Importing Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing Data

```
data = pd.read_csv("Dataset\Breast Cancer\BreastCancer.csv")
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothr
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	

5 rows × 33 columns

Dropping the Id column

```
data = data.drop(['id', 'Unnamed: 32'], axis = 1)
data.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	M	17.99	10.38	122.80	1001.0	0.11840
1	M	20.57	17.77	132.90	1326.0	0.08474
2	M	19.69	21.25	130.00	1203.0	0.10960
3	M	11.42	20.38	77.58	386.1	0.14250
4	M	20.29	14.34	135.10	1297.0	0.10030

5 rows × 31 columns

Encoding diagnosis Malignant and Benign to 1s and 0s

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data['diagnosis'] = encoder.fit_transform(data['diagnosis'])
data.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
0	1	17.99	10.38	122.80	1001.0	0.11840
1	1	20.57	17.77	132.90	1326.0	0.08474
2	1	19.69	21.25	130.00	1203.0	0.10960
3	1	11.42	20.38	77.58	386.1	0.14250
4	1	20.29	14.34	135.10	1297.0	0.10030

5 rows × 31 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   diagnosis                569 non-null    int32
 1   radius_mean              569 non-null    float64
 2   texture_mean             569 non-null    float64
 3   perimeter_mean           569 non-null    float64
 4   area_mean                569 non-null    float64
 5   smoothness_mean          569 non-null    float64
 6   compactness_mean         569 non-null    float64
 7   concavity_mean           569 non-null    float64
 8   concave points_mean      569 non-null    float64
 9   symmetry_mean            569 non-null    float64
 10  fractal_dimension_mean   569 non-null    float64
 11  radius_se                569 non-null    float64
 12  texture_se               569 non-null    float64
 13  perimeter_se             569 non-null    float64
 14  area_se                  569 non-null    float64
 15  smoothness_se            569 non-null    float64
 16  compactness_se           569 non-null    float64
 17  concavity_se             569 non-null    float64
 18  concave points_se        569 non-null    float64
 19  symmetry_se              569 non-null    float64
 20  fractal_dimension_se     569 non-null    float64
 21  radius_worst             569 non-null    float64
 22  texture_worst            569 non-null    float64
 23  perimeter_worst          569 non-null    float64
 24  area_worst               569 non-null    float64
 25  smoothness_worst         569 non-null    float64
 26  compactness_worst        569 non-null    float64
 27  concavity_worst          569 non-null    float64
 28  concave points_worst     569 non-null    float64
 29  symmetry_worst           569 non-null    float64
 30  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

```
data.isnull().sum()
```

```
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

```
X = data.iloc[:, 1:] # Variables
Y = data.iloc[:, 0] # Result
```

Scaling the data values between 0 and 1

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Splitting the Data between Training and Testing

```
from sklearn.model_selection import train_test_split
train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size = 0.2, random_state = 0)
# test_size : amount of data used for testing
# random_state : Controls the shuffling applied tot he data before applying the split

test_X.shape
```

```
(114, 30)
```

Applying ANN to find the best values of hyperparameters

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV


# def Build_Classifier():
#     Classifier = Sequential()
#     Classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 30))
#     Classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu'))
#     Classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
#     # Input Layer : 30
#     # Hidden Layer 1 : 16
#     # Hidden Layer 2 : 16
#     # Output Layer : 1
#     # kernel_initializer : Initialising the Weights
#     Classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
#     # Compile : Defines the loss function, the optimizer and the metrics
#     # 1) Optimizer : algorithms or methods used to change the attributes such as weights and learning rate in order to reduce the losse
#     # 2) loss : To compute the quantity that a model should seek to minimize during training.
#     # 3) Metrices : A function that is used to judge the performance of your model
#     return Classifier

# Classifier = KerasClassifier(build_fn = Build_Classifier)

# parameters = {'batch_size' : [10, 20, 30, 40], 'epochs' : [50, 100, 150, 200]}
# # HyperParameters #
# # Batch : amount of data passed to the ANN at once.
# # Epochs : number of times the same data is pass over and over to the model. (i.e. learning of model)

# grid_search = GridSearchCV(estimator = Classifier, param_grid = parameters, scoring = 'accuracy', cv = 5, n_jobs = -1)
# # Grid search is a model hyperparameter optimization technique.
# # The GridSearchCV process will construct and evaluate one model for each combination of parameters.

# # 1) estimator : Either estimator needs to provide a score function, or scoring must be passed.
# # 2) param_grid : Dictionary of Hyperparameters to evaluate
# # 3) scoring : By default accuracy is the score that is optimized, but other scores can be specified in the score argument.
# # 4) n_jobs : Number of jobs to run parallel, By setting the n_jobs argument to -1, the process will use all cores on your machine
# # 5) cv : Cross validation is used to evaluate each individual model and the default of 3-fold cross validation is used

# grid_search = grid_search.fit(train_X, train_Y)
# best_parameters = grid_search.best_params_
# # Best combination of BatchSize and Epochs


# best_parameters
```

