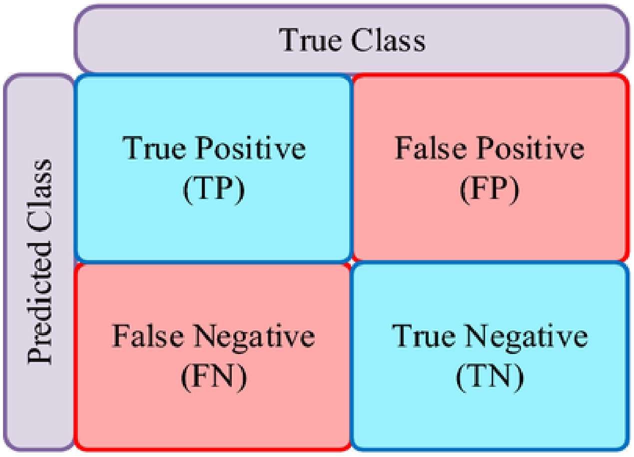
چیست ؟ (Underfitting) سوال ۱ : آندرفیت

نباشد اورفیت Flexible های آن انعطاف پذیر یا Sample آموزش ببیند که نسبت به ویژگی های Train هنگامی که یک مدل آنقدر روی داده های
را هنوز به خوبی استخراج نکرده باشد یا Train های Sample شده باشد که ویژگی های Train رخ داده است . و هنگامی که مدل به صورتی (Overfit)
آن برای اصلاح وزن ها به بهینه سراسری کم و ناکافی باشد و یا تعداد ویژگی ها آن قدر کم باشد که شناخت خوبی به backpropagation آنقدر تعداد
. رخ داده است UnderFitting مدل ندهد . کم برازش یا
چیست ؟ F1-score سوال ۲ : معیار حساسیت یا

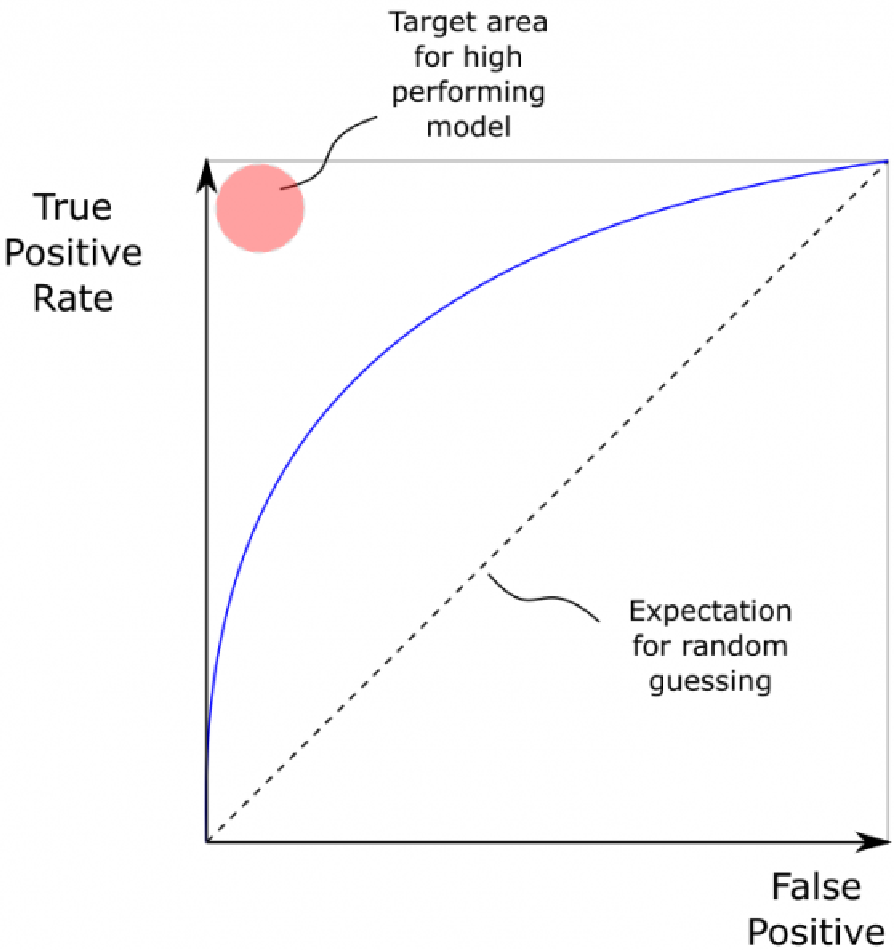
ابزارهایی هستند که برای ارزیابی عملکرد مدل‌های طبقه‌بندی (F1-score) F1 و اسکور (Precision) دقت ،(Recall) معیارهای حساسیت
گرفته می‌شوند. ماتریس درهم‌ریختگی شامل چهار (Confusion Matrix) استفاده می‌شوند. این معیارها از ماتریس درهم‌ریختگی (classification)
(False تعداد نمونه‌های اشتباه مثبت ،(True Negative) تعداد نمونه‌های واقعاً منفی ،(True Positive) خانه است: تعداد نمونه‌های واقعاً مثبت
Positive) و تعداد نمونه‌های اشتباه منفی (False Negative).



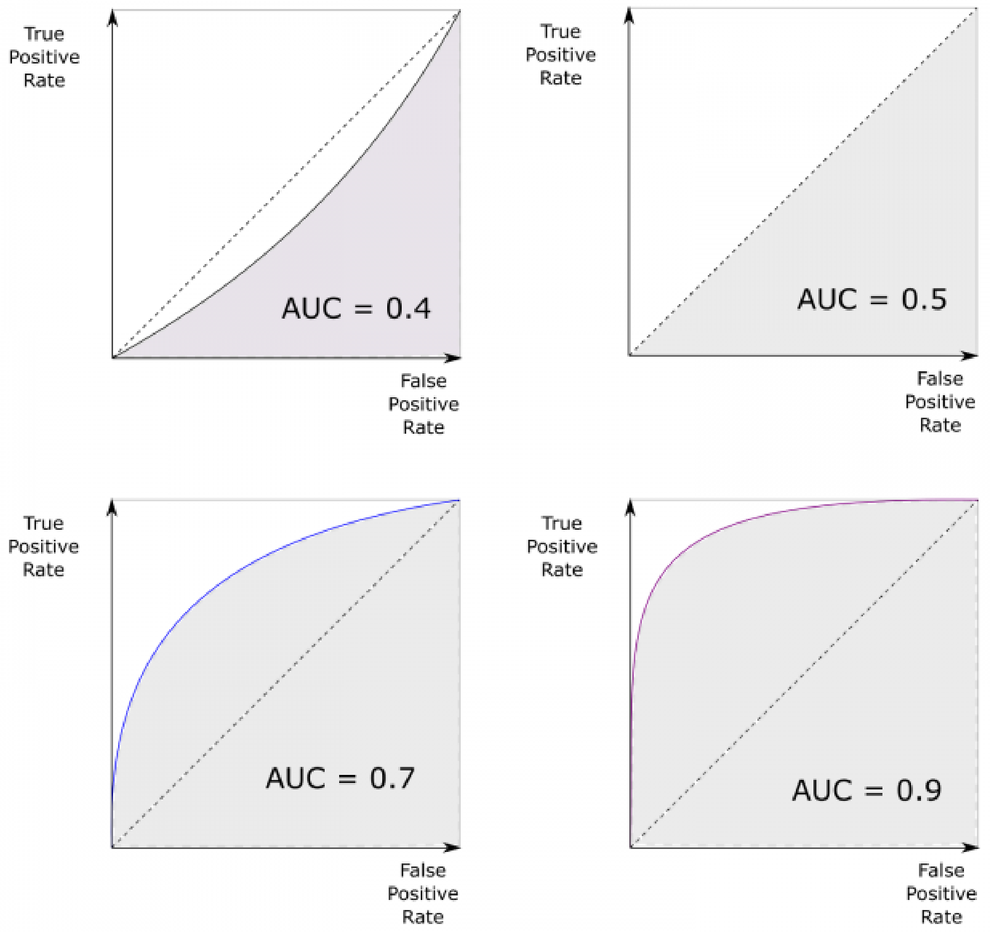
Function Name	Formula
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1 score	$\frac{2.Precision.Recall}{Precision+Recall}$

. را شرح دهید ROC سوال ۳ : معیار یا نمودار

یک ابزار گرافیکی است که برای ارزیابی عملکرد مدل‌های طبقه‌بندی (Receiver Operating Characteristic یا ROC) منحنی عملکرد دریافت‌کننده
False Positive استفاده می‌شود. این منحنی نمایانگر توانایی مدل در تشخیص بین دو کلاس یعنی مثبت و منفی است. در این نمودار، محور افقی
Rate (نرخ اشتباه مثبت) و محور عمودی True Positive Rate (نرخ واقعی مثبت یا حساسیت) قرار دارند. ROC Curve با تغییر آستانه تصمیم، تعداد
مثبت‌های درست و اشتباه مثبت را در مقابل تعداد منفی‌های درست و اشتباه منفی رسم می‌کند. AUC-ROC (مساحت زیر منحنی ROC) نیز یک
معیار جامع است که توانایی کلی مدل در تمایز بین کلاس‌ها را نشان می‌دهد، به‌طوری‌که مقدار بالاتر نشان از عملکرد بهتر مدل در تصمیم‌گیری
ROC است یا میتوان گفت بیشینه کردن انتگرال تابع AUC یا ROC کردن مساحت زیر نمودار Maximum می‌دهد. هدف یک مدل باینری



Rate



تکنیک ONE hot encoding را تحقیق کنید یا روش های دیگر برای رفع مشکل کتگوریکال.

تکنیک One-Hot Encoding یکی از روش‌های رایج برای تبدیل متغیرهای دسته‌ای (categorical variables) به فرمتی قابل استفاده در الگوریتم‌های یادگیری ماشین است. وقتی با داده‌های دسته‌ای سر و کار داریم و می‌خواهیم این داده‌ها را به الگوریتم‌های یادگیری ماشین بدهیم، ممکن است مشکلاتی پیش آید زیرا بسیاری از الگوریتم‌ها با اعداد عددی واژه‌ها و دسته‌ها کار می‌کنند. One-Hot Encoding به این صورت است که برای هر مقدار دسته‌ای ممکن در یک ویژگی، یک ستون جدید ایجاد می‌شود و سطرهای متناظر با داده‌های اصلی، به صورت باینری (0 و 1) پر می‌شوند. به عبارت دیگر، اگر یک مقدار در دسته‌بندی وجود داشته باشد، در ستون مربوطه 1 قرار می‌گیرد و در ستون‌های دیگر 0 می‌ماند. این کار باعث می‌شود که دسته‌های مختلف مستقل از همدیگر در ماتریس قرار گیرند و مدل ماشین قادر به یادگیری روابط بین این دسته‌ها شود.

id	color
1	red
2	blue
3	green
4	blue

One Hot Encoding →

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

روش‌های دیگری نیز برای رفع مشکلات دسته‌بندی و تبدیل داده‌های دسته‌ای به شکل مناسب برای الگوریتم‌های یادگیری ماشین وجود دارد. برخی از این روش‌ها عبارتند از:

1. **Label Encoding:** - در این روش، هر مقدار دسته‌ای به یک عدد نسبت داده می‌شود. این روش برای دسته‌بندی‌هایی که ترتیب خاصی ندارند مناسب است، اما در مواردی که ترتیب اهمیت دارد، ممکن است اشکالاتی ایجاد شود.

1. **Binary Encoding:** - در این روش، هر مقدار دسته‌ای به یک عدد دودویی تبدیل می‌شود. این کار به کاهش ابعاد داده کمک می‌کند و می‌تواند موثر باشد، اما برخی از اطلاعات ممکن است از دست رود.

1. **Ordinal Encoding:** - برای دسته‌بندی‌هایی که ترتیب مهم است، این روش مورد استفاده قرار می‌گیرد. هر مقدار به یک عدد نسبت داده می‌شود و ترتیب اهمیت دسته‌ها حفظ می‌شود. هر یک از این روش‌ها مزایا و معایب خود را دارند و انتخاب بهترین روش بستگی به ماهیت داده‌ها و نوع مدل استفاده شده دارد.

Project 3

- Task 1 : Read (.mat) data and describe it (channels, version, and ...)

- Task 2 : filter EEG data and remove outlier samples
  - subtask 1 : plot graph
  - subtask 2 : plot scatter
- Task 3 : split data and Train binary classifier model
  - subtask 1 : Train SVM (Support Vector Machine)
  - subtask 2 : Train Logistic classifier
  - subtask 3 : Train Random forest
  - subtask 4 : Train KNN classifier model
  - subtask 5 : Train Naive Bayes classifier
- Task 4 : Use Grid search for find best model
  - subtask 1 : Use grid search for Decision Tree classifier
  - subtask 2 : Use grid search for SVM classifier
  - subtask 3 : Use grid search for random forest classifier

Import Libraries

In [1]:

```
# ______________________________________________
# import libraries :
# ______________________________________________

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import *

# for fourier transform
from scipy import stats


# to fetch dataset :
import os
import requests

from numpy import where, arange

# import library for read dataset :
from scipy.io import arff

# Import the required libraries for machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report


# Import models and accuracy algorithms
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier


# method 1 : to calculate accuracy
from sklearn.metrics import accuracy_score

# method 2 : to calculate accuracy
from sklearn.model_selection import cross_val_score

# method 3 : to calculate accuracy (best for binary classification)
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

# library for calculate precision and recall  and f1_score
from sklearn.metrics import precision_score , recall_score
from sklearn.metrics import f1_score

# library for calculate and plot statistics variable
from sklearn.metrics import precision_recall_curve

# for Naive Bayes classifier model
from sklearn.naive_bayes import BernoulliNB

# metric for binary classification
from sklearn.metrics import roc_auc_score

# for calculate FPR and TPR and plot ROC curve
from sklearn.metrics import roc_curve

# library for grid search and random grid search :
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import expon, loguniform
from scipy.stats import randint
```

Download Dataset

In [2]:

```
#
# download dataset
#

github_username = 'darkrams98'
repo_name = 'Brain_Wave_Analysis_python'
file_name = 'EEG_Eye_State.arff'
url = f'https://github.com/{github_username}/{repo_name}/raw/main/Project_Datasets/{file_name}'



# مسیر محل ذخیره فایل در سیستم
save_directory = 'Dataset'  # نام پوشه
save_path = os.path.join(save_directory, file_name)

# بررسی وجود پوشه و ایجاد آن اگر وجود نداشته باشد
if not os.path.exists(save_directory):
    os.makedirs(save_directory)

# بررسی وجود فایل
if not os.path.exists(save_path):
    # اگر فایل وجود نداشته باشد، دانلود کنید
    response = requests.get(url)
    with open(save_path, 'wb') as file:
        file.write(response.content)
    if response.status_code == 200:
        print('The download was successful !')
    else:
        print('An error occurred while downloading !', response.status_code)
else:
    print('The file already exists !')
```

```
The file already exists !
```

Explore Data Structure

In [3]:
```python
# ______________________________________________________________________________
# read dataset
# ______________________________________________________________________________

data_raw, meta_data = arff.loadarff(save_path)

print(type(data_raw))
print(type(meta_data))

# print shape
print("shape EEG data : ",np.shape(data_raw))
print("shape meta data : ",np.shape(meta_data))

# print dataset :
print(data_raw)
print(meta_data)
```

```
<class 'numpy.ndarray'>
<class 'scipy.io.arff._arffread.MetaData'>
shape EEG data :  (14980,)
shape meta data :  ()
[(4329.23, 4009.23, 4289.23, 4148.21, 4350.26, 4586.15, 4096.92, 4641.03, 4222.05, 4238.46, 4211.2
8, 4280.51, 4635.9 , 4393.85, b'0')
 (4324.62, 4004.62, 4293.85, 4148.72, 4342.05, 4586.67, 4097.44, 4638.97, 4210.77, 4226.67, 4207.6
9, 4279.49, 4632.82, 4384.1 , b'0')
 (4327.69, 4006.67, 4295.38, 4156.41, 4336.92, 4583.59, 4096.92, 4630.26, 4207.69, 4222.05, 4206.6
7, 4282.05, 4628.72, 4389.23, b'0')
 ...
 (4277.44, 3990.77, 4246.67, 4113.85, 4333.33, 4615.38, 4072.82, 4623.59, 4193.33, 4212.82, 4160.5
1, 4257.95, 4591.79, 4339.49, b'1')
 (4284.62, 3991.79, 4251.28, 4122.05, 4334.36, 4616.41, 4080.51, 4628.72, 4200.  , 4220.  , 4165.6
4, 4267.18, 4596.41, 4350.77, b'1')
 (4287.69, 3997.44, 4260.  , 4121.03, 4333.33, 4616.41, 4088.72, 4638.46, 4212.31, 4226.67, 4167.6
9, 4274.36, 4597.95, 4350.77, b'1')]
Dataset: EEG_DATA
	AF3's type is numeric
	F7's type is numeric
	F3's type is numeric
	FC5's type is numeric
	T7's type is numeric
	P7's type is numeric
	O1's type is numeric
	O2's type is numeric
	P8's type is numeric
	T8's type is numeric
	FC6's type is numeric
	F4's type is numeric
	F8's type is numeric
	AF4's type is numeric
	eyeDetection's type is nominal, range is ('0', '1')
```

In [4]:

```python
#
# convert data to pandas dataset :
#

# convert
data_raw_df = pd.DataFrame(data_raw)
# get columns :
data_cols = data_raw_df.columns

# print data structure :
print(data_raw_df.shape)
print(data_cols)
data_raw_df.head(10)
```

```
(14980, 15)
Index(['AF3', 'F7', 'F3', 'FC5', 'T7', 'P7', 'O1', 'O2', 'P8', 'T8', 'FC6',
       'F4', 'F8', 'AF4', 'eyeDetection'],
      dtype='object')
```

Out[4]:

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4	eyeDet
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46	4211.28	4280.51	4635.90	4393.85	
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67	4207.69	4279.49	4632.82	4384.10	
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05	4206.67	4282.05	4628.72	4389.23	
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38	4210.77	4287.69	4632.31	4396.41	
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10	4212.82	4288.21	4632.82	4398.46	
5	4321.03	4004.62	4284.10	4153.33	4345.64	4587.18	4093.33	4616.92	4202.56	4232.82	4209.74	4281.03	4628.21	4389.74	
6	4319.49	4001.03	4280.51	4151.79	4343.59	4584.62	4089.74	4615.90	4212.31	4226.67	4201.03	4269.74	4625.13	4378.46	
7	4325.64	4006.67	4278.46	4143.08	4344.10	4583.08	4087.18	4614.87	4205.64	4230.26	4195.90	4266.67	4622.05	4380.51	
8	4326.15	4010.77	4276.41	4139.49	4345.13	4584.10	4091.28	4608.21	4187.69	4229.74	4202.05	4273.85	4627.18	4389.74	
9	4326.15	4011.28	4276.92	4142.05	4344.10	4582.56	4092.82	4608.72	4194.36	4228.72	4212.82	4277.95	4637.44	4393.33	

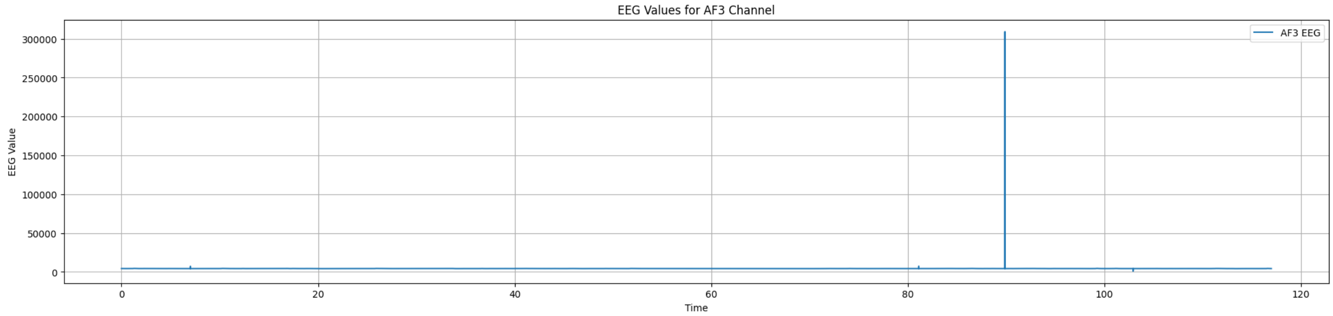
In [5]:

```python
#
# calculate frequency and another statistic variables :
#

# trials for 117 sec of eeg
n_trials = data_raw_df.shape[0]
print(n_trials)

# calculate sampling frequency :
freq_sampling = n_trials / 117
print(freq_sampling)

# calculate time direction :
delta_time = 1 / freq_sampling
time_dir = (np.arange(0, 117, delta_time))[0:-1]
print(time_dir)
print(len(time_dir))
```

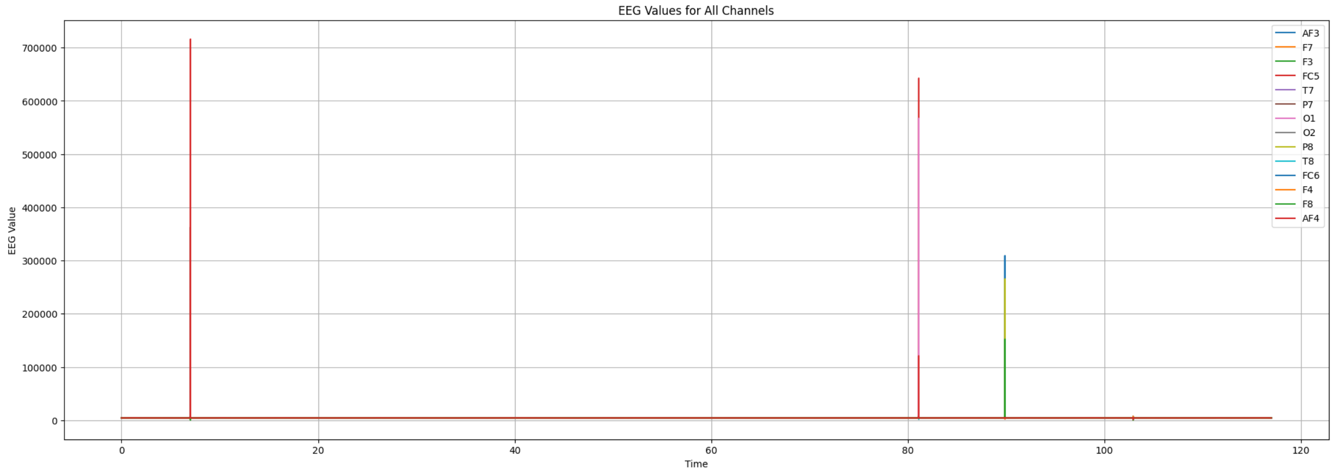
```
14980
128.03418803418805
[0.00000000e+00 7.81041389e-03 1.56208278e-02 ... 1.16976569e+02
 1.16984379e+02 1.16992190e+02]
14980
```

plot EEG and recognize outlier samples

In [6]:
```python
# ─────────────────────────────────────────────
# plot one channel data :
# ─────────────────────────────────────────────

plt.figure(figsize=(24, 5))
plt.plot(time_dir,data_raw_df['AF3'], label='AF3 EEG')
plt.title('EEG Values for AF3 Channel')
plt.xlabel('Time')
plt.ylabel('EEG Value')
plt.legend()
plt.grid(True)
plt.show()
```



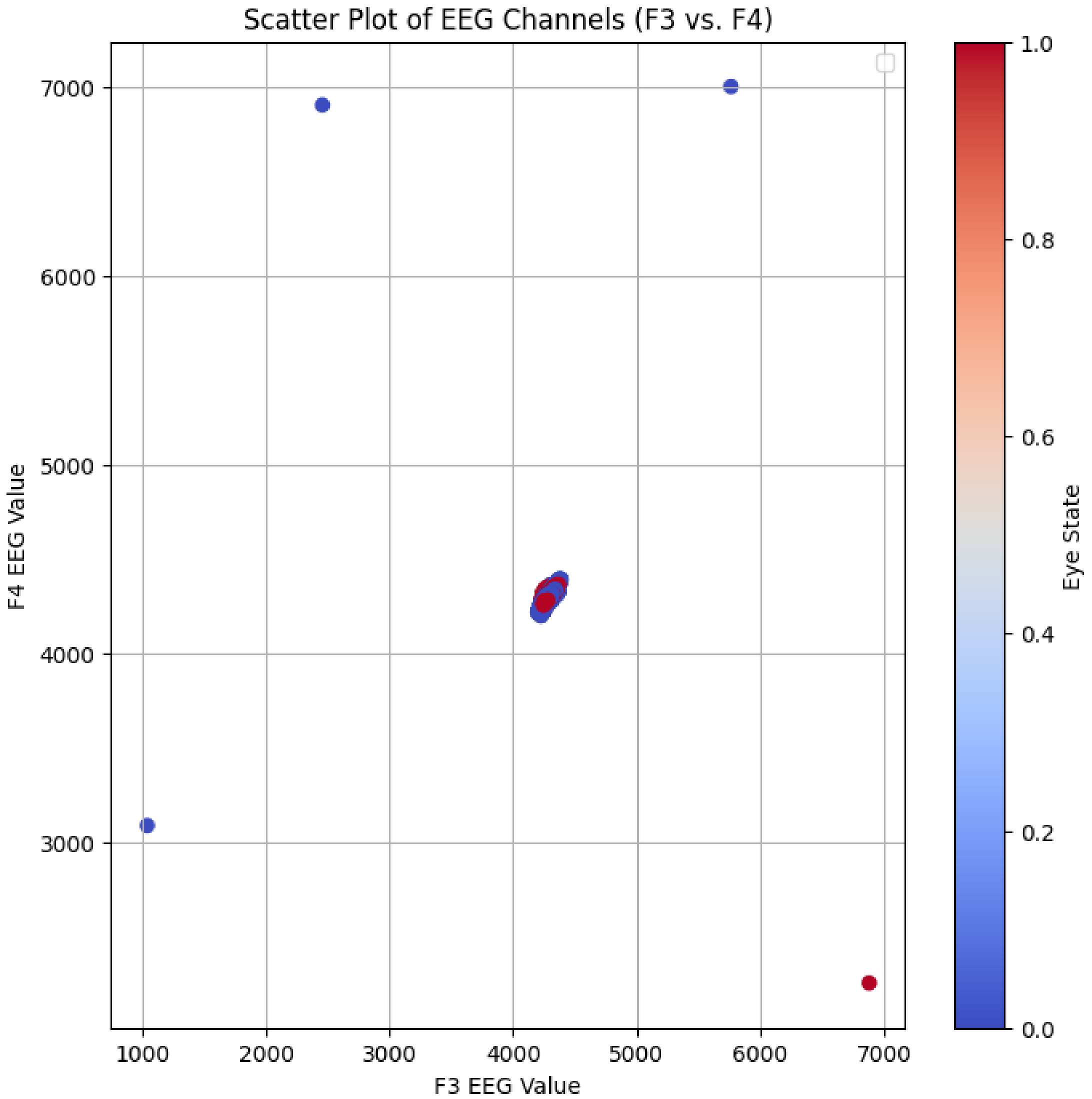
In [7]:
```python
# ─────────────────────────────────────────────
# # Plot the EEG values for all channels without threshold
# ─────────────────────────────────────────────

plt.figure(figsize=(24, 8))
for channel in data_raw_df.columns[:-1]:  # Exclude the last column ('eyeDetection')
    plt.plot(time_dir,data_raw_df[channel], label=channel)
plt.title('EEG Values for All Channels')
plt.xlabel('Time')
plt.ylabel('EEG Value')
plt.legend()
plt.grid(True)
plt.show()
```



plot scatter chart

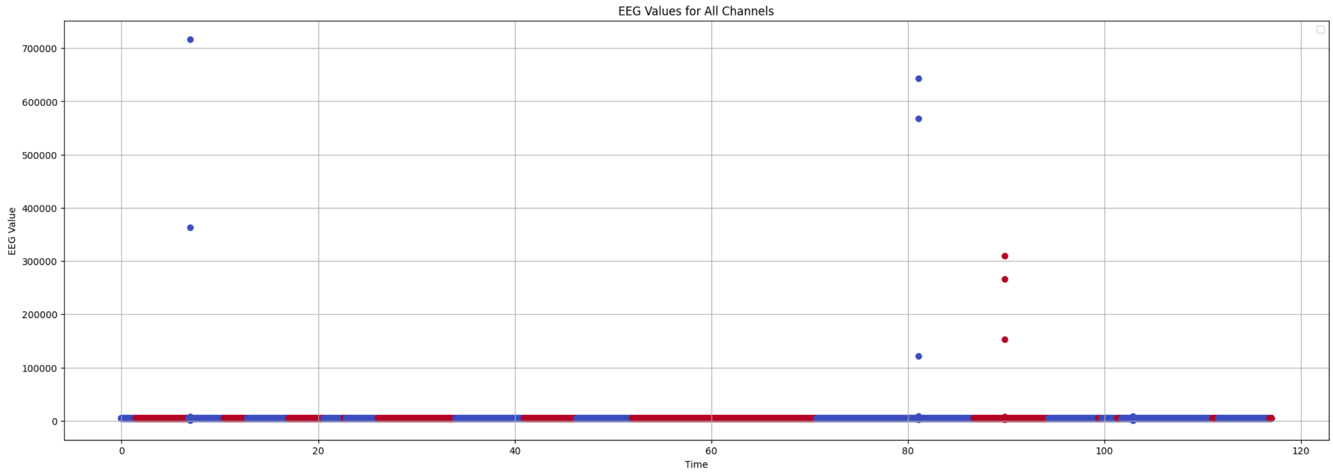
In [8]:
```
#
# plot the scatterplots for two channels
#

plt.figure(figsize=(8, 8))
plt.scatter(data_raw_df['F3'], data_raw_df['F4'], c=data_raw_df['eyeDetection'], cmap='coolwarm')
plt.title('Scatter Plot of EEG Channels (F3 vs. F4)')
plt.xlabel('F3 EEG Value')
plt.ylabel('F4 EEG Value')
plt.colorbar(label='Eye State')
plt.legend()
plt.grid(True)
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an unders
core are ignored when legend() is called with no argument.
```



In [9]:
```
#
# plot scatter plot for all channels :
#

plt.figure(figsize=(24, 8))
for channel in data_raw_df.columns[:-1]:  # Exclude the last column ('eyeDetection')
    plt.scatter(time_dir ,data_raw_df[channel], c=data_raw_df['eyeDetection'],cmap='coolwarm')
plt.title('EEG Values for All Channels')
plt.xlabel('Time')
plt.ylabel('EEG Value')
plt.legend()
plt.grid(True)
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an unders
core are ignored when legend() is called with no argument.
```



In [10]:
```
print()
```

signal filtering and plot all channels

In [11]:
```
#
# create subplot function
#

# set defaults
Vertical_space = 150
# function to plot eeg signals
def plot_subplot(EEG_signal, time_axis) :
  print(EEG_signal.shape)
  for channel in range(EEG_signal.shape[0]):

    channel_signal = EEG_signal.iloc[channel, :] - np.mean(EEG_signal.iloc[channel, :]) + channel *
    plt.plot(time_axis, channel_signal)
    channel_positions = np.arange(EEG_signal.shape[0]) * Vertical_space

    plt.yticks(channel_positions, data_cols[0:-1])
    plt.xlabel('Time(s)')
    plt.ylabel('Channel')
  plt.show()
```

In [12]:

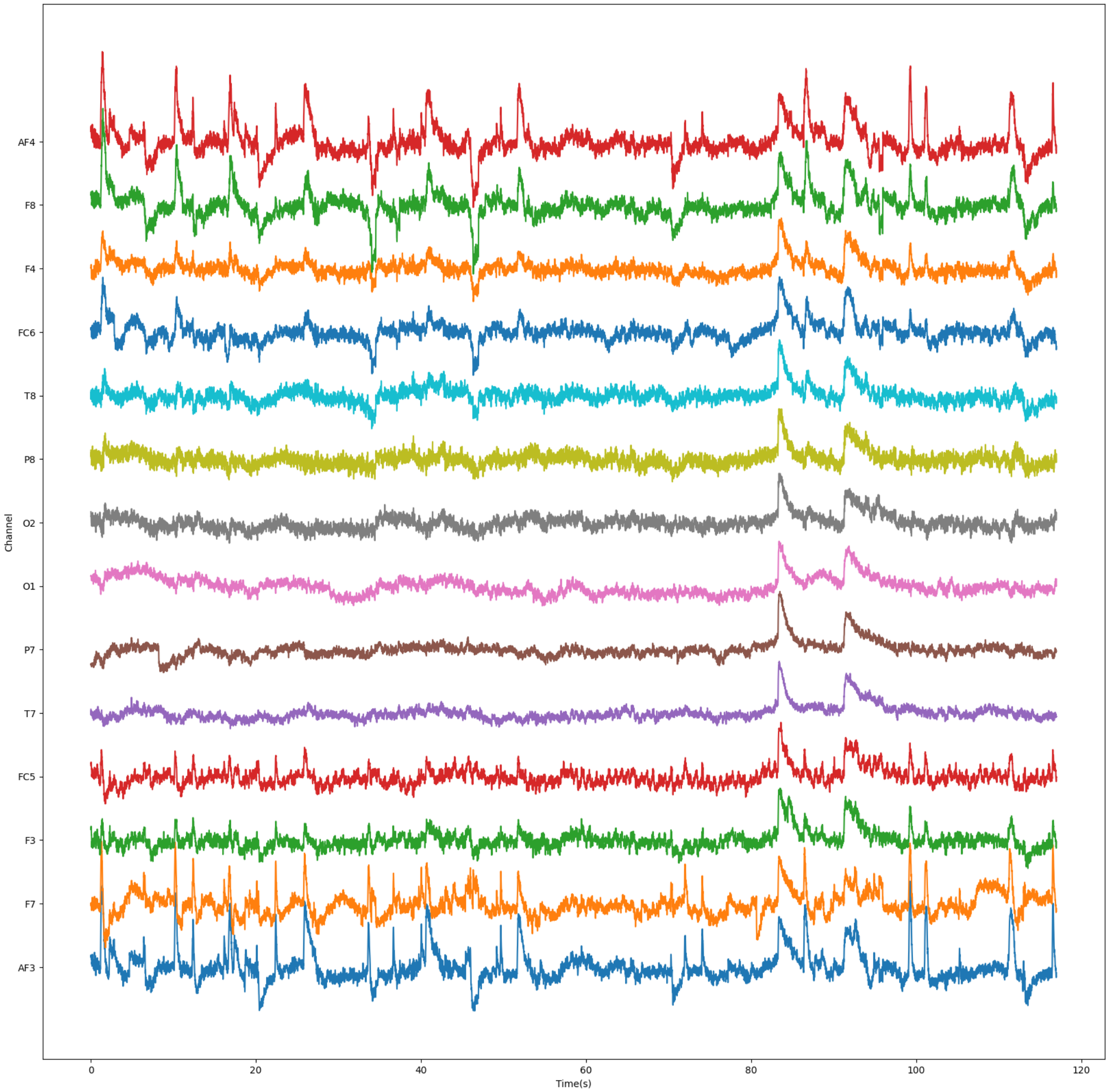
```
#
# filter data and remove outlier indexes
#


# Create a copy of the DataFrame to avoid modifying the original data
filtered_df = data_raw_df.copy()
time_dir_df = pd.DataFrame(time_dir)
print(filtered_df.shape)
print(len(time_dir_df))

# Define the threshold for outlier removal (e.g., 35 standard deviations from the mean)
up_threshold = 10

# Iterate through all columns (channels) except the 'eyeDetection' column
for channel in data_raw_df.columns[:-1]:  # Exclude the last column ('eyeDetection')
    # Calculate the z-scores for each data point in the current channel
    z_scores = np.abs(stats.zscore(filtered_df[channel]))

    # Identify data points where the z-score exceeds the threshold
    outlier_indices = np.where(z_scores > up_threshold)

    # Remove rows containing outliers for the current channel
    filtered_df = filtered_df.drop(filtered_df.index[outlier_indices])
    time_dir_df = time_dir_df.drop(filtered_df.index[outlier_indices])
```

```
(14980, 15)
14980
```

In [13]:

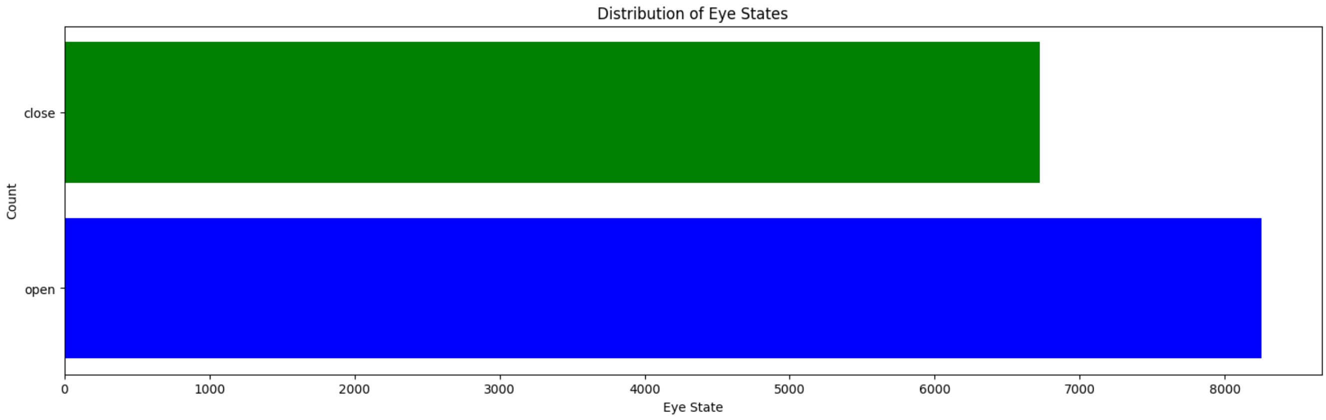
```python
#
# create subplot :
#

plt.figure(figsize=(20 , 20))
print(filtered_df.shape)
print(len(time_dir_df))
plot_subplot(filtered_df.transpose().iloc[0:14,:], time_dir_df)
```

```
(14976, 15)
14976
(14, 14976)
```



count eye state of data

In [14]:
```
#
# plot eye state count :
#

# count dataset with eyeDetection state
count = data_raw_df['eyeDetection'].value_counts()

# calculate direction of eyeDetection state in second

# plot count of eyeDetection state
plt.figure(figsize=(18, 5))
plt.barh(["open" , "close"], count, color=['blue', 'green'])
plt.title('Distribution of Eye States')
plt.xlabel('Eye State')
plt.ylabel('Count')
plt.show()
```



convert eye datatype from str to int

In [15]:
```
#
# Convert 'eyeDetection' column to integers
#

data_raw_df['eyeDetection'] = data_raw_df['eyeDetection'].apply(int)

data_raw_df
```

Out[15]:

	AF3	F7	F3	FC5	T7	P7	O1	O2	P8	T8	FC6	F4	F8	AF4	ey
0	4329.23	4009.23	4289.23	4148.21	4350.26	4586.15	4096.92	4641.03	4222.05	4238.46	4211.28	4280.51	4635.90	4393.85	
1	4324.62	4004.62	4293.85	4148.72	4342.05	4586.67	4097.44	4638.97	4210.77	4226.67	4207.69	4279.49	4632.82	4384.10	
2	4327.69	4006.67	4295.38	4156.41	4336.92	4583.59	4096.92	4630.26	4207.69	4222.05	4206.67	4282.05	4628.72	4389.23	
3	4328.72	4011.79	4296.41	4155.90	4343.59	4582.56	4097.44	4630.77	4217.44	4235.38	4210.77	4287.69	4632.31	4396.41	
4	4326.15	4011.79	4292.31	4151.28	4347.69	4586.67	4095.90	4627.69	4210.77	4244.10	4212.82	4288.21	4632.82	4398.46	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14975	4281.03	3990.26	4245.64	4116.92	4333.85	4614.36	4074.87	4625.64	4203.08	4221.54	4171.28	4269.23	4593.33	4340.51	
14976	4276.92	3991.79	4245.13	4110.77	4332.82	4615.38	4073.33	4621.54	4194.36	4217.44	4162.56	4259.49	4590.26	4333.33	
14977	4277.44	3990.77	4246.67	4113.85	4333.33	4615.38	4072.82	4623.59	4193.33	4212.82	4160.51	4257.95	4591.79	4339.49	
14978	4284.62	3991.79	4251.28	4122.05	4334.36	4616.41	4080.51	4628.72	4200.00	4220.00	4165.64	4267.18	4596.41	4350.77	
14979	4287.69	3997.44	4260.00	4121.03	4333.33	4616.41	4088.72	4638.46	4212.31	4226.67	4167.69	4274.36	4597.95	4350.77	

14980 rows × 15 columns

In [16]:
```python
#
# check if dataframe has none value :
#
data_raw_df.isnull().values.any()

# copy data for machine learning section
main_data = data_raw_df
```

split data into train - test - validation

In [17]:
```python
#
# extract train and test (method one)
#
from sklearn.model_selection import train_test_split

# Define the features (X) and the target (y)
x_data = main_data.drop(columns=['eyeDetection'])  # Features (all columns except 'eyeDetection')
y_data = main_data['eyeDetection']  # Target (labels)

# Split the data into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(x_data, y_data, test_size=0.2, random_state=42)

# Display the shapes of the training and validation sets
print("X_train shape:", X_train.shape)
print("X_val shape:", X_val.shape)
print("y_train shape:", y_train.shape)
print("y_val shape:", y_val.shape)
```

```
X_train shape: (11984, 14)
X_val shape: (2996, 14)
y_train shape: (11984,)
y_val shape: (2996,)
```

Train SVM model

In [18]:
```python
#
# create SVM model for classification
#
svm_classifier = SVC()
```

In [19]:
```python
#
# fit model and calculate accuracy with test data
#
svm_classifier.fit(X_train, y_train)
```

Out[19]:
```
SVC()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [20]:
```python
#
# calculate accuracy for train data (method 1)
#
train_predict = svm_classifier.predict(X_train)
train_accuracy = accuracy_score(y_train, train_predict)
print(train_accuracy)
```

```
0.5567423230974633
```

In [21]:
```
#
# calculate accuracy for validation data (method 1)
#
svm_predictions = svm_classifier.predict(X_val)
svm_accuracy = accuracy_score(y_val, svm_predictions)
print(svm_accuracy)
```

```
0.5293724966622163
```

In [22]:
```
#
# calculate accuracy for Train data (method 2)
#

cross_val_score(svm_classifier, X_train, y_train, cv=3 , scoring="accuracy")
```

Out[22]:
```
array([0.55669587, 0.55669587, 0.55658488])
```

In [23]:
```
#
# calculate accuracy for train data (method 3)
#

y_train_pred = cross_val_predict(svm_classifier, X_train, y_train, cv=3)
y_train_pred = np.array(y_train_pred)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[23]:
```
array([[6671,    0],
       [5313,    0]])
```

Train "Logistic Regression" Model

In [24]:
```
#
# create Logistic Regression model for classification
#
logreg_classifier = LogisticRegression()
```

In [25]:
```
#
# fit model and calculate accuracy with test data
#
logreg_classifier.fit(X_train, y_train)
```

```
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[25]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [26]:
```
#
# calculate accuracy for train data (method 1)
#
logreg_predictions = logreg_classifier.predict(X_train)
logreg_accuracy = accuracy_score(y_train, logreg_predictions)
print("Logistic Regression Accuracy for train data :", logreg_accuracy)
```

```
Logistic Regression Accuracy for train data : 0.6453604806408545
```

In [27]:
```python
#
# calculate accuracy for validation data (method 1)
#
logreg_predictions = logreg_classifier.predict(X_val)
logreg_accuracy = accuracy_score(y_val, logreg_predictions)
print("Logistic Regression Accuracy for test data :", logreg_accuracy)
```

```
Logistic Regression Accuracy for test data : 0.6255006675567423
```

In [28]:
```python
#
# calculate accuracy for Train data (method 2)
#
cross_val_score(logreg_classifier, X_train, y_train, cv=3 , scoring="accuracy")
```

```
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[28]:
```
array([0.64831039, 0.64330413, 0.64496745])
```

In [29]:
```python
#
# calculate accuracy for train data (method 3)
#
y_train_pred = cross_val_predict(logreg_classifier, X_train, y_train, cv=3)
y_train_pred = np.array(y_train_pred)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

11984
11984

/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[29]:
```
array([[5286, 1385],
       [2863, 2450]])
```

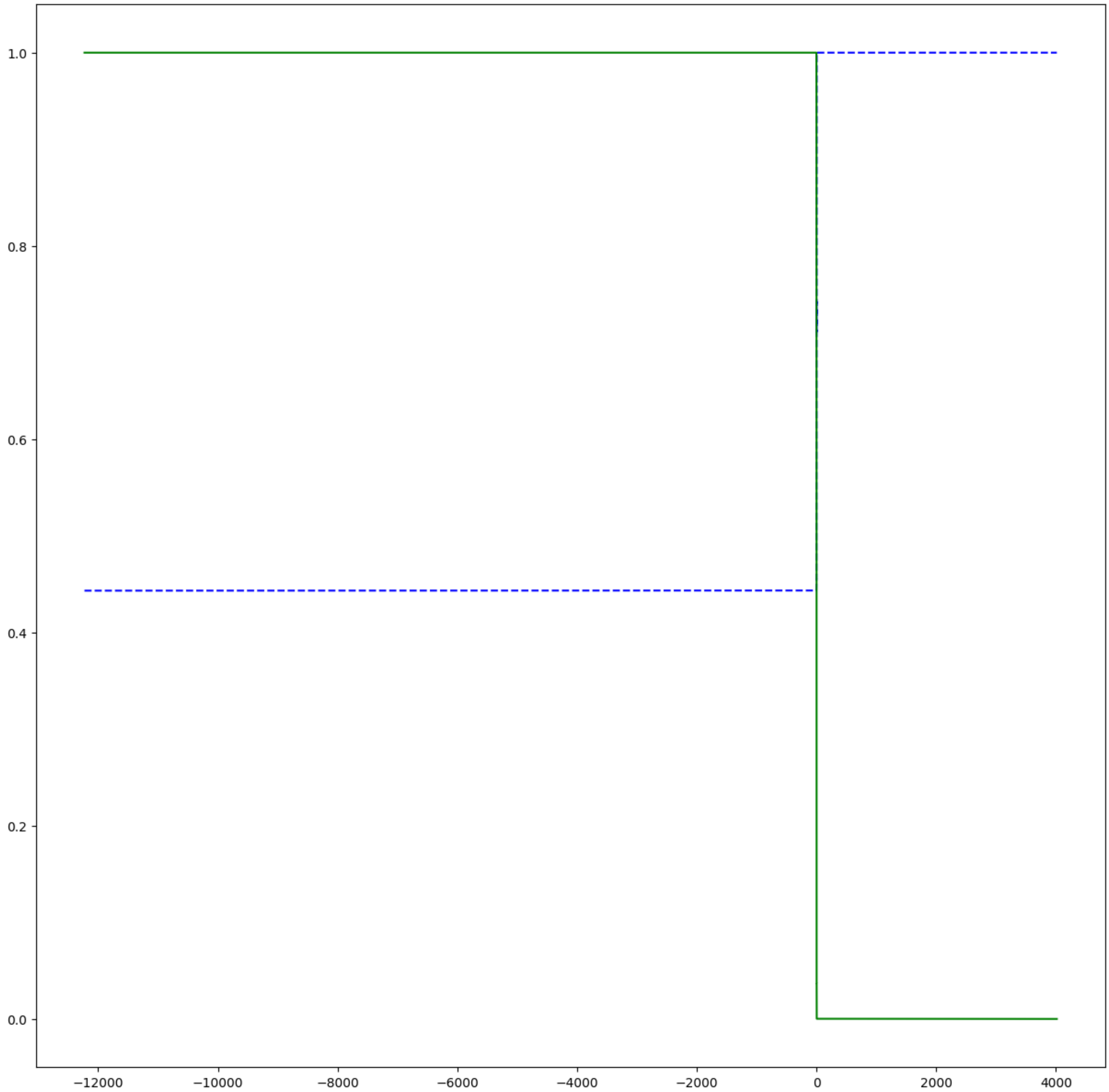
In [30]:
```python
#
# calculate precision and recall and f1_score
#

print(precision_score(y_train, y_train_pred))
print(recall_score(y_train, y_train_pred))
print(f1_score(y_train, y_train_pred))
```

```
0.6388526727509778
0.461133069828722
0.5356362046348929
```

In [31]:

```python
#
# calculate score for each prediction :
#
y_scores = cross_val_predict(logreg_classifier, X_train, y_train, cv=3, method="decision_function")
precisions , recalls , thresholds = precision_recall_curve(y_train, y_scores)
print(precisions)
print(recalls)
print(thresholds)

#
# plot precision and recall chart :
#


def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

plt.figure(figsize=(15, 15))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.show()
```

```
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/linear_model/_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/mo
dules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-
learn.org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

[0.44334112 0.44337812 0.44341512 ... 1.         1.         1.        ]
[1.00000000e+00 1.00000000e+00 1.00000000e+00 ... 3.76435159e-04
 1.88217580e-04 0.00000000e+00]
[-1.22283890e+04 -1.76228507e+03 -3.47859432e+00 ...  3.02571418e+00
  3.09018569e+00  4.01367120e+03]
```



Train "Random Forest" Model

In [32]:
```
# ============================================================================
# create random forest model :
# ============================================================================
rf_classifier = RandomForestClassifier()
```

In [33]:
```
# ============================================================================
# fit random forest model on the data :
# ============================================================================
rf_classifier.fit(X_train, y_train)
```

Out[33]:
```
RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```python
#
# calculate accuracy for train data (method 1)
#
rf_predictions = rf_classifier.predict(X_train)
rf_accuracy = accuracy_score(y_train, rf_predictions)
print("Logistic Regression Accuracy for train data :", rf_accuracy)
```

```
Logistic Regression Accuracy for train data : 1.0
```

In [35]:
```python
#
# calculate accuracy for validation data (method 1)
#
rf_predictions = rf_classifier.predict(X_val)
rf_accuracy = accuracy_score(y_val, rf_predictions)
print("Random Forest Accuracy for test data :", rf_accuracy)
```

```
Random Forest Accuracy for test data : 0.92456608811749
```

In [36]:
```python
#
# calculate accuracy for Train data (method 2)
#
cross_val_score(rf_classifier, X_train, y_train, cv=3 , scoring="accuracy")
```

Out[36]:
```
array([0.9146433 , 0.91389237, 0.91437156])
```

In [37]:
```python
#
# calculate accuracy for train data (method 3)
#
y_train_pred = cross_val_predict(rf_classifier, X_train, y_train, cv=3)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[37]:
```
array([[6331,  340],
       [ 669, 4644]])
```

In [38]:
```python
#
# calculate precision and recall and f1_score
#

print("precision_score : ",precision_score(y_train, y_train_pred).round(2))
print("recall_score : \t",recall_score(y_train, y_train_pred).round(2))
print("f1_score : \t",f1_score(y_train, y_train_pred).round(2))
print("AUC score : \t",roc_auc_score(y_train, y_train_pred).round(2))
```

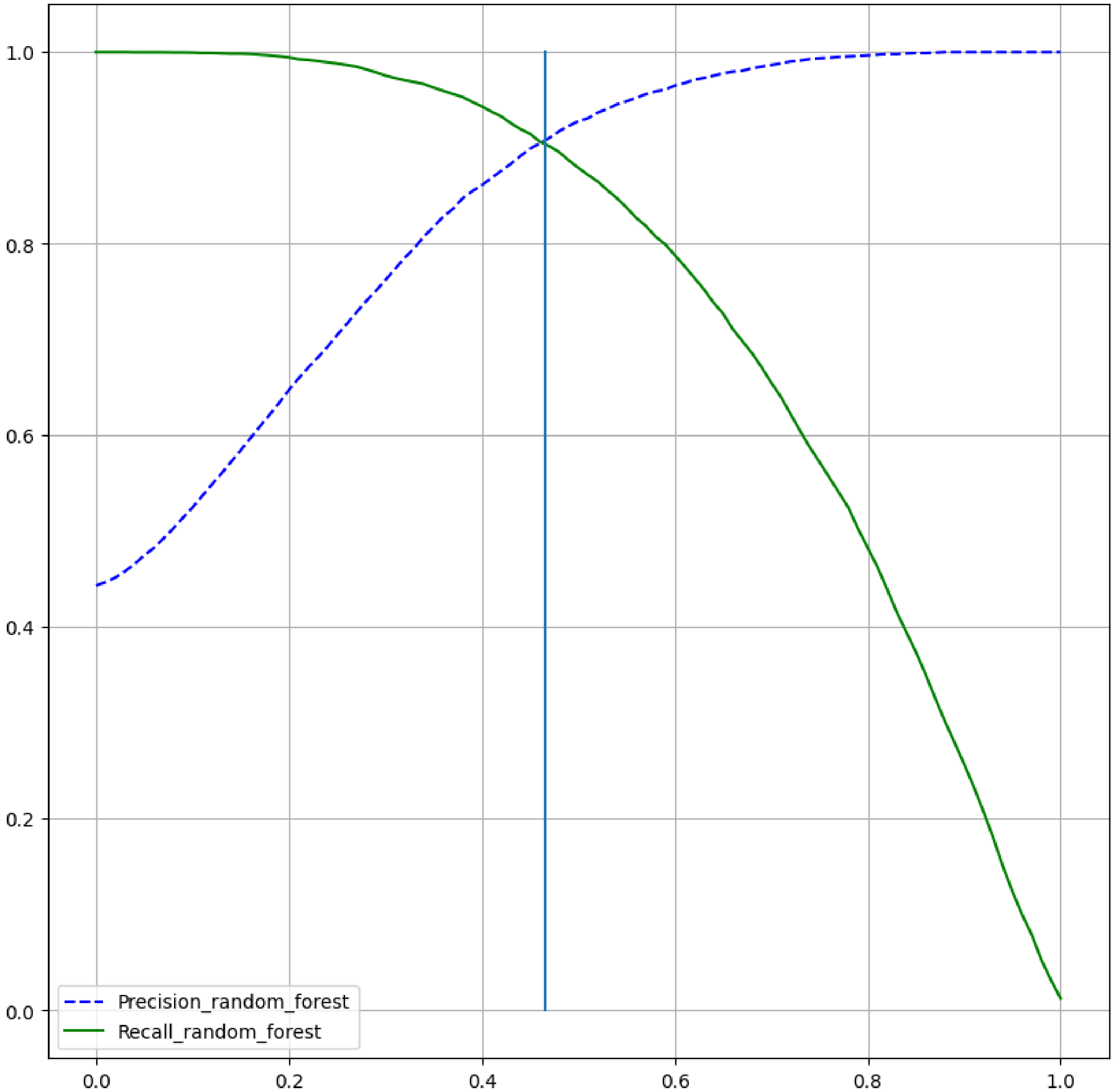
```
precision_score :  0.93
recall_score :   0.87
f1_score :       0.9
AUC score :      0.91
```

In [39]:
```python
#
# calculate score for each prediction :
#
y_probs = cross_val_predict(rf_classifier, X_train, y_train, cv=3, method="predict_proba")
y_scores = y_probs[:, 1]
precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)
fpr , tpr , thresholds_ROC = roc_curve(y_train, y_scores)
print(y_scores)
# print(precisions)
# print(recalls)
# print(thresholds)
```

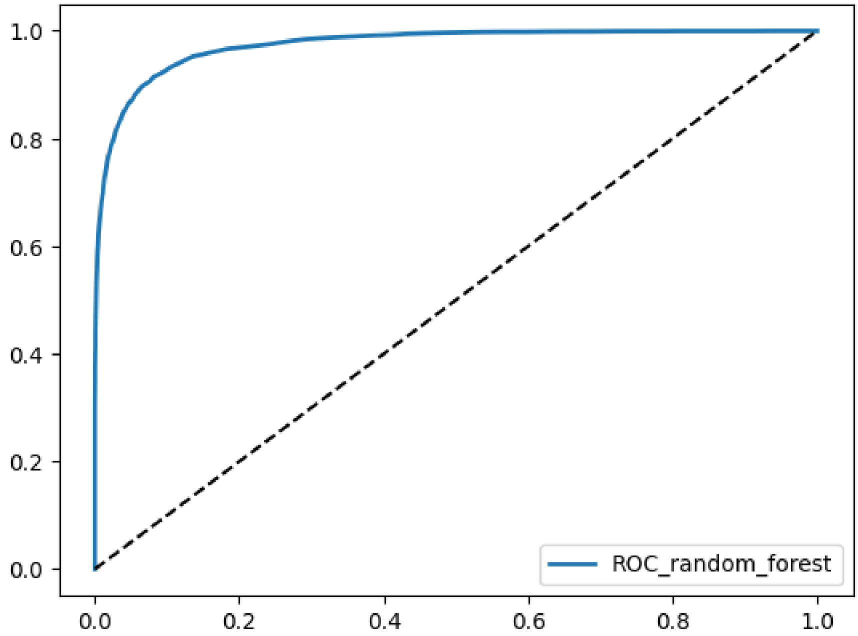
```
[0.53 0.31 0.04 ... 0.89 0.39 0.36]
```

In [40]:

```
# 
# plot precision and recall chart :
# 

def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision_random_forest")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall_random_forest")

plt.figure(figsize=(10, 10))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.grid(True)
plt.legend()
plt.plot([0.465,0.465],[0,1])
plt.show()
```



In [41]:
```
def plot_roc_curve(fpr , tpr , label=None) :
    plt.plot(fpr, tpr, linewidth = 2, label="ROC_random_forest")
    plt.plot([0, 1],[0, 1],'k--')

plot_roc_curve(fpr, tpr)
plt.legend()
plt.show()
```



Train "KNN" Model

In [42]:
```
#
# k-Nearest Neighbors (KNN) model
#
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can specify the number of neighbors
```

In [43]:
```
#
# fit random KNN model on the data :
#
knn_classifier.fit(X_train, y_train)
```

Out[43]:
```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [44]:
```
#
# calculate accuracy for train data (method 1)
#
knn_predictions = knn_classifier.predict(X_train)
knn_accuracy = accuracy_score(y_train, knn_predictions)
print("knn_classifier Accuracy for train data :", knn_accuracy)
```

```
knn_classifier Accuracy for train data : 0.9839786381842457
```

In [45]:
```
#
# calculate accuracy for validation data (method 1)
#
knn_predictions = knn_classifier.predict(X_val)
knn_accuracy = accuracy_score(y_val, knn_predictions)
print("knn_classifier accuracy for test data:", knn_accuracy)
```

```
knn_classifier accuracy for test data: 0.9689586114819759
```

In [46]:

```python
#
# calculate accuracy for Train data (method 2)
#
knn_classifier = KNeighborsClassifier(n_neighbors=5)  # You can specify the number of neighbors
cross_val_score(knn_classifier, X_train, y_train, cv=3 , scoring="accuracy")
```

Out[46]:

```
array([0.95294118, 0.94793492, 0.95468202])
```

In [47]:

```python
#
# calculate accuracy for train data (method 3)
#
y_train_pred = cross_val_predict(knn_classifier, X_train, y_train, cv=3)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[47]:

```
array([[6422,  249],
       [ 328, 4985]])
```

In [48]:

```python
#
# calculate precision and recall and f1_score
#
print("precision_score : ",precision_score(y_train, y_train_pred).round(2))
print("recall_score : \t",recall_score(y_train, y_train_pred).round(2))
print("f1_score : \t",f1_score(y_train, y_train_pred).round(2))
print("AUC score : \t",roc_auc_score(y_train, y_train_pred).round(2))
```

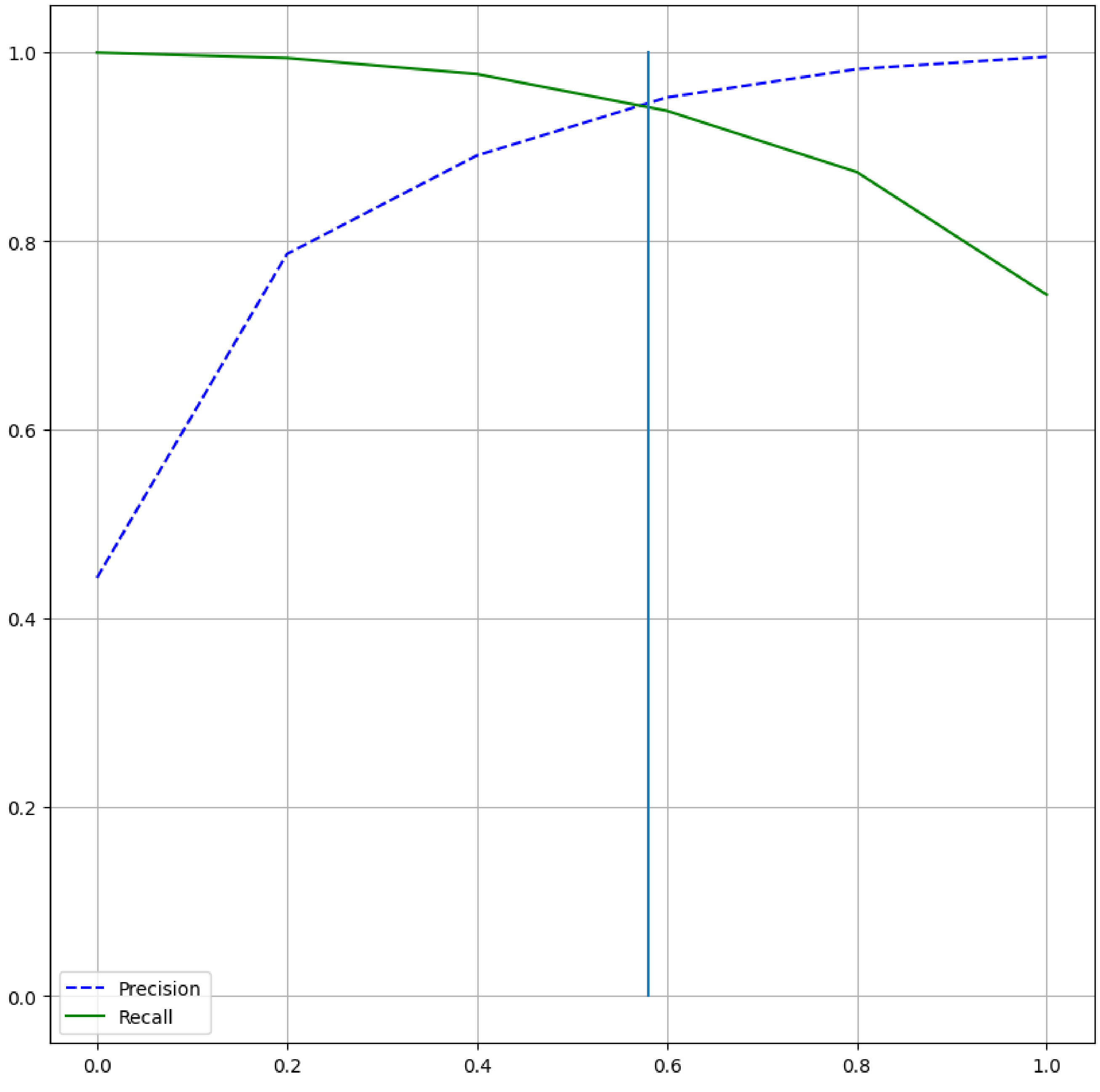
```
precision_score :  0.95
recall_score :   0.94
f1_score :       0.95
AUC score :      0.95
```

In [49]:

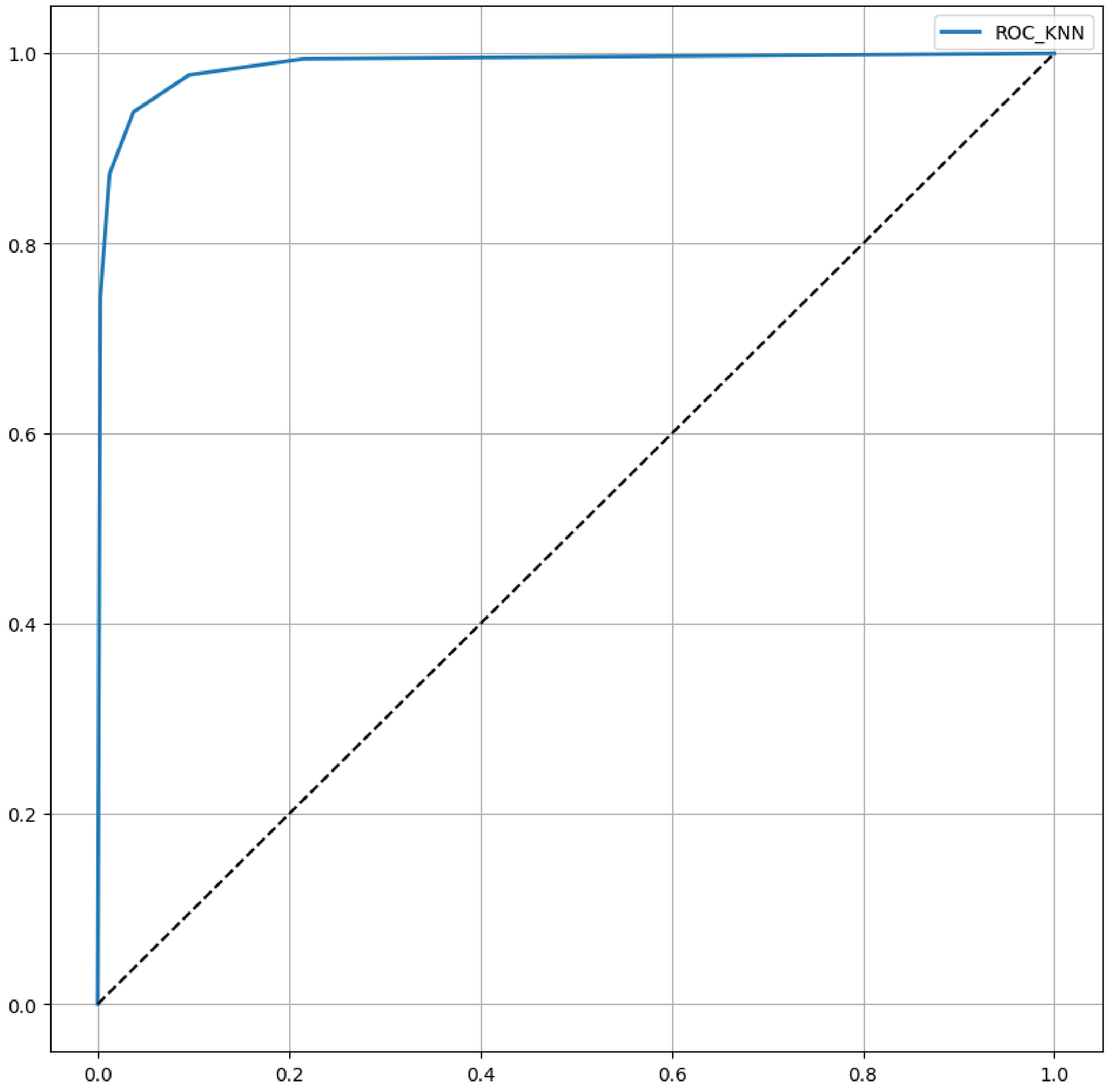
```python
#
# calculate score for each prediction :
#
y_probs = cross_val_predict(knn_classifier, X_train, y_train, cv=3, method="predict_proba")
y_scores = y_probs[:, 1]

# calculate precisions and recalls
precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)

# calculate FPR and TPR for KNN model
fpr , tpr , thresholds_ROC = roc_curve(y_train, y_scores)

print(y_scores)
# print(precisions)
# print(recalls)
# print(thresholds)
```

```
[0.4 0.  0.  ... 1.  0.4 0.8]
```

```
In [50]: # █████████████████████████████████████████████████████████████████████████████████████████████████
         # plot precision and recall chart :
         # █████████████████████████████████████████████████████████████████████████████████████████████████

         def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
             plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
             plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

         plt.figure(figsize=(10, 10))
         plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
         plt.grid(True)
         plt.legend()
         plt.plot([0.58,0.58],[0,1])
         plt.show()
```



In [51]:
```
def plot_roc_curve(fpr , tpr , label=None) :
    plt.plot(fpr, tpr, linewidth = 2, label="ROC_KNN")
    plt.plot([0, 1],[0, 1],'k--')

plt.figure(figsize=(10, 10))
plot_roc_curve(fpr, tpr)
plt.legend()
plt.grid(True)
plt.show()
```



Train "Naive Bayes classifier" Model

In [52]:
```
#
# Naive Bayes classifier model
#
NB_clf = BernoulliNB()
```

In [53]:
```python
# [redacted]
# fit Naive Bayes classifier model on the data :
# [redacted]
NB_clf.fit(X_train, y_train)
y_bernolli_pred = cross_val_predict(NB_clf , X_train , y_train , cv=3)
y_bernolli_pred_proba = cross_val_predict(NB_clf , X_train , y_train , cv=3, method="predict_proba")
print(y_bernolli_pred[:10])
print(y_bernolli_pred_proba[:10])
```

```
[0 0 0 0 0 0 0 0 0 0]
[[0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]
 [0.55683873 0.44316127]]
```

In [54]:
```python
# [redacted]
# calculate accuracy for train data (method 1)
# [redacted]
NB_clf_predictions = NB_clf.predict(X_train)
NB_clf_accuracy = accuracy_score(y_train, NB_clf_predictions)
print("Naive Bayes classifier for train data :", NB_clf_accuracy)
```

```
Naive Bayes classifier for train data : 0.5566588785046729
```

In [55]:
```python
# [redacted]
# calculate accuracy for validation data (method 1)
# [redacted]
NB_clf_predictions = NB_clf.predict(X_val)
NB_clf_accuracy = accuracy_score(y_val, NB_clf_predictions)
print("Naive Bayes classifier Accuracy fro test data:", NB_clf_accuracy)
```

```
Naive Bayes classifier Accuracy fro test data: 0.5293724966622163
```

In [56]:
```python
# [redacted]
# calculate accuracy for Train data (method 2)
# [redacted]
cross_val_score(NB_clf, X_train, y_train, cv=3 , scoring="accuracy")
```

Out[56]:
```
array([0.55669587, 0.55669587, 0.55658488])
```

In [57]:
```python
# [redacted]
# calculate accuracy for train data (method 3)
# [redacted]
y_train_pred = cross_val_predict(NB_clf, X_train, y_train, cv=3)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[57]:
```
array([[6671,    0],
       [5313,    0]])
```

In [58]:
```python
# [redacted]
# calculate precision and recall and f1_score
# [redacted]
print("precision_score : ",precision_score(y_train, y_train_pred).round(2))
print("recall_score : \t",recall_score(y_train, y_train_pred).round(2))
print("f1_score : \t",f1_score(y_train, y_train_pred).round(2))
print("AUC score : \t",roc_auc_score(y_train, y_train_pred).round(2))
```

```
precision_score :  0.0
recall_score :   0.0
f1_score :       0.0
AUC score :      0.5

/home/alireza/anaconda3/envs/brainWave/lib/python3.12/site-packages/sklearn/metrics/_classificatio
n.py:1471: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

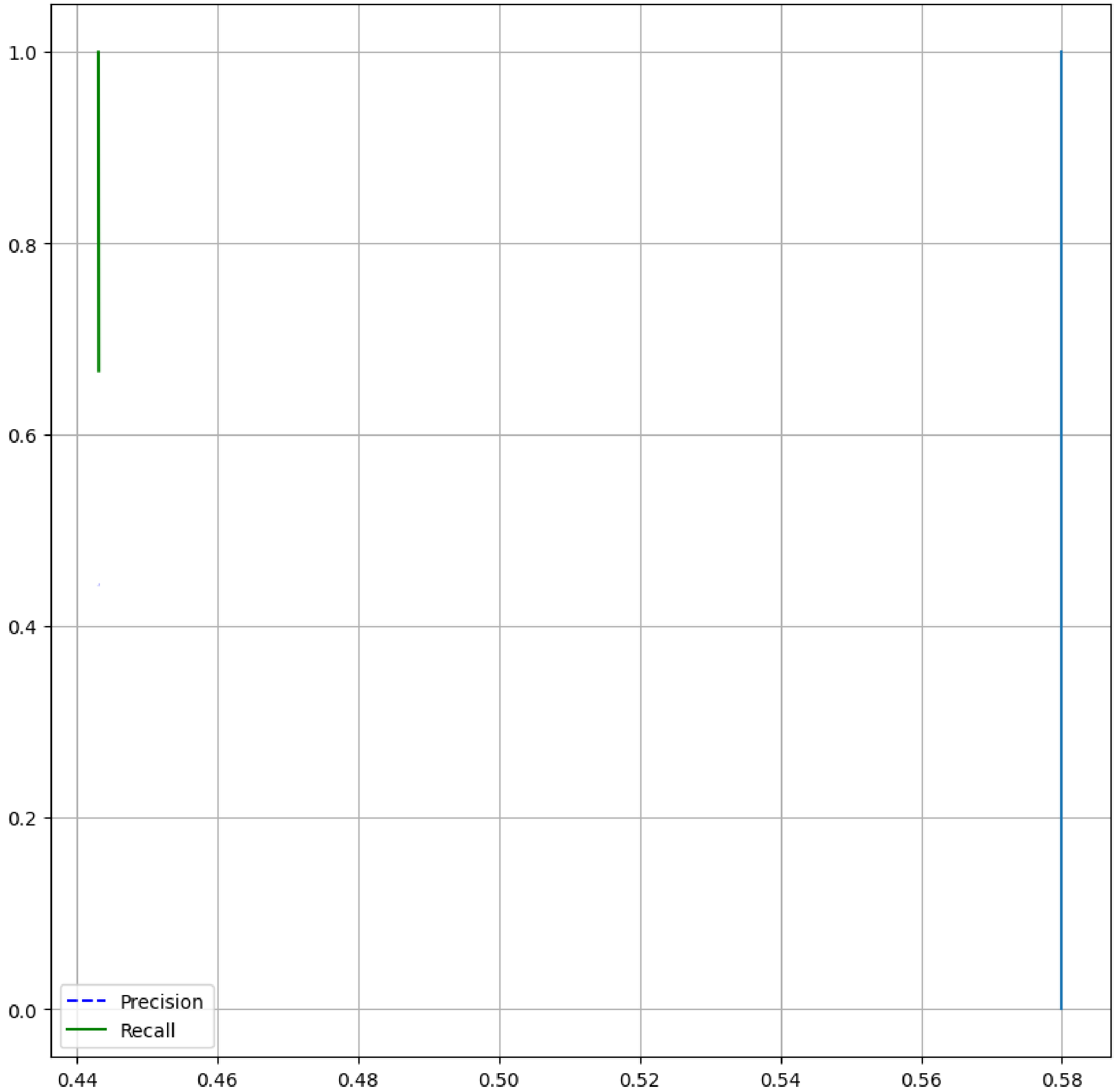
In [59]:

```
# [redacted]
# calculate score for each prediction :
# [redacted]
y_probs = cross_val_predict(NB_clf, X_train, y_train, cv=3, method="predict_proba")
y_scores = y_probs[:, 1]

# calculate precisions and recalls
precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)

# calculate FPR and TPR for KNN model
fpr , tpr , thresholds_ROC = roc_curve(y_train, y_scores)

print(y_scores)
# print(precisions)
# print(recalls)
# print(thresholds)
```

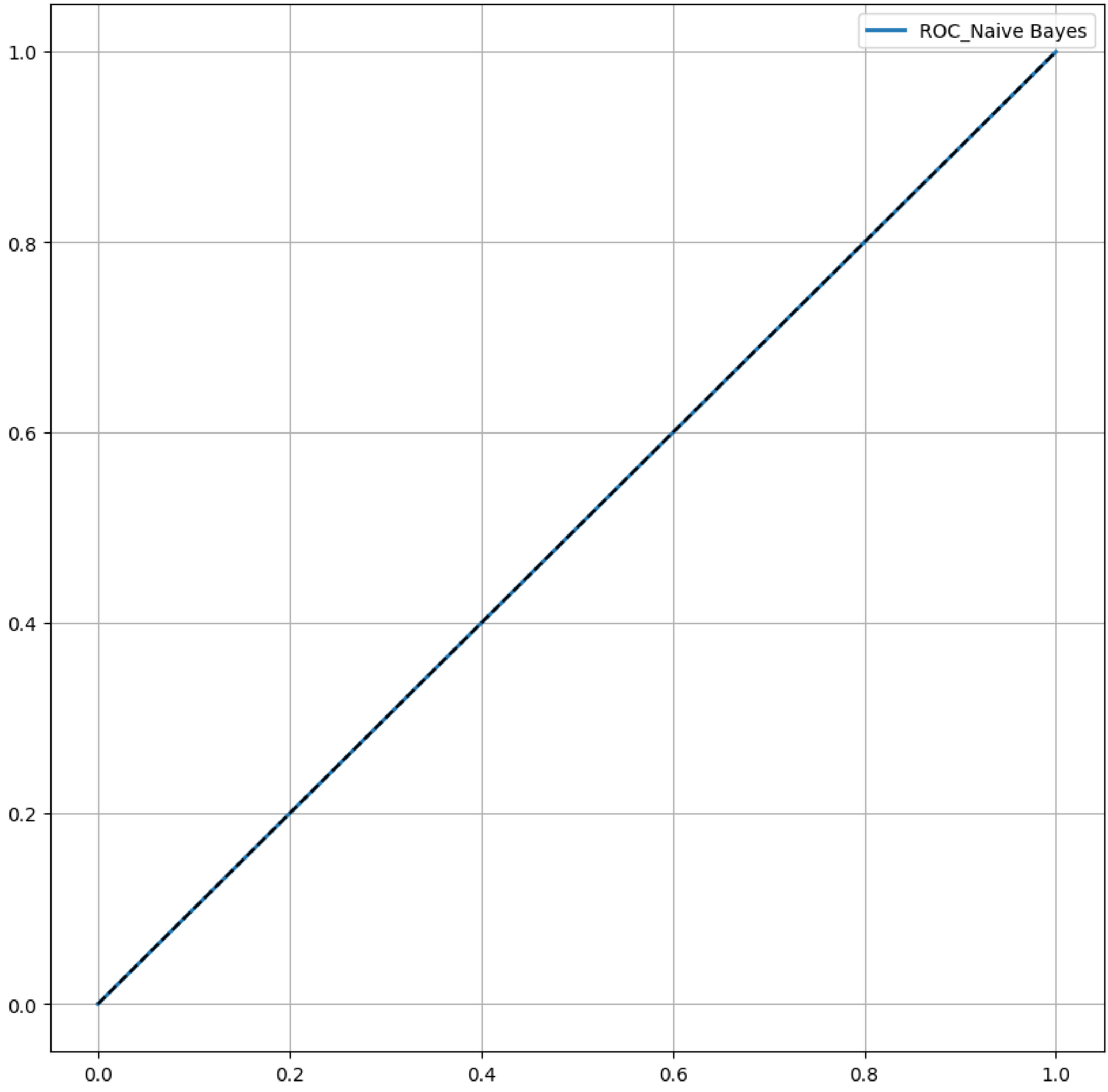
```
[0.44316127 0.44316127 0.44316127 ... 0.44310561 0.44310561 0.44310561]
```

In [60]:

```
#
# plot precision and recall chart :
#
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")

plt.figure(figsize=(10, 10))
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
plt.grid(True)
plt.legend()
plt.plot([0.58,0.58],[0,1])
plt.show()
```



In [61]:
```python
def plot_roc_curve(fpr , tpr , label=None) :
    plt.plot(fpr, tpr, linewidth = 2, label="ROC_Naive Bayes")
    plt.plot([0, 1],[0, 1],'k--')

plt.figure(figsize=(10, 10))
plot_roc_curve(fpr, tpr)
plt.legend()
plt.grid(True)
plt.show()
```



Grid search for Decision Tree Classifier

In [62]:
```
#
# grid search for Decision_Tree_Classifier mode
#

params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
grid_search_clf = GridSearchCV(DecisionTreeClassifier(random_state=42), params, n_jobs=-1, verbose=1

grid_search_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 294 candidates, totalling 882 fits
```

Out[62]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
                                            13, 14, 15, 16, 17, 18, 19, 20, 21,
                                            22, 23, 24, 25, 26, 27, 28, 29, 30,
                                            31, ...],
                         'min_samples_split': [2, 3, 4]},
             verbose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [63]:
```
grid_search_clf.best_estimator_
```

Out[63]:
```
DecisionTreeClassifier(max_leaf_nodes=98, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [64]:
```
#
# calculate accuracy for Train data (method 2)
#
cross_val_score(grid_search_clf.best_estimator_, X_train, y_train, cv=3 , scoring="accuracy")
```

Out[64]:
```
array([0.78548185, 0.81151439, 0.77616425])
```

In [65]:
```
#
# calculate accuracy for train data (method 3)
#
y_train_pred = cross_val_predict(grid_search_clf.best_estimator_, X_train, y_train, cv=3)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[65]:
```
array([[5633, 1038],
       [1466, 3847]])
```

In [66]:
```
#
# grid search for random forest classifier
#
param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_clf = RandomForestClassifier(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search_RF_clf = GridSearchCV(forest_clf, param_grid, cv=3,
                           scoring='neg_mean_squared_error', return_train_score=True)
grid_search_RF_clf.fit(X_train, y_train)
```

Out[66]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [67]:
```
grid_search_RF_clf.best_params_
grid_search_RF_clf.best_estimator_
```

Out[67]:
```
RandomForestClassifier(max_features=8, n_estimators=30, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [68]:
```
#
# calculate accuracy for Train data (method 2)
#
print(grid_search_RF_clf.best_estimator_)
print(grid_search_RF_clf.best_params_)
cross_val_score(grid_search_RF_clf.best_estimator_, X_train, y_train, cv=3 , scoring="accuracy")
```

```
RandomForestClassifier(max_features=8, n_estimators=30, random_state=42)
{'max_features': 8, 'n_estimators': 30}
```

Out[68]:
```
array([0.90513141, 0.91339174, 0.91011517])
```

In [69]:
```
#
# calculate accuracy for train data (method 3)
#
y_train_pred = cross_val_predict(grid_search_RF_clf.best_estimator_, X_train, y_train, cv=3)
print(len(y_train_pred))
print(len(y_train))

# print confusion matrix
confusion_matrix(y_train, y_train_pred)
```

```
11984
11984
```

Out[69]:
```
array([[6311,  360],
       [ 724, 4589]])
```