NAME-TIASA JANA

REGISTRATION NUMBER-21BIT0612

HACKATHON PROJECT

Discovery Phase: Problem Statement

Breast cancer is a disease in which cells in the breast grow out of control. Breast cancer is the second most frequent cancer in women and men globally. In 2020, there were 2.3 million women diagnosed with breast cancer and 685 000 deaths globally. As of the end of 2020, there were 7.8 million women alive who were diagnosed with breast cancer in the past 5 years, making it the world's most prevalent cancer.These cells usually grow a tumor that can frequently be seen on an x-ray or considered a lump. The tumor is malignant (cancer) if the cells can expand into (invade) encompassing tissues or increase (metastasize) to different sections of the body.

Nowadays Classification and data mining methods are very effective ways to classify data. So with the help of Machine learning if we can classify the patient having which type of cancer, then it will be easy for doctors to provide timely treatment to patients and improve the chance of survival.

Classification of Breast Cancer

In this Machine learning project we are going to analyze and classify Breast Cancer (that the breast cancer belongs to which category)using a data set, as basically there are two categories of breast cancer that is:

- Malignant type breast cancer(M)
- Benign type breast cancer(B)

Data Preparation Phase:

URL of Dataset used:-

Data set is basically a collection of data.

It contains attributes(column) and record(rows).

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

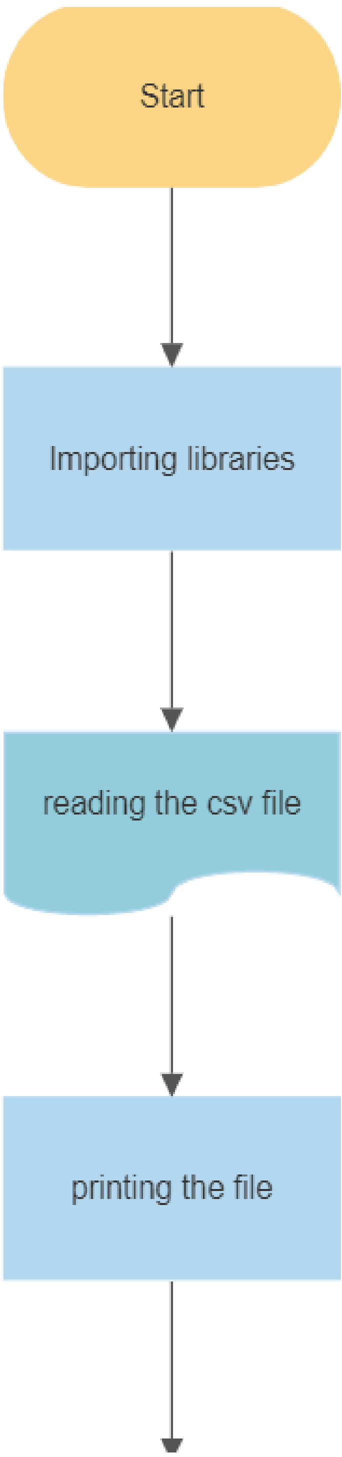
Model Planning Phase:

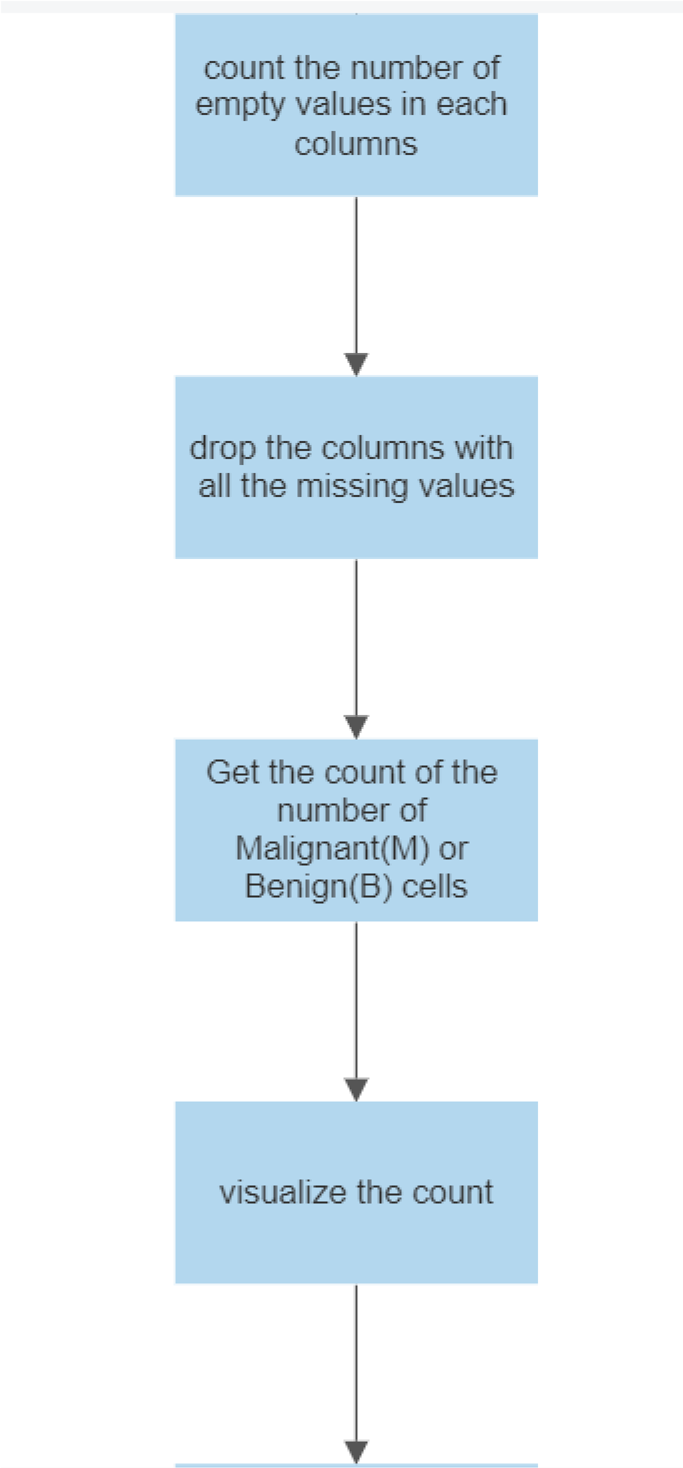
Name of Data Science algorithms used:

