Muhammad Afiq Husyairi (2018769)

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

```
df = pd.read_csv('breast_cancer_Data.csv')
df.head()
```

	id	diagnosis	Radius_mean	Texture_mean	perimeter_mean	area_mean	smoo
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	21.77	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 × 32 columns

```
df.describe()
```

	id	Radius_mean	Texture_mean	perimeter_mean	area_mean	smooth
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	14.127292	19.296678	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301816	24.298981	351.914129	
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	
50%	9.060240e+05	13.370000	18.870000	86.240000	551.100000	
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	

8 rows × 31 columns

```
df.shape
```

```
(569, 32)
```

Drop Null Values

```
print(" \nCount total NaN at each column in a DataFrame : \n\n",df.isnull().sum())

df = df.dropna()
df.shape
```

```
Count total NaN at each column in a DataFrame :

 id                        0
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
(569, 32)
```

No Null values

Drop Duplicated Values

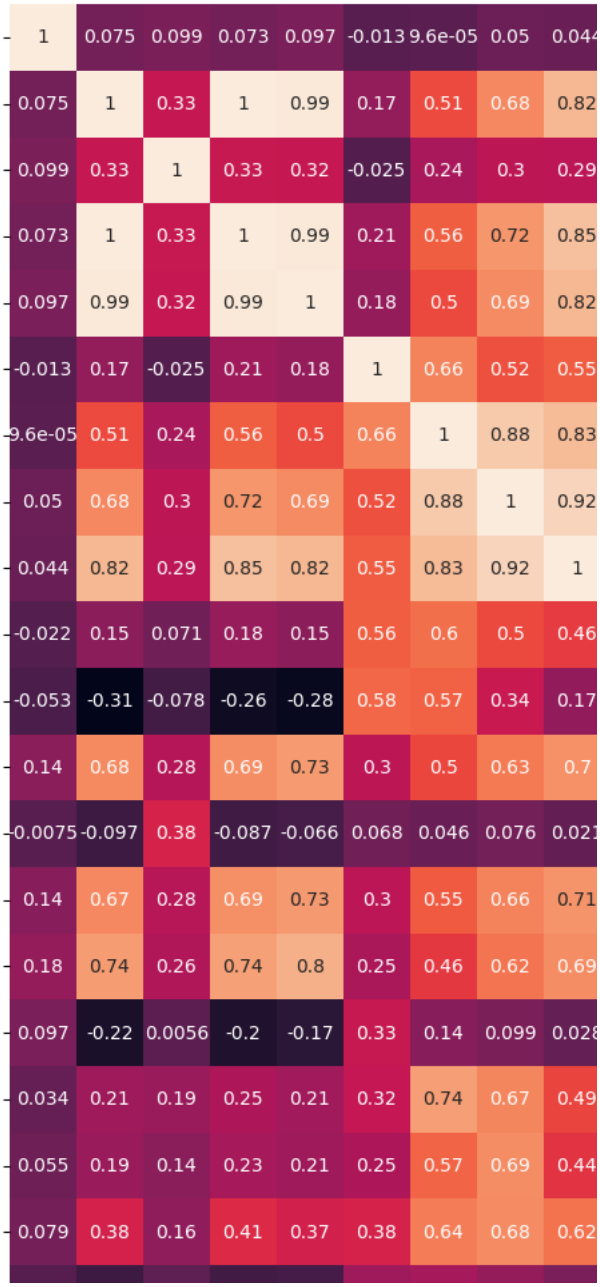
```
df.drop_duplicates(inplace=True)
df.shape
```

```
(569, 32)
```

No duplicates

```
plt.figure(figsize=(25,20))
sns.heatmap(data=df.corr(), annot=True)
```

```
<Axes: >
```



Convert the diagnosis into 0 and 1

```
from sklearn.preprocessing import LabelEncoder

lb = LabelEncoder()
df['diagnosis'] = lb.fit_transform(df['diagnosis'])

df['diagnosis'].unique()
```

```
array([1, 0])
```

```
dataset = df.values
dataset.shape
```

```
(569, 32)
```

Convert from DataFrame to numpy array

```
# load all columns in the array except number and diagnosis
X = dataset[:,2:31]
X.shape
```

```
(569, 29)
```

```
y = dataset[:,1]
y.shape
y[:5]
```

```
array([1., 1., 1., 1., 1.])
```

Normalize the data

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X)
```

```
print(scaler)
```

```
StandardScaler()
```

```
X_standardized = scaler.transform(X)

data = pd.DataFrame(X_standardized)
data.describe()
```

	0	1	2	3	4	
count	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.690000e+02	5.69
mean	-1.373633e-16	-3.434082e-17	-1.248757e-16	-2.185325e-16	-8.366672e-16	1.8
std	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00	1.000880e+00	1.00
min	-2.029648e+00	-2.230480e+00	-1.984504e+00	-1.454443e+00	-3.112085e+00	-1.61
25%	-6.893853e-01	-7.274670e-01	-6.919555e-01	-6.671955e-01	-7.109628e-01	-7.4
50%	-2.150816e-01	-9.927291e-02	-2.359800e-01	-2.951869e-01	-3.489108e-02	-2.2
75%	4.693926e-01	5.824340e-01	4.996769e-01	3.635073e-01	6.361990e-01	4.9
max	3.971288e+00	4.649409e+00	3.976130e+00	5.250529e+00	4.770911e+00	4.56

8 rows × 29 columns

