```
#import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

```
from google.colab import files
upload=files.upload()
```

Browse... No files selected. Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Cancer.csv to Cancer (1).csv
```

```
# read the data set
data=pd.read_csv("Cancer.csv")
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sm
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	

```
# get basic info of data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
```

✓ 0s completed at 11:47 PM ● ×

```
 20  symmetry_se              569 non-null   float64
 21  fractal_dimension_se     569 non-null   float64
 22  radius_worst             569 non-null   float64
 23  texture_worst            569 non-null   float64
 24  perimeter_worst          569 non-null   float64
 25  area_worst               569 non-null   float64
 26  smoothness_worst         569 non-null   float64
 27  compactness_worst        569 non-null   float64
 28  concavity_worst          569 non-null   float64
 29  concave points_worst     569 non-null   float64
 30  symmetry_worst           569 non-null   float64
 31  fractal_dimension_worst  569 non-null   float64
 32  Unnamed: 32              0 non-null     float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
# check missing values in data
data.isnull().sum()
```

```
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
```

```
# Set index to ID
data.set_index("id", inplace=True)
```

```
#Most of the columns are Numerical and only one categorical columns "diagnosis"
data.diagnosis.value_counts()
```

```
    B    357
    M    212
    Name: diagnosis, dtype: int64
```

```
data.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
count	569.000000	569.000000	569.000000	569.000000	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360
std	3.524049	4.301036	24.298981	351.914129	0.014064
min	6.981000	9.710000	43.790000	143.500000	0.052630
25%	11.700000	16.170000	75.170000	420.300000	0.086370
50%	13.370000	18.840000	86.240000	551.100000	0.095870
75%	15.780000	21.800000	104.100000	782.700000	0.105300
max	28.110000	39.280000	188.500000	2501.000000	0.163400

```
from sklearn.preprocessing import LabelEncoder
label_encoder=LabelEncoder()
```

```
#diagnosis
data["diagnosis"]=label_encoder.fit_transform(data["diagnosis"])
```

```
x.shape
```

```
(569, 30)
```

```
y= data.iloc[:,0]
y.head()
```

```
id
842302      1
842517      1
84300903    1
84348301    1
84358402    1
Name: diagnosis, dtype: int64
```

Feature Selection using RandomForestClassifier

```
# lets select important features first using Random Forset classifier.
from sklearn.ensemble import RandomForestClassifier
rf_model= RandomForestClassifier(n_estimators=100)
```

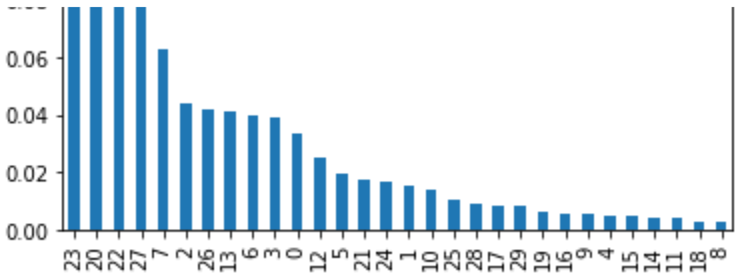
```
#now fit the model
rf_model.fit(x,y)

#now get important features
important_feature= rf_model.feature_importances_
```

```
# now lets visualize the important features
df=pd.DataFrame({"Features":pd.DataFrame(x).columns,"Importance":important_feature}
```

	features	importance
23	area_worst	0.144693
20	radius_worst	0.132321
22	perimeter_worst	0.127033
27	concave points_worst	0.106479
7	concave points_mean	0.062790
2	perimeter_mean	0.043717
26	concavity_worst	0.041974
13	area_se	0.040958
6	concavity_mean	0.039632
3	area_mean	0.038940
0	radius_mean	0.033357
12	perimeter_se	0.024902

```
#Creating final dataframe with important features
final_df=x[['area_worst','radius_worst','concave points_worst','perimeter_worst','c
            'area_mean','area_se','radius_mean','concavity_worst','compactness_wors
```



```
#splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(final_df,y, test_size= 0.2, ran
```

```
# Now we will scale the value to as ANN is sensitive to magnitude of values.
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
```

```
X_train =scaler.fit_transform(X_train)
```

```
#compile the model
model1.compile(optimizer="Adamax", loss="binary_crossentropy", metrics=["accuracy"]
```

```
# lets fit the model
model1_history=model1.fit(X_train,Y_train, validation_split=0.2,batch_size=10,epoch
```

```
    Epoch 1/100
    37/37 [==============================] - 1s 16ms/step - loss: 1.2567 - accurac
    Epoch 2/100
    37/37 [==============================] - 0s 3ms/step - loss: 1.1789 - accuracy
    Epoch 3/100
    37/37 [==============================] - 0s 3ms/step - loss: 0.9322 - accuracy
    37/37 [==============================] - 0s 3ms/step - loss: 0.1655 - accuracy
    Epoch 28/100
    37/37 [==============================] - 0s 3ms/step - loss: 0.1771 - accuracy
    Epoch 29/100
    37/37 [==============================] - 0s 3ms/step - loss: 0.1957 - accuracy
```

```
# now lets visualize the Loss and Val_Loss to check how the loss value is optimized
```

Epochs