- I have downloaded this Breast Cancer datasets from UCI machine learning repository maintained by the University of California, Irvine. The dataset contains 569 samples of malignant and benign tumor cells.
- The first two columns in the dataset store the unique ID numbers of the samples and the corresponding diagnosis (M=malignant, B=benign), respectively.
- The columns 3-32 contain 30 real-value features that have been computed from digitized images of the cell nuclei, which can be used to build a model to predict whether a tumor is benign or malignant.

1= Malignant (Cancerous) - Present (M) 0= Benign (Not Cancerous) -Absent (B)

- Ten real-valued features are computed for each cell nucleus:

  radius (mean of distances from center to points on the perimeter)

  texture (standard deviation of gray-scale values)

  perimeter

  area

  smoothness (local variation in radius lengths)

  compactness (perimeter^2 / area - 1.0)

  concavity (severity of concave portions of the contour)

  concave points (number of concave portions of the contour)

  symmetry

  fractal dimension ("coastline approximation" - 1)

- The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Loading the data
df=pd.read_csv('data.csv')
```

```
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness
0	842302	M	17.99	10.38	122.80	1001.0	0
1	842517	M	20.57	17.77	132.90	1326.0	0
2	84300903	M	19.69	21.25	130.00	1203.0	0
3	84348301	M	11.42	20.38	77.58	386.1	0
4	84358402	M	20.29	14.34	135.10	1297.0	0

5 rows × 33 columns

The last column named "Unaname: 32" seems like an erronous coloumn in our dataset. We might probably just drop it.

Most of the columns seem to have a numeric entry. This would save our time from mapping the variables.

The ID column would not help us contributing to predict about the cancer. We might as well drop it.

```
df.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
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 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
```

Only the 'diagnosis' column, which we have to predict is of object datatype.

There's only ID column of int type. We will probably drop it anyway.

There are a total of 31 columns which are of float datatype

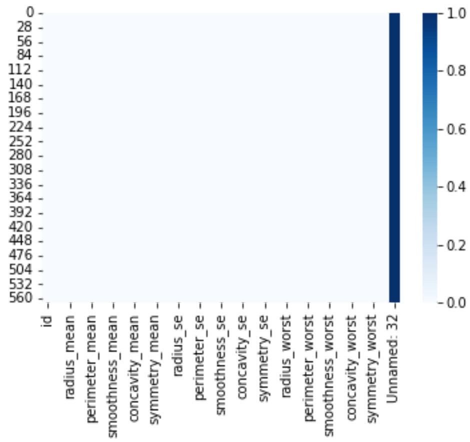
```
#To check the null values from the dataset
df.isnull().sum()
```

```
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
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
```

```
fractal_dimension_worst      0
Unnamed: 32                569
dtype: int64
```

```
#Here i have used heatmap to show the null values visually.
sns.heatmap(df.isnull(),cmap='Blues')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5056d89d0>
```




```
#Here i have dropped the Unnamed: 32 column because it contains maximum null values.
df.drop('Unnamed: 32', axis=1, inplace=True)
```

```
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
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	

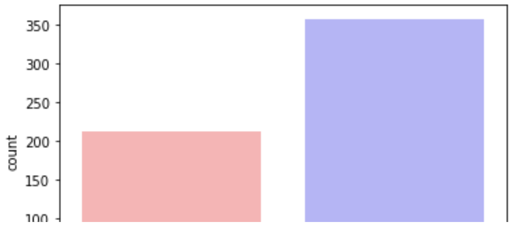
569 rows × 32 columns

```
#Just to check whether our data is balanced or not i have used value_counts.
df["diagnosis"].value_counts()
```

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

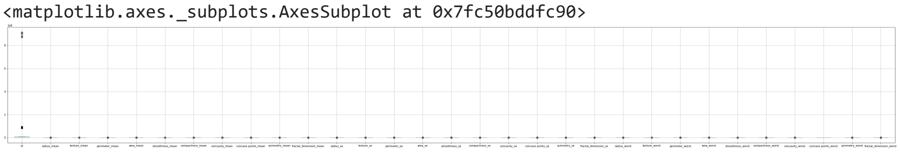
```
#Visual representation of the target column
cnt_plot= sns.countplot(df["diagnosis"],label="Count",palette='bwr_r')
B, M = df["diagnosis"].value_counts()
print('Number of Benign: ',B)
print('Number of Malignant : ',M)
```

