In [62]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [63]:
```
df = pd.read_csv('KNN_Project_Data')
df.head()
```

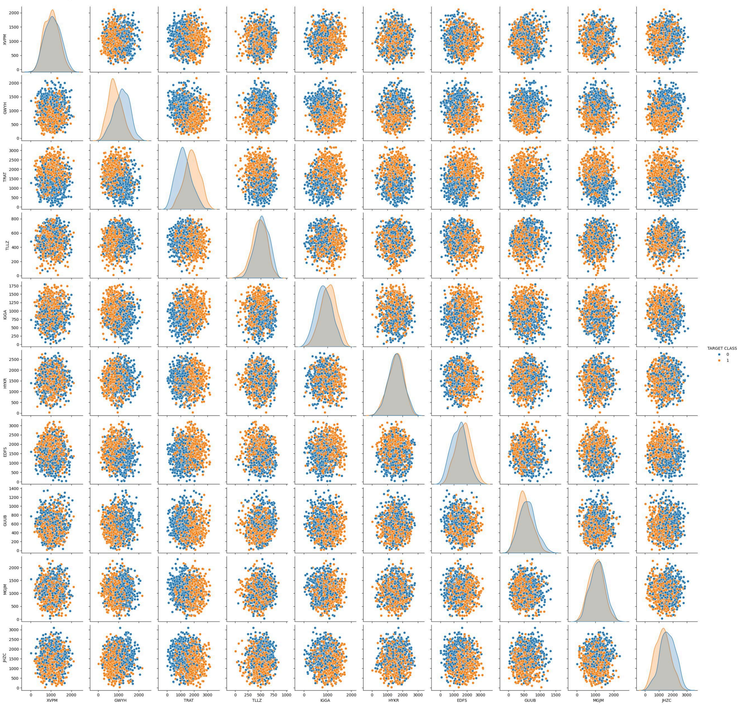
Out[63]:

| | XVPM | GWYH | TRAT | TLLZ | IGGA | HYKR | EDFS | GUUB | MGJM | JHZC | TARGET CLASS |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1636.670614 | 817.988525 | 2565.995189 | 358.347163 | 550.417491 | 1618.870897 | 2147.641254 | 330.727893 | 1494.878631 | 845.136088 | 0 |
| **1** | 1013.402760 | 577.587332 | 2644.141273 | 280.428203 | 1161.873391 | 2084.107872 | 853.404981 | 447.157619 | 1193.032521 | 861.081809 | 1 |
| **2** | 1300.035501 | 820.518697 | 2025.854469 | 525.562292 | 922.206261 | 2552.355407 | 818.676686 | 845.491492 | 1968.367513 | 1647.186291 | 1 |
| **3** | 1059.347542 | 1066.866418 | 612.000041 | 480.827789 | 419.467495 | 685.666983 | 852.867810 | 341.664784 | 1154.391368 | 1450.935357 | 0 |
| **4** | 1018.340526 | 1313.679056 | 950.622661 | 724.742174 | 843.065903 | 1370.554164 | 905.469453 | 658.118202 | 539.459350 | 1899.850792 | 0 |

## Pairplot of Data

In [64]:
```
sns.pairplot(df, hue='TARGET CLASS')
```

Out[64]:
```
<seaborn.axisgrid.PairGrid at 0x1273dd0d630>
```




## Standardizing Features Using StandardScaler in scikit-learn

In [65]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(df.drop('TARGET CLASS', axis=1))
```

Out[65]:
```
StandardScaler()
```

In [66]:
```
scaled_features = scaler.transform(df.drop('TARGET CLASS', axis=1))

df_feat = pd.DataFrame(scaled_features, columns=df.columns[:-1])

df_feat.head()
```

Out[66]:

| | XVPM | GWYH | TRAT | TLLZ | IGGA | HYKR | EDFS | GUUB | MGJM | JHZC |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.568522 | -0.443435 | 1.619808 | -0.958255 | -1.128481 | 0.138336 | 0.980493 | -0.932794 | 1.008313 | -1.069627 |
| **1** | -0.112376 | -1.056574 | 1.741918 | -1.504220 | 0.640009 | 1.081552 | -1.182663 | -0.461864 | 0.258321 | -1.041546 |
| **2** | 0.660647 | -0.436981 | 0.775793 | 0.213394 | -0.053171 | 2.030872 | -1.240707 | 1.149298 | 2.184784 | 0.342811 |
| **3** | 0.011533 | 0.191324 | -1.433473 | -0.100053 | -1.507223 | -1.753632 | -1.183561 | -0.888557 | 0.162310 | -0.002793 |
| **4** | -0.099059 | 0.820815 | -0.904346 | 1.609015 | -0.282065 | -0.365099 | -1.095644 | 0.391419 | -1.365603 | 0.787762 |

## Training and Testing a K-Nearest Neighbors Classifier

In [67]:
```
from sklearn.model_selection import train_test_split

X = df_feat
y = df['TARGET CLASS']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [68]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train,y_train)
```

Out[68]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [69]:
```
predictions = knn.predict(X_test)
```

## Evaluating Model Performance with Classification Report and Confusion Matrix

In [70]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.70      0.73      0.72       163
           1       0.73      0.70      0.71       167

    accuracy                           0.72       330
   macro avg       0.72      0.72      0.72       330
weighted avg       0.72      0.72      0.72       330
```

In [71]:
```
pd.DataFrame(confusion_matrix(y_test, predictions))
```

Out[71]:

| | 0 | 1 |
|---|---|---|
| **0** | 119 | 44 |
| **1** | 50 | 117 |

The output shows how well the KNN model works:

- The model is 72% accurate overall.
- It performs similarly for both classes, with scores like precision and recall around 0.72.
- The confusion matrix shows correct predictions (on the diagonal) and mistakes for each class.

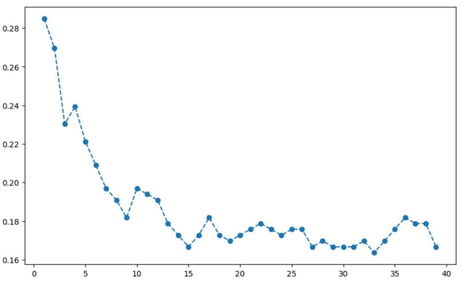
## Plotting Error Rate vs. Number of Neighbors

In [72]:
```
error_rate = []

for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors= i)
    knn.fit(X_train, y_train)
    predict_i = knn.predict(X_test)
    error_rate.append(np.mean(predict_i!=y_test))
```

In [73]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate, ls='--', marker = 'o')
```

Out[73]:
```
[<matplotlib.lines.Line2D at 0x12745939ea0>]
```



## Model Evaluation with k = 30 in KNN

In [74]:
```
knn = KNeighborsClassifier(n_neighbors= 30)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.81      0.87      0.84       163
           1       0.86      0.80      0.83       167

    accuracy                           0.83       330
   macro avg       0.83      0.83      0.83       330
weighted avg       0.83      0.83      0.83       330
```

In [75]:
```
pd.DataFrame(confusion_matrix(y_test, predictions))
```

Out[75]:

| | 0 | 1 |
|---|---|---|
| **0** | 141 | 22 |
| **1** | 33 | 134 |

When comparing k=1 and k=30, the model performs better with k=30. The accuracy improves from 72% to 83%, showing that increasing k leads to fewer mistakes. The confusion matrix also shows fewer misclassifications with k=30 (22 for Class 0 and 33 for Class 1) compared to k=1 (44 for Class 0 and 50 for Class 1). Overall, using k=30 gives a more accurate and reliable model.