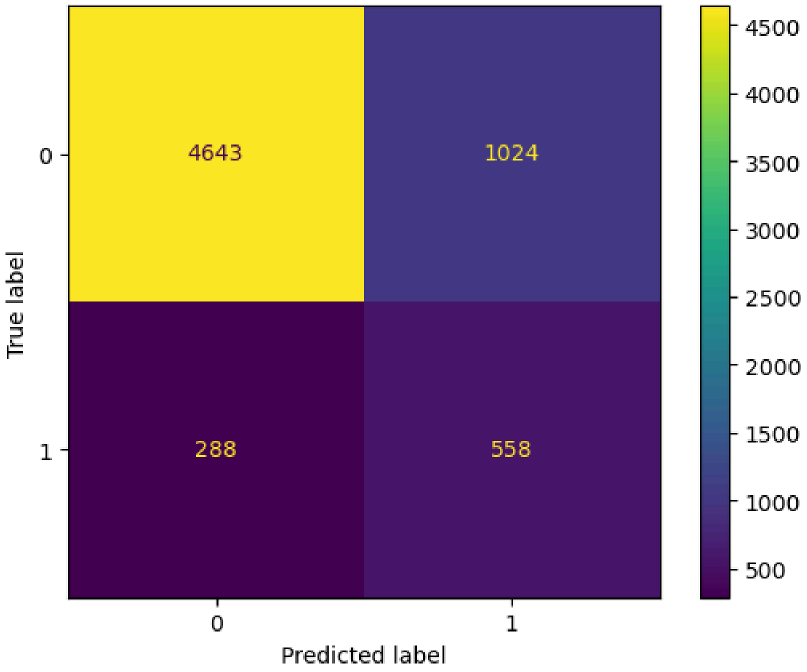
In [50]:
```
cmd = ConfusionMatrixDisplay(cm)
cmd.plot()
plt.show()
```



KNN: K-Nearest Neighbors

KNN is a simple Machine learning algorithm that classifies a new data point based on the majority vote of its k nearest neighbors.

d = /-----------------------
   \/(x2-x1)^2+(y2-y1)^2

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
```

Out[51]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [52]:

```
pred2 = knn.predict(x_test)
pred2
```

Out[52]:

```
array([0, 1, 0, ..., 0, 0, 0])
```

In [53]:

```
from sklearn.metrics import accuracy_score,recall_score,f1_score,precision_score
acc = accuracy_score(pred2,y_test)
recall = recall_score(pred2,y_test)
f1 = f1_score(pred2,y_test)
precision = precision_score(pred2,y_test)
print(f"Accuaracy Score : {acc}")
print(f"Recall Score : {recall}")
print(f"f1 Score : {f1}")
print(f"Precision Score : {precision}")
```

```
Accuaracy Score : 0.8251189927836634
Recall Score : 0.662747979426892
f1 Score : 0.6129799524294938
Precision Score : 0.5701643489254109
```

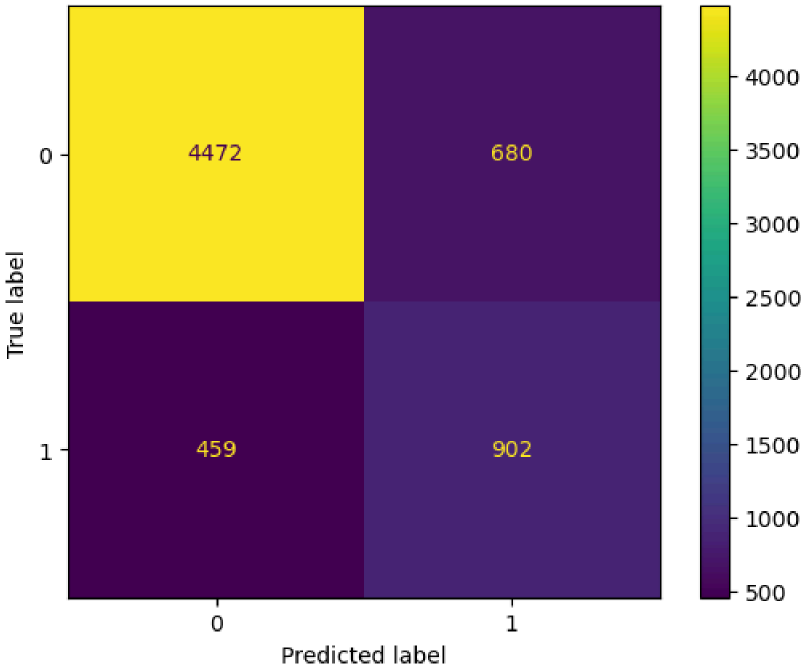
In [54]:

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
cm = confusion_matrix(pred2,y_test)
cm
```

Out[54]:

```
array([[4472,  680],
       [ 459,  902]], dtype=int64)
```

In [55]:
```
cmd = ConfusionMatrixDisplay(cm)
cmd.plot()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

SVM(Support Vector Machine)

SVM is a powerful machine learning algorithm used for classification and regression tasks. It works by finding the best hyperplane that separates the data into different classes.

1. Kernel trick: Uses kernel functions to transform data into higher-dimensional space.
2. Maximum margin: Finds the hyperplane with the maximum margin between classes.
3. Support vectors: Uses support vectors to define the decision boundary.

Here are some common kernel functions used in SVM:

```
1. Linear Kernel: 1D-2D
2. Polynomial Kernel: 1D-3D
3. Radial Basis Function (RBF) Kernel: 1D-2D-3D
4. Sigmoid Kernel: 2D-3D
1. Linear Kernel: Simple, fast, but limited to linearly separable data.
2. Polynomial Kernel: Handles non-linear data, but can be computationally expensive.
3. RBF Kernel: Popular choice, handles non-linear data, and is robust to noise.
4. Sigmoid Kernel: Similar to RBF, but can be less robust to noise.
```

In [56]:
```
from sklearn.svm import SVC
svm = SVC(kernel='rbf',C=1000,gamma=0.01)
svm.fit(x_train,y_train)
```

Out[56]:
```
▾ SVC
SVC(C=1000, gamma=0.01)
```

In [57]:
```
pred3 = svm.predict(x_test)
pred3
```

Out[57]:
```
array([0, 1, 0, ..., 0, 0, 0])
```

In [61]:
```
from sklearn.metrics import accuracy_score,recall_score,f1_score,precision_score
acc = accuracy_score(pred3,y_test)
recall = recall_score(pred3,y_test)
f1 = f1_score(pred3,y_test)
precision = precision_score(pred3,y_test)
print(f"Accuaracy Score : {acc}")
print(f"Recall Score : {recall}")
print(f"f1 Score : {f1}")
print(f"Precision Score : {precision}")
```

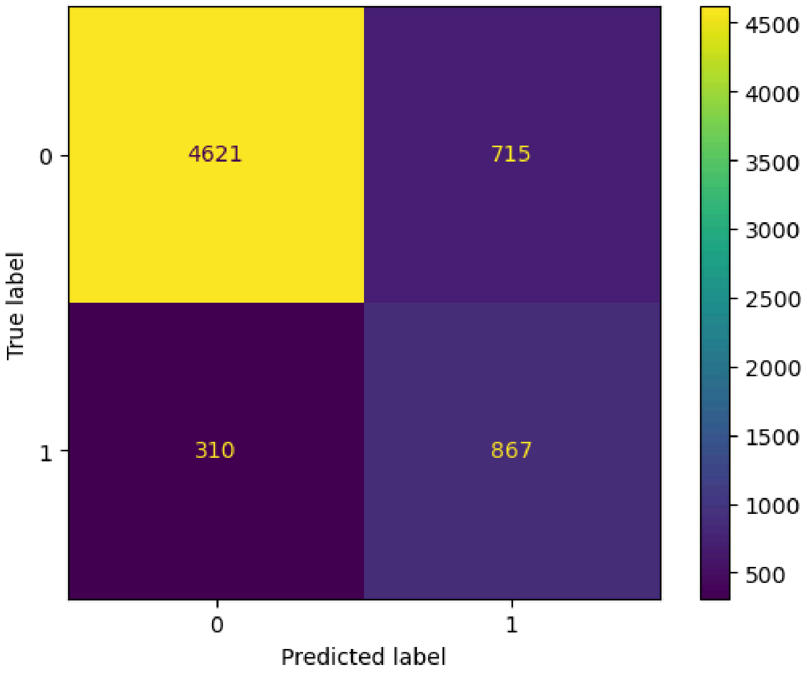
```
Accuaracy Score : 0.8426224474128666
Recall Score : 0.7366185216652507
f1 Score : 0.6284885828198623
Precision Score : 0.5480404551201011
```

```
In [ ]:
```

```
In [62]: from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
         cm = confusion_matrix(pred3,y_test)
         cm
```

```
Out[62]: array([[4621,  715],
                [ 310,  867]], dtype=int64)
```

```
In [63]: cmd = ConfusionMatrixDisplay(cm)
         cmd.plot()
         plt.show()
```



```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```