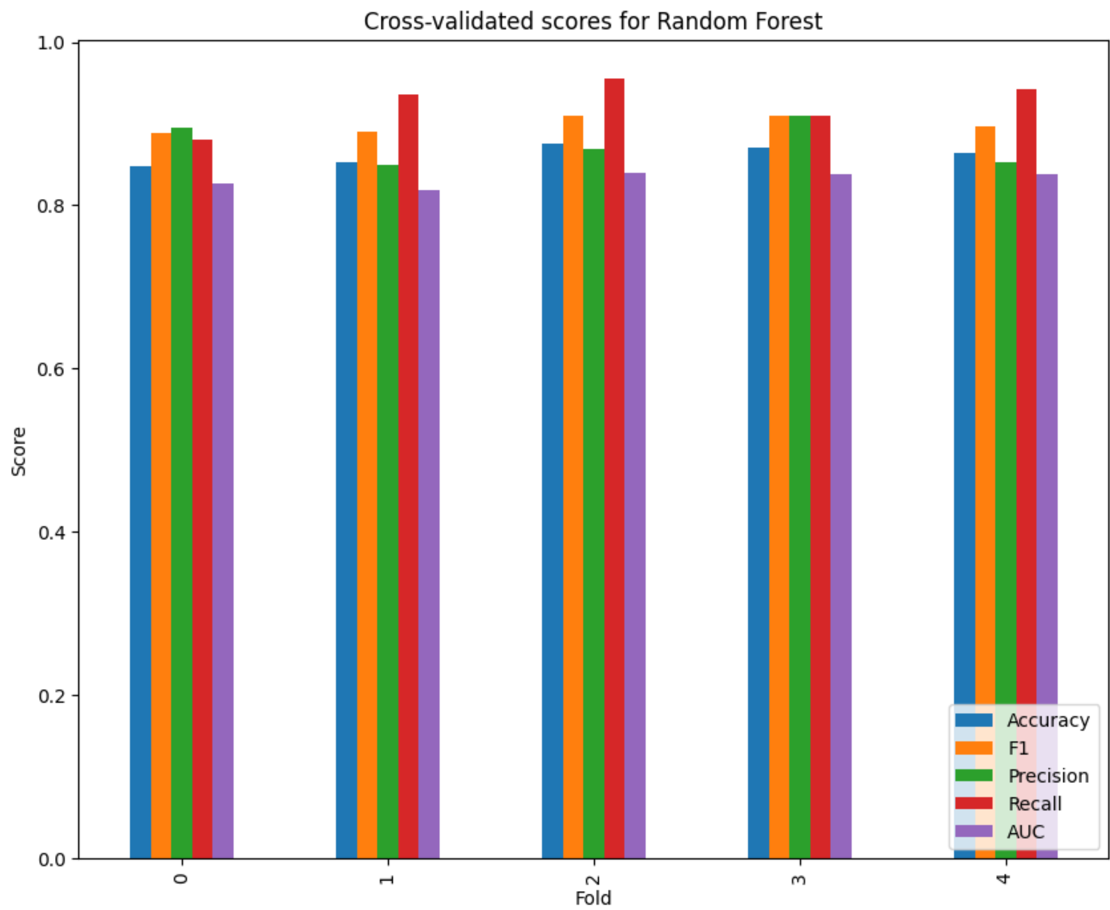
```
0.038\nsamples = 52\nvalue = [51, 1]\nclass = Ready biodegradable'),
 Text(0.8003565062388592, 0.16666666666666666, 'V13 <= 2.946\ngini =
0.153\nsamples = 12\nvalue = [11, 1]\nclass = Ready biodegradable'),
 Text(0.786096256684492, 0.1, 'gini = 0.0\nsamples = 11\nvalue = [11, 0]\nclass
= Ready biodegradable'),
 Text(0.8146167557932263, 0.1, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]\nclass =
Reday non-biodegradable'),
 Text(0.8288770053475936, 0.16666666666666666, 'gini = 0.0\nsamples = 40\nvalue
= [40, 0]\nclass = Ready biodegradable'),
 Text(0.857397504456328, 0.3, 'V14 <= 2.225\ngini = 0.48\nsamples = 5\nvalue =
[3, 2]\nclass = Ready biodegradable'),
 Text(0.8431372549019608, 0.23333333333333334, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]\nclass = Ready biodegradable'),
 Text(0.8716577540106952, 0.23333333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.8360071301247772, 0.43333333333333335, 'gini = 0.0\nsamples = 78\nvalue
= [78, 0]\nclass = Ready biodegradable'),
 Text(0.9429590017825312, 0.5, 'V2 <= 5.053\ngini = 0.367\nsamples = 33\nvalue =
[25, 8]\nclass = Ready biodegradable'),
 Text(0.928698752228164, 0.43333333333333335, 'gini = 0.0\nsamples = 3\nvalue =
[0, 3]\nclass = Reday non-biodegradable'),
 Text(0.9572192513368984, 0.43333333333333335, 'V14 <= 3.148\ngini =
0.278\nsamples = 30\nvalue = [25, 5]\nclass = Ready biodegradable'),
 Text(0.9429590017825312, 0.36666666666666664, 'V3 <= 0.653\ngini =
0.191\nsamples = 28\nvalue = [25, 3]\nclass = Ready biodegradable'),
 Text(0.9144385026737968, 0.3, 'V15 <= 1.085\ngini = 0.444\nsamples = 3\nvalue =
[1, 2]\nclass = Reday non-biodegradable'),
 Text(0.9001782531194296, 0.23333333333333334, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]\nclass = Ready biodegradable'),
 Text(0.928698752228164, 0.23333333333333334, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.9714795008912656, 0.3, 'V4 <= 2.155\ngini = 0.077\nsamples = 25\nvalue =
[24, 1]\nclass = Ready biodegradable'),
 Text(0.9572192513368984, 0.23333333333333334, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable'),
 Text(0.9857397504456328, 0.23333333333333334, 'gini = 0.0\nsamples = 24\nvalue
= [24, 0]\nclass = Ready biodegradable'),
 Text(0.9714795008912656, 0.36666666666666664, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]\nclass = Reday non-biodegradable'),
 Text(0.910873440285205, 0.5666666666666667, 'gini = 0.0\nsamples = 139\nvalue =
[139, 0]\nclass = Ready biodegradable'),
 Text(0.9251336898395722, 0.6333333333333333, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]\nclass = Reday non-biodegradable')]
```



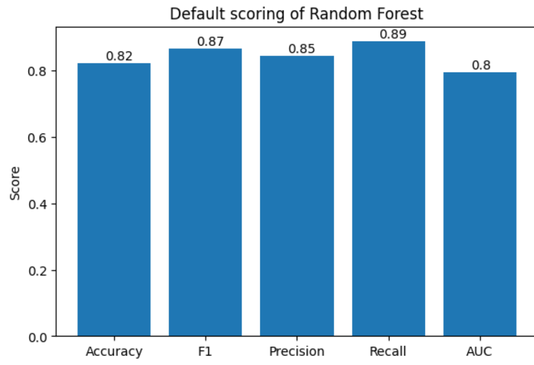
1.2.4 Random Forrest Classifier

[43]:
```
from sklearn.ensemble import RandomForestClassifier

# Score the model with default parameters
score_rf, model_rf, most_important_features_rf = score_the_model(
    model=RandomForestClassifier(),
    model_name='Random Forest',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
```

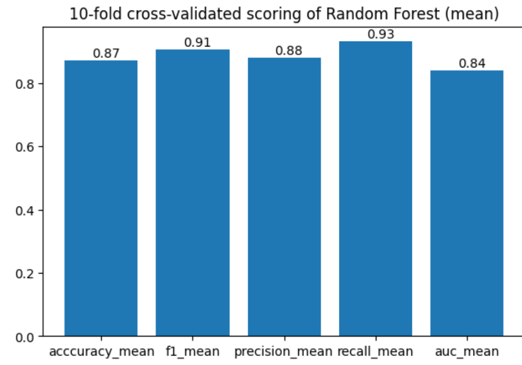
```
    plot=True
)
```



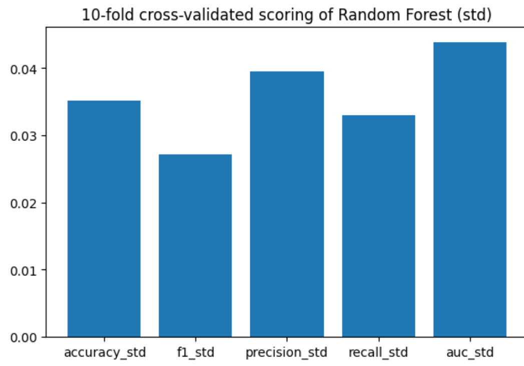

```
Most important features: ['V36', 'V39', 'V22', 'V27', 'V1', 'V12', 'V15', 'V13',
'V18', 'V34', 'V14', 'V37', 'V30', 'V2', 'V17', 'V31', 'V8', 'V16', 'V3', 'V38',
'V10', 'V28', 'V9', 'V11', 'V5', 'V7', 'V6', 'V41']
```



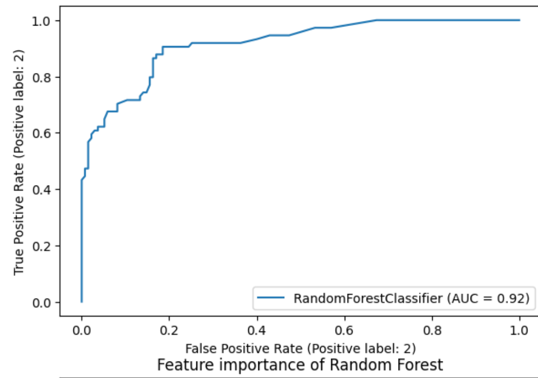
Default scoring of Random Forest
Score
0.0
0.2
0.4
0.6
0.8
0.82
0.87
0.85
0.89
0.8
Accuracy
F1
Precision
Recall
AUC



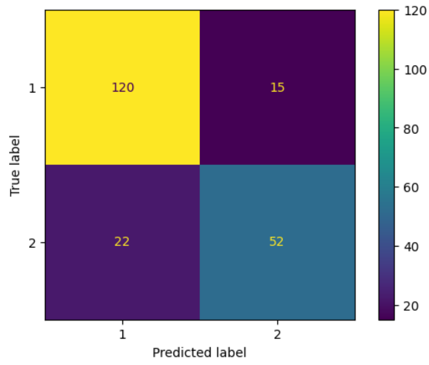
10-fold cross-validated scoring of Random Forest (mean)
0.0
0.2
0.4
0.6
0.8
0.87
0.91
0.88
0.93
0.84
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean



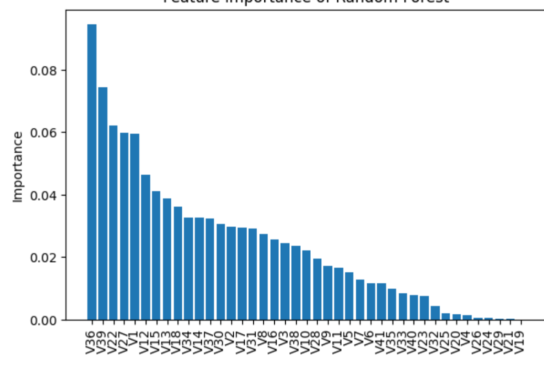
10-fold cross-validated scoring of Random Forest (std)
0.00
0.01
0.02
0.03
0.04
accuracy_std
f1_std
precision_std
recall_std
auc_std



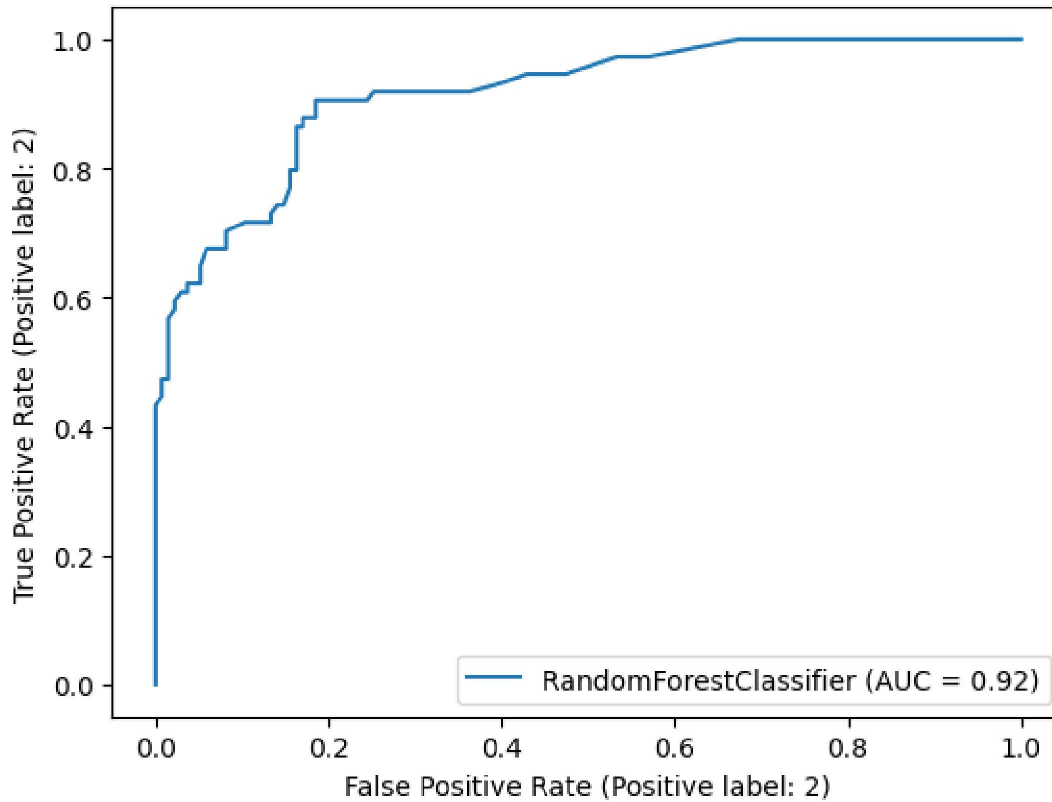
True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
0.0
0.2
0.4
0.6
0.8
1.0
RandomForestClassifier (AUC = 0.92)



True label
Predicted label
1
2
120
15
22
52
20
40
60
80
100
120

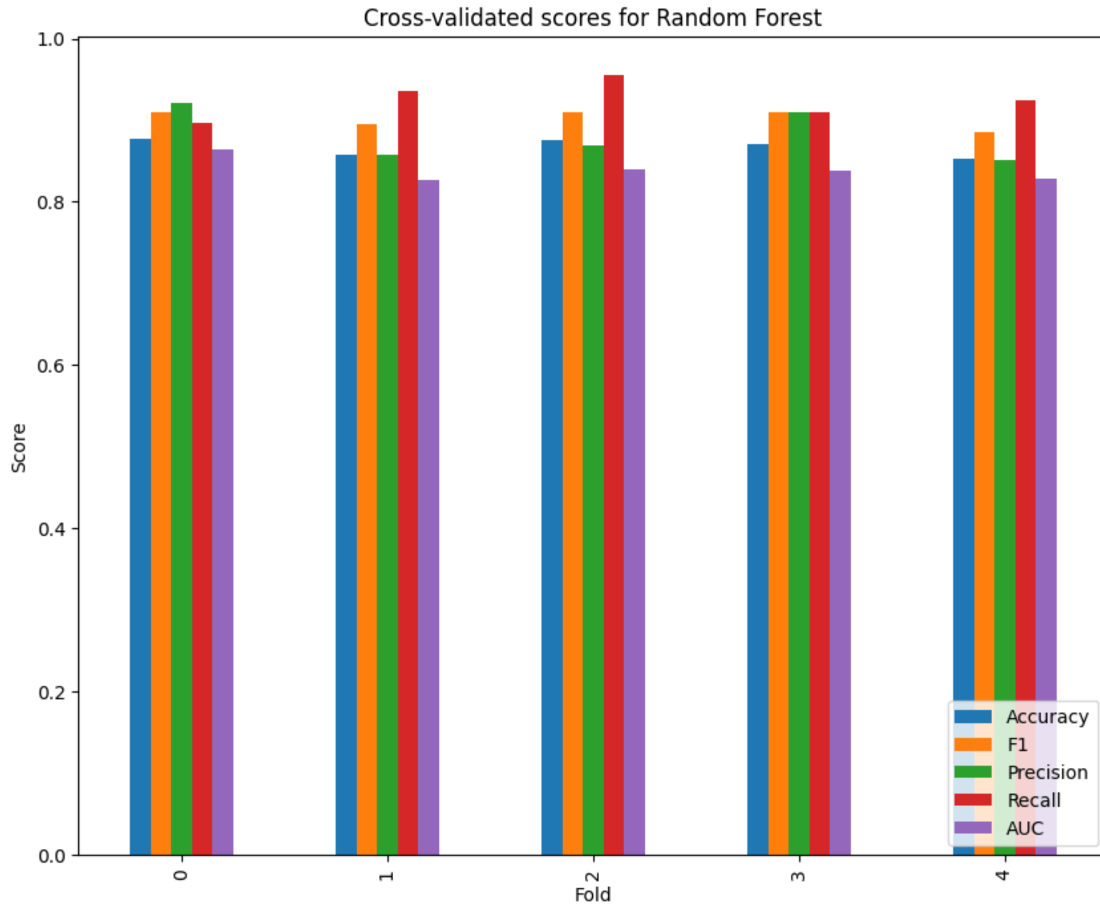


Feature importance of Random Forest
Importance
0.00
0.02
0.04
0.06
0.08
V36 V39 V22 V27 V1 V12 V15 V13 V18 V34 V14 V37 V30 V2 V17 V31 V8 V16 V3 V38 V10 V28 V9 V11 V5 V7 V6 V41 V35 V33 V40 V23 V32 V25 V20 V4 V26 V24 V29 V21 V19

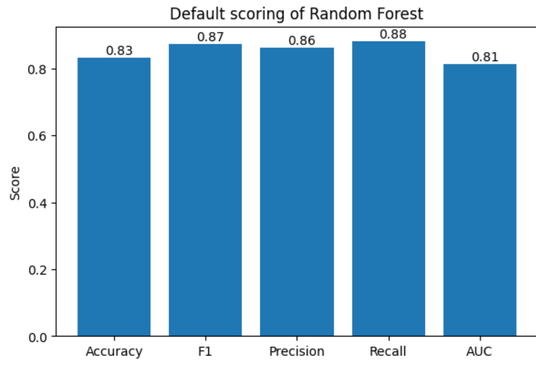


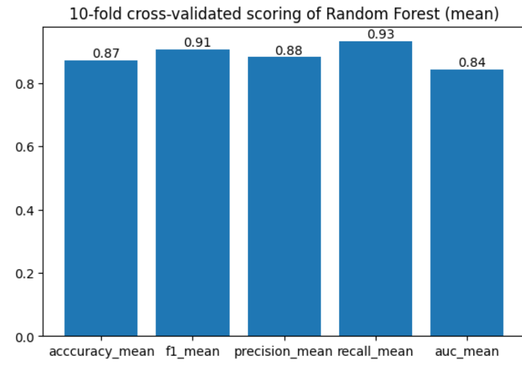
[44]:

