**BREAST CANCER ACCURACY PREDICTION** **USING** **MACHINE LEARNING MODELS**

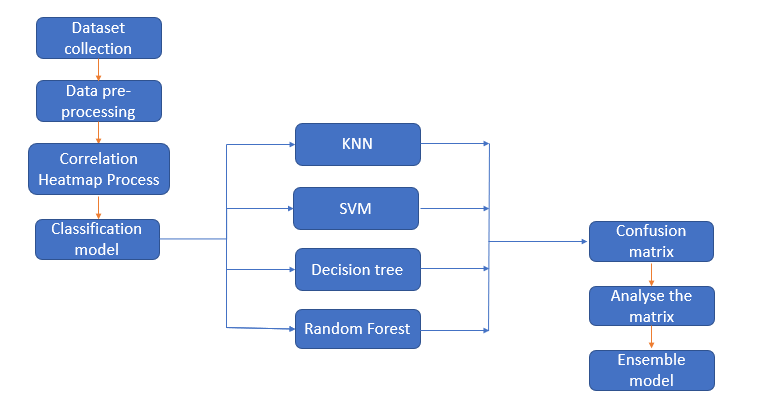
### Aim

To compare classification models in machine learning algorithms for predicting Breast cancer with good accuracy

#### Objectives

* To Prepare and Pre-process the dataset
* To Visualization the dataset
* To Perform Correlation Heatmap process
* To implement Different Classification model
* Analysing the data using performance metrics
* Evaluate using Ensemble model