Applying ANN with the Best values of Hyperparameters

```
from keras.models import Sequential
from keras.layers import Dense


Classifier = Sequential()
Classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu', input_dim = 30))
Classifier.add(Dense(units = 16, kernel_initializer = 'uniform', activation = 'relu'))
```

```
Classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
Classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
history = Classifier.fit(train_X, train_Y, epochs = 50, batch_size = 10, validation_data = (test_X, test_Y))
```

```
Epoch 1/50
46/46 [==============================] - 1s 8ms/step - loss: 0.6894 - accuracy: 0.6374 - val_loss: 0.6851 - val_accuracy: 0.5877
Epoch 2/50
46/46 [==============================] - 0s 5ms/step - loss: 0.6640 - accuracy: 0.6440 - val_loss: 0.6402 - val_accuracy: 0.6316
Epoch 3/50
46/46 [==============================] - 0s 5ms/step - loss: 0.5780 - accuracy: 0.7692 - val_loss: 0.5353 - val_accuracy: 0.7281
Epoch 4/50
46/46 [==============================] - 0s 5ms/step - loss: 0.4542 - accuracy: 0.8571 - val_loss: 0.4153 - val_accuracy: 0.8333
Epoch 5/50
46/46 [==============================] - 0s 4ms/step - loss: 0.3328 - accuracy: 0.9099 - val_loss: 0.2902 - val_accuracy: 0.8947
Epoch 6/50
46/46 [==============================] - 0s 4ms/step - loss: 0.2509 - accuracy: 0.9297 - val_loss: 0.2354 - val_accuracy: 0.9123
Epoch 7/50
46/46 [==============================] - 0s 3ms/step - loss: 0.2070 - accuracy: 0.9385 - val_loss: 0.2108 - val_accuracy: 0.9298
Epoch 8/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1791 - accuracy: 0.9407 - val_loss: 0.1945 - val_accuracy: 0.8947
Epoch 9/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1601 - accuracy: 0.9407 - val_loss: 0.1806 - val_accuracy: 0.9035
Epoch 10/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1532 - accuracy: 0.9473 - val_loss: 0.1715 - val_accuracy: 0.9211
Epoch 11/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1394 - accuracy: 0.9363 - val_loss: 0.1643 - val_accuracy: 0.9298
Epoch 12/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1284 - accuracy: 0.9495 - val_loss: 0.1573 - val_accuracy: 0.9298
Epoch 13/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1163 - accuracy: 0.9626 - val_loss: 0.1583 - val_accuracy: 0.9298
Epoch 14/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1160 - accuracy: 0.9626 - val_loss: 0.1497 - val_accuracy: 0.9123
Epoch 15/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1077 - accuracy: 0.9736 - val_loss: 0.1405 - val_accuracy: 0.9474
Epoch 16/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1047 - accuracy: 0.9648 - val_loss: 0.1366 - val_accuracy: 0.9561
Epoch 17/50
46/46 [==============================] - 0s 3ms/step - loss: 0.1011 - accuracy: 0.9604 - val_loss: 0.1393 - val_accuracy: 0.9211
Epoch 18/50
46/46 [==============================] - 0s 4ms/step - loss: 0.0941 - accuracy: 0.9736 - val_loss: 0.1309 - val_accuracy: 0.9474
Epoch 19/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0915 - accuracy: 0.9648 - val_loss: 0.1313 - val_accuracy: 0.9298
Epoch 20/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0870 - accuracy: 0.9780 - val_loss: 0.1381 - val_accuracy: 0.9298
Epoch 21/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0837 - accuracy: 0.9824 - val_loss: 0.1257 - val_accuracy: 0.9737
Epoch 22/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0830 - accuracy: 0.9714 - val_loss: 0.1245 - val_accuracy: 0.9737
Epoch 23/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0817 - accuracy: 0.9736 - val_loss: 0.1276 - val_accuracy: 0.9649
Epoch 24/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0806 - accuracy: 0.9758 - val_loss: 0.1226 - val_accuracy: 0.9737
Epoch 25/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0752 - accuracy: 0.9780 - val_loss: 0.1249 - val_accuracy: 0.9737
Epoch 26/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0787 - accuracy: 0.9714 - val_loss: 0.1248 - val_accuracy: 0.9737
Epoch 27/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0775 - accuracy: 0.9780 - val_loss: 0.1210 - val_accuracy: 0.9737
Epoch 28/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0711 - accuracy: 0.9824 - val_loss: 0.1209 - val_accuracy: 0.9737
Epoch 29/50
46/46 [==============================] - 0s 3ms/step - loss: 0.0696 - accuracy: 0.9802 - val_loss: 0.1218 - val_accuracy: 0.9649
```

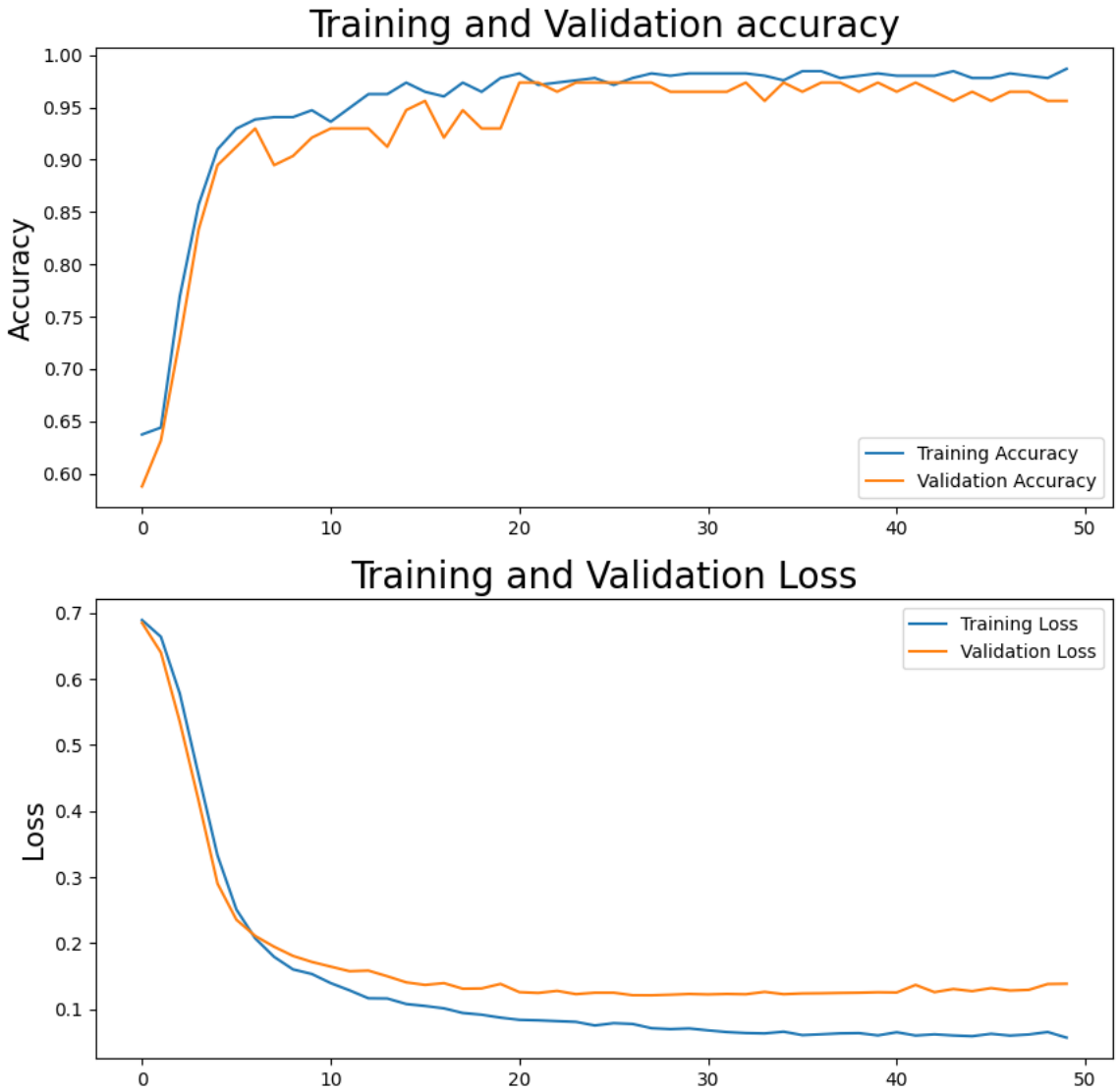
```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Ploting Training & Validation (Accuracy and Loss)

```
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(accuracy))

fig = plt.figure(figsize = (10, 10), edgecolor = 'Black')
ax1 = fig.add_subplot(2, 1, 1)
ax2 = fig.add_subplot(2, 1, 2)

ax1.plot(epochs, accuracy, label = 'Training Accuracy')
ax1.plot(epochs, val_accuracy, label = 'Validation Accuracy')
ax1.set_title("Training and Validation accuracy", fontsize = 20)
ax1.set_ylabel("Accuracy", fontsize = 15)
```

```
ax1.legend()

ax2.plot(epochs, loss, label = 'Training Loss')
ax2.plot(epochs, val_loss, label = 'Validation Loss')
ax2.set_title("Training and Validation Loss", fontsize = 20)
ax2.set_ylabel("Loss", fontsize = 15)
ax2.legend()

plt.show()
```



Prediction on Test Data

```
Y_pred = Classifier.predict(test_X)
# Probability showing the individual having breast cancer (Malignant)
# Probability over 50% would indicate the presence of breast cancer

Y_pred = (Y_pred > 0.5) # Converting the probabilities into 'True' and 'False'
```

```
4/4 [==============================] - 0s 2ms/step
```

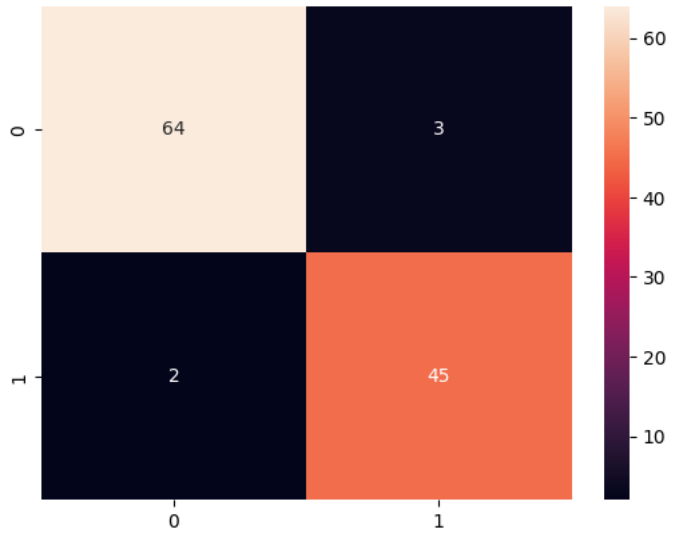
Confusion matrix and Heatmap for visualizing the Accuracy of ANN on test data

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(test_Y, Y_pred)
cm
```

```
array([[64,  3],
       [ 2, 45]], dtype=int64)
```

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

```
<AxesSubplot: >
```



Accuracy

```
print("Accuracy Score:", round((cm[0, 0]+cm[1, 1])/(cm[0, 0]+cm[0, 1]+cm[1, 0]+cm[1, 1])*100)) #, 2
```

```
Accuracy Score: 96
```