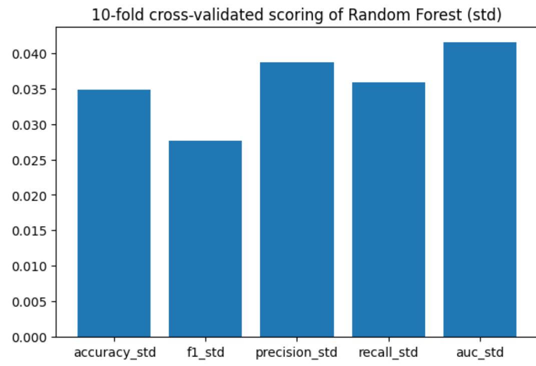
```
# Select the best features
score_rf, model_rf, most_important_features_rf = score_the_model(
    model=RandomForestClassifier(),
    model_name='Random Forest',
    random_seed=42,
    X_train=X_train[most_important_features_rf],
    X_test=X_test[most_important_features_rf],
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

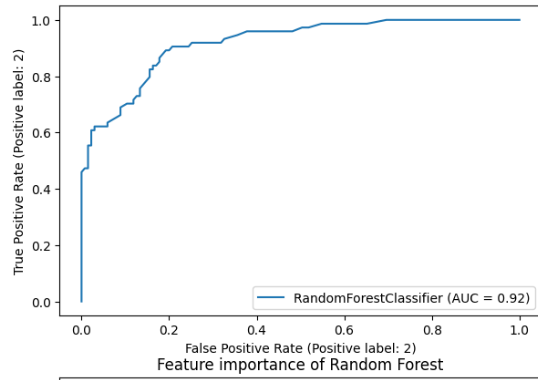


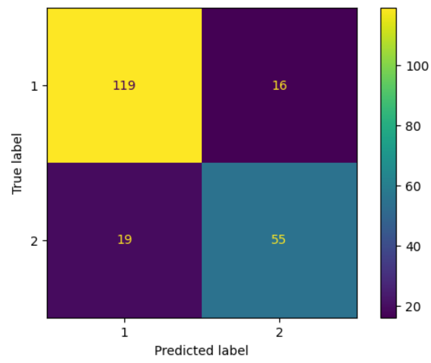

```
Most important features: ['V36', 'V1', 'V39', 'V27', 'V22', 'V12', 'V18', 'V13',
'V30', 'V15', 'V14', 'V37', 'V8', 'V17', 'V34', 'V2', 'V16', 'V38', 'V10',
'V31', 'V3', 'V28', 'V7', 'V11', 'V5', 'V9', 'V41']
```

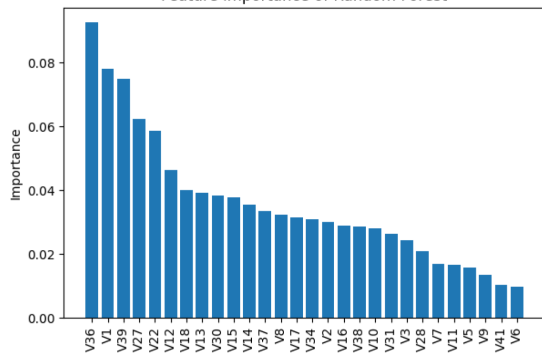

Default scoring of Random Forest
0.83
0.87
0.86
0.88
0.81
Score
Accuracy
F1
Precision
Recall
AUC

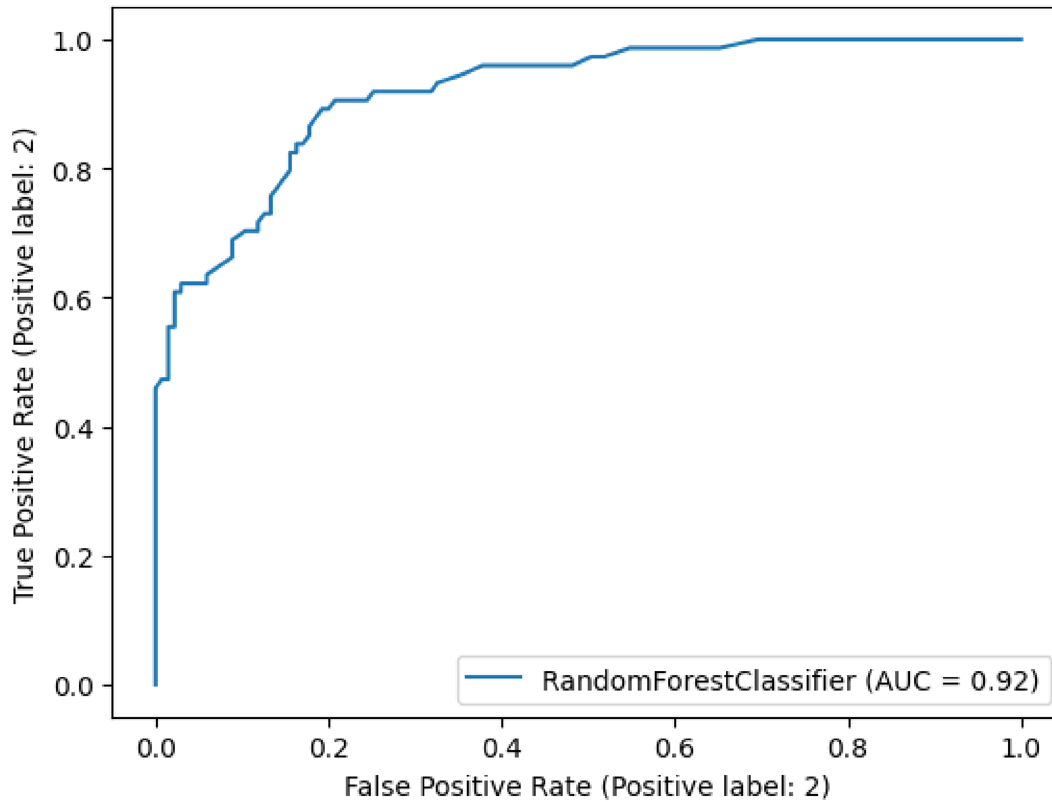

10-fold cross-validated scoring of Random Forest (mean)
0.87
0.91
0.88
0.93
0.84
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Random Forest (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
RandomForestClassifier (AUC = 0.92)

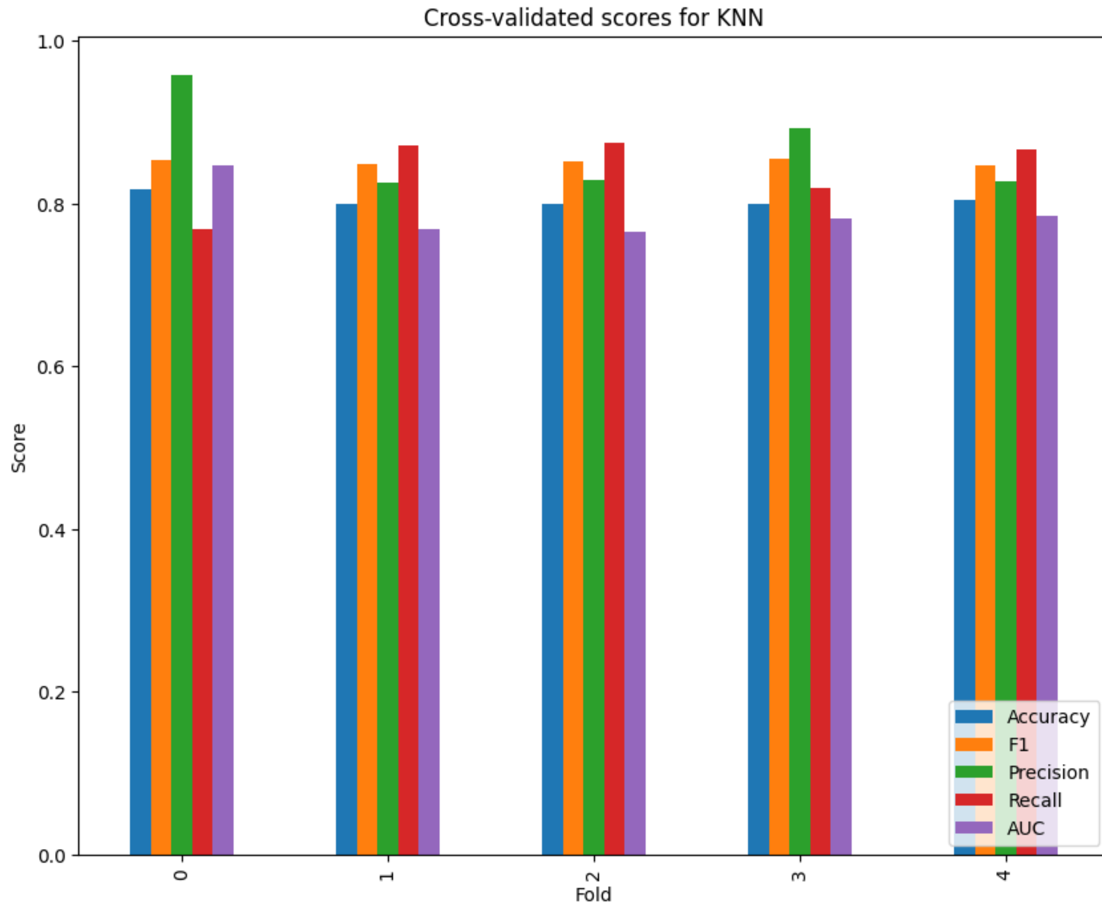

119
16
19
55
True label
Predicted label

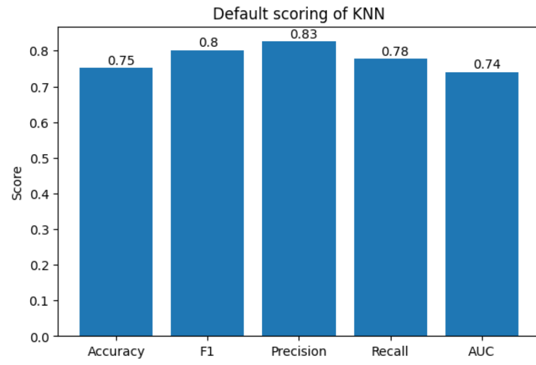
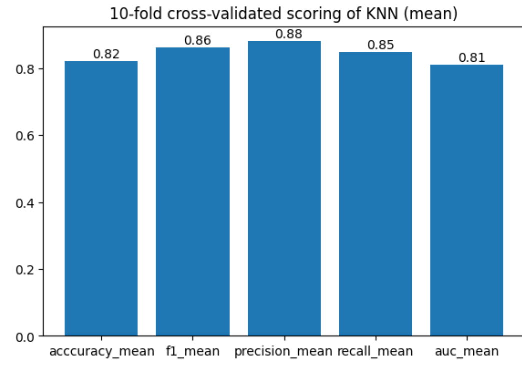
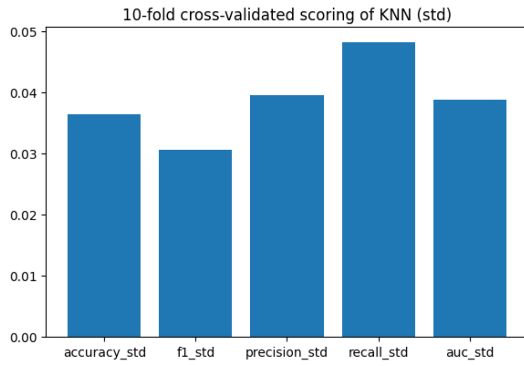
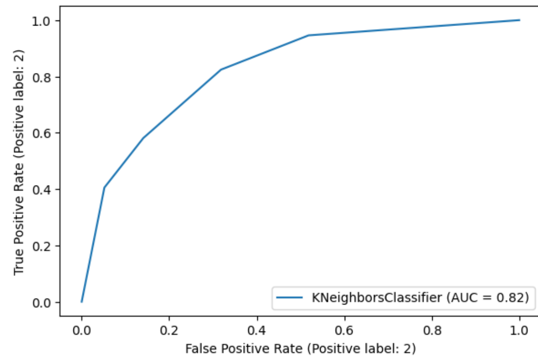
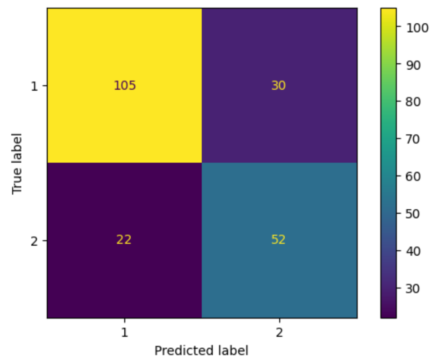

Feature importance of Random Forest
Importance
V36
V1
V39
V27
V22
V12
V18
V13
V30
V15
V14
V37
V8
V17
V34
V2
V16
V38
V10
V31
V3
V28
V7
V11
V5
V9
V41
V6

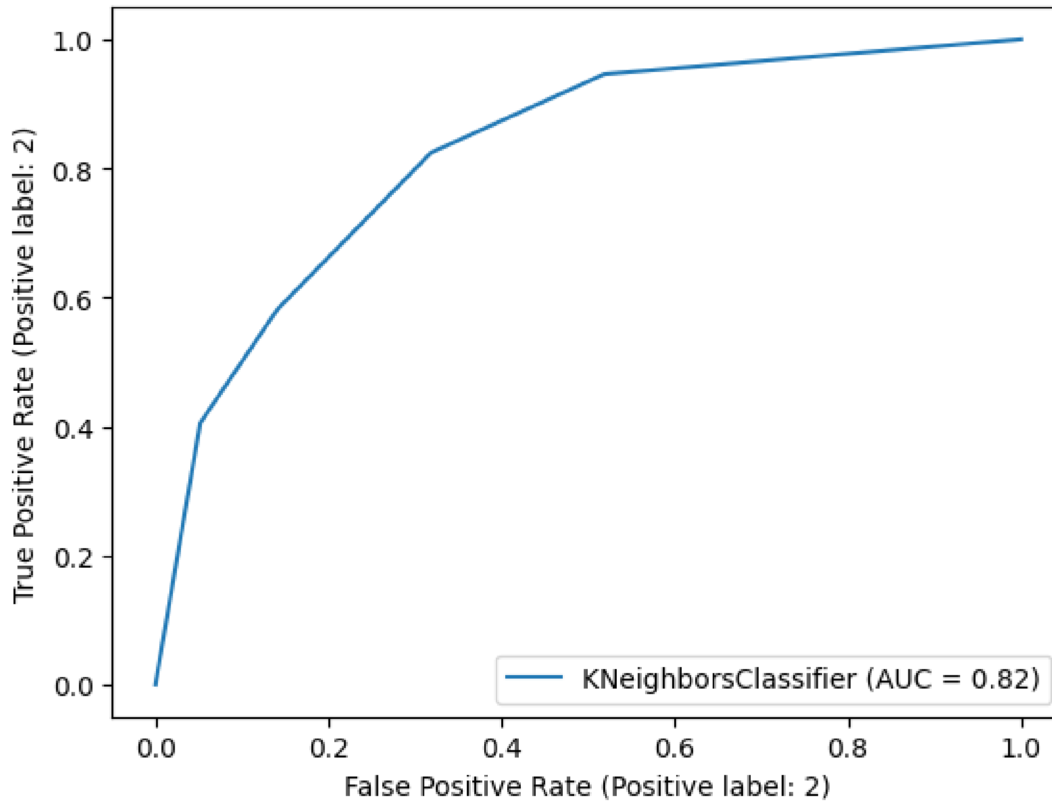


1.2.5 KNN

[45]:
```
from sklearn.neighbors import KNeighborsClassifier

# Score the model with default parameters
score_knn, model_knn, _ = score_the_model(
    model=KNeighborsClassifier(),
    model_name='KNN',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

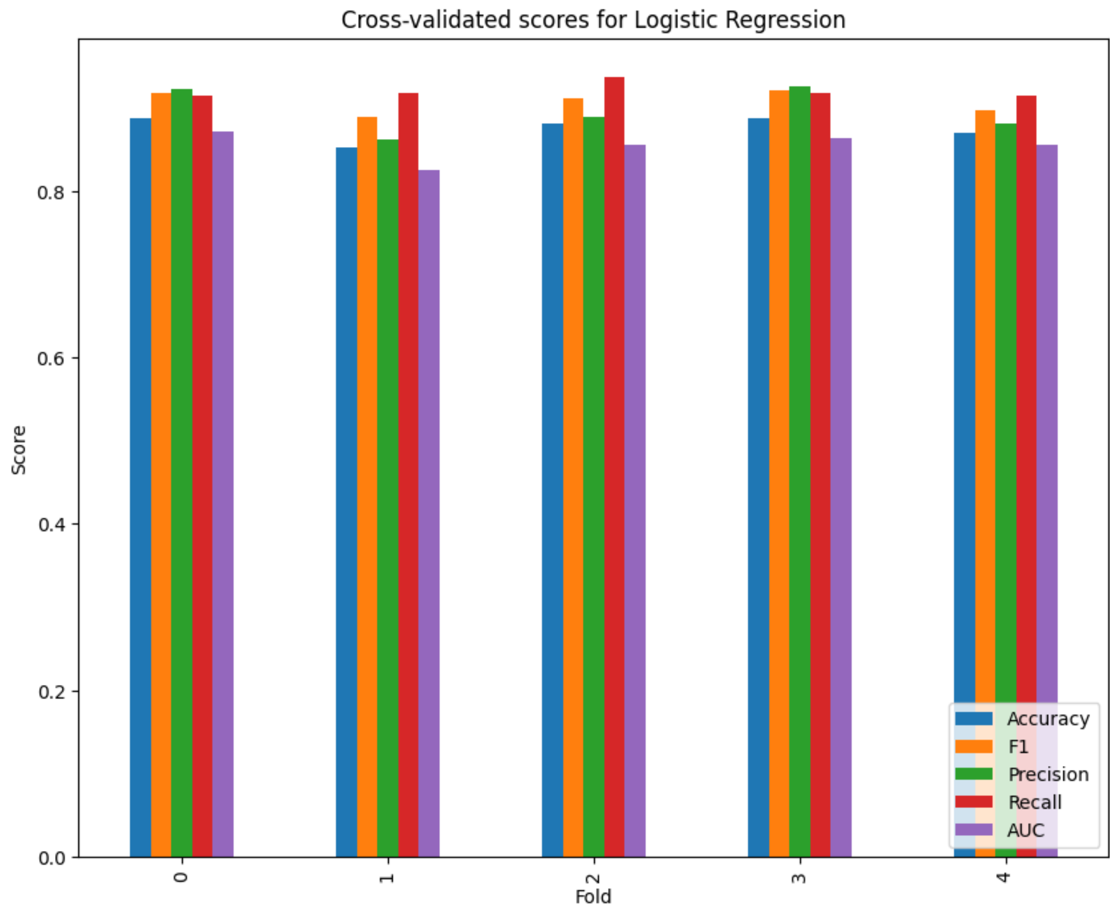

Cross-validated scores for KNN
1.0
0.8
0.6
0.4
0.2
0.0
Score
0
1
2
3
4
Fold
Accuracy
F1
Precision
Recall
AUC