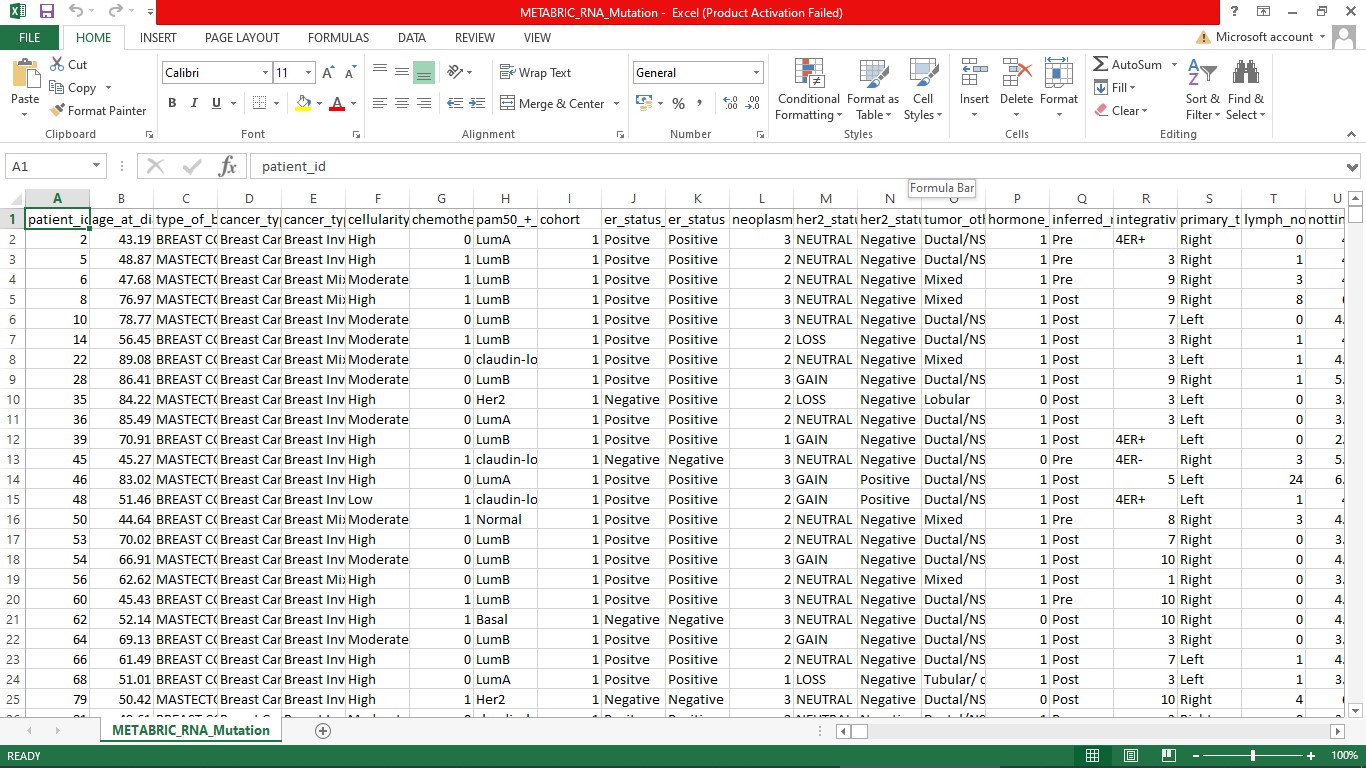
**Method:**

****

#### Dataset

The Molecular Taxonomy of Breast Cancer International Consortium (METABRIC) database is a Canada-UK Project which contains targeted sequencing data of 1,980 primary breast cancer samples. Clinical and genomic data was downloaded from cBioPortal.

The dataset was collected by Professor Carlos Caldas from Cambridge Research Institute and Professor Sam Aparicio from the British Columbia Cancer Centre in Canada and published on Nature Communications.



**Dataset**

### Source code

**Data Pre-processing & Correlation Heatmap Process**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

data pd.read\_csv("/content/METABRIC\_RNA\_Mutation.cs")

X = data.iloc[:, 28:517].values

y = data.iloc[:, 4].values

from sklearn.preprocessing import LabelEncoder

label\_encoder\_y= LabelEncoder()

y= label\_encoder\_y.fit\_transform(y)

**Correlation Heatmap for Gene Expression Profiles**

df = pd.DataFrame(X, columns=data.columns[28:517])

df['cancer\_type\_detailed'] = y

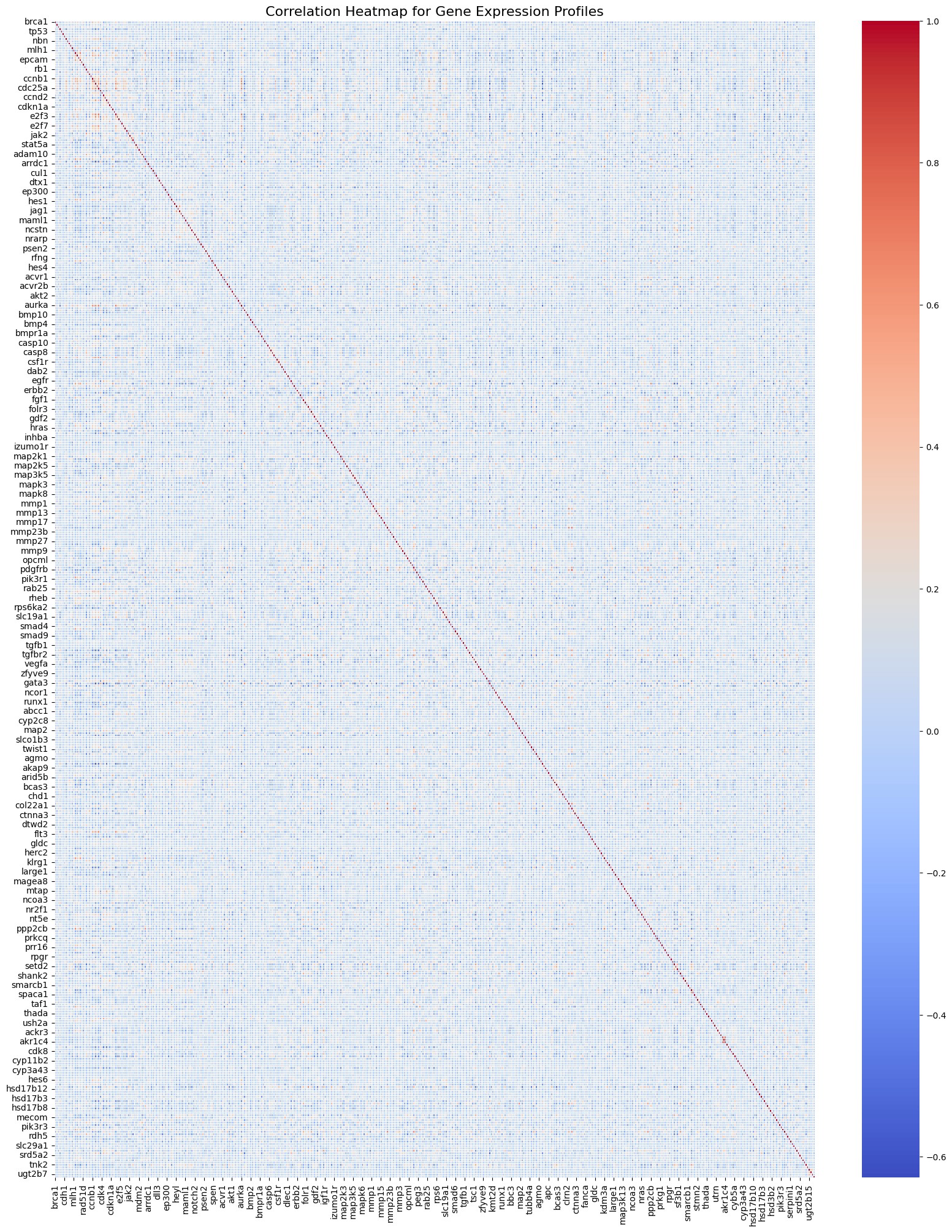
correlation\_matrix = df.corr()

plt.figure(figsize=(20,24.5))

sns.heatmap(correlation\_matrix, annot=False, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap for Gene Expression Profiles', fontsize=16)

plt.show()



**Correlation Heatmap for Gene Expression Profiles**

correlation\_with\_target =

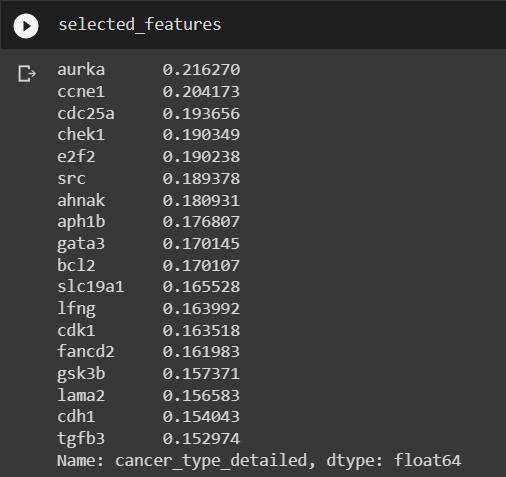
correlation\_matrix['cancer\_type\_detailed'].abs().sort\_values(ascending=False)

selected\_features = correlation\_with\_target[1:]

correlation\_threshold = 0.15

selected\_features = selected\_features[selected\_features > correlation\_threshold**]**

print(selected\_features)



### Selected Gene Expression Profiles

**Correlation Heatmap for Clinical Features**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_breast\_cancer

data=pd.read\_csv("/content/METABRIC\_RNA\_Mutation.csv") X= data.loc[:, "cellularity":"death\_from\_cancer"]

y = data.iloc[:, 4].values

columns\_to\_encode = X.columns[:len(X.columns)]

label\_encoder = LabelEncoder()

for column in columns\_to\_encode:

X[column] =label\_encoder.fit\_transform(X[column]) from sklearn.preprocessing import LabelEncoder label\_encoder\_y= LabelEncoder()