```
from sklearn.model_selection import GridSearchCV, KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.optimizers import Adam
```

Build a neural network with 29 input layer, 16 neuron on the first hidden layer, another hidden layer with 8 neurons, and 1 output. All using relu activation function

```
# Define a random seed
seed = 6
np.random.seed(seed)

# Start defining the model
def create_model():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim = 29, kernel_initializer='normal', activation='relu'))
    model.add(Dense(8, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, activation='sigmoid'))

    # compile the model
    adam = Adam(lr = 0.01)
    model.compile(loss = 'binary_crossentropy', optimizer = adam, metrics = ['accuracy'])
    return model

# create the model
model = KerasClassifier(build_fn = create_model, verbose = 1)

# define the grid search parameters
batch_size = [10, 20, 40]
epochs = [10, 50, 100]

# make a dictionary of the grid search parameters
param_grid = dict(batch_size=batch_size, epochs=epochs)
```

```
# build and fit the GridSearchCV
grid = GridSearchCV(estimator = model, param_grid = param_grid, cv = KFold(random_state=None), verbose = 10)
grid_results = grid.fit(X_standardized, y)

# summarize the results
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Epoch 89/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0174 - accuracy: 0.9956
Epoch 90/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0171 - accuracy: 0.9956
Epoch 91/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0168 - accuracy: 0.9956
Epoch 92/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0166 - accuracy: 0.9956
Epoch 93/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0163 - accuracy: 0.9956
Epoch 94/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0159 - accuracy: 0.9956
Epoch 95/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0157 - accuracy: 0.9956
Epoch 96/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0154 - accuracy: 0.9956
Epoch 97/100
12/12 [==============================] - 0s 3ms/step - loss: 0.0151 - accuracy: 0.9956
Epoch 98/100
12/12 [==============================] - 0s 5ms/step - loss: 0.0149 - accuracy: 0.9956
Epoch 99/100
12/12 [==============================] - 0s 5ms/step - loss: 0.0147 - accuracy: 0.9956
Epoch 100/100
12/12 [==============================] - 0s 4ms/step - loss: 0.0144 - accuracy: 0.9956
3/3 [==============================] - 0s 5ms/step - loss: 0.0875 - accuracy: 0.9558
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 5/5; 9/9] END ....batch_size=40, epochs=100;, score=0.956 total time=   5.8s
Epoch 1/10
57/57 [==============================] - 1s 3ms/step - loss: 0.6008 - accuracy: 0.9121
Epoch 2/10
57/57 [==============================] - 0s 3ms/step - loss: 0.3297 - accuracy: 0.9438
Epoch 3/10
57/57 [==============================] - 0s 4ms/step - loss: 0.1764 - accuracy: 0.9490
Epoch 4/10
57/57 [==============================] - 0s 3ms/step - loss: 0.1196 - accuracy: 0.9719
Epoch 5/10
57/57 [==============================] - 0s 3ms/step - loss: 0.0952 - accuracy: 0.9754
Epoch 6/10
57/57 [==============================] - 0s 2ms/step - loss: 0.0822 - accuracy: 0.9789
Epoch 7/10
57/57 [==============================] - 0s 2ms/step - loss: 0.0743 - accuracy: 0.9824
Epoch 8/10
57/57 [==============================] - 0s 2ms/step - loss: 0.0691 - accuracy: 0.9824
Epoch 9/10
57/57 [==============================] - 0s 2ms/step - loss: 0.0646 - accuracy: 0.9859
Epoch 10/10
57/57 [==============================] - 0s 2ms/step - loss: 0.0609 - accuracy: 0.9859
Best: 0.9754230737686157, using {'batch_size': 10, 'epochs': 10}
0.9754230737686157 (0.012873061611428028) with: {'batch_size': 10, 'epochs': 10}
0.9736376285552979 (0.016643807823830226) with: {'batch_size': 10, 'epochs': 50}
0.9736065626144409 (0.015779015663304582) with: {'batch_size': 10, 'epochs': 100}
0.9736686825752259 (0.014659717026799518) with: {'batch_size': 20, 'epochs': 10}
0.970128858089447 (0.01805369433335564) with: {'batch_size': 20, 'epochs': 50}
0.9753764748573304 (0.014082129144334399) with: {'batch_size': 20, 'epochs': 100}
0.9648812294006348 (0.016610922303183322) with: {'batch_size': 40, 'epochs': 10}
0.9718987703323364 (0.016990291752306482) with: {'batch_size': 40, 'epochs': 50}
0.9700978040695191 (0.01633706373959022) with: {'batch_size': 40, 'epochs': 100}
```