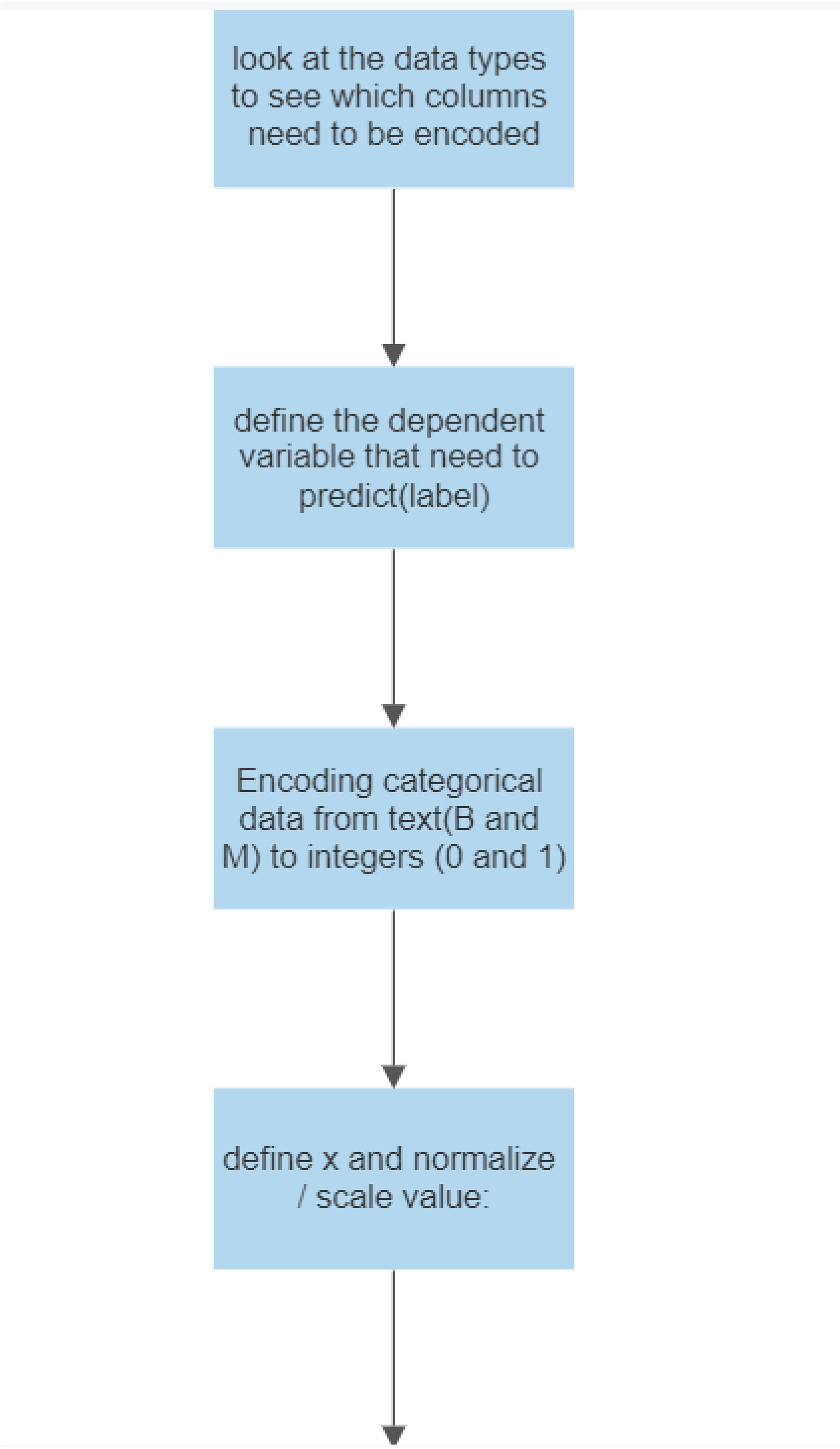
Logistic regression algorithm

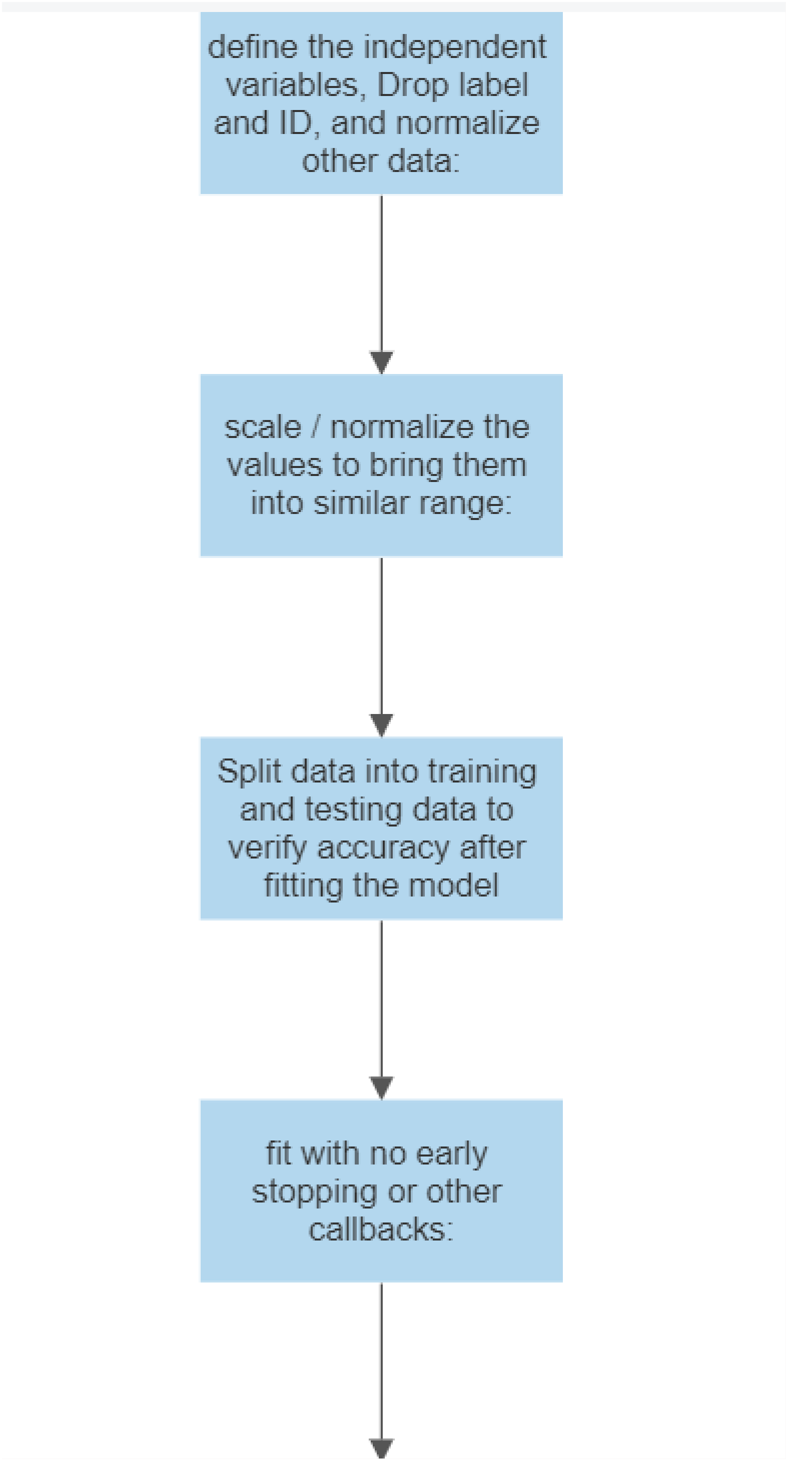
Approach used with flow diagrams of your project

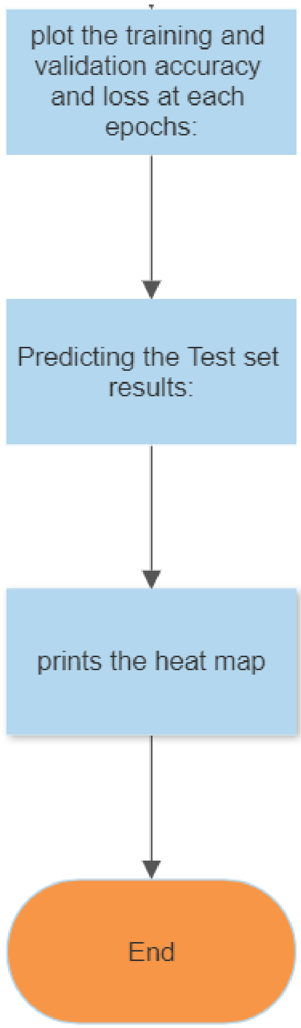
I have used ***jupyter notebook*** to work on this dataset.


