CodeLab 3: Classifacton of WLANs power saving

Energy management is important for electronic devices, and doubly so for portable systems that run on battery power. For this Code Lab, we consider the problem of controlling the power state of the wireless network interface controller. For optimal performance, this power state should adapt to the network usage. We will use a dataset published alongside a paper by Saeed and Kolberg, 2018. The researchers measured the traffic signatures associated with a number of applications, on a mobile phone and labelled each according to the WLAN usage pattern. The dataset comprises 1350 samples with 6 features and a single classification label. The label (the type of usage) was assigned one of four categories: high, varied, buffer, or low.

In this Codelab we will start with binary classification, in which we will see the difference between weighted and non-weighted binary classification. Secondly, we will develop some multiclass models. There we will see the difference between one-versus-all and multinomial classification. Various classification metrics will be used to assess the models' prediction capabilities.

In [1]:
```
# Import packages needed for the lab
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
from sklearn import svm
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, roc_curve, precision_recal
from sklearn.model_selection import train_test_split
```

In this lab, we will work on different classifiers for different classification models. Even though the data is already prepared, it is important to get familiar with the data you will work with.
−Load "Network_Traffic.csv" into Pandas data frame.

In [2]:
```
# Import the data and visualize it
data = pd.read_csv('Network_Traffic.csv')
dataSet = pd.DataFrame(data)
print(dataSet.columns)
```

```
Index(['Receiving-data-rate-in-Kbytes-per-seconds',
       ' Transmitting-data-rate-in-Kbytes-per-seconds',
       ' Total-received-Kbytes', ' Total-Transmitted-Kbytes',
       ' Total-received-packets', ' Total-Transmitted-packets', ' Class'],
      dtype='object')
```

Task 1 - Data Preparation

- Split the data frame into features data frame X and label data frame y.

In [3]:
```
# Seperate features from classes
X = dataSet.drop(" Class",axis=1)
y = dataSet[" Class"]
```

-Calculate and print the correlation matrix of X.

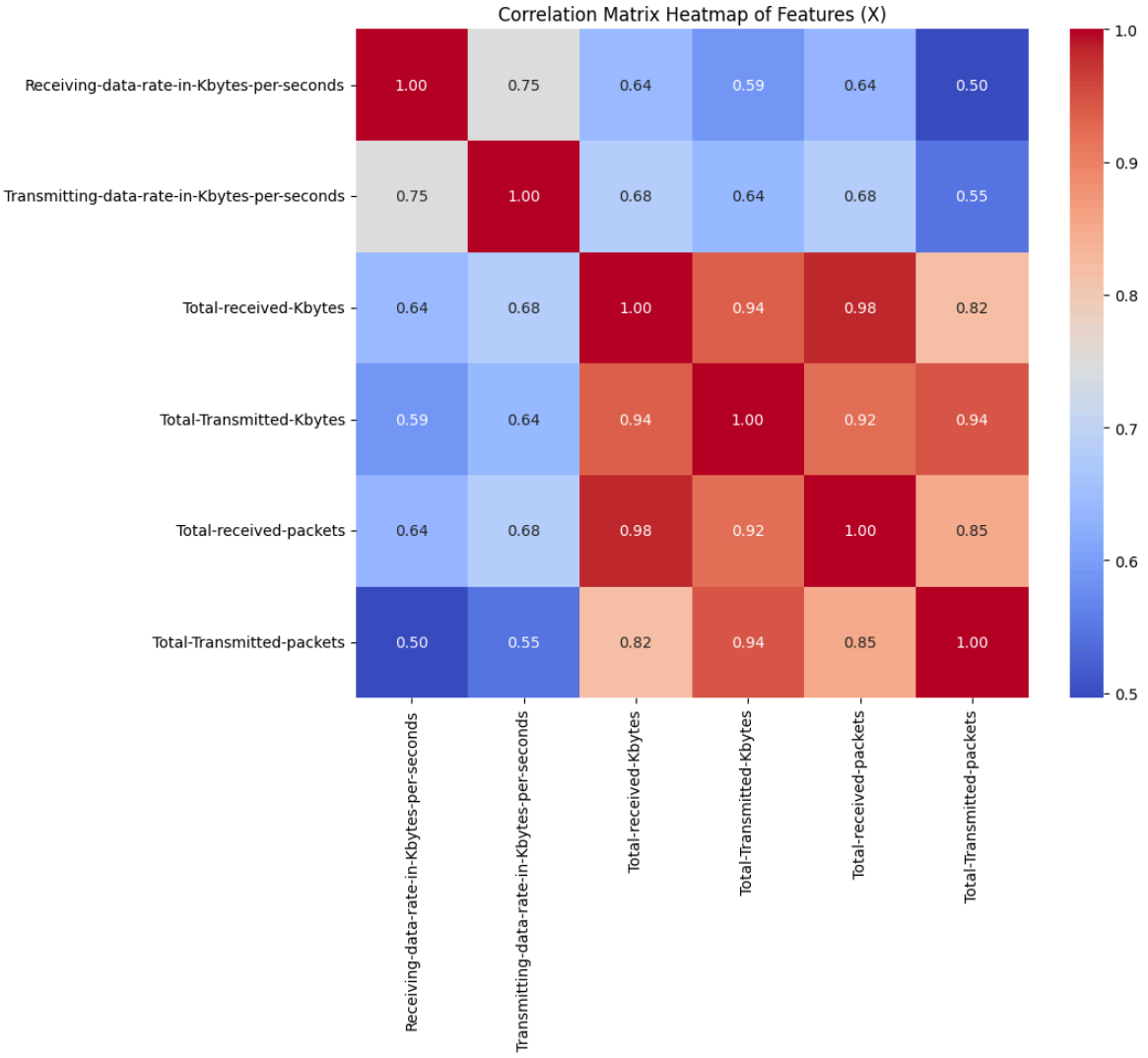
In [4]:
```
#Correlation matrix
corrMatrix = dataSet.corr(method='pearson',numeric_only=True)
print(corrMatrix)
```

```
                                                    Receiving-data-rate-in-Kbytes-per-
seconds  \
Receiving-data-rate-in-Kbytes-per-seconds
1.000000
 Transmitting-data-rate-in-Kbytes-per-seconds
0.748078
 Total-received-Kbytes
0.643117
 Total-Transmitted-Kbytes
0.591491
 Total-received-packets
0.636381
 Total-Transmitted-packets
0.496000

                                                    Transmitting-data-rate-in-Kbytes-
per-seconds  \
Receiving-data-rate-in-Kbytes-per-seconds
0.748078
 Transmitting-data-rate-in-Kbytes-per-seconds
1.000000
 Total-received-Kbytes
0.681459
 Total-Transmitted-Kbytes
0.639431
 Total-received-packets
0.681218
 Total-Transmitted-packets
0.545820

                                                    Total-received-Kbytes  \
Receiving-data-rate-in-Kbytes-per-seconds                        0.643117
 Transmitting-data-rate-in-Kbytes-per-seconds                    0.681459
 Total-received-Kbytes                                           1.000000
 Total-Transmitted-Kbytes                                        0.935787
 Total-received-packets                                          0.981267
 Total-Transmitted-packets                                       0.820463

                                                    Total-Transmitted-Kbytes  \
Receiving-data-rate-in-Kbytes-per-seconds                          0.591491
 Transmitting-data-rate-in-Kbytes-per-seconds                      0.639431
 Total-received-Kbytes                                             0.935787
 Total-Transmitted-Kbytes                                          1.000000
 Total-received-packets                                            0.921450
 Total-Transmitted-packets                                         0.943146

                                                    Total-received-packets  \
Receiving-data-rate-in-Kbytes-per-seconds                        0.636381
 Transmitting-data-rate-in-Kbytes-per-seconds                    0.681218
 Total-received-Kbytes                                           0.981267
 Total-Transmitted-Kbytes                                        0.921450
 Total-received-packets                                          1.000000
 Total-Transmitted-packets                                       0.851075

                                                    Total-Transmitted-packets
Receiving-data-rate-in-Kbytes-per-seconds                          0.496000
 Transmitting-data-rate-in-Kbytes-per-seconds                      0.545820
 Total-received-Kbytes                                             0.820463
 Total-Transmitted-Kbytes                                          0.943146
```

```
Total-received-packets                                  0.851075
Total-Transmitted-packets                               1.000000
```

-Plot the heat map of correlation matrix X using the Seaborn library (heatmap).

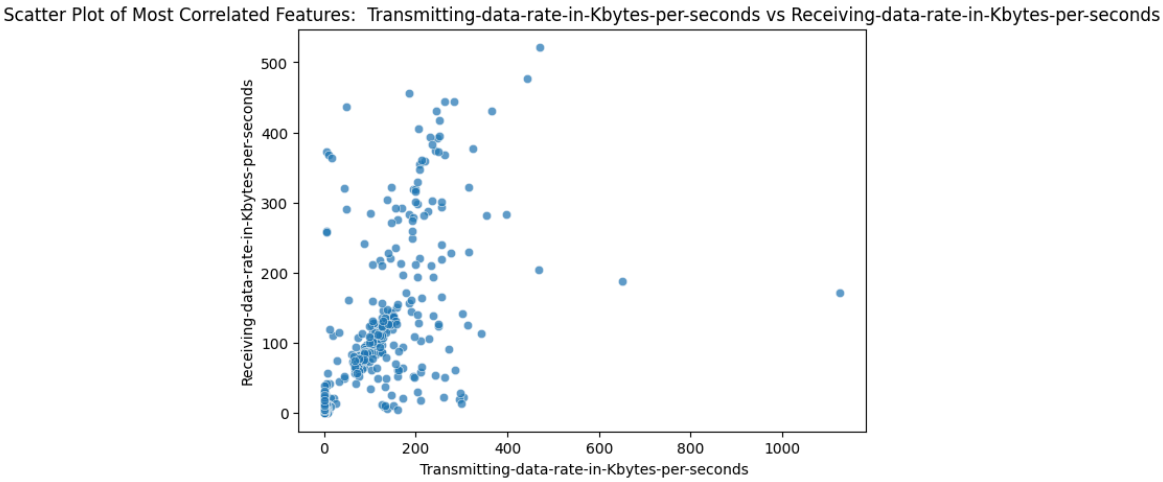
In [5]:
```
#Heatmap

plt.figure(figsize=(10, 8))  # adjust figure size if needed
sn.heatmap(corrMatrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix Heatmap of Features (X)")
plt.show()
```



–Plot the 2 variables with the most correlation in a scatter plot.

In [6]:
```
#Scatter plot
maxCorr = corrMatrix[corrMatrix<1].idxmax()
plt.figure(figsize=(7, 5))
sn.scatterplot(x=dataSet[maxCorr[0]],
               y=dataSet[maxCorr[1]],
               alpha=0.7)
plt.title(f"Scatter Plot of Most Correlated Features: {maxCorr[0]} vs {maxCorr[1
plt.xlabel(maxCorr[0])
plt.ylabel(maxCorr[1])
plt.show()
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_13300\3351030248.py:4: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  sn.scatterplot(x=dataSet[maxCorr[0]],
C:\Users\User\AppData\Local\Temp\ipykernel_13300\3351030248.py:5: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  y=dataSet[maxCorr[1]],
C:\Users\User\AppData\Local\Temp\ipykernel_13300\3351030248.py:7: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  plt.title(f"Scatter Plot of Most Correlated Features: {maxCorr[0]} vs {maxCorr
[1]}")
C:\Users\User\AppData\Local\Temp\ipykernel_13300\3351030248.py:8: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  plt.xlabel(maxCorr[0])
C:\Users\User\AppData\Local\Temp\ipykernel_13300\3351030248.py:9: FutureWarning:
Series.__getitem__ treating keys as positions is deprecated. In a future version,
integer keys will always be treated as labels (consistent with DataFrame behavio
r). To access a value by position, use `ser.iloc[pos]`
  plt.ylabel(maxCorr[1])
```



-Create an array named “y_2c” for two class classification. Convert the samples with the high label as 1 and the rest as 0.

In [7]:
```
# Create one-versus-all with High as 1, all other are labeled 0
print(y)
mapping = {"High": 1}
#y_2c = np.where(dataSet[" Class"] == "High", 1, 0)
y_2c = dataSet[" Class"].map(mapping).fillna(0).astype(int)
print("595th till 610th sample to show correct implementation:")
print(y_2c[595:610])
```

```
0         High
1         High
2         High
3         High
4         High
         ...
1345    Buffer
1346    Buffer
1347    Buffer
1348    Buffer
1349    Buffer
Name:  Class, Length: 1350, dtype: object
595th till 610th sample to show correct implementation:
595    1
596    1
597    1
598    1
599    1
600    0
601    0
602    0
603    0
604    0
605    0
606    0
607    0
608    0
609    0
Name:  Class, dtype: int32
```

– Calculate the ratio of high-class samples to all samples.

In [8]:
```
#Ratio of high-class
ratio = np.count_nonzero(y_2c)/y_2c.size
print(ratio)
```

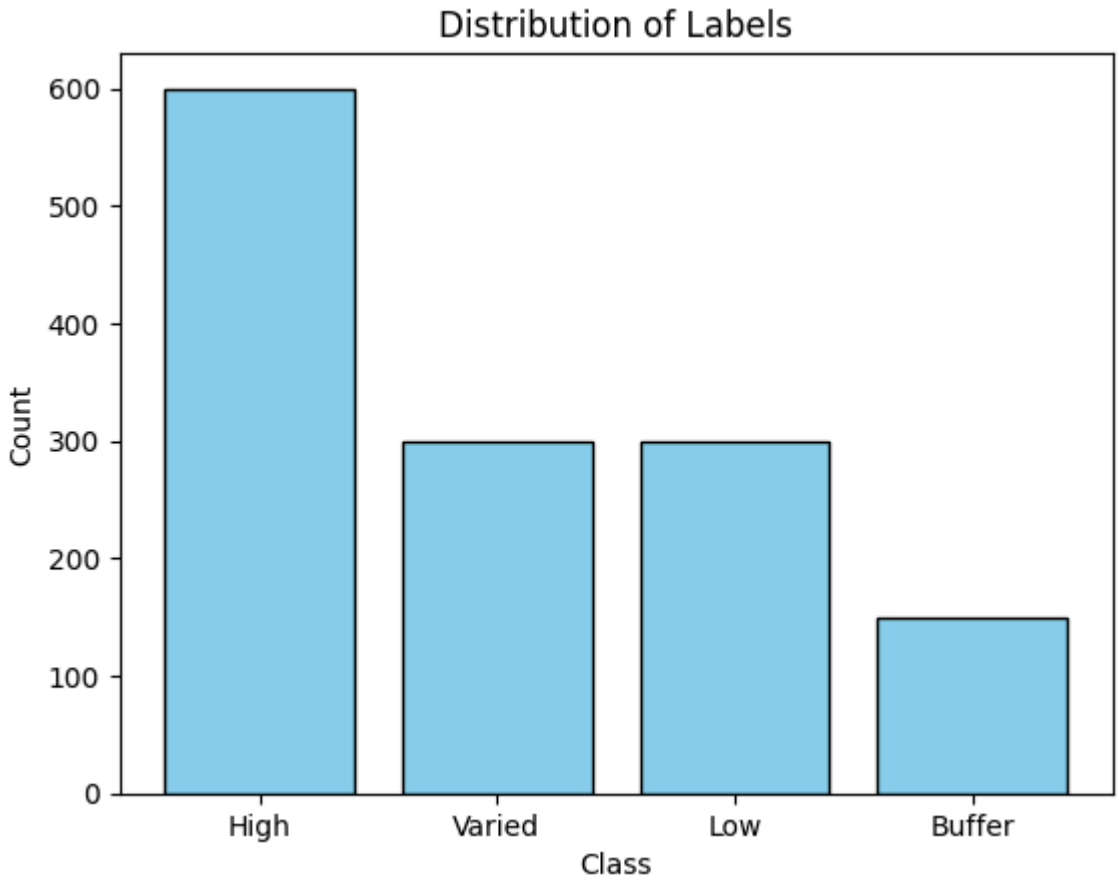
```
0.4444444444444444
```

-Create an array named “y_mc” that can be used for multiclass classification. Use numerical labels according to the class, as follows: 3-High, 2-Varied, 1-Low, 0-Buffer.

