CODE

(A)For Numerical Dataset

1.PredictionForm.html

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
#data = load_breast_cancer()
data= pd.read_csv('BCPD.csv')
X = data[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
#X = data.data
Y = data[["diagnosis"]]
#y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,
random_state=42)

# Create an SVM model with a linear kernel
model = SVC(kernel='linear')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

color = 'white'
matrix = plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
matrix.ax_.set_title('Confusion Matrix', color=color)
```

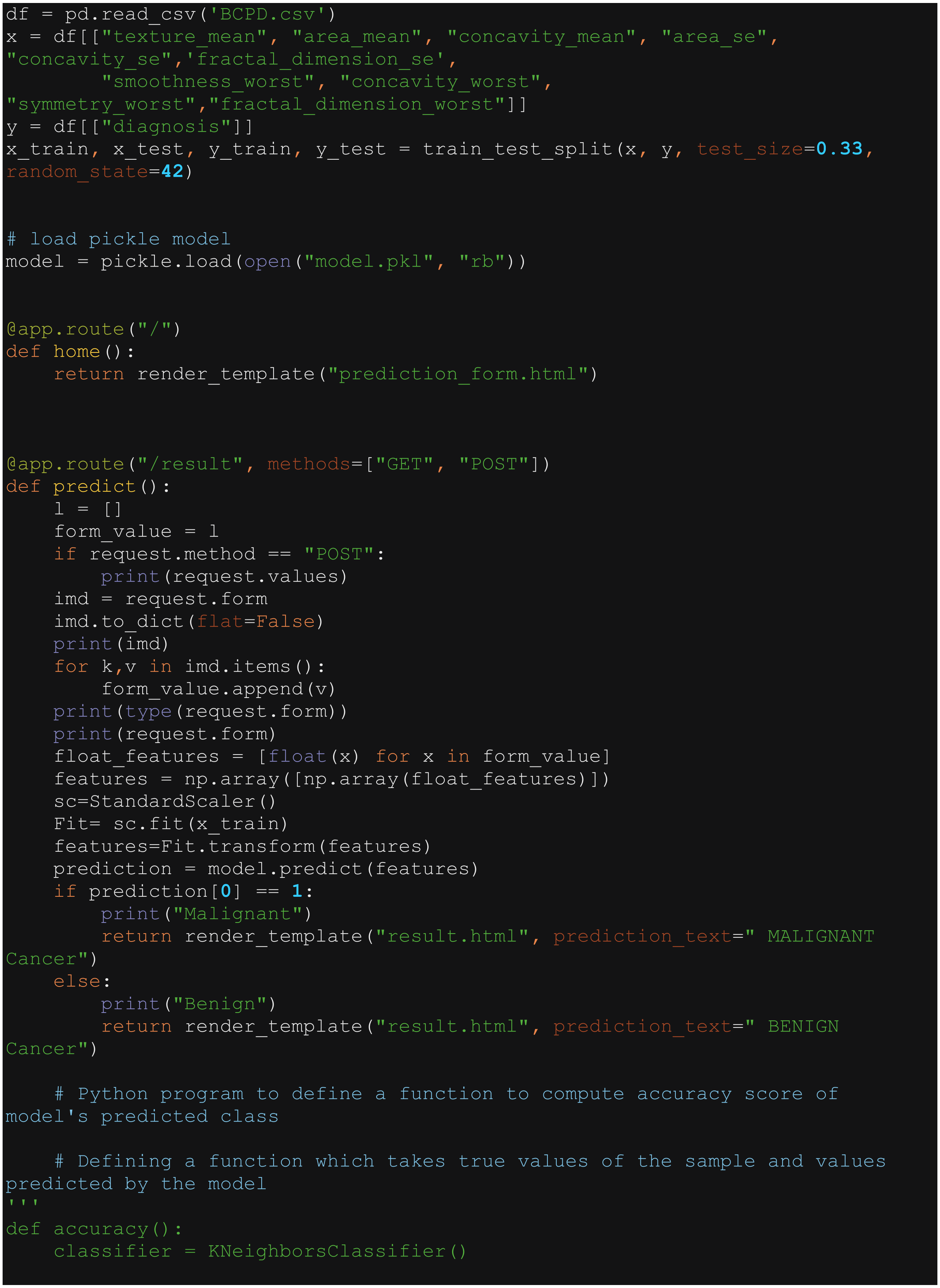
```
plt.xlabel('Predicted Label', color=color)
plt.ylabel('True Label', color=color)
plt.gcf().axes[0].tick_params(colors=color)
plt.gcf().axes[1].tick_params(colors=color)
plt.show()

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

2.result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Result</title>
<style>
      body {
        margin: 8px 16px;
        background-color: LightPink;
        padding: 12px 24px;
        border: 1px solid black;
        border-radius: 4px;
      }
    </style>


</head>

<body style="font-size:20";  background-color="#212F3C"; color=" #FFFFF0";>


<h1>
    The patient is more likely to have {{ prediction_text }}</h1>


</body>
</html>
```

3.App.py

```
import os
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier

# create flask app
app = Flask(__name__)
```



```
df = pd.read_csv('BCPD.csv')
x = df[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
y = df[["diagnosis"]]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,
random_state=42)


# load pickle model
model = pickle.load(open("model.pkl", "rb"))


@app.route("/")
def home():
    return render_template("prediction_form.html")



@app.route("/result", methods=["GET", "POST"])
def predict():
    l = []
    form_value = l
    if request.method == "POST":
        print(request.values)
    imd = request.form
    imd.to_dict(flat=False)
    print(imd)
    for k,v in imd.items():
        form_value.append(v)
    print(type(request.form))
    print(request.form)
    float_features = [float(x) for x in form_value]
    features = np.array([np.array(float_features)])
    sc=StandardScaler()
    Fit= sc.fit(x_train)
    features=Fit.transform(features)
    prediction = model.predict(features)
    if prediction[0] == 1:
        print("Malignant")
        return render_template("result.html", prediction_text=" MALIGNANT
Cancer")
    else:
        print("Benign")
        return render_template("result.html", prediction_text=" BENIGN
Cancer")