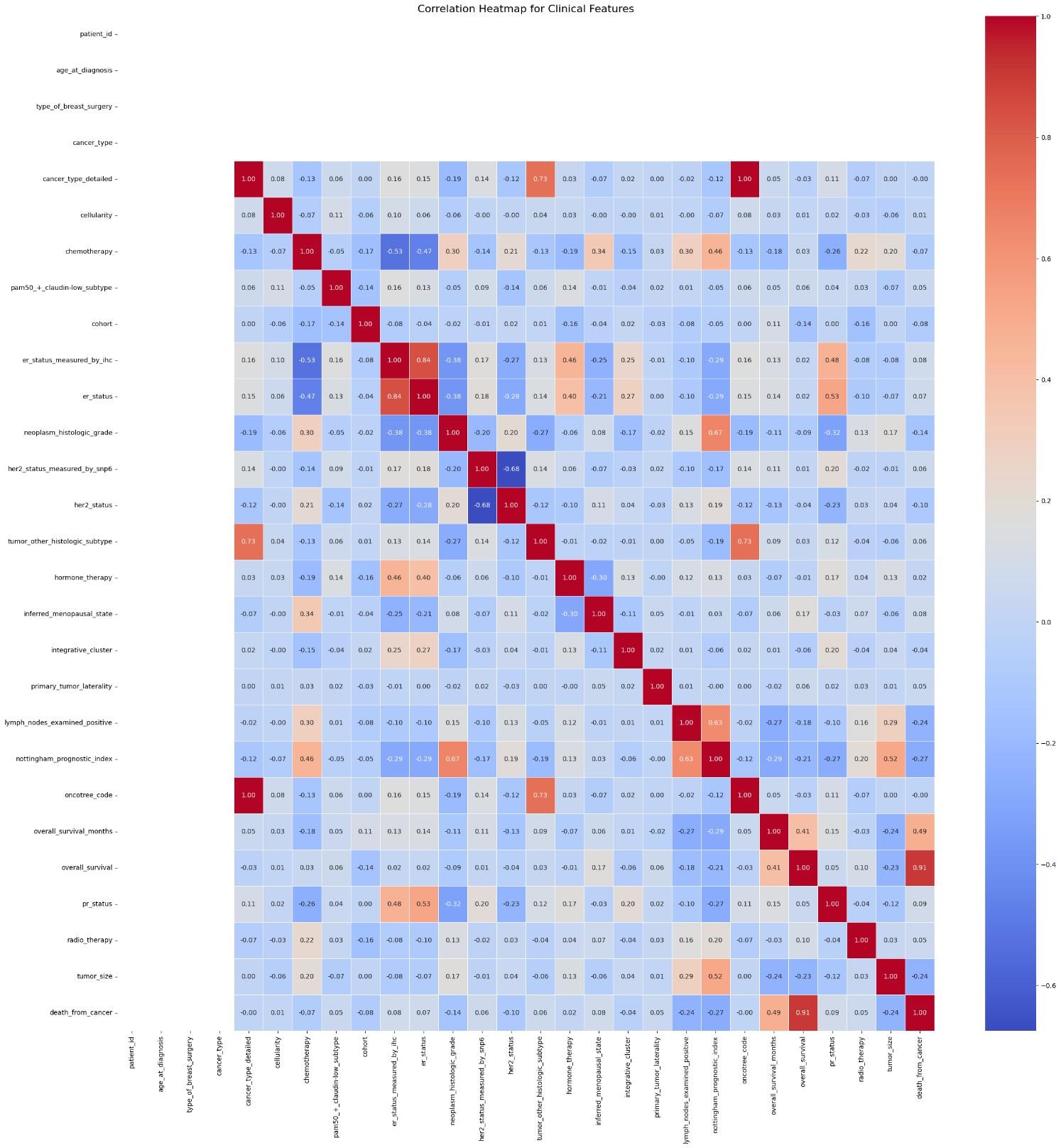
y= label\_encoder\_y.fit\_transform(y)

df = pd.DataFrame(X, columns=data.columns[:28]) df['cancer\_type\_detailed'] = y correlation\_matrix = df.corr() plt.figure(figsize=(28, 28))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)

plt.title('Correlation Heatmap for Clinical Features', fontsize=16)

plt.show()



### Correlation Heatmap for Clinical Features output

correlation\_with\_target = correlation\_matrix['cancer\_type\_detailed'].abs().sort\_values(ascending=False)

selected\_clinical\_features = correlation\_with\_target.drop('cancer\_type\_detailed').index

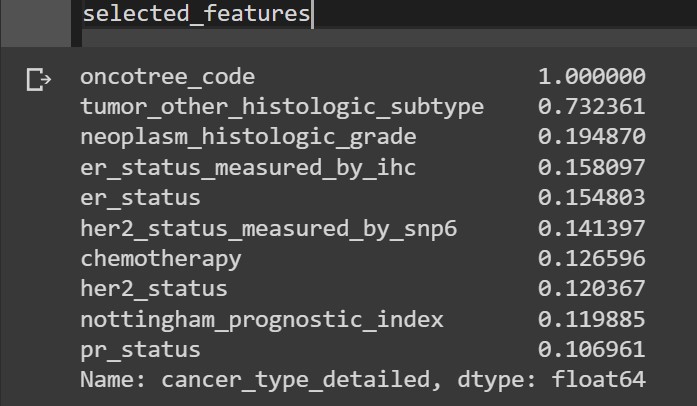
correlation\_with\_target = correlation\_matrix['cancer\_type\_detailed'].abs().sort\_values(ascending=False)

selected\_features = correlation\_with\_target[1:]

correlation\_threshold = 0.1

selected\_features = selected\_features[selected\_features > correlation\_threshold]

selected\_features



### Selected Clinical Features

#### Data Pre-Processing & Mechine learning models

#### import numpy as np

#### import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix y= data[['cancer\_type\_detailed']]

X=data[['oncotree\_code','tumor\_other\_histologic\_subtype','neoplasm\_histologic\_grade','er\_status\_measured\_by\_ihc','er\_status','her2\_status\_measured\_by\_snp6','chemotherapy','her2\_status','nottingham\_prognostic\_index','pr\_status','aurka','ccne1','cdc25a','chek1','e2f2','src','ahnak','aph1b','gata3','bcl2','slc19a1','lfng','cdk1','fancd2','gsk3b','lama2' ,'cdh1','tgfb3']]

columns\_to\_encode =X.columns[:len(X.columns)]

label\_encoder = LabelEncoder()

for column in columns\_to\_encode:

X[column] = label\_encoder.fit\_transform(X[column])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) scaler = StandardScaler()

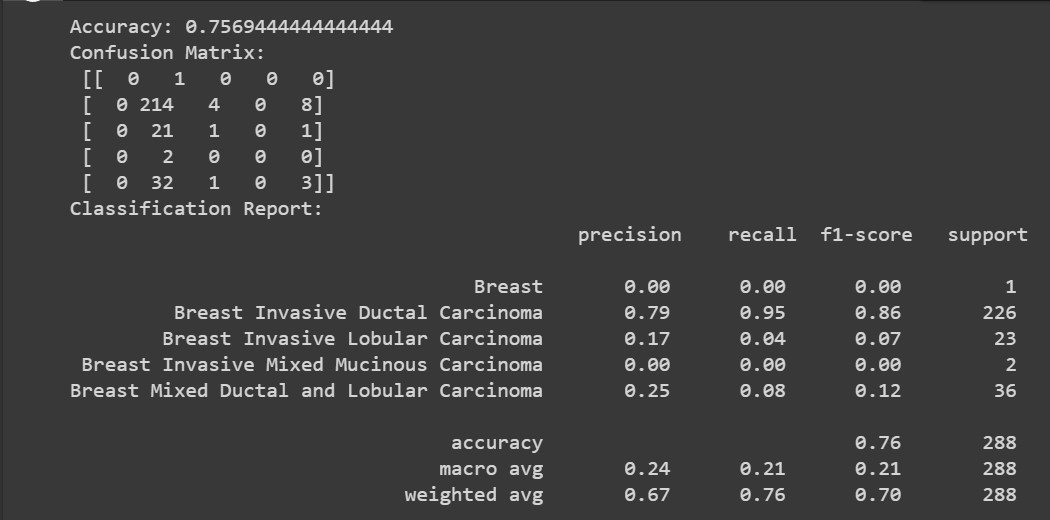
X\_train\_std = scaler.fit\_transform(X\_train)

X\_test\_std = scaler.transform(X\_test)

**KNeighbor (K-NN)**

from sklearn.neighbors import KNeighborsClassifier classifier= KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2 ) classifier.fit(X\_train, y\_train)

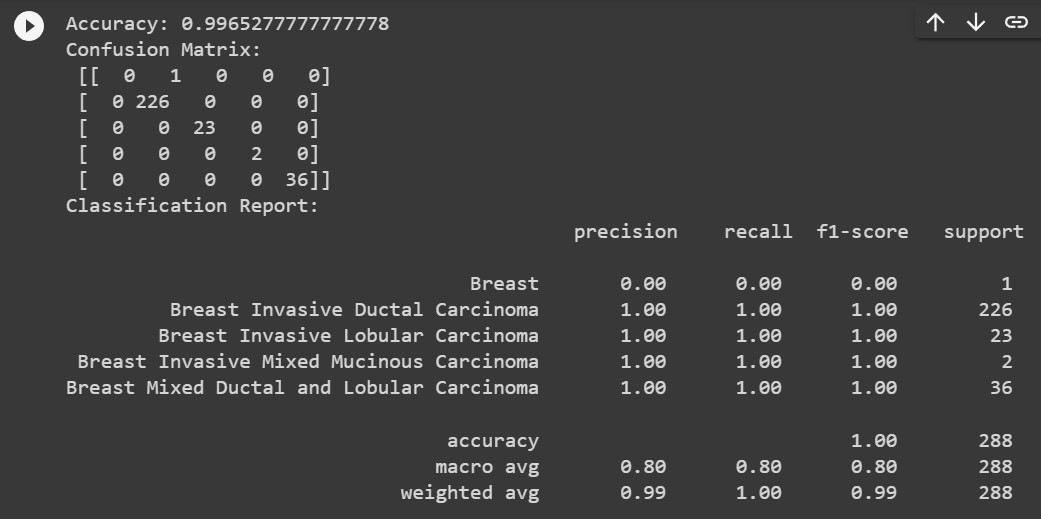
y\_pred= classifier.predict(X\_test) cm= confusion\_matrix(y\_test, y\_pred) K = accuracy\_score(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", K ) print("Confusion Matrix:\n", cm) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report for KNeighbor (K-NN)

**Support Vector Machine (SVM)**

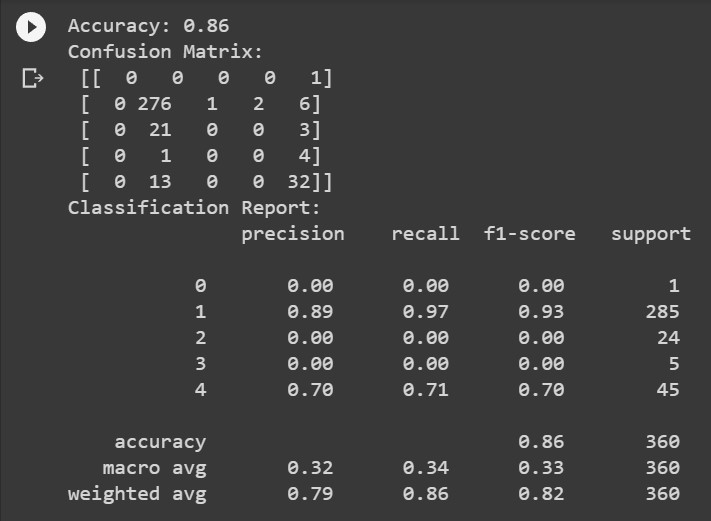
svm\_model = SVC(kernel='linear', random\_state=42) svm\_model.fit(X\_train\_std, y\_train)# Make predictions on the test set y\_pred = svm\_model.predict(X\_test\_std)# Evaluate the model's performance accuracy = accuracy\_score(y\_test, y\_pred) confusion = confusion\_matrix(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", accuracy) print("Confusion Matrix:\n", confusion) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report for Support Vector Machine (SVM)

**Decision Tree**

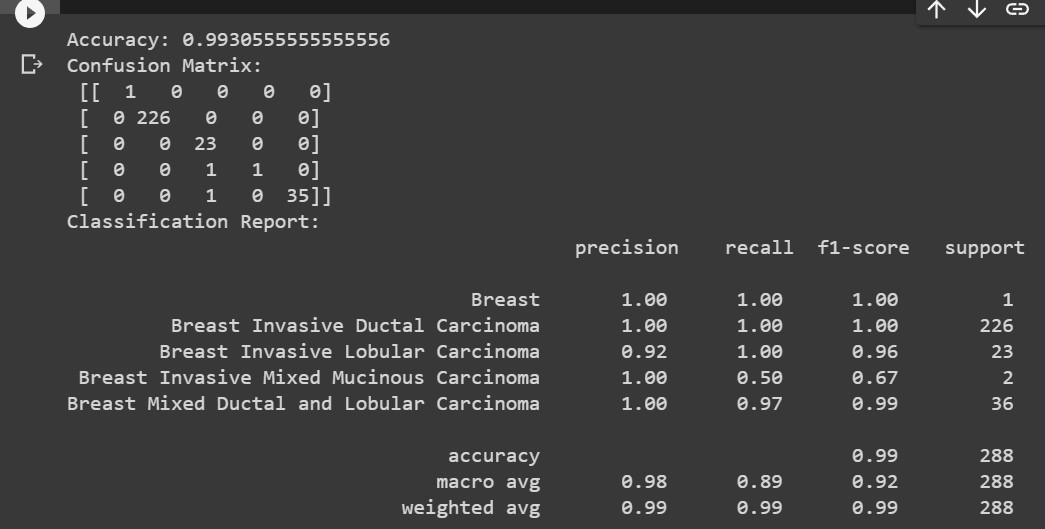
from sklearn.tree import DecisionTreeClassifier classifier= DecisionTreeClassifier(criterion='entropy', random\_state=0) classifier.fit(X\_train, y\_train)#Predicting the test set result y\_pred= classifier.predict(X\_test) cm= confusion\_matrix(y\_test, y\_pred) d = accuracy\_score(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", d ) print("Confusion Matrix:\n", cm) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report for Decision Tree

**Random Forest**

from sklearn.ensemble import RandomForestClassifier classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy") classifier.fit(X\_train, y\_train)#Predicting the test set result y\_pred= classifier.predict(X\_test) cm= confusion\_matrix(y\_test, y\_pred) r= accuracy\_score(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", r ) print("Confusion Matrix:\n", cm) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report for Random Forest

#### Different ensemble models

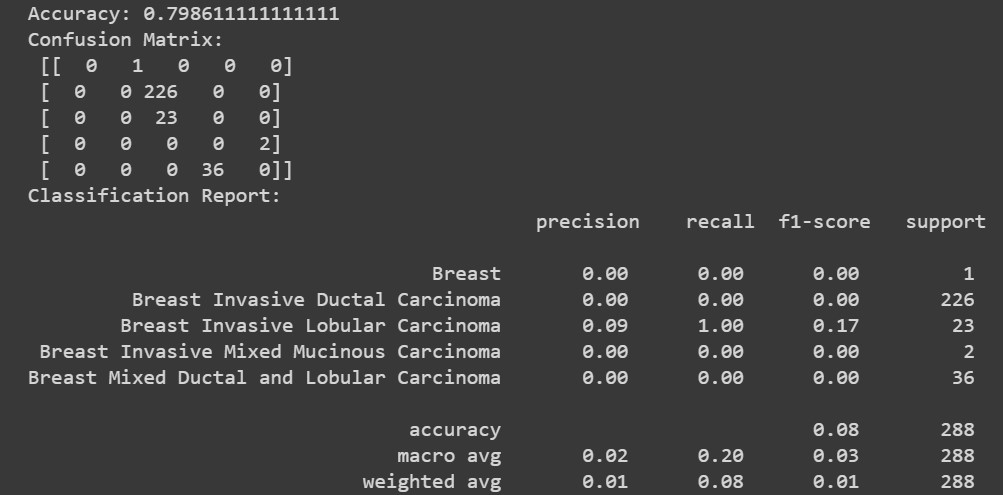
**Gradient Boosting**

from sklearn.ensemble import GradientBoostingClassifier

gradient\_boosting\_model = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

gradient\_boosting\_model.fit(X\_train\_std, y\_train)# Make predictions on the test set

y\_pred = gradient\_boosting\_model.predict(X\_test)# Evaluate the model's performance g = accuracy\_score(y\_test, y\_pred) confusion = confusion\_matrix(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", g) print("Confusion Matrix:\n", confusion) print("Classification Report:\n", classification\_rep)



### Gradient Boosting

#### Stacking Classifier

from sklearn.ensemble import RandomForestClassifier,

GradientBoostingClassifier from sklearn.svm import SVC

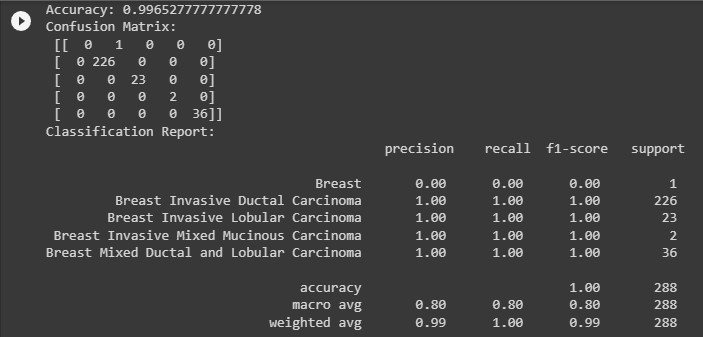
from sklearn.ensemble import StackingClassifier from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import cross\_val\_predict random\_forest = RandomForestClassifier(random\_state=42) svm = SVC(probability=True, random\_state=42) logistic\_regression = LogisticRegression(random\_state=42) stacked\_model = StackingClassifier(

estimators=[('rf', random\_forest), ('svm', svm), ('lr', logistic\_regression)], final\_estimator=LogisticRegression(),

)

stacked\_model.fit(X\_train, y\_train) y\_pred\_stacked = stacked\_model.predict(X\_test) st = accuracy\_score(y\_test, y\_pred\_stacked) confusion = confusion\_matrix(y\_test, y\_pred\_stacked) classification\_rep = classification\_report(y\_test, y\_pred\_stacked) print("Accuracy:", st)

print("Confusion Matrix:\n", confusion) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report

### for Stacking Classifier

**Voting classifier**

from sklearn.ensemble import VotingClassifier

# Initialize the base models model1 = DecisionTreeClassifier() model2 = LogisticRegression() model3 = SVC(probability=True) model4 = GaussianNB() model5 = KNeighborsClassifier() model6 = RandomForestClassifier() ensemble\_model = VotingClassifier( estimators=[

('dt', model1),

('lr', model2),

('svc', model3),

('gnb', model4),

('knn', model5),

('rf', model6)

],

voting='hard'