In [9]:
```
# Create the Multi class y vector
mapping = {"High": 3, "Varied": 2, "Low": 1, "Buffer": 0}
y_mc = dataSet[" Class"].map(mapping)
```

– Plot the distribution of labels in the data set using a bar plot.

In [10]:
```
#Plot the distribution of labels.
c_high,c_var,c_low,c_buff = y.value_counts()
classes = ["High", "Varied", "Low", "Buffer"]
plt.bar(classes, [c_high,c_var,c_low,c_buff], color='skyblue', edgecolor='black'
plt.title("Distribution of Labels")
plt.xlabel("Class")
plt.ylabel("Count")
plt.xticks(rotation=0)   # keep labels horizontal
plt.show()
```



-Convert data frames X, y_2c, and y_mc to NumPy arrays (if they were Pandas objects). Convert the label vector elements to integers using the following notation, y.astype('int'), where y is the label vector.

In [11]:
```
# Convert to array
y_2c_data = y_2c.to_numpy()
y_mc_data = y_mc.to_numpy()
X_data = X.to_numpy()
```

-Scale the feature matrix X with the StandardScaler() function from Sklearn.

In [12]:
```
#Scaling
X_scaled = StandardScaler().fit_transform(X)
print(X_scaled)
```

```
[[ 0.54458282  0.66784597 -0.5022162  -0.459919   -0.54371335 -0.61464375]
 [ 0.74888209  0.72570995 -0.48589312 -0.43621781 -0.52627428 -0.58708251]
 [ 0.4316128   0.53807088 -0.46612267 -0.40815778 -0.50621936 -0.55658142]
 ...
 [-0.17265444 -0.44974057 -0.02108976 -0.47504989 -0.34676093 -0.48639215]
 [-0.14663398 -0.45231803 -0.01931042 -0.47501203 -0.34610696 -0.48602466]
 [-0.24260053 -0.4457455  -0.01456551 -0.47494065 -0.34425406 -0.48510595]]
```

Questions:

1) Which variables have the highest correlation? Does this make sense to you? Why?

2) What is the ratio of high-class labels in binary classification?

3) What is the share of each label in multiclass classification?

4) Why might scaling improve the prediction capability of machine learning?

Task 2: Binary classification with Logistic Regression

In the second task, you will develop a binary classifier (one versus all) to detect whether the wireless network is operating in high consumption mode or not. ML models will be tested based on 25% of the input data. You are free to develop your own functions using Sklearn functions for tasks like training, predicting, or performance evaluation.

– Split the data into training and test sets using the "train_test_split" function from Sklearn. Use 4720 as the random state parameter to control the split.

In [13]:
```
#Train test split
Test_ratio = 0.25
Shuffle_state = 4720
X_train, X_test, y_train, y_test = train_test_split(X_scaled,y_2c_data,test_size
```

–Develop and train a logistic regression model with default parameters.
–Use the trained model to predict test labels.

In [14]:
```
#Logistic regression
clf_lr = LogisticRegression().fit(X_train, y_train)
y_prediction = clf_lr.predict(X_test)
```

–Calculate and print the following performance metrics: Accuracy, recall, precision, and F1 score.

In [15]:
```
#Performance metrics
#f1_score, accuracy_score, roc_curve, precision_recall_curve, PrecisionRecallDis
#roc_auc_score, RocCurveDisplay, confusion_matrix, precision_score, recall_score
Accuracy_LR = accuracy_score(y_test,y_prediction)
F1_LR = f1_score(y_test,y_prediction,average="macro")
Precision_LR = precision_score(y_test,y_prediction, average="macro")
Recall_LR = recall_score(y_test,y_prediction,average="macro")
print("Accuracy: " + str(Accuracy_LR))
print("F1 score: " + str(F1_LR))
print("Recall score: " + str(Recall_LR))
print("Precision score: " + str(Precision_LR))
```

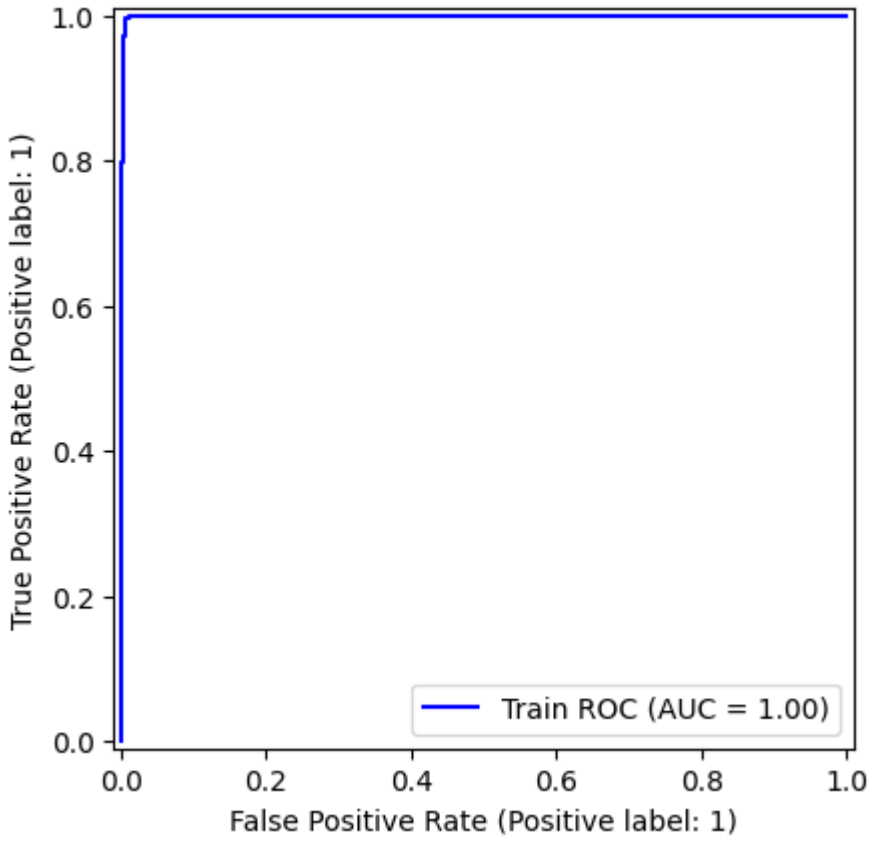
```
Accuracy: 0.9378698224852071
F1 score: 0.936941979158338
Recall score: 0.9339622641509434
Precision score: 0.9475
```

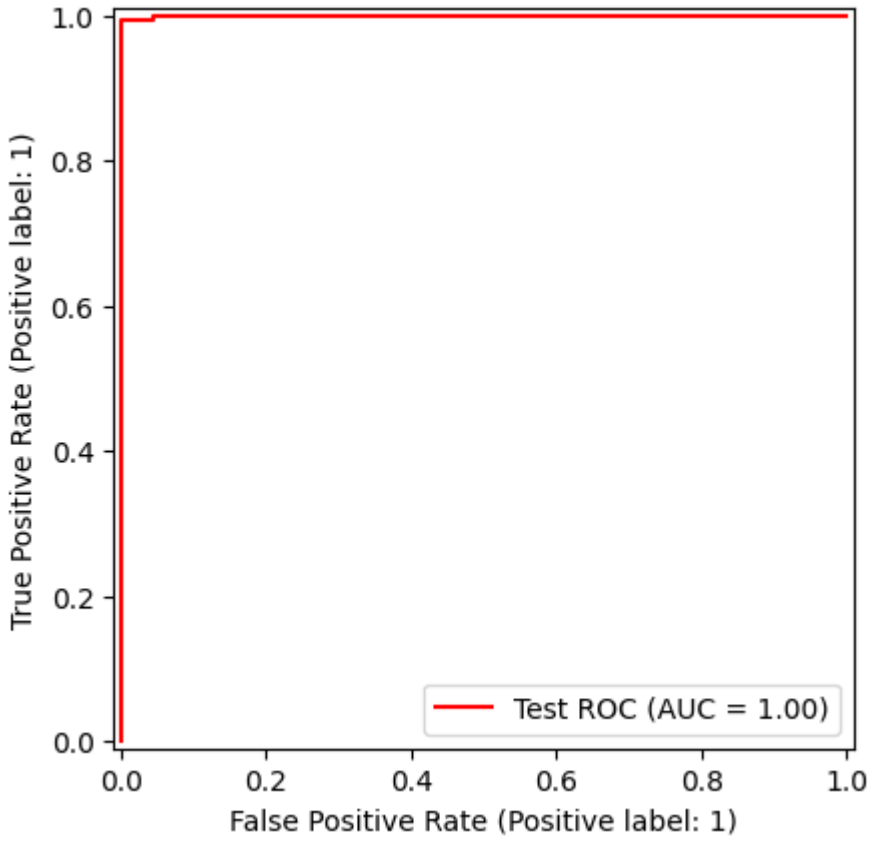
–Plot the receiver operating characteristics (ROC) curve of training and test data in the same figure. By default, the logistic regression model uses a cut-off value of 0.5 to assign a positive or negative prediction to a data point. Different values can be used to bias the prediction towards negative or positive outcomes. The ROC curve plots the combinations of specificity and recall (sensitivity) that result from a range of cut-off values.

In [16]:
```
#ROC curve
```

```
RocCurveDisplay.from_estimator(clf_lr, X_train, y_train, name="Train ROC", color
RocCurveDisplay.from_estimator(clf_lr, X_test, y_test, name="Test ROC", color="r
```

Out[16]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c001f97a70>

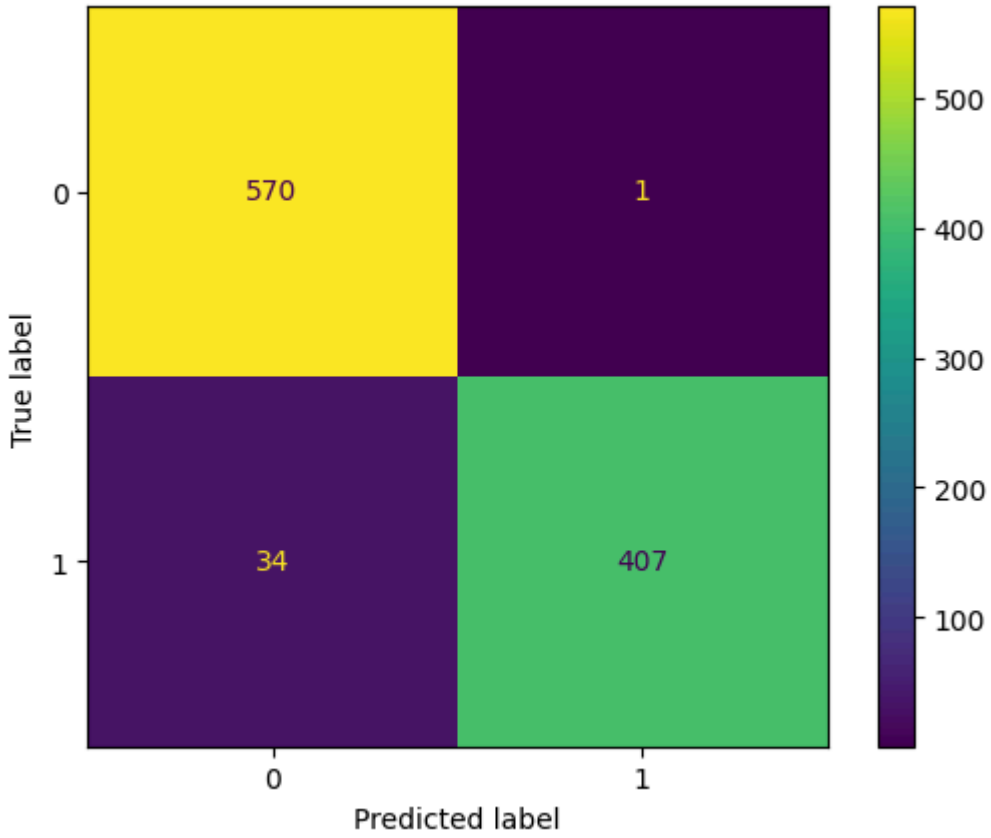


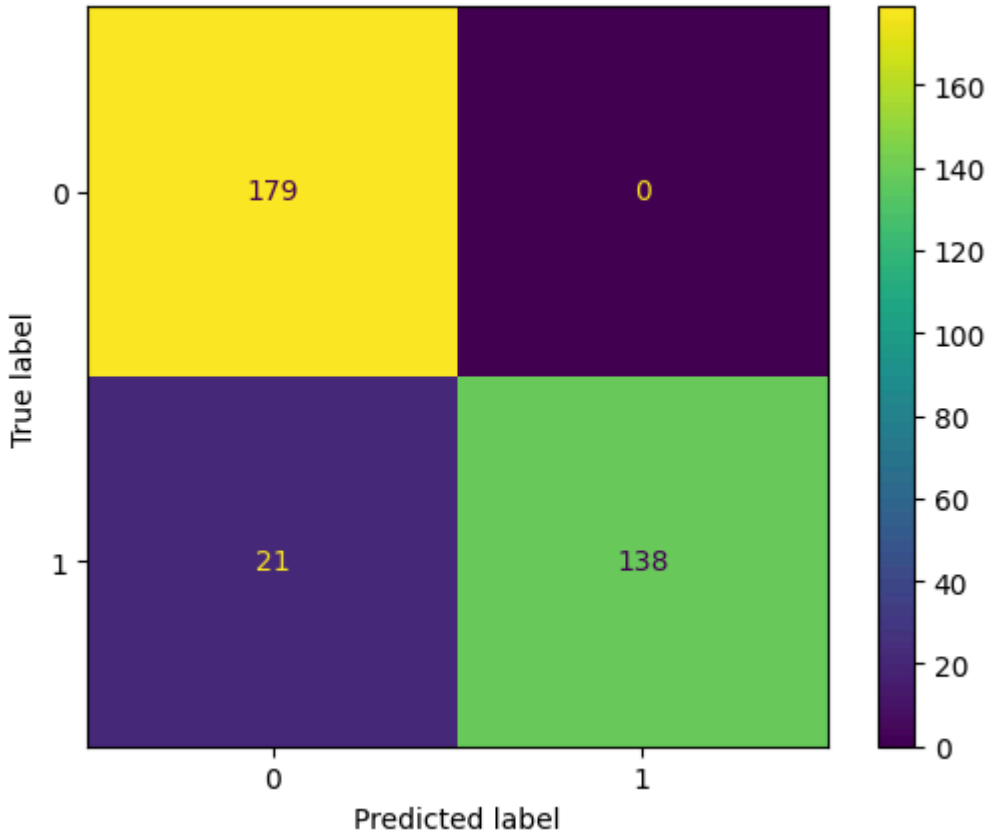


- Generate and plot the confusion matrix.

In [17]:
```
#Confusion Matrix
ConfusionMatrixDisplay.from_estimator(clf_lr, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_lr, X_test, y_test)
```

Out[17]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0041e1580
>
```





The class weight parameter adjusts the penalization factor of misclassified examples in model training. Depending on the application selection of weights varies. For example, in medical testing, the cost of false negatives (absence of a disease when it is present) outweighs the cost of false positives (incorrect indication of a disease when it is not present). In this binary classification, we need to modify the weights to reduce the number of missed high cases (false negatives).
-Develop and train a logistic regression model using the “class_weight” parameter to reduce the number of false negatives in the test data (y_(2c_test)=1, but y_pred = 0). (Tip: In CodeLab2, we had constructed a SVM grid search. Similar to that, you don't have to try all the combinations yourself, you can build a nested loop and add a judgment statement at the end until you have a suitable result)
-Print out the selected weights.

In [18]:
```
#Weighted logistic regression
w_0_list = np.linspace(0.0005,10,100)
w_1_list = np.linspace(0.0005,10,100)
best_recall = 0
best_w0, best_w1 = None, None
best_model = None
best_predict = None
eps = 1e-4
for w_0 in w_0_list:
    for w_1 in w_1_list:
        # Create and train SVR
        clf_lr_w = LogisticRegression(class_weight={0: w_0, 1: w_0, 2: w_0, 3: w
        clf_lr_w.fit(X_train, y_train)

        y_pred = clf_lr_w.predict(X_test)
        recall = recall_score(y_test, y_pred)
        if recall > best_recall+eps:
            best_recall = recall
            best_w0, best_w1 = w_0, w_1
            best_model = clf_lr_w
            best_predict = y_pred


print(best_w0)
print(best_w1)
```

```
9.090954545454546
0.0005
```

−Calculate and print the following performance metrics: Accuracy, recall, precision, and F1 score.

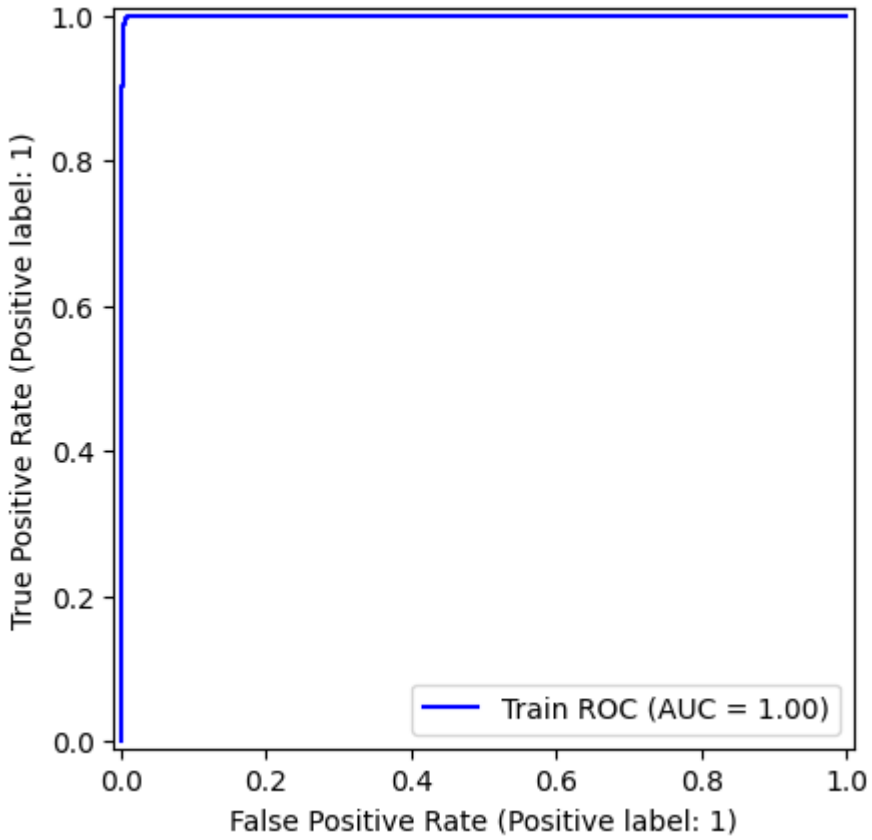
In [19]:
```
#Performance metrics
y_prediction = best_predict
Accuracy_LRw  = accuracy_score(y_test, y_prediction)
F1_LRw        = f1_score(y_test, y_prediction)
Precision_LRw = precision_score(y_test, y_prediction)
Recall_LRw = best_recall
print("Accuracy: " + str(Accuracy_LRw))
print("F1 score: " + str(F1_LRw))
print("Recall score: " + str(Recall_LRw))
print("Precision score: " + str(Precision_LRw))
```

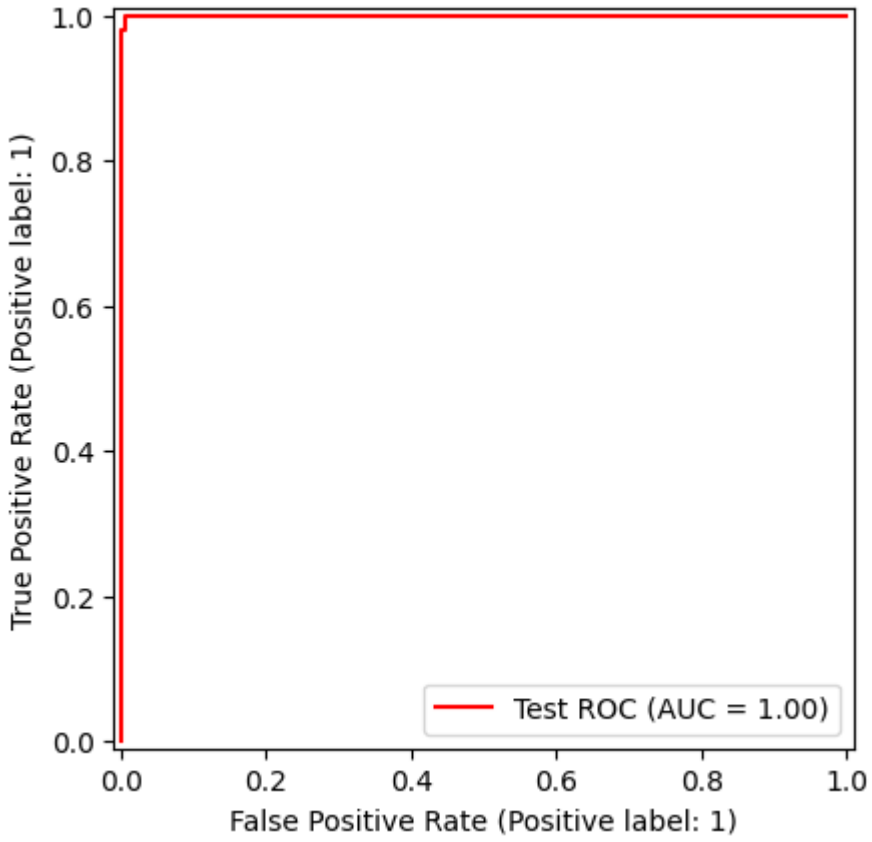
```
Accuracy: 0.9733727810650887
F1 score: 0.970873786407767
Recall score: 0.9433962264150944
Precision score: 1.0
```

−Plot receiver operating characteristics (ROC) curve of training and test data in the same figure.

In [20]:
```
RocCurveDisplay.from_estimator(best_model, X_train, y_train, name="Train ROC", c
RocCurveDisplay.from_estimator(best_model, X_test, y_test, name="Test ROC", colo
```

Out[20]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c0040fb2f0>
```

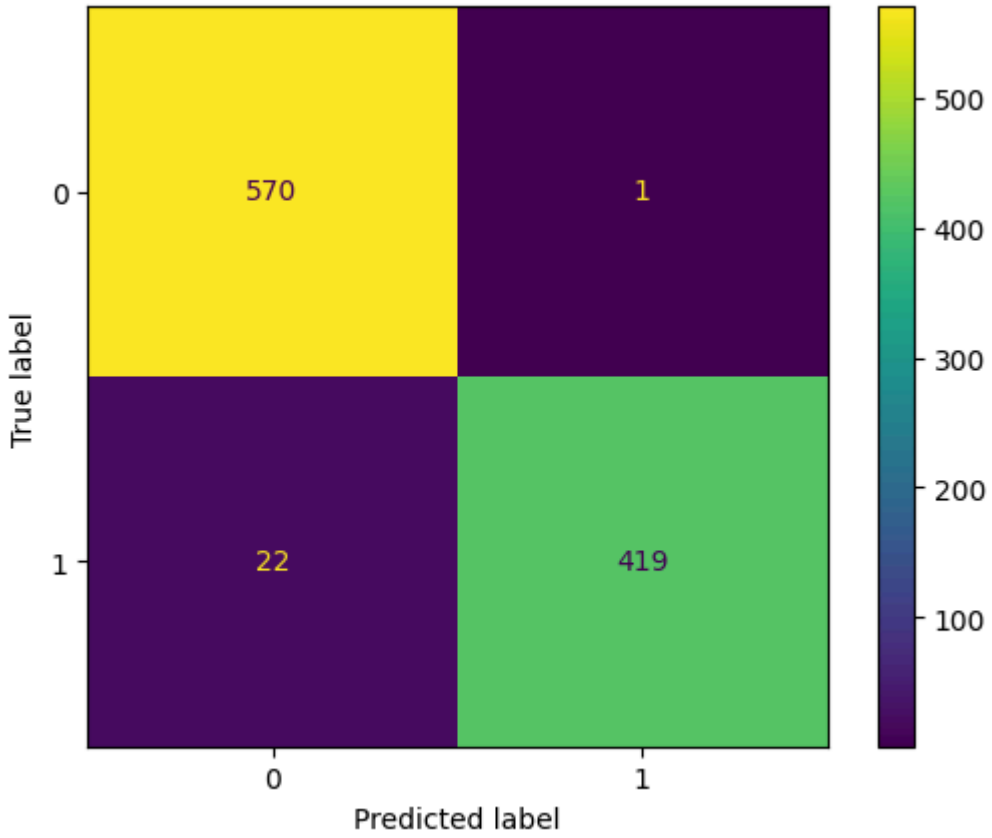


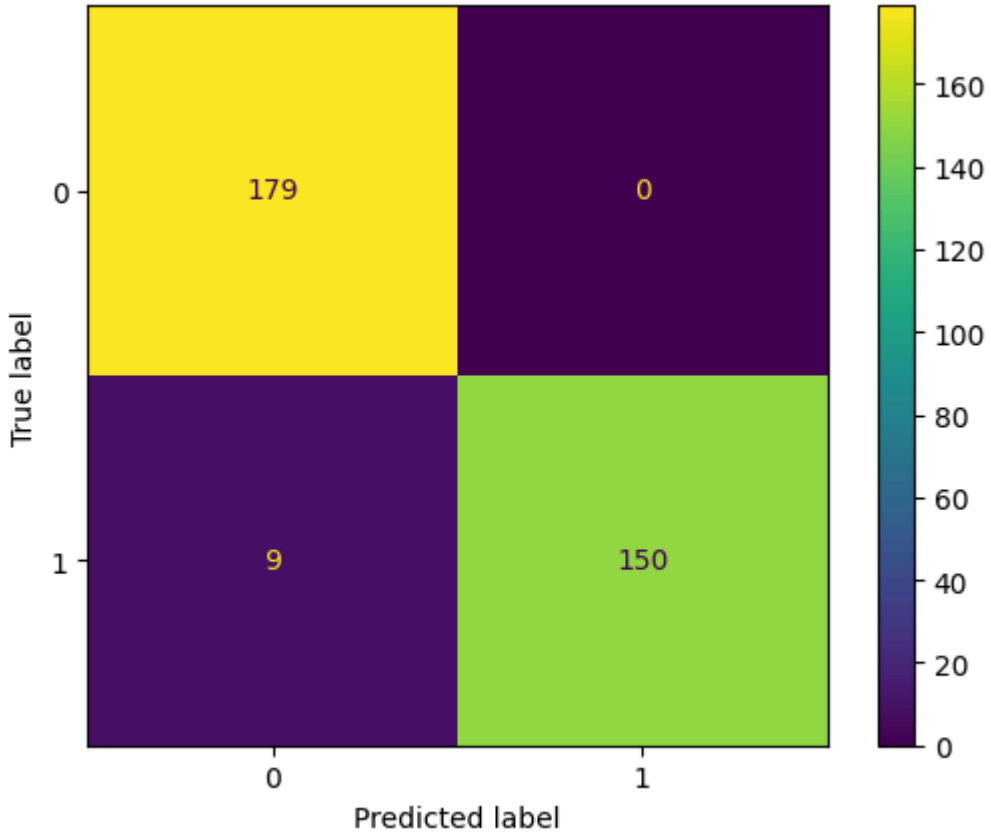


- Generate and plot the confusion matrix.