```
Number of Benign:  357
Number of Malignant :  212
```



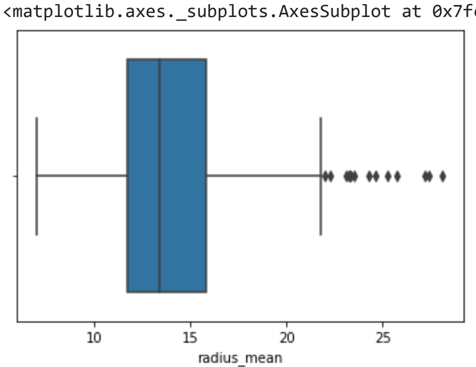

```
#We would need to eliminate the outliers so that it does not affects our model's accuracy. Let us see if there are any outliers present in th
plt.figure(figsize=(60,8))
df.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc50bddfc90>
```



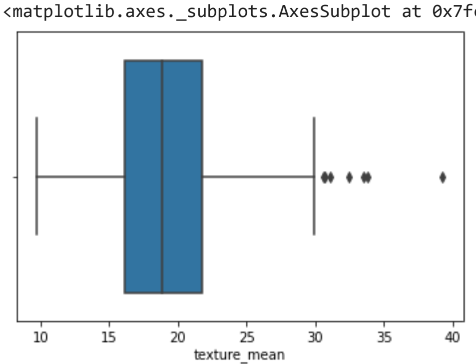
```
sns.boxplot(df['radius_mean'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc50b9758d0>
```




```
sns.boxplot(df['texture_mean'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc50b9a5bd0>
```

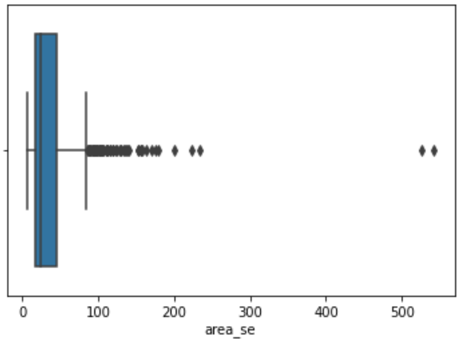



```
sns.boxplot(df['area_worst'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc50b8c3890>

```
sns.boxplot(df['area_se'])
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc50b834990>




```
df = df[(df['radius_mean'] < 23) & (df['texture_mean'] < 35) & (df['area_worst'] < 2300) & (df['area_se'] < 150)]
```

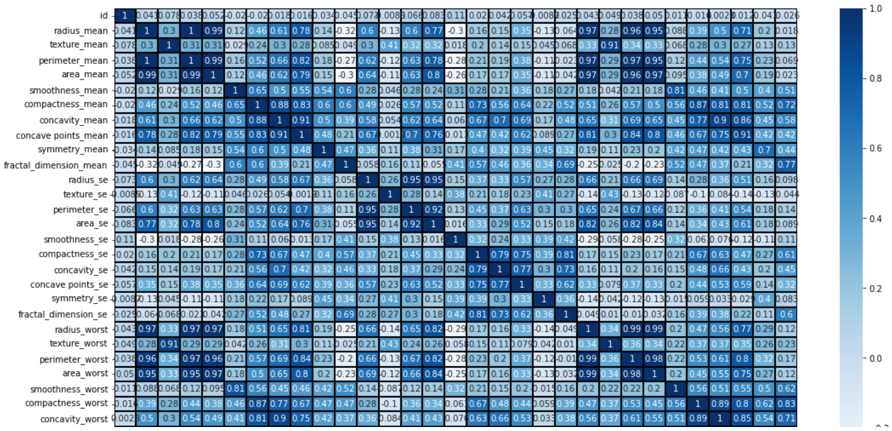
```
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sm
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	843786	M	12.45	15.70	82.57	477.1	
...	...	...	...	...	...	...	
563	926125	M	20.92	25.09	143.00	1347.0	
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	

546 rows × 32 columns

```
plt.figure(figsize = (18, 10))
sns.heatmap(df.corr(),cmap='Blues',linewidths=1,linecolor='black',annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc50b855a10>
```




```
x=df.drop(['id','diagnosis'],axis=1)
x
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
1	20.57	17.77	132.90	1326.0	0.08474	0.(
2	19.69	21.25	130.00	1203.0	0.10960	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.1
5	12.45	15.70	82.57	477.1	0.12780	0.1
...	...	...	...	...	...	
563	20.92	25.09	143.00	1347.0	0.10990	0.2
565	20.13	28.25	131.20	1261.0	0.09780	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.(

546 rows × 30 columns