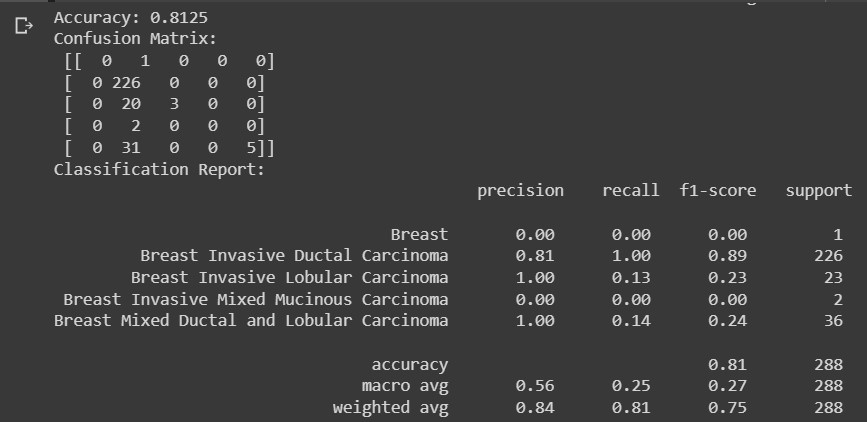
)

ensemble\_model.fit(X\_train, y\_train) y\_pred = ensemble\_model.predict(X\_test)

print("-------------------------------------------------------------------voting model------------

------------------------------------------------------------")

e = accuracy\_score(y\_test, y\_pred) confusion = confusion\_matrix(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", e) print("Confusion Matrix:\n", confusion) print("Classification Report:\n", classification\_rep)



### Accuracy Score, Confusion Matrix and Classification Report for voting model (hard)

print("----------------------------------------------------------------Averaging model---------

---------------------------------------------------------------")

averaging\_model = VotingClassifier( estimators=[

('dt', model1),

('lr', model2),

('svc', model3),

('gnb', model4),

('knn', model5),

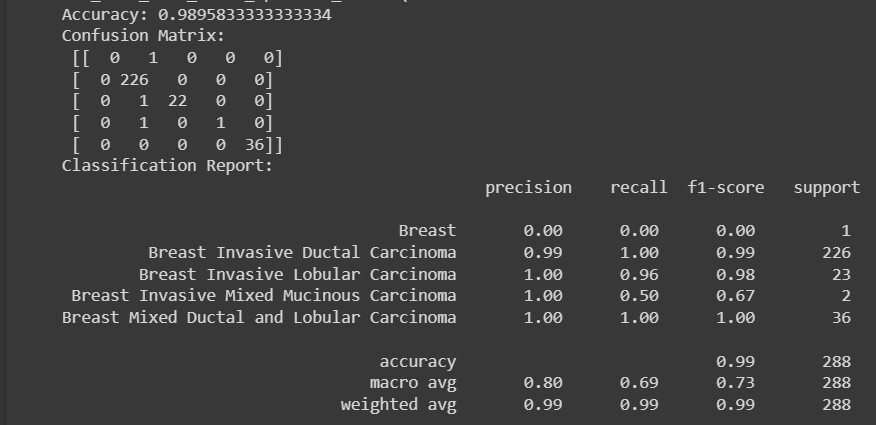
('rf', model6)

],

voting='soft'

)

averaging\_model.fit(X\_train, y\_train) # Make predictions on the test set y\_pred = averaging\_model.predict(X\_test) a = accuracy\_score(y\_test, y\_pred) confusion = confusion\_matrix(y\_test, y\_pred) classification\_rep = classification\_report(y\_test, y\_pred) print("Accuracy:", a) print("Confusion Matrix:\n", confusion) print("Classification Report:\n", classification\_rep)



**Accuracy Score, Confusion Matrix and Classification Report for Averaging model(soft)**

### Result

***Machine Learning models Results***

|  |  |  |
| --- | --- | --- |
| **S.no** | **Machine Learning models** | **Accuracy score** |
| 1 | K-NN classifier | 0.7569444444444444 |
| 2 | Support Vector Machine | 0.9965277777777778 |
| 3 | Decision Tree | 0.86 |
| 4 | Random Forest | 0.9930555555555556 |

***Different ensemble models Results***

|  |  |  |
| --- | --- | --- |
| **S.no** | **Ensemble Models** | **Accuracy score** |
| 1 | Gradient Boosting | 0.798611111111111 |
| 2 | Stacking Classifier | 0.9965277777777778 |
|  | **Voting classifier** |  |
| 3 | Voting model | 0.8125 |
| 4 | Averaging model | 0.9895833333333334 |