In [21]:
```
ConfusionMatrixDisplay.from_estimator(best_model, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(best_model, X_test, y_test)
```

Out[21]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0041745c0
>
```





Questions :
1) What differences do you find between train and test ROC curves in the logistic regression model?
2) What does the confusion matrix represent?
3) How do the weights influence the results of our classification in logistic regression?
4) What metrics are good to use to compare models? How can you decide what is a good model?

Task 3: Binary classification with SVM

In this task you will develop multiple support vector classifiers (SVC) for the binary classification task.
-Develop and train a linear SVC with the given parameters [C=1.0, coef0=0.0, tol=1e-3]. (Note: default kernel is RBF and must be changed to linear)

In [22]:
```
clf_svmlin = svm.SVC(
    kernel='linear',
    C=1.0,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmlin.fit(X_train, y_train)
y_prediction_svmlin = clf_svmlin.predict(X_test)
```

−Calculate and print the following performance metrics: Accuracy, recall, precision, and F1 score.

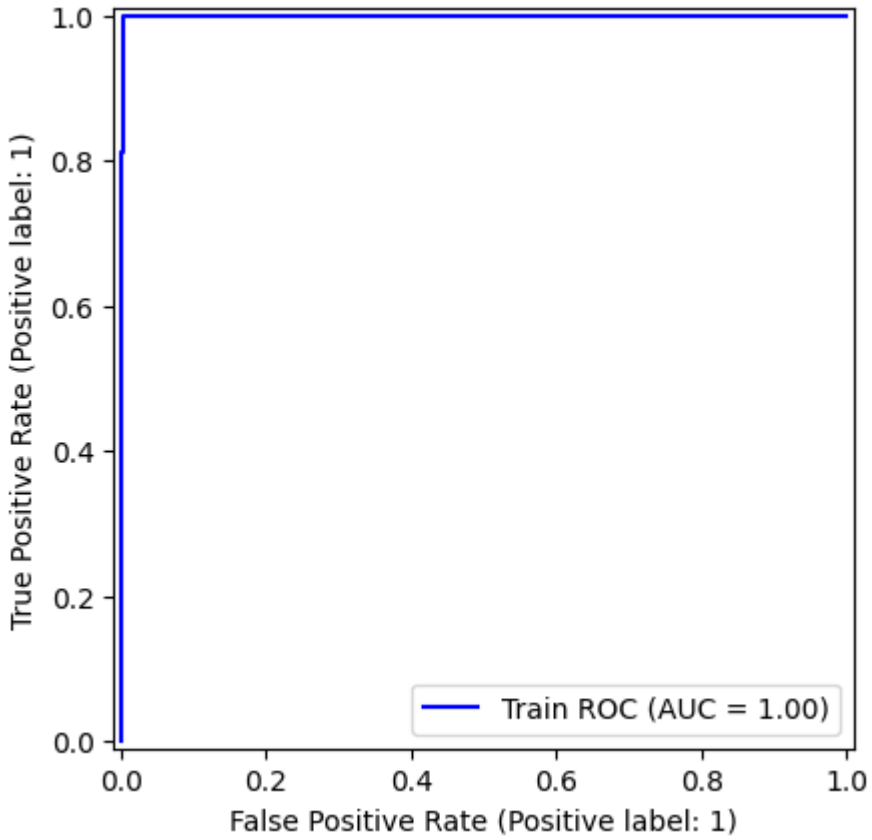
In [23]:
```
#Performance metrics
Accuracy_svmlin = accuracy_score(y_test,y_prediction_svmlin)
F1_svmlin = f1_score(y_test,y_prediction_svmlin)
Precision_svmlin = precision_score(y_test,y_prediction_svmlin)
Recall_svmlin = recall_score(y_test,y_prediction_svmlin)
print("Accuracy: " + str(Accuracy_svmlin))
print("F1 score: " + str(F1_svmlin))
print("Recall score: " + str(Recall_svmlin))
print("Precision score: " + str(Precision_svmlin))
```

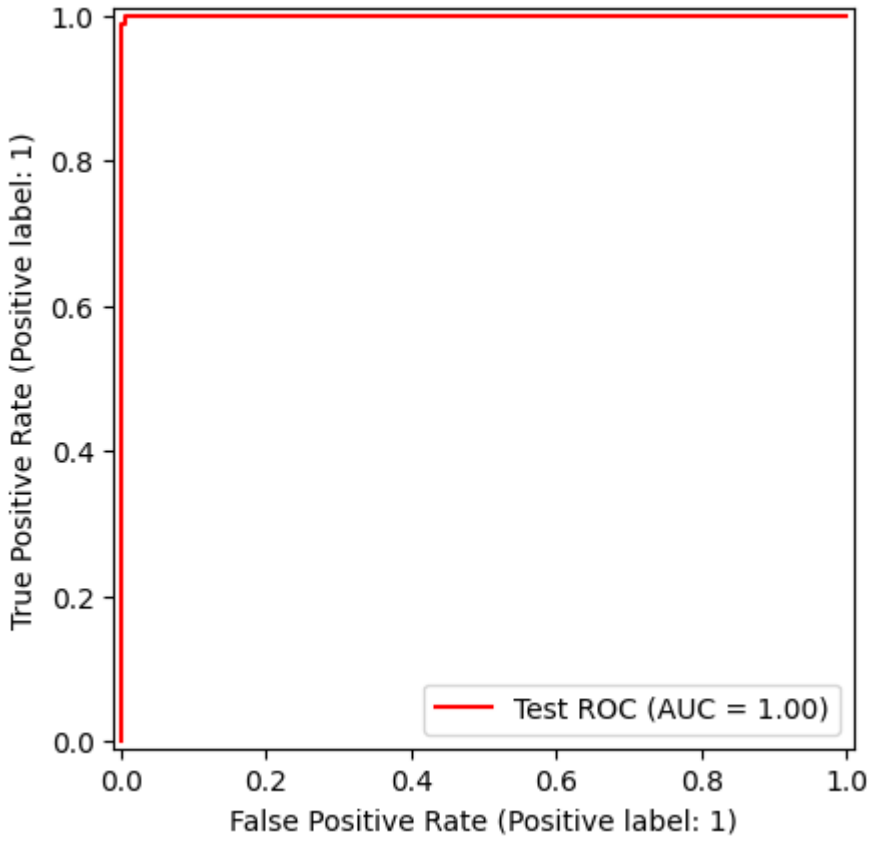
```
Accuracy: 0.9733727810650887
F1 score: 0.970873786407767
Recall score: 0.9433962264150944
Precision score: 1.0
```

−Plot receiver operating characteristics (ROC) curve of training and test data in the same figure.

In [24]:
```
RocCurveDisplay.from_estimator(clf_svmlin, X_train, y_train, name="Train ROC", c
RocCurveDisplay.from_estimator(clf_svmlin, X_test, y_test, name="Test ROC", colo
```

Out[24]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c0042ee210>

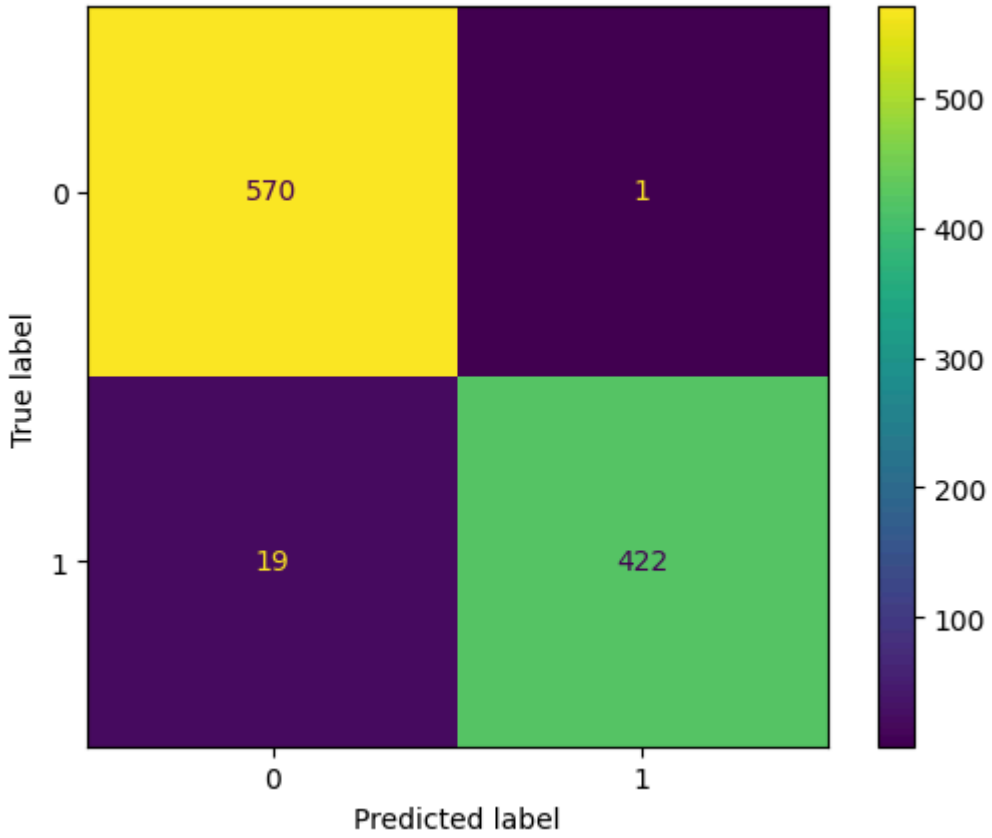


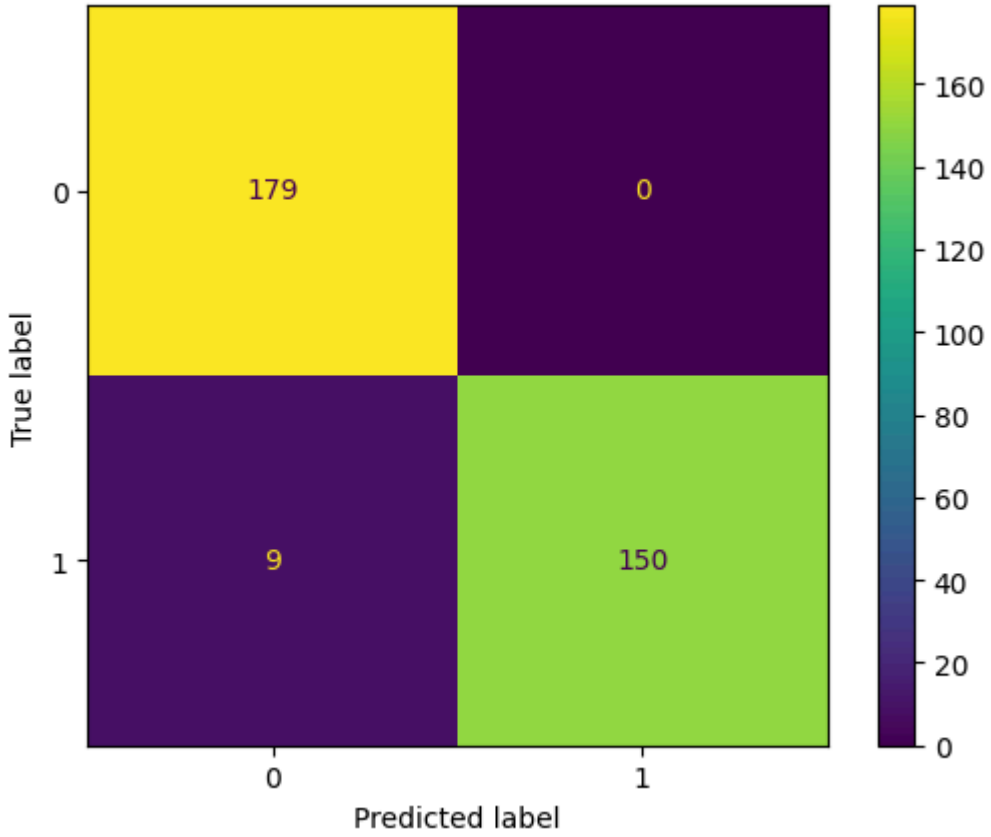


- Generate and plot the confusion matrix.

In [25]:
```
ConfusionMatrixDisplay.from_estimator(clf_svmlin, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmlin, X_test, y_test)
```

Out[25]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c003fdc440
>
```





Next step, you will investigate the impact of hyperparameters. The regularization parameter (C) adjusts the penalty factor for samples in training.
-Develop two linear SVCs, one with parameter C as 0.0001 and one with C as 100, then train both models. (tol=1e-3)

In [26]:
```
#SVC construction for C=0.0001 and C=100
clf_svmlin2 = svm.SVC(
    kernel='linear',
    C=0.0001,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmlin2.fit(X_train, y_train)
y_prediction_svmlin2 = clf_svmlin2.predict(X_test)

clf_svmlin3 = svm.SVC(
    kernel='linear',
    C=100,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmlin3.fit(X_train, y_train)
y_prediction_svmlin3 = clf_svmlin3.predict(X_test)
```

−Calculate and print the following performance metrics for both cases: Accuracy, recall, precision, and F1 score.

In [27]:
```
#Performance metrics C=0.0001

Accuracy_svmlin2 = accuracy_score(y_test,y_prediction_svmlin2)
F1_svmlin2 = f1_score(y_test,y_prediction_svmlin2)
Precision_svmlin2 = precision_score(y_test,y_prediction_svmlin2)
Recall_svmlin2 = recall_score(y_test,y_prediction_svmlin2)
print("Accuracy: " + str(Accuracy_svmlin2))
print("F1 score: " + str(F1_svmlin2))
print("Recall score: " + str(Recall_svmlin2))
print("Precision score: " + str(Precision_svmlin2))
```

```
Accuracy: 0.5473372781065089
F1 score: 0.07272727272727272
Recall score: 0.03773584905660377
Precision score: 1.0
```

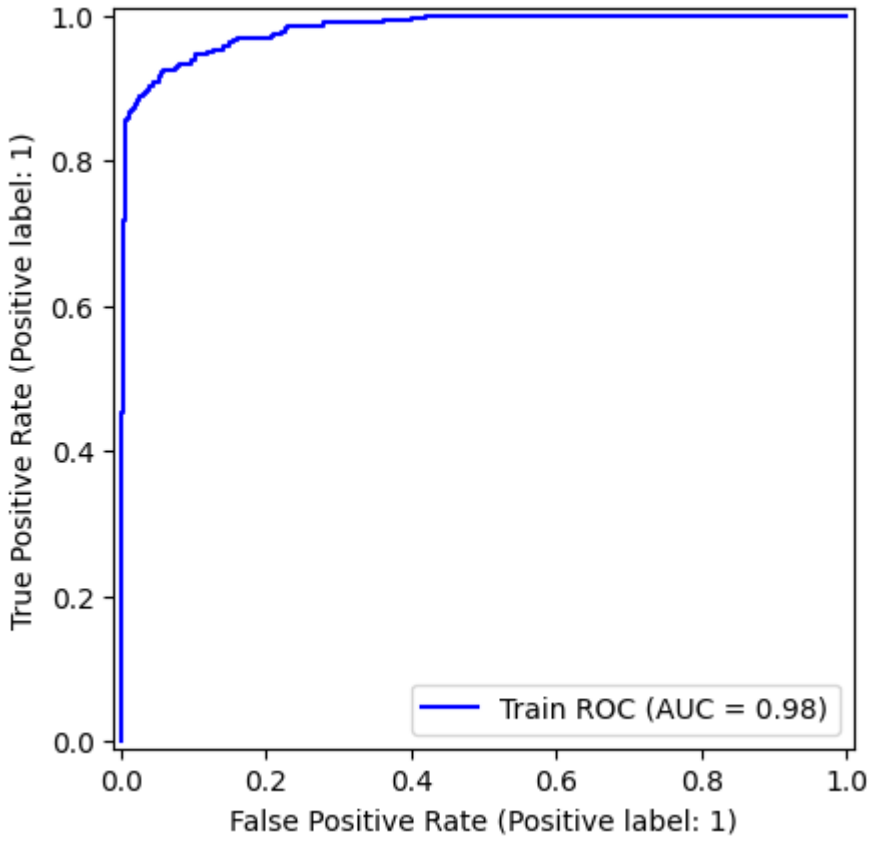
In [28]:
```
#Performance metrics C=100
Accuracy_svmlin3 = accuracy_score(y_test,y_prediction_svmlin3)
F1_svmlin3 = f1_score(y_test,y_prediction_svmlin3)
Precision_svmlin3 = precision_score(y_test,y_prediction_svmlin3)
Recall_svmlin3 = recall_score(y_test,y_prediction_svmlin3)
print("Accuracy: " + str(Accuracy_svmlin3))
print("F1 score: " + str(F1_svmlin3))
print("Recall score: " + str(Recall_svmlin3))
print("Precision score: " + str(Precision_svmlin3))
```

