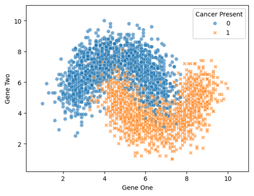
In [ ]:

In [ ]:
```
***Project Topic: Python Machine Learning (K Nearest Neighbors (KNN)) Analysis on "gene expression dataset".

Dataset Source:

Data was downloaded from Kaggle webpage. The license is shown as unknown.

Goals:
This project is totally generated for my educational and portfolio purposes to present data analysis skills in:
•       Data preprocessing,
•       Data analysis
•       KNN modeling including Python and scikit-learn
•       Model evaluation
•       Model visualization

Acknowledgement:

I do not own the dataset and am not redistributing it. All rights to this dataset relate to the original uploader on Kaggle.

***
```

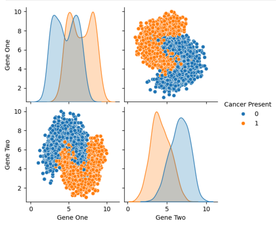
In [41]:
```
# Python_Machine_Learning_K_Nearest_Neighbors (KNN)_project_04
```

K nearest neighbors (KNN), supervised learning algorithms.

In [123]:
```
import pandas as pd
df = pd.read_csv("gene_expression.csv")
df.head

# visualization with scatterplot
import matplotlib.pyplot as plt
import seaborn as sns

sns.scatterplot(data = df, x = "Gene One", y = "Gene Two", hue = "Cancer Present", alpha = 0.6, style="Cancer Present")

plt.xlim(0.2,11)
plt.ylim(0.2, 11)

plt.show()
```



In [125]:
```
# pairplot
sns.pairplot(data = df, hue = "Cancer Present")
plt.show()
```



In [127]:
```
# train test split
from sklearn.model_selection import train_test_split

# Look at "Gene One", "Gene Two" columns
X = df.drop('Cancer Present', axis = 1)

# Look at "Cancer Present" column
y = df["Cancer Present"]

# split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 101)

# scale the dataset
from sklearn.preprocessing import StandardScaler
Standard_Scaler = StandardScaler()

X_train_scale = Standard_Scaler.fit_transform(X_train)  # fit this data
X_test_scale = Standard_Scaler.transform(X_test)        # here only transform data.

# Make a model.
# Load the KNN (KNeighborsClassifie)
from sklearn.neighbors import KNeighborsClassifier

# Generate a model for the number of neaest neighbor = 2
model_KNN = KNeighborsClassifier(n_neighbors = 2)

# Fit the model with scaled_X_train, and y_train
model_KNN.fit(X_train_scale, y_train)

# Model evaluation with scaled_X_test
y_prediction = model_KNN.predict(X_test_scale)

# Load confusion_matrix, classification_report from sklearn.metrics
from sklearn.metrics import confusion_matrix, classification_report

# Check accuracy
confusion_matrix(y_test, y_prediction )
print(confusion_matrix(y_test, y_prediction ))

# Look the classification report.
print(classification_report(y_test, y_prediction ))

#  Count cancer patient and non cancer patient
df['Cancer Present'].value_counts()
print(df['Cancer Present'].value_counts())
```

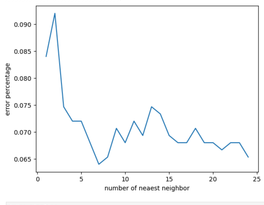
```
[[336  18]
 [ 51 345]]
              precision    recall  f1-score   support

           0       0.87      0.95      0.91       354
           1       0.95      0.87      0.91       396

    accuracy                           0.91       750
   macro avg       0.91      0.91      0.91       750
weighted avg       0.91      0.91      0.91       750

Cancer Present
1    1500
0    1500
Name: count, dtype: int64
```

In [129]:
```
# Load Accuracy scoring function
from sklearn.metrics import accuracy_score

# Find accuracy
accuracy_score_check = accuracy_score(y_test, y_prediction)

print(accuracy_score_check)

# Find error
print( 1 - accuracy_score_check)
```

```
0.908
0.09199999999999997
```

In [131]:
```
# Use "elbo method" to find a k-value at minimum error

# Create a store
error_percentage = []


for knn in range(1, 25):
    model_KNN_1 = KNeighborsClassifier(n_neighbors=knn)
    model_KNN_1.fit(X_train_scale, y_train)
    y_prediction_x_test = model_KNN_1.predict(X_test_scale)
    error_percentage_test = 1 - accuracy_score(y_test, y_prediction_x_test)
    error_percentage.append(error_percentage_test)
print(error_percentage)

# plot "error percentage" as a function of "number of nearest neighbors"
plt.plot(range(1, 25), error_percentage)    # like plt.plot(x, y )

plt.ylabel("error percentage")
plt.xlabel("number of neaest neighbor")

plt.show()
```

```
[0.08399999999999996, 0.09199999999999997, 0.07466666666666666, 0.07199999999999995, 0.07199999999999995, 0.06799999999999995, 0.06399999999999995, 0.06533333333333335, 0.07066666666666666, 0.06799999999999995, 0.07199999999999995, 0.06933333333333336, 0.07466666666666666, 0.07333333333333336, 0.06933333333333336, 0.06799999999999995, 0.06799999999999995, 0.07066666666666666, 0.06799999999999995, 0.06799999999999995, 0.06666666666666665, 0.06799999999999995, 0.06799999999999995, 0.06533333333333335]
```



In [133]:
```
# Use pipeline

# Initialize a KNeighborsClassifier
KNNeg = KNeighborsClassifier()

# Look all parameters
print(KNNeg .get_params().keys() )

# Make a pipeline using 2 steps
scale_KNNeg = [("Standard_Scales", Standard_Scale), ("KNNeg", KNNeg )]

# Load the pipeline
from sklearn.pipeline import Pipeline

# Make a pipeline
pipe_line = Pipeline(scale_KNNeg)

# setup k values
k_list = list(range(1,25))

print(k_list)
```

```
dict_keys(['algorithm', 'leaf_size', 'metric', 'metric_params', 'n_jobs', 'n_neighbors', 'p', 'weights'])
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
```

In [135]:
```
# Load  gridsearch CV

# Load GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define grid
k_list_KNNeg = {'KNNeg__n_neighbors': k_list}

# Make  GridSearchCV
cv_list = GridSearchCV(pipe_line, k_list_KNNeg, scoring='accuracy', cv=7 )

# fit this model
print(cv_list.fit(X_train, y_train))
```

```
GridSearchCV(cv=7,
             estimator=Pipeline(steps=[('Standard_Scales', StandardScaler()),
                                       ('KNNeg', KNeighborsClassifier())]),
             param_grid={'KNNeg__n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
                                                11, 12, 13, 14, 15, 16, 17, 18,
                                                19, 20, 21, 22, 23, 24]},
             scoring='accuracy')
```

In [137]:
```
# get the best estimator
print(cv_list.best_estimator_.get_params())
```

```
{'memory': None, 'steps': [('Standard_Scales', StandardScaler()), ('KNNeg', KNeighborsClassifier(n_neighbors=24))], 'verbose': False, 'Standard_Scales': StandardScaler(), 'KNNeg': KNeighborsClassifier(n_neighbors=24), 'Standard_Scales__copy': True, 'Standard_Scales__with_mean': True, 'Standard_Scales__with_std': True, 'KNNeg__algorithm': 'auto', 'KNNeg__leaf_size': 30, 'KNNeg__metric': 'minkowski', 'KNNeg__metric_params': None, 'KNNeg__n_jobs': None, 'KNNeg__n_neighbors': 24, 'KNNeg__p': 2, 'KNNeg__weights': 'uniform'}
```

In [139]:
```
# Get the best parameters
print("Best_paramet_cv_list:", cv_list.best_params_)

# Get the best estimator
print("best_estimator_n_neighbors:",
      cv_list.best_estimator_.named_steps['KNNeg'].n_neighbors)
```

```
Best_paramet_cv_list: {'KNNeg__n_neighbors': 24}
best_estimator_n_neighbors: 24
```

In [140]:
```
# Fit the cv_list
cv_list.fit(X_train.values, y_train)

# Prediction from test data
total_prediction = cv_list.predict(X_test.values)

# Load classification_report
from sklearn.metrics import classification_report

# Get classification report
print(classification_report(y_test, total_prediction))
```

```
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       354
           1       0.94      0.94      0.94       396

    accuracy                           0.93       750
   macro avg       0.93      0.93      0.93       750
weighted avg       0.93      0.93      0.93       750
```

In [141]:
```
# New patient prediction

import numpy as np

# 2 features[ Gene One = 2.5, Gene Two = 5.7]
patient_visit_hospital = np.array([[2.5, 5.7]])

# Make a prediction using the pipeline.
result = cv_list.predict(patient_visit_hospital )
print(result)                                  #  No Cancer record
```

```
[0]
```

In [145]:
```
# predict the probability
print(cv_list.predict_proba(patient_visit_hospital))      # 100% confident, No Cancer record.
```

```
[[1. 0.]]
```

In [ ]:
```

```