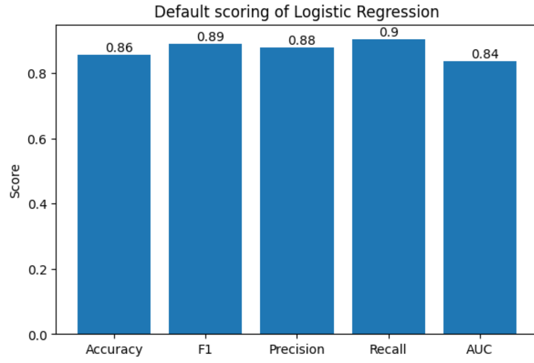

Default scoring of KNN
Score
0.75
0.8
0.83
0.78
0.74
Accuracy
F1
Precision
Recall
AUC

10-fold cross-validated scoring of KNN (mean)
0.82
0.86
0.88
0.85
0.81
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean

10-fold cross-validated scoring of KNN (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std

True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
KNeighborsClassifier (AUC = 0.82)

105
30
22
52
True label
Predicted label

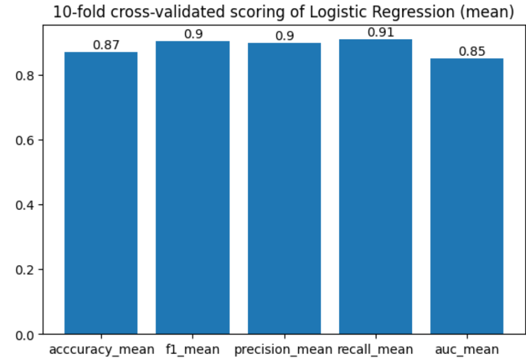


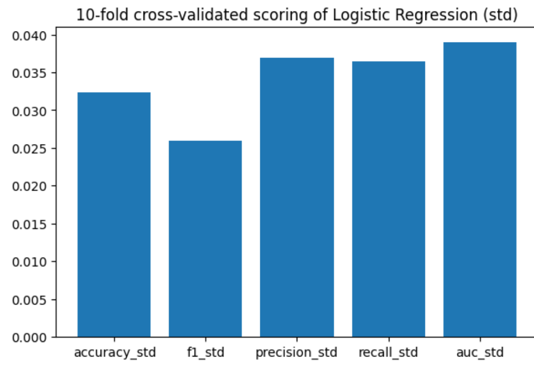
Logistic Regression

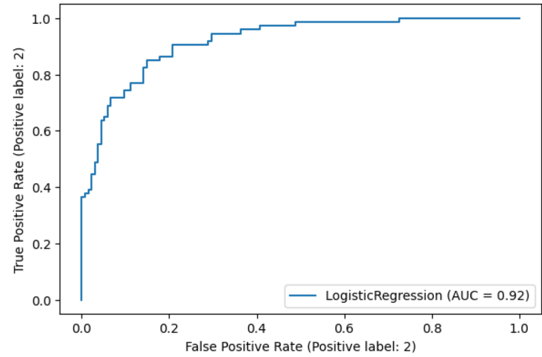
[46]:
```
from sklearn.linear_model import LogisticRegression

# Score the model with default parameters
score_log_reg, model_log_reg,_ = score_the_model(
    model=LogisticRegression(max_iter=100),
    model_name='Logistic Regression',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

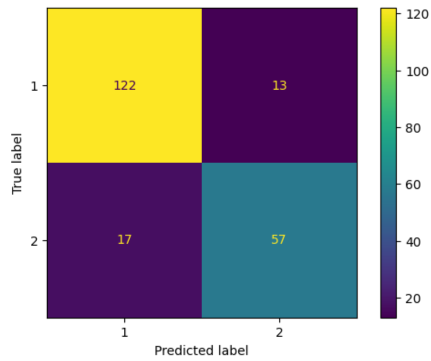

Cross-validated scores for Logistic Regression
Score
0.0
0.2
0.4
0.6
0.8
Fold
0
1
2
3
4
Accuracy
F1
Precision
Recall
AUC

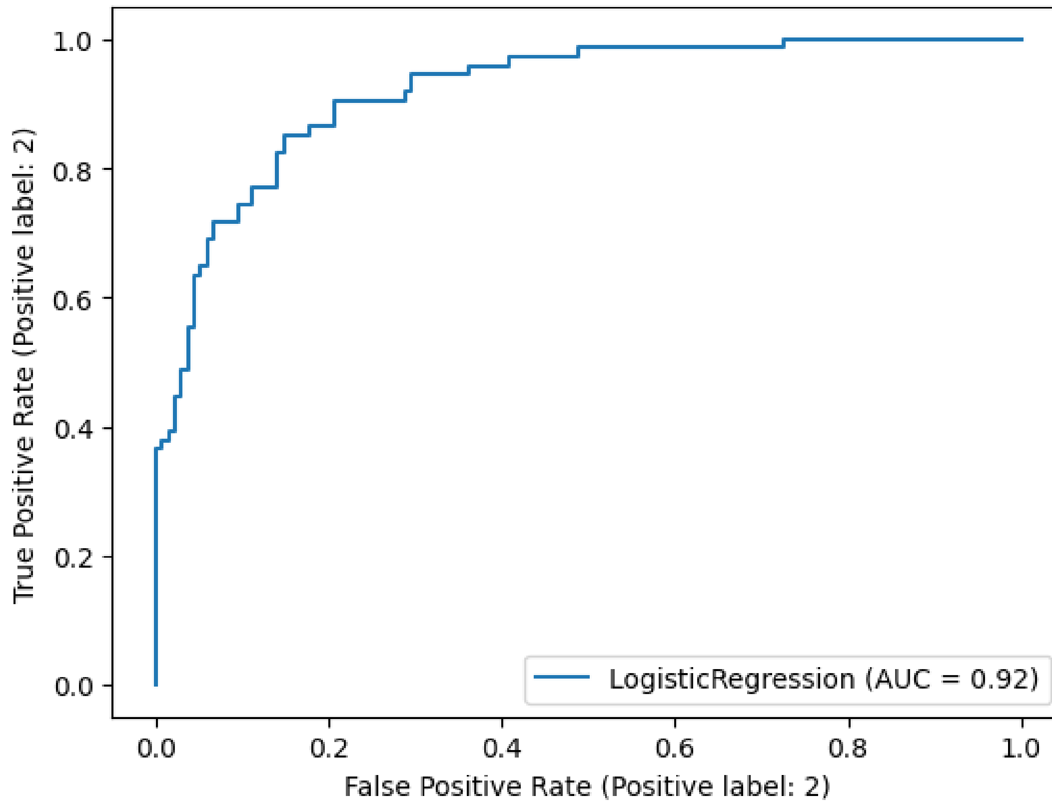

Default scoring of Logistic Regression
Score
0.86
0.89
0.88
0.9
0.84
Accuracy
F1
Precision
Recall
AUC


10-fold cross-validated scoring of Logistic Regression (mean)
0.87
0.9
0.9
0.91
0.85
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Logistic Regression (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
LogisticRegression (AUC = 0.92)


122
13
17
57
True label
Predicted label



SVM

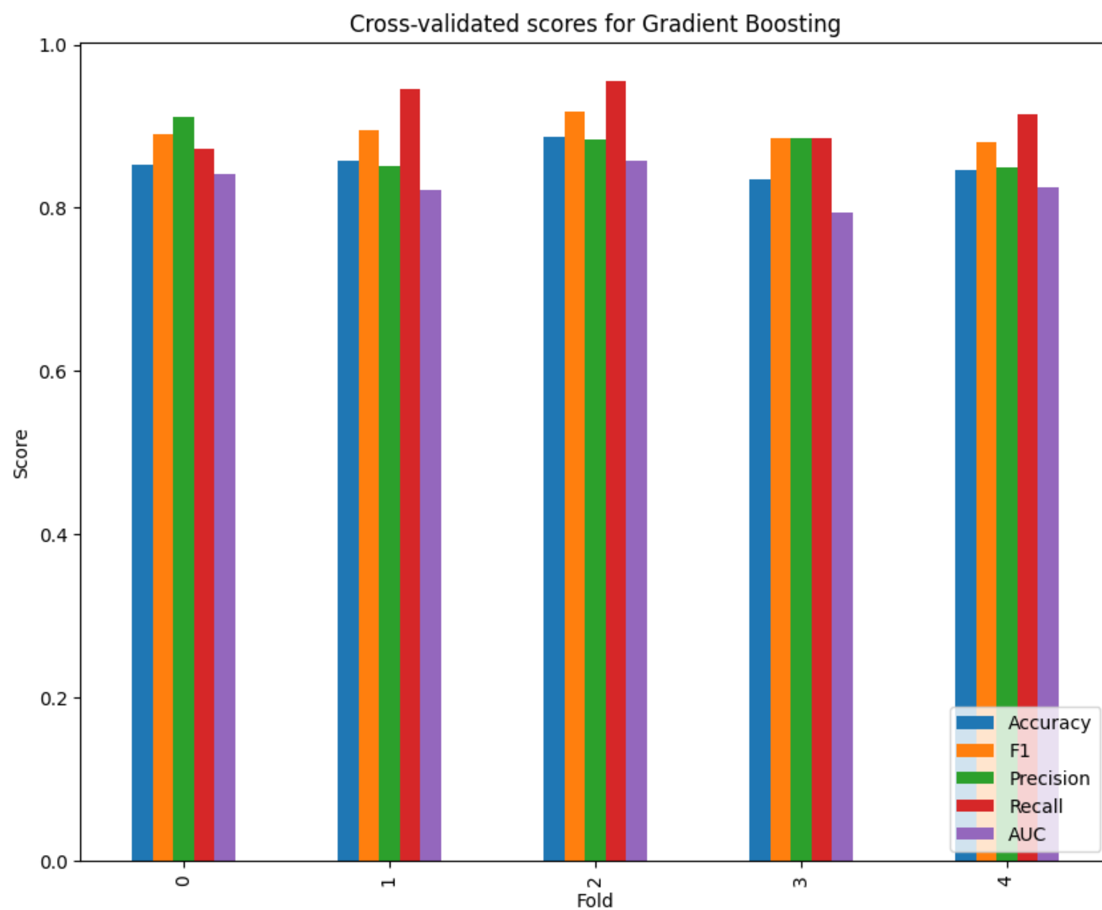
[47]:
```
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Scale the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Score the model with default parameters

scores_svm, model_svm, most_important_features_svm = score_the_model(
    model=SVC(),
    model_name='SVM',
    random_seed=42,
    X_train=X_train_scaled,
    X_test=X_test_scaled,
    y_train=y_train,
    y_test=y_test,
    plot=False
)
```