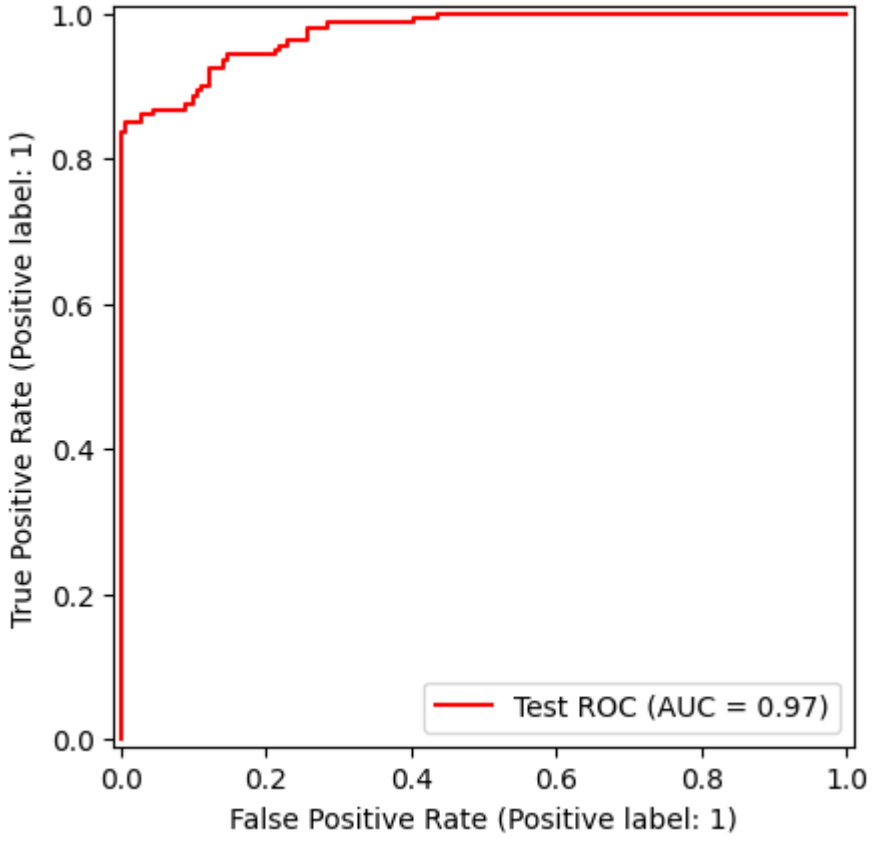
```
Accuracy: 0.985207100591716
F1 score: 0.9841269841269841
Recall score: 0.9748427672955975
Precision score: 0.9935897435897436
```

− Plot the receiver operating characteristic (ROC) curves for both cases separately.

In [29]:
```
RocCurveDisplay.from_estimator(clf_svmlin2, X_train, y_train, name="Train ROC",
RocCurveDisplay.from_estimator(clf_svmlin2, X_test, y_test, name="Test ROC", col
```

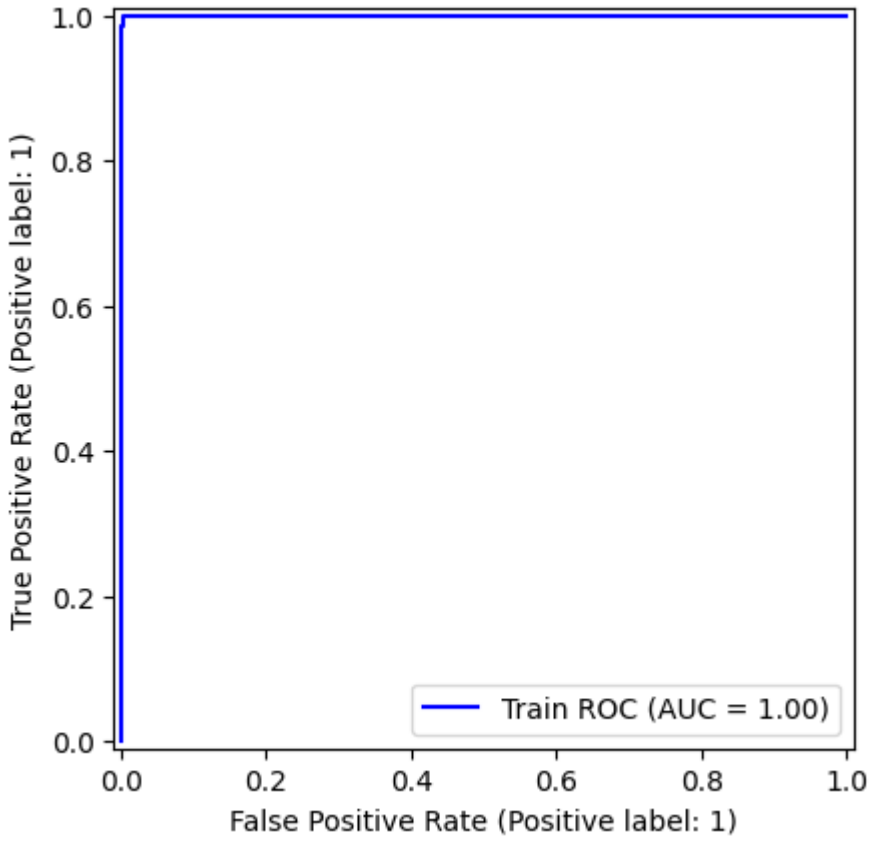
Out[29]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c0043bcda0>
```

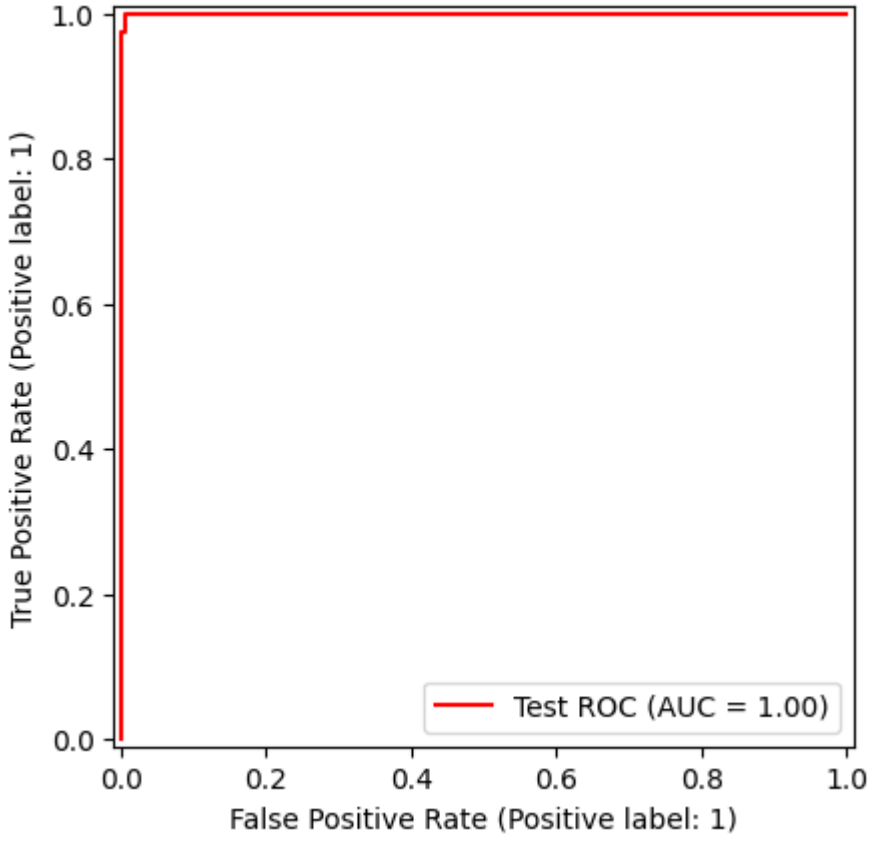




In [30]:
```
RocCurveDisplay.from_estimator(clf_svmlin3, X_train, y_train, name="Train ROC",
RocCurveDisplay.from_estimator(clf_svmlin3, X_test, y_test, name="Test ROC", col
```

Out[30]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c0046976e0>

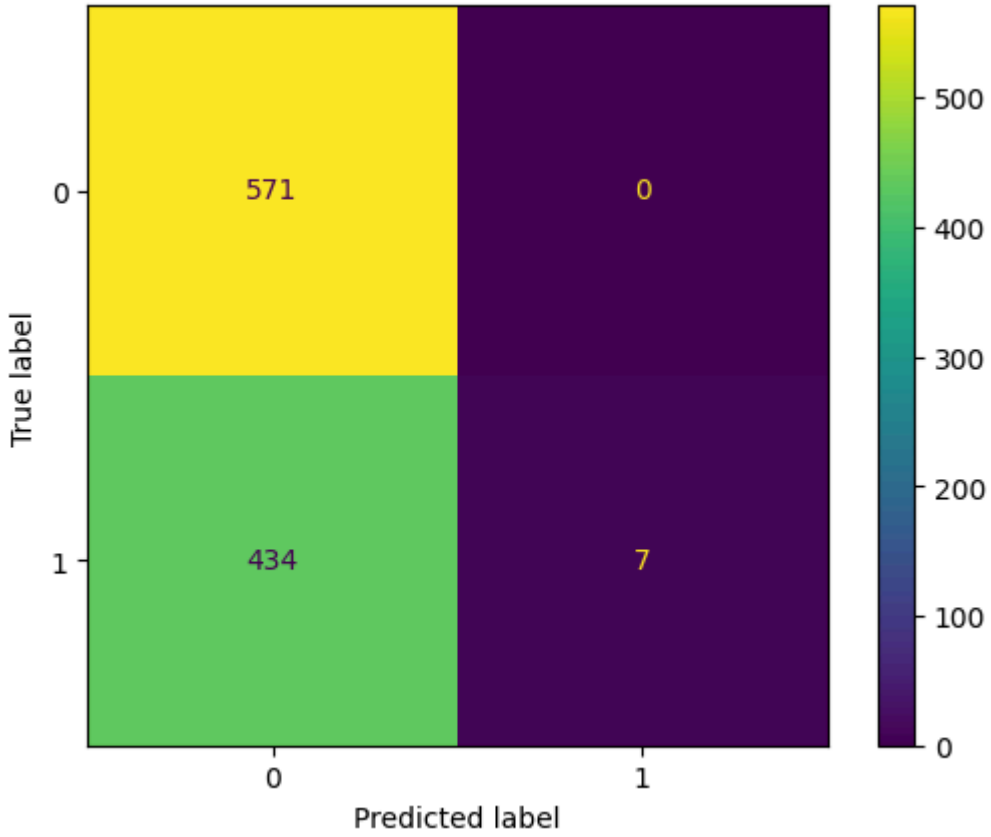


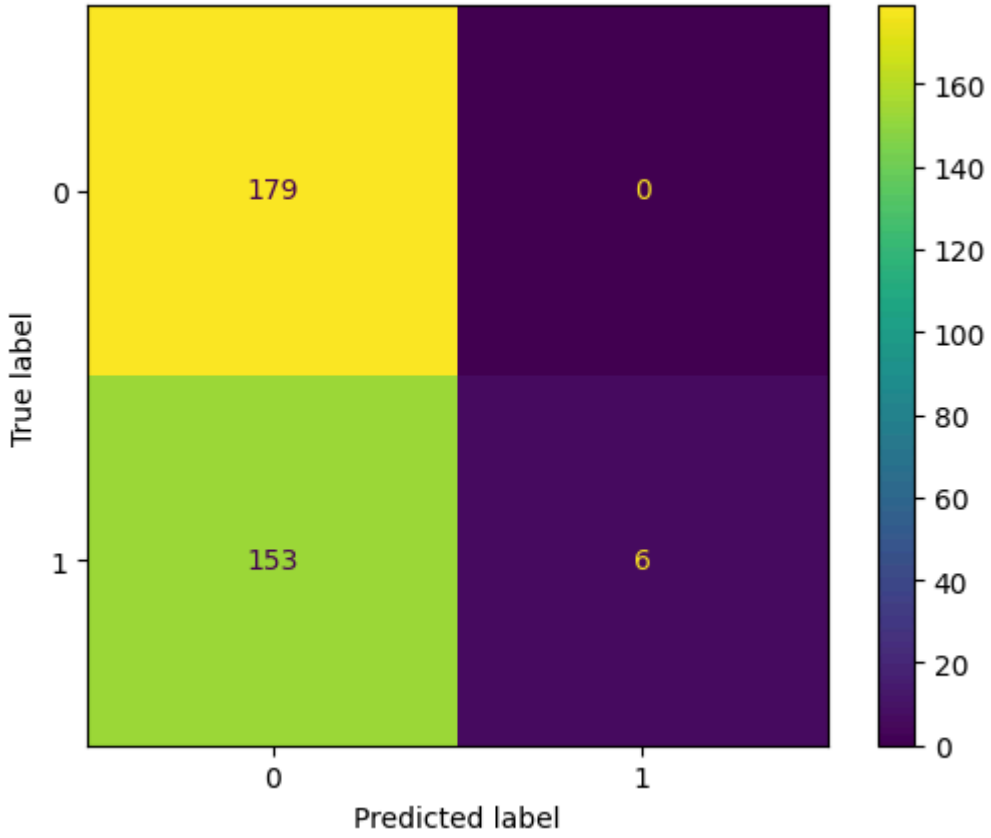


- Generate and plot the confusion matrices for both cases separately.

In [31]:
```
ConfusionMatrixDisplay.from_estimator(clf_svmlin2, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmlin2, X_test, y_test)
```

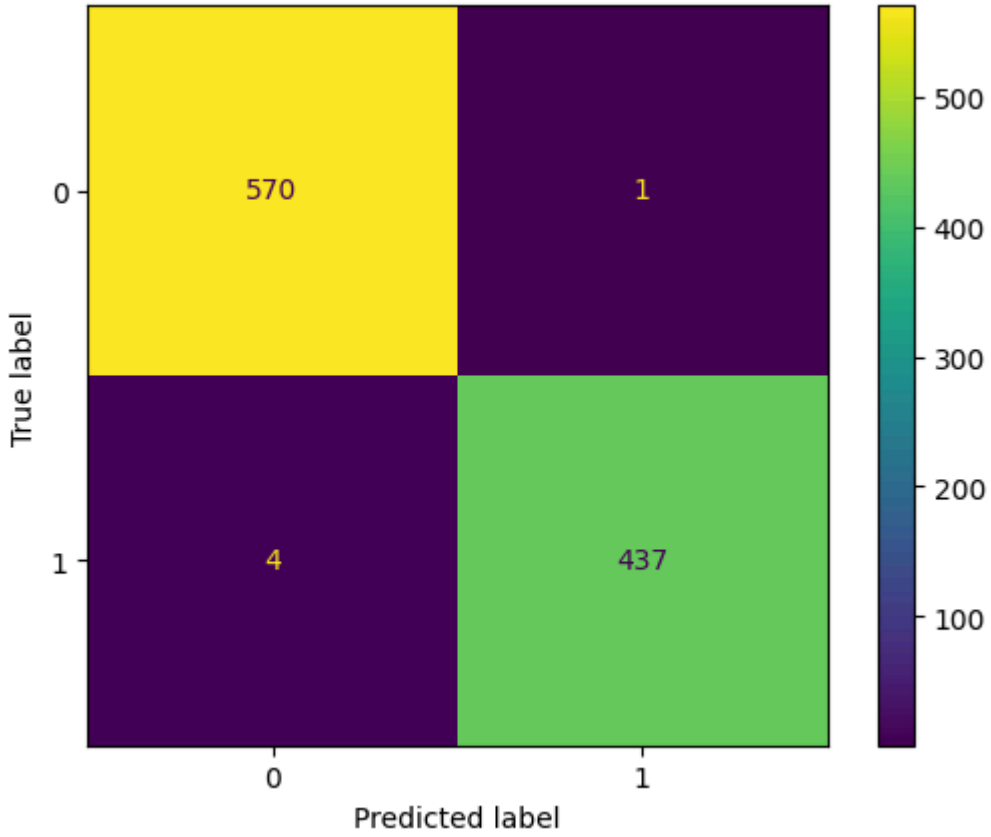
Out[31]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0042af080
>
```

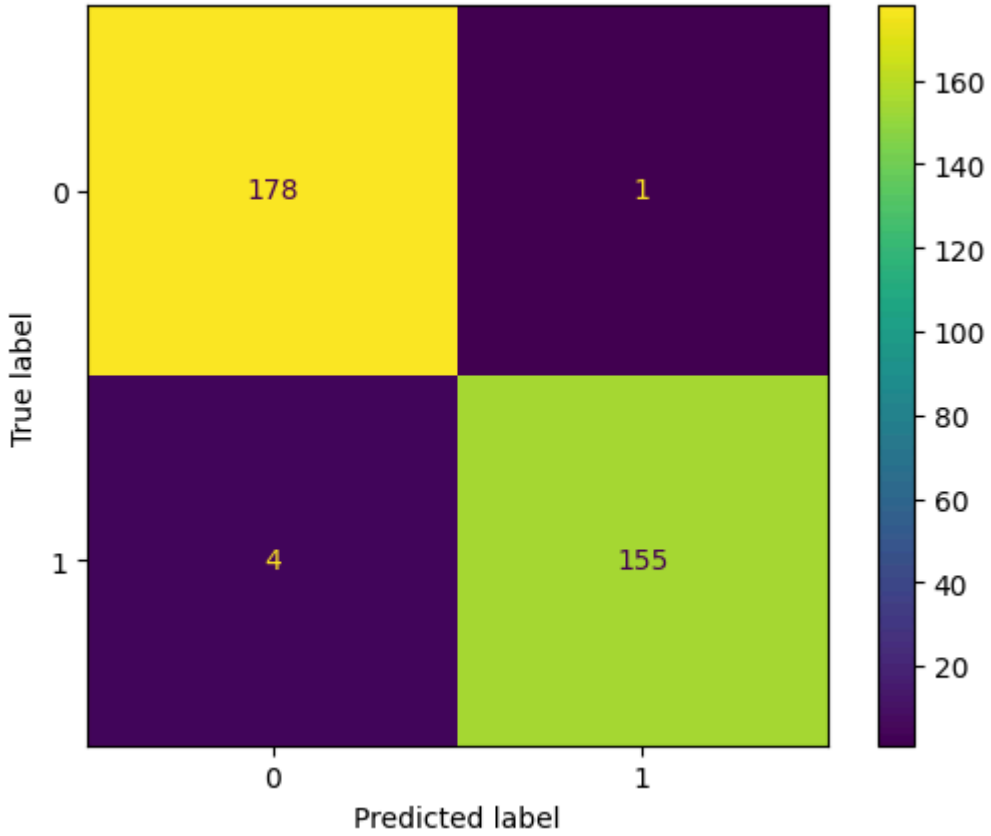




In [32]:
```
ConfusionMatrixDisplay.from_estimator(clf_svmlin3, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmlin3, X_test, y_test)
```

Out[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0045710a0
>
```





The other tunable hyperparameter is tolerance which adjusts the stopping criteria of the optimizer.

– Develop and train a linear SVC with the given parameters [C=1.0, coef0=0.0, tol=10].

In [33]:
```
#Tolerance 10
clf_svmlin4 = svm.SVC(
    kernel='linear',
```

```
    C=1.0,
    coef0=0.0,
    tol=10,
    random_state=Shuffle_state
)

# Train the model
clf_svmlin4.fit(X_train, y_train)
y_prediction_svmlin4 = clf_svmlin4.predict(X_test)
```

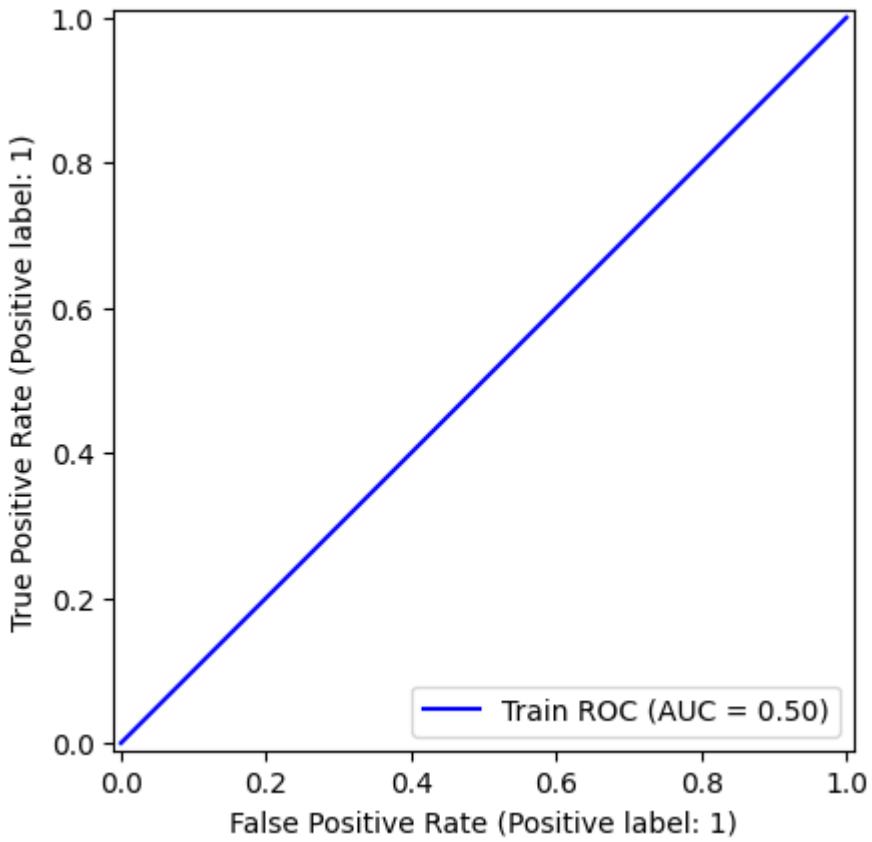
In [34]:
```
#Performance metrics tolerance 10
Accuracy_svmlin4 = accuracy_score(y_test,y_prediction_svmlin4)
F1_svmlin4 = f1_score(y_test,y_prediction_svmlin4)
Precision_svmlin4 = precision_score(y_test,y_prediction_svmlin4)
Recall_svmlin4 = recall_score(y_test,y_prediction_svmlin4)
print("Accuracy: " + str(Accuracy_svmlin4))
print("F1 score: " + str(F1_svmlin4))
print("Recall score: " + str(Recall_svmlin4))
print("Precision score: " + str(Precision_svmlin4))
```

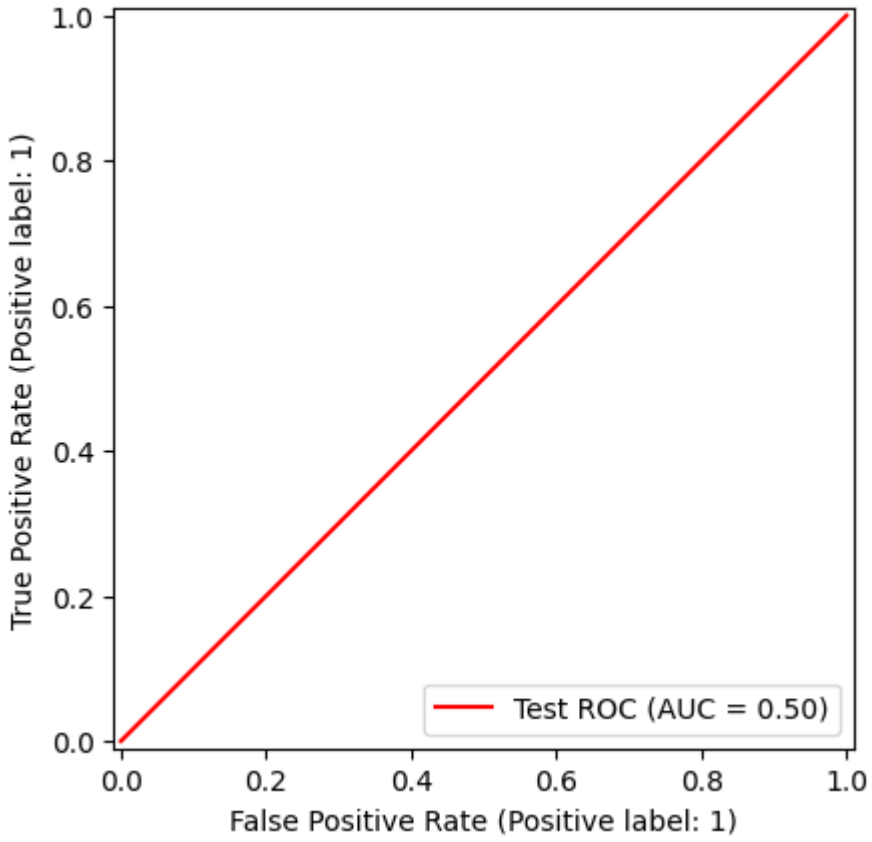
```
Accuracy: 0.47041420118343197
F1 score: 0.6398390342052314
Recall score: 1.0
Precision score: 0.47041420118343197
```

–Plot receiver operating characteristics (ROC) curve of training and test data in the same figure.

In [35]:
```
#ROC curve for tolerance 10
RocCurveDisplay.from_estimator(clf_svmlin4, X_train, y_train, name="Train ROC",
RocCurveDisplay.from_estimator(clf_svmlin4, X_test, y_test, name="Test ROC", col
```

Out[35]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1c001fc6a80>

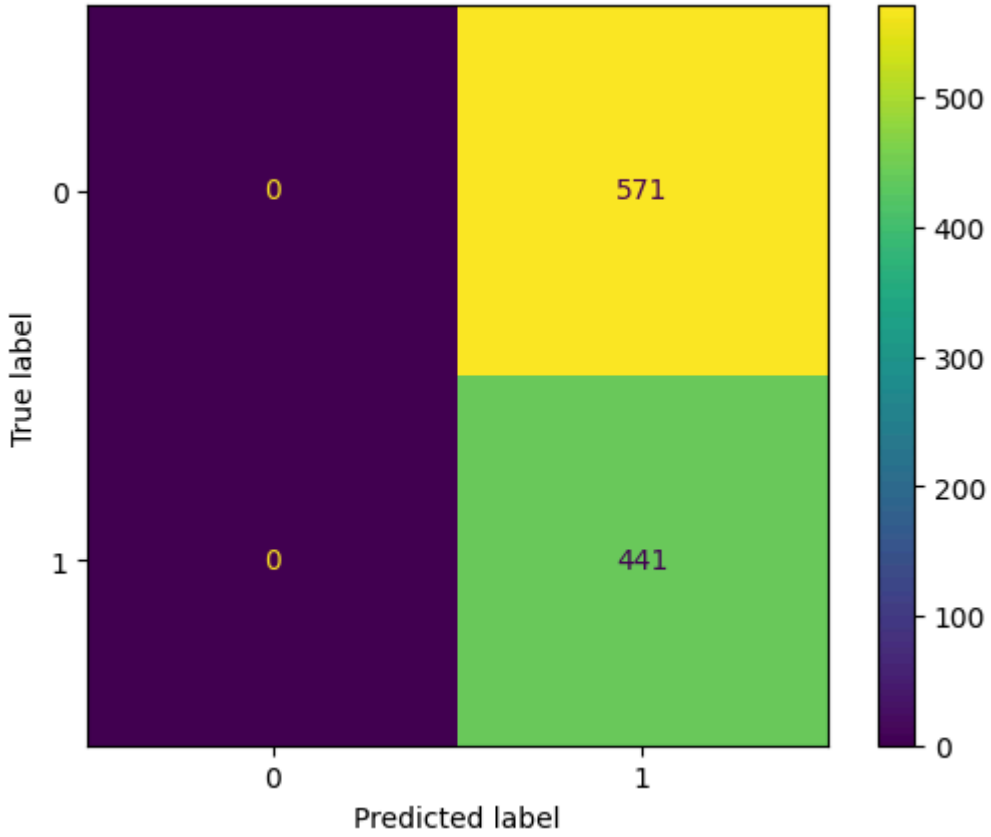


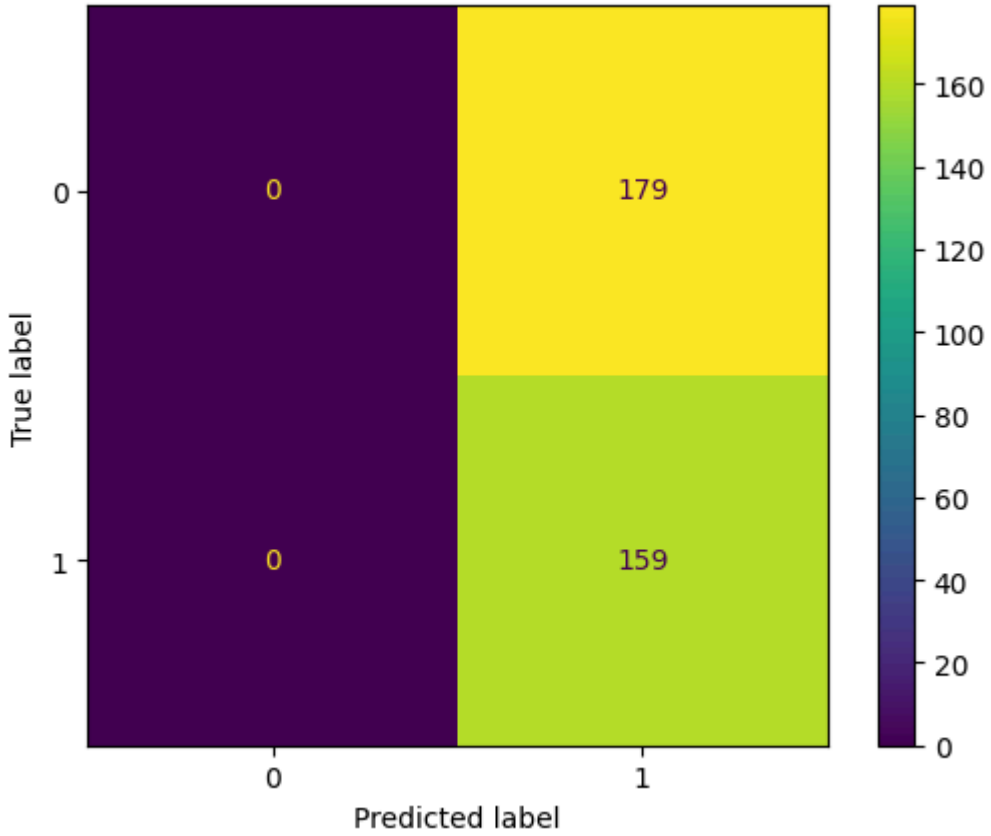


−Generate and plot the confusion matrix.

In [36]:
```
#Confusion matrix tolerance 10
ConfusionMatrixDisplay.from_estimator(clf_svmlin4, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmlin4, X_test, y_test)
```

Out[36]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c004097380
>
```


