BREAST Canceer Detection USing Machine Learning (Classification)

```
# Import essential Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas.util.testing as tm


#Data Load
from sklearn.datasets import load_breast_cancer
cancer_dataset = load_breast_cancer()
cancer_dataset
```

```
     {'DESCR': '.. _breast_cancer_dataset:\n\nBreast
      'data': array([[1.799e+01, 1.038e+01, 1.228e+02
              1.189e-01],
             [2.057e+01, 1.777e+01, 1.329e+02, ..., 1
              8.902e-02],
             [1.969e+01, 2.125e+01, 1.300e+02, ..., 2
              8.758e-02],
             ...,
             [1.660e+01, 2.808e+01, 1.083e+02, ..., 1
              7.820e-02],
             [2.060e+01, 2.933e+01, 1.401e+02, ..., 2
              1.240e-01],
             [7.760e+00, 2.454e+01, 4.792e+01, ..., 0
              7.039e-02]]),
      'feature_names': array(['mean radius', 'mean te
             'mean smoothness', 'mean compactness', '
             'mean concave points', 'mean symmetry',
             'radius error', 'texture error', 'perime
             'smoothness error', 'compactness error',
             'concave points error', 'symmetry error'
             'fractal dimension error', 'worst radius
             'worst perimeter', 'worst area', 'worst
             'worst compactness', 'worst concavity',
             'worst symmetry', 'worst fractal dimensi
      'filename': '/usr/local/lib/python3.6/dist-pack
      'target': array([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, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0
             1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1
             1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1
             1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1
```

```
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0
 'target_names': array(['malignant', 'benign'],
```

```
#Type of the dataset
print(type(cancer_dataset))
```

```
<class 'sklearn.utils.Bunch'>
```

```
#we got data in Dictonary format, to find the whole key of dataset
print( cancer_dataset.keys())
```

```
dict_keys(['data', 'target', 'target_names', 'DE
```

```
# let's find the data of person
data = cancer_dataset['data']
print(data)
```

```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.
```

```
# Now to find "Malignant or Benign Value", Malignant=Effected(1) and Benign=Not Effected(0)
cancer_dataset['target']
```

```
array([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, 0, 0, 0,
```

```
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
```

```
# Target value name Malignant or Benign tumor
cancer_dataset['target_names']
```

```
array(['malignant', 'benign'], dtype='<U9')
```

```
# To check the decribation of the Data
print(cancer_dataset['DESCR'])
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predic

    :Attribute Information:
        - radius (mean of distances from cent
        - texture (standard deviation of gray
        - perimeter
        - area
        - smoothness (local variation in radi
        - compactness (perimeter^2 / area - 1
        - concavity (severity of concave port
        - concave points (number of concave p
        - symmetry
```

```
        - fractal dimension ("coastline appro

        The mean, standard error, and "worst"
        largest values) of these features wer
        resulting in 30 features.  For instan
        13 is Radius SE, field 23 is Worst Ra

        - class:
                - WDBC-Malignant
                - WDBC-Benign

    :Summary Statistics:

    ===================================== ===
                                           Mi
    ===================================== ===
    radius (mean):                        6.9
    texture (mean):                       9.7
    perimeter (mean):                     43.
    area (mean):                          143
    smoothness (mean):                    0.0
    compactness (mean):                   0.0
    concavity (mean):                     0.0
    concave points (mean):                0.0
    symmetry (mean):                      0.1
    fractal dimension (mean):             0.0
    radius (standard error):              0.1
    texture (standard error):             0.3
    perimeter (standard error):           0.7
    area (standard error):                6.8
    smoothness (standard error):          0.0
    compactness (standard error):         0.0
    concavity (standard error):           0.0
    concave points (standard error):      0.0
    symmetry (standard error):            0.0
    fractal dimension (standard error):   0.0
    radius (worst):                       7.9
```

```
# Names of the Features
print(cancer_dataset['feature_names'])
```

```
['mean radius' 'mean texture' 'mean perimeter' '
 'mean smoothness' 'mean compactness' 'mean conc
 'mean concave points' 'mean symmetry' 'mean fra
 'radius error' 'texture error' 'perimeter error
 'smoothness error' 'compactness error' 'concavi
 'concave points error' 'symmetry error' 'fracta
 'worst radius' 'worst texture' 'worst perimeter
 'worst smoothness' 'worst compactness' 'worst c
 'worst concave points' 'worst symmetry' 'worst
```

```
# Let's check the location of the breast cancer file
print(cancer_dataset['filename'])
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/d
```

```
# Create Dataframe to implement the module (Head of Cancer DataFrame)
cancer_dt = pd.DataFrame(np.c_[cancer_dataset['data'],cancer_dataset['target']],
                         columns = np.append(cancer_dataset['feature_names'],['target']
cancer_dt
```

	mean radius	mean texture	mean perimeter	mean area	m smoothn
0	17.99	10.38	122.80	1001.0	0.11
1	20.57	17.77	132.90	1326.0	0.08
2	19.69	21.25	130.00	1203.0	0.10
3	11.42	20.38	77.58	386.1	0.14
4	20.29	14.34	135.10	1297.0	0.10
...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11
565	20.13	28.25	131.20	1261.0	0.09
566	16.60	28.08	108.30	858.1	0.08
567	20.60	29.33	140.10	1265.0	0.11
568	7.76	24.54	47.92	181.0	0.05

569 rows × 31 columns

```
#let's store the dataframe into our system location file
cancer_dt.to_csv(r'B:\DATA_SCIENCE\Data_Science_Project\Breast_Cancer_Classification\breast_c
print('File has been created in the location folder')
```

```
File has been created in the location folder
```

```
# To check the missing value or any blank value
cancer_dt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                  Non-Null Count  Dt
---  ------                  --------------  --
 0   mean radius             569 non-null    fl
 1   mean texture            569 non-null    fl
```

```
 2   mean perimeter           569 non-null    fl
 3   mean area                569 non-null    fl
 4   mean smoothness          569 non-null    fl
 5   mean compactness         569 non-null    fl
 6   mean concavity           569 non-null    fl
 7   mean concave points      569 non-null    fl
 8   mean symmetry            569 non-null    fl
 9   mean fractal dimension   569 non-null    fl
 10  radius error             569 non-null    fl
 11  texture error            569 non-null    fl
 12  perimeter error          569 non-null    fl
 13  area error               569 non-null    fl
 14  smoothness error         569 non-null    fl
 15  compactness error        569 non-null    fl
 16  concavity error          569 non-null    fl
 17  concave points error     569 non-null    fl
 18  symmetry error           569 non-null    fl
 19  fractal dimension error  569 non-null    fl
 20  worst radius             569 non-null    fl
 21  worst texture            569 non-null    fl
 22  worst perimeter          569 non-null    fl
 23  worst area               569 non-null    fl
 24  worst smoothness         569 non-null    fl
 25  worst compactness        569 non-null    fl
 26  worst concavity          569 non-null    fl
 27  worst concave points     569 non-null    fl
 28  worst symmetry           569 non-null    fl
 29  worst fractal dimension  569 non-null    fl
 30  target                   569 non-null    fl
dtypes: float64(31)
memory usage: 137.9 KB
```

```
# To check the data statical detail
cancer_dt.describe()
```

	mean radius	mean texture	mean perimeter	mean
count	569.000000	569.000000	569.000000	569.0
mean	14.127292	19.289649	91.969033	654.8
std	3.524049	4.301036	24.298981	351.9
min	6.981000	9.710000	43.790000	143.5
25%	11.700000	16.170000	75.170000	420.3
50%	13.370000	18.840000	86.240000	551.1
75%	15.780000	21.800000	104.100000	782.7
max	28.110000	39.280000	188.500000	2501.0

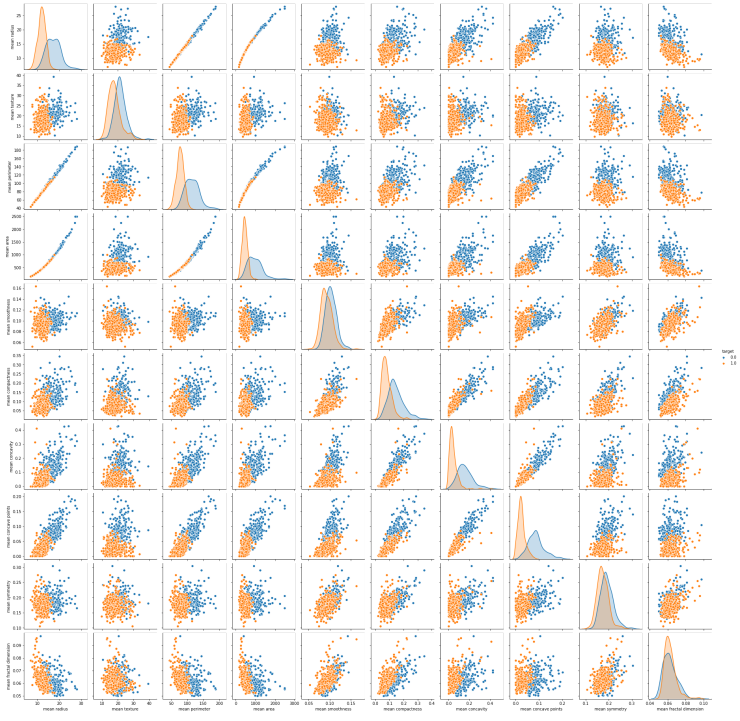
```
# To check the null values
cancer_dt.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

DATA **VISUALIZATION**

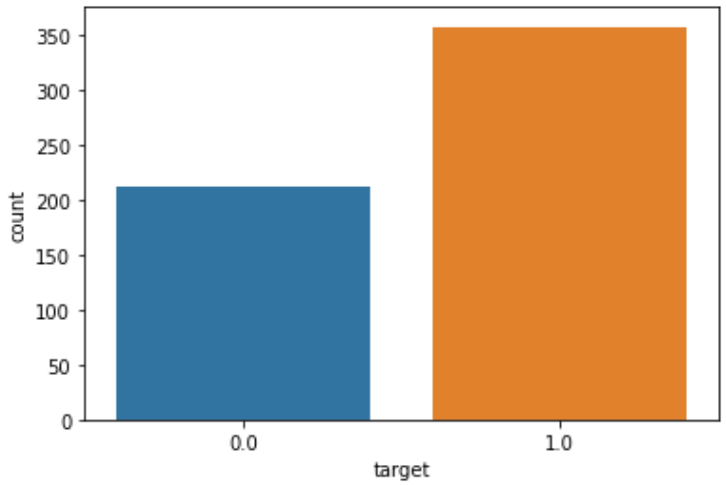
```
# Paiplot of cancer datafram
sns.pairplot(cancer_dt, hue= 'target', vars = ['mean radius', 'mean texture', 'mean perimeter
```

```
<seaborn.axisgrid.PairGrid at 0x7fd1e32c79e8>
```



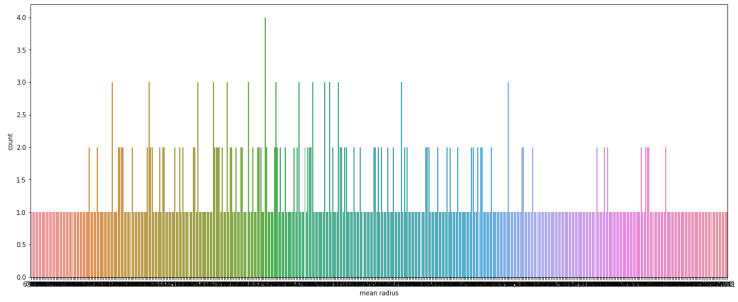
```
# Count the Target Class
sns.countplot(cancer_dt['target'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1
```



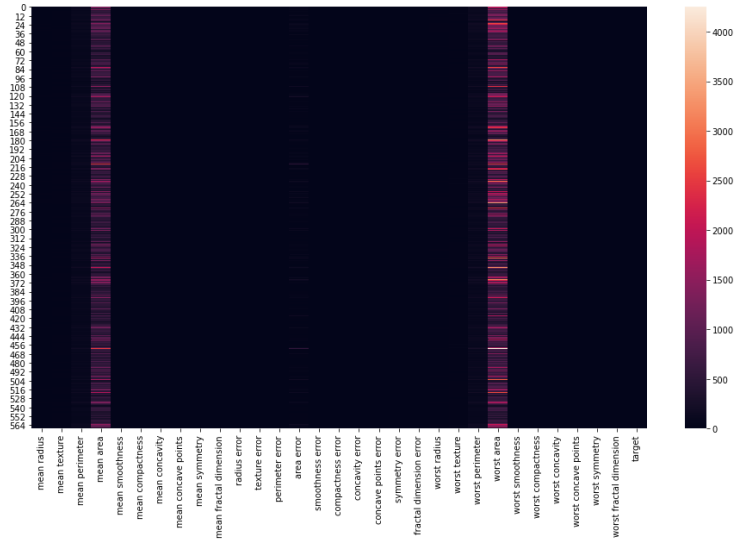

```
#Counter plot to find mean radius to see the nearby value
plt.figure(figsize=(20,8))
sns.countplot(cancer_dt['mean radius'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd2
```



```
# HEATMAP of dataframe
plt.figure(figsize=(16,9))
sns.heatmap(cancer_dt)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1
```



HEATMAP of a correlation matrix

```
cancer_dt.corr()
```

	mean radius	mean texture	mean perimeter	
mean radius	1.000000	0.323782	0.997855	0.
mean texture	0.323782	1.000000	0.329533	0.
mean perimeter	0.997855	0.329533	1.000000	0.
mean area	0.987357	0.321086	0.986507	1.
mean smoothness	0.170581	-0.023389	0.207278	0.
mean compactness	0.506124	0.236702	0.556936	0.
mean concavity	0.676764	0.302418	0.716136	0.
mean concave points	0.822529	0.293464	0.850977	0.
mean symmetry	0.147741	0.071401	0.183027	0.
mean fractal dimension	-0.311631	-0.076437	-0.261477	-0.
radius error	0.679090	0.275869	0.691765	0.
texture error	-0.097317	0.386358	-0.086761	-0.
perimeter error	0.674172	0.281673	0.693135	0.
area error	0.735864	0.259845	0.744983	0.
smoothness error	-0.222600	0.006614	-0.202694	-0.
compactness error	0.206000	0.191975	0.250744	0.
concavity error	0.194204	0.143293	0.228082	0.
concave points error	0.376169	0.163851	0.407217	0.
symmetry error	-0.104321	0.009127	-0.081629	-0.
fractal dimension error	-0.042641	0.054458	-0.005523	-0.
worst radius	0.969539	0.352573	0.969476	0

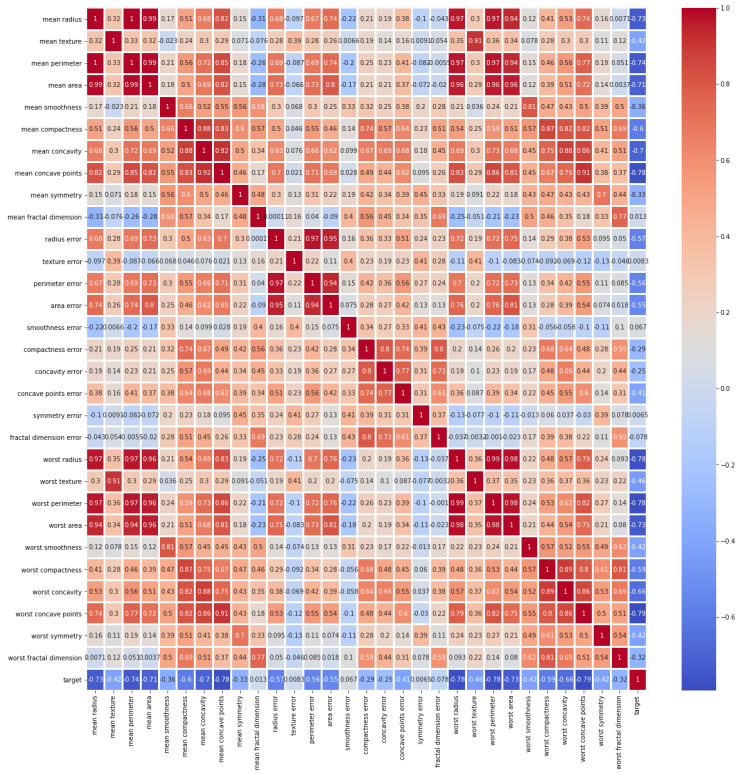
worst radius	0.969539	0.352573	0.969476	0.
worst texture	0.297008	0.912045	0.303038	0.
worst perimeter	0.965137	0.358040	0.970387	0.
worst area	0.941082	0.343546	0.941550	0.
worst smoothness	0.119616	0.077503	0.150549	0.
worst compactness	0.413463	0.277830	0.455774	0.

```
# HeatMap of the correlation matrix of the breast cancer
plt.figure(figsize=(20,20))
sns.heatmap(cancer_dt.corr(), annot=True, cmap='coolwarm', linewidths=2)
```

Ravi Kumar Rahul Resolve
Jun 24, 2020

In the above correlation barplot only feature ‘smoothness error’ is strongly positively correlated with the target than others. The features ‘mean factor dimension’, ‘texture error’, and ‘symmetry error’ are very less positive correlated and others remaining are strongly negatively correlated.

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1
```

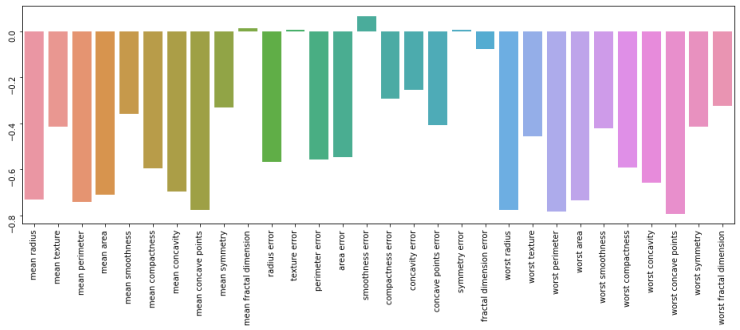


Correlation Barplot

```
#creating second dataframe by droppig target
cancer_dt2 = cancer_dt.drop(['target'], axis=1)
print("The shape of 'Cancer_dt2' is :" , cancer_dt2.shape)
```

```
    The shape of 'Cancer_dt2' is : (569, 30)
```

```
# visualize correlation barplot
plt.figure(figsize = (16,5))
ax = sns.barplot(cancer_dt2.corrwith(cancer_dt.target).index, cancer_dt2.corrwith(cancer_dt.t
ax.tick_params(labelrotation = 90)
```



Data Preprocessing Split DataFrame in train and test

```
# input variable
X = cancer_dt.drop(['target'], axis = 1)
X.head(6)
```

	mean radius	mean texture	mean perimeter	mean area	mea smoothnes
0	17.99	10.38	122.80	1001.0	0.1184
1	20.57	17.77	132.90	1326.0	0.0847
2	19.69	21.25	130.00	1203.0	0.1096
3	11.42	20.38	77.58	386.1	0.1425
4	20.29	14.34	135.10	1297.0	0.1003
5	12.45	15.70	82.57	477.1	0.1278

```
# output variable
y = cancer_dt['target']
y.head(6)
```

```
0    0.0
1    0.0
2    0.0
3    0.0
4    0.0
5    0.0
Name: target, dtype: float64
```

```
# split dataset into train and test
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state= 5)
```

Feature Scaling Converting different units and magnitude data in one unit.

```
# Feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
```

Breast Cancer Detection Machine Learning Model Building

We have clean data to build the Ml model. But which Machine learning algorithm is best for the data we have to find. The output is a categorical format so we will use supervised classification machine learning algorithms.

To build the best model, we have to train and test the dataset with multiple Machine Learning algorithms then we can find the best ML model. So let's try.

First, we need to import the required packages.

```
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
```

Support Vector Classifier

```
# Support vector classifier
from sklearn.svm import SVC
svc_classifier = SVC()
svc_classifier.fit(X_train, y_train)
y_pred_scv = svc_classifier.predict(X_test)
accuracy_score(y_test, y_pred_scv)
```

```
0.9385964912280702
```

```
# Train with Standard scaled Data
svc_classifier2 = SVC()
svc_classifier2.fit(X_train_sc, y_train)
y_pred_svc_sc = svc_classifier2.predict(X_test_sc)
```

```
y_pred_svc_sc = svc_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_svc_sc)
```

```
0.9649122807017544
```

Logistic Regression

```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr_classifier = LogisticRegression(random_state = 51, penalty = 'l2')
lr_classifier.fit(X_train, y_train)
y_pred_lr = lr_classifier.predict(X_test)
accuracy_score(y_test, y_pred_lr)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/l
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or
    https://scikit-learn.org/stable/modules/prep
Please also refer to the documentation for alter
    https://scikit-learn.org/stable/modules/line
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE
0.956140350877193
```

```
# Train with Standard scaled Data
lr_classifier2 = LogisticRegression(random_state = 51, penalty = 'l2')
lr_classifier2.fit(X_train_sc, y_train)
y_pred_lr_sc = lr_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_lr_sc)
```

```
0.45614035087719296
```

K – Nearest Neighbor Classifier

```
# K – Nearest Neighbor Classifier
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_classifier.fit(X_train, y_train)
y_pred_knn = knn_classifier.predict(X_test)
accuracy_score(y_test, y_pred_knn)
```

```
0.9385964912280702
```

```
# Train with Standard scaled Data
knn_classifier2 = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_classifier2.fit(X_train_sc, y_train)
```

```
y_pred_knn_sc = knn_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_knn_sc)
```

```
0.5789473684210527
```

Naive Bayes Classifier

```
# Naive Bayes Classifier
from sklearn.naive_bayes import GaussianNB
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred_nb = nb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_nb)
```

```
0.9473684210526315
```

```
# Train with Standard scaled Data
nb_classifier2 = GaussianNB()
nb_classifier2.fit(X_train_sc, y_train)
y_pred_nb_sc = nb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_nb_sc)
```

```
0.9385964912280702
```

Decision Tree Classifier

```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
dt_classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 51)
dt_classifier.fit(X_train, y_train)
y_pred_dt = dt_classifier.predict(X_test)
accuracy_score(y_test, y_pred_dt)
```

```
0.9473684210526315
```

```
# Train with Standard scaled Data
dt_classifier2 = DecisionTreeClassifier(criterion = 'entropy', random_state = 51)
dt_classifier2.fit(X_train_sc, y_train)
y_pred_dt_sc = dt_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_dt_sc)
```

```
0.7543859649122807
```

Random Forest Classifier

```
# Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf_classifier = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state
rf_classifier.fit(X_train, y_train)
y_pred_rf = rf_classifier.predict(X_test)
accuracy_score(y_test, y_pred_rf)
```

```
0.9736842105263158
```

```
# Train with Standard scaled Data
rf_classifier2 = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_stat
rf_classifier2.fit(X_train_sc, y_train)
y_pred_rf_sc = rf_classifier.predict(X_test_sc)
accuracy_score(y_test, y_pred_rf_sc)
```

```
0.7543859649122807
```

Adaboost Classifier

```
# Adaboost Classifier
from sklearn.ensemble import AdaBoostClassifier
adb_classifier = AdaBoostClassifier(DecisionTreeClassifier(criterion = 'entropy', random_stat
                                    n_estimators=2000,
                                    learning_rate=0.1,
                                    algorithm='SAMME.R',
                                    random_state=1,)
adb_classifier.fit(X_train, y_train)
y_pred_adb = adb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_adb)
```

```
0.9473684210526315
```

```
# Train with Standard scaled Data
adb_classifier2 = AdaBoostClassifier(DecisionTreeClassifier(criterion = 'entropy', random_sta
                                    n_estimators=2000,
                                    learning_rate=0.1,
                                    algorithm='SAMME.R',
                                    random_state=1,)
adb_classifier2.fit(X_train_sc, y_train)
y_pred_adb_sc = adb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_adb_sc)
```

```
0.9473684210526315
```

XGBoost Classifier

```
# XGBoost Classifier
from xgboost import XGBClassifier
xgb_classifier = XGBClassifier()
xgb_classifier.fit(X_train, y_train)
y_pred_xgb = xgb_classifier.predict(X_test)
accuracy_score(y_test, y_pred_xgb)
```

```
0.9824561403508771
```

```
# Train with Standard scaled Data
xgb_classifier2 = XGBClassifier()
xgb_classifier2.fit(X_train_sc, y_train)
y_pred_xgb_sc = xgb_classifier2.predict(X_test_sc)
accuracy_score(y_test, y_pred_xgb_sc)
```

```
0.9824561403508771
```

XGBoost Parameter Tuning

Randomized Search

```
# XGBoost classifier most required parameters
params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
}

# Randomized Search
from sklearn.model_selection import RandomizedSearchCV
random_search = RandomizedSearchCV(xgb_classifier, param_distributions=params, scoring= 'roc_
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, total
[Parallel(n_jobs=-1)]: Using backend LokyBackend
[Parallel(n_jobs=-1)]: Done  36 tasks      | ela
[Parallel(n_jobs=-1)]: Done  50 out of  50 | ela
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=XGBClassifier(base_
                                           colsa
                                           colsa
                                           colsa
                                           learn
                                           max_d
                                           missi
                                           n_job
```

```
                                          objec
                                          rando
                                          reg_l
                                          verbo
                        iid='deprecated', n_iter=10,
                        param_distributions={'colsamp

                                           'gamma':
                                           'learnin

                                           'max_dep

                                           'min_chi
                        pre_dispatch='2*n_jobs', rand
                        return_train_score=False, sco
```

```
random_search.best_params_
```

```
{'colsample_bytree': 0.4,
 'gamma': 0.1,
 'learning_rate': 0.1,
 'max_depth': 15,
 'min_child_weight': 1}
```

```
random_search.best_estimator_
```

```
XGBClassifier(base_score=0.5, booster='gbtree',
              colsample_bynode=1, colsample_bytr
              learning_rate=0.1, max_delta_step=
              min_child_weight=1, missing=None,
              nthread=None, objective='binary:lo
              reg_alpha=0, reg_lambda=1, scale_p
              silent=None, subsample=1, verbosit
```

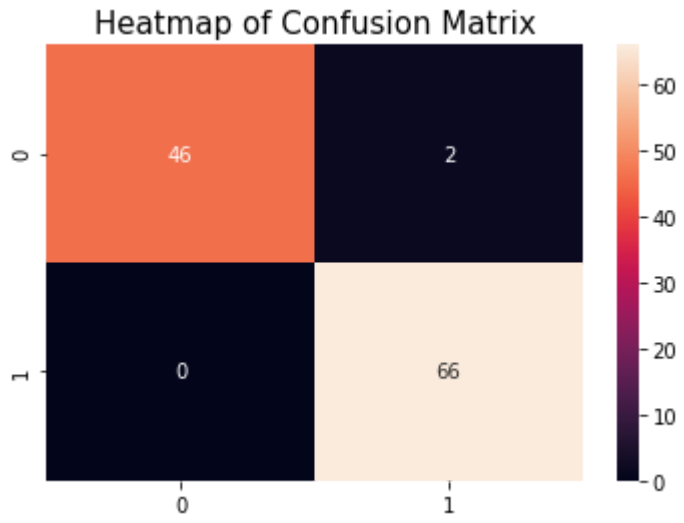
```
# training XGBoost classifier with best parameters
xgb_classifier_pt = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=0.4, gamma=0.2,
       learning_rate=0.1, max_delta_step=0, max_depth=15,
       min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
       nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
       silent=None, subsample=1, verbosity=1)

xgb_classifier_pt.fit(X_train, y_train)
y_pred_xgb_pt = xgb_classifier_pt.predict(X_test)
```

```
accuracy_score(y_test, y_pred_xgb_pt)
```

```
0.9824561403508771
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred_xgb_pt)
plt.title('Heatmap of Confusion Matrix', fontsize = 15)
sns.heatmap(cm, annot = True)
plt.show()
```



Classification Report of Model

```
print(classification_report(y_test, y_pred_xgb_pt))
```

```
                  precision    recall  f1-score   su

             0.0       1.00      0.96      0.98
             1.0       0.97      1.00      0.99

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

Cross-validation of the ML model

To find the ML model is overfitted, under fitted or generalize doing cross-validation.

```
# Cross validation
```

```
from sklearn.model_selection import cross_val_score
cross_validation = cross_val_score(estimator = xgb_model_pt2, X = X_train_sc, y = y_train, cv
print("Cross validation of XGBoost model = ",cross_validation)
print("Cross validation of XGBoost model (in mean) = ",cross_validation.mean())
```

```
from sklearn.model_selection import cross_val_score
cross_validation = cross_val_score(estimator = xgb_classifier_pt, X = X_train_sc,y = y_train,
print("Cross validation accuracy of XGBoost model = ", cross_validation)
print("\nCross validation mean accuracy of XGBoost model = ", cross_validation.mean())
```

```
---------------------------------------------
---------------------------
NameError
Traceback (most recent call last)
<ipython-input-66-43c143b7eb02> in <module>()
      1 # Cross validation
      2 from sklearn.model_selection import
cross_val_score
----> 3 cross_validation =
cross_val_score(estimator = xgb_model_pt, X =
X_train_sc, y = y_train, cv = 10)
      4 print("Cross validation of XGBoost
model = ",cross_validation)
```

The mean accuracy value of cross-validation is 96.24% and XGBoost model accuracy is 98.24%. It showing XGBoost is slightly overfitted but when training data will more it will generalized model.

Save the Machine Learning model

After completion of the Machine Learning project or building the ML model need to deploy in an application. To deploy the ML model need to save it first. To save the Machine Learning project we can use the pickle or joblib package.

Here, we will use pickle, Use anyone which is better for you.

```
## Pickle
import pickle

# save model
pickle.dump(xgb_classifier_pt, open('breast_cancer_detector.pickle', 'wb'))
```

```
# load model
breast_cancer_detector_model = pickle.load(open('breast_cancer_detector.pickle', 'rb'))

# predict the output
y_pred = breast_cancer_detector_model.predict(X_test)

# confusion matrix
print('Confusion matrix of XGBoost model: \n',confusion_matrix(y_test, y_pred),'\n')

# show the accuracy
print('Accuracy of XGBoost model = ',accuracy_score(y_test, y_pred))
```

```
Confusion matrix of XGBoost model:
 [[46  2]
 [ 0 66]]

Accuracy of XGBoost model =  0.9824561403508771
```

Congratulation!!!!!!!

We have completed the Machine learning Project successfully with 98.24% accuracy which is great for 'Breast Cancer Detection using Machine learning' project. Now, we are ready to deploy our ML model in the healthcare project.

Click on the below button to download the ' Breast Cancer Detection ' Machine Learning end to end project in the Jupyter Notebook file.