Entrée [1]:

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
cancer = datasets.load_breast_cancer()
print("Shape: \n\n", cancer.data.shape)
print("Features: \n\n", cancer.feature_names)
print("Target: \n\n",cancer.target_names)
```

```
Shape: 

 (569, 30)
Features: 

 ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Target: 

 ['malignant' 'benign']
```

Entrée [2]:

```
import pandas as pd

cancer_df = pd.DataFrame(data=cancer.data, columns=cancer.feature_names)
cancer_df
```

Out[2]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	w
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	20
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	19
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	17
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	5
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	15
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	20
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	17
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	18
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	2

569 rows × 30 columns

Entrée [3]:

```
import numpy as np

target_np = np.array(cancer.target_names)
target_np
print(cancer.target)
```

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

Entrée [4]:

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

X = cancer.data
y = np.where(cancer.target == 0, 1, 0)

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,random_state=1) # 70% training and 30% test
```

Entrée [24]:

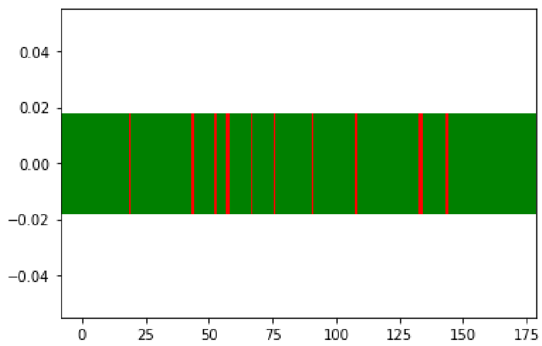
```
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.LinearSVC(max_iter = 10000) # Linear SVC

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
```

```
/home/amine/snap/jupyter/common/lib/python3.7/site-packages/sklearn/svm/_base.py:1208: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  ConvergenceWarning,
```

Entrée [6]:

```
import matplotlib.pyplot as plt

print("Données prédites:\n")
print(y_pred)
print("\n\n")
print("Données réelles:\n")
print(y_test)
print("\n\n")
print("Comparaison:\n")
comparaison_np = y_pred == y_test

colors = np.where(comparaison_np, 'green', 'red')

fig, ax = plt.subplots()
ax.scatter(np.arange(comparaison_np.size), np.zeros_like(comparaison_np), c=colors, s=5000, marker='s')
plt.show()
```

```
Données prédites:

[0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1
 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1
 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]



Données réelles:

[0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1
 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0]



Comparaison:
```



Entrée [7]:

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Accuracy tells you how often the model is correct
accuracy = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy, "\n\n")

# Precision tells you how often your positive predictions are correct
precision = metrics.precision_score(y_test, y_pred)
print("Precision:", precision, "\n\n")

# Recall tells you how often you catch all positive cases
recall = metrics.recall_score(y_test, y_pred)
print("Recall:", recall)

# Metrics data DataFrame
metricsSVC = [["SVC", accuracy, precision, recall]]
metrics_df = pd.DataFrame(metricsSVC, columns=["Model", "Accuracy", "Precision", "Recall"])
metrics_df
```

```
Accuracy: 0.9239766081871345


Precision: 0.9464285714285714


Recall: 0.8412698412698413
```

Out[7]:

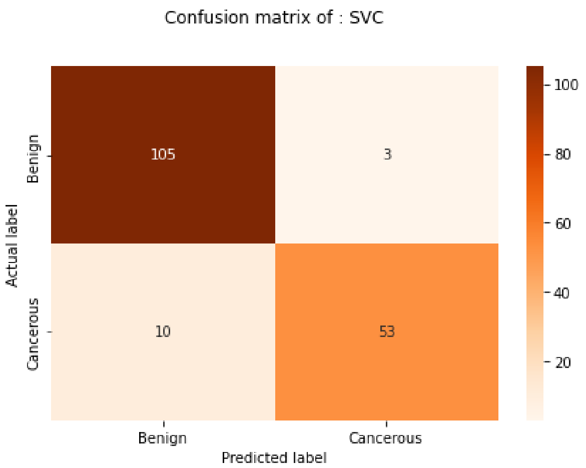
	Model	Accuracy	Precision	Recall
0	SVC	0.923977	0.946429	0.84127

Entrée [8]:

```
import seaborn as sns
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="Oranges" ,fmt='g')
class_names = ['Benign','Cancerous'] # name  of classes
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)
ax.xaxis.set_label_position("bottom")
plt.tight_layout()
plt.title('Confusion matrix of : SVC', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[8]:

```
Text(0.5, 15.0, 'Predicted label')
```