    # Python program to define a function to compute accuracy score of
model's predicted class

    # Defining a function which takes true values of the sample and values
predicted by the model
'''
def accuracy():
    classifier = KNeighborsClassifier()
    y_pred = classifier.predict(x_test)
    av = accuracy_score(y_test, y_pred)
    return render_template('result.html', av=av)
'''


if __name__ == "__main__":
    app.run(debug=True,use_reloader=False,port=8080)
    ''''
    HOST = os.environ.get('SERVER_HOST', 'localhost')
    try:
        PORT = int(os.environ.get('SERVER_PORT', '5555'))
    except ValueError:
        PORT = 5555

    app.run(HOST, PORT)
    '''
```

4.knn.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import io
import os
import math
import pickle
from flask import render_template
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

from sklearn.datasets import load_breast_cancer


df = pd.read_csv('BCPD.csv')
print(df.head())
#print("target name:", df[''])
# select dependent and independent variable
x = df[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
#x = df[[ "radius_mean",  'perimeter_mean', 'area_mean', 'symmetry_mean',
'compactness_mean', 'concave points_mean']]
y = df[["diagnosis"]]

# split the data into train and test

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,
random_state=42)

# feature scaling
sc = StandardScaler()
Fit = sc.fit(x_train)
x_train = Fit.transform(x_train)
x_test = Fit.transform(x_test)

# instantiate model
classifier = KNeighborsClassifier()






# fit the model90
classifier.fit(x_train, y_train)
# Make predictions on the testing data
y_pred = classifier.predict(x_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)


# make pickle file of our model
pickle.dump(classifier, open("model.pkl", "wb"))





from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

color = 'white'
matrix = plot_confusion_matrix(classifier, x_test, y_test, cmap=plt.cm.Blues)
matrix.ax_.set_title('Confusion Matrix', color=color)
plt.xlabel('Predicted Label', color=color)
plt.ylabel('True Label', color=color)
plt.gcf().axes[0].tick_params(colors=color)
plt.gcf().axes[1].tick_params(colors=color)
plt.show()
```

5.svm.py

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

```

```
# Load the breast cancer dataset
#data = load_breast_cancer()
data= pd.read_csv('BCPD.csv')
X = data[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
#X = data.data
Y = data[["diagnosis"]]
#y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,
random_state=42)

# Create an SVM model with a linear kernel
model = SVC(kernel='linear')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

color = 'white'
matrix = plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
matrix.ax_.set_title('Confusion Matrix', color=color)
plt.xlabel('Predicted Label', color=color)
plt.ylabel('True Label', color=color)
plt.gcf().axes[0].tick_params(colors=color)
plt.gcf().axes[1].tick_params(colors=color)
plt.show()

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

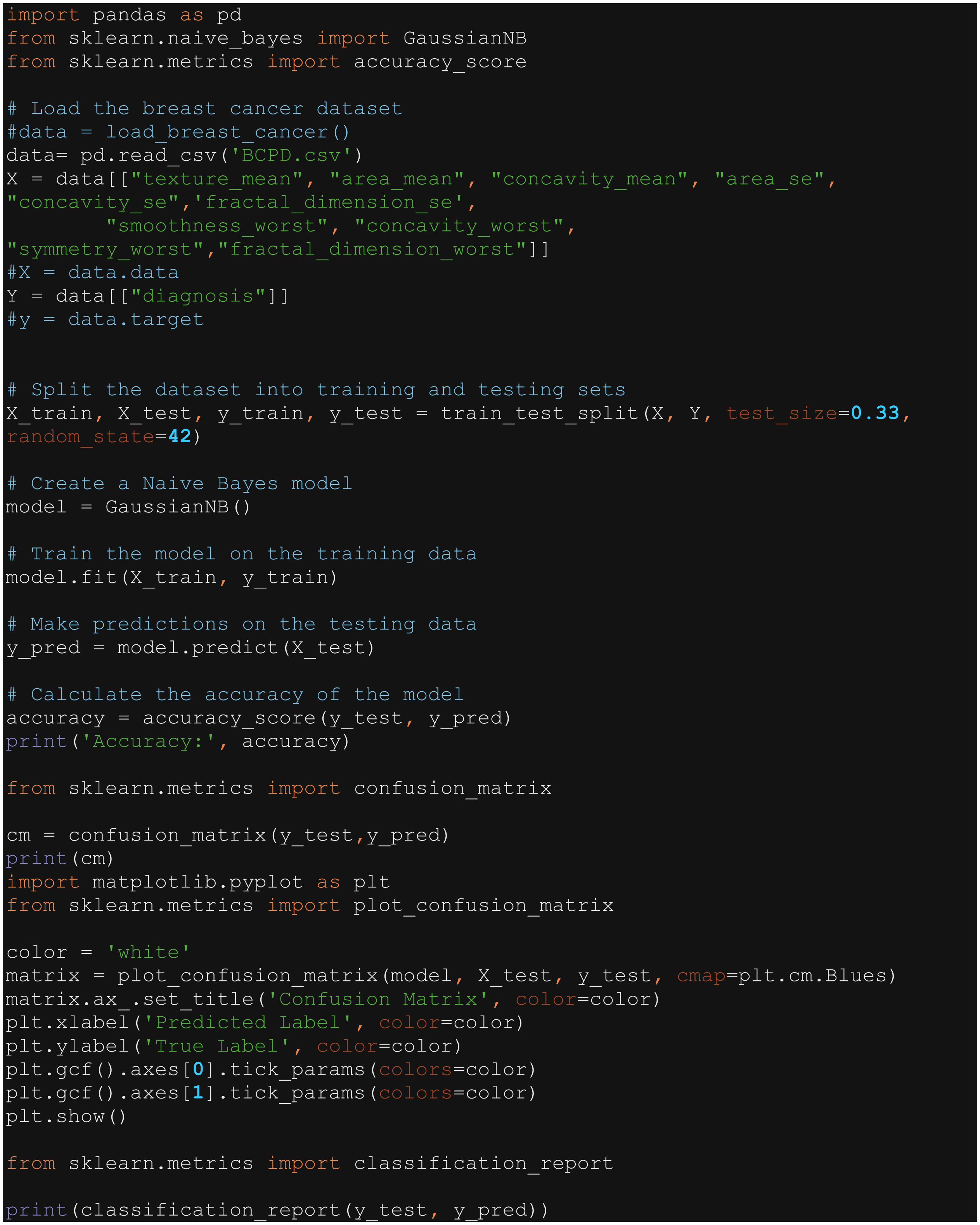
6.lr.py

```
from sklearn.datasets import load_breast_cancer
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Load the breast cancer dataset
#data = load_breast_cancer()


data= pd.read_csv('BCPD.csv')
X = data[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
#X = data.data
Y = data[["diagnosis"]]
#y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,
random_state=42)

# Create a logistic regression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

color = 'white'
matrix = plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
matrix.ax_.set_title('Confusion Matrix', color=color)
plt.xlabel('Predicted Label', color=color)
plt.ylabel('True Label', color=color)
plt.gcf().axes[0].tick_params(colors=color)
plt.gcf().axes[1].tick_params(colors=color)
plt.show()

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

7.nb.py

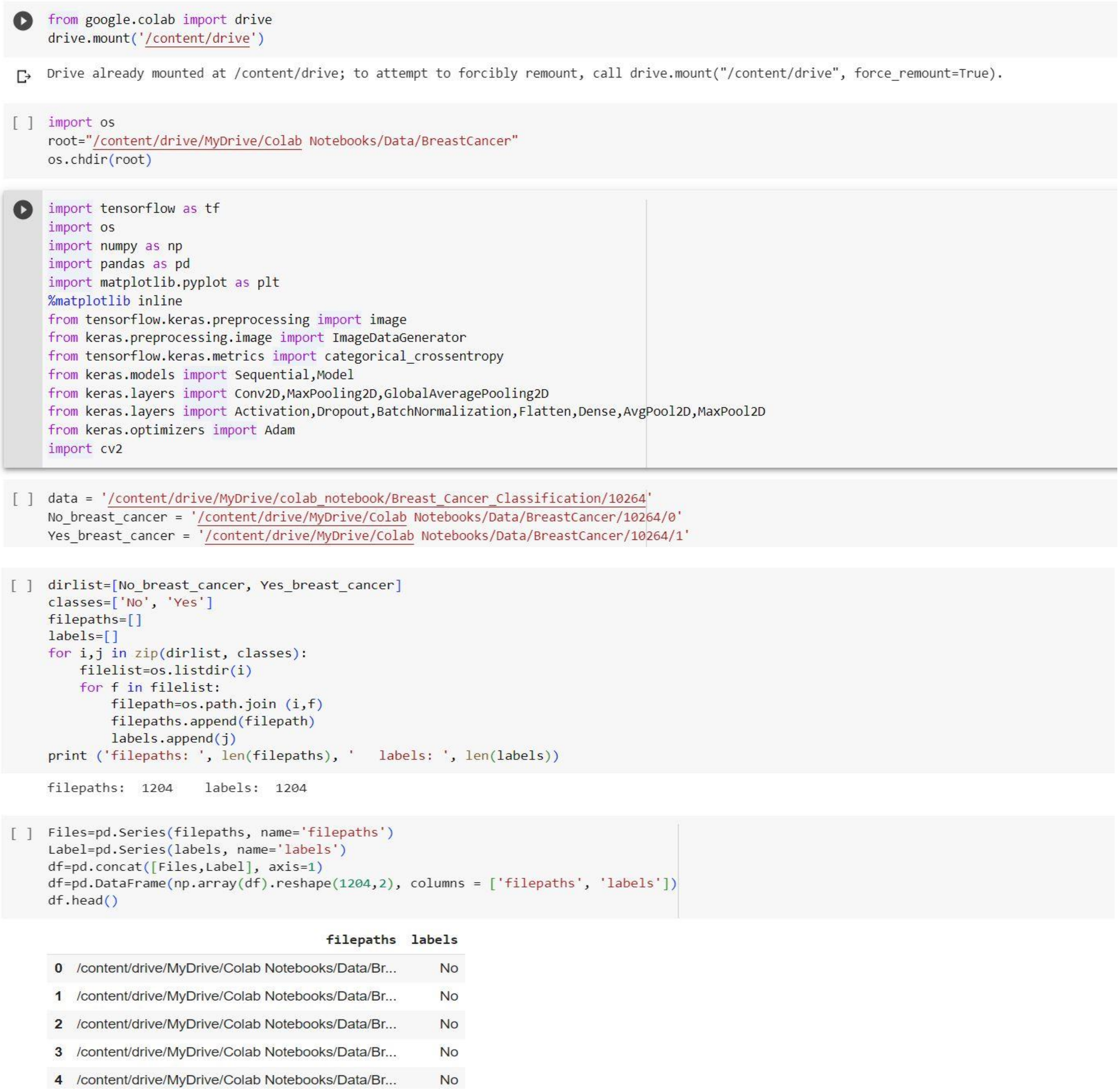
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```



```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
#data = load_breast_cancer()
data= pd.read_csv('BCPD.csv')
X = data[["texture_mean", "area_mean", "concavity_mean", "area_se",
"concavity_se",'fractal_dimension_se',
        "smoothness_worst", "concavity_worst",
"symmetry_worst","fractal_dimension_worst"]]
#X = data.data
Y = data[["diagnosis"]]
#y = data.target


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33,
random_state=42)

# Create a Naive Bayes model
model = GaussianNB()

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy:', accuracy)

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test,y_pred)
print(cm)
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

color = 'white'
matrix = plot_confusion_matrix(model, X_test, y_test, cmap=plt.cm.Blues)
matrix.ax_.set_title('Confusion Matrix', color=color)
plt.xlabel('Predicted Label', color=color)
plt.ylabel('True Label', color=color)
plt.gcf().axes[0].tick_params(colors=color)
plt.gcf().axes[1].tick_params(colors=color)
plt.show()

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

(B)For Image dataset

CNN



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
root="/content/drive/MyDrive/Colab Notebooks/Data/BreastCancer"
os.chdir(root)
```