```
y=df.diagnosis.replace({'B':0,'M':1})
y
```

```
1      1
2      1
3      1
4      1
5      1
      ..
563    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 546, dtype: int64
```

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()

xtrain=sc.fit_transform(xtrain)
xtest=sc.transform(xtest)

#xtrain = (xtrain-xtrain.mean())/(xtrain.max()-xtrain.min())
#xtest = (xtest-xtest.mean())/(xtest.max()-xtest.min())
```

```
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=25)


model = Sequential()

model.add(Dense(20,activation='relu')) #Hidden layer
model.add(Dense(20,activation='relu')) #hidden layer
model.add(Dense(1,activation='sigmoid')) #Output layer (Since it's a binary classification problem)

#Using accuracy as loss function
model.compile(optimizer='sgd',loss='binary_crossentropy',metrics=['accuracy'])

model.fit(xtrain,ytrain,epochs=200,validation_data=(xtest, ytest),verbose=1,batch_size=128,callbacks=[early_stop])
```

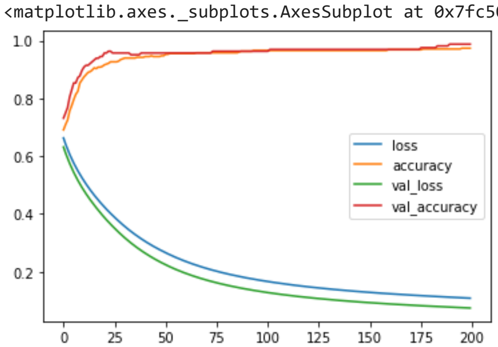
```
3/3 [==============================] - 0s 16ms/step - loss: 0.1189 - accuracy: 0.9686 - val_loss: 0.0840 - val_accuracy: 0.9695
Epoch 173/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1185 - accuracy: 0.9686 - val_loss: 0.0836 - val_accuracy: 0.9695
Epoch 174/200
3/3 [==============================] - 0s 17ms/step - loss: 0.1181 - accuracy: 0.9686 - val_loss: 0.0832 - val_accuracy: 0.9695
Epoch 175/200
3/3 [==============================] - 0s 16ms/step - loss: 0.1177 - accuracy: 0.9686 - val_loss: 0.0829 - val_accuracy: 0.9695
Epoch 176/200
3/3 [==============================] - 0s 23ms/step - loss: 0.1173 - accuracy: 0.9686 - val_loss: 0.0825 - val_accuracy: 0.9756
Epoch 177/200
3/3 [==============================] - 0s 17ms/step - loss: 0.1169 - accuracy: 0.9712 - val_loss: 0.0821 - val_accuracy: 0.9756
Epoch 178/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1165 - accuracy: 0.9712 - val_loss: 0.0818 - val_accuracy: 0.9756
Epoch 179/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1161 - accuracy: 0.9712 - val_loss: 0.0814 - val_accuracy: 0.9756
Epoch 180/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1157 - accuracy: 0.9712 - val_loss: 0.0811 - val_accuracy: 0.9756
Epoch 181/200
3/3 [==============================] - 0s 15ms/step - loss: 0.1153 - accuracy: 0.9712 - val_loss: 0.0807 - val_accuracy: 0.9756
Epoch 182/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1150 - accuracy: 0.9712 - val_loss: 0.0804 - val_accuracy: 0.9756
Epoch 183/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1146 - accuracy: 0.9712 - val_loss: 0.0800 - val_accuracy: 0.9756
Epoch 184/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1142 - accuracy: 0.9712 - val_loss: 0.0797 - val_accuracy: 0.9817
Epoch 185/200
3/3 [==============================] - 0s 25ms/step - loss: 0.1138 - accuracy: 0.9712 - val_loss: 0.0793 - val_accuracy: 0.9817
Epoch 186/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1135 - accuracy: 0.9712 - val_loss: 0.0790 - val_accuracy: 0.9817
Epoch 187/200
3/3 [==============================] - 0s 20ms/step - loss: 0.1131 - accuracy: 0.9712 - val_loss: 0.0787 - val_accuracy: 0.9817
Epoch 188/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1127 - accuracy: 0.9712 - val_loss: 0.0783 - val_accuracy: 0.9817
Epoch 189/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1124 - accuracy: 0.9712 - val_loss: 0.0780 - val_accuracy: 0.9817
Epoch 190/200
3/3 [==============================] - 0s 17ms/step - loss: 0.1120 - accuracy: 0.9712 - val_loss: 0.0777 - val_accuracy: 0.9878
Epoch 191/200
3/3 [==============================] - 0s 16ms/step - loss: 0.1117 - accuracy: 0.9712 - val_loss: 0.0774 - val_accuracy: 0.9878
Epoch 192/200
3/3 [==============================] - 0s 17ms/step - loss: 0.1113 - accuracy: 0.9712 - val_loss: 0.0771 - val_accuracy: 0.9878
Epoch 193/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1110 - accuracy: 0.9712 - val_loss: 0.0767 - val_accuracy: 0.9878
Epoch 194/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1106 - accuracy: 0.9712 - val_loss: 0.0764 - val_accuracy: 0.9878
Epoch 195/200
3/3 [==============================] - 0s 18ms/step - loss: 0.1103 - accuracy: 0.9712 - val_loss: 0.0761 - val_accuracy: 0.9878
Epoch 196/200
3/3 [==============================] - 0s 21ms/step - loss: 0.1100 - accuracy: 0.9738 - val_loss: 0.0758 - val_accuracy: 0.9878
Epoch 197/200
3/3 [==============================] - 0s 16ms/step - loss: 0.1096 - accuracy: 0.9738 - val_loss: 0.0755 - val_accuracy: 0.9878
Epoch 198/200
3/3 [==============================] - 0s 20ms/step - loss: 0.1093 - accuracy: 0.9738 - val_loss: 0.0752 - val_accuracy: 0.9878
Epoch 199/200
3/3 [==============================] - 0s 16ms/step - loss: 0.1090 - accuracy: 0.9738 - val_loss: 0.0749 - val_accuracy: 0.9878
Epoch 200/200
3/3 [==============================] - 0s 19ms/step - loss: 0.1086 - accuracy: 0.9738 - val_loss: 0.0746 - val_accuracy: 0.9878
<keras.callbacks.History at 0x7fc50b3cbd90>
```

```
model.history.history
```

```
        0.5908246040344238,
        0.5761460065841675,
        0.5624207258224487,
        0.5496200323104858,
        0.5374889969825745,
        0.5258851647377014,
        0.5148102045059204,
        0.5040702223777771,
        0.49391359090805054,
        0.48404818773269653,
        0.47464489936828613,
        0.4655056297779083,
        0.4567168354988098,
        0.4481598138809204,
        0.43992751836776733,
        0.43195435404777527,
        0.4241243898868561,
        0.41666674613952637,
        0.40924718976020813,
        0.40216273069381714,
        0.39525529742240906,
        0.3884894847869873,
        0.3819025754928589,
        0.3753448724746704,
        0.3689611852169037,
        0.362742155790329,
        0.35672885179519653,
        0.35083597898483276,
        0.3451557755470276,
        0.33966097235679626,
        0.3342936635017395,
        0.3291337788105011,
        0.32399725914001465,
        0.3190745413303375,
        0.3143277168273926,
        0.30962449312210083,
        0.30508187413215637,
        0.30067890882492065,
        0.2963985800743103,
        0.2922053933143616,
        0.28816086053848267,
        0.28420236706733704,
        0.28036218881607056,
        0.2766161561012268,
        0.2729471027851105,
        0.26931387186050415,
        0.2658396065235138,
        0.26241621375083923,
        0.2590753734111786,
        0.2558042109012604,
        0.2526128888130188,
```

```
lossdf=pd.DataFrame(model.history.history)
lossdf.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc50532b710>
```




```
ypred=model.predict(xtest)
ypred=ypred>0.5
```

```
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       112
           1       0.96      1.00      0.98        52
```

```
    accuracy                           0.99       164
   macro avg       0.98      0.99      0.99       164
weighted avg       0.99      0.99      0.99       164
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(ytest,ypred)
```

```
array([[110,   2],
       [  0,  52]])
```

```
sns.heatmap(confusion_matrix(ytest,ypred), annot=True,cmap='Greens',fmt='g',linewidth=2,linecolor='black')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc5052cb090>
```