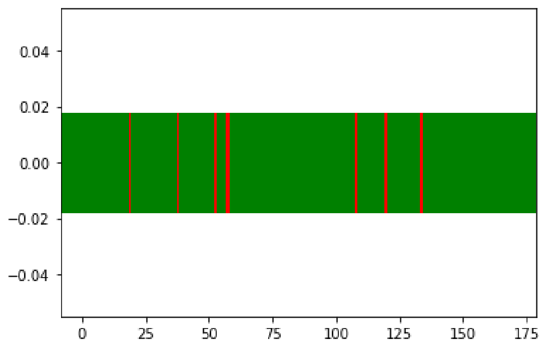
Entrée [9]:

```
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression(max_iter = 10000, random_state=1)

# fit the model with data
logreg.fit(X_train, y_train)

y_pred_log = logreg.predict(X_test)
```

Entrée [10]:

```
print("Données prédites:\n")
print(y_pred)
print("\n\n")
print("Données réelles:\n")
print(y_test)
print("\n\n")
print("Comparaison:\n")
comparaison_np_log = y_pred_log == y_test

colors_log = np.where(comparaison_np_log, 'green', 'red')

fig, ax = plt.subplots()
ax.scatter(np.arange(comparaison_np_log.size), np.zeros_like(comparaison_np_log), c=colors_log, s=5000, marker='s')
plt.show()
```

```
Données prédites:

[0 0 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1
 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1
 0 1 0 0 0 1 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0]



Données réelles:

[0 1 0 1 1 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1
 0 1 0 0 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1 1
 0 1 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 1 1 0 1 0 1 0 0
 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 0 0]



Comparaison:
```



Entrée [11]:

```
# Accuracy tells you how often the model is correct
accuracy = metrics.accuracy_score(y_test, y_pred_log)
print("Accuracy:", accuracy, "\n\n")

# Precision tells you how often your positive predictions are correct
precision = metrics.precision_score(y_test, y_pred_log)
print("Precision:", precision, "\n\n")

# Recall tells you how often you catch all positive cases
recall = metrics.recall_score(y_test, y_pred_log)
print("Recall:", recall)

# Metrics data DataFrame
metricsLogReg = [["LogReg", accuracy, precision, recall]]
metrics_df_log = pd.DataFrame(metricsLogReg, columns=["Model", "Accuracy", "Precision", "Recall"])
metrics_df = metrics_df.append(metrics_df_log, ignore_index = True)
metrics_df
```

```
Accuracy: 0.9473684210526315


Precision: 0.95


Recall: 0.9047619047619048
```

Out[11]:

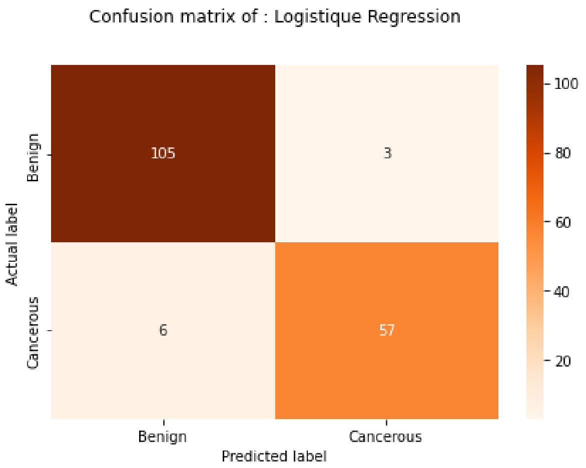
	Model	Accuracy	Precision	Recall
0	SVC	0.923977	0.946429	0.841270
1	LogReg	0.947368	0.950000	0.904762

Entrée [12]:

```
cnf_matrix = metrics.confusion_matrix(y_test, y_pred_log)

# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="Oranges" ,fmt='g')
class_names = ['Benign','Cancerous'] # name  of classes
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks + 0.5, class_names)
plt.yticks(tick_marks + 0.5, class_names)
ax.xaxis.set_label_position("bottom")
plt.tight_layout()
plt.title('Confusion matrix of : Logistique Regression', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

Out[12]:

```
Text(0.5, 15.0, 'Predicted label')
```



Entrée [13]:

```
###############################
```

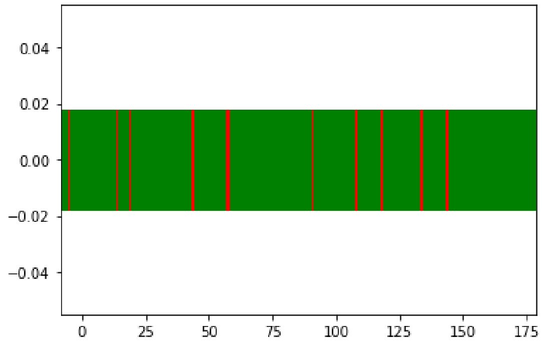
Entrée [14]:

```
#Import knearest neighbors Classifier model
from sklearn.neighbors import KNeighborsClassifier
n_neighbors = [5, 7, 10]
knn = list(range(3))
predictions = list(range(3))

for index, value in enumerate(n_neighbors):
    #Create KNN Classifier
    knn[index] = KNeighborsClassifier(n_neighbors=value)
    #Train the model using the training sets
    knn[index].fit(X_train, y_train)
    #Predict the response for test dataset
    predictions[index] = knn[index].predict(X_test)
```

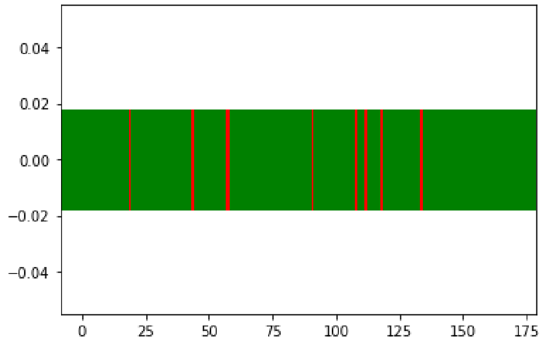
Entrée [15]:

```
for i in range(3):
    print("Comparaison:\n")
    comparaison_np_knn = predictions[i] == y_test

    colors_log = np.where(comparaison_np_knn, 'green', 'red')

    fig, ax = plt.subplots()
    ax.scatter(np.arange(comparaison_np_knn.size), np.zeros_like(comparaison_np_knn), c=colors_log, s=5000, marker='s'
    plt.show()
```

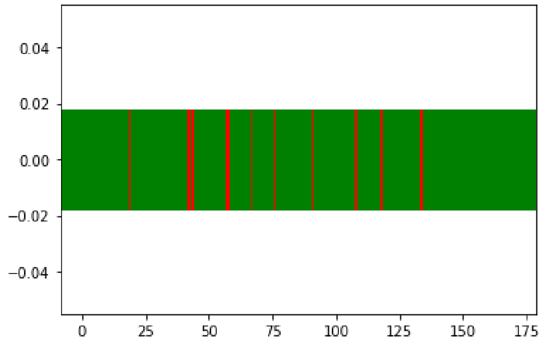
```
Comparaison:
```




```
Comparaison:
```




```
Comparaison:
```



Entrée [16]:

```
for i in range(3):
    # Accuracy tells you how often the model is correct
    accuracy = metrics.accuracy_score(y_test, predictions[i])
    print("Accuracy:", accuracy, "\n\n")

    # Precision tells you how often your positive predictions are correct
    precision = metrics.precision_score(y_test, predictions[i])
    print("Precision:", precision, "\n\n")

    # Recall tells you how often you catch all positive cases
    recall = metrics.recall_score(y_test, predictions[i])
    print("Recall:", recall)

    # Metrics data DataFrame
    metricsknn = [["KNN" + str(n_neighbors[i]), accuracy, precision, recall]]
    metrics_df_knn = pd.DataFrame(metricsknn, columns=["Model", "Accuracy", "Precision", "Recall"])
    metrics_df = metrics_df.append(metrics_df_knn, ignore_index = True)
metrics_df
```

```
Accuracy: 0.9298245614035088


Precision: 0.9180327868852459


Recall: 0.8888888888888888
Accuracy: 0.935672514619883


Precision: 0.9193548387096774


Recall: 0.9047619047619048
Accuracy: 0.9298245614035088


Precision: 0.9473684210526315


Recall: 0.8571428571428571
```

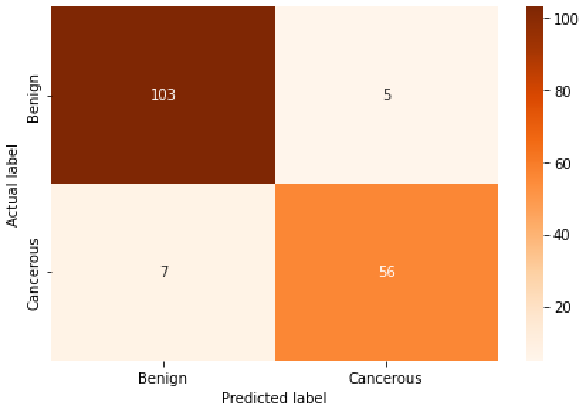
Out[16]:

	Model	Accuracy	Precision	Recall
0	SVC	0.923977	0.946429	0.841270
1	LogReg	0.947368	0.950000	0.904762
2	KNN5	0.929825	0.918033	0.888889
3	KNN7	0.935673	0.919355	0.904762
4	KNN10	0.929825	0.947368	0.857143

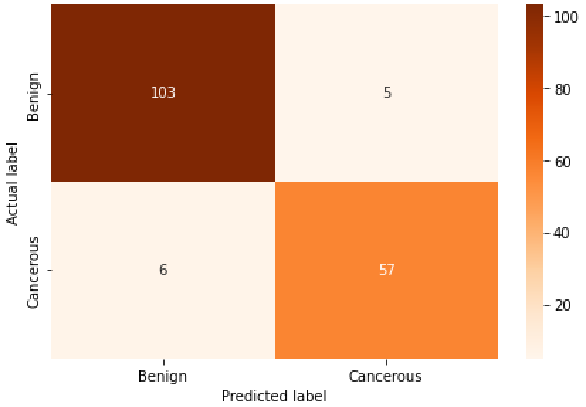
Entrée [17]:

```
for i in range(3):
    cnf_matrix = metrics.confusion_matrix(y_test, predictions[i])

    # create heatmap
    plt.figure(i)
    sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="Oranges" ,fmt='g')
    class_names = ['Benign','Cancerous'] # name  of classes
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks + 0.5, class_names)
    plt.yticks(tick_marks + 0.5, class_names)
    ax.xaxis.set_label_position("bottom")
    plt.tight_layout()
    plt.title('Confusion matrix of : ' + str(knn[i]), y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
```

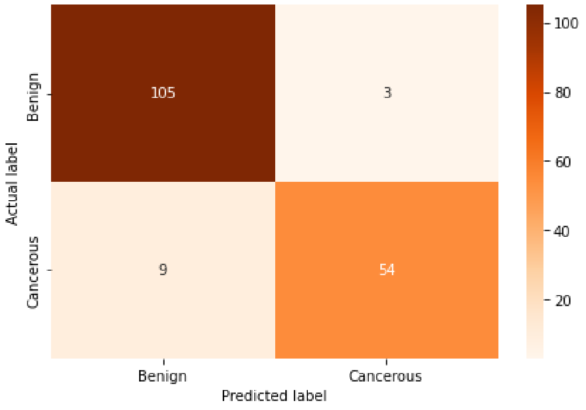
Confusion matrix of : KNeighborsClassifier()



Confusion matrix of : KNeighborsClassifier(n_neighbors=7)



Confusion matrix of : KNeighborsClassifier(n_neighbors=10)



Entrée [18]:

```
# Best models by use:

print("Le model avec les predictions les plus précises est : " + str(metrics_df.loc[metrics_df["Accuracy"].idxmax()][0
print("Le model avec le plus grand taux de vrais positifs est : " + str(metrics_df.loc[metrics_df["Precision"].idxmax(
print("Le model avec le plus grand taux de decouverte de cas positifs est : " + str(metrics_df.loc[metrics_df["Recall"]
```

```
Le model avec les predictions les plus précises est : LogReg avec un taux de 0.9473684210526315
Le model avec le plus grand taux de vrais positifs est : LogReg avec un taux de 0.95
Le model avec le plus grand taux de decouverte de cas positifs est : LogReg avec un taux de 0.90476190476
19048
```

Entrée [19]:

```
# Worst models by use:

print("Le model avec les predictions les moins précises est : " + str(metrics_df.loc[metrics_df["Accuracy"].idxmin()][
print("Le model avec le plus petit taux de vrais positifs est : " + str(metrics_df.loc[metrics_df["Precision"].idxmin(
print("Le model avec le plus petit taux de decouverte de cas positifs est : " + str(metrics_df.loc[metrics_df["Recall"]
```

```
Le model avec les predictions les moins précises est : SVC avec un taux de 0.9239766081871345
Le model avec le plus petit taux de vrais positifs est : KNN5 avec un taux de 0.9180327868852459
Le model avec le plus petit taux de decouverte de cas positifs est : SVC avec un taux de 0.84126984126984
13
```

Entrée [ ]:

```

```