```
print(scores_svm)
```

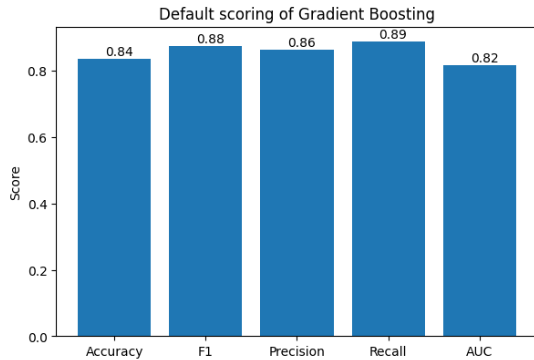
```
[{'Accuracy': 0.8660287081339713, 'F1': 0.9, 'Precision': 0.8689655172413793,
'Recall': 0.9333333333333333, 'AUC': 0.8382882882882883, 'model_name': 'SVM'}]
```

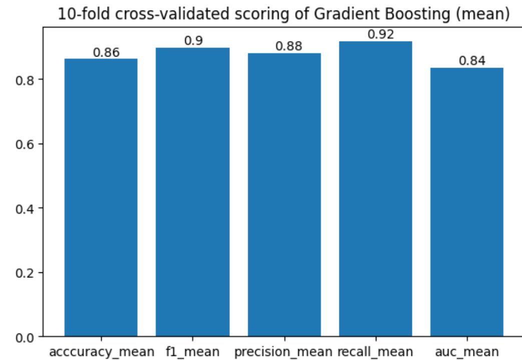
1.2.6 Gradient Boosting Classifier

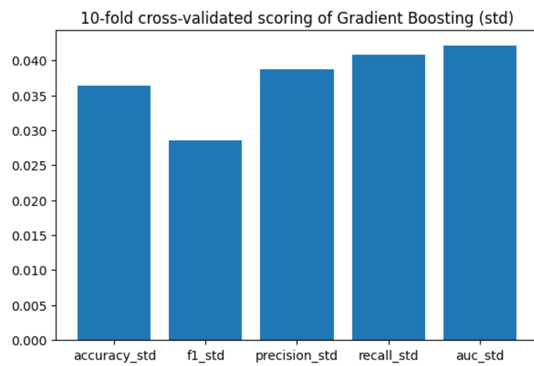
[48]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Score the model with default parameters
score_gb, model_gb, most_important_features_gb = score_the_model(
    model=GradientBoostingClassifier(),
    model_name='Gradient Boosting',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

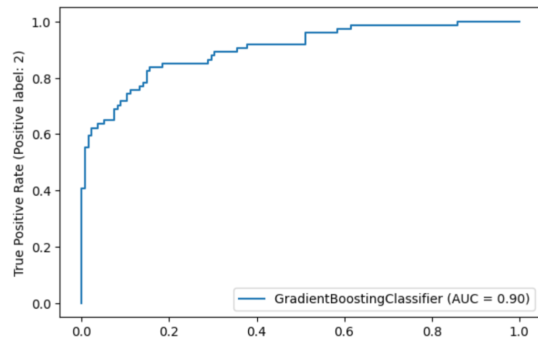


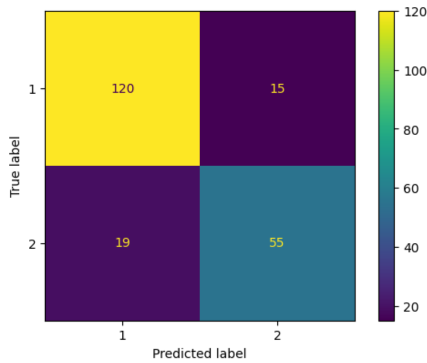
```
Most important features: ['V36', 'V1', 'V12', 'V39', 'V38', 'V34', 'V30', 'V22',
'V27', 'V13', 'V2', 'V14', 'V40', 'V8', 'V37', 'V18', 'V9', 'V28', 'V10', 'V31']
```

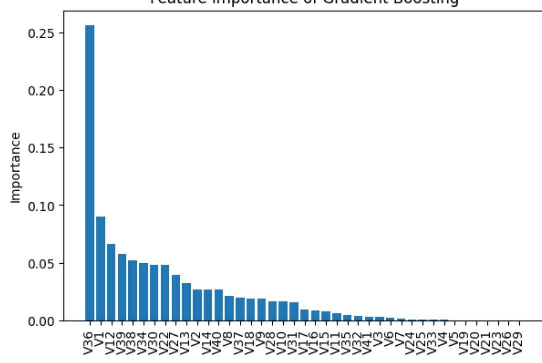


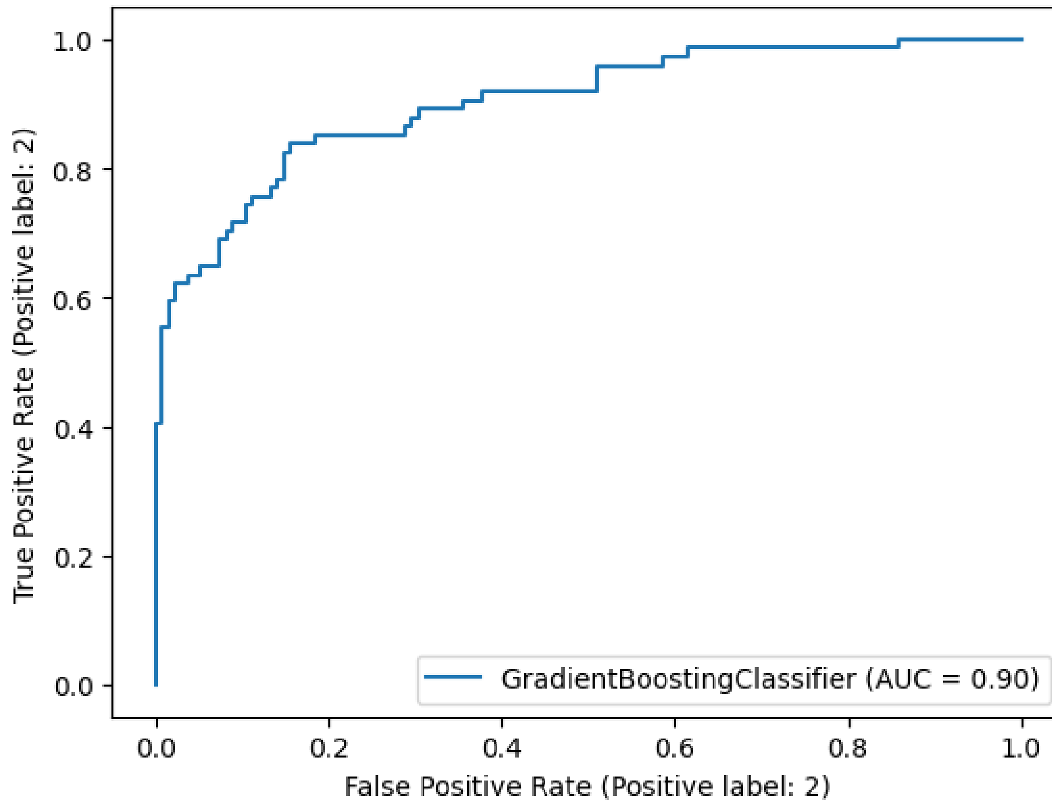






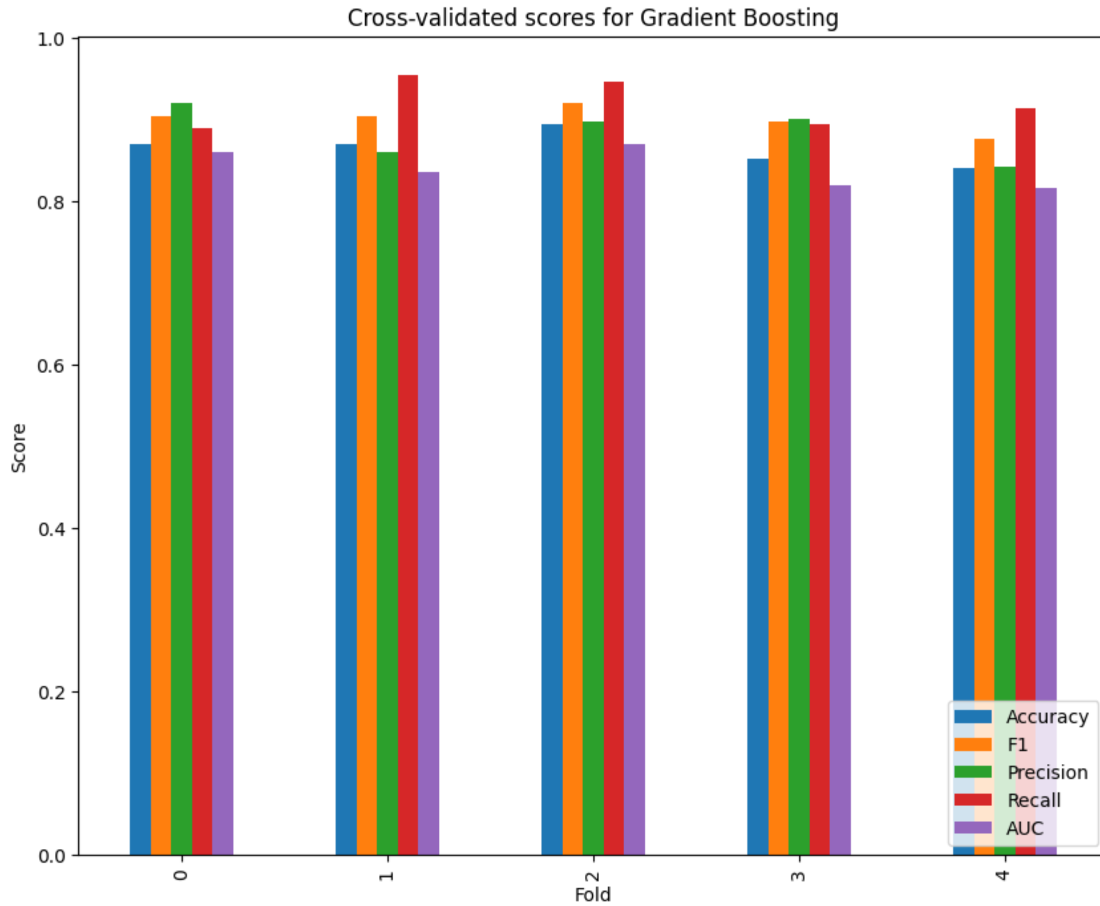




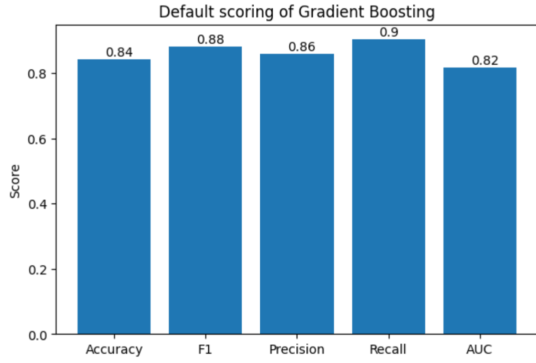
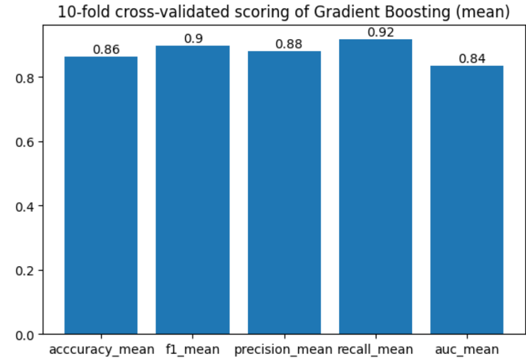
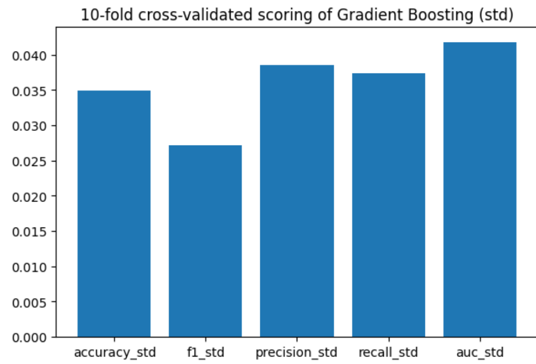
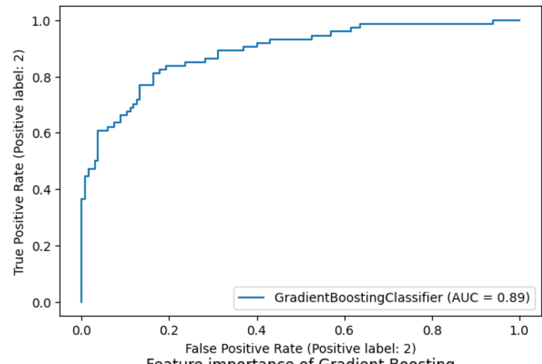
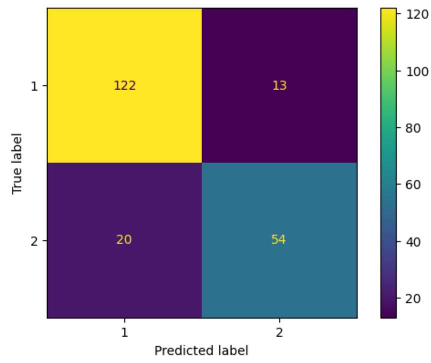
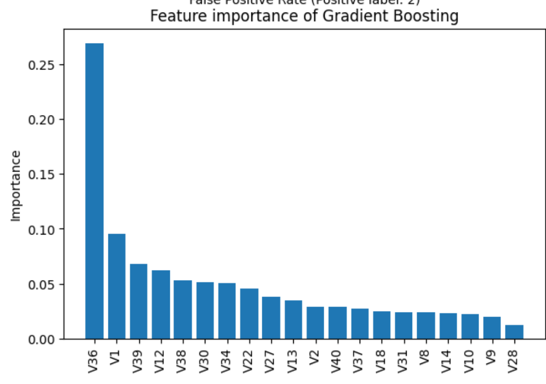


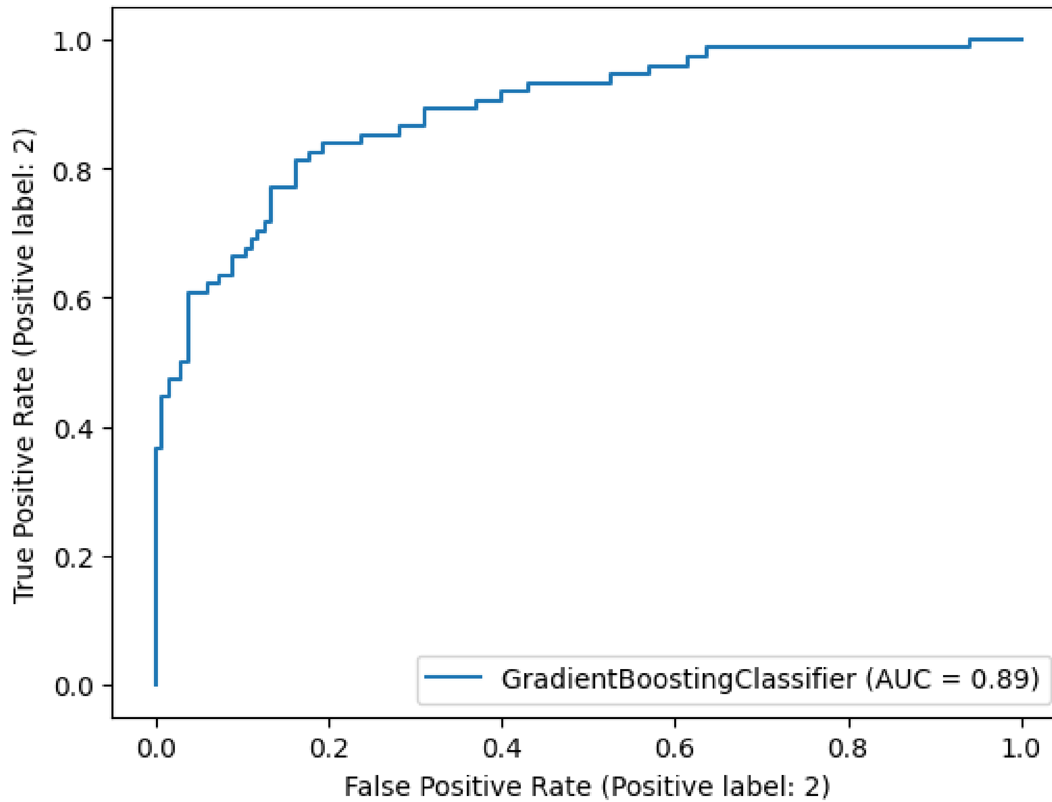
[49]:

```
# Score the gb model with subset of best features
score_gb, model_gb, most_important_features_gb = score_the_model(
    model=GradientBoostingClassifier(),
    model_name='Gradient Boosting',
    random_seed=42,
    X_train=X_train[most_important_features_gb],
    X_test=X_test[most_important_features_gb],
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```



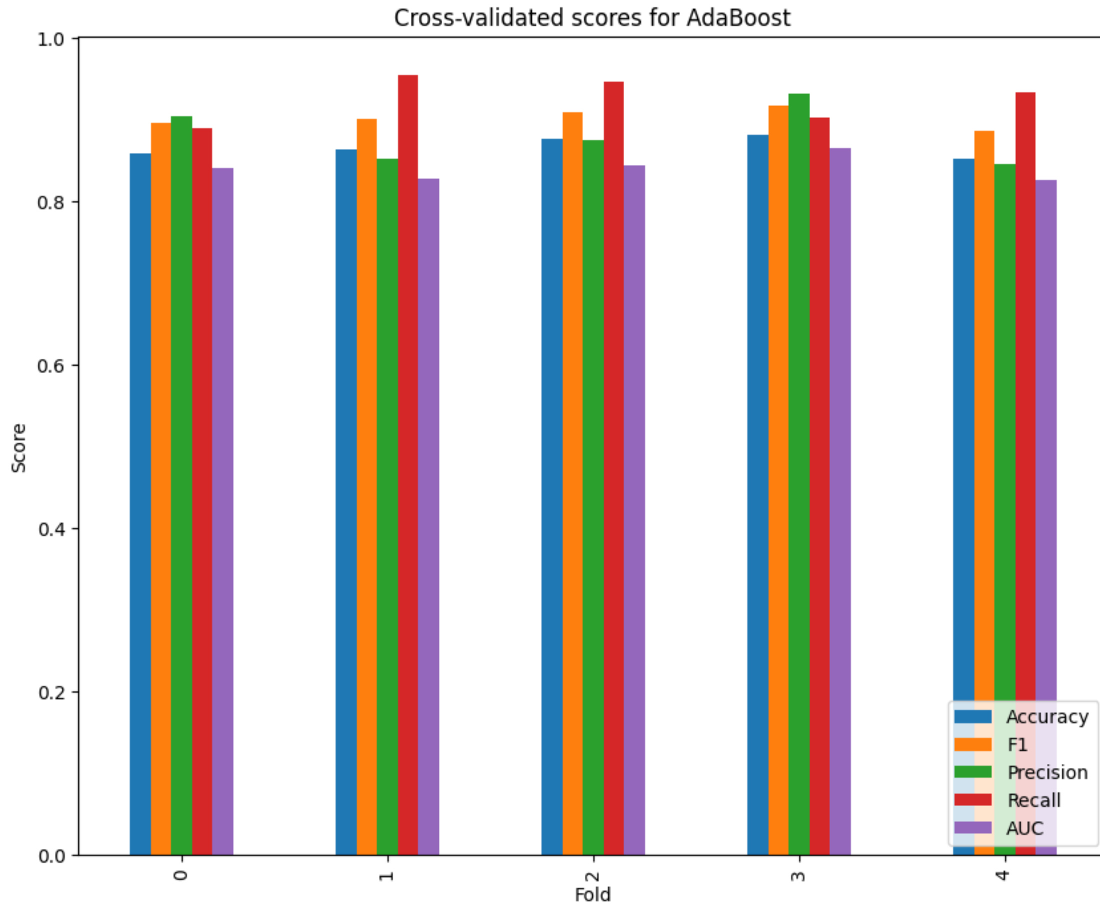

```
Most important features: ['V36', 'V1', 'V39', 'V12', 'V38', 'V30', 'V34', 'V22',
'V27', 'V13', 'V2', 'V40', 'V37', 'V18', 'V31', 'V8', 'V14', 'V10', 'V9', 'V28']
```


Default scoring of Gradient Boosting
Score
0.84
0.88
0.86
0.9
0.82
0.0
0.2
0.4
0.6
0.8
Accuracy
F1
Precision
Recall
AUC

10-fold cross-validated scoring of Gradient Boosting (mean)
0.86
0.9
0.88
0.92
0.84
0.0
0.2
0.4
0.6
0.8
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean

10-fold cross-validated scoring of Gradient Boosting (std)
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
accuracy_std
f1_std
precision_std
recall_std
auc_std

True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
GradientBoostingClassifier (AUC = 0.89)
0.0
0.2
0.4
0.6
0.8
1.0

True label
Predicted label
1
2
122
13
20
54
20
40
60
80
100
120

Feature importance of Gradient Boosting
Importance
0.00
0.05
0.10
0.15
0.20
0.25
V36
V1
V39
V12
V38
V30
V34
V22
V27
V13
V2
V40
V37
V18
V31
V8
V14
V10
V9
V28

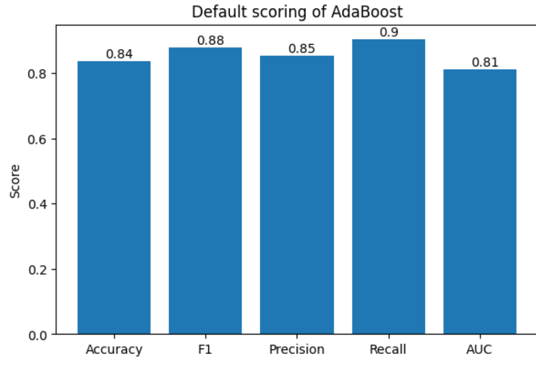


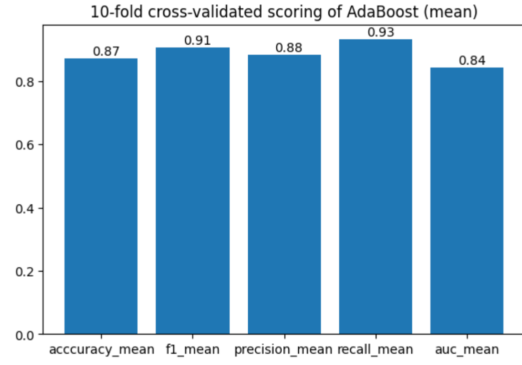
1.2.7 Ada Boost Classifier using RandomForrestClassifier

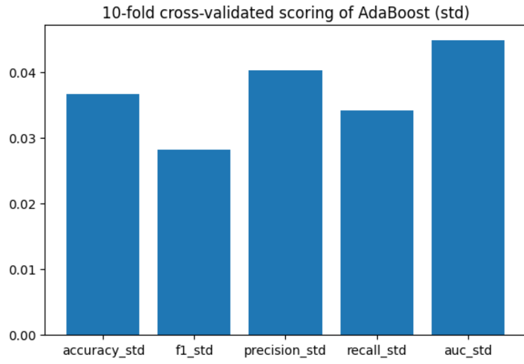
[50]:
```
from sklearn.ensemble import AdaBoostClassifier

# Score the model with default parameters
score_ada, model_ada, most_important_features_ada = score_the_model(
    model=AdaBoostClassifier(
        estimator = RandomForestClassifier(),
        n_estimators=500000,
        learning_rate=0.001,
    ),
    model_name='AdaBoost',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

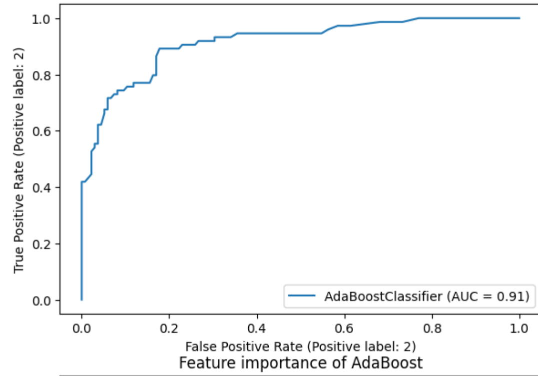


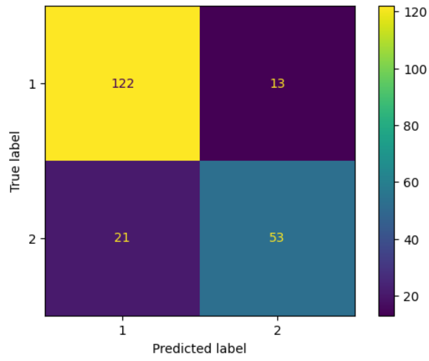

```
Most important features: ['V36', 'V1', 'V39', 'V27', 'V22', 'V12', 'V15', 'V2',
'V37', 'V13', 'V18', 'V30', 'V38', 'V14', 'V17', 'V34', 'V8', 'V10', 'V31',
'V16', 'V3', 'V28', 'V11', 'V7', 'V9', 'V5', 'V40']
```

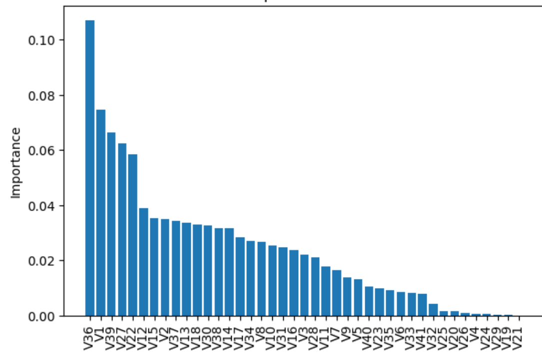

Default scoring of AdaBoost
Score
0.84
0.88
0.85
0.9
0.81
Accuracy
F1
Precision
Recall
AUC

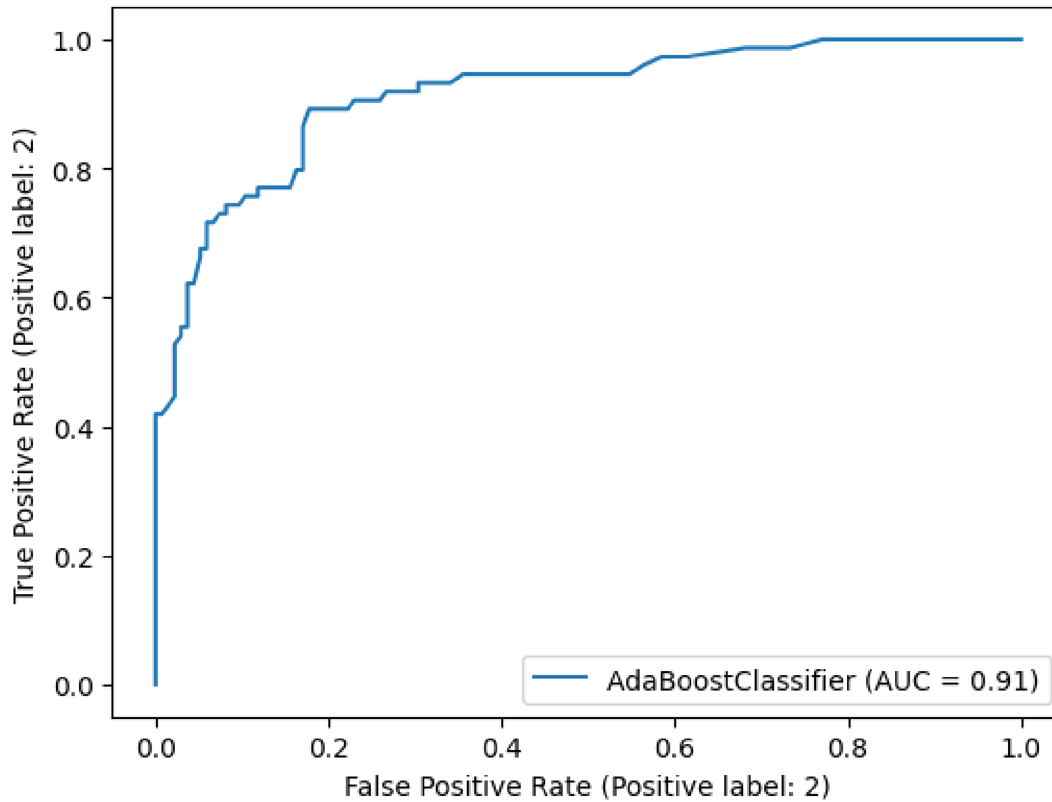

10-fold cross-validated scoring of AdaBoost (mean)
0.87
0.91
0.88
0.93
0.84
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of AdaBoost (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
AdaBoostClassifier (AUC = 0.91)

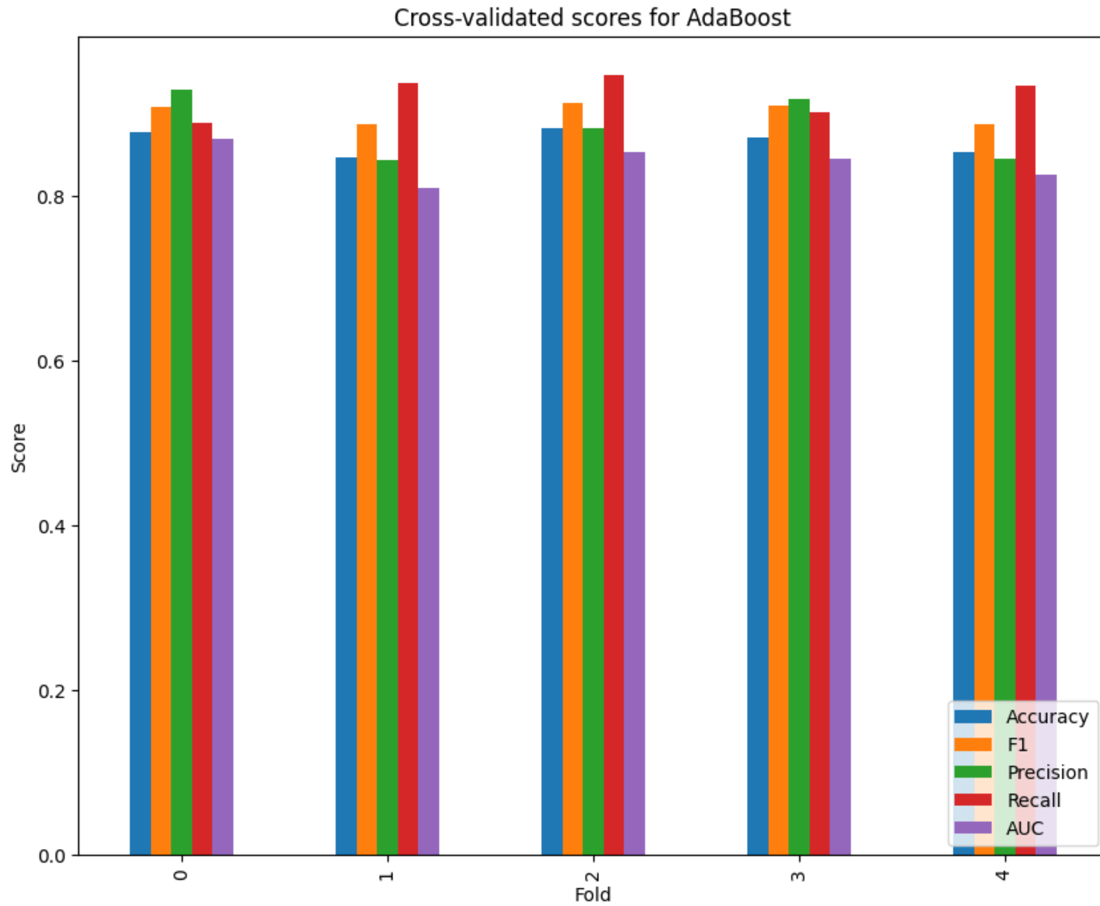

122
13
21
53
True label
Predicted label


Feature importance of AdaBoost
Importance

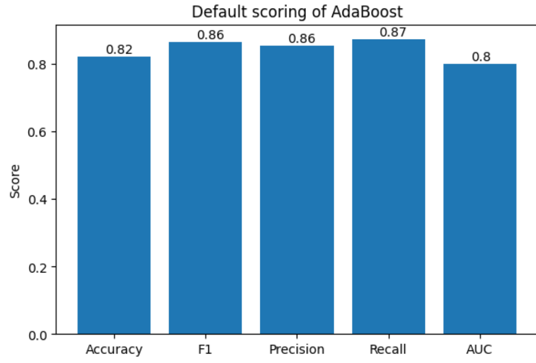


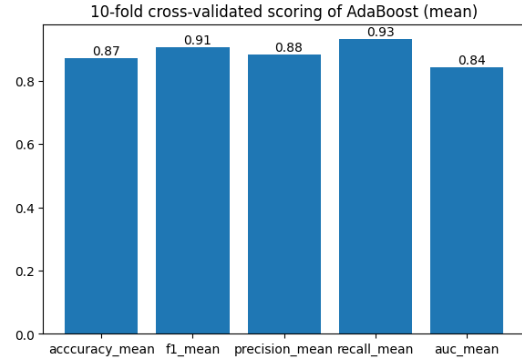
[51]:

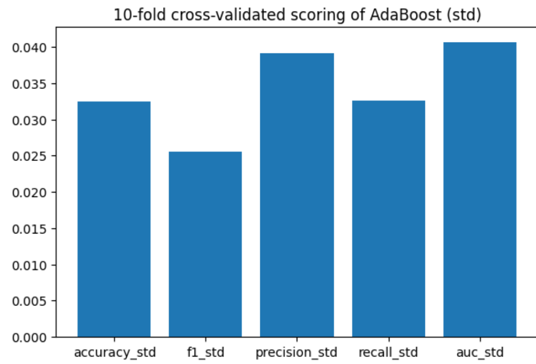
```
# Plot the scores of ada model with subset of best features
score_ada, model_ada, most_important_features_ada = score_the_model(
    model=AdaBoostClassifier(
        estimator = RandomForestClassifier(),
        n_estimators=500000,
        learning_rate=0.001,
    ),
    model_name='AdaBoost',
    random_seed=42,
    X_train=X_train[most_important_features_ada],
    X_test=X_test[most_important_features_ada],
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

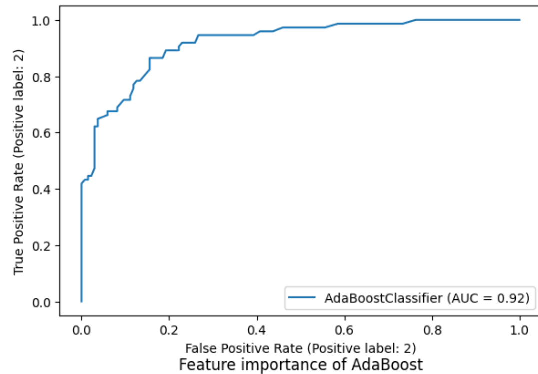


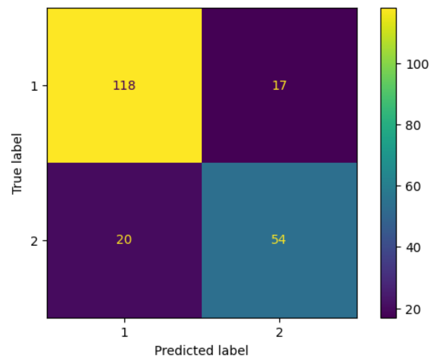

```
Most important features: ['V36', 'V39', 'V1', 'V22', 'V27', 'V15', 'V12', 'V13',
'V37', 'V18', 'V14', 'V2', 'V30', 'V10', 'V31', 'V8', 'V17', 'V38', 'V34',
'V16', 'V28', 'V3', 'V9', 'V5', 'V7', 'V11']
```

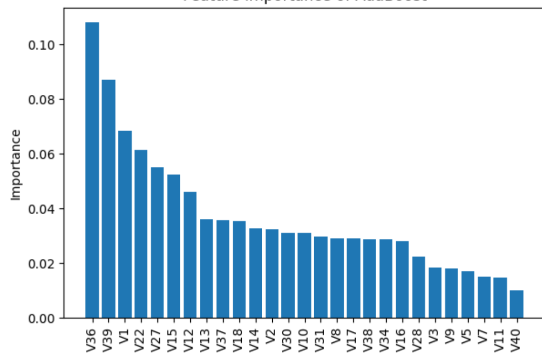

Default scoring of AdaBoost
0.82
0.86
0.86
0.87
0.8
Score
Accuracy
F1
Precision
Recall
AUC

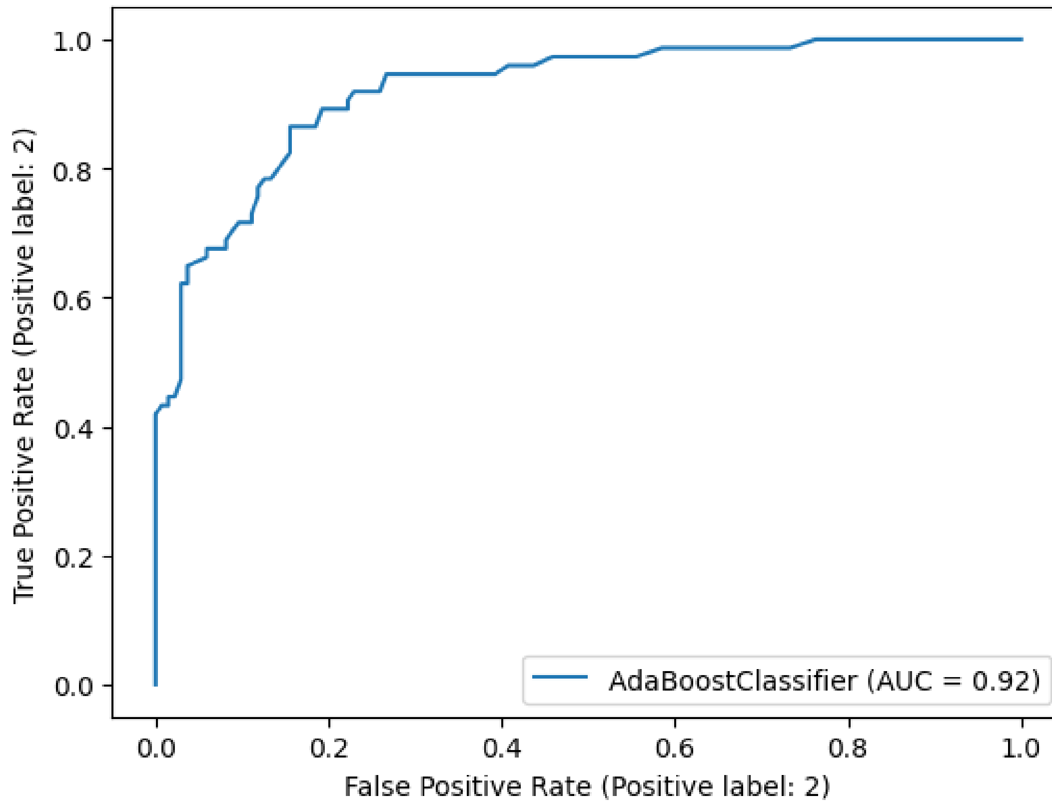

10-fold cross-validated scoring of AdaBoost (mean)
0.87
0.91
0.88
0.93
0.84
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of AdaBoost (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
AdaBoostClassifier (AUC = 0.92)


118
17
20
54
True label
Predicted label

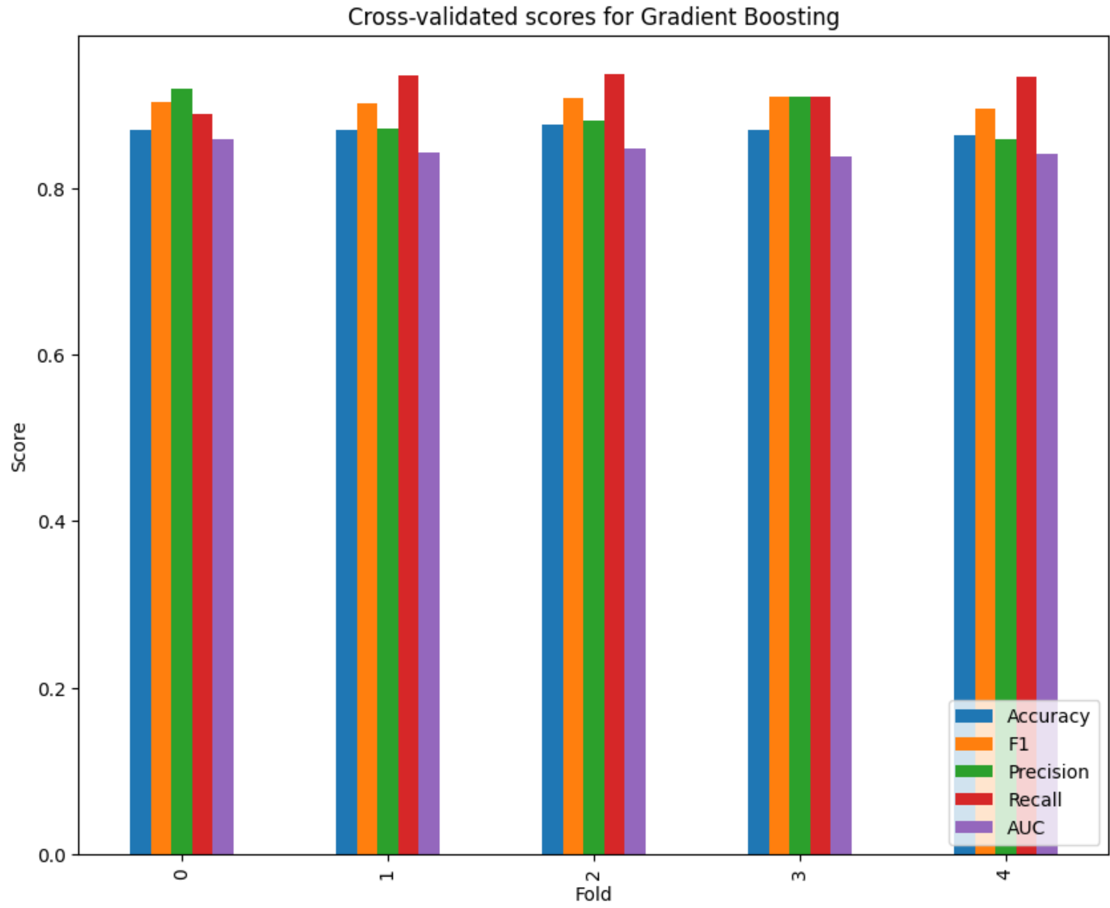

Feature importance of AdaBoost
Importance
V36
V39
V1
V22
V27
V15
V12
V13
V37
V18
V14
V2
V30
V10
V31
V8
V17
V38
V34
V16
V28
V3
V9
V5
V7
V11
V40



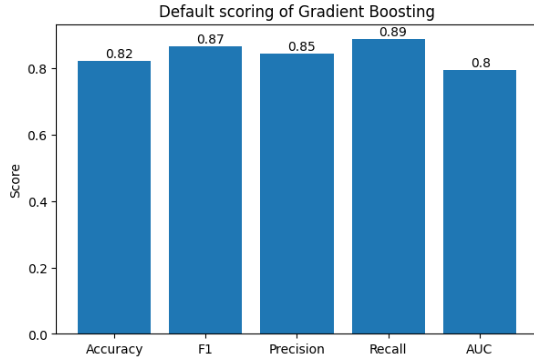
1.2.8 Found using hyper_param.py script

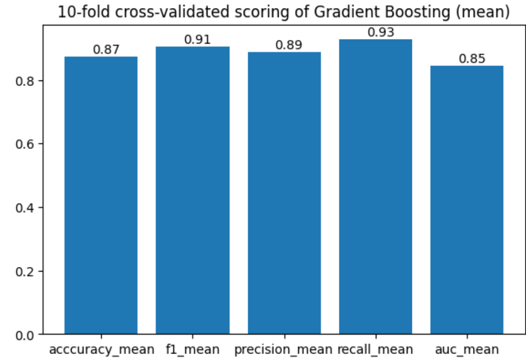
[52]:
```
best_params_gradient_boosting =  {
'validation_fraction': 0.3,
'tol': 0.0001,
'subsample': 0.9,
'n_estimators': 50,
'min_samples_split': 5,
'min_samples_leaf': 5,
'max_features': 'log2',
'max_depth': 10,
'learning_rate': 0.1
}

best_params_grad_boost_scores, best_params_grad_boost_model,␣
↪best_params_grad_boost = score_the_model(
    model=GradientBoostingClassifier(**best_params_gradient_boosting),
    model_name='Gradient Boosting',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
```

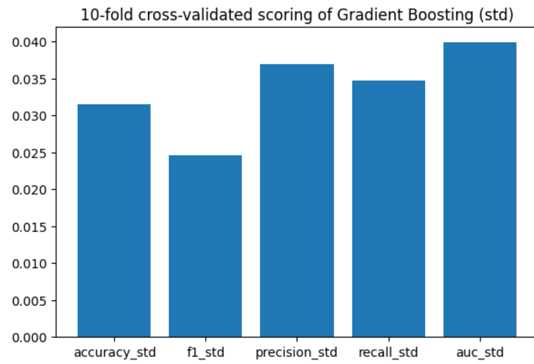
```
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

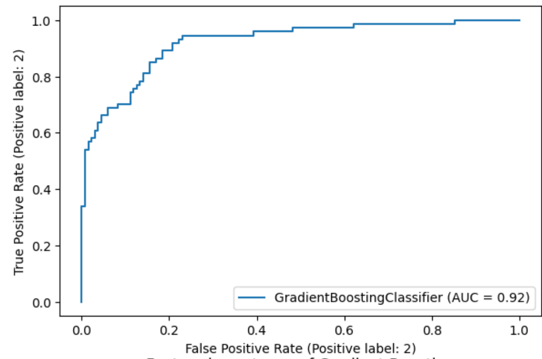


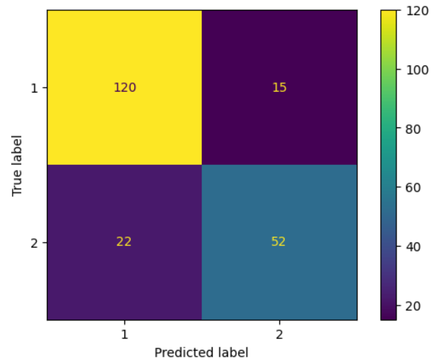

```
Most important features: ['V39', 'V27', 'V36', 'V1', 'V12', 'V22', 'V15', 'V8',
'V37', 'V13', 'V18', 'V2', 'V34', 'V31', 'V38', 'V17', 'V30', 'V5', 'V10', 'V9',
'V16', 'V14', 'V28', 'V11', 'V3', 'V6', 'V7']
```

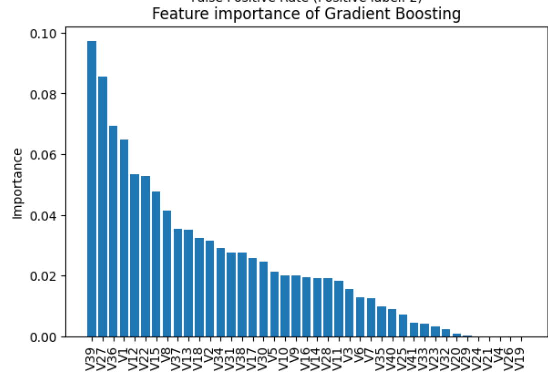

Default scoring of Gradient Boosting
Score
0.82
0.87
0.85
0.89
0.8
Accuracy
F1
Precision
Recall
AUC

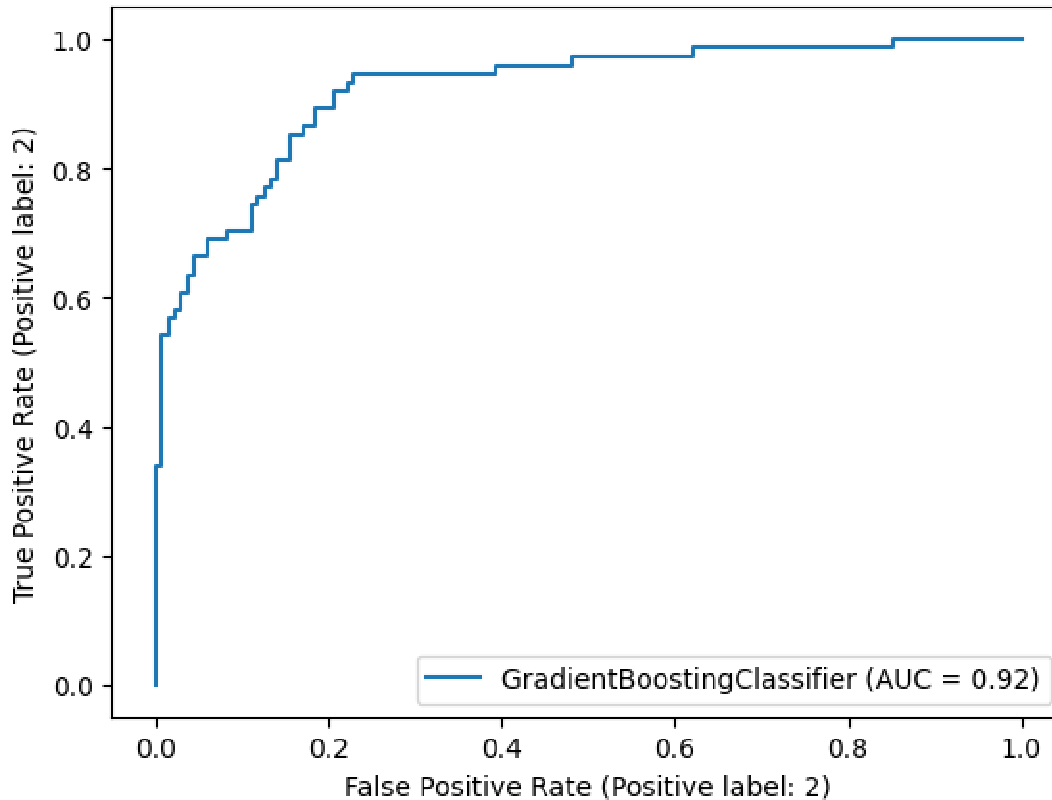

10-fold cross-validated scoring of Gradient Boosting (mean)
0.87
0.91
0.89
0.93
0.85
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Gradient Boosting (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
GradientBoostingClassifier (AUC = 0.92)


120
15
22
52
True label
Predicted label


Feature importance of Gradient Boosting
Importance
V39 V27 V36 V1 V12 V22 V15 V8 V37 V13 V18 V2 V34 V31 V38 V17 V30 V5 V10 V9 V16 V14 V28 V11 V3 V6 V7 V35 V40 V25 V41 V33 V23 V32 V20 V29 V24 V21 V4 V26 V19

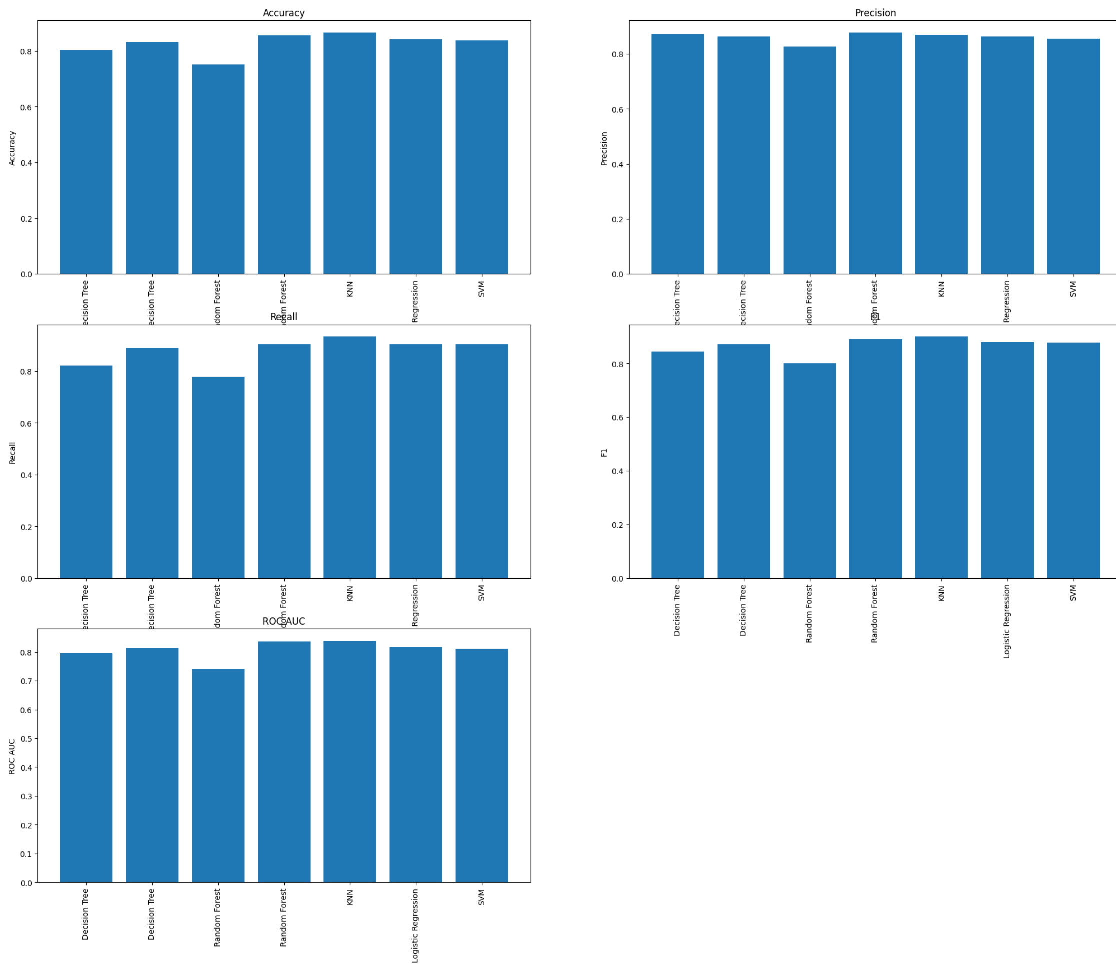


1.2.9 Comparisson between models performances

[53]:
```
# Plot Scores of all models
all_accuracy = [score['Accuracy'] for score in all_scores]
all_precision = [score['Precision'] for score in all_scores]
all_recall = [score['Recall'] for score in all_scores]
all_f1 = [score['F1'] for score in all_scores]
all_roc_auc = [score['AUC'] for score in all_scores]
model_names = [score['model_name'] for score in all_scores]

fig, ax = plt.subplots(3, 2, figsize=(25, 20))
fig.suptitle('Scores of all models', fontsize=20)

ax[0, 0].bar(model_names, all_accuracy)
ax[0, 0].set_title('Accuracy')
ax[0, 0].set_ylabel('Accuracy')
ax[0, 0].set_xticklabels(model_names, rotation=90)

ax[0, 1].bar(model_names, all_precision)
ax[0, 1].set_title('Precision')
ax[0, 1].set_ylabel('Precision')
```

```
ax[0, 1].set_xticklabels(model_names, rotation=90)

ax[1, 0].bar(model_names, all_recall)
ax[1, 0].set_title('Recall')
ax[1, 0].set_ylabel('Recall')
ax[1, 0].set_xticklabels(model_names, rotation=90)

ax[1, 1].bar(model_names, all_f1)
ax[1, 1].set_title('F1')
ax[1, 1].set_ylabel('F1')
ax[1, 1].set_xticklabels(model_names, rotation=90)

ax[2, 0].bar(model_names, all_roc_auc)
ax[2, 0].set_title('ROC AUC')
ax[2, 0].set_ylabel('ROC AUC')
ax[2, 0].set_xticklabels(model_names, rotation=90)

ax[2, 1].set_visible(False)

plt.show()
```

Scores of all models

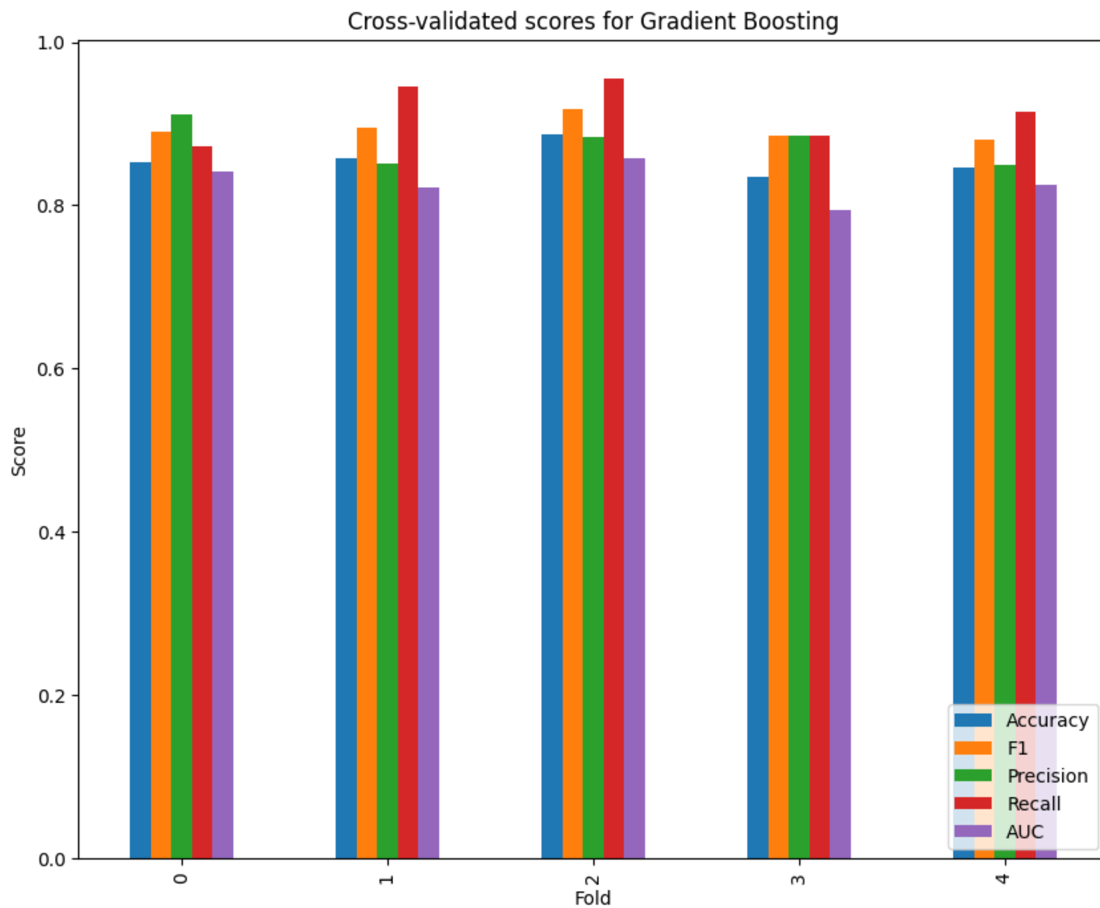


1.2.10 Ok what about converting numerical features into categorical?

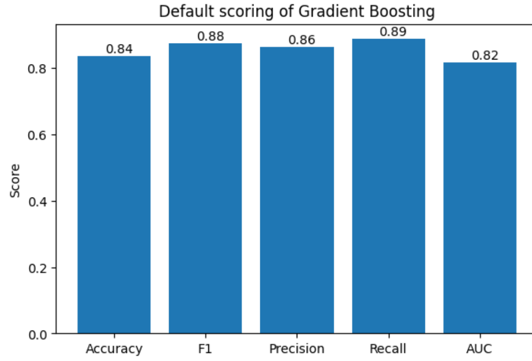
[54]:
```python
# import one hot encoder
from sklearn.preprocessing import OneHotEncoder

# Select the numerical columns, but should not have more than 20 unique values
numerical_cols = [cname for cname in X_train.columns if
                X_train[cname].dtype in ['int64', 'float64'] and X_train[cname].
 ↪nunique() < 20]

# Now onehotencode the numerical columns
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)

encoder_df = pd.DataFrame(OH_encoder.fit_transform(df_train[numerical_cols]))
```

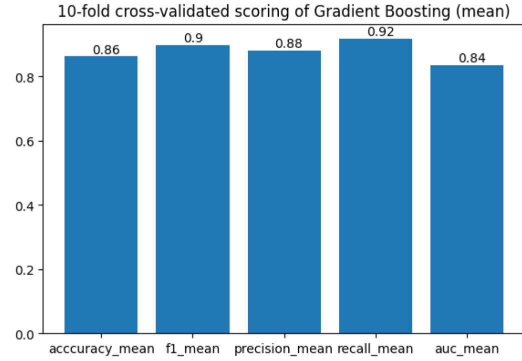
```
df_train = pd.concat([df_train, encoder_df], axis=1)

encoder_df = pd.DataFrame(OH_encoder.fit_transform(df_test[numerical_cols]))
df_test = pd.concat([df_test, encoder_df], axis=1)

# Drop the numerical columns
df_train = df_train.drop(numerical_cols, axis=1)
df_test = df_test.drop(numerical_cols, axis=1)
```

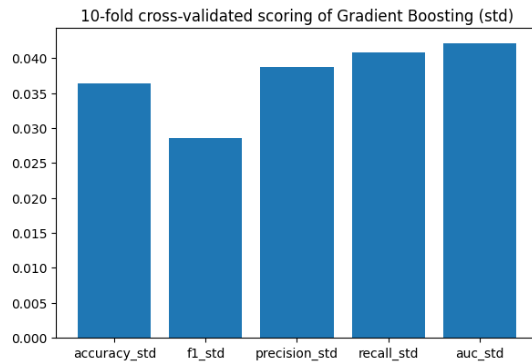
[55]:
```
# Refit the gradient boosting model with the new data
best_params_grad_boost_scores, best_params_grad_boost_model,␣
↪best_params_grad_boost = score_the_model(
    model=GradientBoostingClassifier(),
    model_name='Gradient Boosting',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

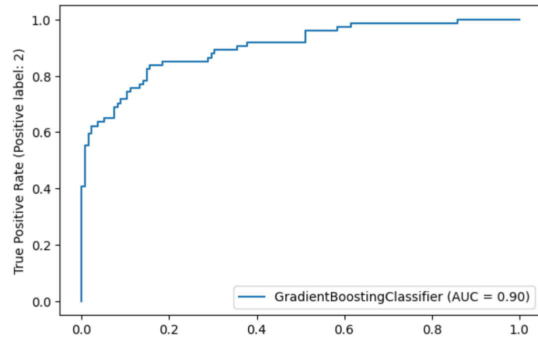


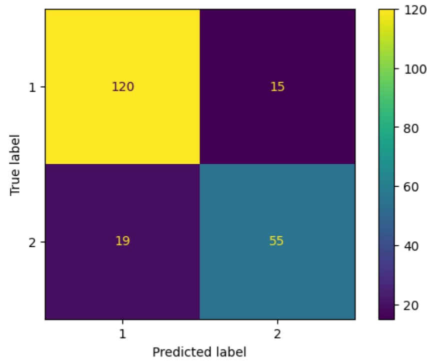
```
Most important features: ['V36', 'V1', 'V12', 'V39', 'V38', 'V34', 'V30', 'V22',
'V27', 'V13', 'V2', 'V14', 'V40', 'V8', 'V37', 'V18', 'V9', 'V28', 'V10', 'V31']
```

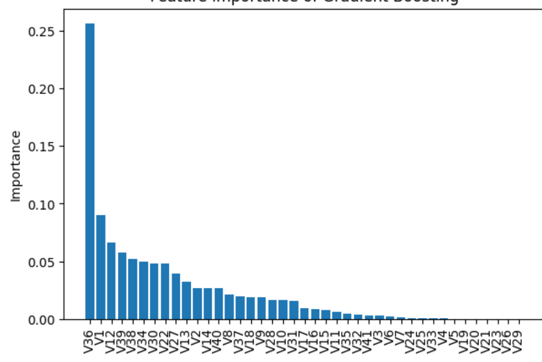


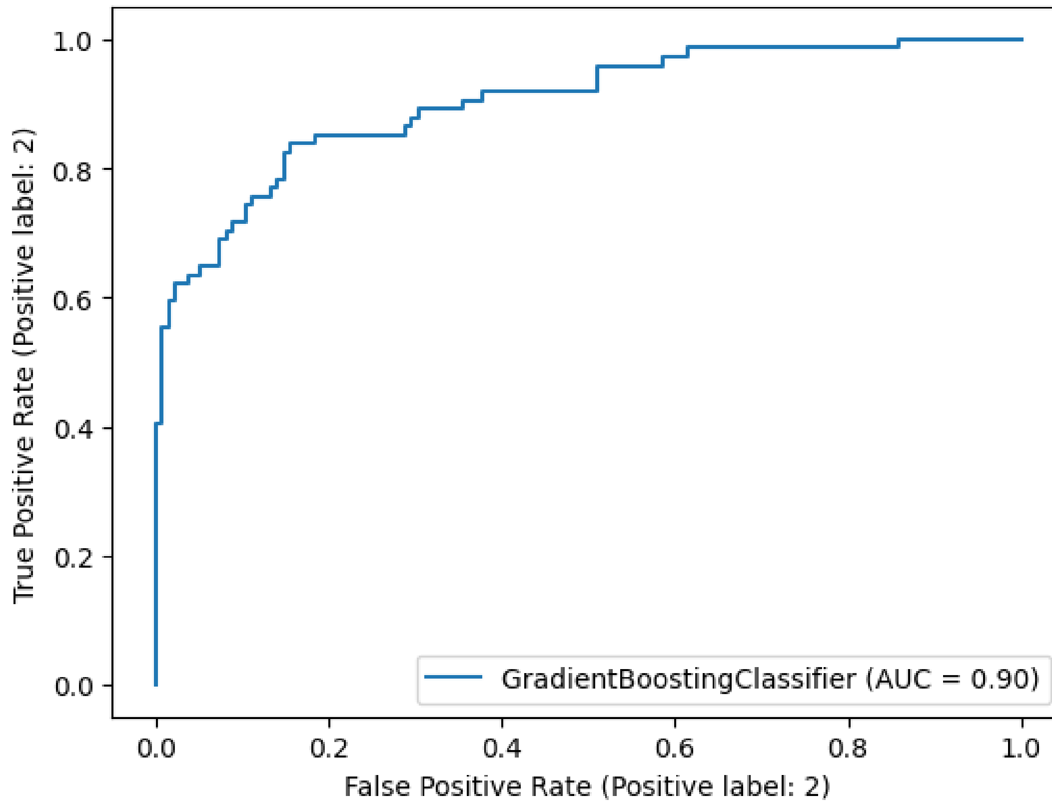






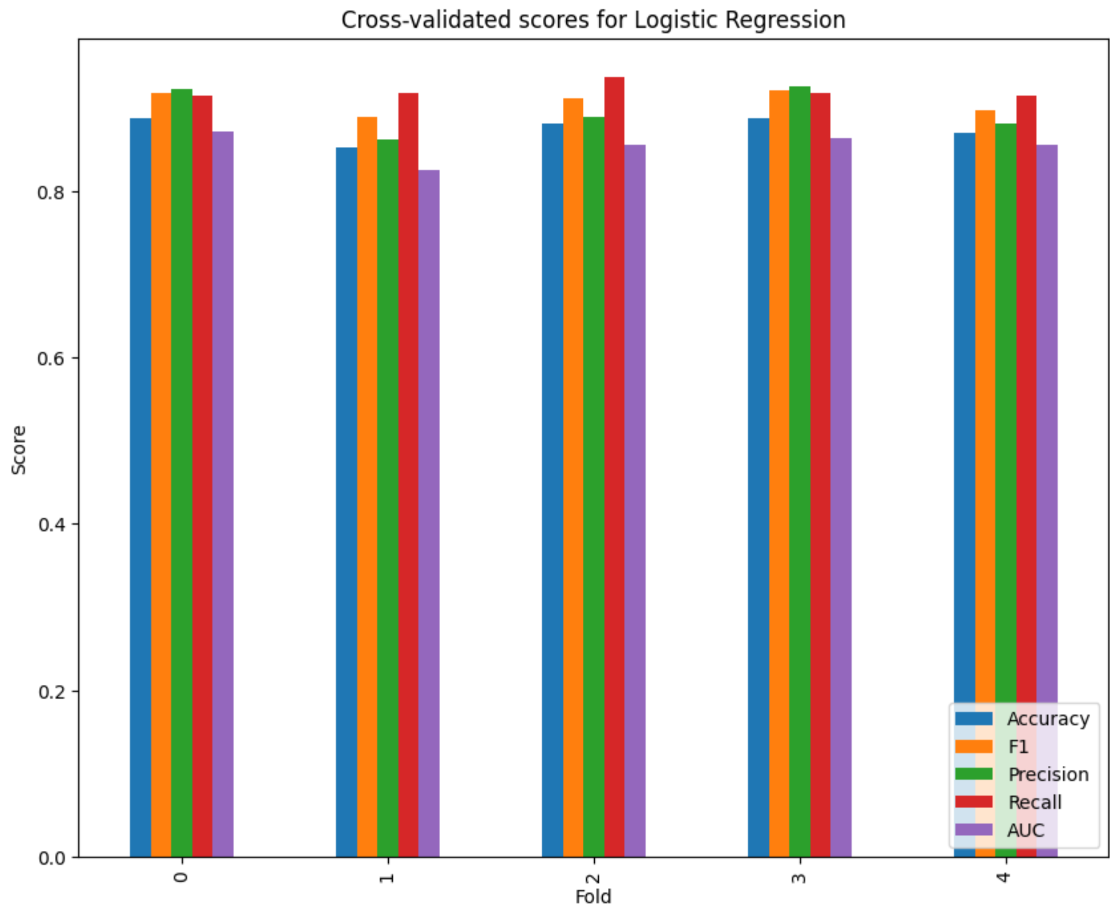


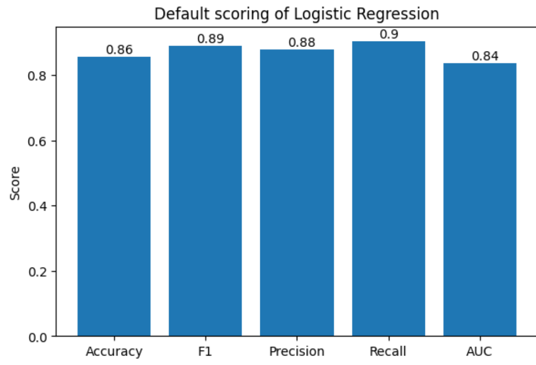


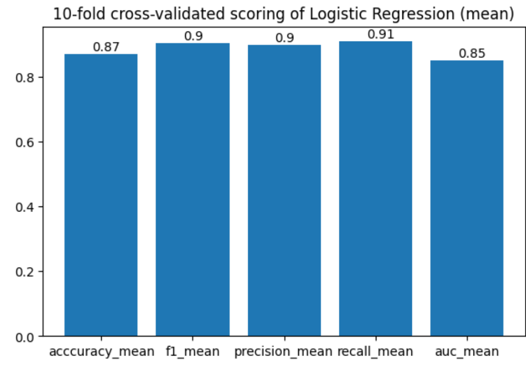


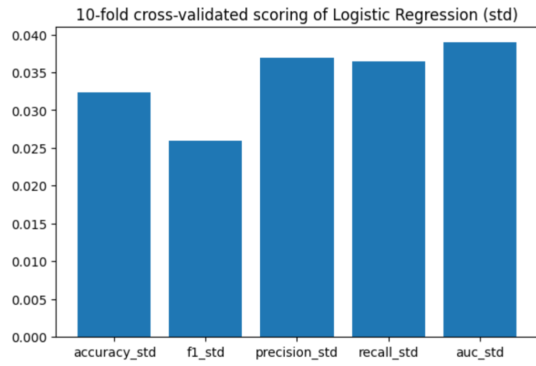
[56]:

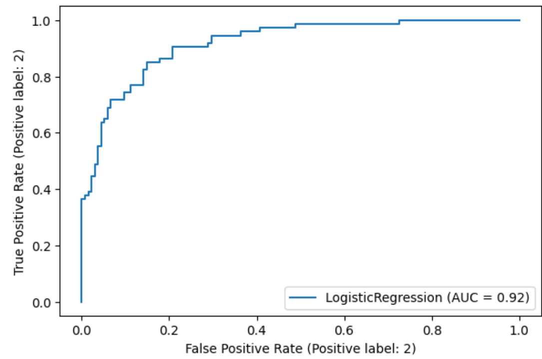
```
# What about logistic regression?

# Score the model with default parameters
score_log_reg, model_log_reg,_ = score_the_model(
    model=LogisticRegression(max_iter=100),
    model_name='Logistic Regression',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

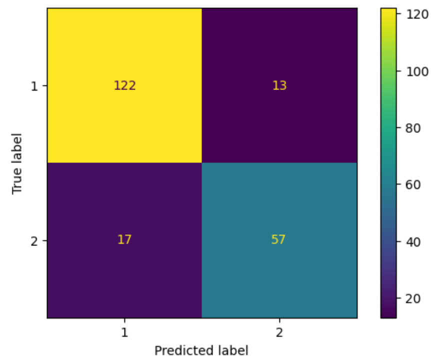

Cross-validated scores for Logistic Regression
Score
0.0
0.2
0.4
0.6
0.8
Fold
0
1
2
3
4
Accuracy
F1
Precision
Recall
AUC

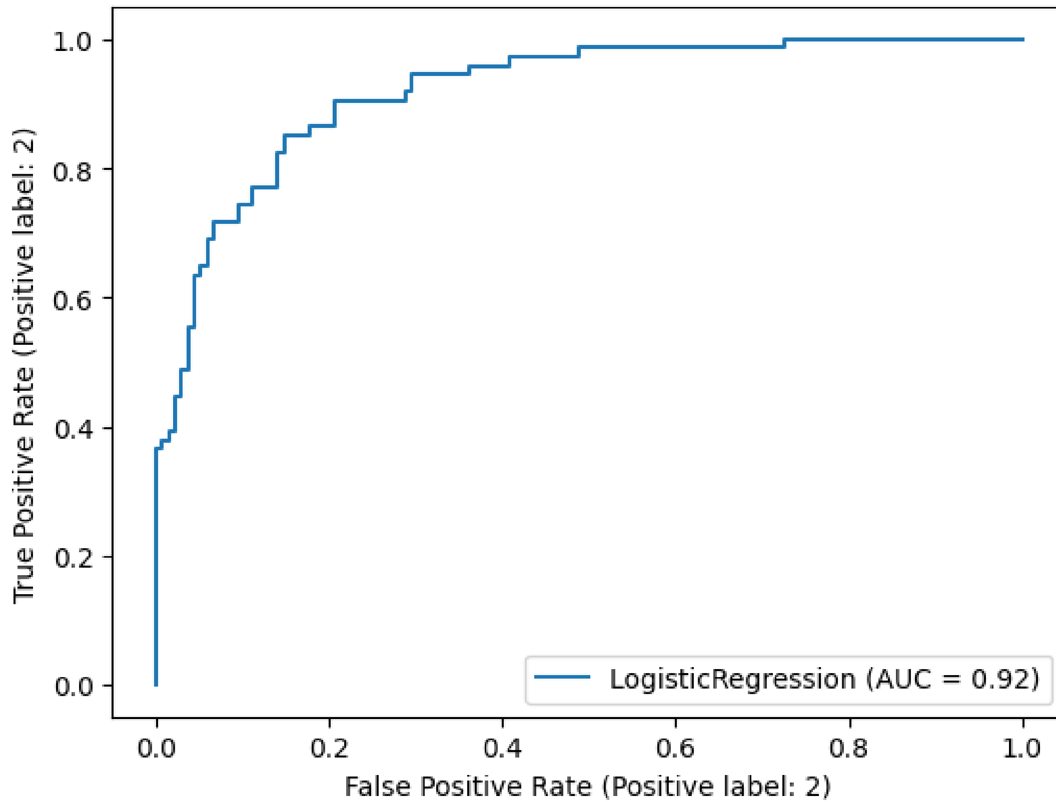

Default scoring of Logistic Regression
Score
0.86
0.89
0.88
0.9
0.84
Accuracy
F1
Precision
Recall
AUC


10-fold cross-validated scoring of Logistic Regression (mean)
0.87
0.9
0.9
0.91
0.85
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Logistic Regression (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std

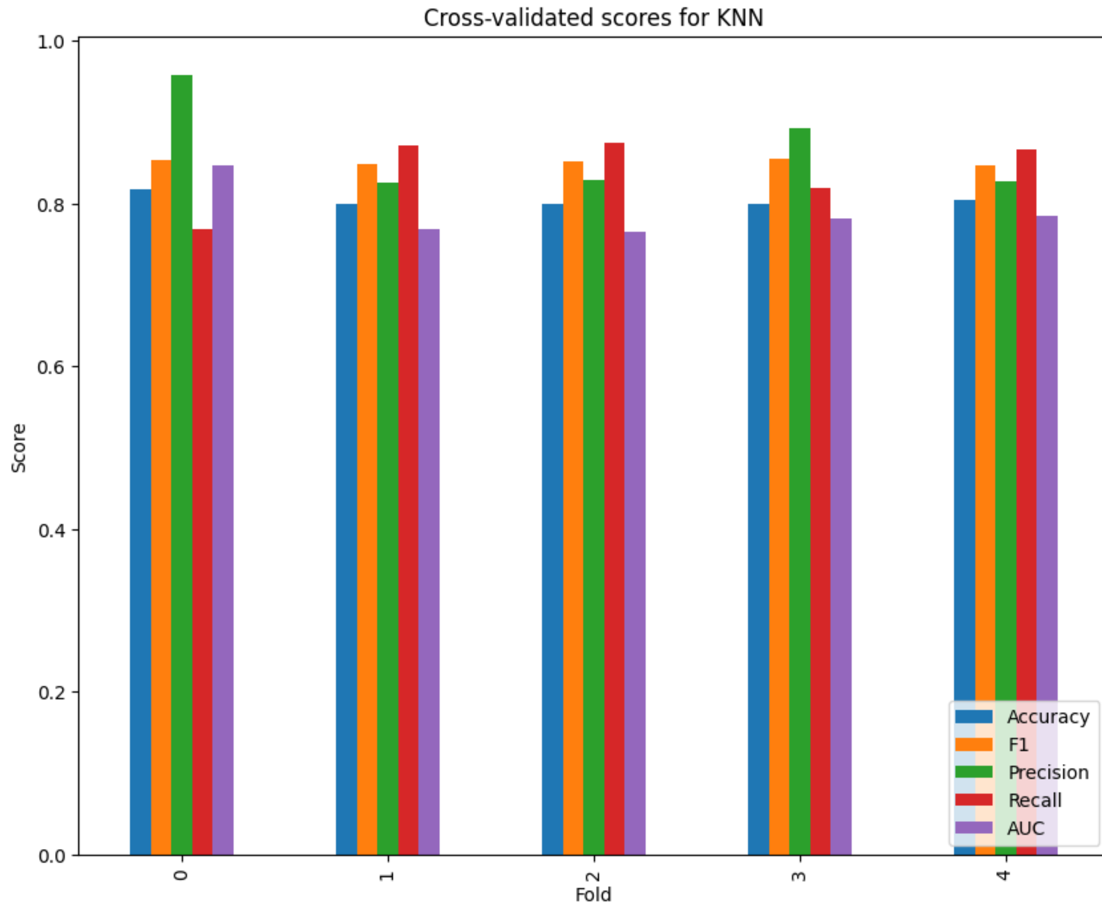

True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
LogisticRegression (AUC = 0.92)

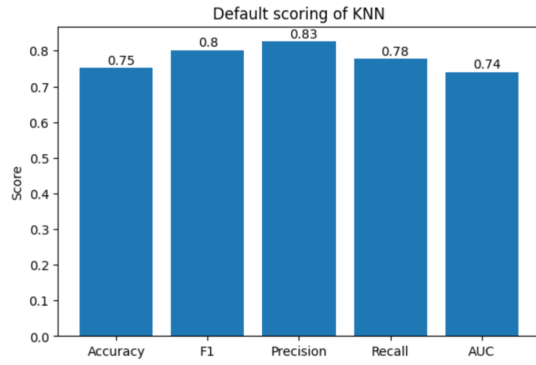

122
13
17
57
True label
Predicted label

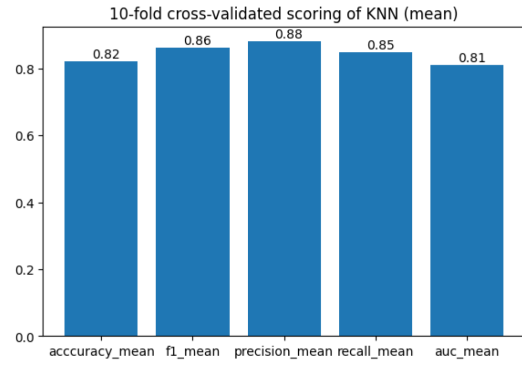


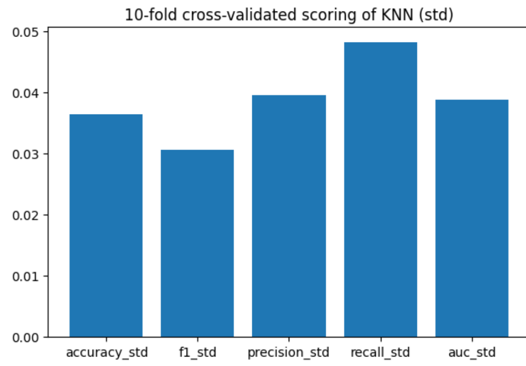
[57]:

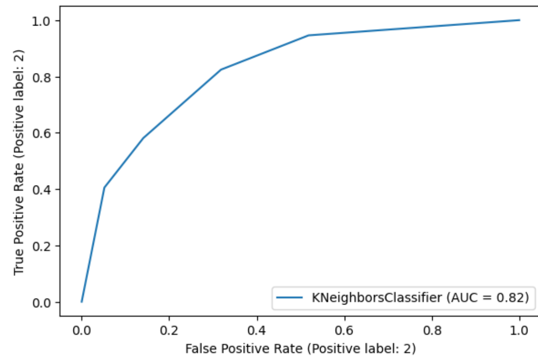
```
# KNN should score much better now
score_knn, model_knn, _ = score_the_model(
    model=KNeighborsClassifier(),
    model_name='KNN',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

