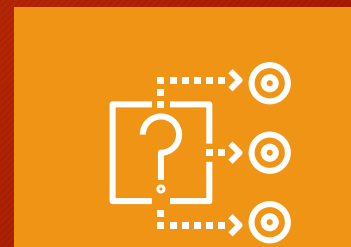
BREAST CANCER DATASET

CLASSIFICATION MODEL

ACKNOWLEDGEMENT

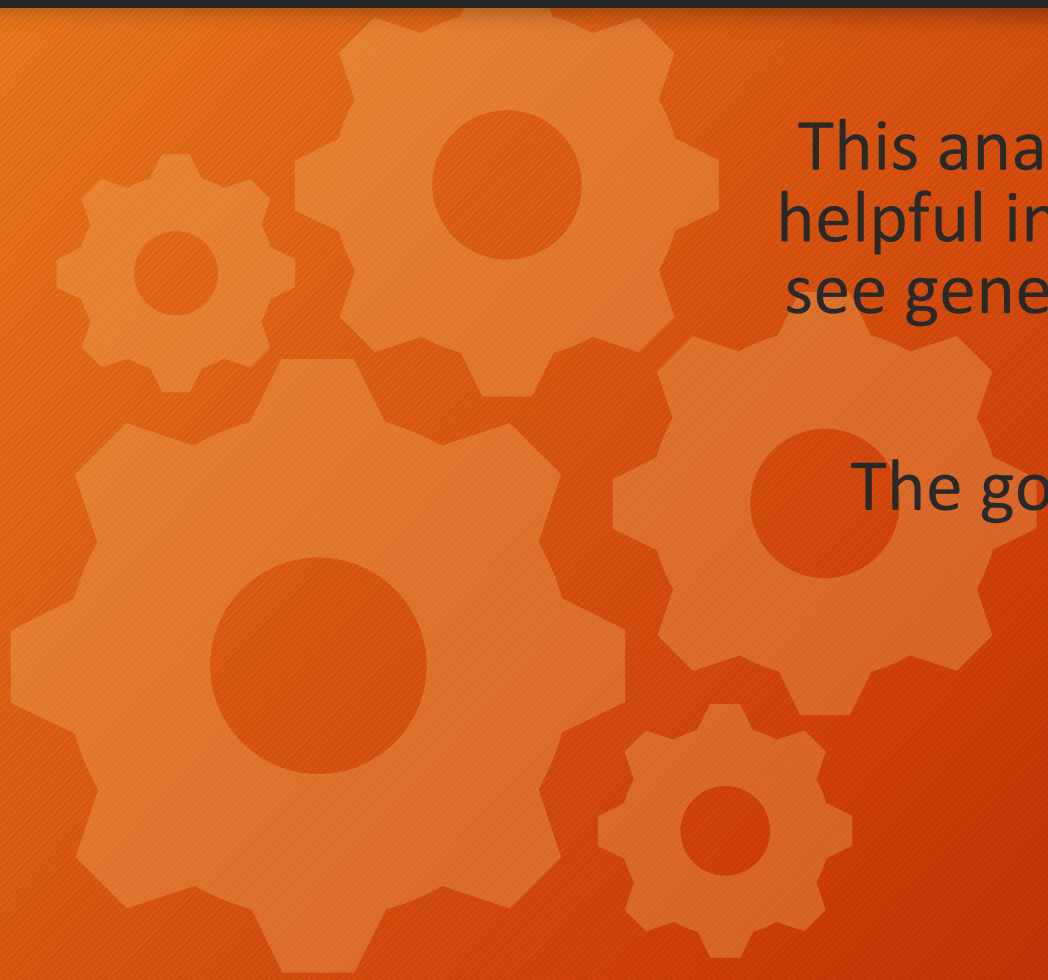


I would like to thanks Mr. Vineet Saxena sir and Ms. Prachi Garg mam for their expert advise and encouragement throughout this Project, as well as Mr. Rahul Garg sir for his supervision.

INTRODUCTION

- Breast cancer is one of the most common cancers among women worldwide, making it a significant public health problem in today’s society

OBJECTIVE



This analysis aims to observe which features are most helpful in predicting malignant or benign cancer and to see general trends that may aid us in model selection .

The goal is to classify whether the breast cancer is benign or malignant.

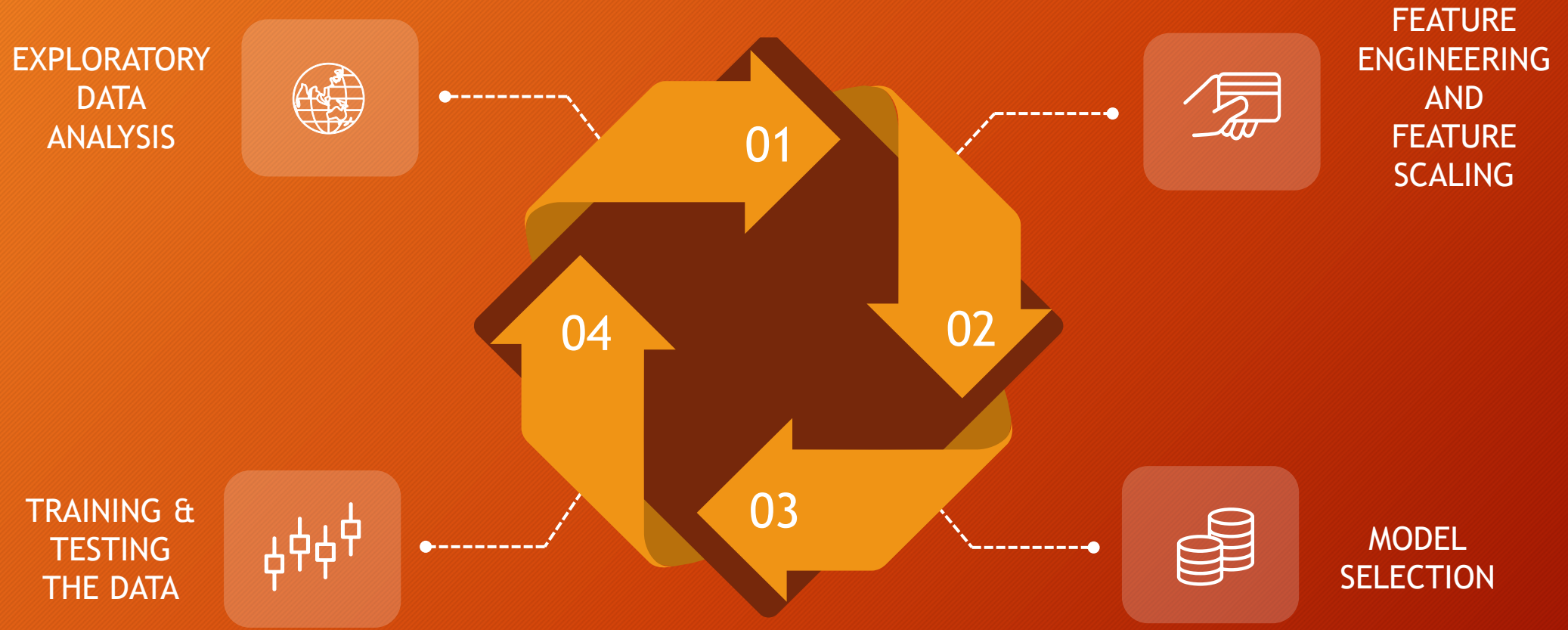
ABOUT THE DATASET

ATTRIBUTE INFORMATION

- Diagnosis:
  M = malignant,
  B = benign

REAL-VALUED FEATURES

- Radius
- Texture
- Perimeter
- Area
- Smoothness
- Symmetry
- Compactness
- Concavity
- Concave points
- Fractal Dimension



Process Of Building A Model

Loading Data

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

## Display all the columns of the dataframe
pd.pandas.set_option('display.max_columns',None)
```

In [2]:
```
dataset=pd.read_csv('data.csv')
dataset.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symme
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	

Checking for missing and categorical values

```
print(dataset.shape)
dataset.info()
```

```
(569, 33)
<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
```

```
 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
```

Unnamed: 32 is all null value and diagnosis is categorical value

Handling Missing Data

```
dataset.drop(['Unnamed: 32','id'],axis=1,inplace=True)
```

```
dataset.shape
```

```
(569, 31)
```

Loading and Handling the data

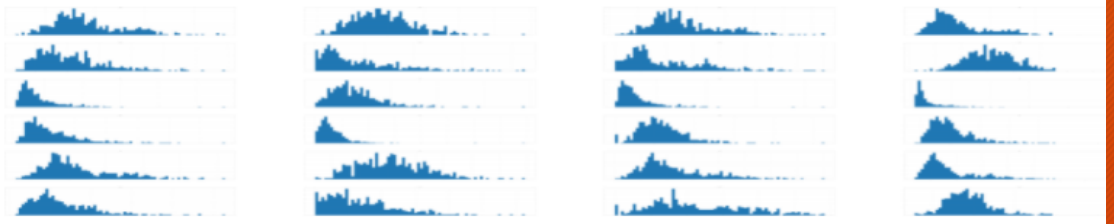
```
x=dataset.iloc[:,2:]
y=dataset.diagnosis
```

```
y.value_counts()
```

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

Visualization of dataset

```
import matplotlib.pyplot as plt
dataset.hist(bins=70, figsize=(800,120))
plt.show()
```



Encoding Categorical data values

```
from sklearn.preprocessing import LabelEncoder
labelencoder_Y=LabelEncoder()
y=labelencoder_Y.fit_transform(y)
```

```
print(x.shape)
print(y.shape)
```

```
(569, 29)
(569,)
```

Splitting the dataset_prep into training and testing test

```
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)
```

Feature Scaling

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

Preparing the Data

Model Selection

```
#Using Logistic Regression Algorithm to the Training Set

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, Y_train)


Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[87  3]
 [ 3 50]]
Accuracy is 95.8041958041958 %
```

```
#Using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor algorithm

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, Y_train)


Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[89  1]
 [ 6 47]]
Accuracy is 95.1048951048951 %
```

```
#Using SVC method of svm class to use Support Vector Machine Algorithm

from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, Y_train)

Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[88  2]
 [ 2 51]]
Accuracy is 97.2027972027972 %
```

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


Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[89  1]
 [ 3 50]]
Accuracy is 97.2027972027972 %
```

```
#Using GaussianNB method of naïve_bayes class to use Naïve Bayes Algorithm

from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, Y_train)


Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
```

Model selection

Trying various models for better accuracy.

```
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[83  7]
 [ 6 47]]
Accuracy is 90.9090909090909 %
```

```
#Using DecisionTreeClassifier of tree class to use Decision Tree Algorithm

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)     #criterion=entropy for i
classifier.fit(X_train, Y_train)                                                  #also gini citerion give



Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[82  8]
 [ 1 52]]
Accuracy is 93.7062937062937 %
```

```
#Using RandomForestClassifier method of ensemble class to use Random Forest Classification algorithm

from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, Y_train)



Y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)   #accuracy=TP + TN / TP + TN + FP + FN.
print(cm)
c=print("Accuracy is",((cm[0, 0] + cm[1, 1]) / (cm[0, 0] + cm[1, 1] + cm[0, 1] + cm[1, 0]))*100 ,"%")
```

```
[[87  3]
 [ 2 51]]
Accuracy is 96.5034965034965 %
```

We can see that Random Forest Classification algorithm gives the best results for our dataset.

CONCLUSION:

As we can see from the attached images, Random Forest Classification algorithm gives the best results about 96.50% for our dataset.

THANK YOU.