Start
Importing libraries
reading the csv file
printing the file


count the number of empty values in each columns
drop the columns with all the missing values
Get the count of the number of Malignant(M) or Benign(B) cells
visualize the count


look at the data types to see which columns need to be encoded
define the dependent variable that need to predict(label)
Encoding categorical data from text(B and M) to integers (0 and 1)
define x and normalize / scale value:


define the independent variables, Drop label and ID, and normalize other data:
scale / normalize the values to bring them into similar range:
Split data into training and testing data to verify accuracy after fitting the model
fit with no early stopping or other callbacks:


plot the training and validation accuracy and loss at each epochs:
Predicting the Test set results:
prints the heat map
End

Model Building Phase: Complete Coding

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

print("hackathon project by Tiasa Jana 21BIT0612")

file = pd.read_csv('data.csv')

#Now let’s view our dataset using head():

file.head(10)

print(file)

file.shape

sns.pairplot(file,hue = 'diagnosis', palette= 'coolwarm', vars = ['radius_mean',
'texture_mean', 'perimeter_mean','area_mean','smoothness_mean'])

# count the number of empty values in each columns:

file.isna().sum()


# drop the columns with all the missing values:

file = file.dropna(axis = 1)


file.shape
```

```
# Get the count of the number of Malignant(M) or Benign(B) cells

file['diagnosis'].value_counts()

# visualize the count:

sns.countplot(file['diagnosis'], label = 'count')

# look at the data types to see which columns need to be encoded:

file.dtypes

file = file.rename(columns = {'diagnosis' : 'label'})

print(file.dtypes)

# define the dependent variable that need to predict(label)

y = file['label'].values

print(np.unique(y))

# Encoding categorical data from text(B and M) to integers (0 and 1)

from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()

Y = labelencoder.fit_transform(y) # M = 1 and B = 0

print(np.unique(Y))

# define x and normalize / scale value:


# define the independent variables, Drop label and ID, and normalize other data:

X  = file.drop(labels=['label','id'],axis = 1)


#scale / normalize the values to bring them into similar range:

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
```

```
scaler.fit(X)

X = scaler.transform(X)


print(X)

# Split data into training and testing data to verify accuracy after fitting the model

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test = train_test_split(X,Y, test_size = 0.3,
random_state=40)

print('Shape of training data is: ', x_train.shape)

print('Shape of testing data is: ', x_test.shape)

import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout


model = Sequential()

model.add(Dense(128, input_dim=30, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(64,activation = 'relu'))

model.add(Dropout(0.5))

model.add(Dense(1))

model.add(Activation('sigmoid'))

model.compile(loss = 'binary_crossentropy', optimizer = 'adam' , metrics =
['accuracy'])


model.summary()
```

```
# fit with no early stopping or other callbacks:

history = model.fit(x_train,y_train,verbose = 1,epochs = 100, batch_size =
64,validation_data = (x_test,y_test))

# plot the training and validation accuracy and loss at each epochs:

loss = history.history['loss']

val_loss = history.history['val_loss']

epochs = range(1,len(loss)+1)

plt.plot(epochs,loss,'y',label = 'Training loss')

plt.plot(epochs,val_loss,'r',label = 'Validation loss')

plt.title('Validation loss by Tiasa Jana')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()


acc = history.history['accuracy']

val_acc = history.history['val_accuracy']

plt.plot(epochs,acc,'y',label = 'accuracy')

plt.plot(epochs,val_acc,'r',label = 'Validation acc')

