In [1]:
```
# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# To tune model, get different metric scores and split data
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_predict
#from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
# To get diferent metric scores
from sklearn.metrics import confusion_matrix, precision_recall_curve, classificat

# To suppress the warning
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# Load the data into a pandas dataframe named data_firstname_df2 where first name
# name.
```

In [3]:
```
data_chitra= pd.read_csv('C:/Users/chitr/OneDrive - Centennial College/Attachment
```

In [4]:
```
# Replace the ‘?’ mark in the ‘bare’ column by np.nan and change the type to ‘flo
```

In [5]:
```
data_chitra['bare']=data_chitra['bare'].replace('?', np.nan)
```

In [6]:
```
data_chitra['bare']=data_chitra['bare'].astype(float)
```

In [7]:
```
data_chitra.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   ID         699 non-null    int64  
 1   thickness  699 non-null    int64  
 2   size       699 non-null    int64  
 3   shape      699 non-null    int64  
 4   Marg       699 non-null    int64  
 5   Epith      699 non-null    int64  
 6   bare       683 non-null    float64
 7   b1         699 non-null    int64  
 8   nucleoli   699 non-null    int64  
 9   Mitoses    699 non-null    int64  
 10  class      699 non-null    int64  
dtypes: float64(1), int64(10)
memory usage: 60.2 KB
```

In [8]:
```
# Drop the ID column
del data_chitra['ID']
```

In [9]:
```
data_chitra.head(3)
```

Out[9]:

	thickness	size	shape	Marg	Epith	bare	b1	nucleoli	Mitoses	class
0	5	1	1	1	2	1.0	3	1	1	2
1	5	4	4	5	7	10.0	3	2	1	2
2	3	1	1	1	2	2.0	3	1	1	2

In [10]:
```
# Separate the features from the class.

X = data_chitra.drop('class', axis=1)
y = data_chitra[['class']]
```

In [11]:
```
# plit your data into train 80% train and 20% test, use the last two digits of yo
# for the seed.
```

In [12]:
```
from sklearn.preprocessing import FunctionTransformer
```

In [13]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=7
print("Size of training set:",X_train.shape)
print("Size of testing set:",X_test.shape)
```

```
Size of training set: (559, 9)
Size of testing set: (140, 9)
```

In [14]:
```
# Using the preprocessing library to define two transformer objects to transform
# data:
# a. Fill the missing values with the median (hint: checkout SimpleImputer)
# b. Scale the data (hint: checkout StandardScaler
```

In [15]:
```

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

In [16]:
```
# combine the two transformers into a pipeline name it num_pipe_firstname.
```

In [17]:
```
transformer_simple_imputer = FunctionTransformer(SimpleImputer)
transformer_normalize_data = FunctionTransformer(StandardScaler)
```

In [18]:
```
from sklearn.pipeline import Pipeline
num_pipe_chitra = Pipeline(steps=[
    ('simple_imputer', SimpleImputer()),
    ('normalize_data', StandardScaler()),
])
num_pipe_chitra
```

Out[18]:
```
Pipeline(steps=[('simple_imputer', SimpleImputer()),
                ('normalize_data', StandardScaler())])
```

In [19]:
```
# Create a new Pipeline that has two steps the first is the num_pipe_firstname ar
# an SVM classifier with random state = last two digits of your student number. N
# pipe_svm_firstname. (make note of the labels)
```

In [20]:
```
from sklearn.pipeline import Pipeline
pipe_svm_chitra = Pipeline(steps=[
    ('num_pipe_chitra', num_pipe_chitra),
    ('svc', SVC(random_state = 74)) ])
pipe_svm_chitra
```

Out[20]:
```
Pipeline(steps=[('num_pipe_chitra',
                 Pipeline(steps=[('simple_imputer', SimpleImputer()),
                                 ('normalize_data', StandardScaler())])),
                ('svc', SVC(random_state=74))])
```

In [21]:
```
pipe_svm_chitra.fit(X_train, y_train)
```

Out[21]:
```
Pipeline(steps=[('num_pipe_chitra',
                 Pipeline(steps=[('simple_imputer', SimpleImputer()),
                                 ('normalize_data', StandardScaler())])),
                ('svc', SVC(random_state=74))])
```

In [22]:
```
pipe_svm_chitra.score(X_train, y_train)
```

Out[22]:
```
0.9713774597495528
```

In [23]:
```
pipe_svm_chitra.score(X_test, y_test)
```

Out[23]:
```
0.9785714285714285
```

In [24]:
```
# Define the grid search parameters in an object and name it param_grid, as follo
# a. 'svc__kernel': ['linear', 'rbf','poly'],
# b. 'svc__C': [0.01,0.1, 1, 10, 100],
# c. 'svc__gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0],
# d. 'svc__degree':[2,3]},
# Make sure you replace svc with the label you used in the pipe_svm_firstname for
```

In [25]:
```
# grid search
param_grid= {'svc__kernel': ['linear', 'rbf','poly'],
             'svc__C':  [0.01,0.1, 1, 10, 100],
              'svc__gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0],
              'svc__degree':[2,3]}
#base_estimator=pipe_svm_chitra()
grid_search_chitra  = GridSearchCV(pipe_svm_chitra,param_grid,n_jobs=8,verbose=Tr
grid_search_chitra.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
```

Out[25]:
```
GridSearchCV(estimator=Pipeline(steps=[('num_pipe_chitra',
                                        Pipeline(steps=[('simple_imputer',
                                                         SimpleImputer()),
                                                        ('normalize_data',
                                                         StandardScaler())])),
                                       ('svc', SVC(random_state=74))]),
             n_jobs=8,
             param_grid={'svc__C': [0.01, 0.1, 1, 10, 100],
                         'svc__degree': [2, 3],
                         'svc__gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0],
                         'svc__kernel': ['linear', 'rbf', 'poly']},
             verbose=True)
```

In [26]:
```
# Fit your training data to the gird search object. (This will take some time but
# results on the console
```

In [27]:
```
grid_search_chitra = GridSearchCV(estimator = pipe_svm_chitra, param_grid = param

grid_search_chitra.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 180 candidates, totalling 900 fits
[CV 1/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=linear;
total time=   0.0s
[CV 2/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=linear;
total time=   0.0s
[CV 3/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=linear;
total time=   0.0s
[CV 4/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=linear;
total time=   0.0s
[CV 5/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=linear;
total time=   0.0s
[CV 1/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=rbf; to
tal time=   0.0s
[CV 2/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=rbf; to
tal time=   0.0s
[CV 3/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=rbf; to
tal time=   0.0s
[CV 4/5] END svc__C=0.01, svc__degree=2, svc__gamma=0.01, svc__kernel=rbf; to
tal time=   0.0s
```

In [28]:
```
# Print out the best parameters and note it in your written response
```

In [29]:
```
grid_search_chitra.best_params_
```

Out[29]:
```
{'svc__C': 0.1, 'svc__degree': 2, 'svc__gamma': 0.01, 'svc__kernel': 'linear'}
```

In [30]:
```
# Printout the best estimator and note it in your written response
```

In [31]:
```
grid_search_chitra.best_estimator_
```

Out[31]:
```
Pipeline(steps=[('num_pipe_chitra',
                 Pipeline(steps=[('simple_imputer', SimpleImputer()),
                                 ('normalize_data', StandardScaler())])),
                ('svc',
                 SVC(C=0.1, degree=2, gamma=0.01, kernel='linear',
                     random_state=74))])
```

In [32]:
```
grid_search_chitra.score(X_train, y_train)
```

Out[32]:
```
0.9695885509838998
```

In [33]:
```
# Predict the test data using the fine-tuned model identified during grid search
# estimator saved in the grid search object and note it in your written response.
```

In [34]:
```
grid_search_chitra.score(X_test, y_test)
```

Out[34]:
```
0.9785714285714285
```

In [35]:
```
#  Printout the accuracy score and note it in your written response.
```

In [36]:
```
# Create an object that holds the best model i.e. best estimator to an object nam
# best_model_firstname
# 20. Save the model using the joblib (dump).
# 21. Save the full pipeline using the joblib - (dump).
```

In [37]:
```
import joblib

best_model_chitra = grid_search_chitra.best_estimator_
joblib.dump(best_model_chitra, 'best_model_chitra.pkl', compress = 1)
joblib.dump(pipe_svm_chitra, 'svm_pipeline.pkl', compress = 1)
```

Out[37]:
```
['svm_pipeline.pkl']
```

In [ ]: