PROBLEM STATEMENT

Breast cancer is one of the most common type of cancer in women. Many devices are built which detect the breast cancer but many times lead to false positives, which results is patients undergoing painful, expensive surgeries that were not even necessary. These type of cancers are called benign which do not require surgeries and we can reduce these unnecessary surgeries by using Machine Learning. We use a previous breast cancer patients dataset and train a model to predict whether the cancer is benign or malignant. These predictions can help doctors to do surgeries only when the cancer is not benign but malignant.

Let's import the following dependencies

1. Numpy
2. Pandas
3. Matplotlib
4. Seaborn
5. Sklearn

```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
sns.set_style("whitegrid")
import sklearn
```

Let's load the dataset

data.csv

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

Read the dataset df

```
df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	s
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 × 33 columns

Understanding the dataset

Check for the following

1. Shape
2. Columns
3. Description
4. Unique values
5. Missing values

```
# shape of the data
df.shape
```

```
(569, 33)
```

```
# Columns
df.columns
```

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
```

```
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

```
# Description
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	19.289649	91.969033	654.889104	
std	1.250206e+08	3.524049	4.301036	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	18.840000	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 × 32 columns

```
# Unique values
df.nunique()
```

```
id                         569
diagnosis                    2
radius_mean                456
texture_mean               479
perimeter_mean             522
area_mean                  539
smoothness_mean            474
compactness_mean           537
concavity_mean             537
concave points_mean        542
symmetry_mean              432
fractal_dimension_mean     499
radius_se                  540
texture_se                 519
perimeter_se               533
area_se                    528
smoothness_se              547
compactness_se             541
concavity_se               533
concave points_se          507
symmetry_se                498
fractal_dimension_se       545
```

```
radius_worst               457
texture_worst              511
perimeter_worst            514
area_worst                 544
smoothness_worst           411
compactness_worst          529
concavity_worst            539
concave points_worst       492
symmetry_worst             500
fractal_dimension_worst    535
Unnamed: 32                  0
dtype: int64
```

```
# Missing values
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
```

```
""" we can noticed that there is no missing
values except int the Unnamed column which
consistent of 569 missing values """
```

```
' we can noticed that there is no missing\nvalues except int the Unnamed column which \nconsistent of 569 missing values '
```

Taking care of missing values

```
# Let's drop the Unnamed Columbia
df.drop(['Unnamed: 32'],axis=1,inplace=True)
```

```
df_mean = df[['radius_mean',
'texture_mean',
'perimeter_mean',
'area_mean',
'smoothness_mean',
'compactness_mean',
'concavity_mean',
'concave points_mean',
'symmetry_mean',
'fractal_dimension_mean']]
```

```
df_mean
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	
565	20.13	28.25	131.20	1261.0	0.09780	
566	16.60	28.08	108.30	858.1	0.08455	
567	20.60	29.33	140.10	1265.0	0.11780	
568	7.76	24.54	47.92	181.0	0.05263	

569 rows × 10 columns

```
df_se = df[['radius_se',
'texture_se',
'perimeter_se',
'area_se',
```

```
'smoothness_se',
'compactness_se',
'concavity_se',
'concave points_se',
'symmetry_se',
'fractal_dimension_se']]
```

```
df_se
```

	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se
0	1.0950	0.9053	8.589	153.40	0.006399	0.04904
1	0.5435	0.7339	3.398	74.08	0.005225	0.01308
2	0.7456	0.7869	4.585	94.03	0.006150	0.04006
3	0.4956	1.1560	3.445	27.23	0.009110	0.07458
4	0.7572	0.7813	5.438	94.44	0.011490	0.02461
...	...	...	...	...	...	...
564	1.1760	1.2560	7.673	158.70	0.010300	0.02891
565	0.7655	2.4630	5.203	99.04	0.005769	0.02423
566	0.4564	1.0750	3.425	48.55	0.005903	0.03731
567	0.7260	1.5950	5.772	86.22	0.006522	0.06158
568	0.3857	1.4280	2.548	19.15	0.007189	0.00466

569 rows × 10 columns

```
df_worst = df[['radius_worst',
'texture_worst',
'perimeter_worst',
'area_worst',
'smoothness_worst',
'compactness_worst',
'concavity_worst',
'concave points_worst',
'symmetry_worst',
'fractal_dimension_worst']]
```

```
df_worst
```

	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst
0	25.380	17.33	184.60	2019.0	0.16220
1	24.990	23.41	158.80	1956.0	0.12380
2	23.570	25.53	152.50	1709.0	0.14440
3	14.910	26.50	98.87	567.7	0.20980
4	22.540	16.67	152.20	1575.0	0.13740
...	...	...	...	...	...
564	25.450	26.40	166.10	2027.0	0.14100
565	23.690	38.25	155.00	1731.0	0.11660

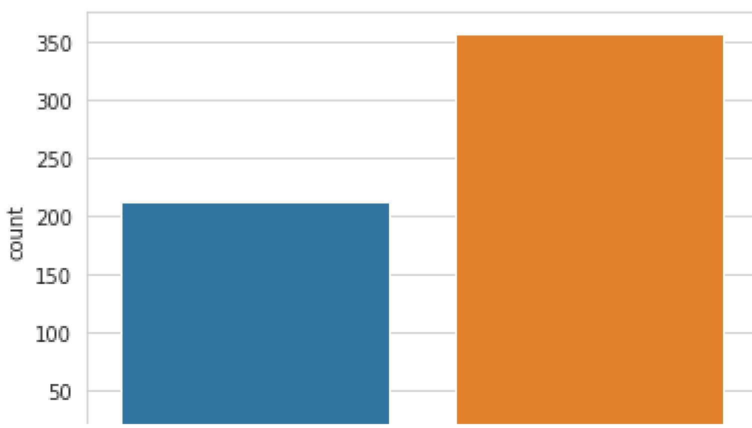
EDA

```
df.diagnosis.value_counts()
```

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

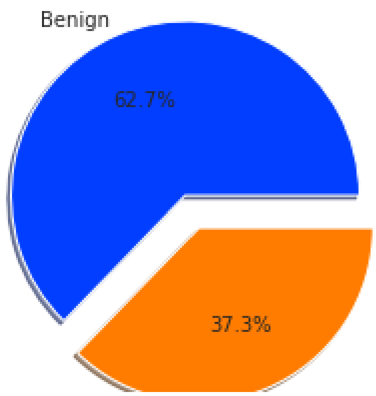
```
sns.countplot(df.diagnosis)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c4b1d8fd0>
```




```
colors=sns.color_palette('bright')
plt.pie(df['diagnosis'].value_counts(),
labels = ['Benign','Malignant'],
autopct=('%1.1f%%'),
explode = [0.1,0.1],
colors = colors,
shadow='True')
```

```
([<matplotlib.patches.Wedge at 0x7f7c4b131810>,
  <matplotlib.patches.Wedge at 0x7f7c4b16b310>],
 [Text(-0.46762330904436317, 1.105137295017411, 'Benign'),
  Text(0.46762320557394, -1.1051373387994603, 'Malignant')],
 [Text(-0.2727802636092118, 0.6446634220934897, '62.7%'),
  Text(0.27278020325146496, -0.6446634476330184, '37.3%')])
```



Let's check for the correlation between the variables of the dataset

```
corr = df.corr()
sns.heatmap(corr, annot = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7c4bfccd10>
```

Splitting the dataset

```
X = df.iloc[:,2:].values
y = df.iloc[:,1].values
Y = np.reshape(y, (-1,1))
```

```
print(X.shape)
print(Y.shape)
```

```
(569, 30)
(569, 1)
```

Categorical Data

Categorical data are variables that contain label values rather than numeric values.

We will use Label Encoder to label the categorical data. Label Encoder is the part of SciKit Learn library in Python and used to convert categorical data, or text data, into numbers, which our predictive models can better understand.

```
# Encoding labelEncoder
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()
Y = labelencoder.fit_transform(Y)
```

Splitting the dataset into Training set and Test set

```
# Train and Test set
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.25,random_state=0)
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(426, 30)
(143, 30)
(426,)
(143,)
```

Feature Scaling

We need to bring all features to the same level of magnitudes. This can be achieved by scaling. This means that you're transforming your data so that it fits within a specific scale

```
# StandardScaler
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

Model Selection

In the dataset we have the outcome variable or Dependent variable i.e Y having only two set of values, either M (Malign) or B(Benign). So we will use Classification algorithm of supervised learning.

We have different types of classification algorithms in Machine Learning and we are going to use all:-

1. Logistic Regression
2. Nearest Neighbor
3. Support Vector Machines
4. Kernel SVM
5. Naïve Bayes
6. Decision Tree Algorithm
7. Random Forest Classification

```
# Using scikit learn to import all algorithms

# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(random_state = 0)
lr.fit(X_train,y_train)

# Prediction
lr_pred = lr.predict(X_test)
lr_pred

# Let's check the score
print('Training Accuracy is :', lr.score(X_train,y_train))
print('Testing Accuracy is :', lr.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,lr_pred)
cm
```

```
Training Accuracy is : 0.9906103286384976
Testing Accuracy is : 0.958041958041958
array([[87,  3],
       [ 3, 50]])
```

```
# KNearestNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
Knn = KNeighborsClassifier(n_neighbors = 5,metric = 'minkowski',p=2)
Knn.fit(X_train,y_train)

# Prediction
Knn_pred = Knn.predict(X_test)
Knn_pred
```

```
# Let's check the score
print('Training Accuracy is :', Knn.score(X_train,y_train))
print('Testing Accuracy is :', Knn.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,Knn_pred)
cm
```

```
Training Accuracy is : 0.9741784037558685
Testing Accuracy is : 0.951048951048951
array([[89,  1],
       [ 6, 47]])
```

```
# Support Vector Machine (SVM)
from sklearn.svm import SVC
svc = SVC(kernel = 'linear',random_state=0)
svc.fit(X_train,y_train)

# Prediction
svc_pred = svc.predict(X_test)
svc_pred

# Let's check the score
print('Training Accuracy is :', svc.score(X_train,y_train))
print('Testing Accuracy is :', svc.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,svc_pred)
cm
```

```
Training Accuracy is : 0.9859154929577465
Testing Accuracy is : 0.972027972027972
array([[88,  2],
       [ 2, 51]])
```

```
from sklearn.svm import SVC
svc = SVC(kernel = 'rbf',random_state = 0)
svc.fit(X_train,y_train)

# Prediction
svc_pred = svc.predict(X_test)
svc_pred

# Let's check the score
print('Training Accuracy is :', svc.score(X_train,y_train))
print('Testing Accuracy is :', svc.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,svc_pred)
cm
```

```
Training Accuracy is : 0.9859154929577465
Testing Accuracy is : 0.965034965034965
array([[88,  2],
       [ 3, 50]])
```

```
# Naive Bayes
```

```
from sklearn.naive_bayes import GaussianNB
gb = GaussianNB()
gb.fit(X_train,y_train)

# Prediction
gb_pred = gb.predict(X_test)
gb_pred

# Let's check the score
print('Training Accuracy is :', gb.score(X_train,y_train))
print('Testing Accuracy is :', gb.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,gb_pred)
cm
```

```
Training Accuracy is : 0.9483568075117371
Testing Accuracy is : 0.916083916083916
array([[84,  6],
       [ 6, 47]])
```

```
# Decision Tree Algorithm
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train,y_train)

# Prediction
dtc_pred = dtc.predict(X_test)
dtc_pred

# Let's check the score
print('Training Accuracy is :', dtc.score(X_train,y_train))
print('Testing Accuracy is :', dtc.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,dtc_pred)
cm
```

```
Training Accuracy is : 1.0
Testing Accuracy is : 0.8741258741258742
array([[75, 15],
       [ 3, 50]])
```

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)

# Prediction
rfc_pred = rfc.predict(X_test)
rfc_pred

# Let's check the score
print('Training Accuracy is :', rfc.score(X_train,y_train))
print('Testing Accuracy is :', rfc.score(X_test,y_test))

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,rfc_pred)
cm
```

```
Training Accuracy is : 1.0
Testing Accuracy is : 0.972027972027972
array([[87,  3],
       [ 1, 52]])
```

We can see the accuracy of the algorithms used by using confusion_matrix method of metrics class. The confusion matrix is a way of tabulating the number of mis-classifications, i.e., the number of predicted classes which ended up in a wrong classification bin based on the true classes.

After applying the different classification models, we have got below accuracies with different models:

1. Logistic Regression — 99%
2. Nearest Neighbor — 97%
3. Support Vector Machines — 98%
4. Kernel SVM — 98%
5. Naive Bayes — 94%
6. Decision Tree Algorithm — 100%
7. Random Forest Classification — 100%

Now, we can see that Decision Tree Algorithm and Random Forest Classification gives the best results for the classification....