```
# Define a random seed
seed = 6
np.random.seed(seed)

# Start defining the model
def create_model(neuron1, neuron2):
    # create model
    model = Sequential()
    model.add(Dense(neuron1, input_dim=29, kernel_initializer='uniform', activation='linear'))
    model.add(Dense(neuron2, kernel_initializer='uniform', activation='linear'))
    model.add(Dense(1, activation='sigmoid'))

    # compile the model
    adam = Adam(lr=0.001)
    model.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

```
    return model

# create the model
model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=20, verbose=0)

# define the grid search parameters
neuron1 = [4, 8, 16]
neuron2 = [2, 4, 8]

# make a dictionary of the grid search parameters
param_grid = dict(neuron1=neuron1, neuron2=neuron2)

# build and fit the GridSearchCV
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=KFold(), refit=True, verbose=10)
grid_results = grid.fit(X_standardized, y)

# summarize the results
print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
means = grid_results.cv_results_['mean_test_score']
stds = grid_results.cv_results_['std_test_score']
params = grid_results.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
<ipython-input-18-7b0b0152be14>:19: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb
  model = KerasClassifier(build_fn=create_model, epochs=100, batch_size=20, verbose=0)
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV 1/5; 1/9] START neuron1=4, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 1/5; 1/9] END .........neuron1=4, neuron2=2;, score=0.974 total time=   9.7s
[CV 2/5; 1/9] START neuron1=4, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 2/5; 1/9] END .........neuron1=4, neuron2=2;, score=0.947 total time=   6.9s
[CV 3/5; 1/9] START neuron1=4, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 3/5; 1/9] END .........neuron1=4, neuron2=2;, score=0.982 total time=   5.0s
[CV 4/5; 1/9] START neuron1=4, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 4/5; 1/9] END .........neuron1=4, neuron2=2;, score=0.974 total time=   6.4s
[CV 5/5; 1/9] START neuron1=4, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 5/5; 1/9] END .........neuron1=4, neuron2=2;, score=0.982 total time=   6.3s
[CV 1/5; 2/9] START neuron1=4, neuron2=4.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 1/5; 2/9] END .........neuron1=4, neuron2=4;, score=0.974 total time=   6.2s
[CV 2/5; 2/9] START neuron1=4, neuron2=4.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 2/5; 2/9] END .........neuron1=4, neuron2=4;, score=0.947 total time=   6.3s
[CV 3/5; 2/9] START neuron1=4, neuron2=4.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 3/5; 2/9] END .........neuron1=4, neuron2=4;, score=0.982 total time=   5.6s
[CV 4/5; 2/9] START neuron1=4, neuron2=4.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 4/5; 2/9] END .........neuron1=4, neuron2=4;, score=0.974 total time=   5.0s
[CV 5/5; 2/9] START neuron1=4, neuron2=4.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 5/5; 2/9] END .........neuron1=4, neuron2=4;, score=0.982 total time=   6.3s
[CV 1/5; 3/9] START neuron1=4, neuron2=8.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 1/5; 3/9] END .........neuron1=4, neuron2=8;, score=0.965 total time=   5.1s
[CV 2/5; 3/9] START neuron1=4, neuron2=8.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 2/5; 3/9] END .........neuron1=4, neuron2=8;, score=0.947 total time=   6.3s
[CV 3/5; 3/9] START neuron1=4, neuron2=8.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 3/5; 3/9] END .........neuron1=4, neuron2=8;, score=0.982 total time=   5.6s
[CV 4/5; 3/9] START neuron1=4, neuron2=8.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 4/5; 3/9] END .........neuron1=4, neuron2=8;, score=0.974 total time=   4.9s
[CV 5/5; 3/9] START neuron1=4, neuron2=8.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 5/5; 3/9] END .........neuron1=4, neuron2=8;, score=0.982 total time=   6.9s
[CV 1/5; 4/9] START neuron1=8, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 1/5; 4/9] END .........neuron1=8, neuron2=2;, score=0.974 total time=   5.0s
[CV 2/5; 4/9] START neuron1=8, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
[CV 2/5; 4/9] END .........neuron1=8, neuron2=2;, score=0.947 total time=   7.0s
[CV 3/5; 4/9] START neuron1=8, neuron2=2.....................................
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
```

```
y_pred = grid.predict(X_standardized)

y_pred.shape
```

```
18/18 [==============================] - 0s 1ms/step
(569, 1)
```

```
from sklearn.metrics import classification_report, accuracy_score
```

```
print(accuracy_score(y, y_pred))
```

```
0.9876977152899824
```

```
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.99      0.99       357
         1.0       0.99      0.98      0.98       212

    accuracy                           0.99       569
   macro avg       0.99      0.99      0.99       569
weighted avg       0.99      0.99      0.99       569
```