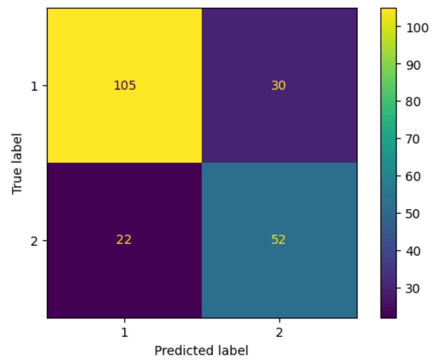

Cross-validated scores for KNN
Score
Fold
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
Accuracy
F1
Precision
Recall
AUC

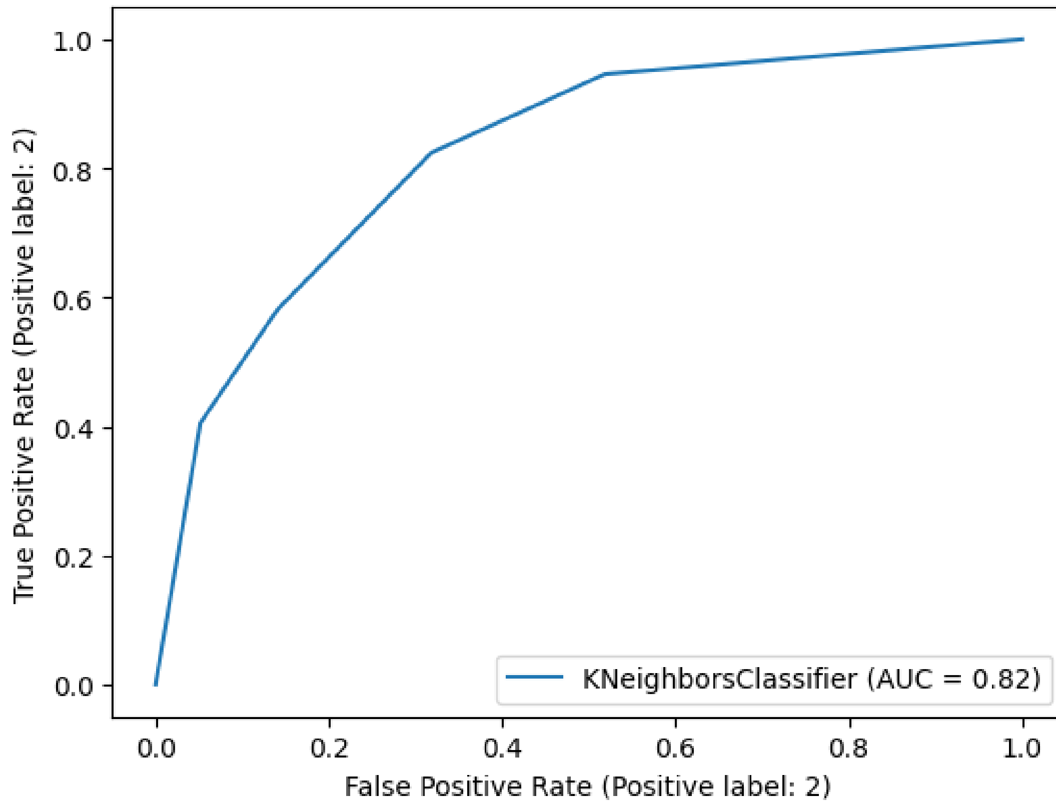

Default scoring of KNN
Score
0.75
0.8
0.83
0.78
0.74
Accuracy
F1
Precision
Recall
AUC


10-fold cross-validated scoring of KNN (mean)
0.82
0.86
0.88
0.85
0.81
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of KNN (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std

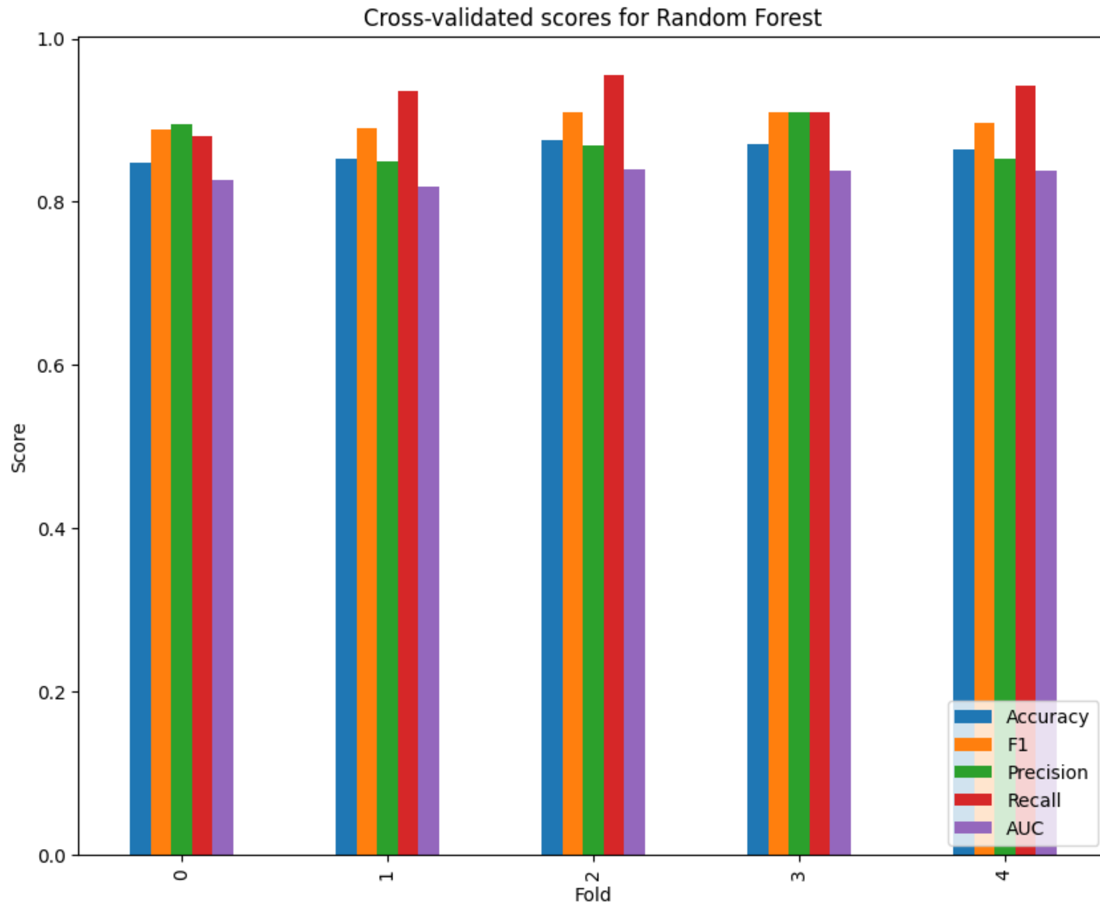

True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
KNeighborsClassifier (AUC = 0.82)


105
30
22
52
True label
Predicted label

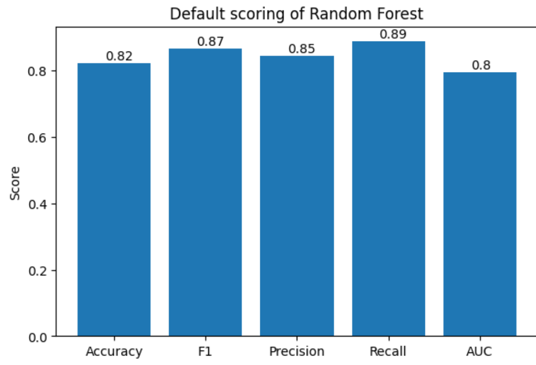


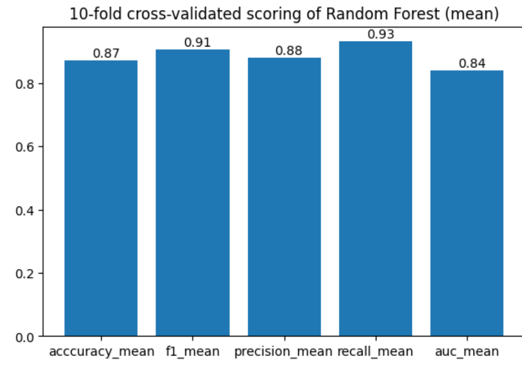
[58]:

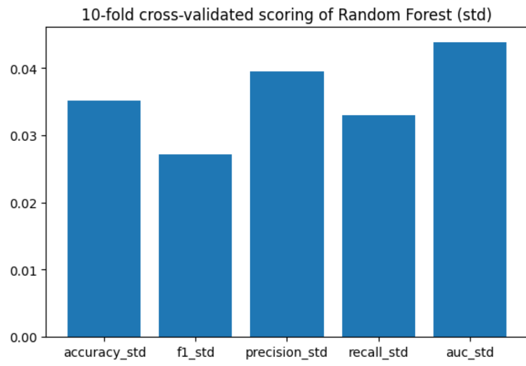
```
# Select the best features
score_rf, model_rf, most_important_features_rf = score_the_model(
    model=RandomForestClassifier(),
    model_name='Random Forest',
    random_seed=42,
    X_train=X_train,
    X_test=X_test,
    y_train=y_train,
    y_test=y_test,
    plot=True
)
```

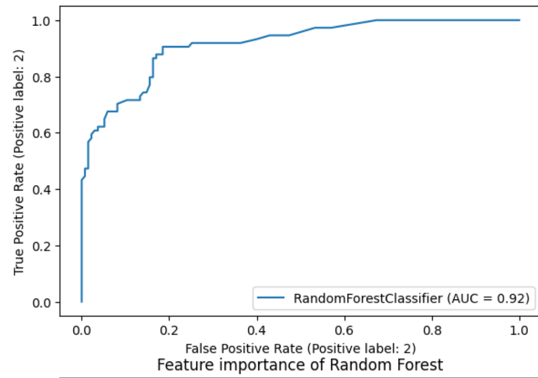


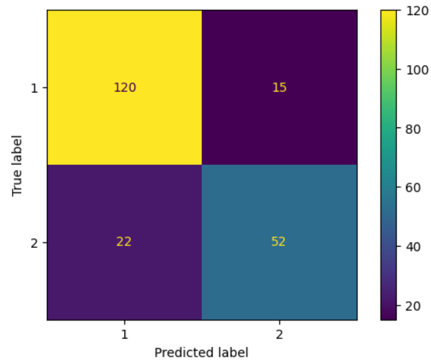

```
Most important features: ['V36', 'V39', 'V22', 'V27', 'V1', 'V12', 'V15', 'V13',
'V18', 'V34', 'V14', 'V37', 'V30', 'V2', 'V17', 'V31', 'V8', 'V16', 'V3', 'V38',
'V10', 'V28', 'V9', 'V11', 'V5', 'V7', 'V6', 'V41']
```

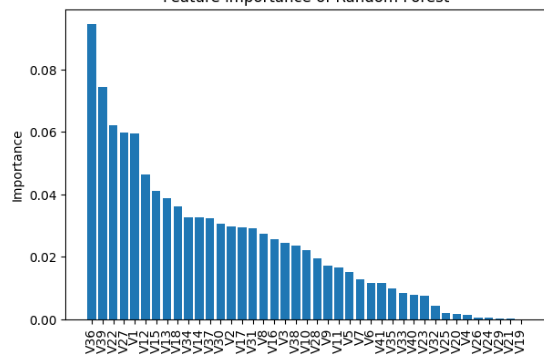

Default scoring of Random Forest
Score
0.82
0.87
0.85
0.89
0.8
Accuracy
F1
Precision
Recall
AUC

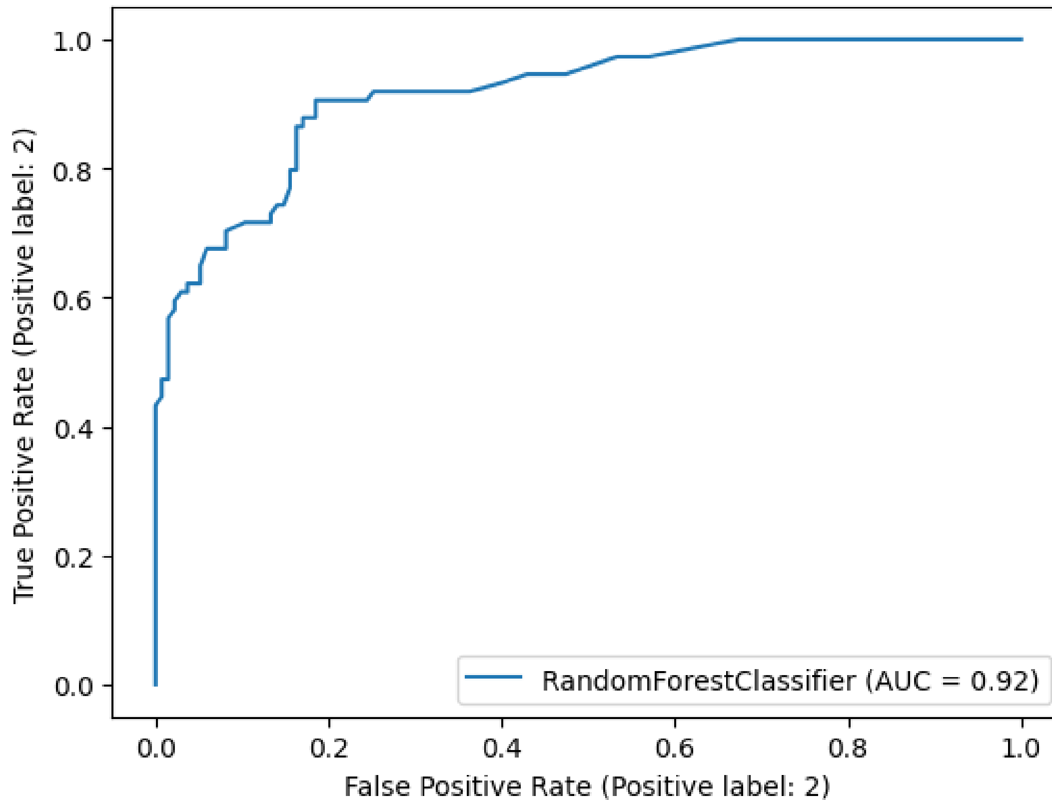

10-fold cross-validated scoring of Random Forest (mean)
0.87
0.91
0.88
0.93
0.84
acccuracy_mean
f1_mean
precision_mean
recall_mean
auc_mean


10-fold cross-validated scoring of Random Forest (std)
accuracy_std
f1_std
precision_std
recall_std
auc_std


True Positive Rate (Positive label: 2)
False Positive Rate (Positive label: 2)
RandomForestClassifier (AUC = 0.92)


True label
Predicted label
120
15
22
52


Feature importance of Random Forest
Importance
V36 V39 V22 V27 V1 V12 V15 V13 V18 V34 V14 V37 V30 V2 V17 V31 V8 V16 V3 V38 V10 V28 V9 V11 V5 V7 V6 V41 V35 V33 V40 V23 V32 V25 V20 V4 V26 V24 V29 V21 V19



1.2.11 2.3 Evaluation

Given that the data set is not in the "big data" category, implement a cross-validation procedure based on five folds (approximately equal sized) of your data. Furthermore, repeat the experiment 10 times with different folds and average the results (include standard deviation). You are expected to report the following metrics: - F1 - Precision - Recall - AUC Comment on the performance of algorithms and visualize their final scores. How do they perform against the random baseline? What about the constant one? How do different learning scenarios impact the final score? Are the differences between the models statistically significant?

1.2.12 F1 score

The F1 score is a metric that combines precision and recall. It is often used in classification tasks as a way to balance the two metrics, as it can be difficult to optimize for both at the same time. To compute the F1 score, you first need to calculate the precision and recall for a given model.

Precision P = TP/(TP + FP)

Recall R = TP/ (TP + FN)

Once you have calculated precision and recall, the F1 score is simply the harmonic mean of the two, computed using the following formula:

F1 = 2 * (precision * recall) / (precision + recall)

The F1 score ranges from 0 to 1, with a higher score indicating better performance. A perfect score is achieved when the precision and recall are both 1.

1.2.13 Precision score

Precision is a metric that measures the accuracy of a classifier when it predicts the positive class. It is defined as the number of true positive predictions made by the classifier, divided by the total number of positive predictions made by the classifier.

In other words, precision is a measure of the proportion of positive predictions that are actually correct. It is a useful metric to consider when the cost of false positives is high, such as in cases where the classifier is being used to make important decisions (e.g. medical diagnosis, fraud detection).

Precision = True Positives / (True Positives + False Positives)

1.2.14 Recall score

Recall is a metric that measures the ability of a classifier to detect all instances of the positive class. It is defined as the number of true positive predictions made by the classifier, divided by the total number of actual positive cases in the data.

In other words, recall is a measure of the proportion of actual positive cases that the classifier is able to identify. It is a useful metric to consider when the cost of false negatives is high, such as in cases where it is important to identify all instances of the positive class (e.g. cancer diagnosis, intrusion detection).

Recall = True Positives / (True Positives + False Negatives)

1.2.15 AUC score

The AUC is calculated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various classification thresholds. The TPR is defined as the number of true positive predictions made by the classifier, divided by the total number of actual positive cases in the data. The FPR is defined as the number of false positive predictions made by the classifier, divided by the total number of actual negative cases in the data.

The AUC is then calculated by computing the area under this curve. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 indicates a classifier that is no better than random.

The AUC can be calculated using the following formula:

AUC = (TPR1 - TPR0) + (TPR2 - TPR1) + … + (TPRn - TPRn-1)

where TPRi is the TPR at the ith classification threshold and TPRi-1 is the TPR at the previous classification threshold.

1.3 Report and presentation

The assignment has to be submitted in the form of two files: a markdown file and a PDF file created from the R Studio markdown file (in RStudio → file - new file - R Markdown), where you write both the code, as well as the text of answers (echo = T option must be enabled for each code block). Markdown files can easily be exported to PDF using ("Knit") button in R Studio. If you are using Python, you can produce a similar report with Jupyter Notebook.