```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import scipy
from sklearn.preprocessing import StandardScaler
from scipy import signal
from scipy import fft
from scipy.fft import fft, ifft, fftfreq
import matplotlib
import matplotlib. pyplot as plt
from sklearn.preprocessing import RobustScaler
```

```
df = pd.read_csv('/content/signal.csv') # change file name acc. to your file as "('/content/"Your_file_name.csv")"
df
```

```
        2026-01-09 13:36:48.182318  595
0       2026-01-09 13:36:48.189856  545
1       2026-01-09 13:36:48.191889  499
2       2026-01-09 13:36:48.191889  466
3       2026-01-09 13:36:48.191889  414
4       2026-01-09 13:36:48.197570  427
...                            ...  ...
614434  2026-01-09 14:01:23.132119  516
614435  2026-01-09 14:01:23.134130  513
614436  2026-01-09 14:01:23.136130  512
614437  2026-01-09 14:01:23.138115  509
614438  2026-01-09 14:01:23.139621  511
614439 rows × 2 columns
```

**Change the Labeling according to your data here **

```
df.loc[:307220,'label'] = 0
df.loc[307220:,'label'] = 1
```

```
df.isnull().sum()
```

```
                            0
2026-01-09 13:36:48.182318  0
                       595  0
                     label  0

dtype: int64
```

```
df.drop(df.index[150000:310001], inplace=True)
df.reset_index(drop=True, inplace=True)
df.drop(df.index[600000:750000], inplace=True)
#df = df.loc[600000:]
df.reset_index(drop=True, inplace=True)
```

```
df.drop(columns=df.columns[0], axis=1, inplace=True)
df
```

```
        595  label
0       545    0.0
1       499    0.0
2       466    0.0
3       414    0.0
4       427    0.0
...     ...    ...
454433  516    1.0
454434  513    1.0
454435  512    1.0
454436  509    1.0
454437  511    1.0

454438 rows × 2 columns
```

```
df.columns = ['raw_eeg', 'label']
df
```

```
        raw_eeg  label
0           545    0.0
1           499    0.0
2           466    0.0
3           414    0.0
4           427    0.0
...         ...    ...
454433      516    1.0
454434      513    1.0
454435      512    1.0
454436      509    1.0
454437      511    1.0

454438 rows × 2 columns
```

```
data = df['raw_eeg']
labels_old = df['label']


sampling_rate = 512

notch_freq = 50.0  # for the notch filter
lowcut, highcut = 0.5, 30.0  # for the bandpass filter

#  notch filter
nyquist = (0.5 * sampling_rate)
notch_freq_normalized = notch_freq / nyquist
b_notch, a_notch = signal.iirnotch(notch_freq_normalized, Q=0.05, fs=sampling_rate)

#  bandpass filter
lowcut_normalized = lowcut / nyquist
highcut_normalized = highcut / nyquist
b_bandpass, a_bandpass = signal.butter(4, [lowcut_normalized, highcut_normalized], btype='band')

features = []
labels = []
additional_features_list = []

def calculate_psd_features(segment, sampling_rate):
    f, psd_values = scipy.signal.welch(segment, fs=sampling_rate, nperseg=len(segment))

    alpha_indices = np.where((f >= 8) & (f <= 13))
    beta_indices = np.where((f >= 14) & (f <= 30))
    theta_indices = np.where((f >= 4) & (f <= 7))
    delta_indices = np.where((f >= 0.5) & (f <= 3))

    energy_alpha = np.sum(psd_values[alpha_indices])
    energy_beta = np.sum(psd_values[beta_indices])
```

```
        energy_theta = np.sum(psd_values[theta_indices])
        energy_delta = np.sum(psd_values[delta_indices])

        # Calculate the alpha-beta ratio feature
        alpha_beta_ratio = energy_alpha / energy_beta

        return {
            'E_alpha': energy_alpha,
            'E_beta': energy_beta,
            'E_theta': energy_theta,
            'E_delta': energy_delta,
            'alpha_beta_ratio': alpha_beta_ratio
        }

    def calculate_additional_features(segment, sampling_rate):
        f, psd = scipy.signal.welch(segment, fs=sampling_rate, nperseg=len(segment))

        # Peak frequency
        peak_frequency = f[np.argmax(psd)]

        # Spectral centroid
        spectral_centroid = np.sum(f * psd) / np.sum(psd)

        # Spectral slope
        log_f = np.log(f[1:])
        log_psd = np.log(psd[1:])
        spectral_slope = np.polyfit(log_f, log_psd, 1)[0]

        return {
            'peak_frequency': peak_frequency,
            'spectral_centroid': spectral_centroid,
            'spectral_slope': spectral_slope
        }


    for i in range(0, len(data) - 512, 256):
        segment = data.loc[i:i+512]
        segment = pd.to_numeric(segment, errors='coerce')

        #  notch filter
        segment = signal.filtfilt(b_notch, a_notch, segment)

        #  bandpass filter
        segment = signal.filtfilt(b_bandpass, a_bandpass, segment)


        segment_features = calculate_psd_features(segment,512)
        additional_features = calculate_additional_features(segment, 512)

        segment_features = {**segment_features, **additional_features}

        features.append(segment_features)
        labels.append(labels_old[i])

    X = np.array(features)
    y = np.array(labels)
```

```
#features
segment_features
```

```
{'E_alpha': np.float64(0.07559928209871208),
 'E_beta': np.float64(0.3510668825801971),
 'E_theta': np.float64(0.1649183896304594),
 'E_delta': np.float64(1.805717319303421),
 'alpha_beta_ratio': np.float64(0.21534153704014597),
 'peak_frequency': np.float64(0.9980506822612085),
 'spectral_centroid': np.float64(3.356698318698777),
 'spectral_slope': np.float64(-9.834275214581078)}
```

```
columns = ['E_alpha', 'E_beta', 'E_theta', 'E_delta', 'alpha_beta_ratio','peak_frequency','spectral_centroid','spectral_sl

# Create a DataFrame
df = pd.DataFrame(features, columns=columns)

df['label'] = y
```

```
df.describe()
```

	E_alpha	E_beta	E_theta	E_delta	alpha_beta_ratio	peak_frequency	spectral_centroid	spectral_slope
count	1774.000000	1774.000000	1774.000000	1774.000000	1774.000000	1774.000000	1774.000000	1774.000000
mean	0.379481	0.460112	0.344179	65.075945	0.792186	1.398059	4.146025	-9.481037
std	1.489735	0.540006	1.569842	134.166951	0.619171	1.932904	3.242252	0.742958
min	0.015325	0.073787	0.005175	0.028158	0.038045	0.000000	0.784520	-12.081999
25%	0.122482	0.241039	0.082738	0.778873	0.374814	0.998051	1.474573	-9.970749
50%	0.218501	0.354432	0.154789	3.118584	0.640064	0.998051	2.978806	-9.532921
75%	0.407769	0.541223	0.286528	34.841949	1.011352	0.998051	6.342743	-9.042483
max	60.088018	15.300105	57.926182	586.718193	6.424725	22.955166	17.350520	-7.941325

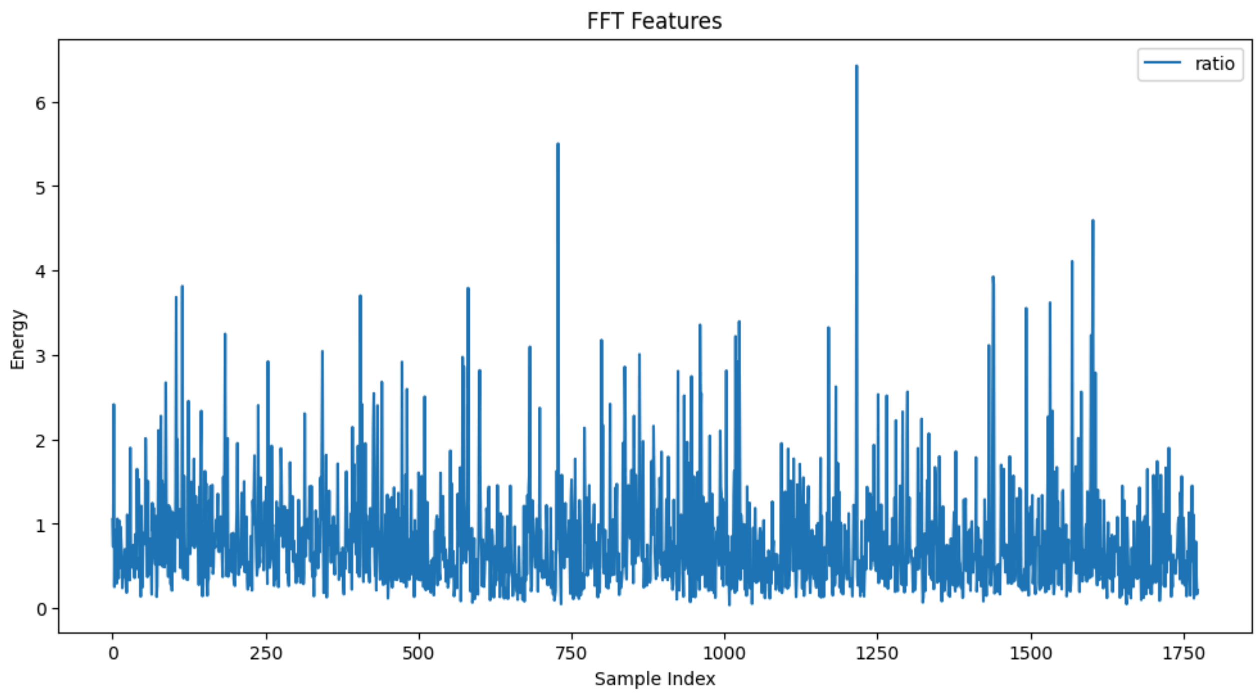
```
#df.to_csv('ready.csv')
```

```
scaler = StandardScaler()
#scaler = RobustScaler()
X_scaled = scaler.fit_transform(df.drop('label', axis=1))
df_scaled = pd.DataFrame(X_scaled, columns=columns)

# Add labels to the DataFrame
df_scaled['label'] = df['label']
```

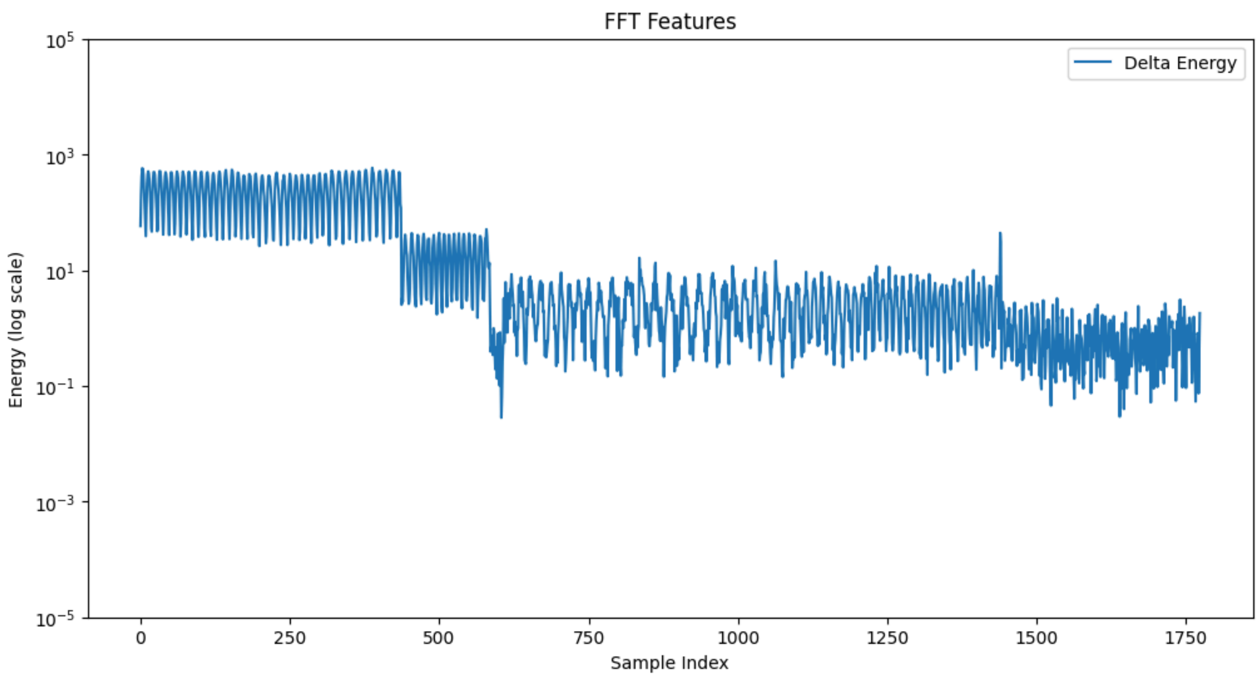
```
#df_scaled
X_scaled
```

```
array([[ 0.06189658, -0.02420454,  0.15837727, ..., -0.20700494,
        -0.83657858, -0.44954174],
       [ 0.17425308,  0.76979996,  0.40360315, ..., -0.20700494,
        -0.98141679,  0.39735532],
       [ 0.08347964, -0.46573206, -0.09385594, ..., -0.20700494,
        -1.032183  ,  1.46086289],
       ...,
       [-0.18803801, -0.38864419, -0.18803075, ..., -0.20700494,
         3.25835678, -1.21089296],
       [-0.22226553, -0.31774083, -0.05896843, ..., -0.20700494,
         1.01918978, -1.68822795],
       [-0.20404125, -0.20199037, -0.1142227 , ..., -0.20700494,
        -0.24351889, -0.47558243]])
```

```
import matplotlib.pyplot as plt



# Plot FFT features
plt.figure(figsize=(12, 6))
#plt.plot(df.index, df['E_alpha'], label='Alpha Energy')
#plt.plot(df.index, df['E_beta'], label='Beta Energy')
###plt.plot(df.index, df['E_theta'], label='theta Energy')
#plt.plot(df.index, df['E_delta'], label='delta Energy')

plt.plot(df.index, df['alpha_beta_ratio'], label='ratio')

plt.xlabel('Sample Index')
plt.ylabel('Energy')
plt.title('FFT Features')
plt.legend()
plt.show()
```



```
import matplotlib.pyplot as plt

# Plot FFT features
plt.figure(figsize=(12, 6))
#plt.plot(df.index, df['E_alpha'], label='Alpha Energy')
#plt.plot(df.index, df['E_beta'], label='Beta Energy')
#plt.plot(df.index, df['E_theta'], label='Theta Energy')
plt.plot(df.index, df['E_delta'], label='Delta Energy')
#plt.plot(df.index, df['alpha_beta_ratio'], label='Alpha/Beta Ratio')

plt.yscale('log')

threshold = 1e5
plt.ylim([1e-5, threshold])

plt.xlabel('Sample Index')
plt.ylabel('Energy (log scale)')
plt.title('FFT Features')
plt.legend()
plt.show()
```



```
df_cleaned = df_scaled.dropna(subset=['label'])
X = df_cleaned.drop('label', axis=1)
y = df_cleaned['label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf = SVC(probability=True, random_state=42)


clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Classification Accuracy: {accuracy}")
```

```
Classification Accuracy: 0.9464788732394366
```

```
import numpy as np
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

X = df_scaled.drop('label', axis=1)
y = df_scaled['label']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': ['scale', 'auto', 0.1, 0.01, 0.001, 0.0001],
    'kernel': ['rbf']
}

svc = SVC(probability=True)

grid_search = GridSearchCV(estimator=svc, param_grid=param_grid, cv=5, verbose=2, n_jobs=-1)

grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

model = grid_search.best_estimator_
y_pred = model.predict(X_test)
test_accuracy = model.score(X_test, y_test)
print("Test set accuracy: {:.2f}".format(test_accuracy))
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
Best parameters: {'C': 100, 'gamma': 'scale', 'kernel': 'rbf'}
Best cross-validation score: 0.97
Test set accuracy: 0.95
```

```
X_test
```

	E_alpha	E_beta	E_theta	E_delta	alpha_beta_ratio	peak_frequency	spectral_centroid	spectral_slope
999	0.244763	0.249382	0.442216	-0.466677	0.741188	-0.207005	0.443396	-0.316921
596	-0.158342	-0.363658	-0.087540	-0.481704	-0.399984	-0.207005	1.186089	-1.373899
1132	-0.184564	-0.248835	-0.206813	-0.477541	-0.761048	-0.207005	0.459467	0.551357
270	0.113024	0.006009	-0.158286	2.830191	0.630179	-0.207005	-1.032833	1.720390
414	-0.050409	0.483859	-0.145504	2.014416	-0.598028	-0.207005	-1.020050	1.924875
...	...	...	...	...	...	...	...	...
1222	-0.209812	0.007804	-0.198778	-0.438816	-1.046661	-0.207005	-0.686073	0.522795
1613	-0.181224	-0.248525	-0.158523	-0.484161	-0.736664	-0.207005	2.730963	-1.090367
584	-0.045968	-0.122748	-0.040088	-0.385894	-0.004020	-0.207005	-0.688396	-0.148971
198	-0.121017	0.218898	-0.147804	-0.145072	-0.723166	-0.207005	-0.603994	1.210080
15	0.071891	0.503975	-0.153008	2.130754	-0.206253	-0.207005	-1.017870	1.988917

355 rows × 8 columns

Next steps: Generate code with X_test New interactive sheet

```
import pickle

model_filename = 'model.pkl'

with open(model_filename, 'wb') as file:
    pickle.dump(model, file)
```

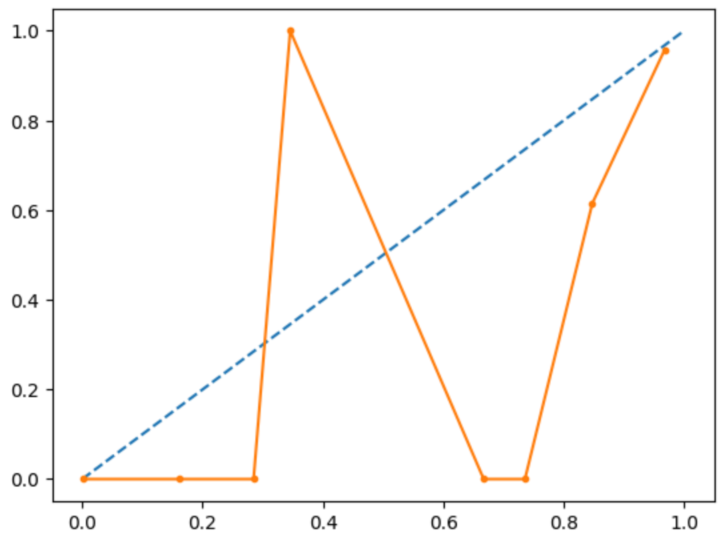
```
import pickle

scaler_filename = 'scaler.pkl'

with open(scaler_filename, 'wb') as file:
    pickle.dump(scaler, file)
```

```
probabilities = model.predict_proba(X_test)[:,1]
print(f"Class Probabilities: {probabilities}")
```

```
 9.78152242e-01 9.99999641e-01 9.30032761e-01 9.67963766e-01
 1.00000010e-07 9.99999983e-01 9.50678949e-01 9.80924841e-01
 9.91077463e-01 1.00000010e-07 1.00000010e-07 1.00000010e-07
 9.91902274e-01 9.47343239e-01 2.85209077e-07 9.81436745e-01
 5.26954603e-06 1.00000010e-07 1.00000010e-07 2.50739337e-05
 6.88338341e-04 9.66198130e-01 9.97299940e-01 8.68975328e-05
 1.00000010e-07 9.52113837e-01 1.00000010e-07 9.92471742e-01
 1.00000010e-07 1.00000010e-07 9.21119207e-01 9.60831595e-01
 1.00000010e-07 9.79228151e-01 9.60466558e-01 9.96294921e-01
 9.71724853e-01 9.34850339e-01 9.73672578e-01 1.00000010e-07
 9.78449809e-01 1.00000010e-07 8.76558927e-05 9.84300729e-01
 9.54536461e-01 9.82643058e-01 1.00000010e-07 3.74374765e-03
 1.00000010e-07 9.30887077e-01 9.99993997e-01 9.66210833e-01
 1.00000010e-07 9.48499048e-01 9.87591360e-01 9.79274542e-01
 9.67616516e-01 9.74978334e-01 9.73281619e-01 9.60688484e-01
 9.15869432e-01 9.50377582e-01 1.00000010e-07 8.11913206e-01
 9.64519563e-01 9.85821630e-01 2.84729145e-01 9.19088641e-01
 9.14476905e-01 1.00000010e-07 9.86481358e-01 9.83997562e-01
 4.52196565e-07 1.00000010e-07 9.29037633e-01 1.62366346e-01
 8.65371759e-01 1.00000010e-07 9.83220558e-01 1.00000010e-07
 1.00000010e-07 9.33653217e-01 9.62524670e-01 9.60203134e-01
 9.99999977e-01 1.00000010e-07 9.99996936e-01 9.52120215e-01
 9.97287899e-01 1.00000010e-07 1.00000010e-07 9.99999932e-01
 9.94255149e-01 9.49896637e-01 9.77005856e-01 9.81172060e-01
 9.65451176e-01 1.00000010e-07 1.05108287e-03 9.63980742e-01
 1.00000010e-07 9.57766423e-01 9.84546711e-01 9.96745803e-01
 8.98557650e-01 6.84771501e-04 9.67718541e-01 9.79013702e-01
 9.10998752e-01 1.85254885e-03 9.99990675e-01 1.00000010e-07
 1.00000010e-07 5.42825905e-05 1.00000010e-07 9.79735109e-01
 9.60886396e-01 6.64763655e-05 9.27692289e-01 9.61907504e-01
 9.99989321e-01 9.38987249e-01 9.94443837e-01 9.46699883e-01
 1.00000010e-07 9.79419464e-01 9.99999208e-01 9.85464795e-01
 8.19231866e-01 1.00000010e-07 1.00000010e-07 9.99999871e-01
 9.33282766e-01 9.93094703e-01 9.93508191e-01 9.60505589e-01
 8.75531427e-01 9.53487558e-01 8.51267304e-01 9.99999689e-01
 9.51188698e-01 9.86083267e-01 9.62858874e-01 1.00000010e-07
 9.69459238e-01 1.00000010e-07 9.27319153e-01 9.75645257e-01
 9.72566633e-01 9.69819146e-01 9.92778106e-01 9.80541536e-01
 1.00000010e-07 9.59667122e-01 1.00000010e-07 1.00000010e-07
 9.63929207e-01 9.83992189e-01 9.27814988e-01 9.65324366e-01
 1.01216957e-05 1.00000010e-07 9.72266547e-01 1.00000010e-07
 9.51772307e-01 9.76844733e-01 9.56172479e-01 9.68171972e-01
 9.69773425e-01 9.71464492e-01 1.00000010e-07 9.81986171e-01
 9.43421479e-01 9.78530692e-01 9.69539760e-01 9.57607412e-01
 1.00000010e-07 1.00000010e-07 4.71585446e-04 9.86736104e-01
 1.00000010e-07 6.06768070e-06 9.68453227e-01 9.14701969e-01
 9.99999901e-01 3.94172974e-03 9.59749613e-01 9.99981639e-01
 9.54658685e-01 9.52235060e-01 1.00000010e-07 9.73264877e-01
 9.41992978e-01 6.58788166e-07 9.93533704e-01 9.61274186e-01
 1.00000010e-07 9.44349348e-01 9.99997197e-01 9.68769403e-01
 9.56822848e-01 9.61548217e-01 9.97138755e-01 1.00000010e-07
 9.30687566e-01 9.96526523e-01 9.49564343e-01 1.00000010e-07
 9.64374309e-01 9.70597787e-01 9.62673807e-01 9.48350535e-01
 9.57638413e-01 9.99998902e-01 9.63075918e-01 9.73700581e-01
 1.12554042e-05 1.00000010e-07 9.92448766e-01 2.04461131e-06
 1.00000010e-07 9.63684463e-01 9.63782961e-01 9.99999333e-01
 9.42066235e-01 9.86664752e-01 8.32445299e-01 9.99999968e-01
 7.35620353e-01 1.00000010e-07 1.00000010e-07]
```

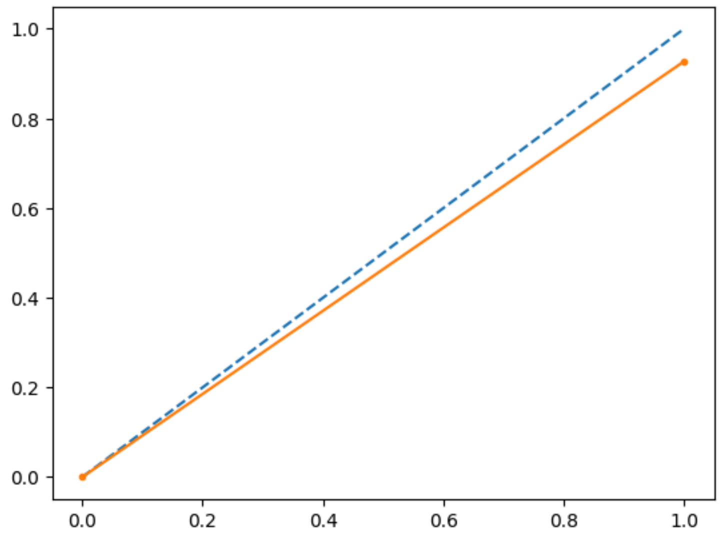
```
from sklearn.calibration import calibration_curve
```

```
fop, mpv = calibration_curve(y_test, probabilities, n_biwwns=10)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(mpv, fop, marker='.')
plt.show()
```



```
from sklearn.calibration import CalibratedClassifierCV

calibrator = CalibratedClassifierCV(model, cv=3)
calibrator.fit(X_train,y_train)
yhat = calibrator.predict(X_test)
```

```
fop, mpv = calibration_curve(y_test, yhat, n_bins=10)
plt.plot([0, 1], [0, 1], linestyle='--')
plt.plot(mpv, fop, marker='.')
plt.show()
```



```
accuracy = accuracy_score(y_test, yhat)
print(f"Classification Accuracy: {accuracy}")
```

```
Classification Accuracy: 0.9492957746478873
```

Start coding or generate with AI.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_repor

y_true = df_scaled['label']
y_pred = model.predict(X_scaled)

# Accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f'Accuracy: {accuracy:.4f}')

# Precision
precision = precision_score(y_true, y_pred)
```

```
print(f'Precision: {precision:.4f}')

# Recall
recall = recall_score(y_true, y_pred)
print(f'Recall: {recall:.4f}')

# F1 Score
f1 = f1_score(y_true, y_pred)
print(f'F1 Score: {f1:.4f}')

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print('Confusion Matrix:')
print(conf_matrix)

# Classification Report
class_report = classification_report(y_true, y_pred)
print('Classification Report:')
print(class_report)
```

```
Accuracy: 0.9707
Precision: 0.9588
Recall: 0.9992
F1 Score: 0.9786
Confusion Matrix:
[[ 535   51]
 [   1 1187]]
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      0.91      0.95       586
         1.0       0.96      1.00      0.98      1188
```