0
571
0
441
True label
0
1
Predicted label
500
400
300
200
100
0


0
179
0
159
True label
0
1
Predicted label
160
140
120
100
80
60
40
20
0

Kernels are decision functions applied to transform the feature space into higher dimensions. Nonlinear relationships between features can be found in high-dimensional space. In this CodeLab you will use a polynomial SVC (SVM classifier with a polynomial kernel).

−Develop and train two polynomial SVCs [kernel='poly'] one with the degree of 3 and one with the degree of 2. [C=1.0, coef0=0.0, tol=1e-3].

In [37]:
```
#Polynomial SVM with 2nd and 3rd degree.
clf_svmpoly1 = svm.SVC(
    kernel='poly',
    degree=3,
    C=1.0,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmpoly1.fit(X_train, y_train)
y_prediction_svmpoly1 = clf_svmpoly1.predict(X_test)

clf_svmpoly2 = svm.SVC(
    kernel='poly',
    degree=2,
    C=1.0,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmpoly2.fit(X_train, y_train)
y_prediction_svmpoly2 = clf_svmpoly2.predict(X_test)
```

−Calculate and print the following performance metrics for both cases: Accuracy, recall, precision, and F1 score.

In [38]:
```
#Performance metrics polynomial 3rd
Accuracy_svmpoly = accuracy_score(y_test,y_prediction_svmpoly1)
F1_svmpoly = f1_score(y_test,y_prediction_svmpoly1)
Precision_svmpoly = precision_score(y_test,y_prediction_svmpoly1)
Recall_svmpoly = recall_score(y_test,y_prediction_svmpoly1)
print("Accuracy: " + str(Accuracy_svmpoly))
print("F1 score: " + str(F1_svmpoly))
print("Recall score: " + str(Precision_svmpoly))
print("Precision score: " + str(Recall_svmpoly))
```

```
Accuracy: 0.9349112426035503
F1 score: 0.9261744966442953
Recall score: 0.9928057553956835
Precision score: 0.8679245283018868
```

In [39]:
```
#Performance metrics polynomial 2nd
Accuracy_svmpoly2 = accuracy_score(y_test,y_prediction_svmpoly2)
F1_svmpoly2 = f1_score(y_test,y_prediction_svmpoly2)
Precision_svmpoly2 = precision_score(y_test,y_prediction_svmpoly2)
Recall_svmpoly2 = recall_score(y_test,y_prediction_svmpoly2)
```

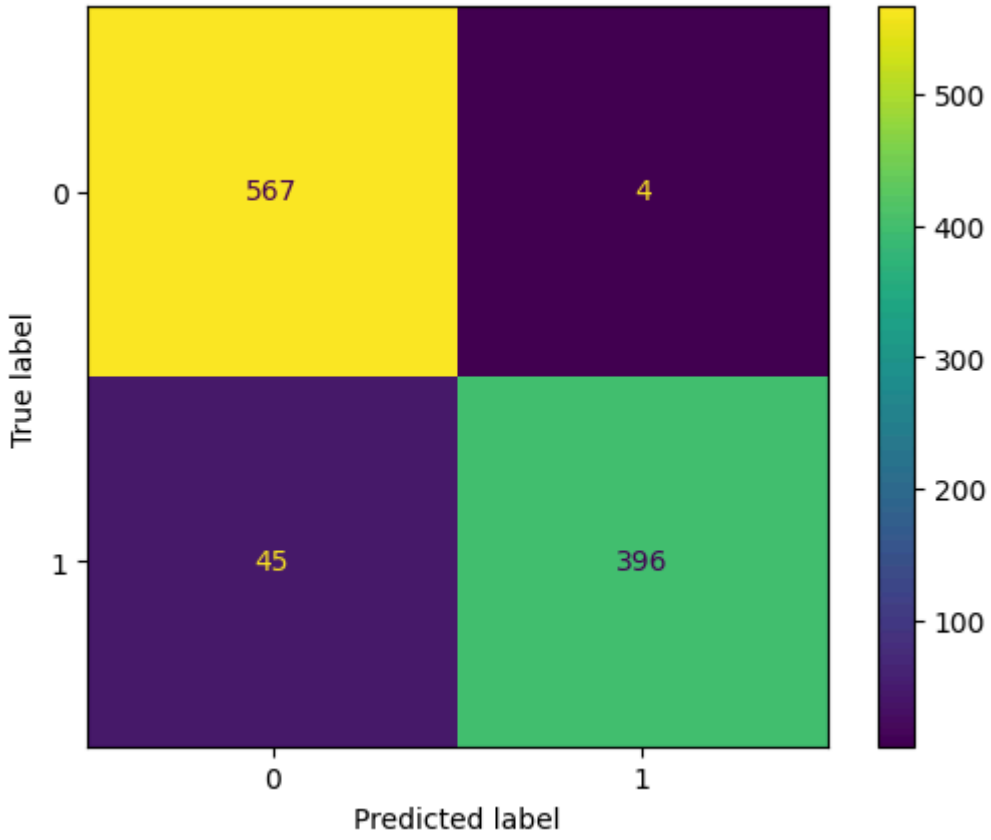
```
print("Accuracy: " + str(Accuracy_svmpoly2))
print("F1 score: " + str(F1_svmpoly2))
print("Recall score: " + str(Precision_svmpoly2))
print("Precision score: " + str(Recall_svmpoly2))
```

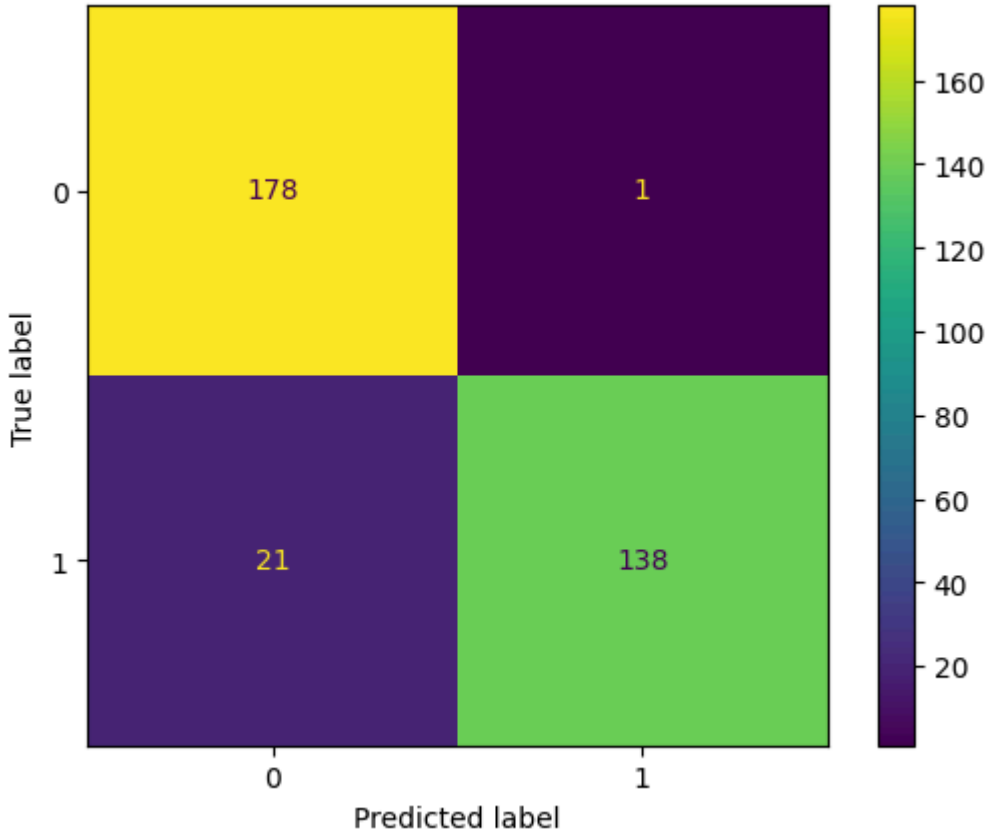
```
Accuracy: 0.8757396449704142
F1 score: 0.8478260869565217
Recall score: 1.0
Precision score: 0.7358490566037735
```

– Generate and plot the confusion matrices for both cases separately.

In [40]:
```
#Confusion matrix poly 3rd
ConfusionMatrixDisplay.from_estimator(clf_svmpoly1, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmpoly1, X_test, y_test)
```

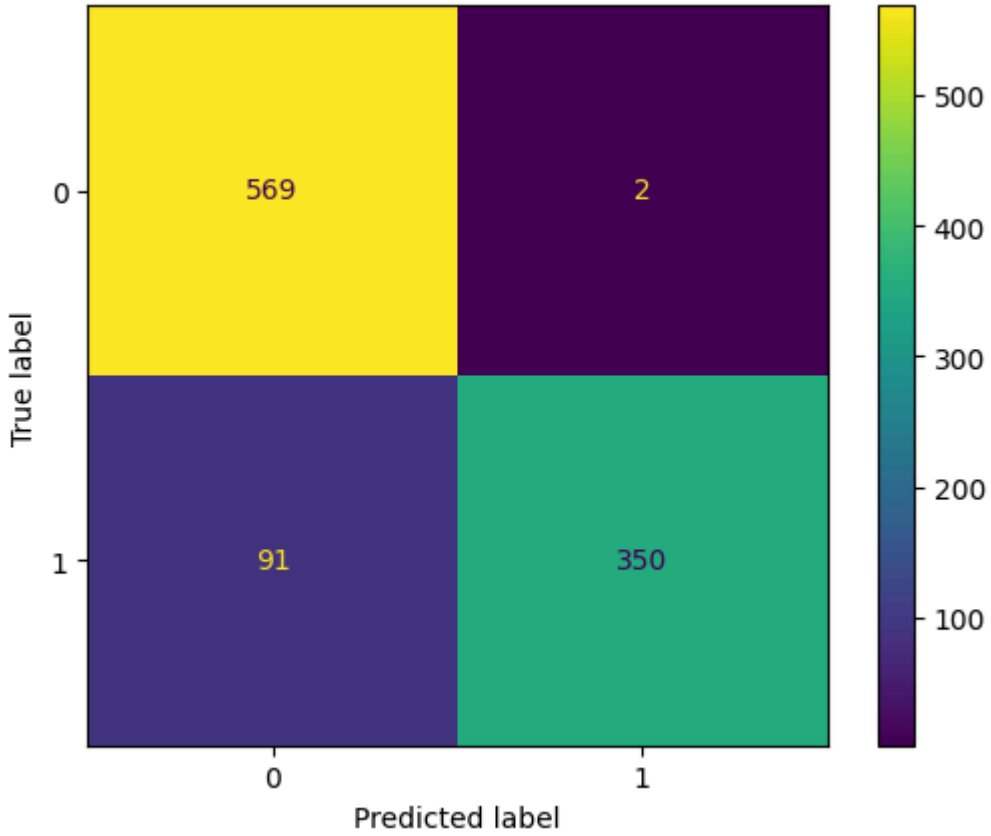
Out[40]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0041743b0
>
```

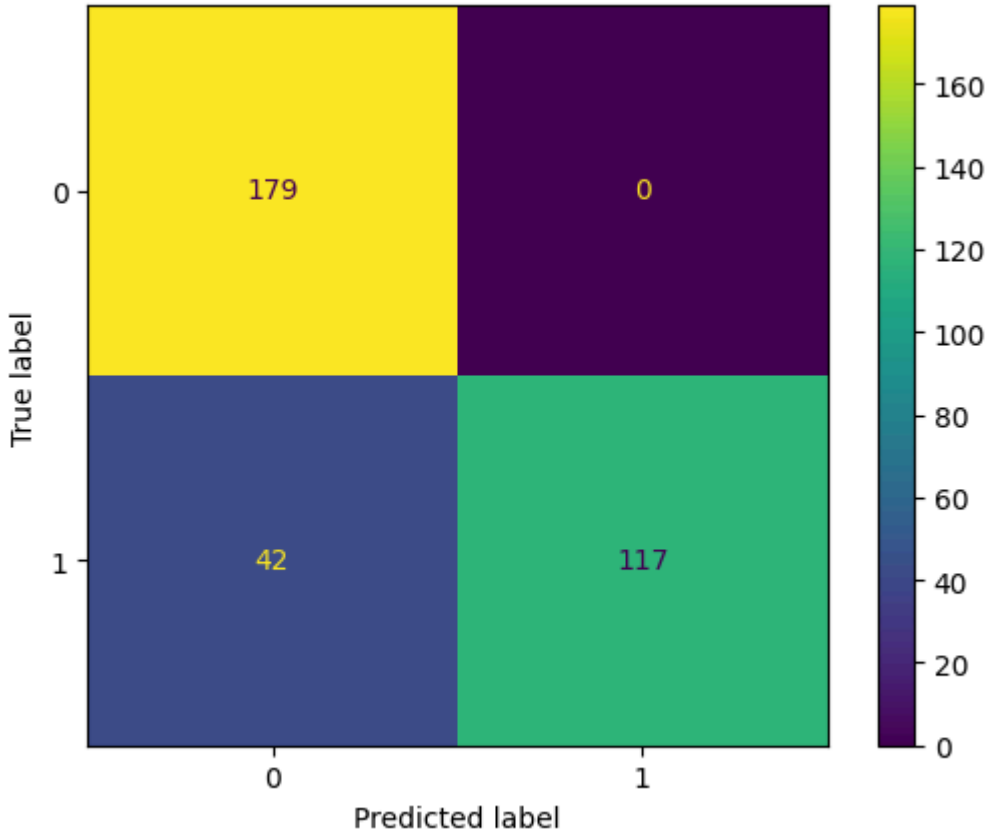




In [41]:
```
#Confusion matrix poly 2nd
ConfusionMatrixDisplay.from_estimator(clf_svmpoly2, X_train, y_train)
ConfusionMatrixDisplay.from_estimator(clf_svmpoly2, X_test, y_test)
```

Out[41]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0048dc2c0
>
```





Questions
1) Compare linear SVM and logistic regression models.
2) What did you observe when parameter C is varied? Explain the role of parameter C in SVC.
3) What did you observe when tolerance is changed to 10? Explain the role of parameter tolerance in SVC.
4) Which SVM model do you prefer for this problem, polynomial or linear kernel? Why? What is the potential benefit of using a nonlinear kernel?
5) Rank the models from the highest performance to the lowest one with F1 scores.

Task 4: Multi-class classification

Multi-class classification predicts the best among a set of labels (four, in this case). We consider two different model classes (logistic regression and SVM) and within the logistic regression model, both the One vs. All and the multi-class approach.
The One vs. All (called 'one versus rest' in sklearn) method constructs binary classifiers for each class. The final prediction is based on the model with the highest score or the largest (signed) distance from the classification boundary.

-Split the data (X, and y_mc) into training (75%) and test sets (25%) using the "train_test_split" function from Sklearn. Use 4720 as the random state parameter to control the split.

In [42]:
```
Test_ratio = 0.25
Shuffle_state = 4720
```

```
X_train_mc, X_test_mc, y_train_mc, y_test_mc = train_test_split(X_scaled,y_mc_da
```

-Develop and train a logistic regression model using the one vs. rest (parameter multi_class='ovr') method using the "y_mc" label vector.

In [43]:
```
clf_lr_ovr = LogisticRegression(
    multi_class='ovr',
    solver='lbfgs',
    max_iter=1000,
    random_state=4720
)

# Train the model
clf_lr_ovr.fit(X_train_mc, y_train_mc)

# Make predictions
y_prediction_lrovr = clf_lr_ovr.predict(X_test_mc)
```

```
c:\Users\User\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\l
inear_model\_logistic.py:1256: FutureWarning: 'multi_class' was deprecated in ver
sion 1.5 and will be removed in 1.7. Use OneVsRestClassifier(LogisticRegression
(..)) instead. Leave it to its default value to avoid this warning.
  warnings.warn(
```

– Calculate and print the accuracy of the model.

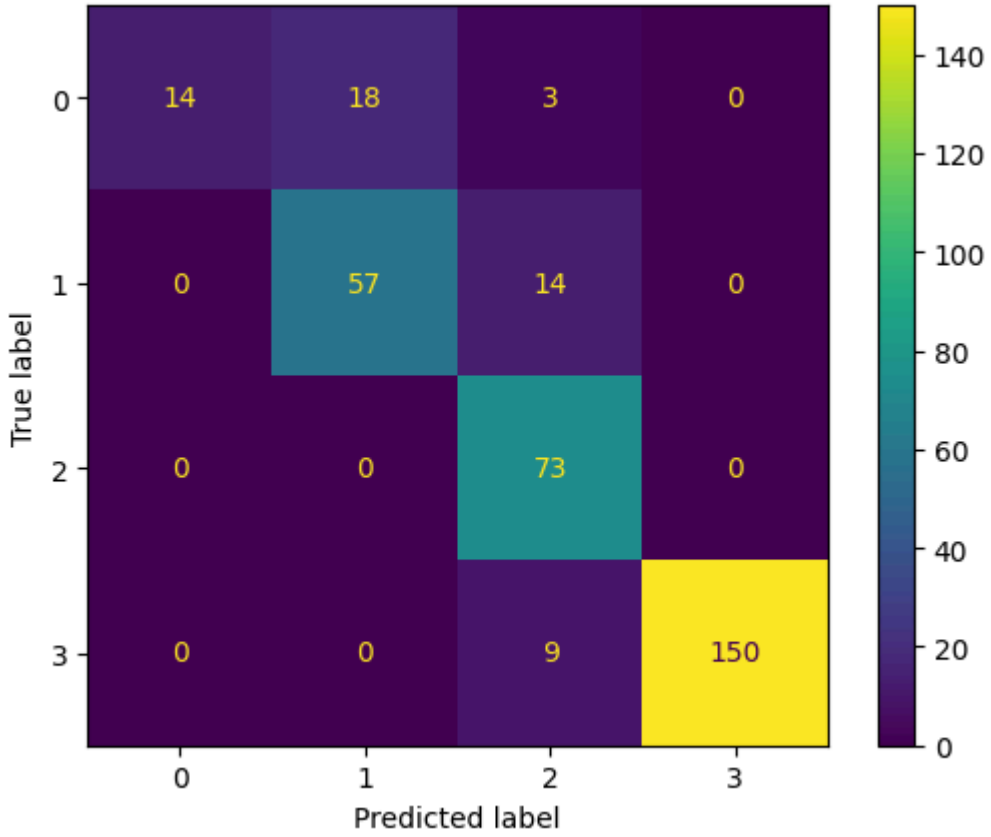
In [44]:
```
#Accuracy of OVR LR
Accuracy_lrovr = accuracy_score(y_test_mc,y_prediction_lrovr)
print("Accuracy: " + str(Accuracy_lrovr))
```

```
Accuracy: 0.8698224852071006
```

–Generate and plot the confusion matrix.

In [45]:
```
#Confusion matrix OVR LR
ConfusionMatrixDisplay.from_estimator(clf_lr_ovr, X_test_mc, y_test_mc)
```

Out[45]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c003fdd7f0
>
```



For logistic regression, we can also directly train a multi-class classifier using the multi-class cross-entropy loss (sklearn calls this the multinomial cross-entropy loss).

−Develop and train a multi-class logistic regression model (parameter multi_class='multinomial')

In [46]:
```
clf_lr_mult = LogisticRegression(
    multi_class='multinomial',
    solver='lbfgs',
    max_iter=1000,
    random_state=4720
)

# Train the model
clf_lr_mult.fit(X_train_mc, y_train_mc)

# Make predictions
y_prediction_lrmult = clf_lr_mult.predict(X_test_mc)
```

```
c:\Users\User\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
```

− Calculate and print the accuracy of the model.

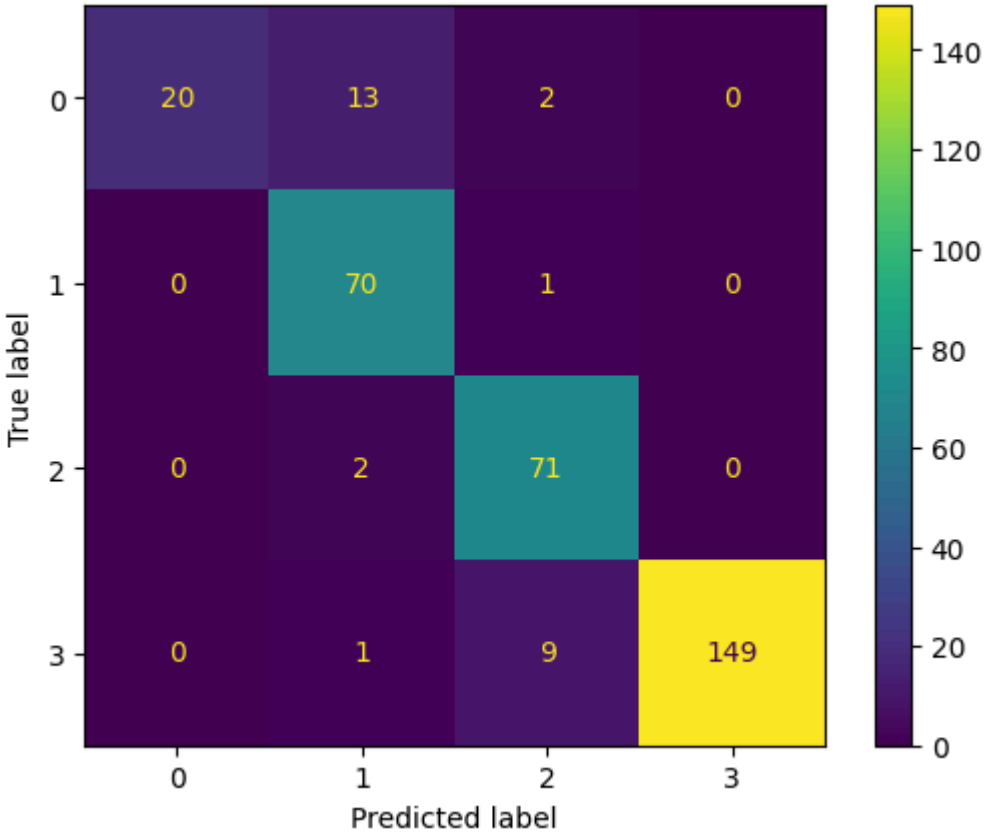
In [47]:
```
#Accuracy of Mult LR
Accuracy_lrmult = accuracy_score(y_test_mc,y_prediction_lrmult)
print("Accuracy: " + str(Accuracy_lrmult))
```

```
Accuracy: 0.9171597633136095
```

−Generate and plot the confusion matrix.

In [48]:
```
#Confusion matrix mult LR
ConfusionMatrixDisplay.from_estimator(clf_lr_mult, X_test_mc, y_test_mc)
```

Out[48]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c006c71b50
>
```



SVM can be used in multi-class classification tasks as well, but it will rely on the one-versus-all approach (or, for non-linear kernels, on the related one-versus-one approach).

- Develop and train a linear SVC [kernel='linear'] as a one vs. all classifier with the given parameters [C=1.0, coef0=0.0, tol=1e-3].

In [49]:
```
#OVR SVM
clf_svmlin_mc_ovr = svm.SVC(
    kernel='linear',
    degree=3,
    C=1.0,
    coef0=0.0,
    tol=1e-3,
    random_state=Shuffle_state
)

# Train the model
clf_svmlin_mc_ovr.fit(X_train_mc, y_train_mc)
y_prediction_svmovr = clf_svmlin_mc_ovr.predict(X_test_mc)
```

− Calculate and print the accuracy of the model.

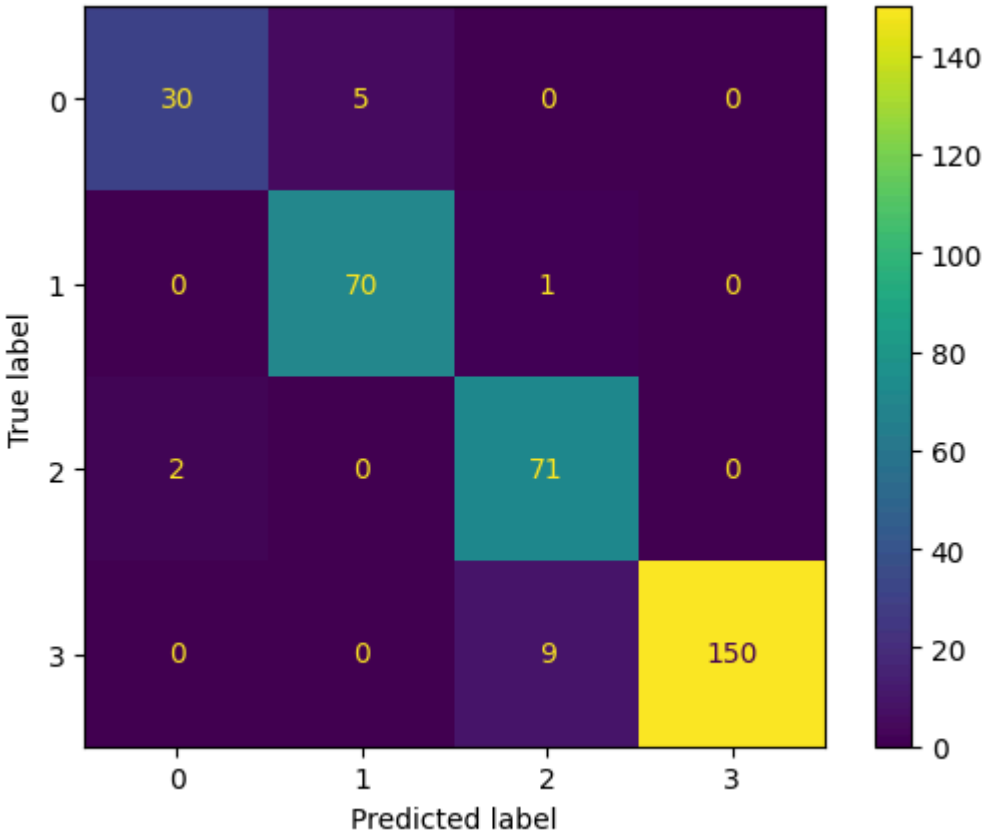
In [50]:
```
#Accuracy of SVM
Accuracy_svmovr = accuracy_score(y_test_mc,y_prediction_svmovr)
print("Accuracy: " + str(Accuracy_svmovr))
```

```
Accuracy: 0.9497041420118343
```

−Generate and plot the confusion matrix.

In [51]:
```
#Confusion matrix SVM OVR
ConfusionMatrixDisplay.from_estimator(clf_svmlin_mc_ovr, X_test_mc, y_test_mc)
```

Out[51]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1c0046c4920
>
```



Questions:

1) How does the performance change for both classification methods for the logistic regression and why?

2) Which classification method is more computationally demanding and why?

3) Why other classification metrics cannot be used directly in multi-class classification tasks?

4) Why can multi-class classification be directly applied to logistic regression but not SVM?

5) Compare all three models' performance in multi-class classification.

6) Which classes have higher mismatches in logistic regression?

7) Which classes have higher mismatches in SVM?

Bonus Task

Conduct a multi-class classification using only binary classifiers. You can use either logistic regression or SVM. Describe the steps you follow and how it works. Calculate each classifier's output probabilities for the test data and plot them in ascending order.

In [ ]: