

## Problem Statement:

Our aim is to diagnose patients with breast cancer by analyzing the data of patients and classifying them into two categories, having diagnosis results as : (1) Benign (**B**) and (2) Malignant (**M**).

## Libraries used:

```
In [1]: # Pandas library in python is used to read in the csv file.
import pandas as pd
# Numpy library for numerical computations
import numpy as np
# Seaborn, Matplotlib for graphs and plots to gets insights of data
import seaborn as sns
import matplotlib.pyplot as plt
# Split data into testing and training data using sklearn's model_selecion module
from sklearn.model_selection import train_test_split
# Different evaluation matrices import from sklearn.metrics
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

- 1) NumPy : The fundamental package for scientific computing with Python
- 2) Pandas : An open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- 3) sklearn.model\_selection :
  - Split arrays or matrices into random train and test subsets
  - Selecting optimal features
- 4) matplotlib.pyplot : Provides a MATLAB-like plotting framework.
- 5) Seaborn : Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical Graphics.
- 6) XGBoost : XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.
- 7) sklearn.metrics :
  - Accuracy score: In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y\_true.
  - Classification report: Build a report showing the main classification metrics
  - Confusion matrix: Compute confusion matrix to evaluate the accuracy of classification.

## Details of Data:

The data was collected from Kaggle:

<https://www.kaggle.com/uciml/breast-cancer-wisconsin-data>

No. of rows : 569

No. of columns : 33

```
In [3]: #shape command gives number of rows/samples/examples and number of columns/features/predictors in dataset
        #(rows,columns)
        df.shape
```

```
Out[3]: (569, 33)
```

Attributes:

```
In [5]: #info method provides information about dataset like
        #total values in each column, null/not null, datatype, memory occupied etc
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
id                569 non-null int64
diagnosis         569 non-null object
radius_mean       569 non-null float64
texture_mean      569 non-null float64
perimeter_mean    569 non-null float64
area_mean         569 non-null float64
smoothness_mean   569 non-null float64
compactness_mean  569 non-null float64
concavity_mean    569 non-null float64
concave points_mean 569 non-null float64
symmetry_mean     569 non-null float64
fractal_dimension_mean 569 non-null float64
radius_se         569 non-null float64
texture_se        569 non-null float64
perimeter_se      569 non-null float64
area_se           569 non-null float64
smoothness_se     569 non-null float64
compactness_se    569 non-null float64
concavity_se      569 non-null float64
concave points_se 569 non-null float64
symmetry_se       569 non-null float64
fractal_dimension_se 569 non-null float64
radius_worst      569 non-null float64
texture_worst     569 non-null float64
perimeter_worst   569 non-null float64
area_worst        569 non-null float64
smoothness_worst  569 non-null float64
compactness_worst 569 non-null float64
concavity_worst   569 non-null float64
concave points_worst 569 non-null float64
symmetry_worst    569 non-null float64
fractal_dimension_worst 569 non-null float64
Unnamed: 32       0 non-null float64
dtypes: float64(31), int64(1), object(1)
```

1) ID number

The unique ID allotted to each patient to differentiate between them.

2) Diagnosis (M = malignant, B = benign)

The target variable .i.e the conclusion if the tumour is malignant or benign.

3-33) Ten real-valued features are computed for each cell nucleus:

a) Radius

The mean of distances from center to points on the perimeter.

b) Texture

The standard deviation of gray-scale values.

c) Perimeter

d) Area

e) Smoothness

The local variation in radius lengths.

f) Compactness

$\text{Perimeter}^2 / \text{Area} - 1.0$

g) Concavity

The severity of concave portions of the contour.

h) Concave Points

Number of concave portions of the contour.

i) Symmetry

j) 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.

All feature values are recorded with four significant digits.

Missing attribute values: There are no missing values in the dataset.

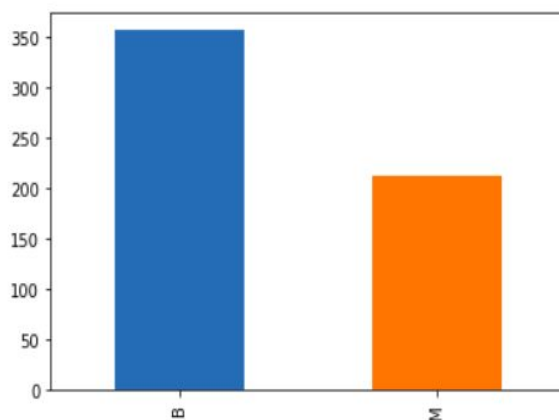
```
In [14]: # To check presence of missing (NaN) values
df.isnull().any()
```

```
Out[14]: diagnosis                False
radius_mean                      False
texture_mean                     False
perimeter_mean                   False
area_mean                        False
smoothness_mean                  False
compactness_mean                 False
concavity_mean                   False
concave points_mean              False
symmetry_mean                    False
fractal_dimension_mean           False
radius_se                        False
texture_se                       False
perimeter_se                     False
area_se                          False
smoothness_se                    False
compactness_se                   False
concavity_se                     False
concave points_se                False
symmetry_se                      False
fractal_dimension_se             False
radius_worst                     False
texture_worst                    False
perimeter_worst                  False
area_worst                       False
smoothness_worst                 False
compactness_worst                False
concavity_worst                  False
concave points_worst             False
symmetry_worst                   False
fractal_dimension_worst          False
dtype: bool
```

Class distribution: 357 benign, 212 malignant

```
In [7]: # Understanding the frequency distribution of data in target variable "diagnosis"
df.diagnosis.value_counts().plot(kind="bar")
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7f29404b27b8>
```





```
In [8]: # Frequency of unique values of target variable
df.diagnosis.value_counts()
```

```
Out[8]: B    357
        M    212
        Name: diagnosis, dtype: int64
```

## Preprocessing of Data:

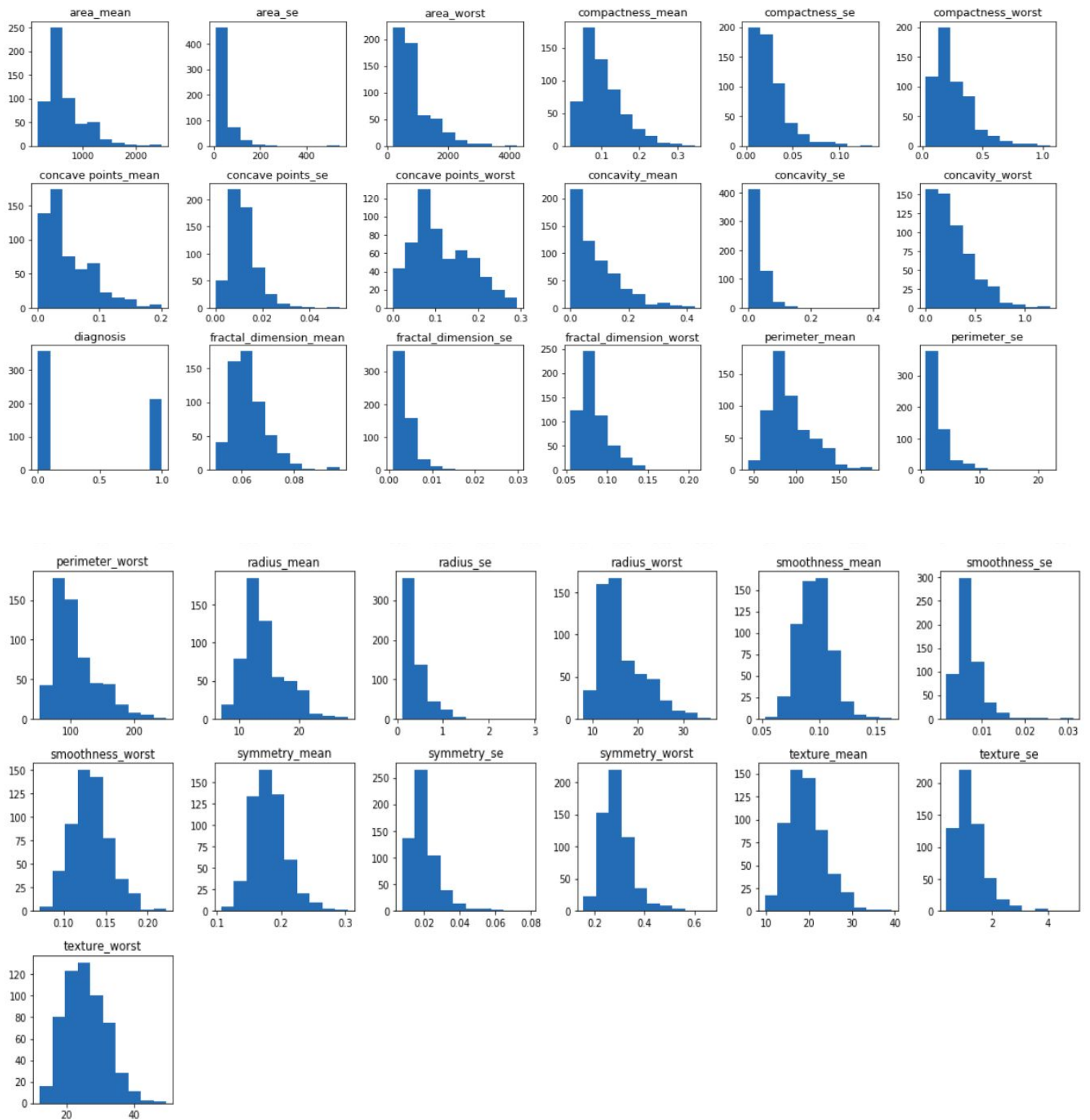
### 1. Visualization

Visual representation of data using histograms for each feature to get insights about the frequency of various feature values in data.

```
In [15]: # Getting insight of data distribution based on frequency of unique values in the features
df.hist(bins=10,figsize=(20,20),grid=False)
```

```
Out[15]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f2940132b38>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f294016e358>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f2940115588>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f29400be7f0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f29400e6a58>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f2940091cc0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f294003cf28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f294006d208>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f294006d240>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293ffb6a0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293ffea908>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293ff93b70>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f293ff3ddd8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fef3080>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293ffa2e8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fec1550>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fee97b8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fe95a20>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fe3fc88>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fe68ef0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fed1d98>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fdc6400>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fd6f668>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fd9b8d0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fd43b38>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fd6dda0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fd22048>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fccb2b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fc72518>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f294049a6a0>],
[<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fc3b9b0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fc63f28>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fc134e0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fbbba58>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293f4fd0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x7f293fb96588>]],
dtype=object)
```

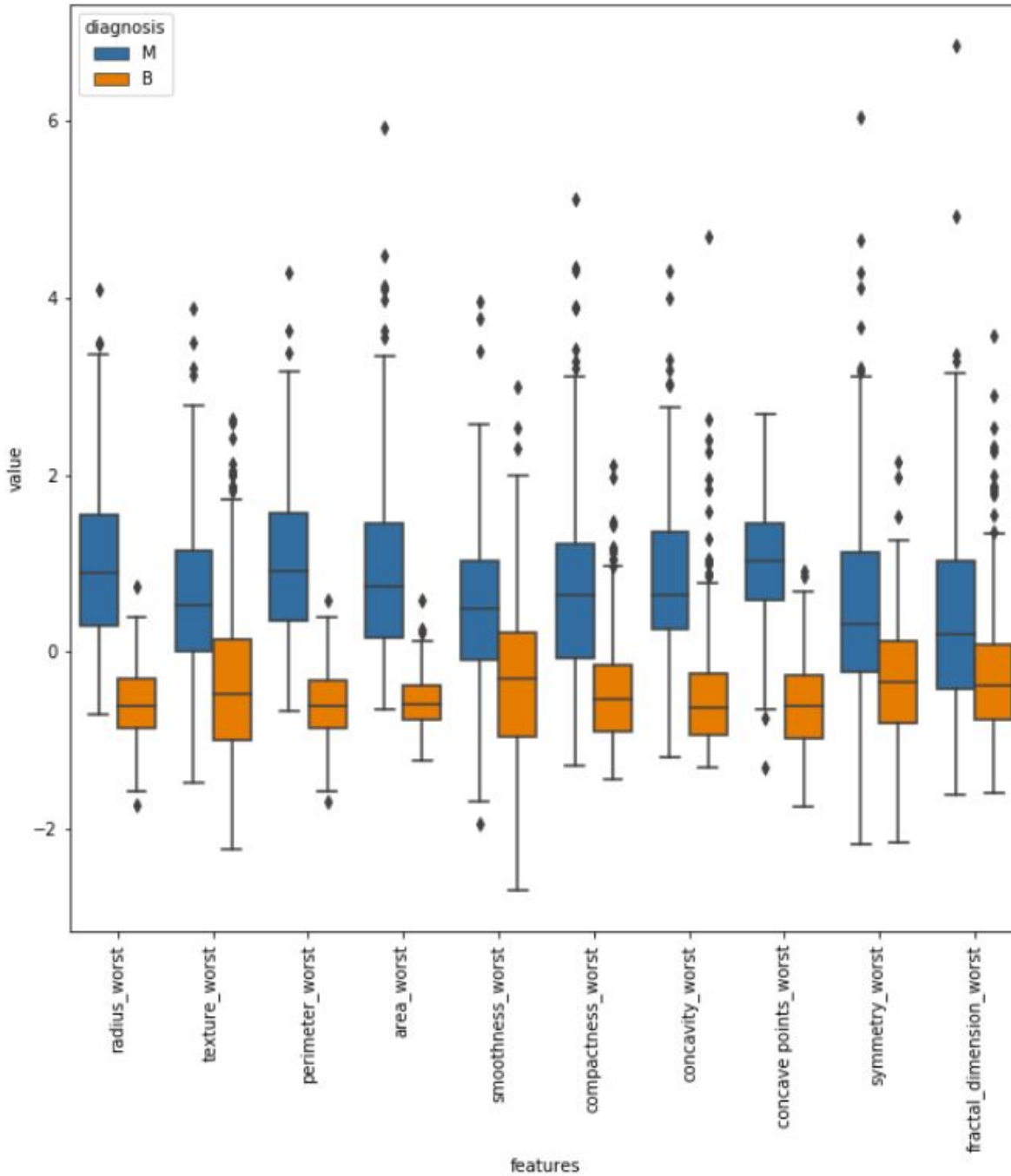
# Breast Cancer Diagnosis Wisconsin Data Set



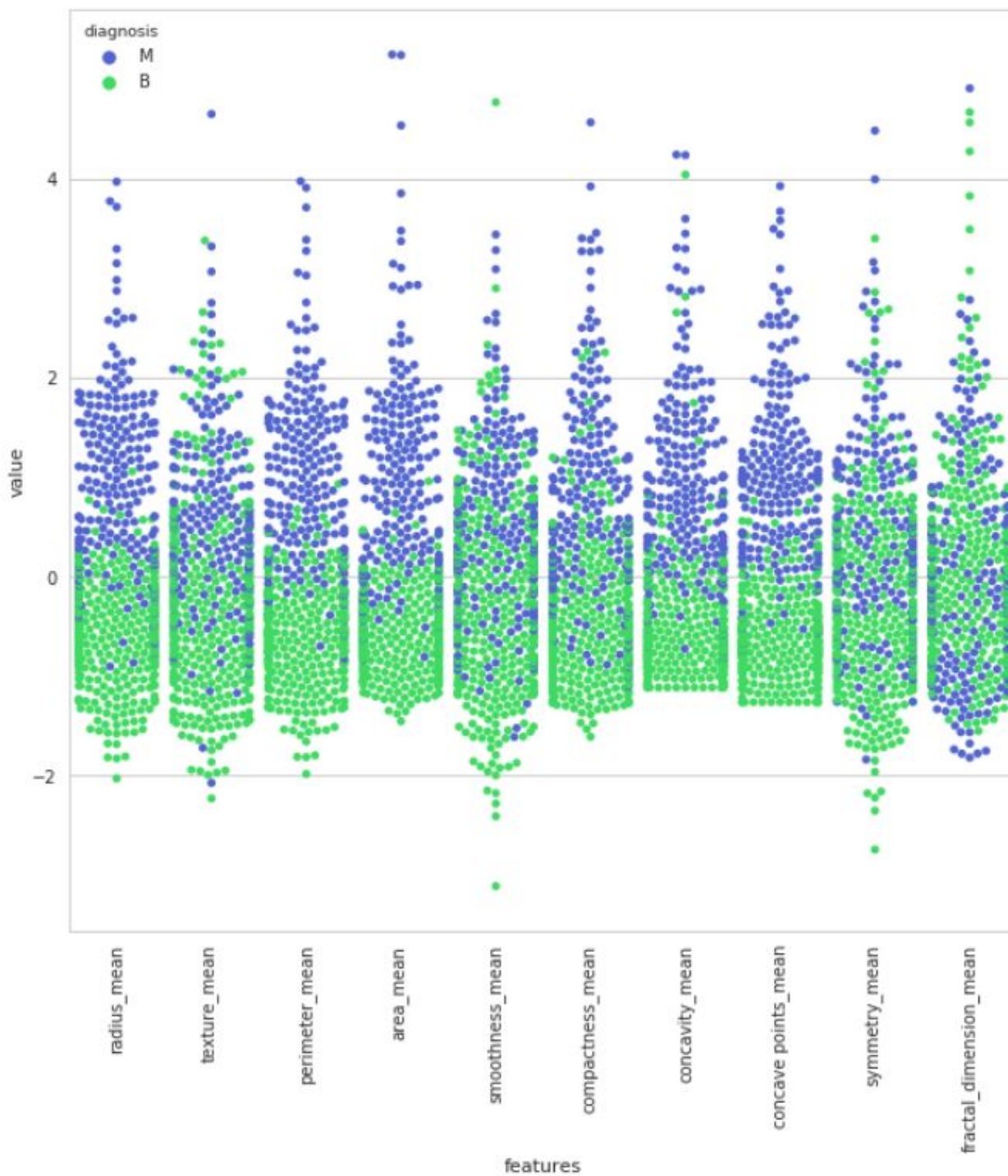
**Box Plots for attributes:**

It is used to visualize the distribution of values of attributes and to see the outliers.

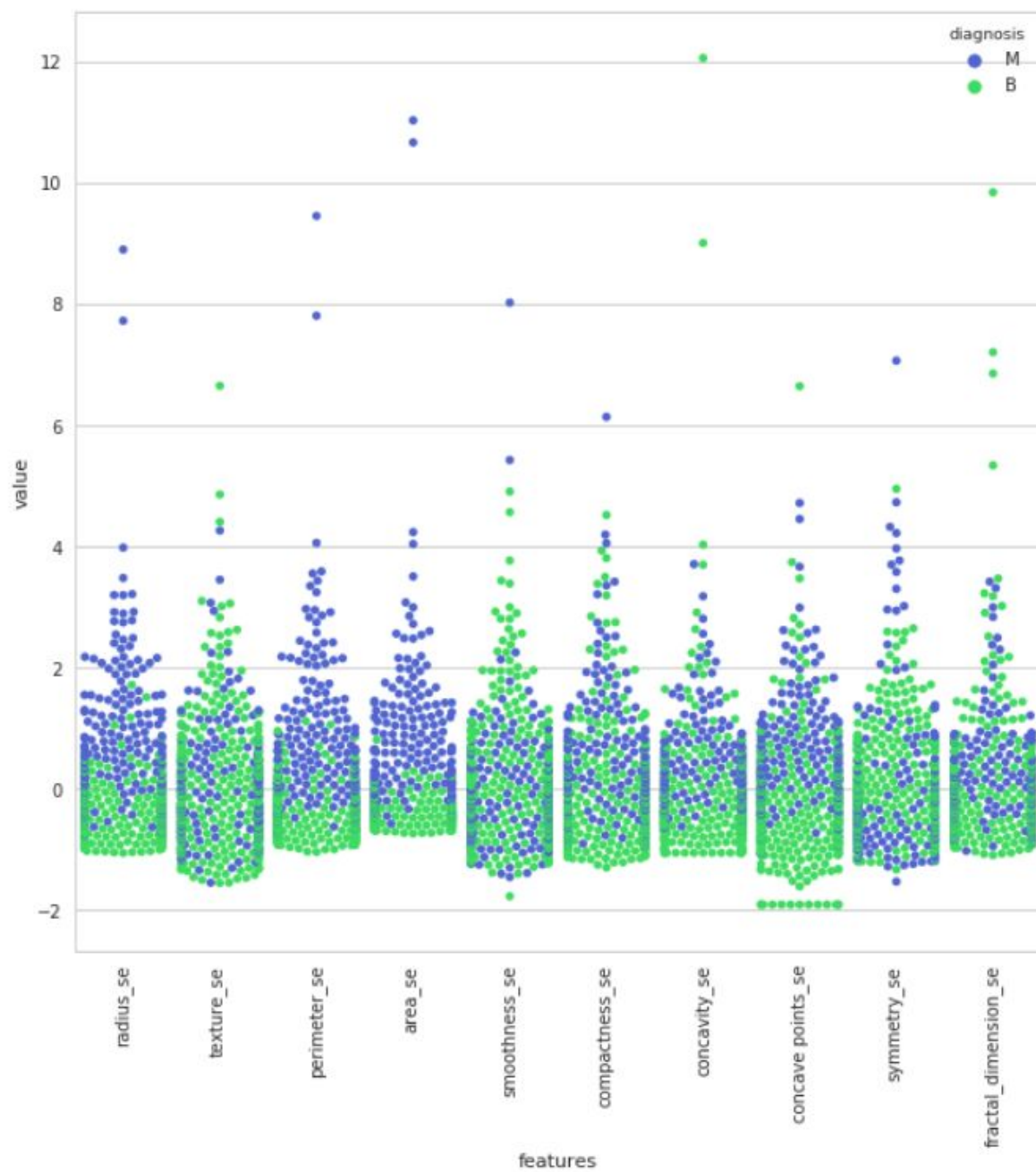
As an example, we have plotted box plots for few of the attributes.

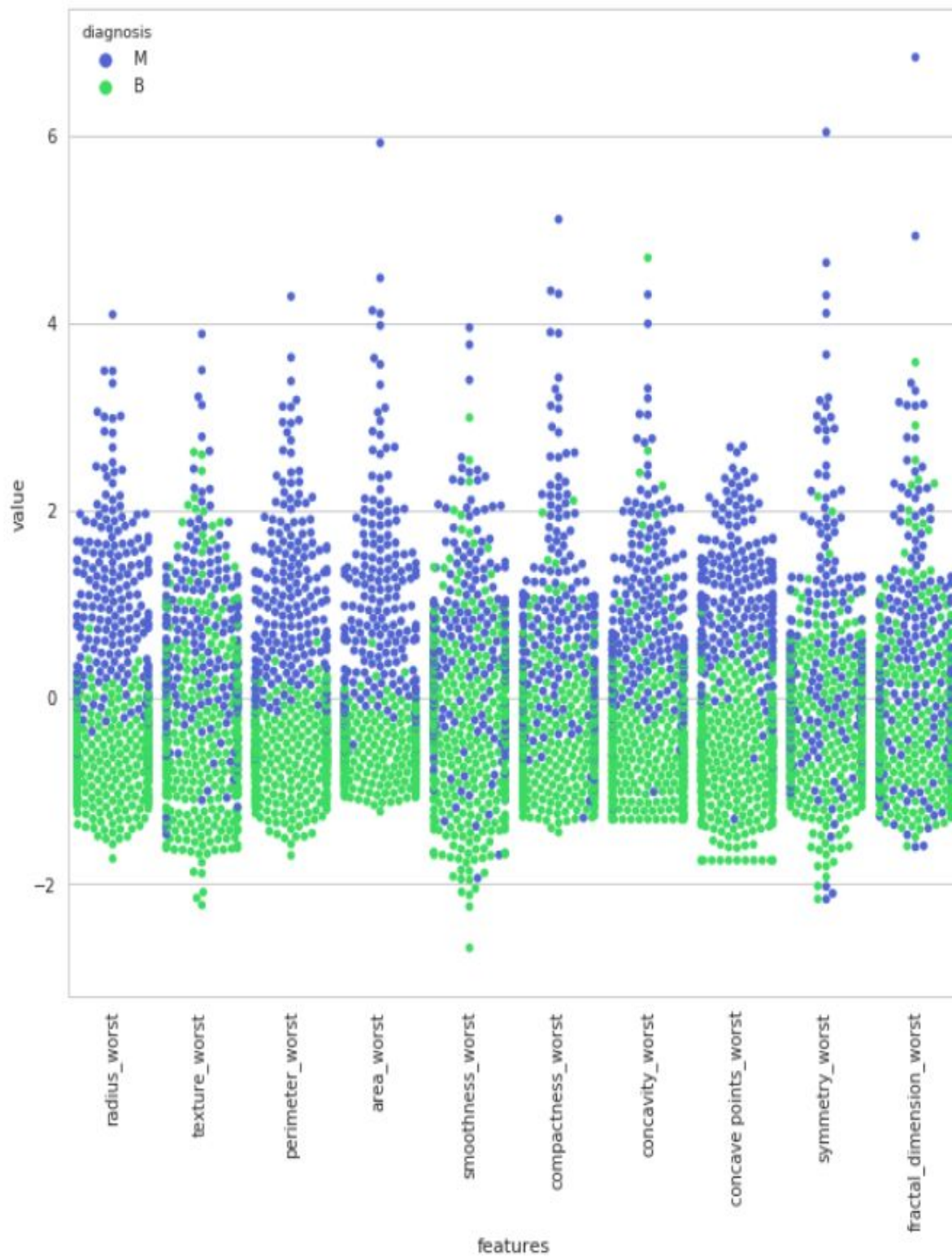




**Swarm Plots:**







## 2. Mapping string to numeric data

The feature “diagnosis” has values in the form of string (‘object’) which is mapped to numeric form for analysis.

The mapping function is:

‘B’  $\rightarrow$  0

‘M’  $\rightarrow$  1

```
In [9]: # Mapping string value feature to numerical value feature using map function
cancer_mapping = {'B':0, 'M':1}
df.diagnosis = df.diagnosis.map(cancer_mapping)
```

## 3. Missing values

The data contains no missing values for any of the attributes as inferred from the in the figure below. So, we don’t have to deal with them.

```
▶ In [14]: # To check presence of missing (NaN) values
Run this cell df.isnull().any()
```

```
Out[14]: diagnosis                False
radius_mean                      False
texture_mean                     False
perimeter_mean                   False
area_mean                       False
smoothness_mean                  False
compactness_mean                 False
concavity_mean                   False
concave points_mean              False
symmetry_mean                    False
fractal_dimension_mean           False
radius_se                        False
texture_se                       False
perimeter_se                     False
area_se                          False
smoothness_se                    False
compactness_se                   False
concavity_se                     False
concave points_se                False
symmetry_se                      False
```

#### **4. Duplicate data**

The data contains no duplicate instances of patient data.

```
In [13]: # Checking if any duplicate values in the df
df.duplicated()

Out[13]: 0    False
         1    False
         2    False
         3    False
         4    False
         5    False
         6    False
         7    False
         8    False
         9    False
        10    False
        11    False
        12    False
        13    False
        14    False
        15    False
        16    False
        17    False
        18    False
        19    False
```

#### **5. Feature subset selection**

The feature “id” is a unique attribute for each patient and it is irrelevant to the process of classification, so it can be eliminated.

```
In [10]: # Checking if any duplicates in "id" column by finding unique values and if their frequency is greater than 1
df.id.value_counts().unique()

Out[10]: array([1])
```

This gives that all instances in dataset have unique id. Meaning it has little impact on the prediction of target value.



```
In [11]: # Since "id" column has no direct impact on target value (just patient id or index)
df.drop('id',axis=1,inplace=True)
df.drop('Unnamed: 32',axis=1,inplace=True)
```

So, we drop the features : 'id' and 'unnamed 32'

Resultant columns in the data :

```
In [12]: # Names of columns in the df
df.columns
```

```
Out[12]: Index(['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'],
              dtype='object')
```

## 6. Dimensionality reduction

The features which are highly correlated (show dependency) with the target variable ('diagnosis') are highly relevant for our classification problem.

We can remove irrelevant attributes in order to reduce the size of data for easier computation.

Here, y stores the target or class variable & x stores the non-class attributes

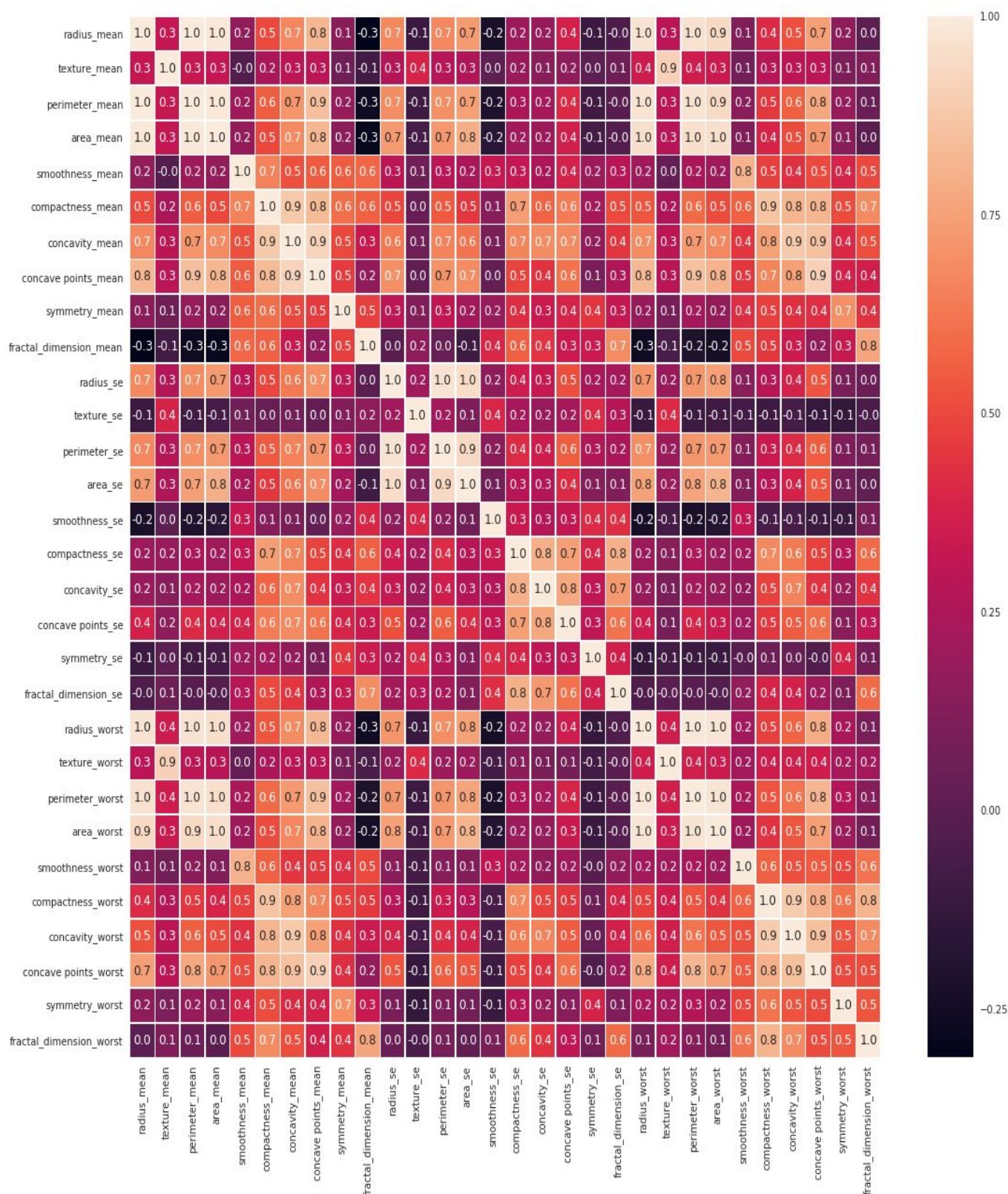
```
In [16]: y = df['diagnosis']
X = df[['radius_mean', 'texture_mean', 'perimeter_mean',
        'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
        'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se', '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']]
```

The irrelevant attributes can be found by computing the correlation among non-class attributes, and then we can reduce a subset of highly co-relevant non-class attributes to a single or less number of attributes which would reduce the size of data.

Therefore, we create a heatmap to display the correlation between all the features:

```
In [17]: #correlation map
f,ax = plt.subplots(figsize=(18, 18))
sns.heatmap(X.corr(), annot=True, linewidths=.5, fmt= '.1f',ax=ax)
```

# Breast Cancer Diagnosis Wisconsin Data Set





This heatmap visualizes the correlation between each pair of attribute in the dataset. We can select those attributes which have high correlation with each other according to a threshold value, and then we will implement the classification algorithm with a reduced set of attributes by taking various smaller subsets of the these highly correlated (dependent) attributes, and compare the results.

Here, we take **threshold value: 0.90**

From the heat plot we infer that the following features are the most related ( $\text{corr} > 0.9$ ) to the other features:

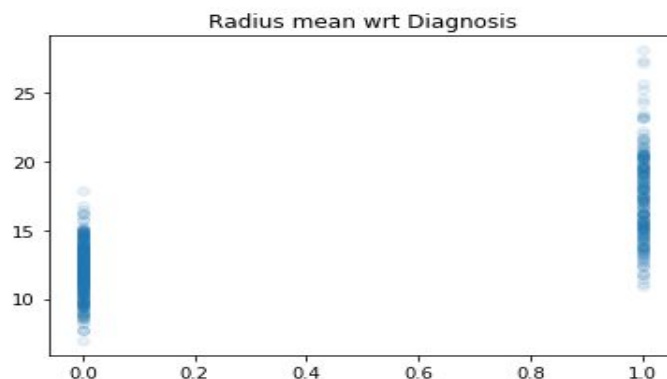
1. Radius\_mean
2. Perimeter\_mean
3. Area\_mean
4. Radius\_worst
5. Perimeter\_worst

```
In [18]: # The features with high correlation with other features are: (threshold = 90%)
high_corr_pts = X[['radius_mean', 'perimeter_mean', 'area_mean', 'radius_worst', 'perimeter_worst']]
```

Then we plot scatter plots for all the high correlation features with respect to the target feature(diagnosis):

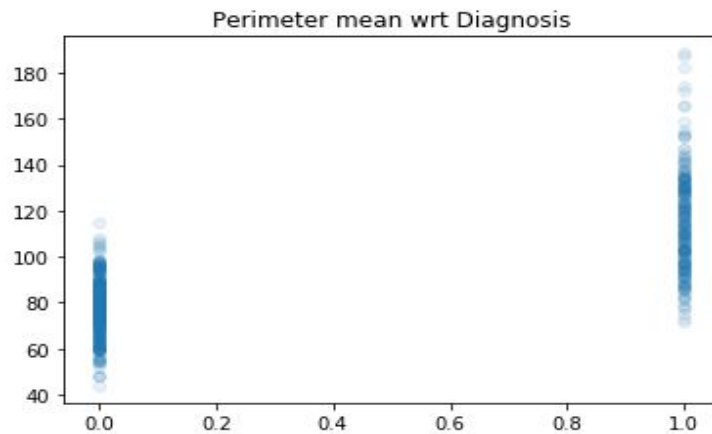
```
► In [19]: plt.scatter(df.diagnosis, df.radius_mean, alpha=0.1)
# here the plot has to be transparent so we need to pick low alpha value
plt.title("Radius mean wrt Diagnosis")
```

```
Out[19]: Text(0.5, 1.0, 'Radius mean wrt Diagnosis')
```



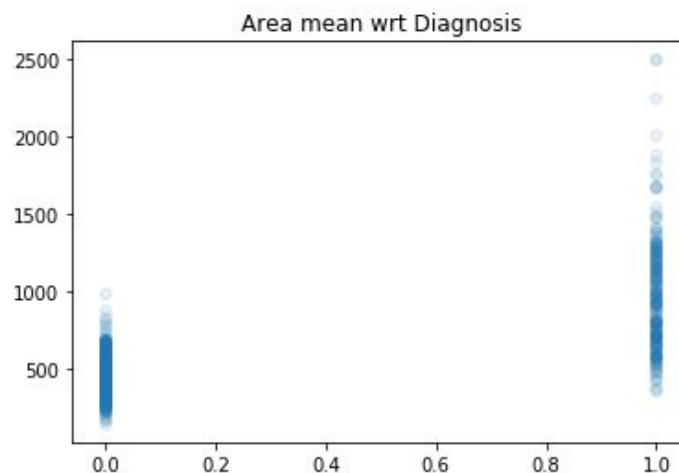
```
In [20]: plt.scatter(df.diagnosis, df.perimeter_mean, alpha=0.1)
# here the plot has to be transparent so we need to pic low alpha value
plt.title("Perimeter mean wrt Diagnosis")
```

```
Out[20]: Text(0.5, 1.0, 'Perimeter mean wrt Diagnosis')
```



```
In [22]: plt.scatter(df.diagnosis, df.area_mean, alpha=0.1)
# here the plot has to be transparent so we need to pic low alpha value
plt.title("Area mean wrt Diagnosis")
```

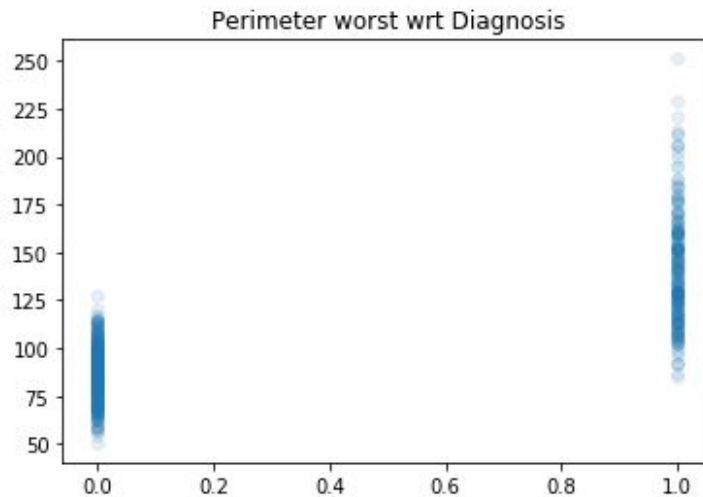
```
Out[22]: Text(0.5, 1.0, 'Area mean wrt Diagnosis')
```





```
In [23]: plt.scatter(df.diagnosis, df.perimeter_worst, alpha=0.1)
# here the plot has to be transparent so we need to pick low alpha value
plt.title("Perimeter worst wrt Diagnosis")
```

```
Out[23]: Text(0.5, 1.0, 'Perimeter worst wrt Diagnosis')
```



## Splitting of data into test and train sets

We now split the data into a test set and a training set.

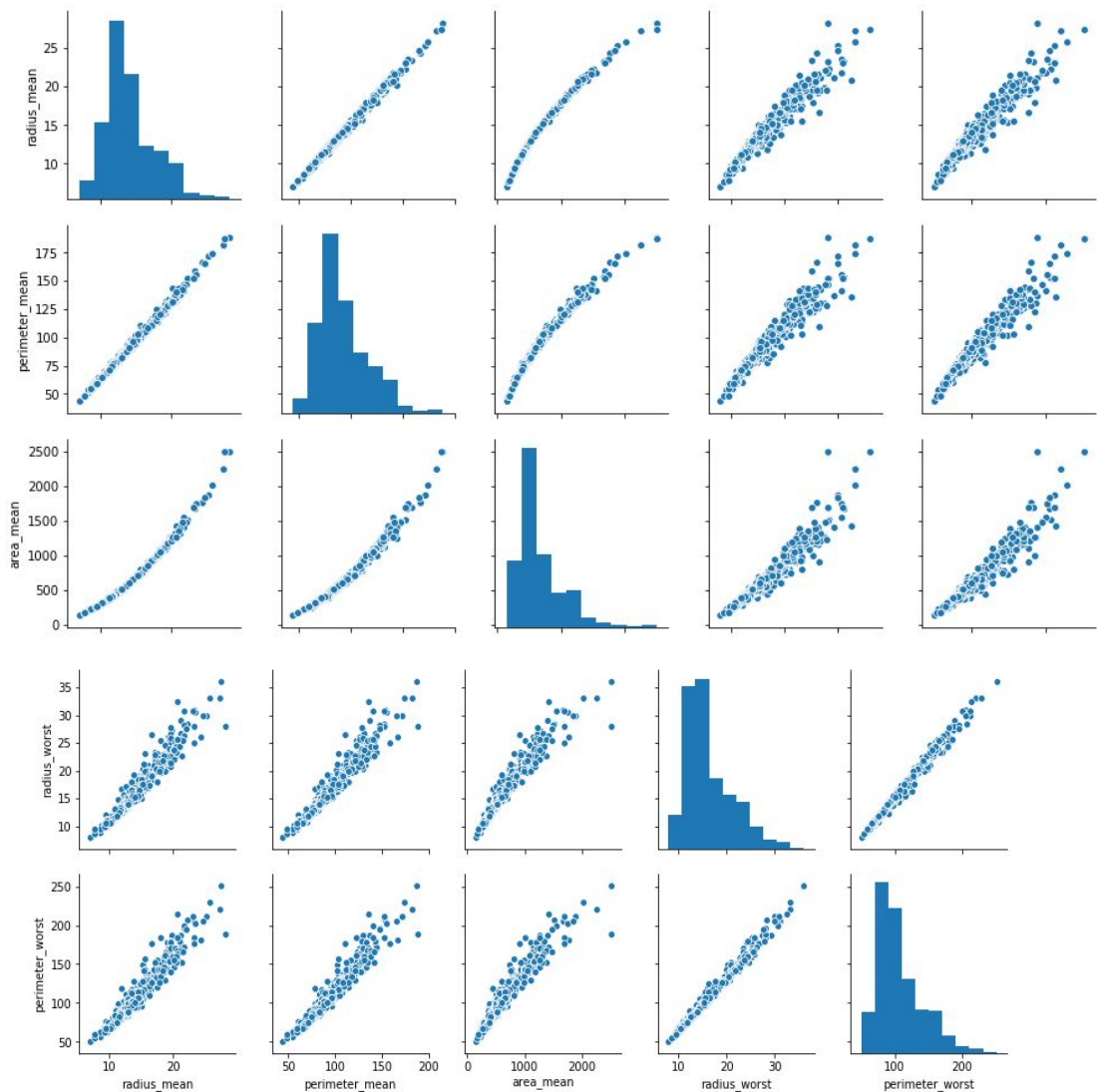
```
In [24]: # Splitting data with 40% test data and rest 60% is training data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

The test set size is chosen to be 40% of the whole data.

We then make a pairplot between all the high correlation features:

```
In [25]: sns.pairplot(high_corr_pts)
```

```
Out[25]: <seaborn.axisgrid.PairGrid at 0x7f293f558d30>
```



## Training the model and predicting the target variable:

We are using a gradient boosting framework provided by XGBoost to train the model. XGBoost is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms.

It's main goal is to push the extreme of the computation limits of machines to provide a *scalable*, *portable* and *accurate* for large scale tree boosting.

Features:

- Speed: it can automatically do parallel computation on Windows and Linux, with OpenMP. It is generally over 10 times faster than the classical gbm.
- Input Type: it takes several types of input data:
- Dense Matrix: R's dense matrix, i.e. matrix ;
- Sparse Matrix: R's sparse matrix, i.e. Matrix::dgCMatrix ;
- Data File: local data files ;
- xgb.DMatrix: its own class (recommended).
- Sparsity: it accepts sparse input for both tree booster and linear booster, and is optimized for sparse input ;
- Customization: it supports customized objective functions and evaluation functions.

Gradient boosting involves three elements:

1. A loss function to be optimized.
2. A weak learner to make predictions.
3. An additive model to add weak learners to minimize the loss function.

```
In [29]: import xgboost as xgb
```

```
In [30]: model_all =xgb.XGBClassifier()
```

```
In [31]: model_all.fit(X_train,y_train)
```

```
Out[31]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                        colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
                        max_depth=3, 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=True, subsample=1)
```

```
In [32]: pred=model_all.predict(X_test)
```

## Analyzing the predictions:

We find the accuracy and confusion matrix of the predictions:

```
In [33]: accuracy_score(y_test,pred)
```

```
Out[33]: 0.9517543859649122
```

```
In [34]: print(classification_report(y_test,pred))
```

	precision	recall	f1-score	support
0	0.93	0.99	0.96	139
1	0.99	0.89	0.93	89
micro avg	0.95	0.95	0.95	228
macro avg	0.96	0.94	0.95	228
weighted avg	0.95	0.95	0.95	228

```
In [35]: confusion_matrix(y_test,pred)
```

```
Out[35]: array([[138,  1],
                [ 10,  79]])
```

**We get an accuracy of 0.9517.**

## Finding the optimal features:

Now we find the optimal number of features and what those features are. We do this by using RFECV.



```
In [37]: from sklearn.feature_selection import RFECV

# The "accuracy" scoring is proportional to the number of correct classifications
clf = xgb.XGBClassifier()
rfecv = RFECV(estimator=clf, step=1, cv=4, scoring='accuracy') #5-fold cross-validation
rfecv = rfecv.fit(X_train, y_train)

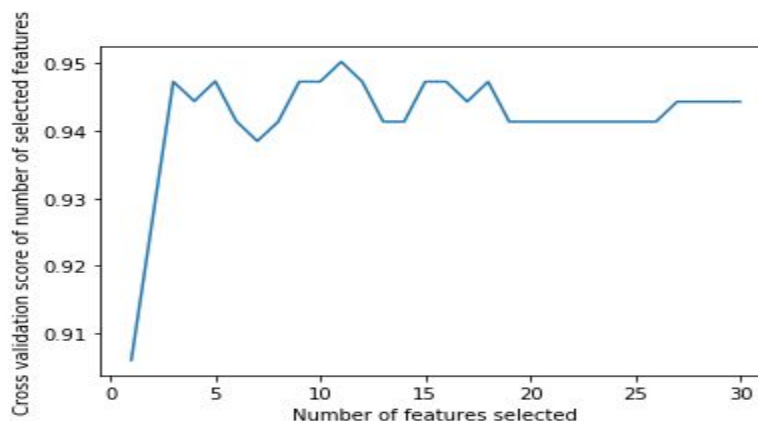
print('Optimal number of features :', rfecv.n_features_)
print('Best features :', X_train.columns[rfecv.support_])

Optimal number of features : 11
Best features : Index(['concave points_mean', 'area_se', 'compactness_se',
                      'fractal_dimension_se', 'radius_worst', 'texture_worst',
                      'perimeter_worst', 'area_worst', 'concavity_worst',
                      'concave points_worst', 'symmetry_worst'],
                      dtype='object')
```

**11 optimal features** are selected.

We now plot the number of features VS. cross-validation scores to see how it affects the predication:

```
In [38]: # Plot number of features VS. cross-validation scores
import matplotlib.pyplot as plt
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score of number of selected features")
plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
plt.show()
```



The graph shows how the score varies with varying number of features. The score is highest when the number of features is 11.

We now rank the features by their importance:



## Training and analyzing the model with only the optimal features:

We train the model and predict the target variable using only the selected optimal features (11) and get an **accuracy of 0.9649**

```
In [99]: X_train_optimal = X_train[['texture_mean', 'concave points_mean', 'area_se', 'compactness_se',
    'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst',
    'area_worst', 'concavity_worst', 'concave points_worst']]
X_test_optimal = X_test[['texture_mean', 'concave points_mean', 'area_se', 'compactness_se',
    'concave points_se', 'radius_worst', 'texture_worst', 'perimeter_worst',
    'area_worst', 'concavity_worst', 'concave points_worst']]
# Load classifier in model_optimal (considering all optimal features)
model_optimal = xgb.XGBClassifier()
# Training model
model_optimal.fit(X_train_optimal, y_train)
# Predict based on all testing features
pred = model_optimal.predict(X_test_optimal)
# Get accuracy of model
accuracy_score(y_test, pred)
```

Out[99]: 0.9649122807017544

## Why we chose this algorithm:

1. A benefit of the gradient boosting framework is that a new boosting algorithm does not have to be derived for each loss function that may want to be used, instead, it is a generic enough framework that any differentiable loss function can be used.
2. Regression trees are used that output real values for splits and whose output can be added together, allowing subsequent models outputs to be added and “correct” the residuals in the predictions. Trees are constructed in a greedy manner, choosing the best split points based on purity scores like Gini or to minimize the loss.
3. Instead of parameters, we have weak learner sub-models or more specifically decision trees.
4. After calculating the loss, to perform the gradient descent procedure, we must add a tree to the model that reduces the loss (i.e. follow the gradient). We do this by parameterizing the tree, then modify the parameters of the tree and move in the right direction by (reducing the residual loss).

5. Xgboost manages only numeric vectors. What to do when you have categorical data?  
A simple method to convert categorical variable into numeric vector is One Hot Encoding.
6. Xgboost can scale with hundreds of workers (with each worker utilizing multiple processors) smoothly and solve machine learning problems involving Terabytes of real world data.

Other four enhancements to basic gradient boosting:

1. Tree Constraints
2. Shrinkage
3. Random sampling
4. Penalized Learning

## Conclusion:

- We started by exploring the data with EDA(*exploratory data analysis*) methods. We analyzed the data types of the features and converted the string data type to integer( using hot encoding ).
- We then checked for missing values and duplicates and dealt with them.
- After performing preliminary analysis, we plotted the data to get more insights.
- Further we dropped certain features that, upon analysis, we found to be irrelevant to the target variable.
- We found the correlation of all the features with the target variable as well as among each other. We then found the 11 most optimal feature.

As we can observe, we have *reduced the number of features from 30 to 11*, and have also *increased the accuracy of our model from 95.17% To 96.49%* during the process.