```
import tensorflow as tf
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from tensorflow.keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.metrics import categorical_crossentropy
from keras.models import Sequential,Model
from keras.layers import Conv2D,MaxPooling2D,GlobalAveragePooling2D
from keras.layers import Activation,Dropout,BatchNormalization,Flatten,Dense,AvgPool2D,MaxPool2D
from keras.optimizers import Adam
import cv2
```

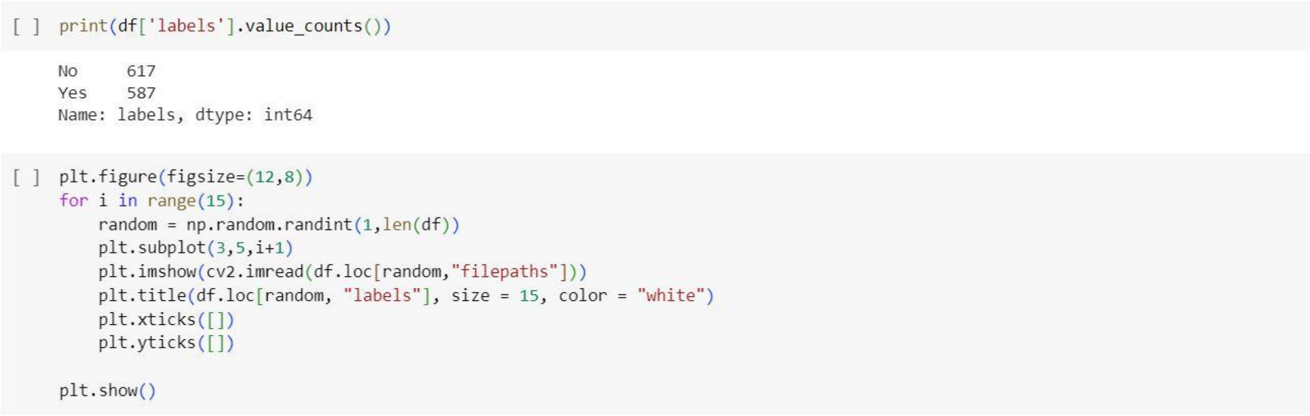
```
data = '/content/drive/MyDrive/colab_notebook/Breast_Cancer_Classification/10264'
No_breast_cancer = '/content/drive/MyDrive/Colab Notebooks/Data/BreastCancer/10264/0'
Yes_breast_cancer = '/content/drive/MyDrive/Colab Notebooks/Data/BreastCancer/10264/1'
```

```
dirlist=[No_breast_cancer, Yes_breast_cancer]
classes=['No', 'Yes']
filepaths=[]
labels=[]
for i,j in zip(dirlist, classes):
    filelist=os.listdir(i)
    for f in filelist:
        filepath=os.path.join (i,f)
        filepaths.append(filepath)
        labels.append(j)
print ('filepaths: ', len(filepaths), '   labels: ', len(labels))
```

```
filepaths:  1204    labels:  1204
```

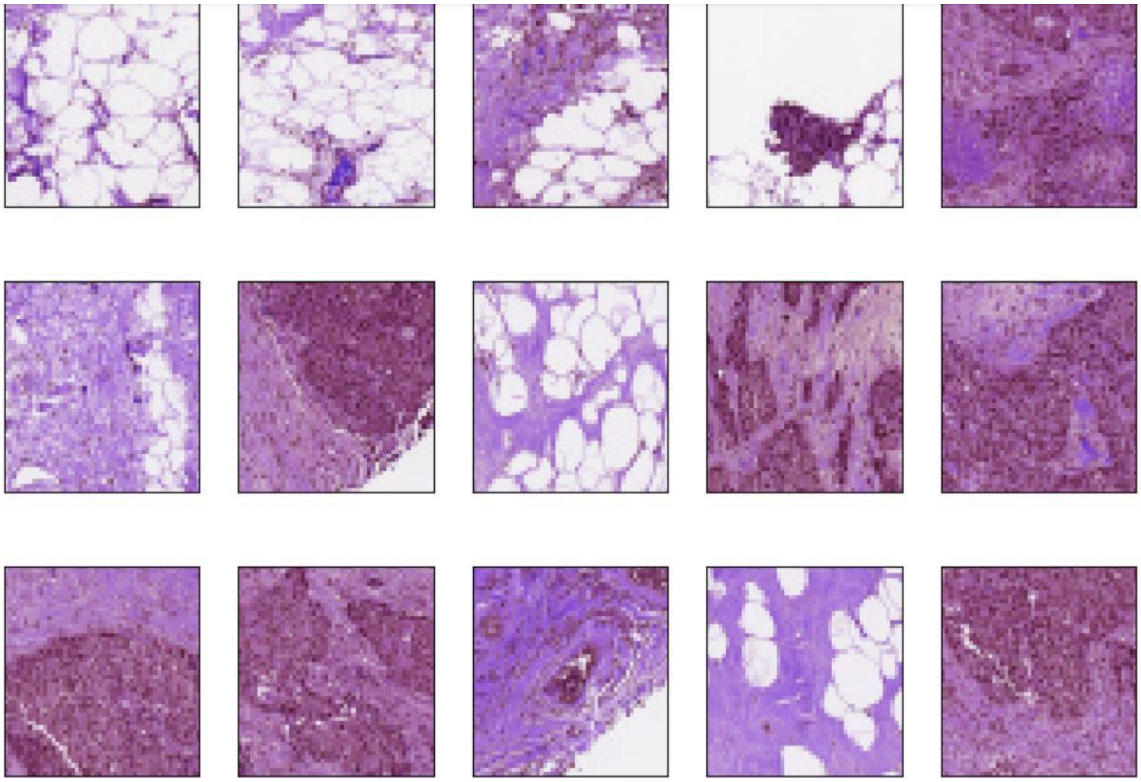
```
Files=pd.Series(filepaths, name='filepaths')
Label=pd.Series(labels, name='labels')
df=pd.concat([Files,Label], axis=1)
df=pd.DataFrame(np.array(df).reshape(1204,2), columns = ['filepaths', 'labels'])
df.head()
```

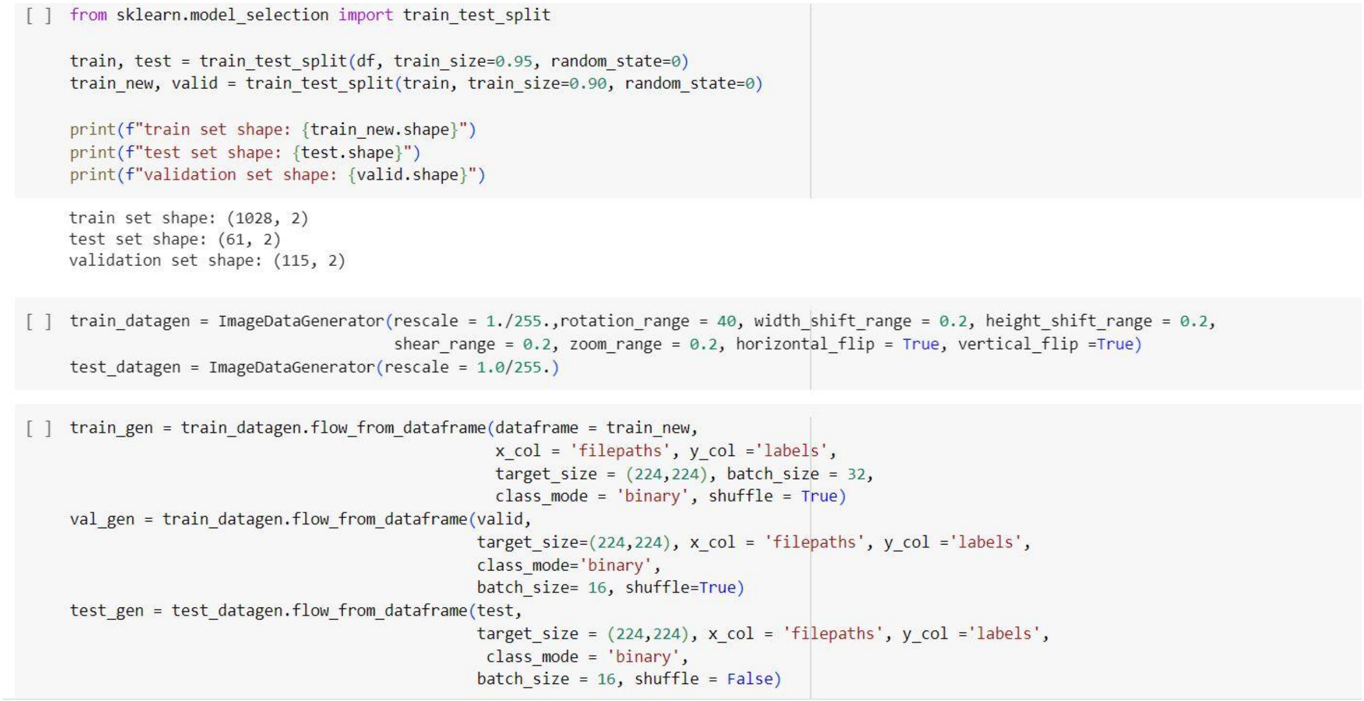
	filepaths	labels
0	/content/drive/MyDrive/Colab Notebooks/Data/Br...	No
1	/content/drive/MyDrive/Colab Notebooks/Data/Br...	No
2	/content/drive/MyDrive/Colab Notebooks/Data/Br...	No
3	/content/drive/MyDrive/Colab Notebooks/Data/Br...	No
4	/content/drive/MyDrive/Colab Notebooks/Data/Br...	No



```
[ ]  print(df['labels'].value_counts())

     No     617
     Yes    587
     Name: labels, dtype: int64

[ ]  plt.figure(figsize=(12,8))
     for i in range(15):
         random = np.random.randint(1,len(df))
         plt.subplot(3,5,i+1)
         plt.imshow(cv2.imread(df.loc[random,"filepaths"]))
         plt.title(df.loc[random, "labels"], size = 15, color = "white")
         plt.xticks([])
         plt.yticks([])

     plt.show()
```

[ ]

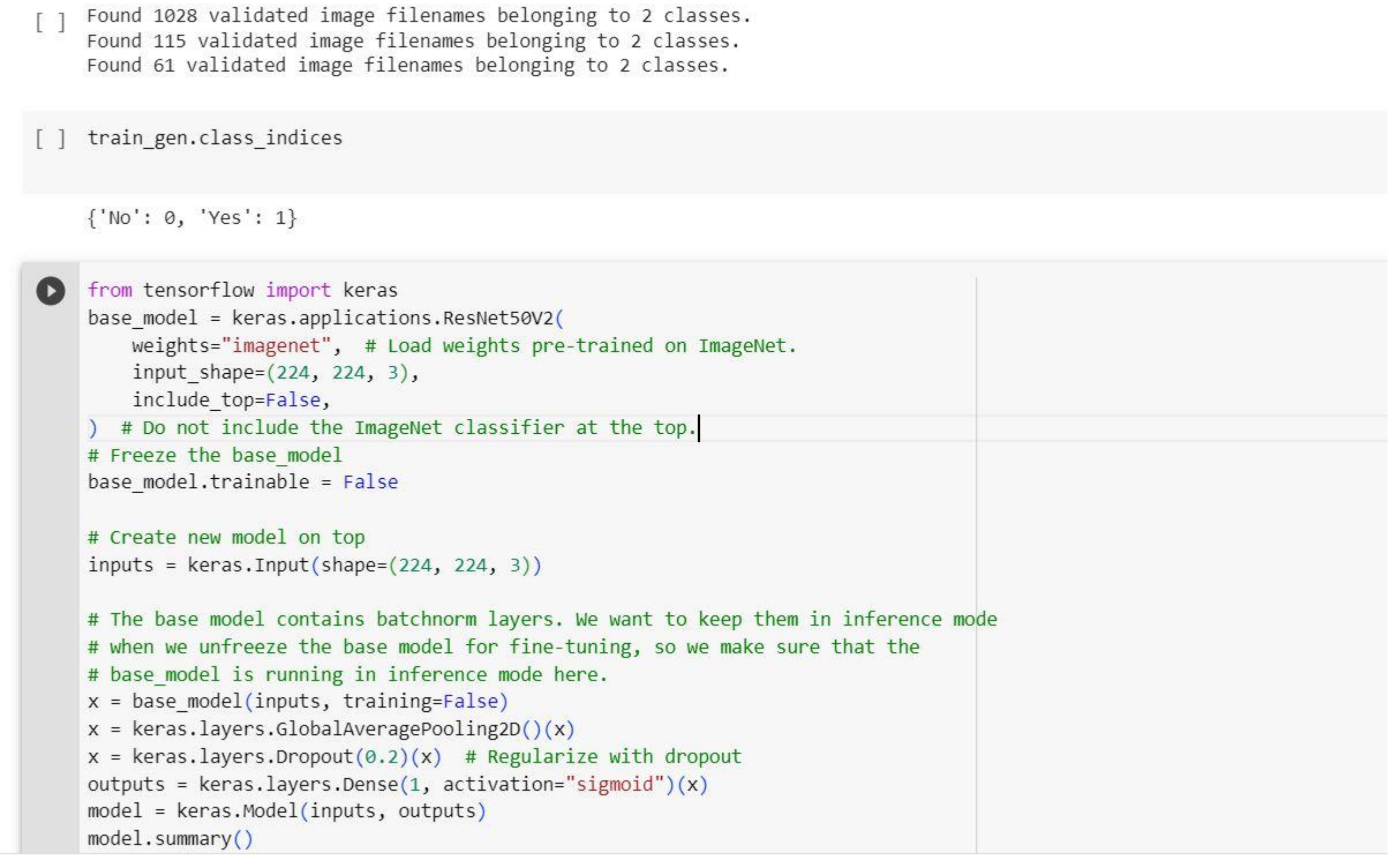




```
[ ] from sklearn.model_selection import train_test_split

    train, test = train_test_split(df, train_size=0.95, random_state=0)
    train_new, valid = train_test_split(train, train_size=0.90, random_state=0)

    print(f"train set shape: {train_new.shape}")
    print(f"test set shape: {test.shape}")
    print(f"validation set shape: {valid.shape}")

    train set shape: (1028, 2)
    test set shape: (61, 2)
    validation set shape: (115, 2)
```

```
[ ] train_datagen = ImageDataGenerator(rescale = 1./255.,rotation_range = 40, width_shift_range = 0.2, height_shift_range = 0.2,
                                   shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True, vertical_flip =True)
    test_datagen = ImageDataGenerator(rescale = 1.0/255.)
```

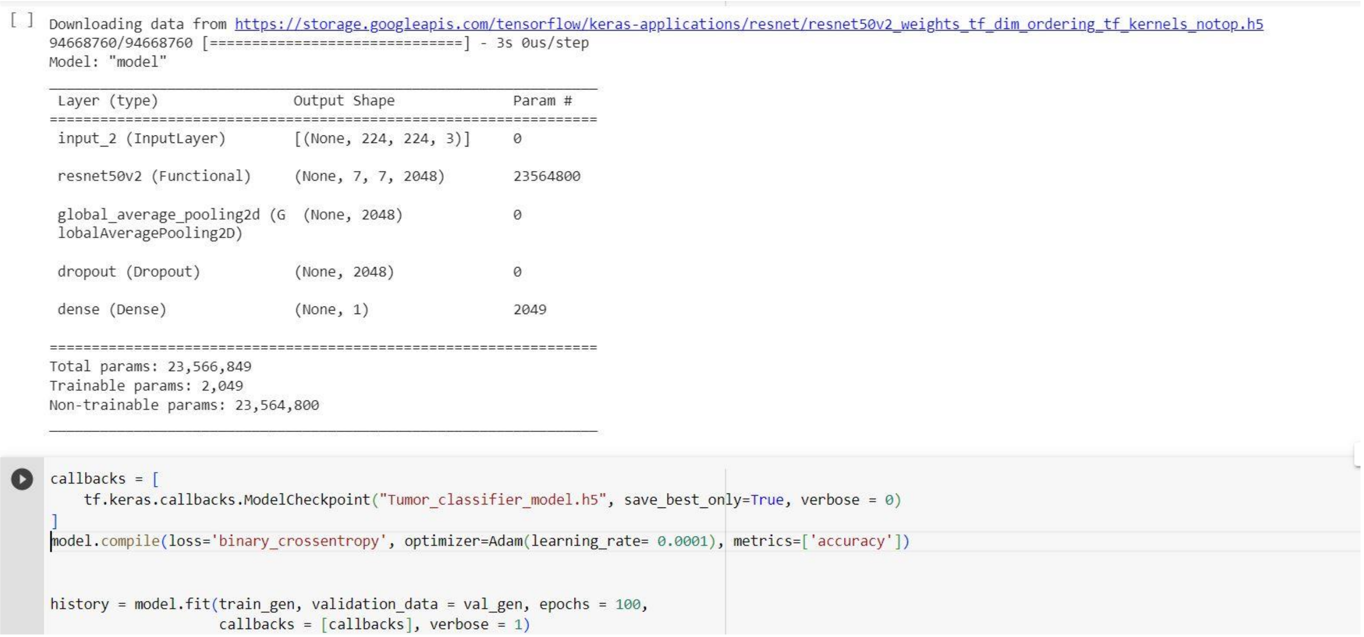
```
[ ] train_gen = train_datagen.flow_from_dataframe(dataframe = train_new,
                                                  x_col = 'filepaths', y_col ='labels',
                                                  target_size = (224,224), batch_size = 32,
                                                  class_mode = 'binary', shuffle = True)
    val_gen = train_datagen.flow_from_dataframe(valid,
                                                target_size=(224,224), x_col = 'filepaths', y_col ='labels',
                                                class_mode='binary',
                                                batch_size= 16, shuffle=True)
    test_gen = test_datagen.flow_from_dataframe(test,
                                                target_size = (224,224), x_col = 'filepaths', y_col ='labels',
                                                 class_mode = 'binary',
                                                batch_size = 16, shuffle = False)
```



```
[ ] Found 1028 validated image filenames belonging to 2 classes.
    Found 115 validated image filenames belonging to 2 classes.
    Found 61 validated image filenames belonging to 2 classes.
```

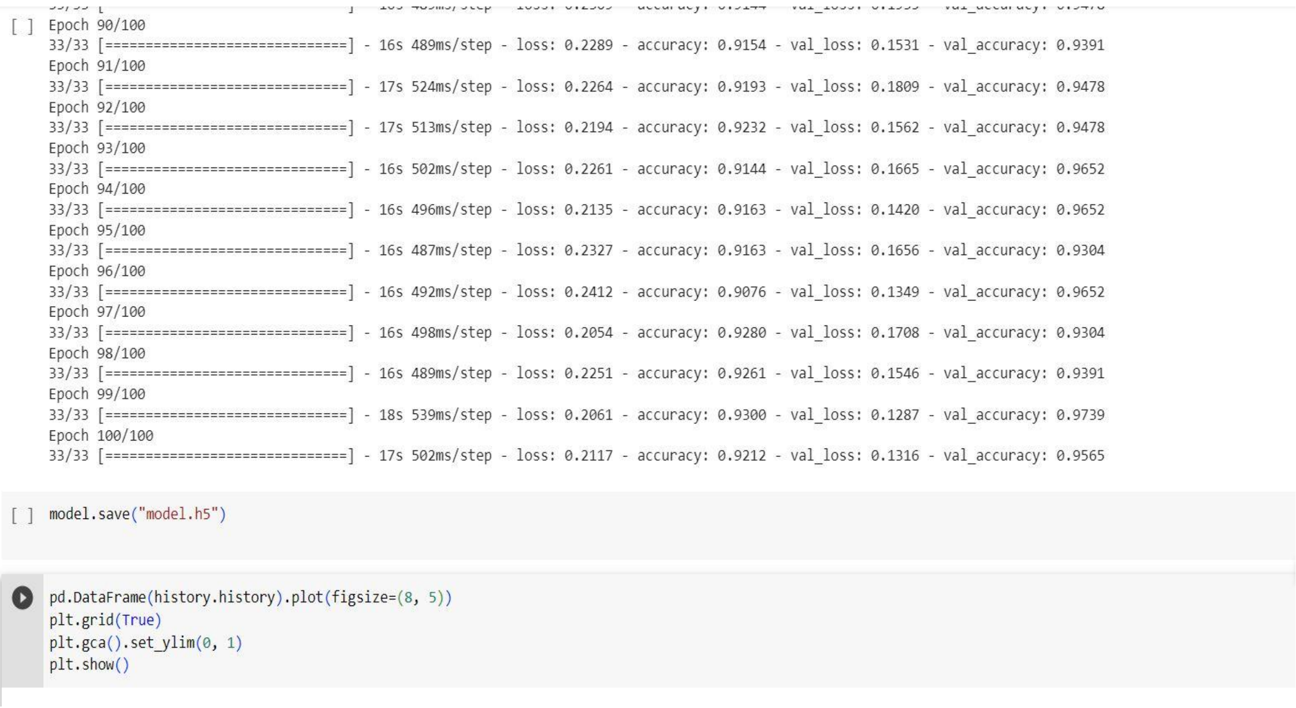
```
[ ] train_gen.class_indices

    {'No': 0, 'Yes': 1}
```

```
from tensorflow import keras
base_model = keras.applications.ResNet50V2(
    weights="imagenet",  # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False,
)  # Do not include the ImageNet classifier at the top.
# Freeze the base_model
base_model.trainable = False

# Create new model on top
inputs = keras.Input(shape=(224, 224, 3))

# The base model contains batchnorm layers. We want to keep them in inference mode
# when we unfreeze the base model for fine-tuning, so we make sure that the
# base_model is running in inference mode here.
x = base_model(inputs, training=False)
x = keras.layers.GlobalAveragePooling2D()(x)
x = keras.layers.Dropout(0.2)(x)  # Regularize with dropout
outputs = keras.layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.summary()
```



```
[ ] Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_kernels_notop.h5
    94668760/94668760 [==============================] - 3s 0us/step
    Model: "model"
    _________________________________________________________________
     Layer (type)                Output Shape              Param #
    =================================================================
     input_2 (InputLayer)        [(None, 224, 224, 3)]     0

     resnet50v2 (Functional)     (None, 7, 7, 2048)        23564800

     global_average_pooling2d (G  (None, 2048)             0
     lobalAveragePooling2D)

     dropout (Dropout)           (None, 2048)              0

     dense (Dense)               (None, 1)                 2049

    =================================================================
    Total params: 23,566,849
    Trainable params: 2,049
    Non-trainable params: 23,564,800
    _________________________________________________________________
```

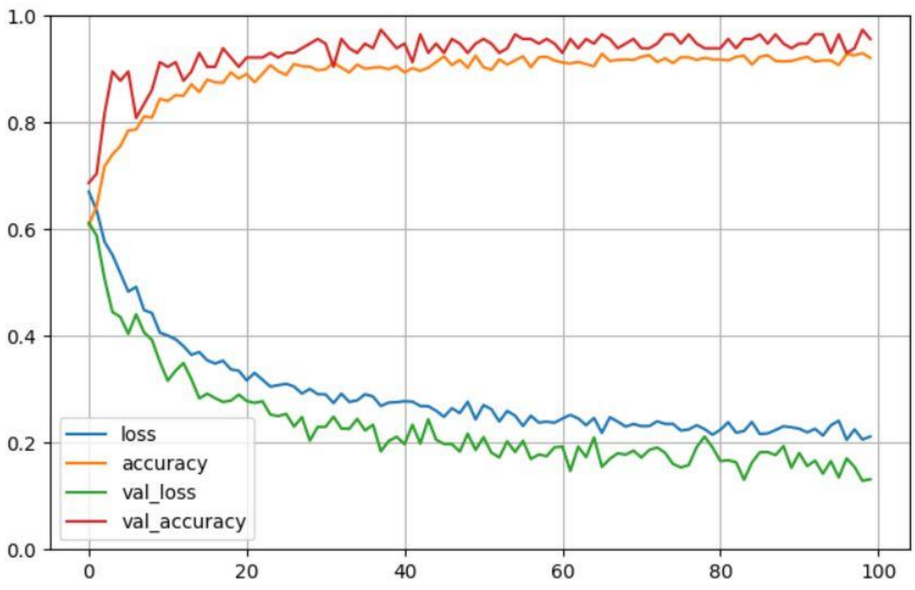
```
callbacks = [
    tf.keras.callbacks.ModelCheckpoint("Tumor_classifier_model.h5", save_best_only=True, verbose = 0)
]
model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate= 0.0001), metrics=['accuracy'])


history = model.fit(train_gen, validation_data = val_gen, epochs = 100,
                    callbacks = [callbacks], verbose = 1)
```

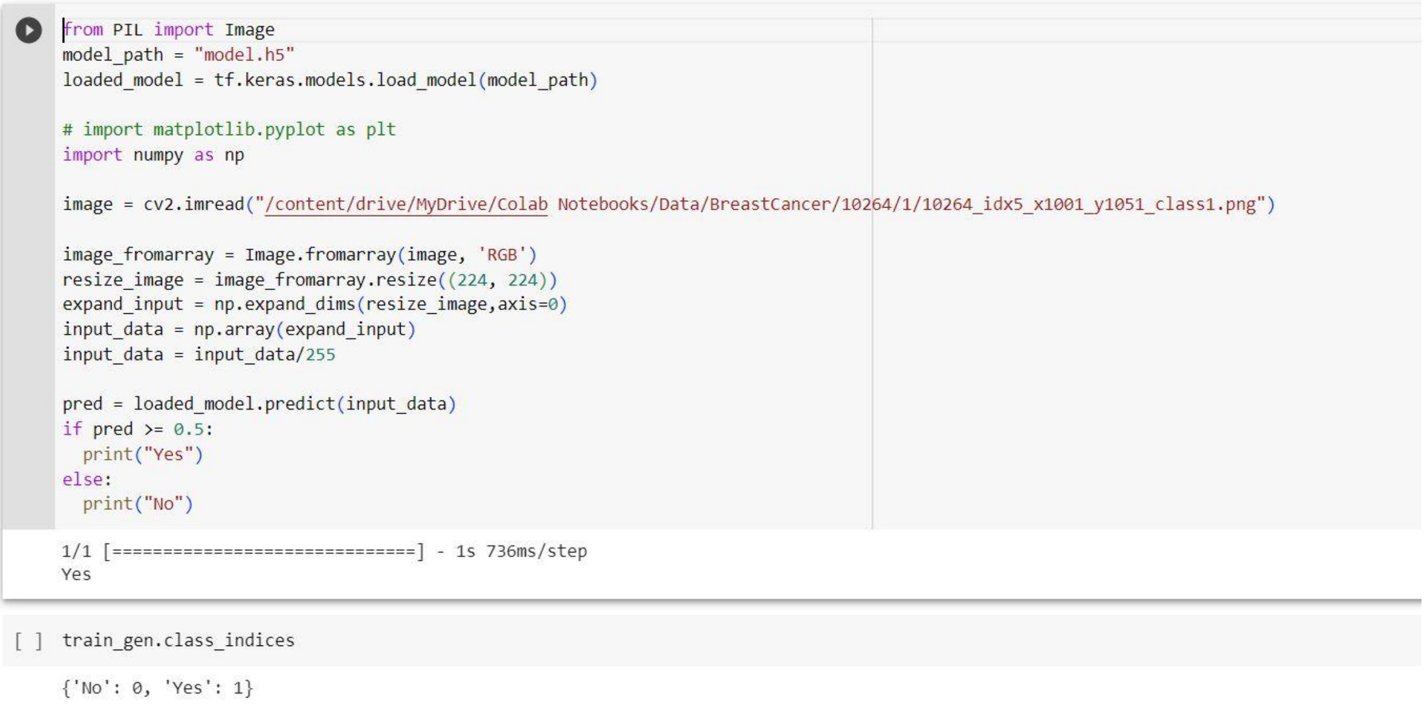


```
[ ] Epoch 90/100
    33/33 [==============================] - 16s 489ms/step - loss: 0.2289 - accuracy: 0.9154 - val_loss: 0.1531 - val_accuracy: 0.9391
    Epoch 91/100
    33/33 [==============================] - 17s 524ms/step - loss: 0.2264 - accuracy: 0.9193 - val_loss: 0.1809 - val_accuracy: 0.9478
    Epoch 92/100
    33/33 [==============================] - 17s 513ms/step - loss: 0.2194 - accuracy: 0.9232 - val_loss: 0.1562 - val_accuracy: 0.9478
    Epoch 93/100
    33/33 [==============================] - 16s 502ms/step - loss: 0.2261 - accuracy: 0.9144 - val_loss: 0.1665 - val_accuracy: 0.9652
    Epoch 94/100
    33/33 [==============================] - 16s 496ms/step - loss: 0.2135 - accuracy: 0.9163 - val_loss: 0.1420 - val_accuracy: 0.9652
    Epoch 95/100
    33/33 [==============================] - 16s 487ms/step - loss: 0.2327 - accuracy: 0.9163 - val_loss: 0.1656 - val_accuracy: 0.9304
    Epoch 96/100
    33/33 [==============================] - 16s 492ms/step - loss: 0.2412 - accuracy: 0.9076 - val_loss: 0.1349 - val_accuracy: 0.9652
    Epoch 97/100
    33/33 [==============================] - 16s 498ms/step - loss: 0.2054 - accuracy: 0.9280 - val_loss: 0.1708 - val_accuracy: 0.9304
    Epoch 98/100
    33/33 [==============================] - 16s 489ms/step - loss: 0.2251 - accuracy: 0.9261 - val_loss: 0.1546 - val_accuracy: 0.9391
    Epoch 99/100
    33/33 [==============================] - 18s 539ms/step - loss: 0.2061 - accuracy: 0.9300 - val_loss: 0.1287 - val_accuracy: 0.9739
    Epoch 100/100
    33/33 [==============================] - 17s 502ms/step - loss: 0.2117 - accuracy: 0.9212 - val_loss: 0.1316 - val_accuracy: 0.9565
```

```
[ ] model.save("model.h5")
```

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```





```
from PIL import Image
model_path = "model.h5"
loaded_model = tf.keras.models.load_model(model_path)

# import matplotlib.pyplot as plt
import numpy as np

image = cv2.imread("/content/drive/MyDrive/Colab Notebooks/Data/BreastCancer/10264/1/10264_idx5_x1001_y1051_class1.png")

image_fromarray = Image.fromarray(image, 'RGB')
resize_image = image_fromarray.resize((224, 224))
expand_input = np.expand_dims(resize_image,axis=0)
input_data = np.array(expand_input)
input_data = input_data/255

pred = loaded_model.predict(input_data)
if pred >= 0.5:
  print("Yes")
else:
  print("No")
```

```
1/1 [==============================] - 1s 736ms/step
Yes
```

```
[ ] train_gen.class_indices
```

```
{'No': 0, 'Yes': 1}
```