In [1]:
```python
import pandas as pd
import numpy as np
from sklearn import metrics
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```python
df = pd.read_csv("smart_grid_stability_augmented.csv")
df.head()
```

Out[2]:

	tau1	tau2	tau3	tau4	p1	p2	p3	p4	g1	g2	g3	g4	stab	stabf
0	2.959060	3.079885	8.381025	9.780754	3.763085	-0.782604	-1.257395	-1.723086	0.650456	0.859578	0.887445	0.958034	0.055347	unstable
1	9.304097	4.902524	3.047541	1.369357	5.067812	-1.940058	-1.872742	-1.255012	0.413441	0.862414	0.562139	0.781760	-0.005957	stable
2	8.971707	8.848428	3.046479	1.214518	3.405158	-1.207456	-1.277210	-0.920492	0.163041	0.766689	0.839444	0.109853	0.003471	unstable
3	0.716415	7.669600	4.486641	2.340563	3.963791	-1.027473	-1.938944	-0.997374	0.446209	0.976744	0.929381	0.362718	0.028871	unstable
4	3.134112	7.608772	4.943759	9.857573	3.525811	-1.125531	-1.845975	-0.554305	0.797110	0.455450	0.656947	0.820923	0.049860	unstable

In [3]:
```python
df.shape
```

Out[3]:
```
(60000, 14)
```

Data Cleaning

In [4]:
```python
df.isnull().sum()
```

Out[4]:
```
tau1     0
tau2     0
tau3     0
tau4     0
p1       0
p2       0
p3       0
p4       0
g1       0
g2       0
g3       0
g4       0
stab     0
stabf    0
dtype: int64
```

since there is no null values in all the columns , we proceed to next step

Data Preprocessing

In [5]:
```python
df.dtypes
```

Out[5]:
```
tau1     float64
tau2     float64
tau3     float64
tau4     float64
p1       float64
p2       float64
p3       float64
p4       float64
g1       float64
g2       float64
g3       float64
g4       float64
stab     float64
stabf     object
dtype: object
```

In [6]:
```python
from sklearn.preprocessing import LabelEncoder
l = LabelEncoder()
df['stabf'] = l.fit_transform(df['stabf'])
```

In [7]:
```python
df.value_counts('stabf')
```

Out[7]:
```
stabf
1    38280
0    21720
dtype: int64
```

Splitting data into training and test data

In [8]:
```python
x = df.drop('stabf',axis = 1)
y = df['stabf']
print(x.shape)
print(y.shape)
```

```
(60000, 13)
(60000,)
```

In [9]:
```python
from sklearn.model_selection import train_test_split
```

In [10]:
```python
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25)
```

In [11]:
```python
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(45000, 13)
(45000,)
(15000, 13)
(15000,)
```

Model Implimentation

In [12]:
```python
from sklearn.metrics import confusion_matrix,classification_report
```

1.Logistic Regression

In [13]:
```python
from sklearn.linear_model import LogisticRegression
```

In [14]:
```python
m1 = LogisticRegression(max_iter = 1000)
m1.fit(x_train,y_train)
print('Training score',m1.score(x_train,y_train))
print('Testing score',m1.score(x_test,y_test))
ypred = m1.predict(x_test)
print(ypred)
cm = confusion_matrix(y_test,ypred)
print('Confusion_Matrix\n',cm)
print('Classification Report\n',classification_report(y_test,ypred,zero_division=0))
```

```
Training score 0.96
Testing score 0.9611333333333333
[1 1 1 ... 0 1 1]
Confusion_Matrix
 [[5159  352]
 [ 231 9258]]
Classification Report
               precision    recall  f1-score   support

           0       0.96      0.94      0.95      5511
           1       0.96      0.98      0.97      9489

    accuracy                           0.96     15000
   macro avg       0.96      0.96      0.96     15000
weighted avg       0.96      0.96      0.96     15000
```

In [15]:
```python
cm = pd.DataFrame(data=confusion_matrix(y_test, ypred, labels=[0, 1]),
                  index=["Actual Unstable", "Actual Stable"],
                  columns=["Predicted Unstable", "Predicted Stable"])
cm
```

Out[15]:

	Predicted Unstable	Predicted Stable
Actual Unstable	5159	352
Actual Stable	231	9258

In [16]:
```python
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) / len(y_test) * 100):.2f}%')
```

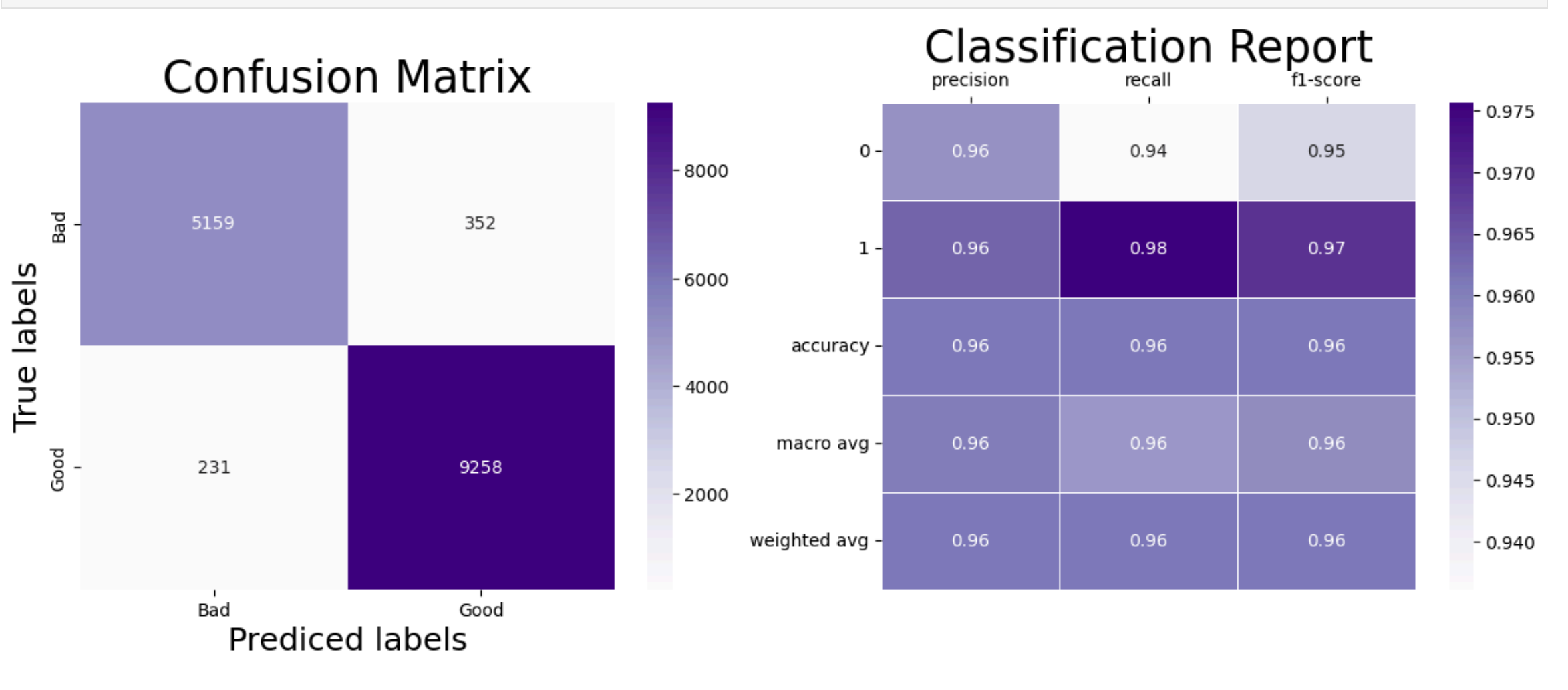
```
Accuracy per the confusion matrix: 96.11%
```

In [17]:
```python
def plot_confusion_matrix(y_test, ypred):
    '''Plotting Confusion Matrix'''
    cm = metrics.confusion_matrix(y_test, ypred)
    ax = plt.subplot()
    ax = sns.heatmap(cm, annot=True, fmt='', cmap="Purples")
    ax.set_xlabel('Prediced labels', fontsize=18)
    ax.set_ylabel('True labels', fontsize=18)
    ax.set_title('Confusion Matrix', fontsize=25)
    ax.xaxis.set_ticklabels(['Bad', 'Good', 'Middle'])
    ax.yaxis.set_ticklabels(['Bad', 'Good', 'Middle'])
    plt.show()
```

In [18]:
```python
def clfr_plot(y_test, y_pred) :
    ''' Plotting Classification report'''
    cr = pd.DataFrame(metrics.classification_report(y_test, y_pred_rf, digits=3,
                                                    output_dict=True)).T
    cr.drop(columns='support', inplace=True)
    sns.heatmap(cr, cmap='Purples', annot=True, linecolor='white', linewidths=0.5).xaxis.tick_top()
```

In [19]:
```python
def clf_plot(y_pred) :
    '''
    1) Ploting Confusion Matrix
    2) Plotting Classification Report'''
    cm = metrics.confusion_matrix(y_test, ypred)
    cr = pd.DataFrame(metrics.classification_report(y_test, ypred, digits=3, output_dict=True)).T
    cr.drop(columns='support', inplace=True)

    fig, ax = plt.subplots(1, 2, figsize=(15, 5))

    # Left AX : Confusion Matrix
    ax[0] = sns.heatmap(cm, annot=True, fmt='', cmap="Purples", ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['Bad', 'Good'])
    ax[0].yaxis.set_ticklabels(['Bad', 'Good'])

    # Right AX : Classification Report
    ax[1] = sns.heatmap(cr, cmap='Purples', annot=True, linecolor='white', linewidths=0.5, ax=ax[1])
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [20]:
```python
clf_plot(ypred)
```



Decision Tree

In [21]:
```python
from sklearn.tree import DecisionTreeClassifier
```

In [22]:
```python
m2 = DecisionTreeClassifier(criterion='gini',min_samples_split=20,max_depth=13)
m2.fit(x_train,y_train)
print('Training score',m2.score(x_train,y_train))
print('Testing score',m2.score(x_test,y_test))
ypred = m2.predict(x_test)
print(ypred)
cm = confusion_matrix(y_test,ypred)
print('Confusion_Matrix\n',cm)
print('Classification Report\n',classification_report(y_test,ypred,zero_division=0))
```

```
Training score 1.0
Testing score 1.0
[1 1 1 ... 0 1 1]
Confusion_Matrix
 [[5511    0]
 [   0 9489]]
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      5511
           1       1.00      1.00      1.00      9489

    accuracy                           1.00     15000
   macro avg       1.00      1.00      1.00     15000
weighted avg       1.00      1.00      1.00     15000
```

In [23]:
```python
cm = pd.DataFrame(data=confusion_matrix(y_test, ypred, labels=[0, 1]),
                  index=["Actual Unstable", "Actual Stable"],
                  columns=["Predicted Unstable", "Predicted Stable"])
cm
```

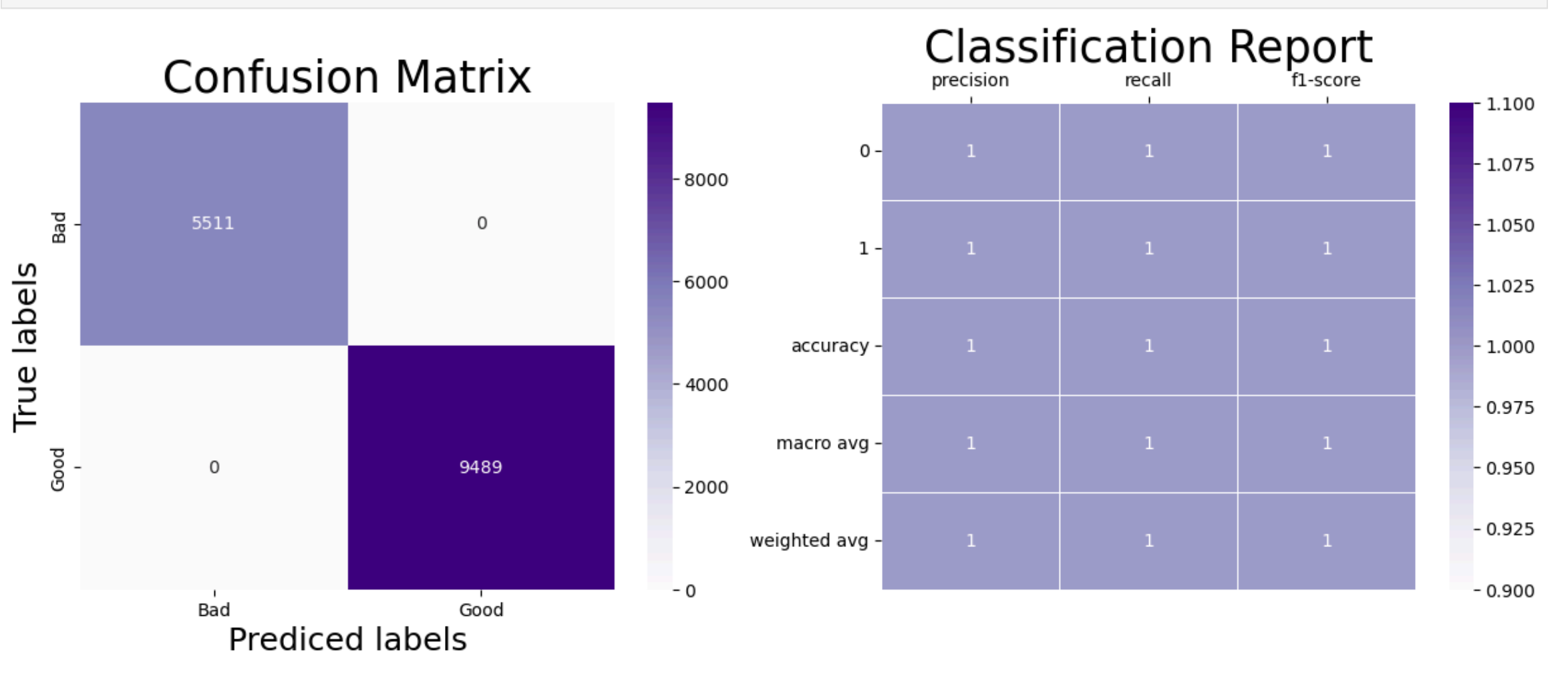
Out[23]:

	Predicted Unstable	Predicted Stable
Actual Unstable	5511	0
Actual Stable	0	9489

In [24]:
```python
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) / len(y_test) * 100):.2f}%')
```

```
Accuracy per the confusion matrix: 100.00%
```

In [25]:
```python
clf_plot(ypred)
```



Random Forest

In [26]:
```python
from sklearn.ensemble import RandomForestClassifier
```

In [27]:
```python
m3 = RandomForestClassifier(n_estimators=51,criterion='entropy',min_samples_split=20,max_depth=13)
m3.fit(x_train,y_train)
print('Training score',m3.score(x_train,y_train))
print('Testing score',m3.score(x_test,y_test))
ypred = m3.predict(x_test)
print(ypred)
cm = confusion_matrix(y_test,ypred)
print('Confusion_Matrix\n',cm)
print('Classification Report\n',classification_report(y_test,ypred,zero_division=0))
```

```
Training score 1.0
Testing score 1.0
[1 1 1 ... 0 1 1]
Confusion_Matrix
 [[5511    0]
 [   0 9489]]
Classification Report
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      5511
           1       1.00      1.00      1.00      9489

    accuracy                           1.00     15000
   macro avg       1.00      1.00      1.00     15000
weighted avg       1.00      1.00      1.00     15000
```

In [28]:
```python
cm = pd.DataFrame(data=confusion_matrix(y_test, ypred, labels=[0, 1]),
                  index=["Actual Unstable", "Actual Stable"],
                  columns=["Predicted Unstable", "Predicted Stable"])
cm
```

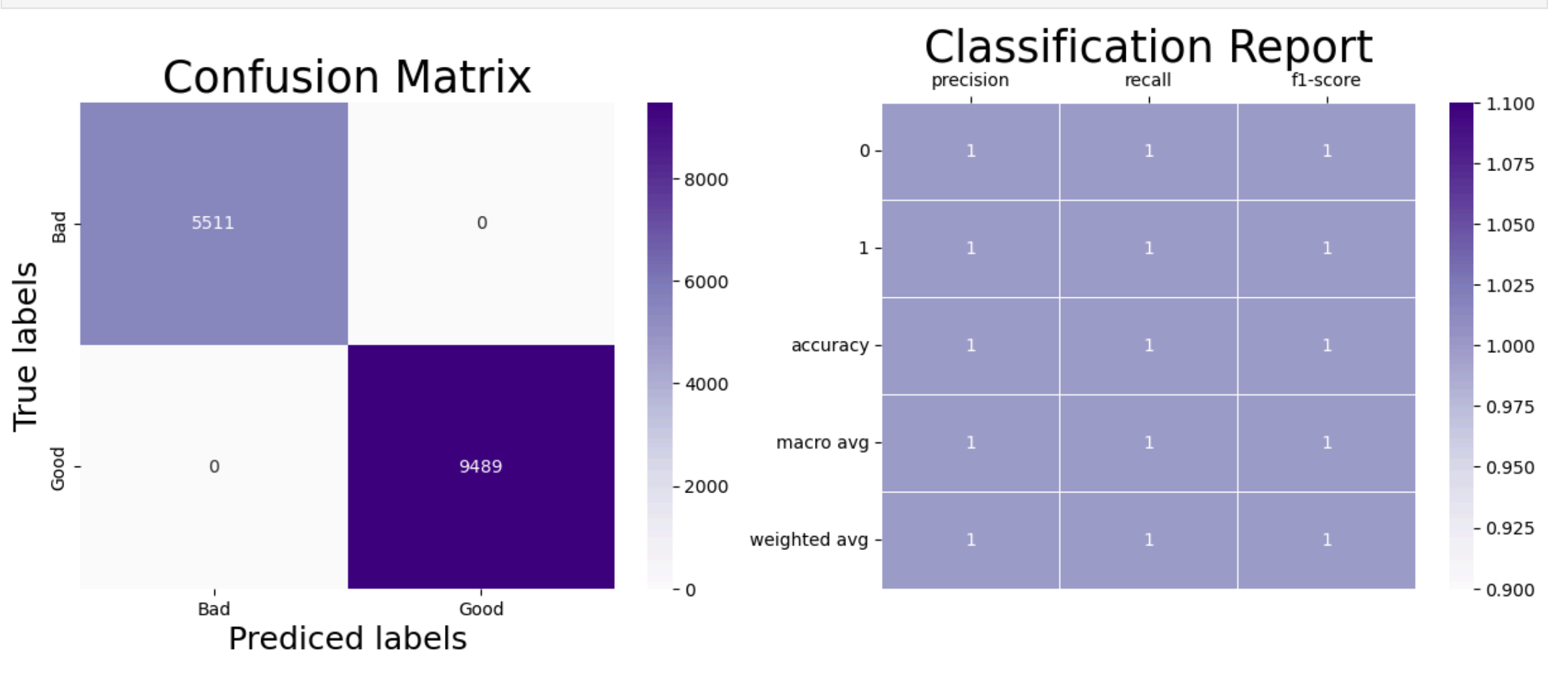
Out[28]:

	Predicted Unstable	Predicted Stable
Actual Unstable	5511	0
Actual Stable	0	9489

In [29]:
```python
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) / len(y_test) * 100):.2f}%')
```

```
Accuracy per the confusion matrix: 100.00%
```

In [30]:
```python
clf_plot(ypred)
```



K Nearest Neighbor

In [31]:
```python
from sklearn.neighbors import KNeighborsClassifier
```

In [32]:
```python
m4 = KNeighborsClassifier(n_neighbors=21)
m4.fit(x_train,y_train)
print('Training score',m4.score(x_train,y_train))
print('Testing score',m4.score(x_test,y_test))
ypred = m4.predict(x_test)
print(ypred)
cm = confusion_matrix(y_test,ypred)
print('Confusion_Matrix\n',cm)
print('Classification Report\n',classification_report(y_test,ypred,zero_division=0))
```

```
Training score 0.8418666666666667
Testing score 0.8196666666666667
[1 1 1 ... 0 1 1]
Confusion_Matrix
 [[3718 1793]
 [ 912 8577]]
Classification Report
               precision    recall  f1-score   support

           0       0.80      0.67      0.73      5511
           1       0.83      0.90      0.86      9489

    accuracy                           0.82     15000
   macro avg       0.82      0.79      0.80     15000
weighted avg       0.82      0.82      0.82     15000
```

In [33]:
```python
cm = pd.DataFrame(data=confusion_matrix(y_test, ypred, labels=[0, 1]),
                  index=["Actual Unstable", "Actual Stable"],
                  columns=["Predicted Unstable", "Predicted Stable"])
cm
```

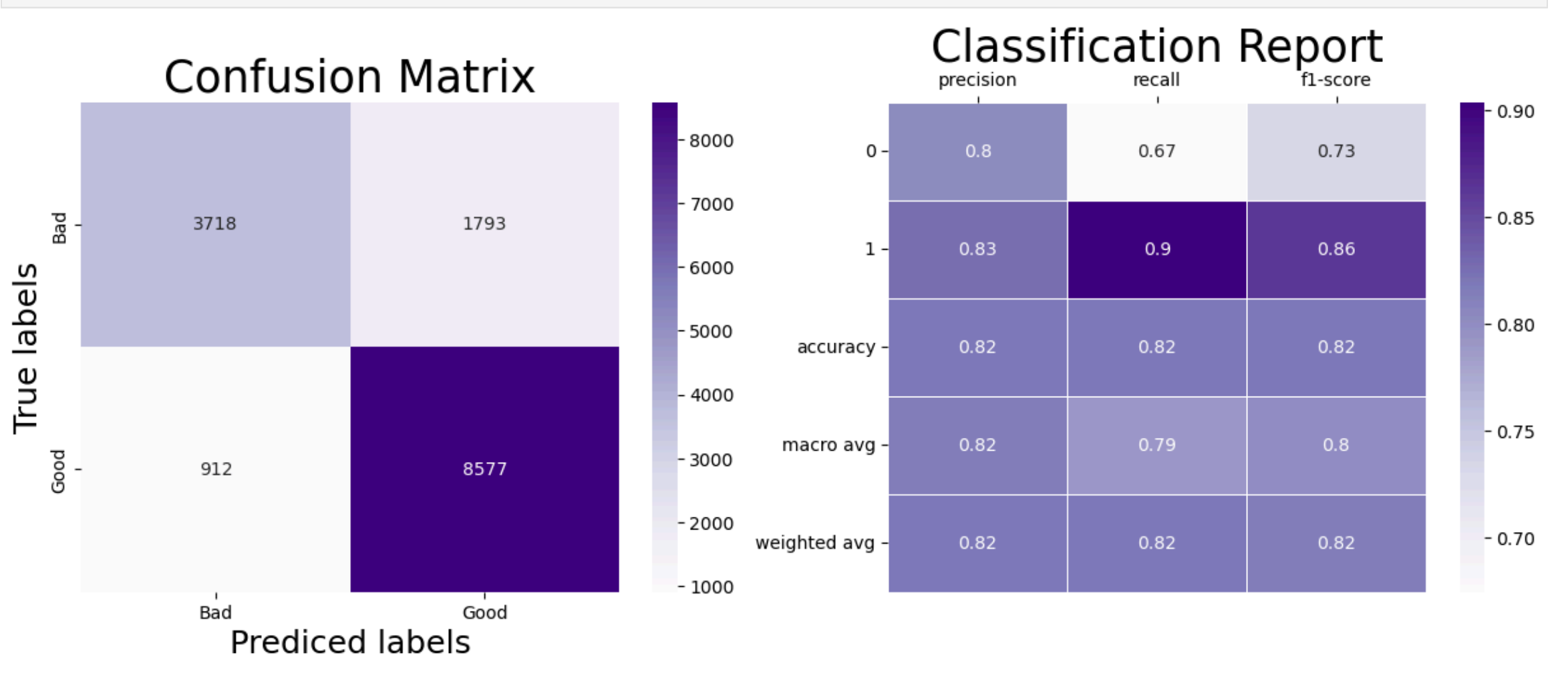
Out[33]:

	Predicted Unstable	Predicted Stable
Actual Unstable	3718	1793
Actual Stable	912	8577

In [34]:
```python
print(f'Accuracy per the confusion matrix: {((cm.iloc[0, 0] + cm.iloc[1, 1]) / len(y_test) * 100):.2f}%')
```

```
Accuracy per the confusion matrix: 81.97%
```

In [35]:
```python
clf_plot(ypred)
```



Artificial Neural Network

In [36]:
```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense
from tensorflow.keras import layers
from tensorflow.keras import models
```

In [38]:
```python
model = keras.models.Sequential([
    keras.layers.Dense(256,activation = 'relu',input_shape = [13]),
    keras.layers.Dense(128,activation = 'relu'),
    keras.layers.Dense(64,activation = 'relu'),
    keras.layers.Dense(32,activation = 'relu'),
    keras.layers.Dense(1,activation = 'sigmoid')
])
```

In [39]:
```python
from tensorflow .keras import optimizers
```

In [43]:
```python
model.compile(loss = 'binary_crossentropy',
             optimizer = optimizers.RMSprop(lr = 1e-4),
             metrics = ['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSpro
p.
```

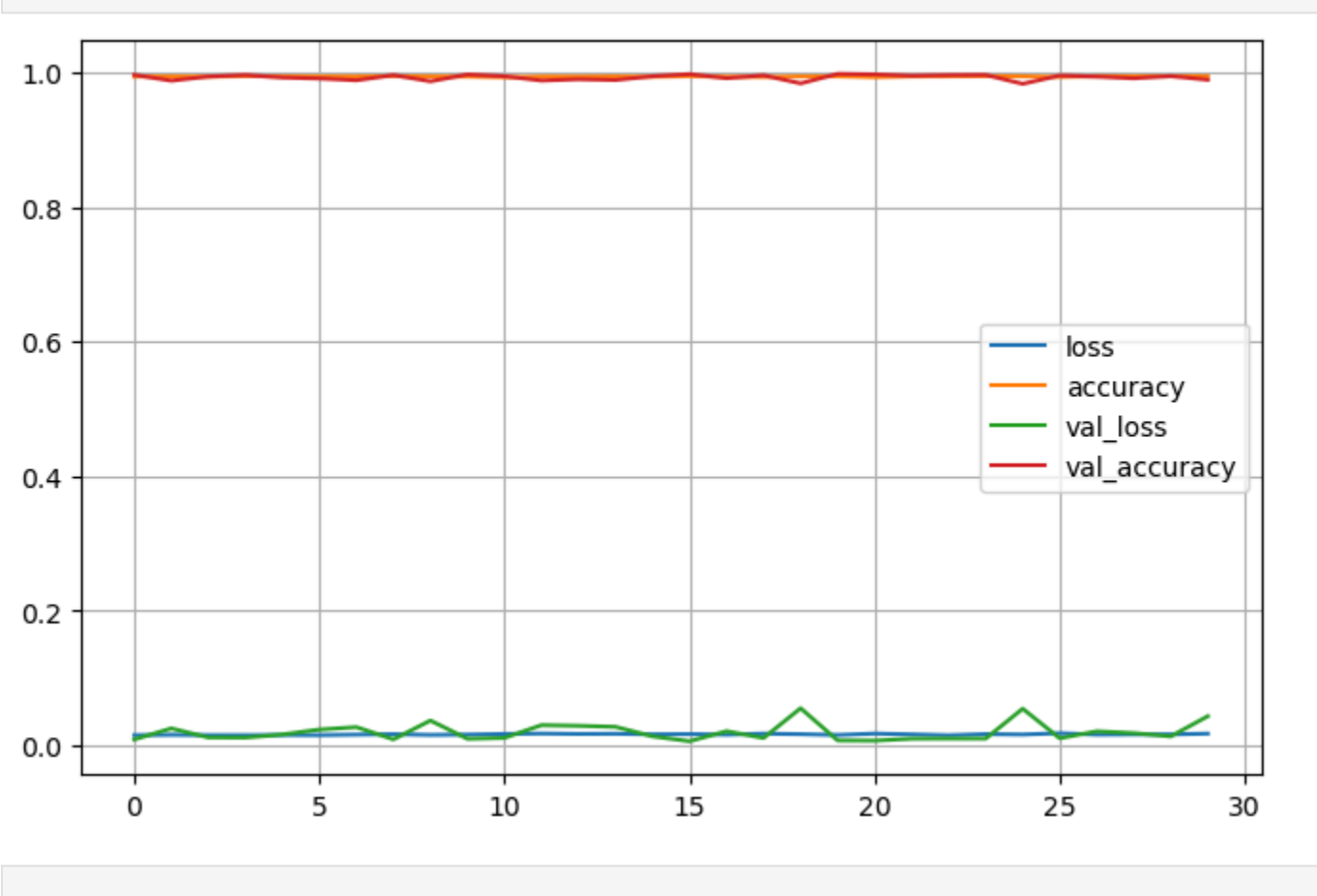
In [44]:
```python
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 256)               3584

 dense_5 (Dense)             (None, 128)               32896

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 32)                2080

 dense_8 (Dense)             (None, 1)                 33

=================================================================
Total params: 46849 (183.00 KB)
Trainable params: 46849 (183.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [45]:
```python
history = model.fit(x_train,y_train,epochs = 30,validation_data = (x_test,y_test))
```

```
Epoch 1/30
1407/1407 [==============================] - 8s 5ms/step - loss: 0.0147 - accuracy: 0.9942 - val_loss: 0.0084 - val_accuracy: 0.9966
Epoch 2/30
1407/1407 [==============================] - 7s 5ms/step - loss: 0.0154 - accuracy: 0.9943 - val_loss: 0.0249 - val_accuracy: 0.9880
Epoch 3/30
1407/1407 [==============================] - 6s 5ms/step - loss: 0.0148 - accuracy: 0.9940 - val_loss: 0.0114 - val_accuracy: 0.9941
Epoch 4/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0147 - accuracy: 0.9947 - val_loss: 0.0112 - val_accuracy: 0.9963
Epoch 5/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0150 - accuracy: 0.9943 - val_loss: 0.0158 - val_accuracy: 0.9927
Epoch 6/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0149 - accuracy: 0.9942 - val_loss: 0.0231 - val_accuracy: 0.9913
Epoch 7/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0156 - accuracy: 0.9943 - val_loss: 0.0267 - val_accuracy: 0.9886
Epoch 8/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0163 - accuracy: 0.9946 - val_loss: 0.0085 - val_accuracy: 0.9963
Epoch 9/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0151 - accuracy: 0.9943 - val_loss: 0.0366 - val_accuracy: 0.9871
Epoch 10/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0156 - accuracy: 0.9941 - val_loss: 0.0094 - val_accuracy: 0.9972
Epoch 11/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0166 - accuracy: 0.9930 - val_loss: 0.0107 - val_accuracy: 0.9946
Epoch 12/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0172 - accuracy: 0.9942 - val_loss: 0.0297 - val_accuracy: 0.9882
Epoch 13/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0166 - accuracy: 0.9941 - val_loss: 0.0288 - val_accuracy: 0.9901
Epoch 14/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0169 - accuracy: 0.9944 - val_loss: 0.0272 - val_accuracy: 0.9890
Epoch 15/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0160 - accuracy: 0.9939 - val_loss: 0.0130 - val_accuracy: 0.9945
Epoch 16/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0164 - accuracy: 0.9946 - val_loss: 0.0058 - val_accuracy: 0.9977
Epoch 17/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0156 - accuracy: 0.9939 - val_loss: 0.0207 - val_accuracy: 0.9917
Epoch 18/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0170 - accuracy: 0.9934 - val_loss: 0.0106 - val_accuracy: 0.9959
Epoch 19/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0163 - accuracy: 0.9944 - val_loss: 0.0549 - val_accuracy: 0.9840
Epoch 20/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0150 - accuracy: 0.9945 - val_loss: 0.0070 - val_accuracy: 0.9981
Epoch 21/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0171 - accuracy: 0.9931 - val_loss: 0.0067 - val_accuracy: 0.9973
Epoch 22/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0160 - accuracy: 0.9944 - val_loss: 0.0096 - val_accuracy: 0.9953
Epoch 23/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0145 - accuracy: 0.9945 - val_loss: 0.0097 - val_accuracy: 0.9962
Epoch 24/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0164 - accuracy: 0.9947 - val_loss: 0.0097 - val_accuracy: 0.9965
Epoch 25/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0158 - accuracy: 0.9945 - val_loss: 0.0541 - val_accuracy: 0.9833
Epoch 26/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0176 - accuracy: 0.9936 - val_loss: 0.0102 - val_accuracy: 0.9956
Epoch 27/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0156 - accuracy: 0.9946 - val_loss: 0.0205 - val_accuracy: 0.9940
Epoch 28/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0159 - accuracy: 0.9945 - val_loss: 0.0178 - val_accuracy: 0.9915
Epoch 29/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0160 - accuracy: 0.9945 - val_loss: 0.0130 - val_accuracy: 0.9948
Epoch 30/30
1407/1407 [==============================] - 6s 4ms/step - loss: 0.0171 - accuracy: 0.9946 - val_loss: 0.0430 - val_accuracy: 0.9893
```

In [46]:
```python
pd.DataFrame(history.history).plot(figsize = (8,5))
plt.grid(True)
plt.show()
```



In [ ]: