COMPARING LOGISTIC REGRESSION, SVM, RANDOM FOREST, K-NN AND NAIVE-BAYES

Basic Machine Learning Final Project

By : Group 1

Contain

- Dataset Description
- Data Loading
- Data Preprocessing
- Predictive Model
- Summary

Dataset Description

- Dataset mengenai kanker payudara yang ada didapat melalui perhitungan terhadap gambar digital atas uji Aspirasi Jarum Halus (FNA) dari massa payudara. Data ini menggambarkan karakteristik dari inti sel
- Tujuan dibuatnya dataset ini adalah untuk mengidentifikasi jumlah kelas kanker yang jinak (benign) atau ganas (malignant)
- Sample ini dikumpulkan secara periodical atas laporan klinis dari Dr. Wolberg, oleh karena itu database ini mencerminkan pengelompokan secara kronologis

Data Loading

1. Loading dataset

In [2]:
```
df = pd.read_csv("data.csv")
df
```

0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430
...	...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000

569 rows × 33 columns

2. Getting data information

```
In [3]: df.info()
```

```
---  ------                  --------------  -----
0    id                      569 non-null    int64
1    diagnosis               569 non-null    object
2    radius_mean             569 non-null    float64
3    texture_mean            569 non-null    float64
4    perimeter_mean          569 non-null    float64
5    area_mean               569 non-null    float64
6    smoothness_mean         569 non-null    float64
7    compactness_mean        569 non-null    float64
8    concavity_mean          569 non-null    float64
9    concave points_mean     569 non-null    float64
10   symmetry_mean           569 non-null    float64
11   fractal_dimension_mean  569 non-null    float64
12   radius_se               569 non-null    float64
13   texture_se              569 non-null    float64
14   perimeter_se            569 non-null    float64
15   area_se                 569 non-null    float64
16   smoothness_se           569 non-null    float64
17   compactness_se          569 non-null    float64
18   concavity_se            569 non-null    float64
19   concave points_se       569 non-null    float64
20   symmetry_se             569 non-null    float64
21   fractal_dimension_se    569 non-null    float64
22   radius_worst            569 non-null    float64
23   texture_worst           569 non-null    float64
24   perimeter_worst         569 non-null    float64
25   area_worst              569 non-null    float64
26   smoothness_worst        569 non-null    float64
27   compactness_worst       569 non-null    float64
28   concavity_worst         569 non-null    float64
29   concave points_worst    569 non-null    float64
30   symmetry_worst          569 non-null    float64
31   fractal_dimension_worst 569 non-null    float64
32   Unnamed: 32             0 non-null      float64
```

```
In [4]: df.isnull().sum()
```

```
id                           0
diagnosis                    0
radius_mean                  0
texture_mean                 0
perimeter_mean               0
area_mean                    0
smoothness_mean              0
compactness_mean             0
concavity_mean               0
concave points_mean          0
symmetry_mean                0
fractal_dimension_mean       0
radius_se                    0
texture_se                   0
perimeter_se                 0
area_se                      0
smoothness_se                0
compactness_se               0
concavity_se                 0
concave points_se            0
symmetry_se                  0
fractal_dimension_se         0
radius_worst                 0
texture_worst                0
perimeter_worst              0
area_worst                   0
smoothness_worst             0
compactness_worst            0
concavity_worst              0
concave points_worst         0
symmetry_worst               0
fractal_dimension_worst      0
Unnamed: 32                569
```

```
In [5]: df["diagnosis"].value_counts()
Out[5]: B    357
        M    212
        Name: diagnosis, dtype: int64
```

Data Preprocessing

1. Drop features

```
In [6]: df = df.drop('Unnamed: 32',axis = 1)
        df = df.drop('id',axis = 1)
```

```
In [8]: features = df.iloc[:,0:31]
        features = features.drop('diagnosis',axis = 1)
```

2. Checking and handling outliers

```
In [9]: from scipy import stats

        z = np.abs(stats.zscore(features._get_numeric_data()))
        not_outlier = df[(z < 3).all(axis = 1)]
        print(not_outlier.shape)

        (495, 31)
```

```
In [10]: pd_outlier = pd.DataFrame([])
         pd_outlier["Notes"] = ["Not Outlier","Outlier"]
         pd_outlier["Number of Observations"] = [not_outlier.shape[0], df.shape[0]-not_outlier.shape[0]]
         pd_outlier
```

Out[10]:

	Notes	Number of Observations
0	Not Outlier	495
1	Outlier	74

Overcoming the existance outlier

In [11]:
```
from sklearn.preprocessing import StandardScaler

features_std= StandardScaler().fit_transform(features)
features_std= pd.DataFrame(features)
features_std.columns = features.columns.tolist()
```

In [12]:
```
z = np.abs(stats.zscore(features_std._get_numeric_data()))
not_outlier = features_std[(z < 3).all(axis = 1)]
print(not_outlier.shape)
```

```
(495, 30)
```

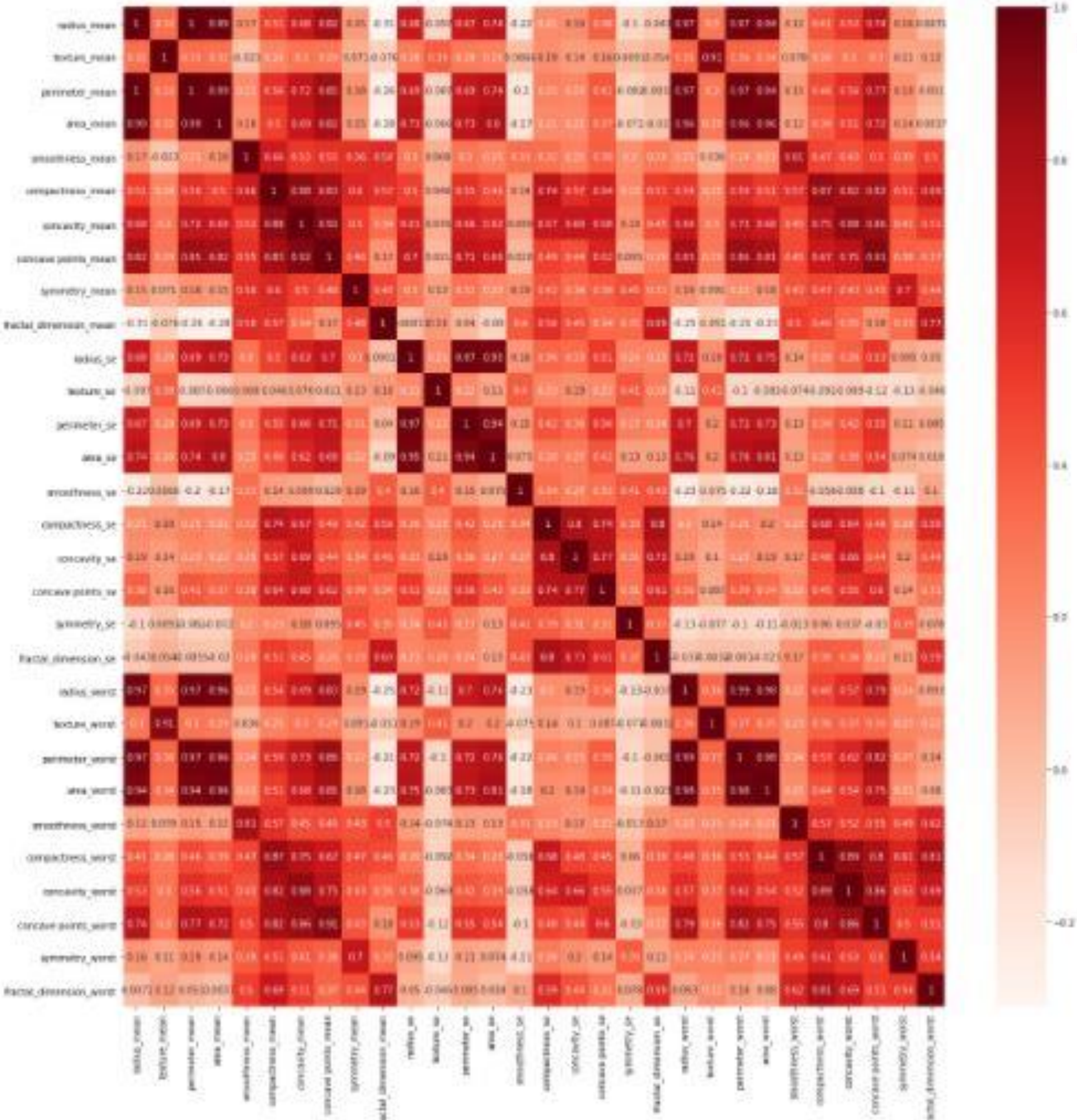
In [13]:
```
pd_outlier = pd.DataFrame([])
pd_outlier["Notes"] = ["Not Outlier","Outlier"]
pd_outlier["Number of Observations"] = [not_outlier.shape[0], features_std.shape[0]-not_outlier.shape[0]]
pd_outlier
```

Out[13]:

	Notes	Number of Observations
0	Not Outlier	495
1	Outlier	74

There is no different between standardized data and the original

3. Checking correlation matrix of each features



In [14]:
```
corr1 = features_std.corr()
fig, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(corr1,annot=True,cmap=plt.cm.Reds,ax=ax)
```

4. Doing Principal Component Analysis (PCA) for reducing the features

Calculating n-PCA number

In [15]:
```
# Calculating eigen value and eigen vector from correlation matrix
import scipy.linalg as la
eigen_value, eigen_vector = la.eig(corr1)
```

In [16]:
```
# getting ideal number of PC
eigen_value[eigen_value > 1]
```

Out[16]:
```
array([13.28160768+0.j,  5.69135461+0.j,  2.81794898+0.j,  1.98064047+0.j,
        1.64873055+0.j,  1.20735661+0.j])
```

From the results above, it is found that there are 6 values of eigenvalues that exceed the value of 1. That means the ideal PC that can be formed is 6 pieces.

In [17]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=6)
principalcomponents = pca.fit_transform(features_std) # using standardized data
new_df = pd.DataFrame(principalcomponents)
new_df.columns = ['PC1','PC2','PC3','PC4',"PC5",'PC6']
new_df
```

Out[17]:

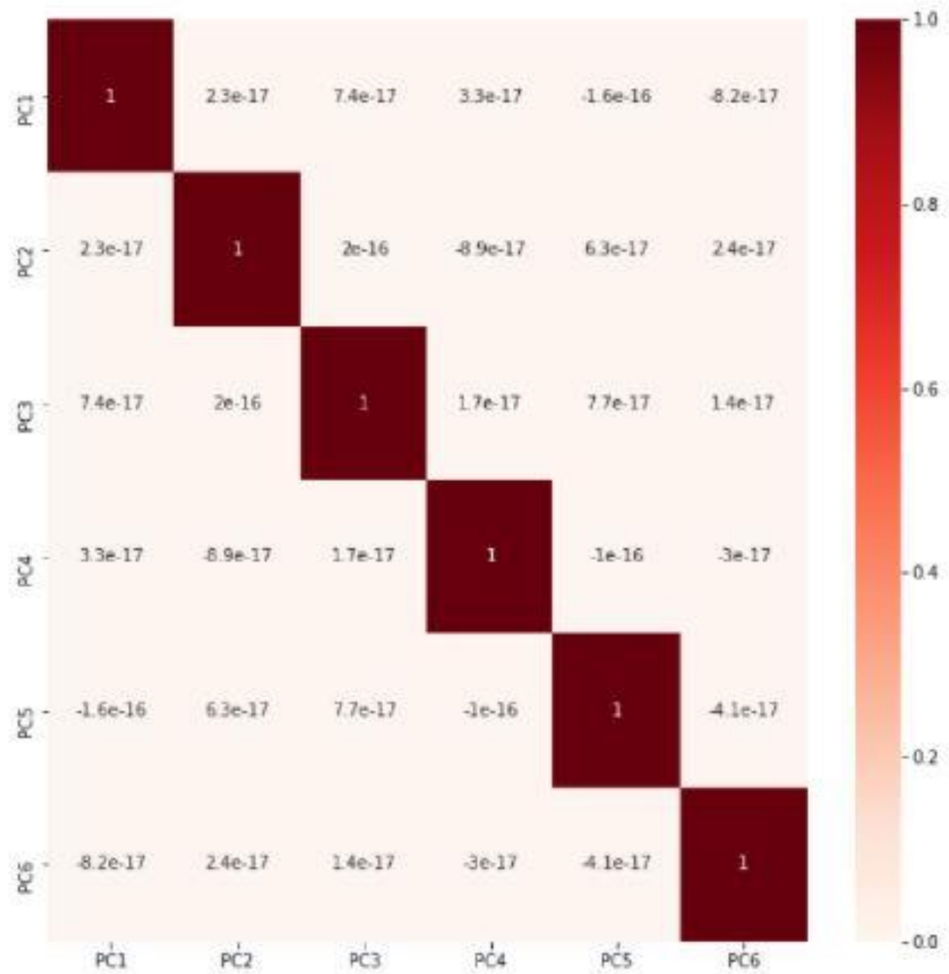
	PC1	PC2	PC3	PC4	PC5	PC6
0	1160.142574	-293.917544	48.578398	-8.711975	32.000486	1.265415
1	1269.122443	15.630182	-35.394534	17.861283	-4.334874	-0.225872
2	995.793889	39.156743	-1.709753	4.199340	-0.466529	-2.652811
3	-407.180803	-67.380320	8.672848	-11.759867	7.115461	1.299436
4	930.341180	189.340742	1.374801	8.499183	7.613289	1.021160
...	...	...	...	...	...	...
564	1414.126684	110.222492	40.065944	6.562240	-5.102856	-0.395424
565	1045.018854	77.057589	0.036669	-4.753245	-12.417863	-0.059637
566	314.501756	47.553525	-10.442407	-9.771881	-6.156213	-0.870726
567	1124.858115	34.129225	-19.742087	-23.660881	3.565133	4.086390
568	-771.527622	-88.643106	23.889032	2.547249	-14.717566	4.418123

569 rows × 6 columns

5. Rechecking correlation matrix

```
In [18]: # Checking new_df correlation features
         corr = new_df.corr()
         fig, ax = plt.subplots(figsize=(10, 10))
         sns.heatmap(corr,annot=True,cmap=plt.cm.Reds,ax=ax)

Out[18]: <AxesSubplot:>
```



Predictive Model

1. Prepare training and testing data

In [19]:
```
new_df['diagnosis'] = df['diagnosis']
new_df
```

Out[19]:

	PC1	PC2	PC3	PC4	PC5	PC6	diagnosis
0	1160.142574	-293.917544	48.578398	-8.711975	32.000486	1.265415	M
1	1269.122443	15.630182	-35.394534	17.861283	-4.334874	-0.225872	M
2	995.793889	39.156743	-1.709753	4.199340	-0.466529	-2.652811	M
3	-407.180803	-67.380320	8.672848	-11.759867	7.115461	1.299436	M
4	930.341180	189.340742	1.374801	8.499183	7.613289	1.021160	M
...	...	...	...	...	...	...	...
564	1414.126684	110.222492	40.065944	6.562240	-5.102856	-0.395424	M
565	1045.018854	77.057589	0.036669	-4.753245	-12.417863	-0.059637	M
566	314.501756	47.553525	-10.442407	-9.771881	-6.156213	-0.870726	M
567	1124.858115	34.129225	-19.742087	-23.660881	3.565133	4.086390	M
568	-771.527622	-88.643106	23.889032	2.547249	-14.717566	4.418123	B

569 rows × 7 columns

```
In [22]: from sklearn.model_selection import train_test_split

         x = new_df.iloc[:,:6]
         y = new_df.iloc[:,-1]

         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 12)
         print(x_train.shape)
         print(y_train.shape)
         print(x_test.shape)
         print(y_test.shape)

(455, 6)
(455,)
(114, 6)
(114,)
```

2. Logistic Regression

```
In [25]: from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import confusion_matrix, accuracy_score
         logreg = LogisticRegression()
         logreg.fit(x_train, y_train)

         y_predict = logreg.predict(x_test)
         cm = confusion_matrix(y_predict, y_test)
         print(cm)
         print('\n')
         print(f'akurasi Logistic Regression : {logreg.score(x_test, y_test)*100} %')

[[65  4]
 [ 1 44]]


akurasi Logistic Regression : 95.6140350877193 %
```

3. Support Vector Machine

Linear kernel

In [23]:
```
from sklearn import svm
svm1 = svm.SVC(kernel = 'linear')
```

In [24]:
```
svm1.fit(x_train,y_train)
```

Out[24]:
```
SVC(kernel='linear')
```

In [25]:
```
y_pred = svm1.predict(x_test)
```

In [26]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_pred, y_test)
print(cm)
print('\n')
print(f'akurasi SVM linear kernel : {svm1.score(x_test, y_test)*100} %')
```

```
[[65  6]
 [ 1 42]]


akurasi SVM linear kernel : 93.85964912280701 %
```

RBF Kernel

In [27]:
```
svm2 = svm.SVC(kernel = 'rbf')
svm2.fit(x_train,y_train)
y_pred2 = svm2.predict(x_test)
cm = confusion_matrix(y_pred2, y_test)
print(cm)
print('\n')
print(f'akurasi SVM rbf kernel : {svm2.score(x_test, y_test)*100} %')
```

```
[[65 12]
 [ 1 36]]


akurasi SVM rbf kernel : 88.59649122807018 %
```

Using GridSearch CV

In [28]:
```
from sklearn.model_selection import GridSearchCV #mencari parameter optimal
from sklearn.metrics import make_scorer #mencari make score

svm_tune = svm.SVC()
scorer = make_scorer(accuracy_score, greater_is_better = True)
parameters = {
    'C' : [0, 0.001, 0.01, 0.1, 0,5],
    'kernel' : ['linear', 'rbf'],
    'gamma' : [0.1, 0.5, 1, 1.1]
}

#svm_grid = GridSearchCV(svm1, parameters, n_jobs = 1, cv = 5)
svm_grid = GridSearchCV(svm_tune, param_grid = parameters, scoring = scorer, cv = 5)
svm_grid.fit(x, y)
```

In [29]:
```
print('Best Parameter :', svm_grid.best_params_)
print('Best Score     :', svm_grid.best_score_)
```

```
Best Parameter : {'C': 0.1, 'gamma': 0.1, 'kernel': 'linear'}
Best Score     : 0.9490607048594939
```

In [30]:
```
svm3 = svm.SVC(C = 0.1 ,gamma = 0.1 ,kernel = 'linear')
svm3.fit(x_train,y_train)
```

Out[30]:
```
SVC(C=0.1, gamma=0.1, kernel='linear')
```

In [31]:
```
y_pred3 = svm3.predict(x_test)
cm = confusion_matrix(y_pred3, y_test)
print(cm)
print('\n')
print(f'akurasi SVM with tuning parameter : {svm3.score(x_test, y_test)*100} %')
```

```
[[65  6]
 [ 1 42]]


akurasi SVM with tuning parameter : 93.85964912280701 %
```

4. Random Forest

```
In [32]: from sklearn.ensemble import RandomForestClassifier
         rf = RandomForestClassifier()
         rf.fit(x_train,y_train)
         y_pred4 = rf.predict(x_test)
         cm = confusion_matrix(y_pred4, y_test)
         print(cm)
         print('\n')
         print(' :', )
         print(f'akurasi Random Forest: {rf.score(x_test, y_test)*100} %')
```

```
[[66  7]
 [ 0 41]]


 :
akurasi Random Forest: 93.85964912280701 %
```

5. K Nearest Neighbor

```
In [33]: from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score

         model = KNeighborsClassifier(n_neighbors = 5, weights = 'uniform', p = 2)
         model.fit(x_train, y_train)
         y_pred5 = model.predict(x_test)

         cm = confusion_matrix(y_pred5, y_test)
         accuracy4 = accuracy_score(y_pred5, y_test)

         print(cm)
         print('\n') # enter
         print(f'akurasi K-NN : {accuracy4*100} %')
```

```
[[64 10]
 [ 2 38]]


akurasi K-NN : 89.47368421052632 %
```

Using GridSearch CV

In [34]:
```
from sklearn.model_selection import GridSearchCV #mencari parameter optimal
from sklearn.metrics import make_scorer #mencari make score

model = KNeighborsClassifier()
scorer = make_scorer(accuracy_score, greater_is_better = True)
parameters = [{'n_neighbors': [3,5,7,9,11,13,15,17],
              'weights' : ['uniform', 'distance'],
              'p' : [1,2]}]

model_gscv = GridSearchCV(model, param_grid = parameters, scoring = scorer, cv = 5)
model_gscv.fit(x, y)
print('Best Parameter :', model_gscv.best_params_)
print('Best Score     :', model_gscv.best_score_)
```

```
Best Parameter : {'n_neighbors': 11, 'p': 1, 'weights': 'uniform'}
Best Score     : 0.9402577239559076
```

In [36]:
```
model = KNeighborsClassifier(n_neighbors = 11, weights = 'uniform', p = 1)
model.fit(x_train, y_train)
y_pred6 = model.predict(x_test)

cm = confusion_matrix(y_pred6, y_test)
accuracy5 = accuracy_score(y_pred6, y_test)

print(cm)
print('\n') # enter
print(f'akurasi K-NN with tuning parametes : {accuracy5*100} %')
```

```
[[66  9]
 [ 0 39]]


akurasi K-NN with tuning parametes : 92.10526315789474 %
```

6. Naive Bayes

In [37]:
```
bayes_df = new_df
```

In [38]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
bayes_df['diagnosis'] = le.fit_transform(bayes_df["diagnosis"])
bayes_df
```

Out[38]:

	PC1	PC2	PC3	PC4	PC5	PC6	diagnosis
0	1160.142574	-293.917544	48.578398	-8.711975	32.000486	1.265415	1
1	1269.122443	15.630182	-35.394534	17.861283	-4.334874	-0.225872	1
2	995.793889	39.156743	-1.709753	4.199340	-0.466529	-2.652811	1
3	-407.180803	-67.380320	8.672848	-11.759867	7.115461	1.299436	1
4	930.341180	189.340742	1.374801	8.499183	7.613289	1.021160	1
...	...	...	...	...	...	...	...
564	1414.126684	110.222492	40.065944	6.562240	-5.102856	-0.395424	1
565	1045.018854	77.057589	0.036669	-4.753245	-12.417863	-0.059637	1
566	314.501756	47.553525	-10.442407	-9.771881	-8.156213	-0.870726	1
567	1124.858115	34.129225	-19.742087	-23.660881	3.565133	4.086390	1
568	-771.527622	-88.643106	23.889032	2.547249	-14.717566	4.418123	0

569 rows × 7 columns

Notes :

```
1 >> M >> Malignant (Ganas)

0 >> B >> Benign (Jinak)
```

In [39]:
```
x2 = bayes_df.iloc[:,:6]
y2 = bayes_df.iloc[:,-1]

x_train2, x_test2, y_train2, y_test2 = train_test_split(x2, y2, test_size = 0.2, random_state = 12)
print(x_train2.shape)
print(y_train2.shape)
print(x_test2.shape)
print(y_test2.shape)
```

```
(455, 6)
(455,)
(114, 6)
(114,)
```

In [40]:
```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(x_train2,y_train2)
```

Out[40]:
```
GaussianNB()
```

In [41]:
```
y_pred7 = model.predict(x_test2)
cm = confusion_matrix(y_pred7, y_test2)
accuracy6 = accuracy_score(y_pred7, y_test2)

print(cm)
print('\n') # enter
print(f'akurasi Naive-Bayes : {accuracy6*100} %')
```

```
[[62 13]
 [ 4 35]]


akurasi Naive-Bayes : 85.08771929824562 %
```

Summary

```
--------------------------------------------------------------------------------
akurasi Logistic Regression : 95.6140350877193 %

              precision    recall  f1-score   support

           B       0.94      0.98      0.96        66
           M       0.98      0.92      0.95        48

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

--------------------------------------------------------------------------------
akurasi SVM linear kernel : 93.85964912280701 %

              precision    recall  f1-score   support

           B       0.92      0.98      0.95        66
           M       0.98      0.88      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114

--------------------------------------------------------------------------------
akurasi SVM rbf kernel : 88.59649122807018 %

              precision    recall  f1-score   support

           B       0.84      0.98      0.91        66
           M       0.97      0.75      0.85        48

    accuracy                           0.89       114
   macro avg       0.91      0.87      0.88       114
weighted avg       0.90      0.89      0.88       114

--------------------------------------------------------------------------------
akurasi SVM with tuning parameter : 93.85964912280701 %

              precision    recall  f1-score   support

           B       0.92      0.98      0.95        66
           M       0.98      0.88      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114

--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
akurasi Random Forest: 93.85964912280701 %

              precision    recall  f1-score   support

           B       0.90      1.00      0.95        66
           M       1.00      0.85      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114

--------------------------------------------------------------------------------
akurasi K-NN : 89.47368421052632 %

              precision    recall  f1-score   support

           B       0.86      0.97      0.91        66
           M       0.95      0.79      0.86        48

    accuracy                           0.89       114
   macro avg       0.91      0.88      0.89       114
weighted avg       0.90      0.89      0.89       114

--------------------------------------------------------------------------------
akurasi K-NN with tuning parametes : 92.10526315789474 %

              precision    recall  f1-score   support

           B       0.88      1.00      0.94        66
           M       1.00      0.81      0.90        48

    accuracy                           0.92       114
   macro avg       0.94      0.91      0.92       114
weighted avg       0.93      0.92      0.92       114

--------------------------------------------------------------------------------
akurasi Naive-Bayes : 85.08771929824562 %

              precision    recall  f1-score   support

           0       0.83      0.94      0.88        66
           1       0.90      0.73      0.80        48

    accuracy                           0.85       114
   macro avg       0.86      0.83      0.84       114
weighted avg       0.86      0.85      0.85       114

--------------------------------------------------------------------------------
```

Thank You …