# Detection of Breast Cancer with Python

# INTRODUCTION

-Women diagnosed with breast cancer each year reflects majority of new cancer cases and related deaths.

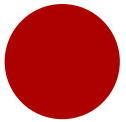
-Early diagnosis of breast cancer with timely and effective treatment services improves the prognosis and survival of patients.

-Accurate classification can prevent patients from unnecessary treatments.

-Study is based on machine learning (ML) algorithms, aiming to review python technique and its application in breast cancer diagnosis and prognosis by building simple machine learning model.

-It has unique advantage as it detects critical features from complex breast cancer datasets.

-To test the machine learning technique in dataset via different algorithms, a benchmark dataset for this study is extracted from Wisconsin breast cancer database (WBCD).



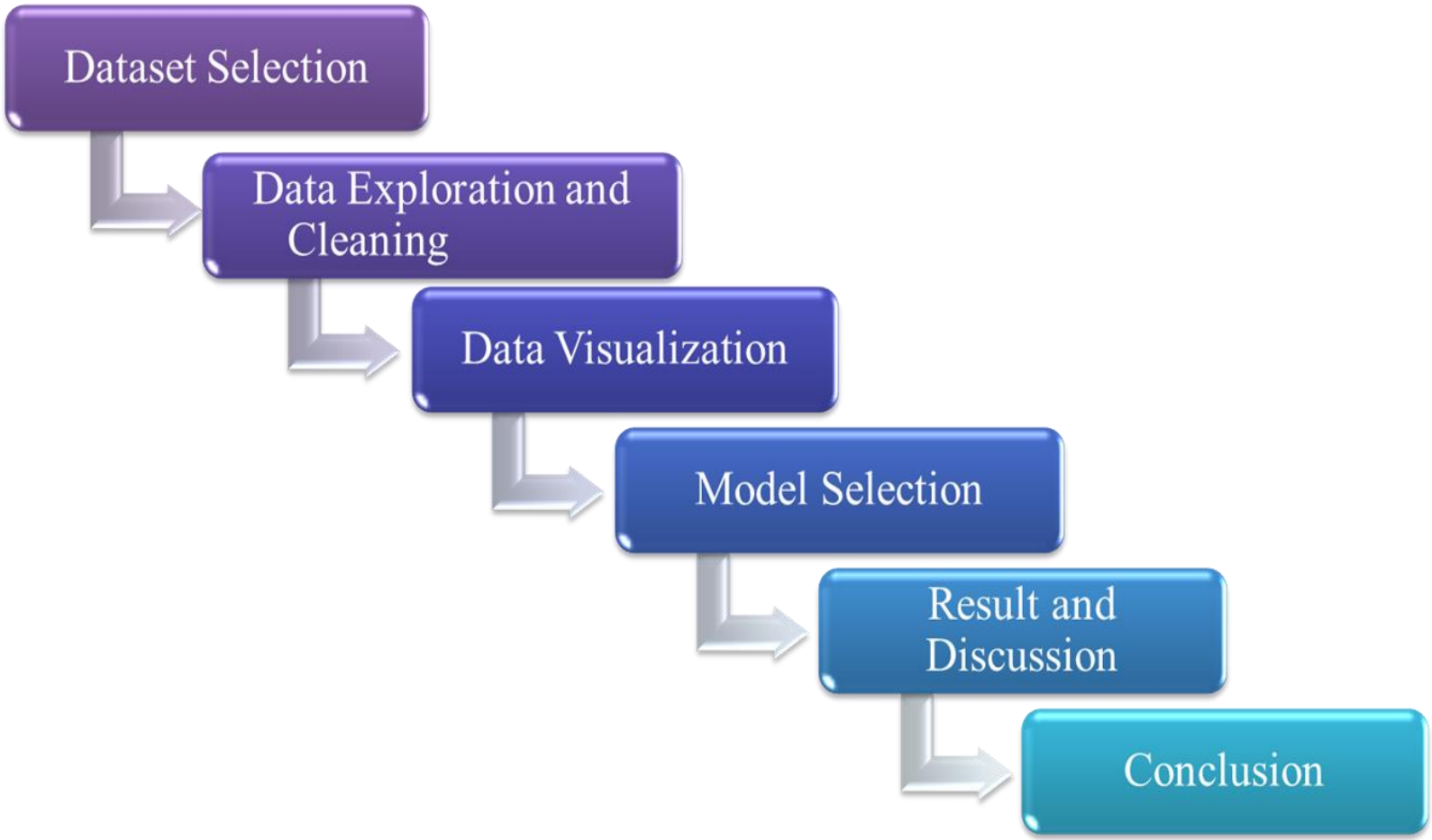
# Objectives

- Classify benign and malignant tumors of breast cancer with the help of python.

- The focus is on using :

- Logistic Regression,
- K-Neighbours Classifier,
- Support Vector Classifier linear (SVC linear),
- Gaussian Naive Bayes (GaussianNB), and
- Decision Tree Classifier (DT).

# METHODOLOGY



# 1. Dataset

- Wisconsin Breast Cancer Diagnostics (WBCD) dataset from UCI Machine Learning Repository.
- Highlights of characteristics :

# 2. Data Exploration and Cleaning

## 2.1. Importing Libraries

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

## 2.2. Loading Data

In [2]:
```
#Load the data
df = pd.read_csv('data.csv')
df.head()
```

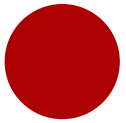
Out[2]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean |
|---|---|---|---|---|---|---|---|
| 0 | 842302 | M | 17.99 | 10.38 | 122.80 | 1001.0 | 0.11840 |
| 1 | 842517 | M | 20.57 | 17.77 | 132.90 | 1326.0 | 0.08474 |
| 2 | 84300903 | M | 19.69 | 21.25 | 130.00 | 1203.0 | 0.10960 |
| 3 | 84348301 | M | 11.42 | 20.38 | 77.58 | 386.1 | 0.14250 |
| 4 | 84358402 | M | 20.29 | 14.34 | 135.10 | 1297.0 | 0.10030 |

## 2.3. Counting columns of empty values

```
In [4]: #Count the empty (NaN, NAN, na) values in each column
        df.isna().sum()

Out[4]: 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
```



## 2.4. Dropping empty column and counting number of rows and column of new data frame

```
In [5]: df = df.dropna(axis =1 )
```

```
In [6]: df.shape

Out[6]: (569, 32)
```

## 2.5. Counting number of Malignant and Benign patients

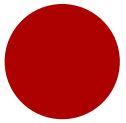
```
In [7]: #Get a count of the number of 'M' & 'B' cells
        df['diagnosis'].value_counts()

Out[7]: B    357
        M    212
        Name: diagnosis, dtype: int64
```

## 2.6. Listing columns and their data type

```
In [9]: #Look at the data types
        df.dtypes

Out[9]: id                         int64
        diagnosis                 object
        radius_mean              float64
        texture_mean             float64
        perimeter_mean           float64
        area_mean                float64
        smoothness_mean          float64
        compactness_mean         float64
        concavity_mean           float64
        concave points_mean      float64
        symmetry_mean            float64
        fractal_dimension_mean   float64
        radius_se                float64
        texture_se               float64
        perimeter_se             float64
        area_se                  float64
        smoothness_se            float64
        compactness_se           float64
        concavity_se             float64
        concave points_se        float64
        symmetry_se              float64
        fractal_dimension_se     float64
        radius_worst             float64
        texture_worst            float64
        perimeter_worst          float64
        area_worst               float64
        smoothness_worst         float64
        compactness_worst        float64
        concavity_worst          float64
        concave points_worst     float64
        symmetry_worst           float64
        fractal_dimension_worst  float64
        dtype: object
```



## 2.7. Encoding categorical data

```
In [10]: #Encoding categorical data values (
         from sklearn.preprocessing import LabelEncoder
         labelencoder_Y = LabelEncoder()
         df.iloc[:,1]= labelencoder_Y.fit_transform(df.iloc[:,1].values)
         print(labelencoder_Y.fit_transform(df.iloc[:,1].values))#Encoding categorical data
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 0 1 1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 1
 0 1 0 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0
 0 0 0 0 0 0 1 1 1 0 1 1 0 0 0 1 1 0 1 0 1 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 0 0 1 0 0 1 1 1 0 1
 0 1 0 0 0 1 0 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 1 1 0 0
 0 1 0 0 0 0 0 1 1 0 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 1 1 0 0
 0 0 1 0 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1
 1 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 0 0 0 0 0 1 0 0
 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 1 1
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0]
```

# 3. Data Visualization

## 3.1. Count Plot

```
In [8]: #Visualize this count
        sns.countplot(df['diagnosis'],label="Count")

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x233b7e1f848>
```

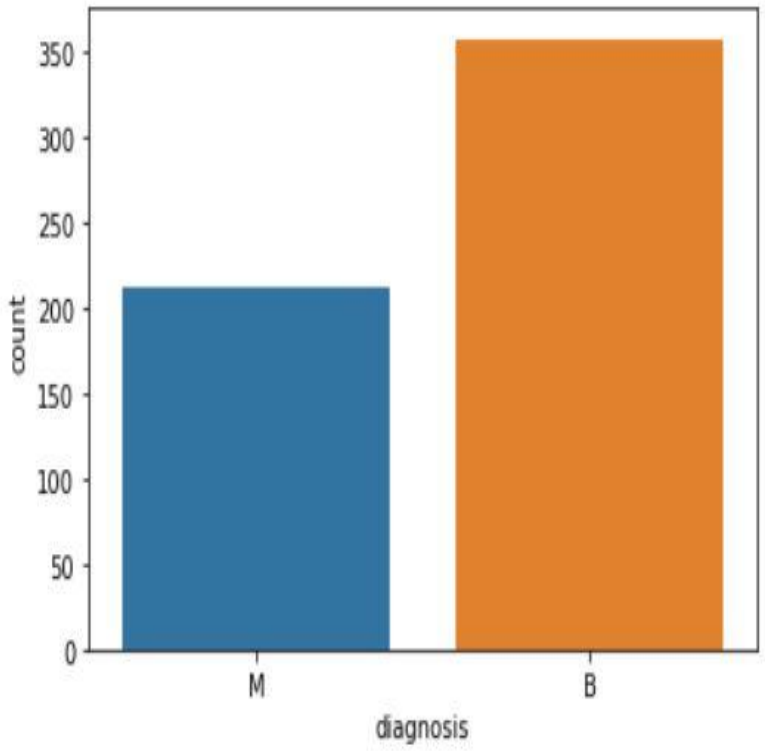



**Fig.1** Count Plot for Malignant & Benign patients

# 3.2. Pair Plot

```
In [11]: #create a pair plot
         sns.pairplot(df, hue="diagnosis")

Out[11]: <seaborn.axisgrid.PairGrid at 0x233bacb9fc8>
```

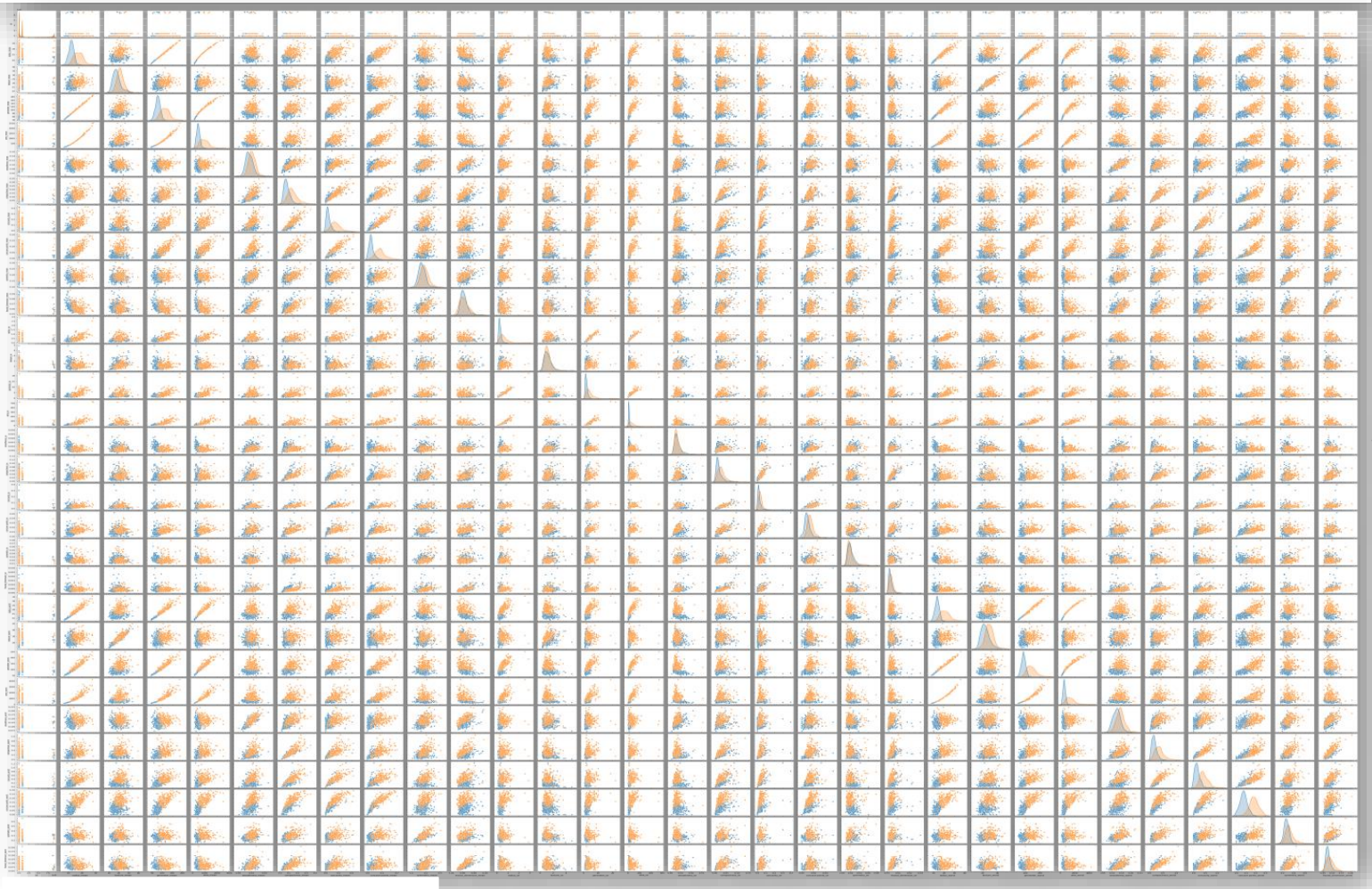


**Fig. 2** Pair Plot

## 3.3. Heat Map

In [13]:
```
#Get the correlation of the columns
df.corr().head()
```

Out[13]:

| | id | diagnosis | radius_mean | texture_mean | perimeter_mean | area_mean | smoothness_mean | c |
|---|---|---|---|---|---|---|---|---|
| **id** | 1.000000 | 0.039769 | 0.074626 | 0.099770 | 0.073159 | 0.096893 | -0.012968 | |
| **diagnosis** | 0.039769 | 1.000000 | 0.730029 | 0.415185 | 0.742636 | 0.708984 | 0.358560 | |
| **radius_mean** | 0.074626 | 0.730029 | 1.000000 | 0.323782 | 0.997855 | 0.987357 | 0.170581 | |
| **texture_mean** | 0.099770 | 0.415185 | 0.323782 | 1.000000 | 0.329533 | 0.321086 | -0.023389 | |
| **perimeter_mean** | 0.073159 | 0.742636 | 0.997855 | 0.329533 | 1.000000 | 0.986507 | 0.207278 | |

5 rows × 32 columns

In [14]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.corr(), annot=True, fmt='.0%')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd76e7ec88>
```

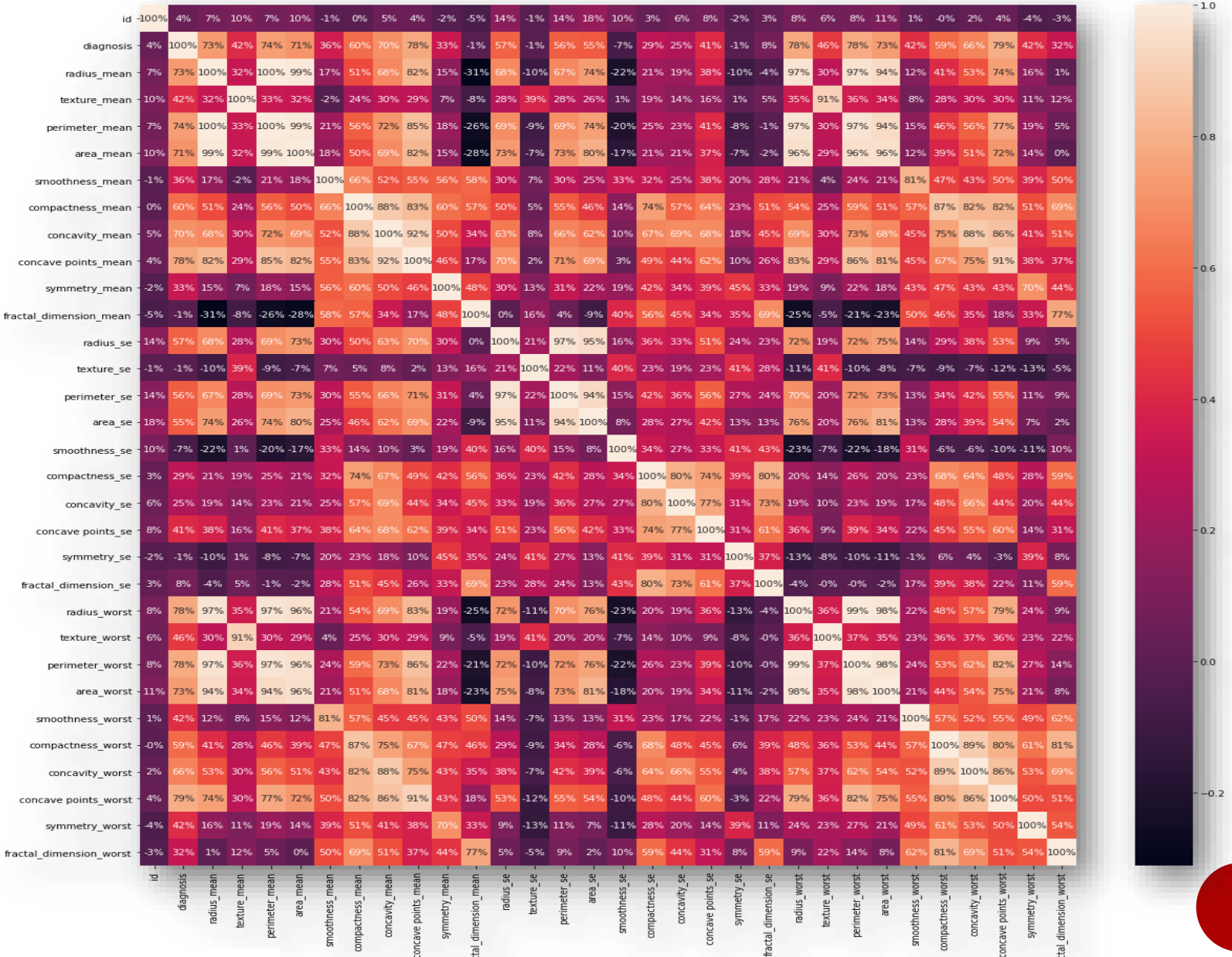



**Fig. 3** Heat Map

# 4. Model Selection

## 4.1. Split data set

```
In [15]: X = df.iloc[:, 2:31].values
         Y = df.iloc[:, 1].values
```

```
In [16]: 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)
```

## 4.2. Feature Scaling

```
In [17]: #Feature Scaling
         from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

## 4.3. Create functions to hold different models

In [27]:
```
def models(X_train,Y_train):
  #Using Logistic Regression
  from sklearn.linear_model import LogisticRegression
  log = LogisticRegression(random_state = 0)
  log.fit(X_train, Y_train)
  #Using KNeighborsClassifier
  from sklearn.neighbors import KNeighborsClassifier
  knn = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
  knn.fit(X_train, Y_train)
  #Using SVC linear
  from sklearn.svm import SVC
  svc_lin = SVC(kernel = 'linear', random_state = 0)
  svc_lin.fit(X_train, Y_train)
  #Using GaussianNB
  from sklearn.naive_bayes import GaussianNB
  gauss = GaussianNB()
  gauss.fit(X_train, Y_train)
```

```
#Using DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
tree.fit(X_train, Y_train)
#Using RandomForestClassifier method of ensemble class to use Random Forest Classification algorithm
from sklearn.ensemble import RandomForestClassifier
#print model accuracy on the training data.
print('[0]Logistic Regression Training Accuracy:', log.score(X_train, Y_train))
print('[1]K Nearest Neighbor Training Accuracy:', knn.score(X_train, Y_train))
print('[2]Support Vector Machine (Linear Classifier) Training Accuracy:', svc_lin.score(X_train, Y_train))
print('[3]Gaussian Naive Bayes Training Accuracy:', gauss.score(X_train, Y_train))
print('[4]Decision Tree Classifier Training Accuracy:', tree.score(X_train, Y_train))
return log, knn, svc_lin, gauss, tree
```

## 4.4. Creating model

```
In [28]: model = models(X_train,Y_train)

[0]Logistic Regression Training Accuracy: 0.9906103286384976
[1]K Nearest Neighbor Training Accuracy: 0.9765258215962441
[2]Support Vector Machine (Linear Classifier) Training Accuracy: 0.9882629107981221
[3]Gaussian Naive Bayes Training Accuracy: 0.9507042253521126
[4]Decision Tree Classifier Training Accuracy: 1.0
```

# Confusion matrix

In [29]:
```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
  cm = confusion_matrix(Y_test, model[i].predict(X_test))

  TN = cm[0][0]
  TP = cm[1][1]
  FN = cm[1][0]
  FP = cm[0][1]

  print(cm)
  print('Model[{}] Testing Accuracy = "{}!"'.format(i,  (TP + TN) / (TP + TN + FN + FP)))
  print()# Print a new line
```

```
[[86  4]
 [ 4 49]]
Model[0] Testing Accuracy = "0.9440559440559441!"

[[89  1]
 [ 5 48]]
Model[1] Testing Accuracy = "0.958041958041958!"

[[87  3]
 [ 2 51]]
Model[2] Testing Accuracy = "0.965034965034965!"

[[85  5]
 [ 6 47]]
Model[3] Testing Accuracy = "0.9230769230769231!"

[[84  6]
 [ 1 52]]
Model[4] Testing Accuracy = "0.951048951048951!"
```

## 4.5. Cross Validation

```
from sklearn.model_selection import cross_val_score
cross_validation0 = cross_val_score(model[0],  X= X_train, y = Y_train, cv= 10)
cross_validation1 = cross_val_score(model[1],  X= X_train, y = Y_train, cv= 10)
cross_validation2 = cross_val_score(model[2],  X= X_train, y = Y_train, cv= 10)
cross_validation3 = cross_val_score(model[3],  X= X_train, y = Y_train, cv= 10)
cross_validation4 = cross_val_score(model[4],  X= X_train, y = Y_train, cv= 10)

cross_validation0.mean(), cross_validation1.mean(), cross_validation2.mean(), cross_validation3.mean(),cross_validation4.mean()
```

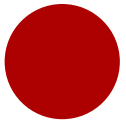
```
(0.9860465116279069,
 0.9648394241417497,
 0.9719269102990034,
 0.9437984496124031,
 0.9270764119601329)
```

## 5. Result and Discussion

- SVM linear model is slightly imbalanced with cross validation 97.19%, training set 98.83% and testing 96.50%
- SVM linear is comparatively more accurate than Logistic Regression, KNN, GaussianNB, and Decision Tree in detecting breast cancer.

## 6. Conclusion

- The analysis of result signifies that, integration of data, feature scaling along with different classification method and analysis provide markedly successful tool in prediction.
- Although, the model is accurate but when dealing with lives of people, further research in building techniques to get the accuracy as close to 100% as possible.



# References


- http:// www.imaginis.com/breasthealth/breast_cancer.asp, Last Accessed August 2007.
- West, D., Mangiameli, P., Rampal, R., & West, V. (2005). Ensemble strategies for a medical diagnosis decision support system: A breast cancer diagnosis application. European Journal of Operational Research(162), 532–551
- Pavlopoulos, S.A.; Delopoulos, A.N. Designing and implementing the transition to a fully digital hospital. IEEE Trans. Inf. Technol. Biomed. 1999, 3, 6–19.
- Barracliffe, L.; Arandjelović, O.; Humphris, G. A pilot study of breast cancer patients: Can machine learning predict healthcare professionals' responses to patient emotions? In Proceedings of the International Conference on Bioinformatics and Computational Biology, Honolulu, HI, USA, 20–22 March 2017; pp. 101–106.

- G. Valvano, G. Santini, N. Martini et al., “Convolutional neural networks for the segmentation of microcalcification in mammography imaging,” Journal of Healthcare Engineering, vol. 2019, Article ID 9360941, 9 pages, 2019.
- M. F. Akay, “Support vector machines combined with feature selection for breast cancer diagnosis,” Expert Systems with Applications, vol. 36, no. 2, pp. 3240–3247, 2009.
- P. Salembier and L. Garrido, “Binary partition tree as an efficient representation for image processing, segmentation, and information retrieval,” IEEE Transactions on Image Processing, vol. 9, no. 4, pp. 561–576, 2000.
- Mangasarian, O.L.; Setiono, R.; Wolberg, W.H. Pattern recognition via linear programming: Theory and application to medical diagnosis. In Large-Scale Numerical Optimization; SIAM: Philadelphia, PA, USA, 1990; pp. 22–31.

# Annexure

- The benchmark data of interest for detection of breast cancer can be referred from https://www.kaggle.com/uciml/breast-cancer-wisconsin-data
- For more reference, the program of this paper is uploaded in GitHub link https://github.com/neha-2568/Detection-of-Breast-Cancer
- The more clear vision of Pair Plot and Heat Map is available in GitHub link https://github.com/neha-2568/Detection-of-Breast-Cancer

# Thank You

**Prepared by : Neha Singh**
**Roll no. : 0943**
**HM- 24, 2019-21**