plt.title('Validation accuracy by Tiasa jana ')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Predicting the Test set results:
```

y_pred = model.predict(x_test)

y_pred = (y_pred > 0.5)

Making the Confusion Matrix:

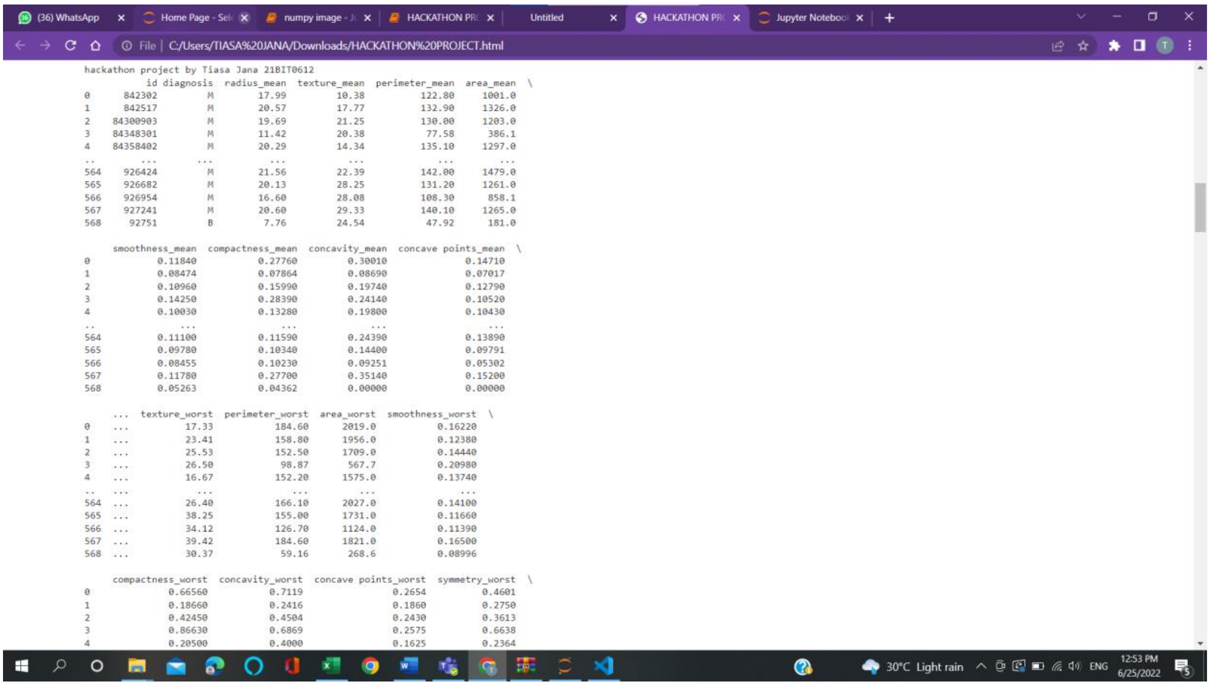
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)

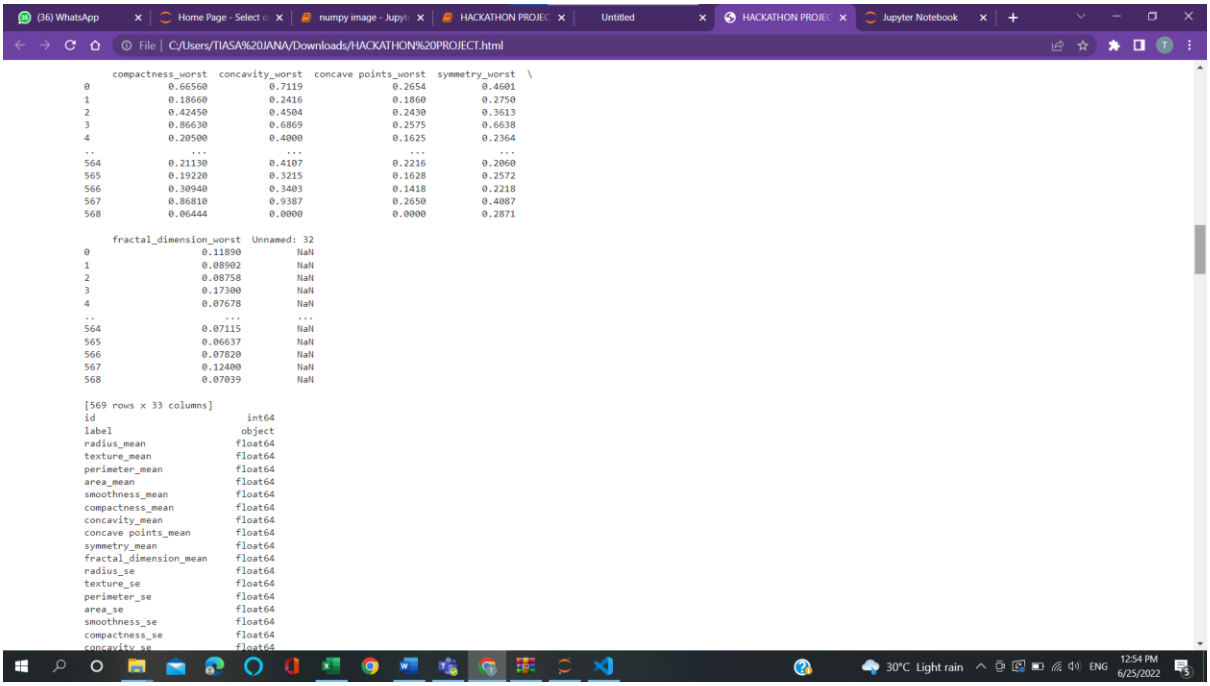
sns.heatmap(cm, annot = True)

Communicate Results Phase:

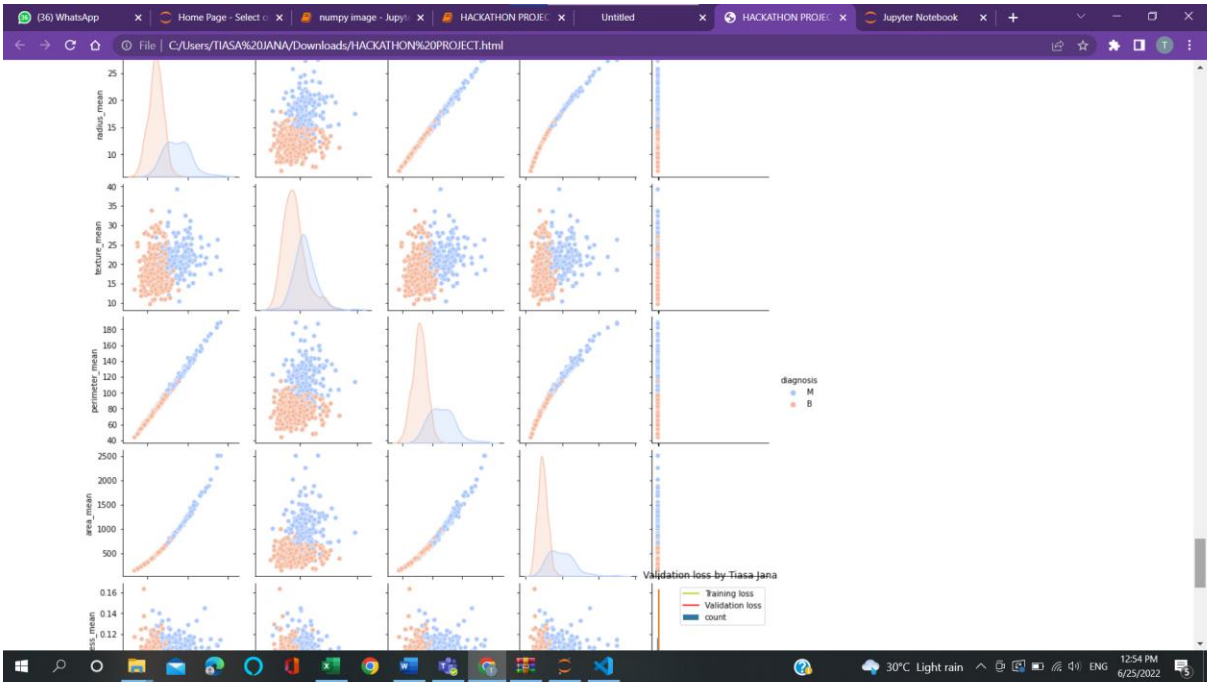
Snapshots of Result

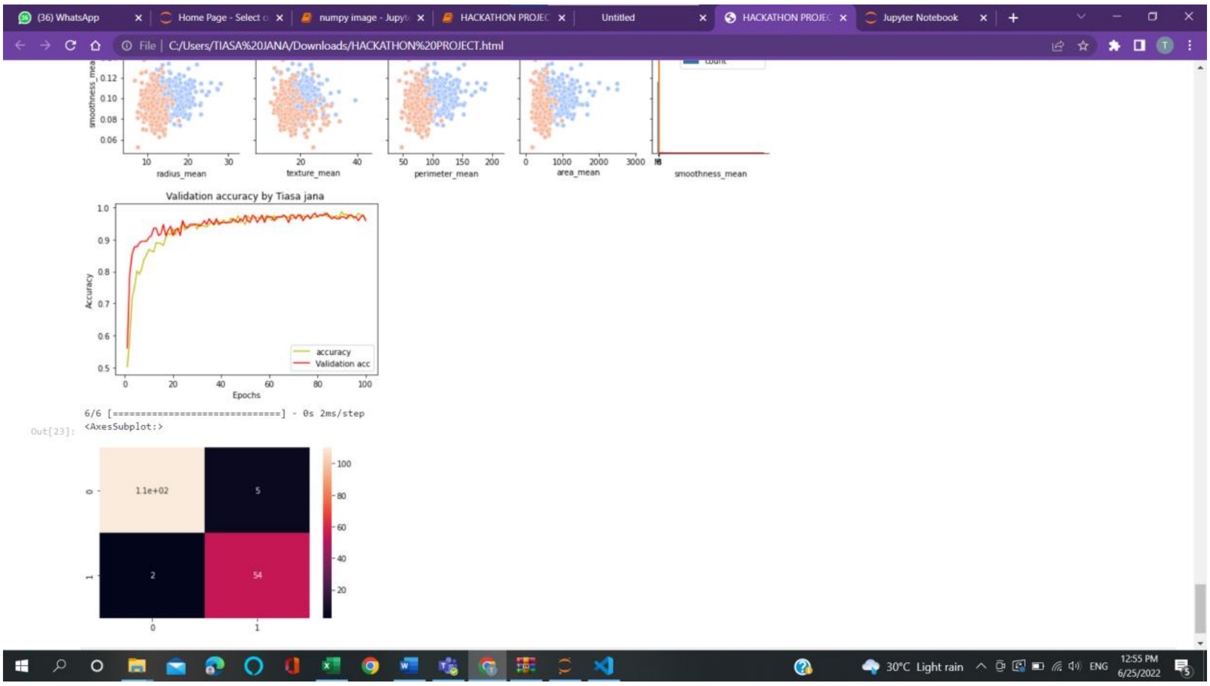


```
hackathon project by Tiasa Jana 21BIT0612
           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0      842302         M        17.99         10.38          122.80     1001.0
1      842517         M        20.57         17.77          132.90     1326.0
2    84300903         M        19.69         21.25          130.00     1203.0
3    84348301         M        11.42         20.38           77.58      386.1
4    84358402         M        20.29         14.34          135.10     1297.0
..        ...       ...          ...           ...             ...        ...
564    926424         M        21.56         22.39          142.00     1479.0
565    926682         M        20.13         28.25          131.20     1261.0
566    926954         M        16.60         28.08          108.30      858.1
567    927241         M        20.60         29.33          140.10     1265.0
568     92751         B         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
1            0.08474           0.07864         0.08690              0.07017
2            0.10960           0.15990         0.19740              0.12790
3            0.14250           0.28390         0.24140              0.10520
4            0.10030           0.13280         0.19800              0.10430
..               ...               ...             ...                  ...
564          0.11100           0.11590         0.24390              0.13890
565          0.09780           0.10340         0.14400              0.09791
566          0.08455           0.10230         0.09251              0.05302
567          0.11780           0.27700         0.35140              0.15200
568          0.05263           0.04362         0.00000              0.00000

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220
1    ...          23.41           158.80      1956.0           0.12380
2    ...          25.53           152.50      1709.0           0.14440
3    ...          26.50            98.87       567.7           0.20980
4    ...          16.67           152.20      1575.0           0.13740
..   ...            ...              ...         ...               ...
564  ...          26.40           166.10      2027.0           0.14100
565  ...          38.25           155.00      1731.0           0.11660
566  ...          34.12           126.70      1124.0           0.11390
567  ...          39.42           184.60      1821.0           0.16500
568  ...          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
```



```
     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 ...              ...                   ...             ...
564            0.21130           0.4107                0.2216          0.2060
565            0.19220           0.3215                0.1628          0.2572
566            0.30940           0.3403                0.1418          0.2218
567            0.86810           0.9387                0.2650          0.4087
568            0.06444           0.0000                0.0000          0.2871

     fractal_dimension_worst  Unnamed: 32
0                    0.11890          NaN
1                    0.08902          NaN
2                    0.08758          NaN
3                    0.17300          NaN
4                    0.07678          NaN
..                       ...          ...
564                  0.07115          NaN
565                  0.06637          NaN
566                  0.07820          NaN
567                  0.12400          NaN
568                  0.07039          NaN

[569 rows x 33 columns]
id                           int64
label                       object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
```





(36) WhatsApp
Home Page - Select
numpy image - Jupyt
HACKATHON PROJEC
Untitled
HACKATHON PROJEC
Jupyter Notebook
File | C:/Users/TIASA%20JANA/Downloads/HACKATHON%20PROJECT.html
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
Validation accuracy by Tiasa jana
Accuracy
Epochs
accuracy
Validation acc
6/6 [==============================] - 0s 2ms/step
Out[23]:
<AxesSubplot:>
1.1e+02
5
2
54
30°C Light rain
ENG
12:55 PM
6/25/2022

In [ ]:

In [23]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
print("hackathon project by Tiasa Jana 21BIT0612")
file = pd.read_csv('data.csv')
#Now let’s view our dataset using head():
file.head(10)
print(file)
file.shape
sns.pairplot(file,hue = 'diagnosis', palette= 'coolwarm', vars = ['radius_mean', 'textu
# count the number of empty values in each columns:
file.isna().sum()

# drop the columns with all the missing values:
file = file.dropna(axis = 1)

file.shape

# Get the count of the number of Malignant(M) or Benign(B) cells
file['diagnosis'].value_counts()
# visualize the count:
sns.countplot(file['diagnosis'], label = 'count')
# look at the data types to see which columns need to be encoded:
file.dtypes
file = file.rename(columns = {'diagnosis' : 'label'})
print(file.dtypes)
# define the dependent variable that need to predict(label)
y = file['label'].values
print(np.unique(y))
# Encoding categorical data from text(B and M) to integers (0 and 1)
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
Y = labelencoder.fit_transform(y) # M = 1 and B = 0
print(np.unique(Y))
# define x and normalize / scale value:

# define the independent variables, Drop label and ID, and normalize other data:
X  = file.drop(labels=['label','id'],axis = 1)

#scale / normalize the values to bring them into similar range:
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)

print(X)
# Split data into training and testing data to verify accuracy after fitting the model
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y, test_size = 0.3, random_state=40)
print('Shape of training data is: ', x_train.shape)
print('Shape of testing data is: ', x_test.shape)
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
```

```
model = Sequential()
model.add(Dense(128, input_dim=30, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64,activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss = 'binary_crossentropy', optimizer = 'adam' , metrics = ['accuracy']

model.summary()
# fit with no early stopping or other callbacks:
history = model.fit(x_train,y_train,verbose = 1,epochs = 100, batch_size = 64,validatio
# plot the training and validation accuracy and loss at each epochs:
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1,len(loss)+1)
plt.plot(epochs,loss,'y',label = 'Training loss')
plt.plot(epochs,val_loss,'r',label = 'Validation loss')
plt.title('Validation loss by Tiasa Jana')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs,acc,'y',label = 'accuracy')
plt.plot(epochs,val_acc,'r',label = 'Validation acc')
plt.title('Validation accuracy by Tiasa jana ')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
# Predicting the Test set results:
y_pred = model.predict(x_test)
y_pred = (y_pred > 0.5)

# Making the Confusion Matrix:
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)

sns.heatmap(cm, annot = True)
```

```
hackathon project by Tiasa Jana 21BIT0612
           id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0      842302         M        17.99         10.38          122.80     1001.0
1      842517         M        20.57         17.77          132.90     1326.0
2    84300903         M        19.69         21.25          130.00     1203.0
3    84348301         M        11.42         20.38           77.58      386.1
4    84358402         M        20.29         14.34          135.10     1297.0
..        ...       ...          ...           ...             ...        ...
564    926424         M        21.56         22.39          142.00     1479.0
565    926682         M        20.13         28.25          131.20     1261.0
566    926954         M        16.60         28.08          108.30      858.1
567    927241         M        20.60         29.33          140.10     1265.0
568     92751         B         7.76         24.54           47.92      181.0

     smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0            0.11840           0.27760         0.30010              0.14710
```

```
1          0.08474          0.07864         0.08690             0.07017
2          0.10960          0.15990         0.19740             0.12790
3          0.14250          0.28390         0.24140             0.10520
4          0.10030          0.13280         0.19800             0.10430
..             ...              ...             ...                 ...
564        0.11100          0.11590         0.24390             0.13890
565        0.09780          0.10340         0.14400             0.09791
566        0.08455          0.10230         0.09251             0.05302
567        0.11780          0.27700         0.35140             0.15200
568        0.05263          0.04362         0.00000             0.00000

     ...  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0    ...          17.33           184.60      2019.0           0.16220
1    ...          23.41           158.80      1956.0           0.12380
2    ...          25.53           152.50      1709.0           0.14440
3    ...          26.50            98.87       567.7           0.20980
4    ...          16.67           152.20      1575.0           0.13740
..   ...            ...              ...         ...               ...
564  ...          26.40           166.10      2027.0           0.14100
565  ...          38.25           155.00      1731.0           0.11660
566  ...          34.12           126.70      1124.0           0.11390
567  ...          39.42           184.60      1821.0           0.16500
568  ...          30.37            59.16       268.6           0.08996

     compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0              0.66560           0.7119                0.2654          0.4601
1              0.18660           0.2416                0.1860          0.2750
2              0.42450           0.4504                0.2430          0.3613
3              0.86630           0.6869                0.2575          0.6638
4              0.20500           0.4000                0.1625          0.2364
..                 ...              ...                   ...             ...
564            0.21130           0.4107                0.2216          0.2060
565            0.19220           0.3215                0.1628          0.2572
566            0.30940           0.3403                0.1418          0.2218
567            0.86810           0.9387                0.2650          0.4087
568            0.06444           0.0000                0.0000          0.2871

     fractional_dimension_worst  Unnamed: 32
0                   0.11890          NaN
1                   0.08902          NaN
2                   0.08758          NaN
3                   0.17300          NaN
4                   0.07678          NaN
..                      ...          ...
564                 0.07115          NaN
565                 0.06637          NaN
566                 0.07820          NaN
567                 0.12400          NaN
568                 0.07039          NaN

[569 rows x 33 columns]
id                           int64
label                       object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
```

```
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
dtype: object
['B' 'M']
[0 1]
[[0.52103744 0.0226581  0.54598853 ... 0.91202749 0.59846245 0.41886396]
 [0.64314449 0.27257355 0.61578329 ... 0.63917526 0.23358959 0.22287813]
 [0.60149557 0.3902604  0.59574321 ... 0.83505155 0.40370589 0.21343303]
 ...
 [0.45525108 0.62123774 0.44578813 ... 0.48728522 0.12872068 0.1519087 ]
 [0.64456434 0.66351031 0.66553797 ... 0.91065292 0.49714173 0.45231536]
 [0.03686876 0.50152181 0.02853984 ... 0.         0.25744136 0.10068215]]
Shape of training data is:  (398, 30)
Shape of testing data is:  (171, 30)
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_18 (Dense)            (None, 128)               3968

 dropout_12 (Dropout)        (None, 128)               0

 dense_19 (Dense)            (None, 64)                8256

 dropout_13 (Dropout)        (None, 64)                0

 dense_20 (Dense)            (None, 1)                 65

 activation_6 (Activation)   (None, 1)                 0

=================================================================
Total params: 12,289
Trainable params: 12,289
Non-trainable params: 0
_________________________________________________________________
C:\Users\TIASA JANA\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarnin
g: Pass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit keywor
```

```
d will result in an error or misinterpretation.
  warnings.warn(
Epoch 1/100
7/7 [==============================] - 1s 47ms/step - loss: 0.6904 - accuracy: 0.5025 -
val_loss: 0.6651 - val_accuracy: 0.5614
Epoch 2/100
7/7 [==============================] - 0s 11ms/step - loss: 0.6637 - accuracy: 0.5854 -
val_loss: 0.6339 - val_accuracy: 0.7895
Epoch 3/100
7/7 [==============================] - 0s 11ms/step - loss: 0.6284 - accuracy: 0.7161 -
val_loss: 0.5989 - val_accuracy: 0.8538
Epoch 4/100
7/7 [==============================] - 0s 11ms/step - loss: 0.5959 - accuracy: 0.7513 -
val_loss: 0.5579 - val_accuracy: 0.8772
Epoch 5/100
7/7 [==============================] - 0s 11ms/step - loss: 0.5560 - accuracy: 0.8015 -
val_loss: 0.5159 - val_accuracy: 0.8772
Epoch 6/100
7/7 [==============================] - 0s 10ms/step - loss: 0.5284 - accuracy: 0.7915 -
val_loss: 0.4675 - val_accuracy: 0.8889
Epoch 7/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4870 - accuracy: 0.8090 -
val_loss: 0.4155 - val_accuracy: 0.8947
Epoch 8/100
7/7 [==============================] - 0s 11ms/step - loss: 0.4427 - accuracy: 0.8392 -
val_loss: 0.3652 - val_accuracy: 0.8947
Epoch 9/100
7/7 [==============================] - 0s 10ms/step - loss: 0.4066 - accuracy: 0.8518 -
val_loss: 0.3263 - val_accuracy: 0.8947
Epoch 10/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3583 - accuracy: 0.8693 -
val_loss: 0.2980 - val_accuracy: 0.9064
Epoch 11/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3341 - accuracy: 0.8643 -
val_loss: 0.2708 - val_accuracy: 0.9123
Epoch 12/100
7/7 [==============================] - 0s 11ms/step - loss: 0.3178 - accuracy: 0.8618 -
val_loss: 0.2442 - val_accuracy: 0.9357
Epoch 13/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2697 - accuracy: 0.8894 -
val_loss: 0.2245 - val_accuracy: 0.9357
Epoch 14/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2701 - accuracy: 0.8894 -
val_loss: 0.2297 - val_accuracy: 0.9123
Epoch 15/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2879 - accuracy: 0.8869 -
val_loss: 0.2023 - val_accuracy: 0.9181
Epoch 16/100
7/7 [==============================] - 0s 12ms/step - loss: 0.2574 - accuracy: 0.8819 -
val_loss: 0.1917 - val_accuracy: 0.9474
Epoch 17/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2429 - accuracy: 0.9070 -
val_loss: 0.1897 - val_accuracy: 0.9123
Epoch 18/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2152 - accuracy: 0.9196 -
val_loss: 0.1775 - val_accuracy: 0.9298
Epoch 19/100
7/7 [==============================] - 0s 10ms/step - loss: 0.2194 - accuracy: 0.9146 -
val_loss: 0.1698 - val_accuracy: 0.9415
Epoch 20/100
```

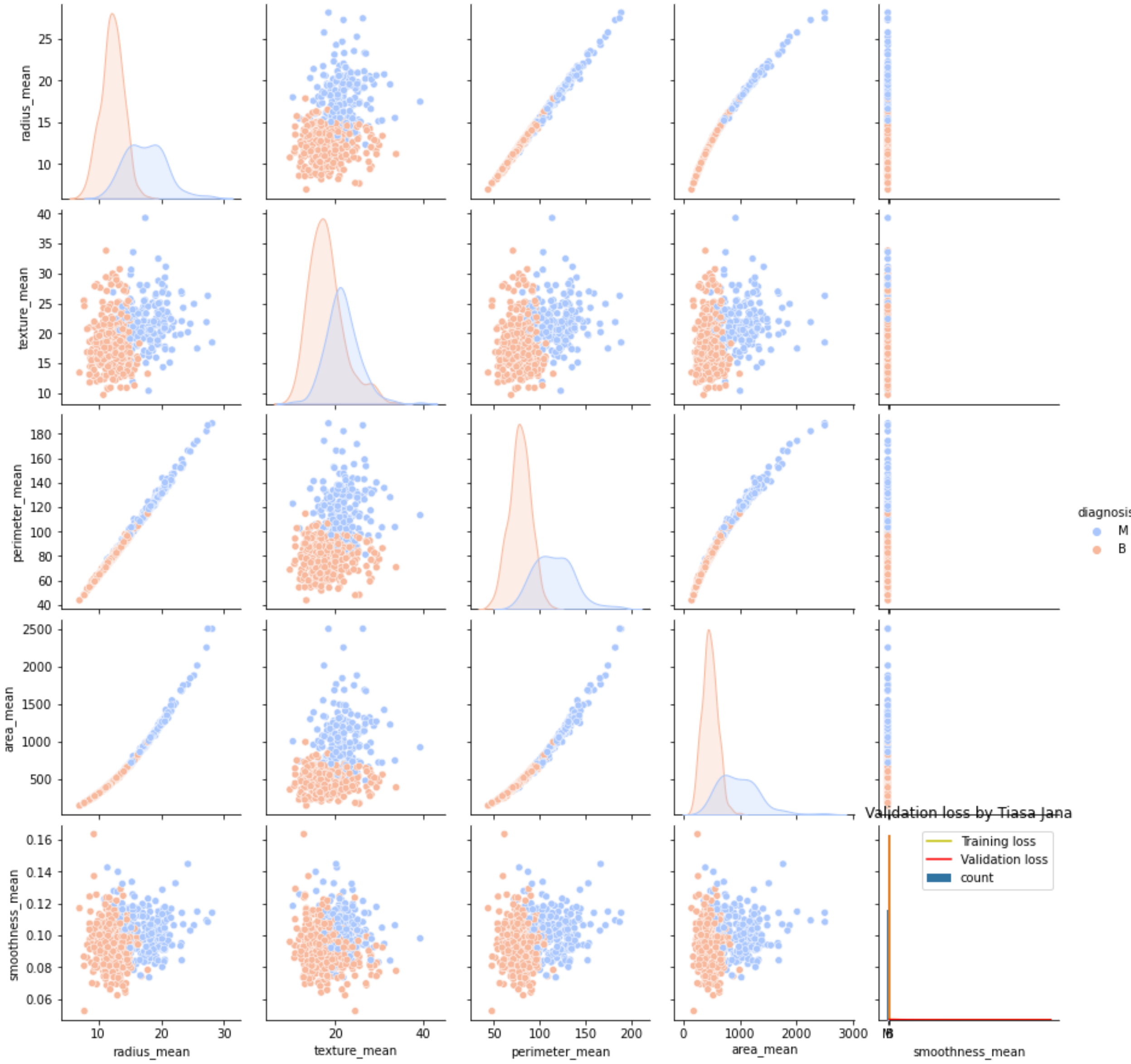
```
7/7 [==============================] - 0s 11ms/step - loss: 0.1835 - accuracy: 0.9347 -
val_loss: 0.1697 - val_accuracy: 0.9123
Epoch 21/100
7/7 [==============================] - 0s 11ms/step - loss: 0.2104 - accuracy: 0.9146 -
val_loss: 0.1615 - val_accuracy: 0.9298
Epoch 22/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1798 - accuracy: 0.9322 -
val_loss: 0.1561 - val_accuracy: 0.9357
Epoch 23/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1850 - accuracy: 0.9347 -
val_loss: 0.1567 - val_accuracy: 0.9123
Epoch 24/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1509 - accuracy: 0.9573 -
val_loss: 0.1414 - val_accuracy: 0.9591
Epoch 25/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1783 - accuracy: 0.9296 -
val_loss: 0.1371 - val_accuracy: 0.9415
Epoch 26/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1628 - accuracy: 0.9397 - v
al_loss: 0.1451 - val_accuracy: 0.9357
Epoch 27/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1577 - accuracy: 0.9422 -
val_loss: 0.1307 - val_accuracy: 0.9474
Epoch 28/100
7/7 [==============================] - 0s 9ms/step - loss: 0.1499 - accuracy: 0.9422 - v
al_loss: 0.1273 - val_accuracy: 0.9474
Epoch 29/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1333 - accuracy: 0.9497 -
val_loss: 0.1243 - val_accuracy: 0.9474
Epoch 30/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1463 - accuracy: 0.9322 -
val_loss: 0.1226 - val_accuracy: 0.9474
Epoch 31/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1339 - accuracy: 0.9497 -
val_loss: 0.1259 - val_accuracy: 0.9415
Epoch 32/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1330 - accuracy: 0.9422 -
val_loss: 0.1212 - val_accuracy: 0.9474
Epoch 33/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1340 - accuracy: 0.9422 -
val_loss: 0.1131 - val_accuracy: 0.9591
Epoch 34/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1323 - accuracy: 0.9397 -
val_loss: 0.1174 - val_accuracy: 0.9474
Epoch 35/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1325 - accuracy: 0.9472 -
val_loss: 0.1111 - val_accuracy: 0.9649
Epoch 36/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1173 - accuracy: 0.9573 -
val_loss: 0.1135 - val_accuracy: 0.9532
Epoch 37/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1383 - accuracy: 0.9447 -
val_loss: 0.1160 - val_accuracy: 0.9474
Epoch 38/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1455 - accuracy: 0.9548 -
val_loss: 0.1092 - val_accuracy: 0.9649
Epoch 39/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1131 - accuracy: 0.9573 -
val_loss: 0.1143 - val_accuracy: 0.9474
Epoch 40/100
```

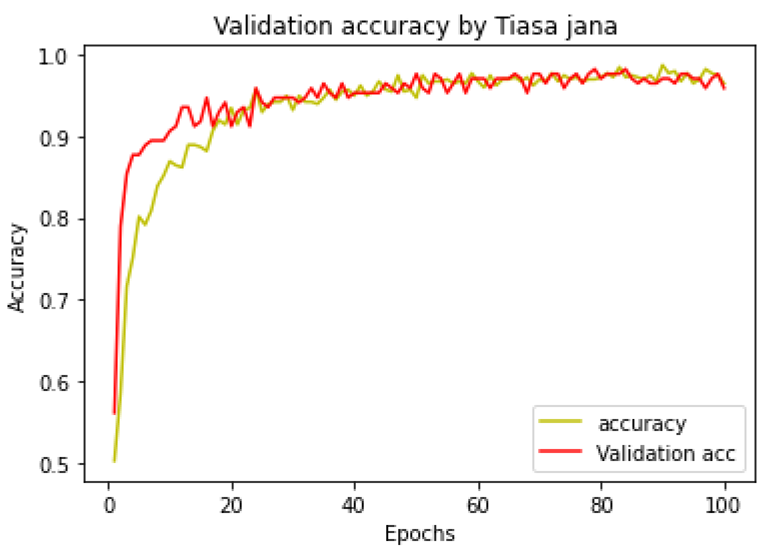
```
7/7 [==============================] - 0s 10ms/step - loss: 0.1192 - accuracy: 0.9497 - val_loss: 0.1107 - val_accuracy: 0.9532
Epoch 41/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1099 - accuracy: 0.9623 - val_loss: 0.1082 - val_accuracy: 0.9532
Epoch 42/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1182 - accuracy: 0.9497 - val_loss: 0.1148 - val_accuracy: 0.9532
Epoch 43/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1088 - accuracy: 0.9548 - val_loss: 0.1209 - val_accuracy: 0.9532
Epoch 44/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1000 - accuracy: 0.9673 - val_loss: 0.1064 - val_accuracy: 0.9532
Epoch 45/100
7/7 [==============================] - 0s 12ms/step - loss: 0.1043 - accuracy: 0.9573 - val_loss: 0.1001 - val_accuracy: 0.9649
Epoch 46/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1016 - accuracy: 0.9548 - val_loss: 0.1041 - val_accuracy: 0.9591
Epoch 47/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0918 - accuracy: 0.9749 - val_loss: 0.1055 - val_accuracy: 0.9532
Epoch 48/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1130 - accuracy: 0.9548 - val_loss: 0.1024 - val_accuracy: 0.9649
Epoch 49/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1068 - accuracy: 0.9573 - val_loss: 0.1043 - val_accuracy: 0.9591
Epoch 50/100
7/7 [==============================] - 0s 11ms/step - loss: 0.1043 - accuracy: 0.9472 - val_loss: 0.0974 - val_accuracy: 0.9766
Epoch 51/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0911 - accuracy: 0.9749 - val_loss: 0.1027 - val_accuracy: 0.9591
Epoch 52/100
7/7 [==============================] - 0s 12ms/step - loss: 0.0917 - accuracy: 0.9648 - val_loss: 0.1110 - val_accuracy: 0.9532
Epoch 53/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0907 - accuracy: 0.9673 - val_loss: 0.0958 - val_accuracy: 0.9766
Epoch 54/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1011 - accuracy: 0.9673 - val_loss: 0.0972 - val_accuracy: 0.9708
Epoch 55/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0819 - accuracy: 0.9698 - val_loss: 0.1085 - val_accuracy: 0.9532
Epoch 56/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0874 - accuracy: 0.9648 - val_loss: 0.1011 - val_accuracy: 0.9649
Epoch 57/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0853 - accuracy: 0.9673 - val_loss: 0.0944 - val_accuracy: 0.9766
Epoch 58/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0951 - accuracy: 0.9648 - val_loss: 0.1089 - val_accuracy: 0.9532
Epoch 59/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0862 - accuracy: 0.9774 - val_loss: 0.1011 - val_accuracy: 0.9708
Epoch 60/100
```

```
7/7 [==============================] - 0s 10ms/step - loss: 0.0869 - accuracy: 0.9673 -
val_loss: 0.1004 - val_accuracy: 0.9708
Epoch 61/100
7/7 [==============================] - 0s 10ms/step - loss: 0.1103 - accuracy: 0.9598 -
val_loss: 0.0999 - val_accuracy: 0.9708
Epoch 62/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0735 - accuracy: 0.9749 -
val_loss: 0.1037 - val_accuracy: 0.9591
Epoch 63/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0908 - accuracy: 0.9623 -
val_loss: 0.0982 - val_accuracy: 0.9708
Epoch 64/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0718 - accuracy: 0.9698 -
val_loss: 0.0982 - val_accuracy: 0.9708
Epoch 65/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0776 - accuracy: 0.9698 -
val_loss: 0.0956 - val_accuracy: 0.9708
Epoch 66/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0773 - accuracy: 0.9724 - v
al_loss: 0.0938 - val_accuracy: 0.9766
Epoch 67/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0863 - accuracy: 0.9673 -
val_loss: 0.1010 - val_accuracy: 0.9708
Epoch 68/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0803 - accuracy: 0.9724 -
val_loss: 0.1117 - val_accuracy: 0.9532
Epoch 69/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0807 - accuracy: 0.9623 -
val_loss: 0.0959 - val_accuracy: 0.9766
Epoch 70/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0767 - accuracy: 0.9698 -
val_loss: 0.0963 - val_accuracy: 0.9766
Epoch 71/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0779 - accuracy: 0.9673 -
val_loss: 0.1024 - val_accuracy: 0.9649
Epoch 72/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0729 - accuracy: 0.9774 -
val_loss: 0.0974 - val_accuracy: 0.9766
Epoch 73/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0861 - accuracy: 0.9673 -
val_loss: 0.0963 - val_accuracy: 0.9766
Epoch 74/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0839 - accuracy: 0.9749 -
val_loss: 0.1083 - val_accuracy: 0.9591
Epoch 75/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0829 - accuracy: 0.9698 -
val_loss: 0.0970 - val_accuracy: 0.9708
Epoch 76/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0709 - accuracy: 0.9724 -
val_loss: 0.0945 - val_accuracy: 0.9766
Epoch 77/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0805 - accuracy: 0.9673 -
val_loss: 0.1014 - val_accuracy: 0.9649
Epoch 78/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0700 - accuracy: 0.9698 -
val_loss: 0.0949 - val_accuracy: 0.9766
Epoch 79/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0807 - accuracy: 0.9698 -
val_loss: 0.0903 - val_accuracy: 0.9825
Epoch 80/100
```

```
7/7 [==============================] - 0s 10ms/step - loss: 0.0762 - accuracy: 0.9724 -
val_loss: 0.0957 - val_accuracy: 0.9708
Epoch 81/100
7/7 [==============================] - 0s 9ms/step - loss: 0.0686 - accuracy: 0.9774 - v
al_loss: 0.0938 - val_accuracy: 0.9766
Epoch 82/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0722 - accuracy: 0.9724 -
val_loss: 0.0943 - val_accuracy: 0.9766
Epoch 83/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0616 - accuracy: 0.9849 -
val_loss: 0.0909 - val_accuracy: 0.9766
Epoch 84/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0771 - accuracy: 0.9724 -
val_loss: 0.0898 - val_accuracy: 0.9825
Epoch 85/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0750 - accuracy: 0.9749 -
val_loss: 0.0933 - val_accuracy: 0.9708
Epoch 86/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0688 - accuracy: 0.9724 -
val_loss: 0.1015 - val_accuracy: 0.9649
Epoch 87/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0690 - accuracy: 0.9698 -
val_loss: 0.0936 - val_accuracy: 0.9708
Epoch 88/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0699 - accuracy: 0.9749 -
val_loss: 0.0978 - val_accuracy: 0.9649
Epoch 89/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0739 - accuracy: 0.9673 -
val_loss: 0.0981 - val_accuracy: 0.9649
Epoch 90/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0547 - accuracy: 0.9874 -
val_loss: 0.0946 - val_accuracy: 0.9708
Epoch 91/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0688 - accuracy: 0.9774 -
val_loss: 0.0959 - val_accuracy: 0.9708
Epoch 92/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0627 - accuracy: 0.9799 -
val_loss: 0.0976 - val_accuracy: 0.9649
Epoch 93/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0749 - accuracy: 0.9673 -
val_loss: 0.0939 - val_accuracy: 0.9766
Epoch 94/100
7/7 [==============================] - 0s 10ms/step - loss: 0.0689 - accuracy: 0.9774 -
val_loss: 0.0935 - val_accuracy: 0.9766
Epoch 95/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0686 - accuracy: 0.9648 -
val_loss: 0.1008 - val_accuracy: 0.9708
Epoch 96/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0768 - accuracy: 0.9698 -
val_loss: 0.0974 - val_accuracy: 0.9708
Epoch 97/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0604 - accuracy: 0.9824 -
val_loss: 0.1060 - val_accuracy: 0.9591
Epoch 98/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0700 - accuracy: 0.9774 -
val_loss: 0.0973 - val_accuracy: 0.9708
Epoch 99/100
7/7 [==============================] - 0s 11ms/step - loss: 0.0744 - accuracy: 0.9749 -
val_loss: 0.0957 - val_accuracy: 0.9766
Epoch 100/100
```

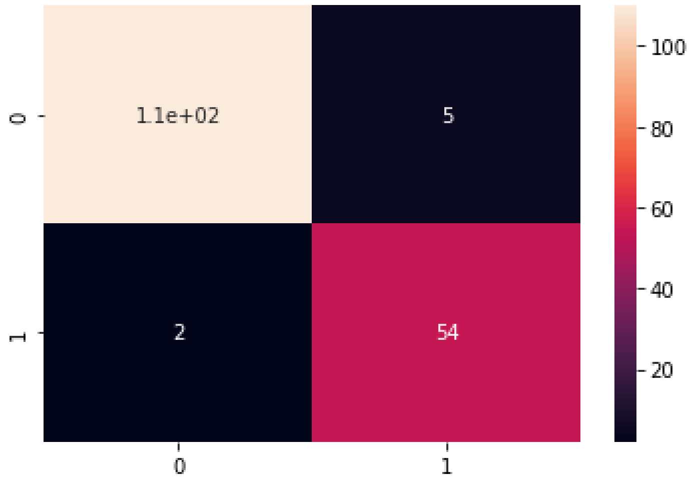
```
7/7 [==============================] - 0s 11ms/step - loss: 0.0671 - accuracy: 0.9648 -
val_loss: 0.1069 - val_accuracy: 0.9591
```






```
6/6 [==============================] - 0s 2ms/step
```

Out[23]: <AxesSubplot:>



In [ ]: