Project Increment -1

Title: Breast Cancer Prediction Using Machine Learning Algorithms

Team Members:

- Dinesh Bhogadi
- Veena Ravuri
- Kota Sai Teja Jana

1. INTRODUCTION

According to statistics from throughout the world, breast cancer is one of the most prevalent malignancies among women and accounts for most new cancer cases and cancer-related deaths, making it a serious public health issue today.

By encouraging patients to receive prompt therapeutic care, an early diagnosis of Breast cancer can considerably enhance the prognosis and likelihood of survival. Patients can avoid receiving therapies they don't need by receiving a more precise classification of benign tumors. Therefore, significant study has been done on determining the proper breast cancer diagnosis and grouping patients into benign or malignant groups. Machine learning is widely acknowledged as the preferred technology for breast cancer pattern classification and forecast modeling due to its distinct benefits in the discovery of important characteristics from complex BC datasets.

2. IDEOLOGY

In this project we will analyze the data set of Cancer Wisconsin diagnostic dataset. We will analyze the dataset and perform the basic EDA on the dataset. We will Implement the below five algorithms on the dataset:

- RandomForestClassifier
- SupportVectorMachine
- DecisionTree
- k- Nearest Neighbors (KNN)
- Logistic regression

Once the results are obtained a performance comparison, evaluation is done between various classifiers. The major goal of this study is to identify which machine learning algorithm is suitable for the successful predicting and diagnosis of breast cancer in terms of Accuracy, precision, and confusion matrices. This analysis is done in python using Jupyter notebook. Scikit-learn and matplot lib are the libraries we are using in this project.

3. Goals and Objectives

With this study, we hope to identify the characteristics that are most useful in forecasting whether a cancer will be malignant or benign as well as discover broad trends that could help us choose the right model and hyperparameters. Whether the breast cancer is benign or aggressive is the main objective.

The goal is to utilize the machine learning algorithms and classification techniques to fit a data that can be used for forecast the discrete class of new data to analyze the dataset.

4. Motivation

Finding the relationships between variables in health-related data through exploratory data analysis (EDA) and data visualization aids in identifying risk factors and other crucial details needed to create preventative, screening, and treatment programs. A longer average pack per year of smoking and HPV infection were associated with more occurrences of breast cancer. Use of IUDs for contraception may be advantageous and may offer further protection against breast cancer. The risk of breast cancer is most strongly impacted by HPV infection.

5. Significance

About 264,000 women and 2,400 men in the United States receive a diagnosis of breast cancer each year. In the United States, breast cancer claims the lives of over 42,000 women and 500 men annually. In comparison to White women, Black women experience a greater mortality from breast cancer. With the help of this project, we can predict the cancer in initial stages and mitigate the issue in the initial days. Clinicians, scientists, and other healthcare experts collaborated to develop this new gap analysis, which is a broadened, evidence-based follow-up. The goal is to guarantee that the breast cancer research road map continues to be a reliable, authoritative resource for identifying current requirements. It expands on the preceding gap analysis by quickly summarizing the state of important areas, evaluating critically any outstanding

problems and any fresh difficulties brought on by recent research, and offering suggestions for practical implementation techniques.

Please note that we have submitted Introduction, Goals and Objectives, Motivation, significance, Objectives, and features are submitted in previous submission. In order to remove the plagiarism, we are not adding the same here.

1. Related work (Background):

Machine learning techniques have seen tremendous progress, which has sparked a lot of interest in using them to solve medical imaging issues. Here, we develop a machine learning algorithm that can correctly identify breast cancer on screening mammograms data set using an "end-to-end" training technique that effectively utilizes training datasets with either full clinical annotation or only the cancer status of the data available.

[1] Siyabend Turgut et al., “Microarray Breast Cancer Data Classification Using Machine Learning Methods” [IEEE 2018]

In the study, patient classification using machine learning techniques is done using data from microarrays of breast cancer. In the first example, the dataset is subjected to the application of eight different machine learning algorithms, and the classification outcomes are recorded. Then, in the second instance, the microarray breast cancer dataset was subjected to two distinct feature selection methods, such as Recursive Feature Elimination (RFE) and Randomized Logistic Regression (RLR), and 50 features were selected as the stop criterion.

2. Dataset

In this project, we are using Breast Cancer Wisconsin (Diagnostic) Data Set. This is an open-source dataset can be downloaded from Kaggle. Coming to the dataset, the data present in the csv file is from a digital image of a fine needle aspirate (FNA) of a breast mass, features are calculated. They characterize the traits of the visible cell nuclei in the picture. a classification technique that builds several machine learning algorithms using linear programming. An exhaustive search in the domain of 1-4 features and 1-3 separation planes was used to select relevant features. Information on 569 women across 32 different qualities can be found in the dataset. The first group of variables is Mean (3–13), followed by Stranded Error (1–23) and Worst (23–32), each of which has ten parameters. Mean, Standard Error, and Worst all refer to the average of all the cells, respectively. Every instance contains a parameter of cancerous and non-cancerous cells, and we can forecast cancer simply by inputting features. The feature values are presented in a numeric format. The term "Target" refers to the patient who is suffering from either "Benign" or "Malignant" cancer. Malignant denotes the presence of cancer, while benign denotes the

absence of cancer. Features in the dataset have a wide range of units and magnitudes. Therefore, it is necessary to equalize the magnitude of all aspects.

3. Detail design of Features

In this data set we have 569 rows and 32 columns. Below are some of the features that will be used for the working model. There are total of benign 357 and malignant 212 values.

There are some real value features that can be calculated from each cell nuclei

- Radius which is the mean of distances from center to points on the perimeter
- Texture which is the standard deviation of gray-scale values.
- Perimeter which is the perimeter of the cell
- Area is the total area of the contour
- Smoothness which is the local variation in radius lengths
- Compactness can be calculated using ((perimeter)^2/area -1)
- Concavity is the severity of concave portions of the contour
- Concave points are the number of concave portions of the contour
- Symmetry is defined as the symmetry as the contour
- Fractal dimension: A fractal dimension is a ratio that compares how a pattern's level of detail alters depending on the size at which it is measured. This can be calculated by coastline approximation – 1.

4. Analysis

A) Data Collection:

(I) Importing the dataset

We have imported the dataset with the help of pandas. The code is shown below for importing the dataset using pandas.

```
import numpy as np
import pandas as pd
cancer_data = pd.read_csv("BreastCancer.csv")
```

After successfully importing the dataset we will check for the shape of the dataset which can be done using the Cancer_data.shape. To print the first 5 and last five cells of the data frame we are using the commands cancer_data.head() and cancer_data.tail().

Displaying the first five rows of data frame.

```
cancer_data.head() #printing the first five cells
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactnes
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0	0.1622	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0	0.1238	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0	0.1444	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7	0.2098	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0	0.1374	

5 rows × 33 columns

Displaying the last five rows of dataset

```
cancer_data.tail() #printing the last 5 cells
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactnes
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.40	166.10	2027.0	0.14100	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.25	155.00	1731.0	0.11660	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.12	126.70	1124.0	0.11390	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.42	184.60	1821.0	0.16500	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37	59.16	268.6	0.08996	

5 rows × 33 columns

Once the dataset is loaded into the data frame we can proceed with the EDA of the dataset.

B) EXPLORATORY DATA ANALYSIS:

I) Basic information about the data:

For any dataset we need to know about the basic structure of the data before working on the dataset. The cancer_data.info() will give us the required information about the dataset.

```
cancer_data.info() #getting the info of the dataset
```

```
<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
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_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
```

The cancer_data.describe() will give us the statistic view of the dataset with all the information like mean, count, std.

```
cancer_data.describe()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perimeter_worst	area_worst	smoo
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000000	569.000000	569.000000	
mean	3.037183e+07	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	...	16.269190	25.677223	107.261213	880.583128	
std	1.250206e+08	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	...	4.833242	6.146258	33.602542	569.356993	
min	8.670000e+03	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	...	7.930000	12.020000	50.410000	185.200000	
25%	8.692180e+05	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	...	13.010000	21.080000	84.110000	515.300000	
50%	9.060240e+05	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	...	14.970000	25.410000	97.660000	686.500000	
75%	8.813129e+06	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	...	18.790000	29.720000	125.400000	1084.000000	
max	9.113205e+08	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	...	36.040000	49.540000	251.200000	4254.000000	

8 rows × 32 columns

II) Finding the Null values:

This is an important step during the EDA of data. Ensuring the quality of data is important. Lets see how to find the null values.

```
cancer_data.isna().sum()  #checking the sum of null values
```

```
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
```

From the above image we can see that a column named unnamed : 32 has 569 null values so we need to drop the entire column. This can be done using the data.dropna.

```
cancer_data = cancer_data.dropna(axis='columns')
```

III) Checking for duplicated values:

The df.duplicate can be used. The sum of any duplicate values is calculated using the ancer_data.diagnosis.unique() function. If duplicate values are present in the data, it will display how many of them are there.

```
diagnosis_unique_data = cancer_data.diagnosis.unique() #checking for unique values
```

IV) Knowing the data types:

It is essential to know about the data types of each and every column of a dataset. This can be done using cancer_data.dtypes

```
cancer_data.dtypes #checking data type of each column
```

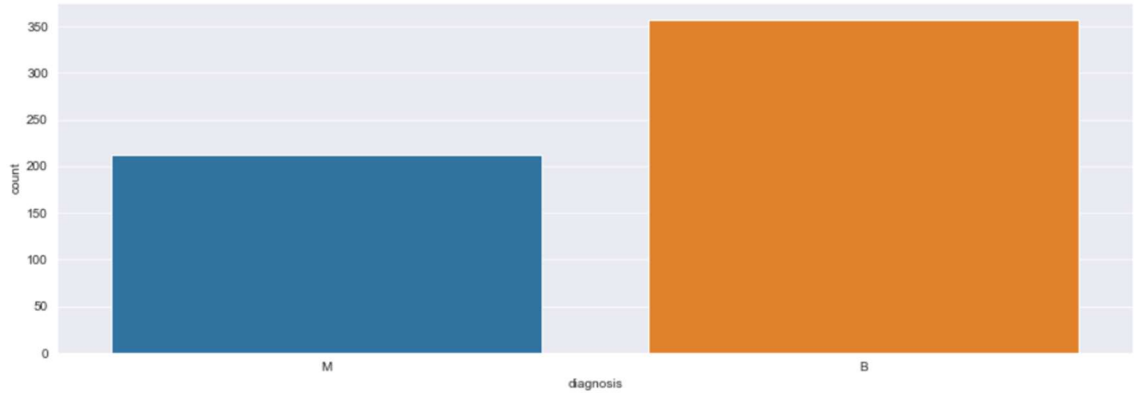
```
id                          int64
diagnosis                  object
radius_mean               float64
texture_mean              float64
perimeter_mean            float64
area_mean                 float64
smoothness_mean           float64
compactness_mean          float64
concavity_mean            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
```

C) Data visualization:

I) Bar graph:

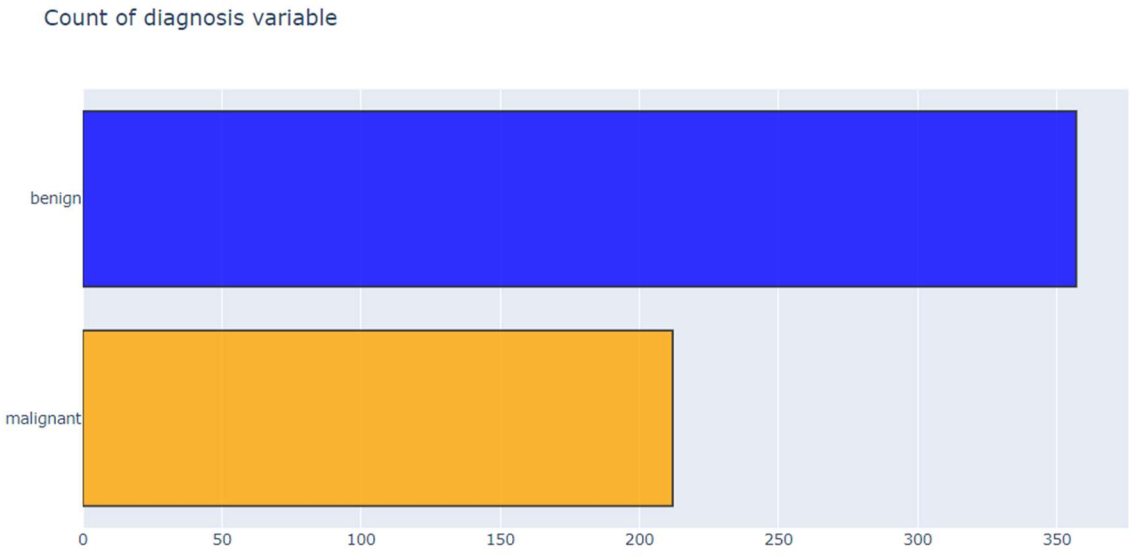
In this bar graph we have plotted the graph on number of counts of ‘Malignant’ and ‘Benign’ nodes. These different types can be seen on the X-axis and count on the Y-axis. We have drawn using the graphs using matplotlib and plotly.

```
plt.figure(figsize=(15, 5)) #size of the plot
sns.countplot('diagnosis', data=cancer_data);
```



Histogram

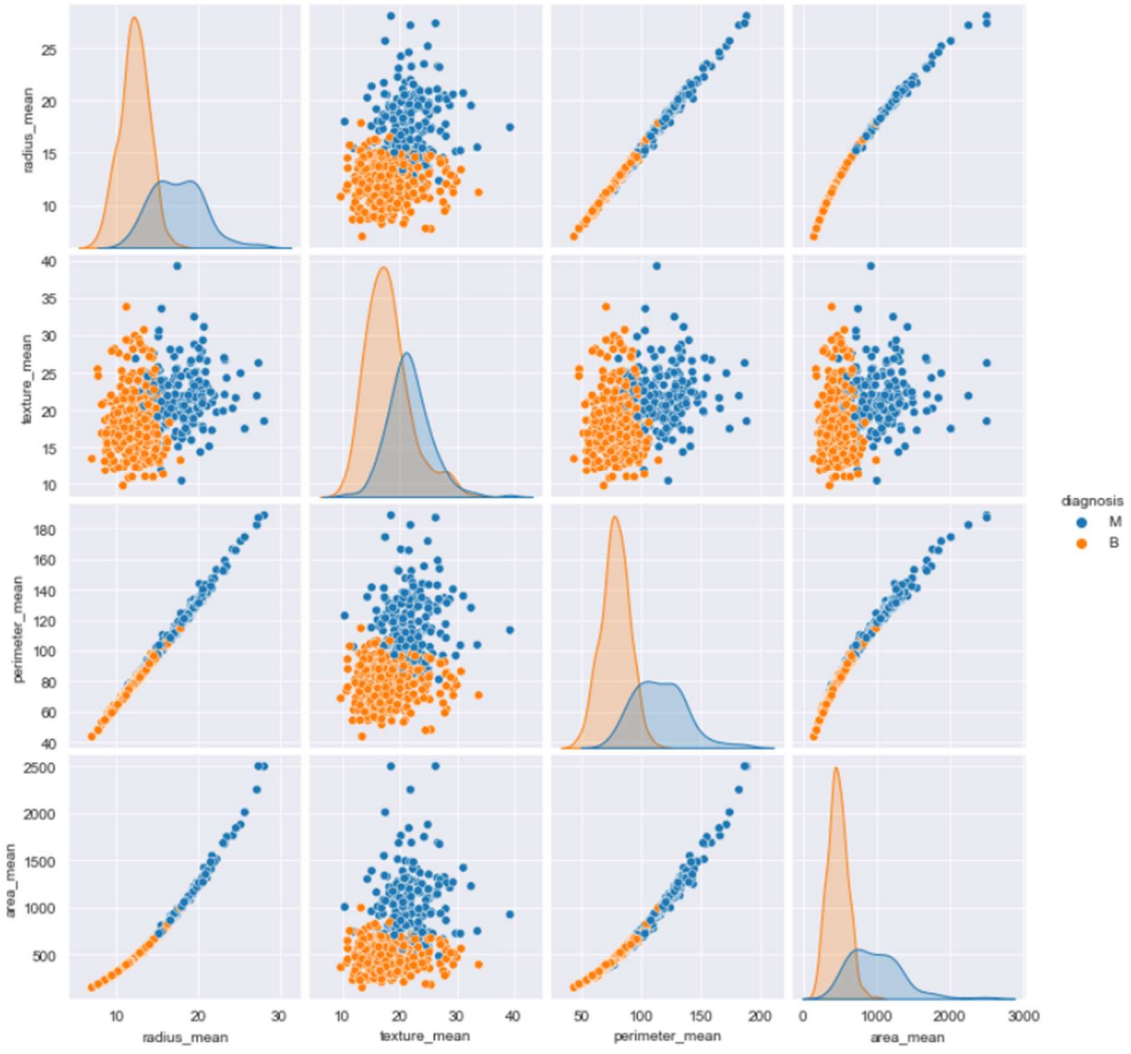
II) Bar graph Using plotly



II) Pair Plot:

For plotting the pairplot we have selected few columns. These columns are stored in the list. Sns.pairplot helps us to plot the pair plot. This pair plot has been drawn between 'Malignant' and 'Bengin'.

```
columns = ["diagnosis", "radius_mean", "texture_mean", "perimeter_mean", "area_mean"]
sns.pairplot(cancer_data[columns], hue="diagnosis") #plotting pairplot
plt.show()
```

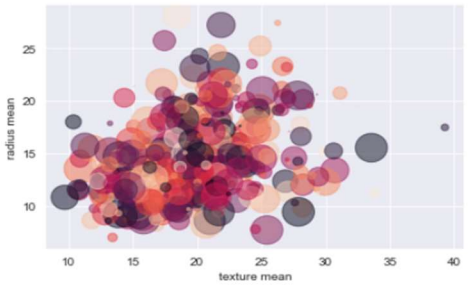


III) Scatter plot

Scatter plot has been drawn between the texture_mean and radius_mean. The radius_mean is plotted on the y-axis and texture_mean is plotted on the X-axis. In the below scatter plot to increase the size of the each plot. Each plot will be shown with the area. With the help of plt.scatter we can plot the graph.

Scatter Plot

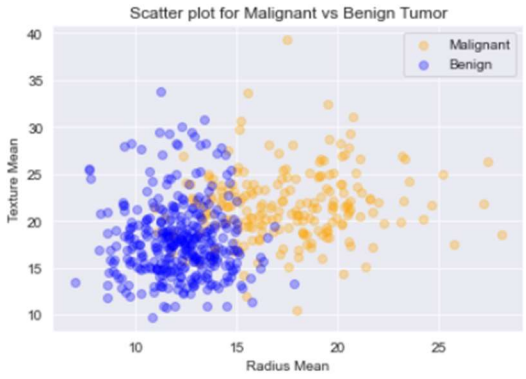
```
radius = len(cancer_data['texture_mean'])

area = np.pi * (15 * np.random.rand( radius ))**2
colors = np.random.rand( radius )

plt.xlabel("texture mean")
plt.ylabel("radius mean")
plt.scatter(cancer_data['texture_mean'], cancer_data['radius_mean'], s=area, c=colors, alpha=0.5); #plotting a scatter plot
```



From the below scatter plot we have directly drawn the graph by plotting the radius_mean and texture_mean these are plotted on the X-axis and y-axis respectively. Below figure shows how the scatter plot has been plotted.

ScatterPlot

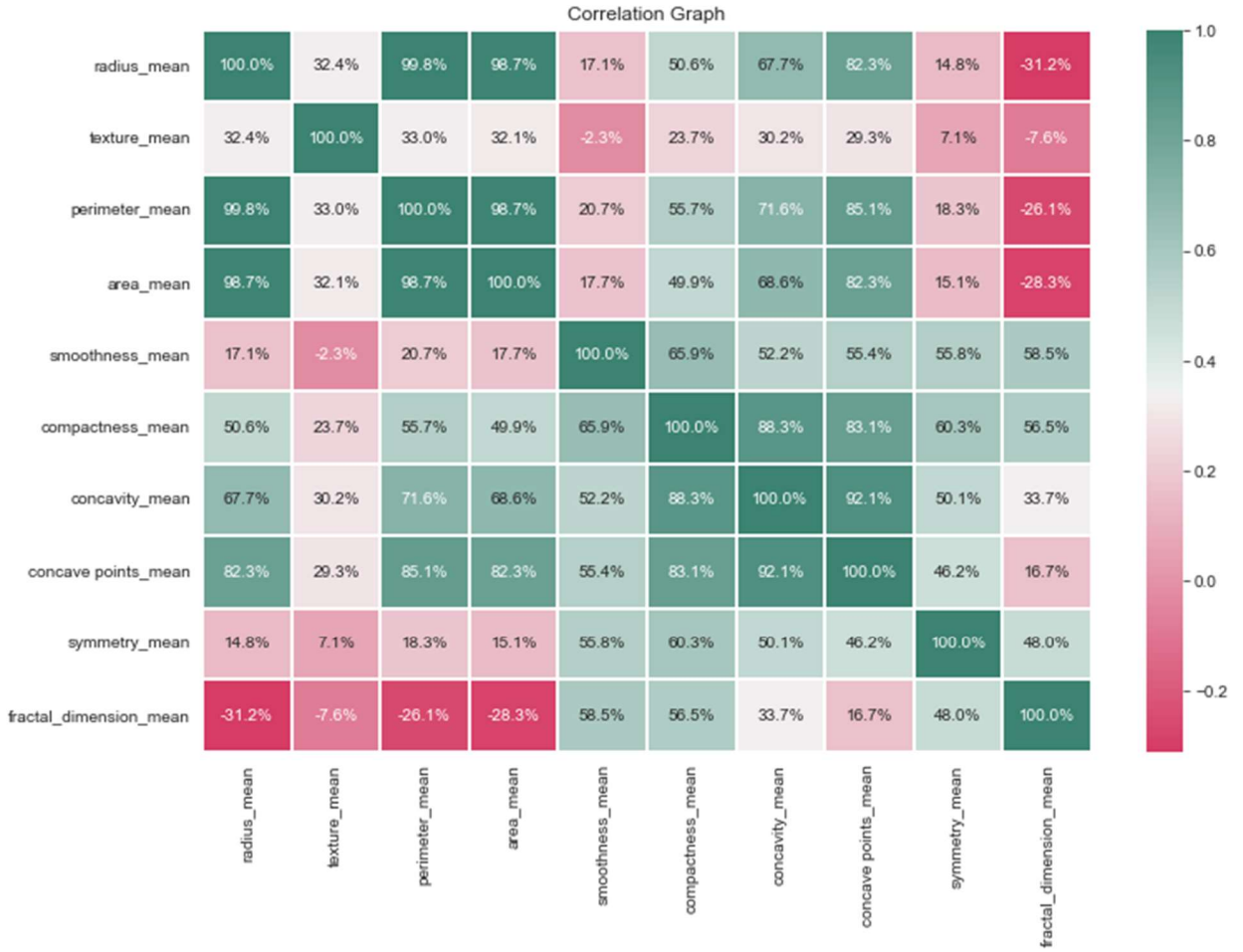
```
Malignant = cancer_data[cancer_data.diagnosis == "M"]
Benign_Tumor = cancer_data[cancer_data.diagnosis == "B"]

plt.title("Scatter plot for Malignant vs Benign Tumor")
plt.xlabel("Radius Mean")
plt.ylabel("Texture Mean")
plt.scatter(Malignant.radius_mean, Malignant.texture_mean, color = "Orange", label = "Malignant", alpha = 0.3)
plt.scatter(Benign_Tumor.radius_mean, Benign_Tumor.texture_mean, color = "blue", label = "Benign", alpha = 0.3)
plt.legend()
plt.show()
```



IV) Heatmap using matplot:

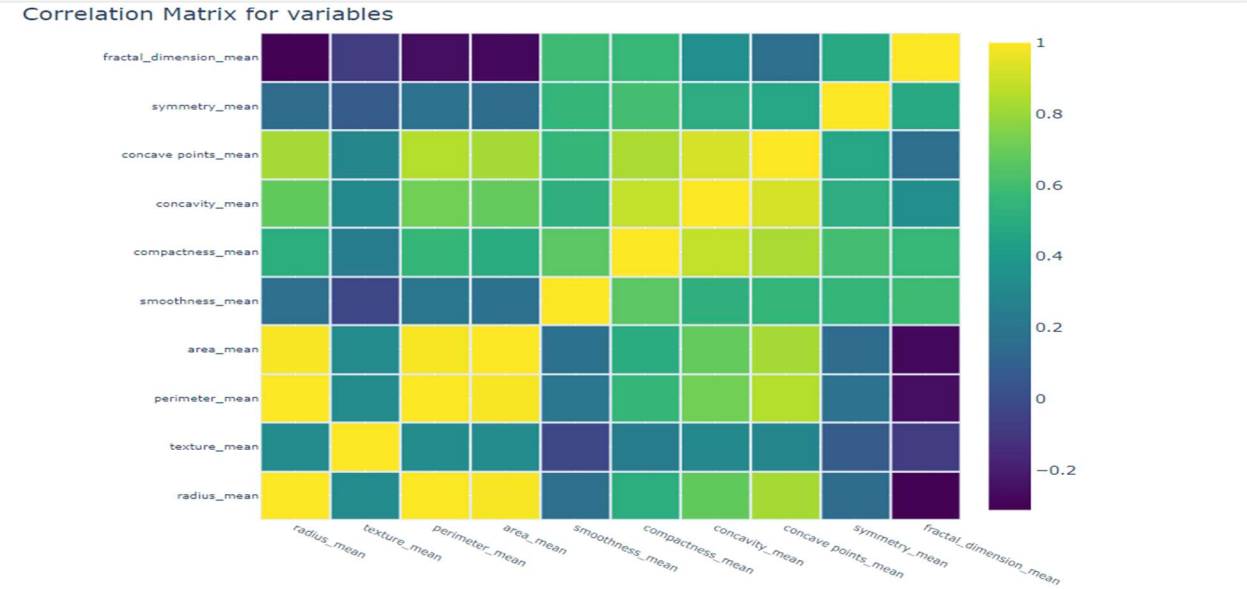
A table of the correlation coefficients between different variables is called a correlation matrix. The correlation between any two variables is displayed in each table cell. Data are summarized,

input into more complex studies, and diagnostics for complex analyses are all done using correlation matrices. We have drawn the correlation matrix. Below is the correlation graph



V) Heat map using plotly

Below is the Heap map drawn using the plotly. Plotly is an interactive graph by hovering the values on the cell we can get the value of the particular cell. This can help to look the graph more efficiently.



massive matrix with a large number of numbers. These integers fall between -1 and 1.

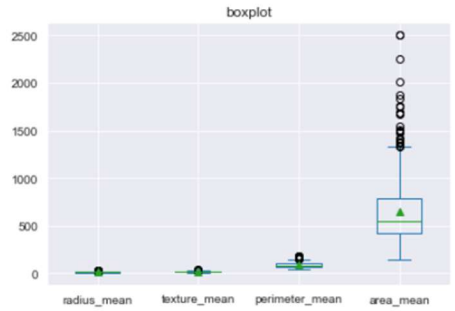
Meaning of 1: Area mean and radius mean are two variables that have a positive correlation with one another. The definition of zero is the absence of any relationship between fractal dimension se and parameters like radius mean. The definition of -1 is that two variables, such as the mean of the radius and the mean of the fractal dimension, are negatively connected. The actual correlation between them is -0.3 instead of 1, but the idea is that if the sign of the correlation is negative, there is a negative correlation.

VI) Box Plot

Box plots are basically used to summarize the data values with characteristics like minimum, first quartile, median, third quartile, and maximum. The first quartile to the third quartile are taken as the boundaries of the box plot. We have taken 5 columns in the box plot one for each diagnosis, radius_mean, area_mean, texture_mean, perimeter_mean. Then these are passed through the DataFrame, now we are plotting the graph foir Box plot where kind= ‘Box’.

BoxPlot

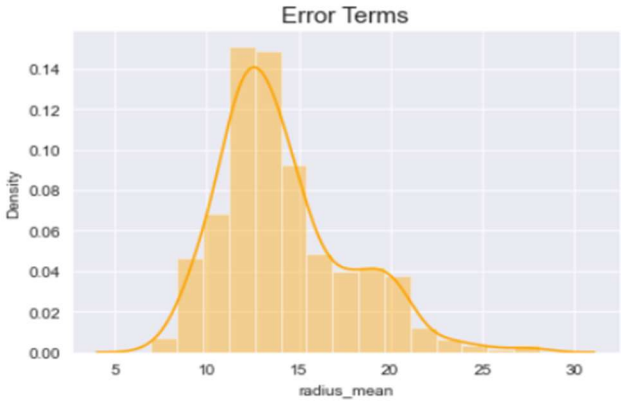
```
col1 = cancer_data['diagnosis']
col2 = cancer_data['radius_mean']
col3 = cancer_data['texture_mean']
col4 = cancer_data['perimeter_mean']
col5 = cancer_data['area_mean']

DF = pd.DataFrame({'diagnosis': col1, 'radius_mean': col2,'texture_mean': col3, 'perimeter_mean': col4, 'area_mean': col5})
#plotting Box plot
ax = DF[['diagnosis', 'radius_mean','texture_mean', 'perimeter_mean','area_mean']].plot(kind='box', title='boxplot', showmeans=True)
plt.show()
```



VII) Distplot

A distribution plot, often known as a distplot, shows how the data distribution varies. The total distribution of continuous data variables is depicted by a Seaborn Distplot. The distplot with several modifications is shown using the Seaborn and Matplotlib modules. This Distplot has been drawn using sns.distplot for the radius_mean of cancer data and we have divided the data into 15 bins and the color is chosen as 'Orange'. Below is the output for Distplot with code.

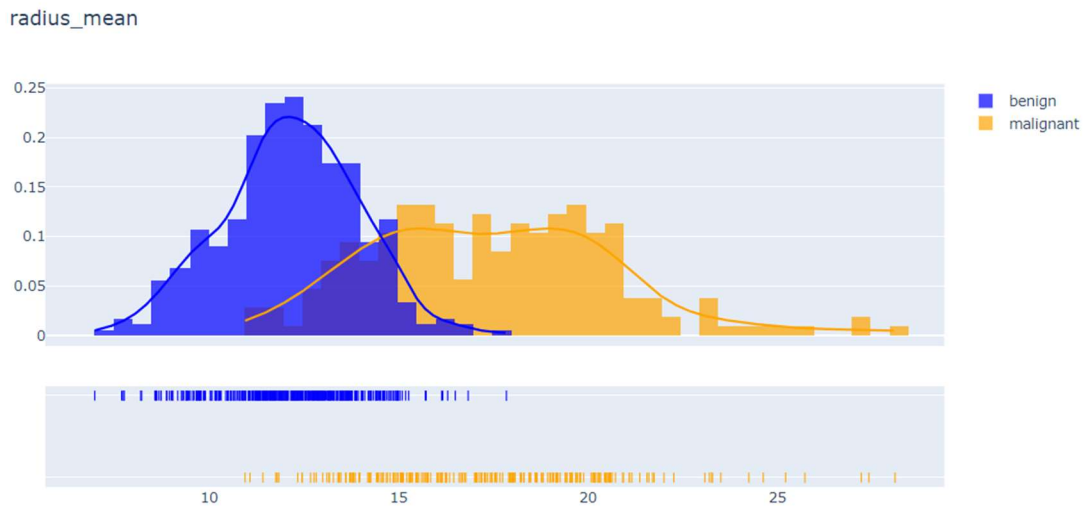
```
fig = plt.figure()
sns.distplot(cancer_data['radius_mean'], bins = 15, color='Orange') #plotting displot
plt.title('Error Terms', fontsize = 15)
plt.show()
```



Distplot using plotly:

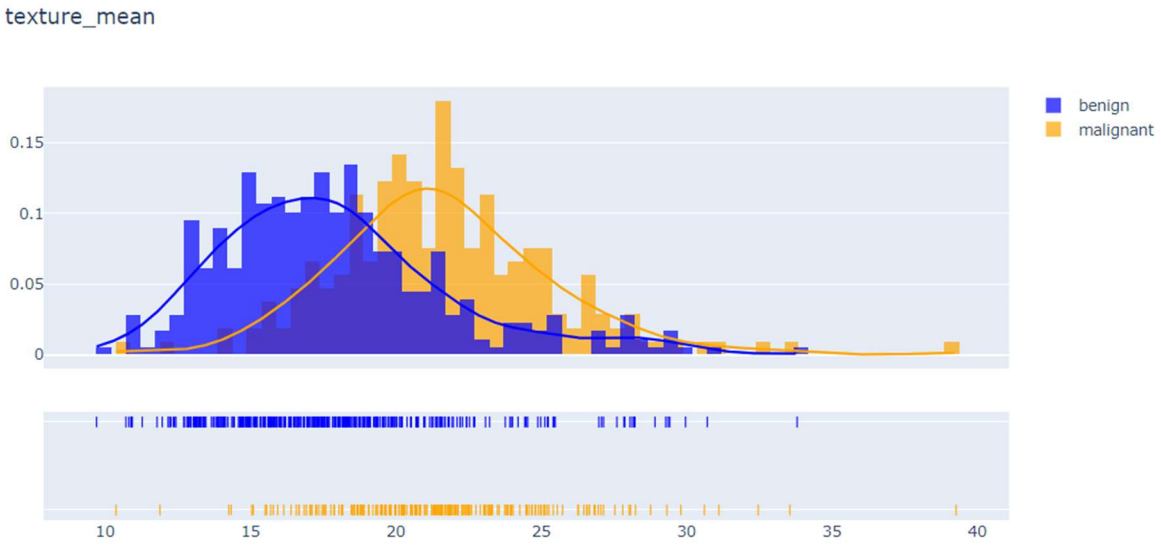
We have used the radius_mean for the distribution of plot. The code in python for the plot.

```
#plot distribution 'mean'
plot_distribution('radius_mean', .5)
```



We have used the Texture_mean for the distribution of plot. The code in python for the plot.

```
plot_distribution('texture_mean', .5)
```

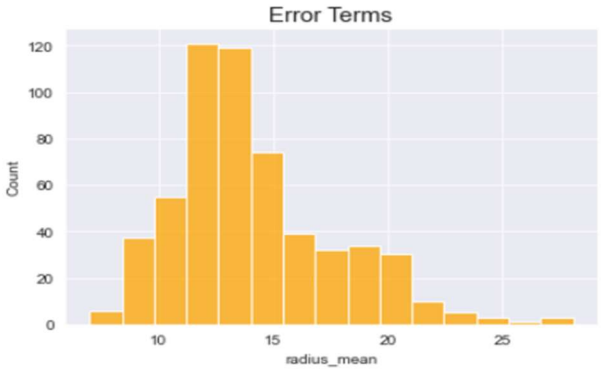


VIII) Histogram:

While a bar graph can be used to compare two entities, a histogram plot can be used when the data is still scattered. Despite having similar appearances, histograms and bar plots are employed

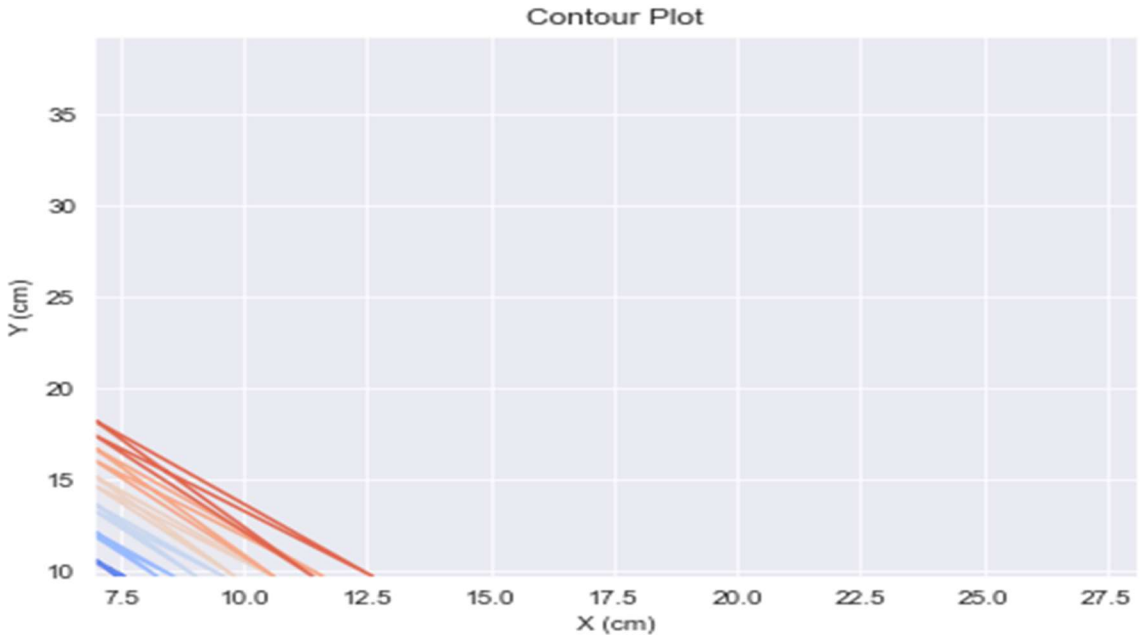
in various contexts. This is represented by Matplotlib's hist() function. Histplot can be drawn using sns.histplot(). Here we are drawing histplot for ‘radius_mean’ so it will be the data. The bins we are choosing are 15 and the color is orange.

```
fig = plt.figure()
sns.histplot(cancer_data['radius_mean'], bins = 15, color='Orange') #plotting histogram
plt.title('Error Terms', fontsize = 15)
plt.show()
```



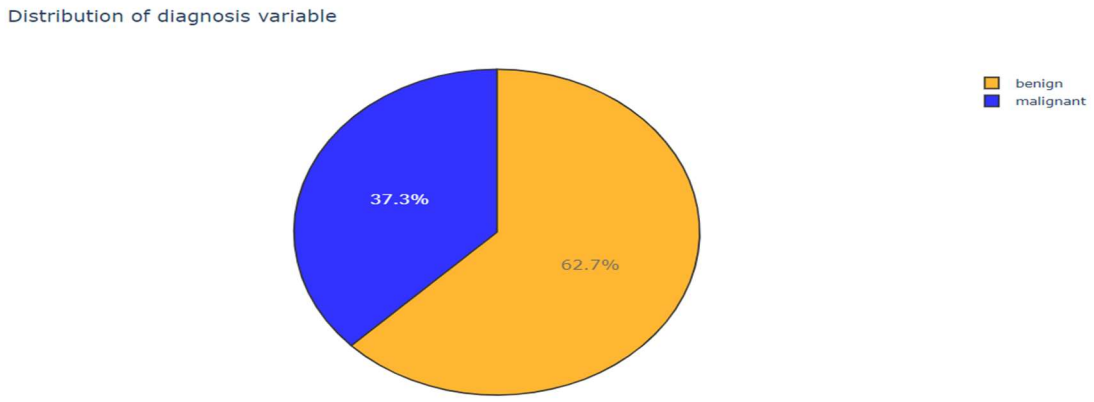
IX) Contour plot

Using constant z slices, often known as contours, a contour plot is a graphicl method for expressing a 3d surface in a two-dimensionl format. To put it another way, lines are drawn linking the (x,y) coordinates where a certain value of z appears.



X) Pie chart:

A circular graph known as a "pie plot" shows data as slices, components, or portions of a pie. Each pie slice represents a different item or category of data, and data analysts utilize them to illustrate the percentage or proportionate data. Pie () is used in Matplotlib to visualize it. In the below example we have drawn chart for the benign and malignant data of the diagnosis filed and printed the percentage of each benign and malignant data



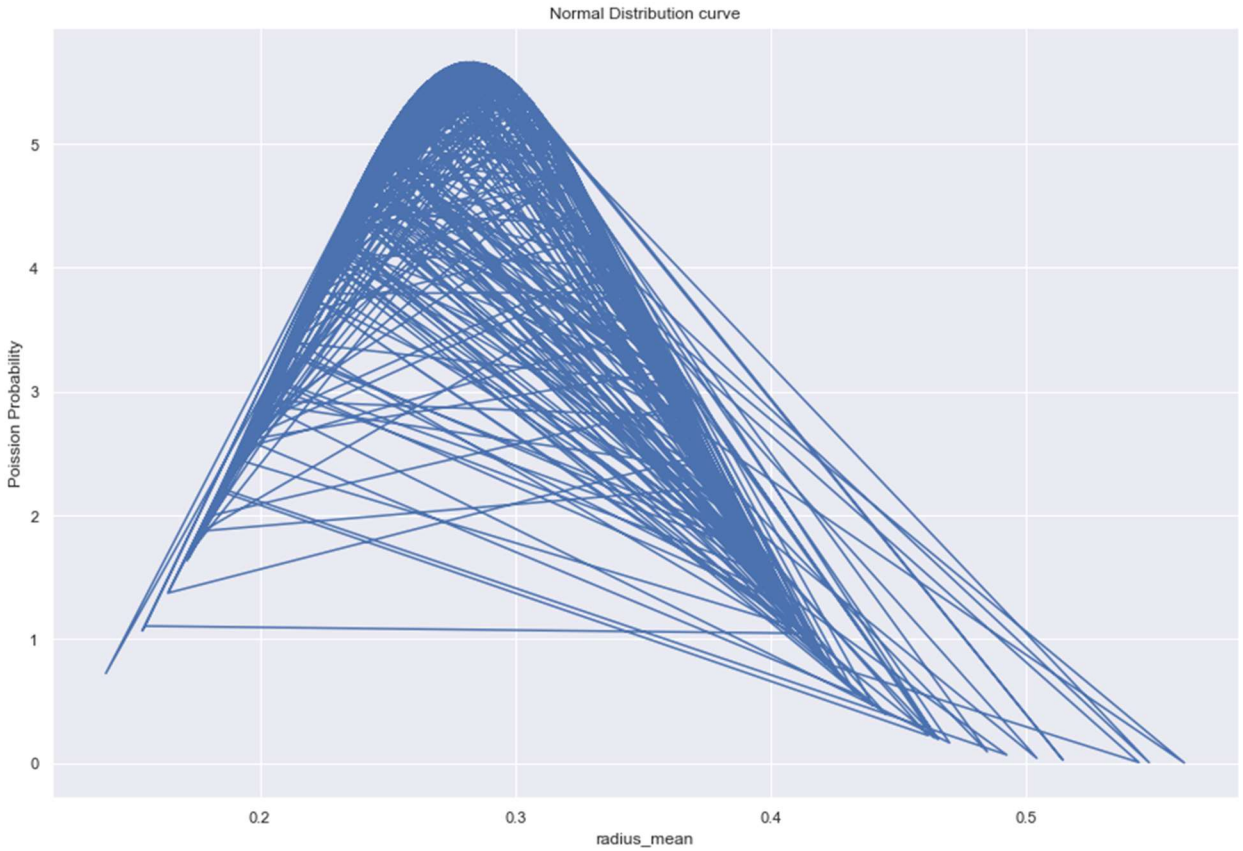
D) Probability Models:

I) Poisson Distribution:

A Poisson point process, also known as a Poisson process, is a group of points dispersed at random in mathematical space.

The Poisson process is frequently defined on a real line because of its many characteristics, where it can be viewed as a random (stochastic) process in one dimension. This further enables the creation of mathematical systems and the investigation of specific random events.

Every point in the process is stochastically independent from every other point, which is one of its key characteristics.



II) Poisson CDF (cumulative distribution function)

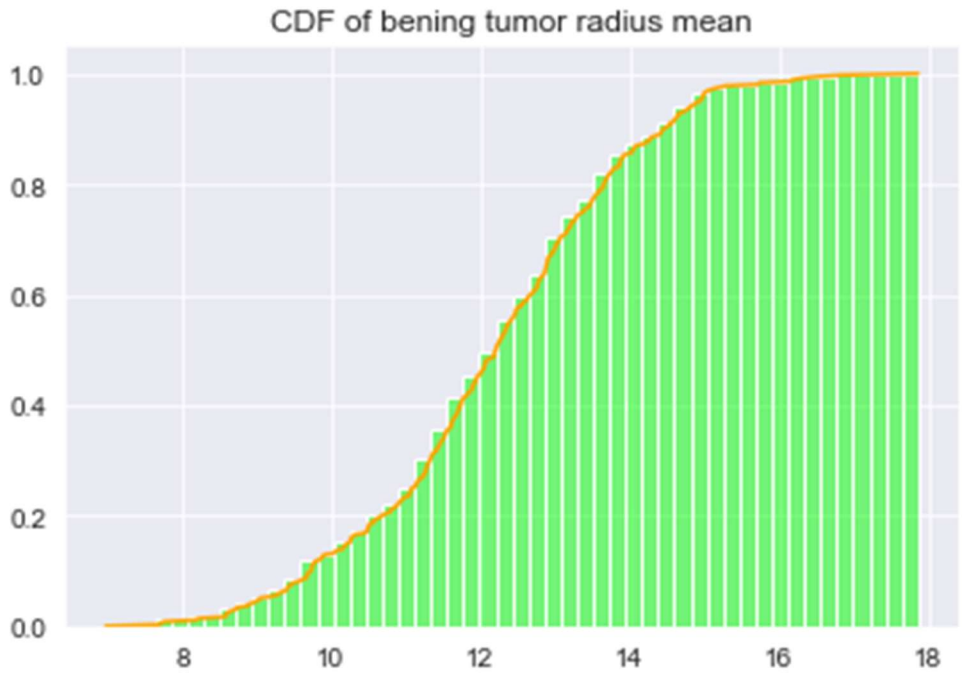
The Poisson cumulative distribution function allows you to determine the likelihood that an event will occur less frequently than or equally frequently than x times during a certain time or space interval if the event occurs on average times within that interval.

$$p = F(x|\lambda) = e^{-\lambda} \sum_{i=0}^{floor(x)} \frac{\lambda^i}{i!}.$$

The probability that a variable will have a value less than or equal to x is expressed by the cumulative distribution function. P(X <= x)

Let's clarify what P(12 <X) means in the cdf graph of the bening radiues. The response is 0.5. There is a 0.5 percent chance that the variable will have a value less than or equal to 12 (radius mean).

There are two different ways to plot a cdf.

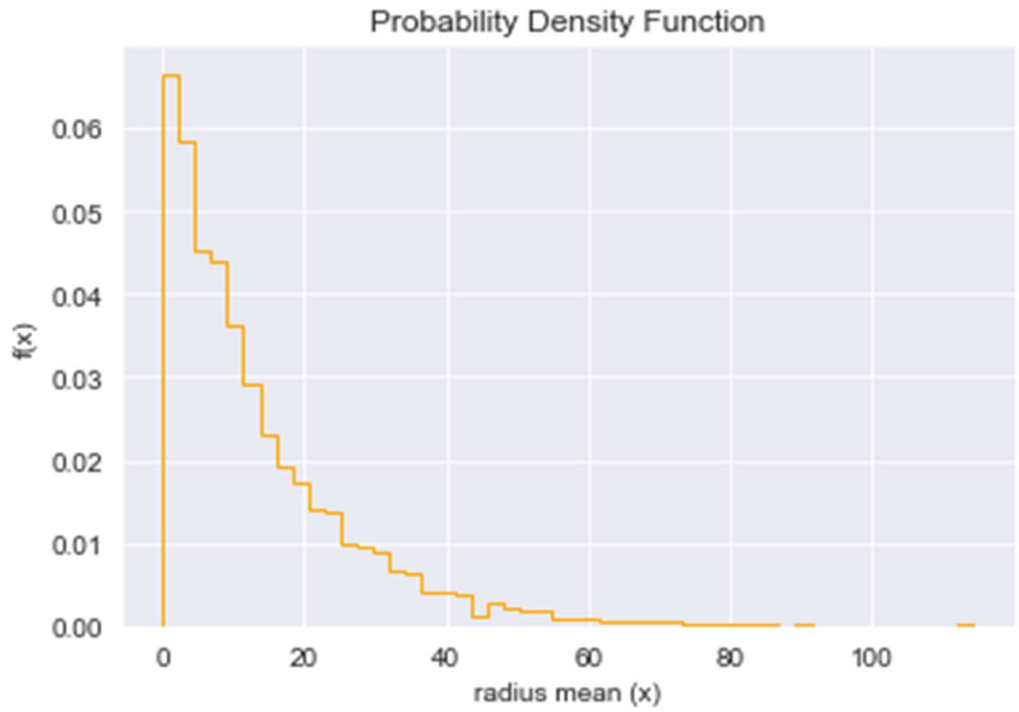


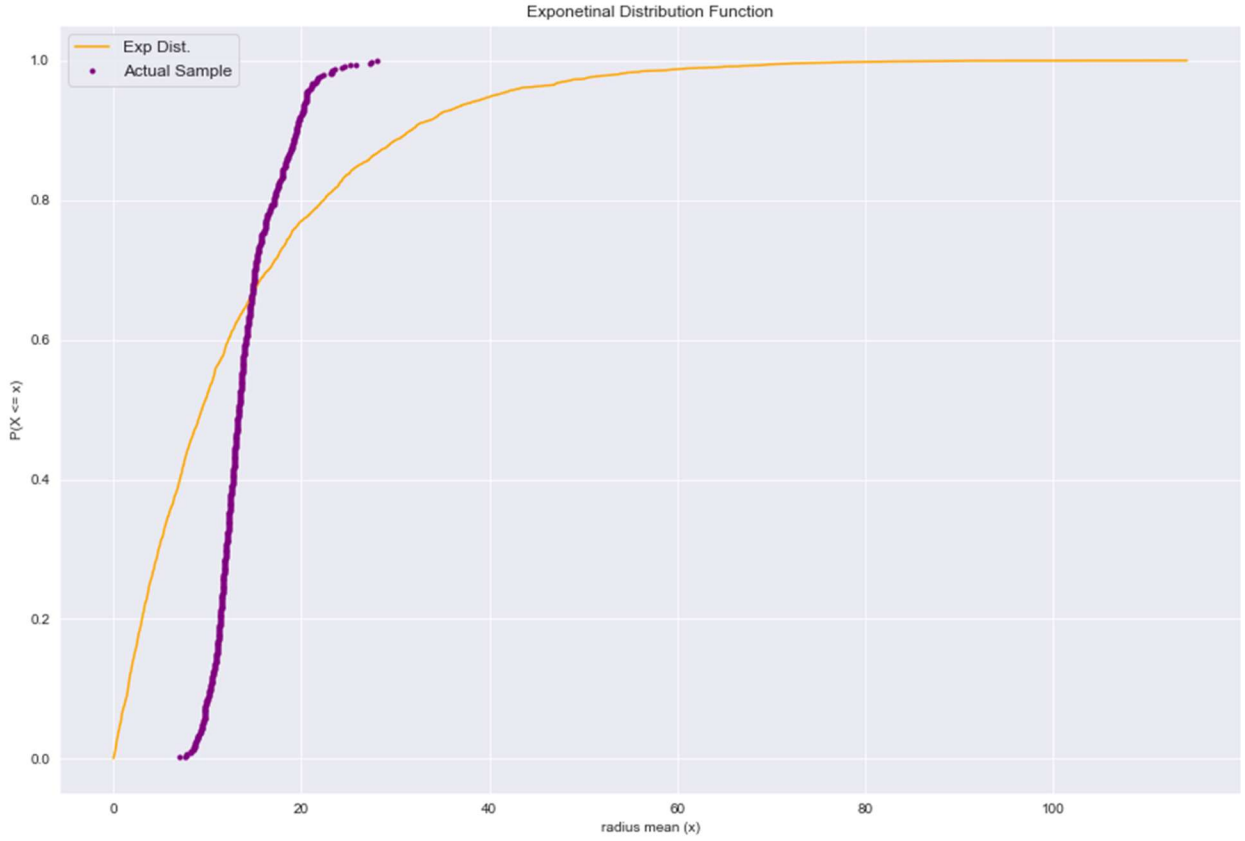
III) Exponential Distribution

The length of time we must wait until an event occurs is modeled using the exponential distribution, a probability distribution.

The cdf of a random variable X can be expressed as follows if it has an exponential distribution

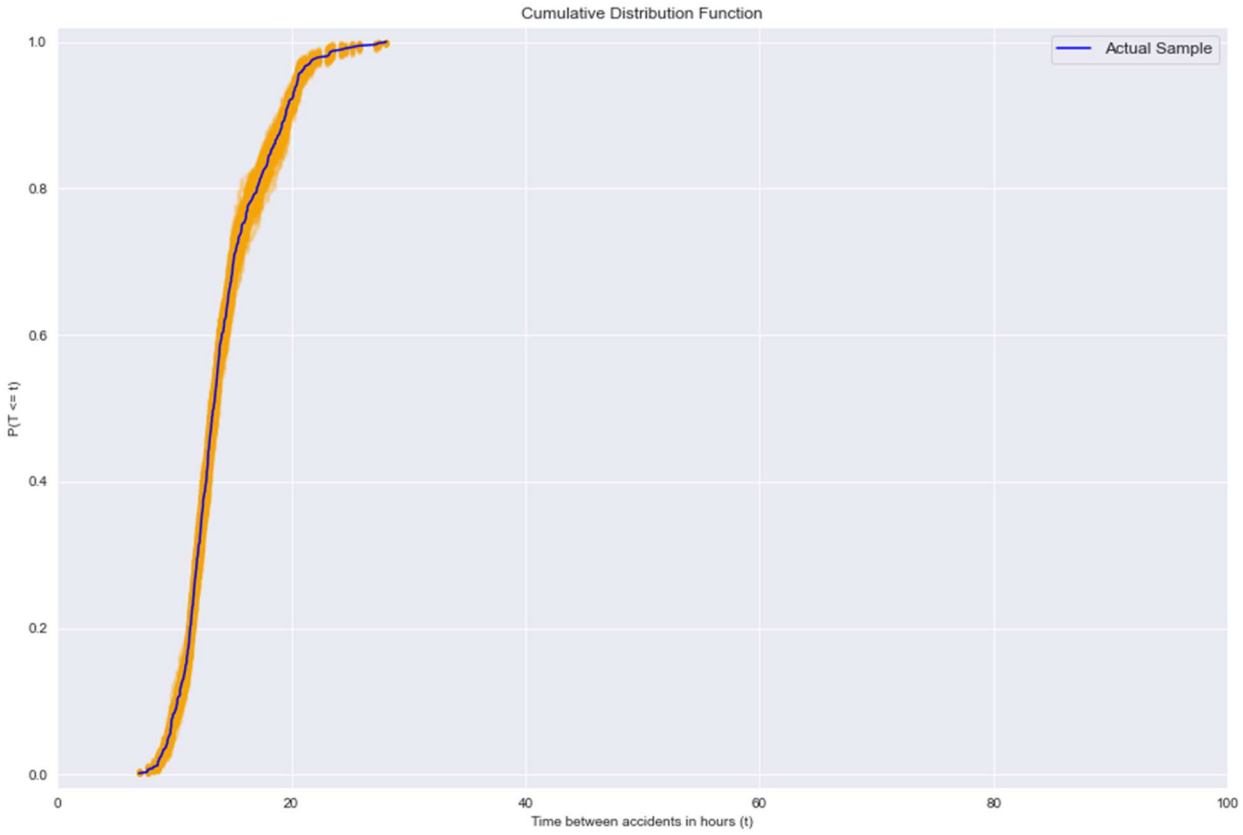
$$F(x; \lambda) = 1 - e^{-\lambda x}$$


Probability Density Function
f(x)
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
100
radius mean (x)

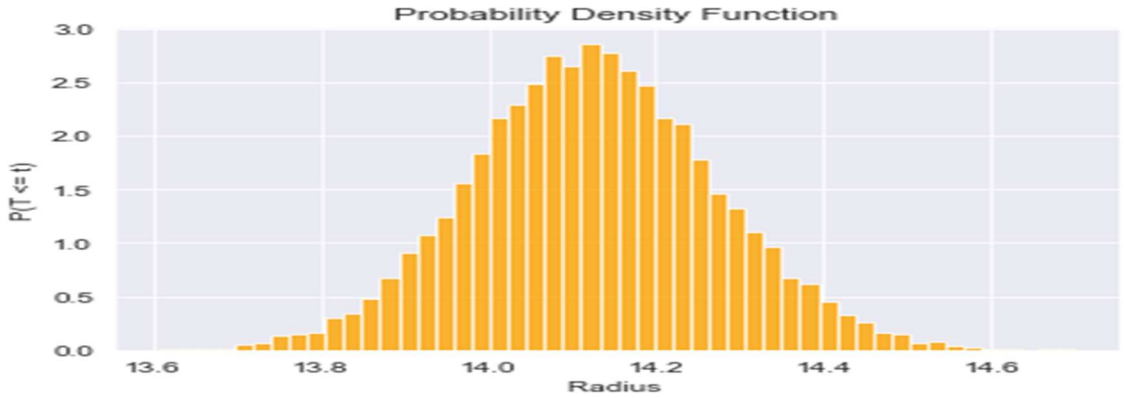

Exponetinal Distribution Function
Exp Dist.
Actual Sample
P(X <= x)
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
radius mean (x)

IV) Confidence interval

A confidence interval is a range of values that are bound by the mean of the statistic and are most likely to contain an unidentified population parameter. Confidence level is the chance, or degree of certainty, that the confidence interval would include the true population parameter when a random sample is drawn numerous times.



V) Bootstrap Replicate



E) Statistical Models:

I) One Way anova:

a one-way ANOVA is performed when comparison is needed. Each participant's score on a category X predictor variable is used to determine their participation in a group. ANOVA is an extension of the t test. In an ANOVA, the categorical predictor variable could reflect naturally occurring groups or groups that are created and subsequently subjected to various treatments.

II) Calculating f_oneway ()

One-way anova can be calculated using the f_oneway(data). We need to import the SciPy. Stats package for this f_oneway. From this we can retrieve p-value.

F_oneway

```
from scipy.stats import f_oneway

f_oneway(cancer_data['radius_mean'],cancer_data['texture_mean'])
```

```
F_onewayResult(statistic=490.4551601701196, pvalue=1.2675924841686346e-90)
```

III) Printing Anova Table

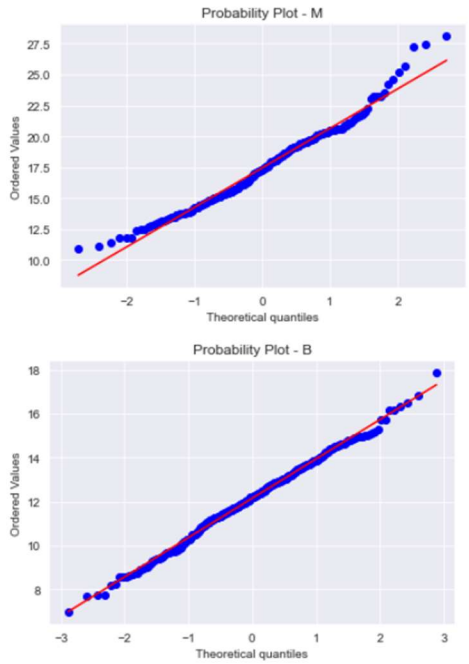
The sum of squares (SS), mean squares, and between-group and within-group sources of variance are all displayed in the ANOVA table (MS). The sum of the within- and between-group variances is the overall variation. The F value is a comparison of the mean squares between and within groups (MS). The degree of freedoms and F value are used to estimate the p value.

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	109410.76756	1	109410.76756	18829.543546	0.0	5.050677
Within Groups	3294.604835	567	5.810591			
Total	112705.372395	568	198.424951			

IV) Normality assumption check

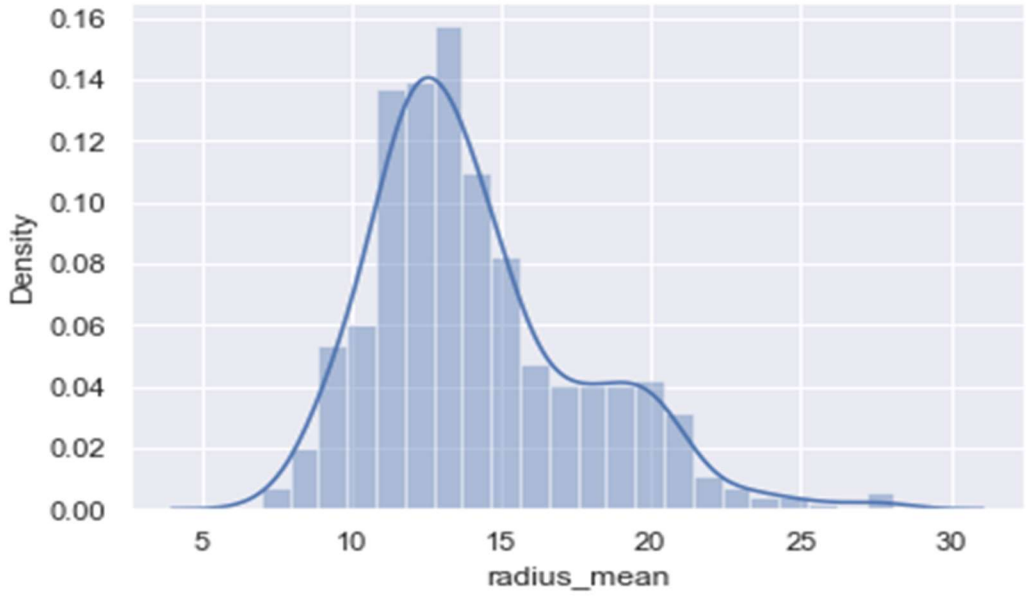
When the study is being planned, this assumption is examined. All groups are thus mutually exclusive, i.e., a person can only be a member of one group. Additionally, this implies that the data are not from repeated measures. This prerequisite is fulfilled in this instance. When a model is derived from an ANOVA or regression framework, the assumption of normality is checked on

the model's residuals. The Shapiro-Wilk test is one strategy for examining the validity of the assumption of normality.



V) Type 1 errors:

A Type I error in statistical hypothesis testing is simply the rejection of the actual null hypothesis. Type I errors are frequently referred to as false positive errors. In other words, it makes a misleading assumption about the existence of a nonexistent phenomenon. We should always know that a type I error does not indicate that we accept the alternative hypothesis of an experiment in error.



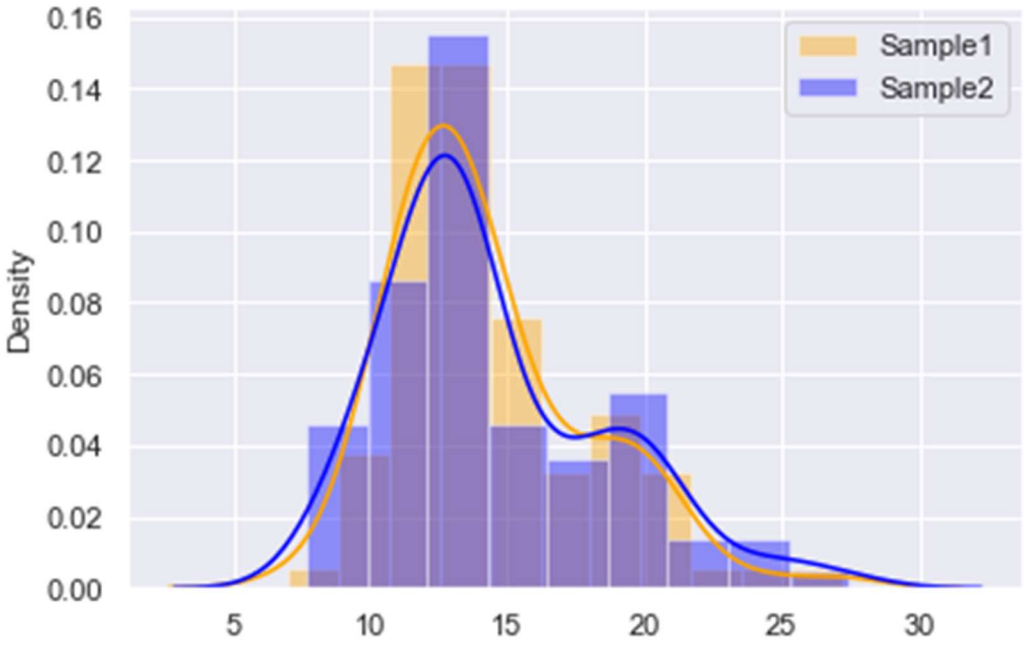
VI) Summary of both sample 1 and sample2:

The Summary of the dataset can be easily calculated in python. This can be done by calling stats. Describe (data sample). Below is the image for code in python.

```
Sample 1 Summary

k = 100
sample1 = np.random.choice(pop,100,replace=True)

print ("Sample 1 Summary")
stats.describe(sample1)

Sample 1 Summary
DescribeResult(nobs=100, minmax=(6.981, 27.22), mean=14.262279999999999, variance=13.17138598141414, skewness=0.9751117048030952, kurtosis=0.8729699629107515)

Sample 2 Summary

sample2 = np.random.choice(pop,100,replace=True)
print ("Sample 2 Summary")
stats.describe(sample2)
# test the sample means

Sample 2 Summary
DescribeResult(nobs=100, minmax=(7.691, 27.42), mean=14.47933, variance=16.469118991010102, skewness=0.936837425830828, kurtosis=0.43948561492321403)
```

VII) Distplot:

The below graph shows the density of sample 1 and sample 2.

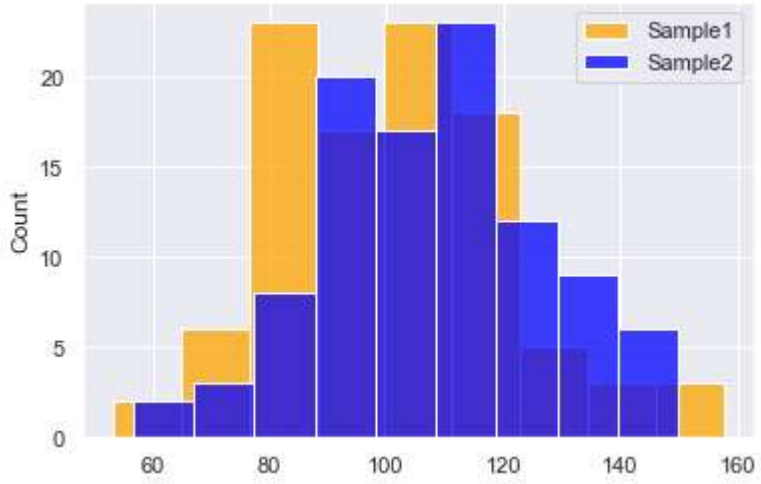


VIII) T-test on samples

IX) Type II errors :

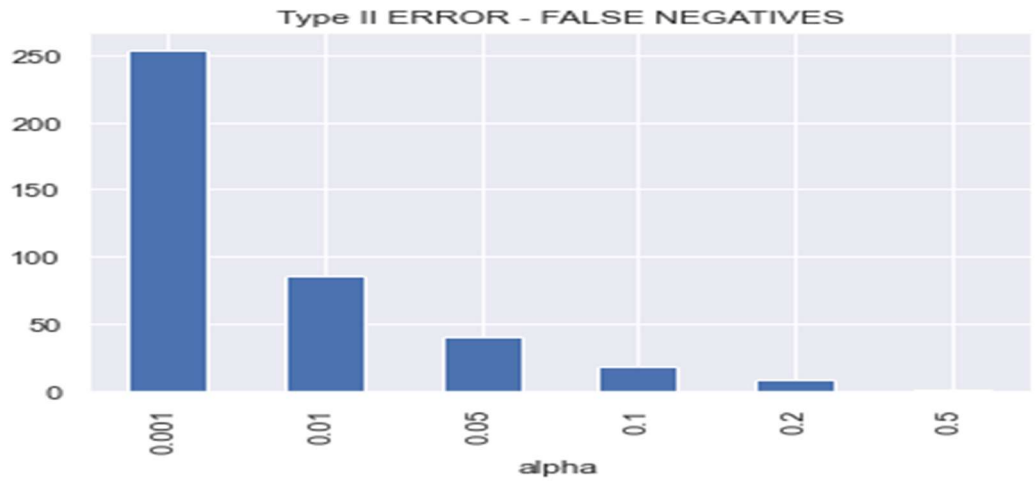
A type II error, often known as a false negative finding or conclusion, happens when a faulty null hypothesis is not rejected. A type II error has happened in statistical hypothesis testing if a faulty null hypothesis is not rejected when it ought to have been. A genuine null hypothesis is rejected while a false null hypothesis is accepted in this kind of error.

X) Displot for Type II errors:



XI) Type II ERROR - FALSE NEGATIVES

We can clearly see that value of alpha is decreases from .001 to 0.5

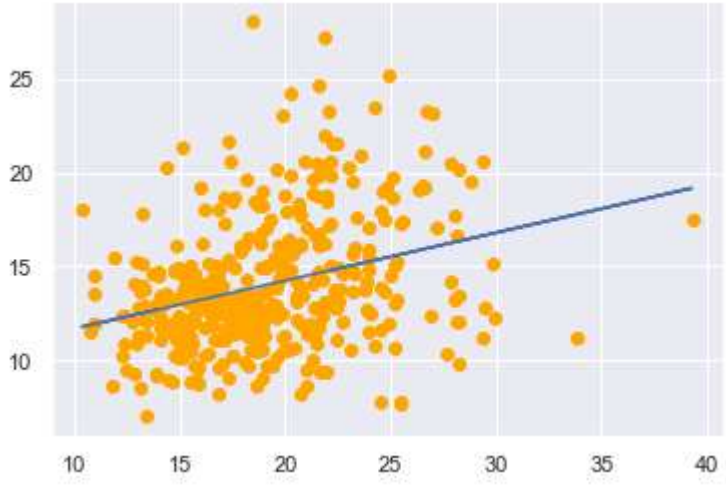


F) Implementing linear regression

Modeling the relationship between two continuous variables using simple linear regression. Predicting the value of an output variable (or response) from the value of an input variable (or predictor variable) is frequently the goal.

I) Best fit regression line

The phrase "line of best fit" describes a line that best depicts the relationship between the data points in a scatter plot. The geometric equation for the line is often determined by statisticians using the least squares approach, also known as ordinary least squares (OLS), either manually or using software.



II) Residual analysis

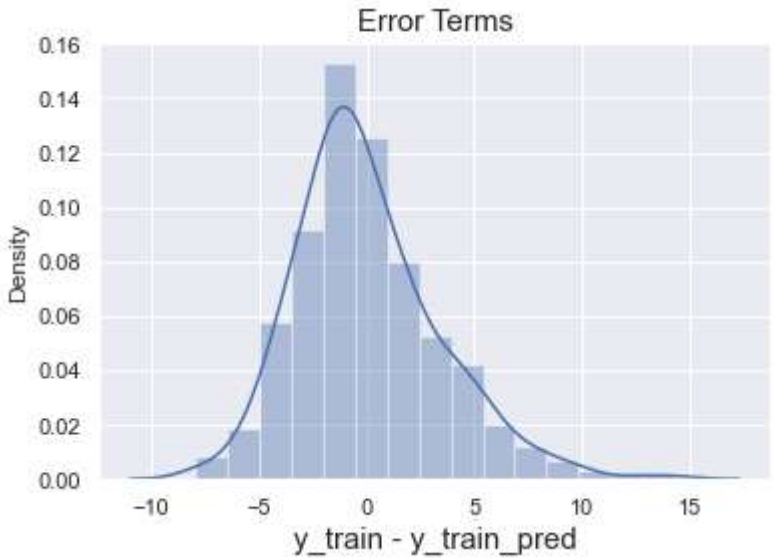
The term "residuals" refers to the discrepancies between the model's one-step predicted output and the measurd output from the validation data set. As a result, residuals are the portion of the validation data that the model is unable to account for.

Two tests make up residual analysis: the independence test and the whiteness test.

Residual analysis

```
# Predicting y_value using traingn data of X
y_train_pred = lr.predict(X_train_sm)

# Creating residuals from the y_train data and predicted y_data
res = (y_train - y_train_pred)
```

```
# Plotting the histogram using the residual values
fig = plt.figure()
sns.distplot(res, bins = 15)
plt.title('Error Terms', fontsize = 15)
plt.xlabel('y_train - y_train_pred', fontsize = 15)
plt.show()
```



III) Calculating r2- score:

Code in python for calculating r2-score

```
# Importing r2_square
from sklearn.metrics import r2_score

# Checking the R-squared value
r_squared = r2_score(y_test, y_test_pred)
r_squared
```

```
0.11955670316468836
```

We can see that value is 0.11

IV) Visualizing the line on test data :

Below graph shows the fit of test values of x_test and y_test as scatter and predicted values are shown in the form of a line.

4) Model Implementation

A) Label Encoder:

Using label encoder to decode the diagnosis parameters. The labels of 'M' and 'B' will be replaced as 0 and 1. From this we can do the desired operations of our dataset.

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

B) Train Test Splitting

It is a quick and simple technique to complete, and the outcomes let you compare how well machine learning methods work when applied to your particular predictive modeling issue. Although the process is straightforward to use and understand, there are some circumstances in which it should not be applied, such as when the dataset is tiny or when further setup is necessary, such as when it is applied for classification and the dataset is unbalanced.

Importing the header file

```
from sklearn.model_selection import train_test_split
```

C) Feature Selection

A predictive model's input variables are minimized through the process of feature selection.

This can be classified positive correlated features, uncorrelated features and negative correlated features.

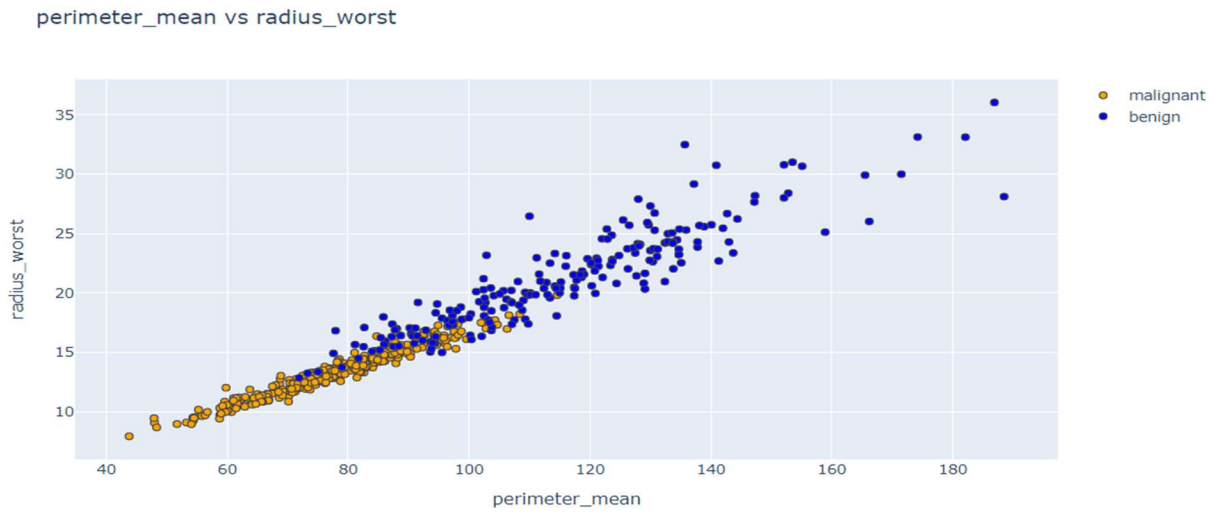
If one variable provides information about the other(s), then we can say that the two variables are related. such as cost and distance. You will pay more if you take a taxi a long distance. As a result, we can conclude that cost and distance are positively correlated.

Simplest method for determining the relationship between two variables

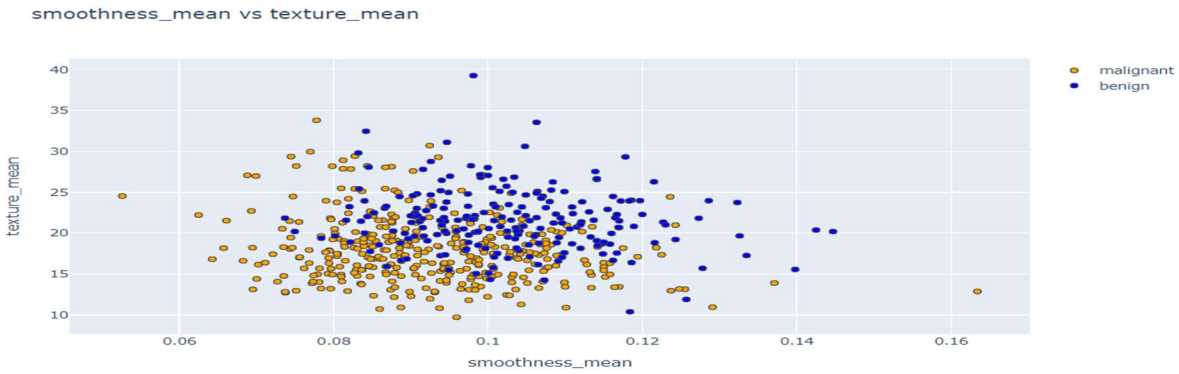
Let's examine how area mean and radius mean relate to one another.

You can see in the scatter plot that as the radius mean rises, so does the area mean. As a result, there is a positive correlation between them. Area mean and fractal dimension themselves do not correlate. Due to the fact that the chance of area mean is unaffected by changes in area mean, fractal dimension se

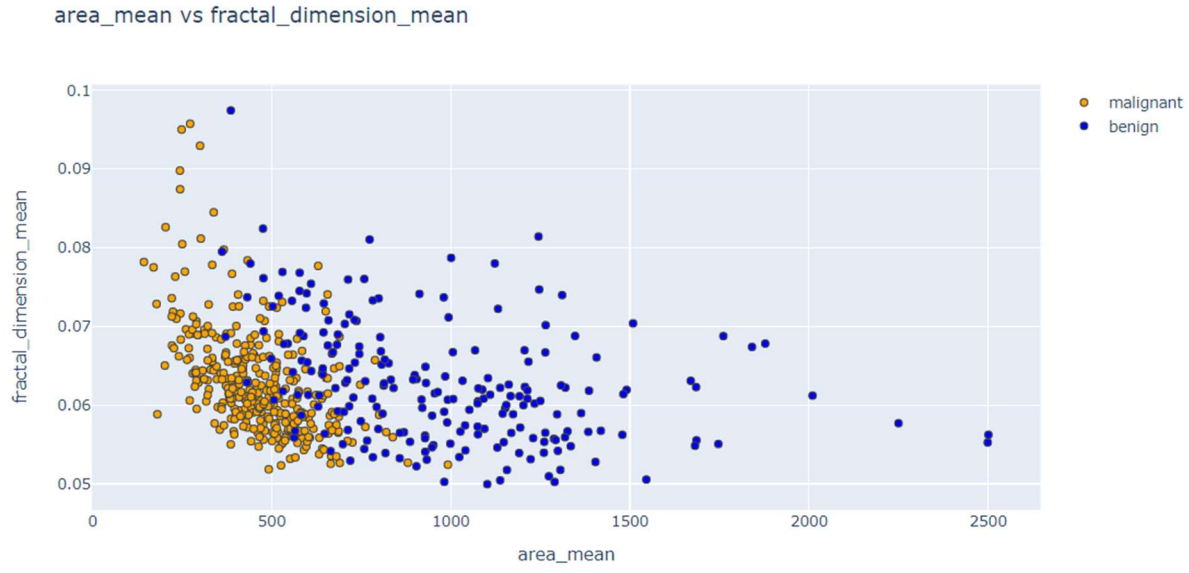
Positive correlated



Un correlated features



Negative corelated features



Feature selection is the process of deciding which current features will be most helpful in training the model. In our implementation we are taking 6 columns which are dependent on the data.

```
prediction_feature = [ "radius_mean",  'perimeter_mean', 'area_mean', 'symmetry_mean', 'compactness_mean', 'concave points_mean']

targeted_feature = 'diagnosis'

len(prediction_feature)
```

```
6
```

Split the dataset into Training Set and Testing Set by 33% and set the 15 fixed records

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=15)

print(X_train)
```

D) Perform Feature Standard Scaling

A technique for normalizing the variety of independent variables or features in data is called feature scaling. It is typically carried out during the data preprocessing step and is sometimes referred to as data normalization in the context of data processing.

By eliminating the mean and scaling to the unit variance, standardize the features.

A sample x's average score is determined as follows:

$$z = (x - u) / s$$

Code for implementation:

```
sc = StandardScaler()

X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

5) ML Model Selecting and Model Prediction

A) Model Building

In this we have created a function model building which have five arguments. From this function we get prediction, accuracy score.

Arguments that need to be passed through the function.

1. Model which is Machine Learning Model object
2. Feature Training data
3. Feature Testing data
4. Targeted Training data
5. Targeted Testing data

```
def model_building(model, X_train, X_test, y_train, y_test):
    """

    Model Fitting, Prediction And Other stuff
    return ('score', 'accuracy_score', 'predictions' )
    """

    model.fit(X_train, y_train)
    score = model.score(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(predictions, y_test)

    return (score, accuracy, predictions)
```

B) Model Implementation and Preliminary results

IT depends on the application or the situation that we consider whether the accuracy is important, or the specificity is important in that or any other metric. For this case since we are trying to analyze and predict if a person has breast cancer or not, the model can have a chance of classifying a

person without breast cancer as having one. But cannot tolerate to predict a person with breast cancer that he/she is benign.

This can be measured by the False Negative Rate this has to be low It can be obtained by the sensitivity form the Classification Report.

The False Negative Rate is the proportion of the actual positives but are predicted as negatives. It is called as sensitivity.

Implementation of Logistic Regression

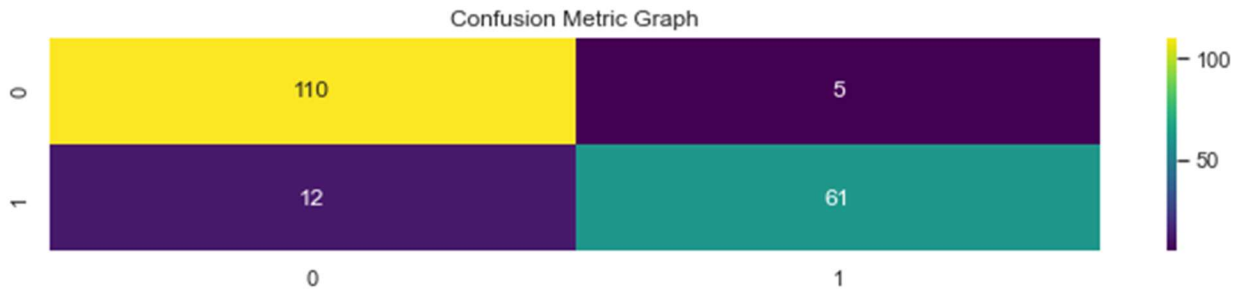
a) Classification report:

Classification Report of 'LogisticRegression '

	precision	recall	f1-score	support
0	0.90	0.96	0.93	115
1	0.92	0.84	0.88	73
accuracy			0.91	188
macro avg	0.91	0.90	0.90	188
weighted avg	0.91	0.91	0.91	188

The above image shows the classification report for the Logistic regression
The Actual Positives are: 115, F1-score for this is 93%
The Predicted negatives are: 73.
The prediction for this algorithm is nearly 91%

b) Confusion Matrix



From the above confusion matrix, we can classify the TP, TN, FP and FN.

The true positive in this case is 110 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 5 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 12. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 61.

C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 90%.

```
Full-Data Accuracy: 0.9
Cross Validation Score of'LogisticRegression '

Score: 0.91
Score: 0.91
Score: 0.9
Score: 0.9
Score: 0.9
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score, best fit and best estimator for our algorithm.

```
# Pick the model
model = LogisticRegression()
param_grid = [
              {'fit_intercept': [1],
               'penalty' : ['12'],
             'C' : [0.001 , 0.1,1]
               }
]
gsc = GridSearchCV(model,param_grid) # 10 Cross Validation

# Model Fitting
gsc.fit(X_train, y_train)

print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)

print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9132262474367737

 Best Estinator is
LogisticRegression(C=1, fit_intercept=1)

 Best Parametes are
{'C': 1, 'fit_intercept': 1, 'penalty': '12'}
```

Implementation of RandomForestClassifier

A) Classification Report:

The above image shows the classification report for the RandomForestClassifier

The Actual Positives are: 115, F1-score for this is 94%

The Predicted negatives are: 73.

The prediction for this algorithm is nearly 93%

```
Classification Report of 'RandomForestClassifier '
```

	precision	recall	f1-score	support
0	0.92	0.96	0.94	115
1	0.93	0.88	0.90	73
accuracy			0.93	188
macro avg	0.93	0.92	0.92	188
weighted avg	0.93	0.93	0.93	188

B) Confusion Matrix:

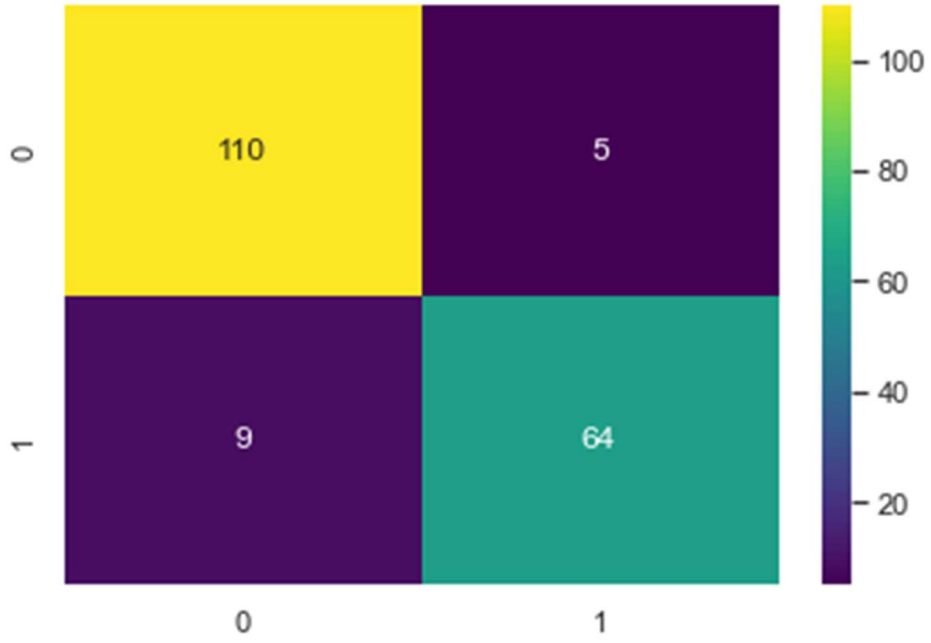
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 110 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 5 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 9. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 64.



C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 100%.

```
Full-Data Accuracy: 1.0
Cross Validation Score of'RandomForestClassifier '

Score: 0.99
Score: 0.99
Score: 0.99
Score: 1.0
Score: 1.0
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score , best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators are also shown.

```
# Pick the model
model = RandomForestClassifier()


# Tunning Params
random_grid = {'bootstrap': [True, False],
 'max_depth': [40, 50, None], # 10, 20, 30, 60, 70, 100,
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2], # , 4
 'min_samples_split': [2, 5], # , 10
 'n_estimators': [200, 400]} # , 600, 800, 1000, 1200, 1400, 1600, 1800, 2000

# Implement GridSearchCV
gsc = GridSearchCV(model, random_grid, cv=10) # 10 Cross Validation

# Model Fitting
gsc.fit(X_train, y_train)

print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)

print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9105937921727396

 Best Estinator is
RandomForestClassifier(max_depth=50, n_estimators=200)

 Best Parametes are
{'bootstrap': True, 'max_depth': 50, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Implementation of DecisionTreeClassifier

A) Classification Report:

The above image shows the classification report for the DecisionTreeClassifier

The Actual Positives are: 115, F1-score for this is 93%

The Predicted negatives are: 73.

The prediction for this algorithm is nearly 91%

Classification Report of 'DecisionTreeClassifier '

	precision	recall	f1-score	support
0	0.90	0.96	0.93	115
1	0.92	0.84	0.88	73
accuracy			0.91	188
macro avg	0.91	0.90	0.90	188
weighted avg	0.91	0.91	0.91	188

B) Confusion Matrix:

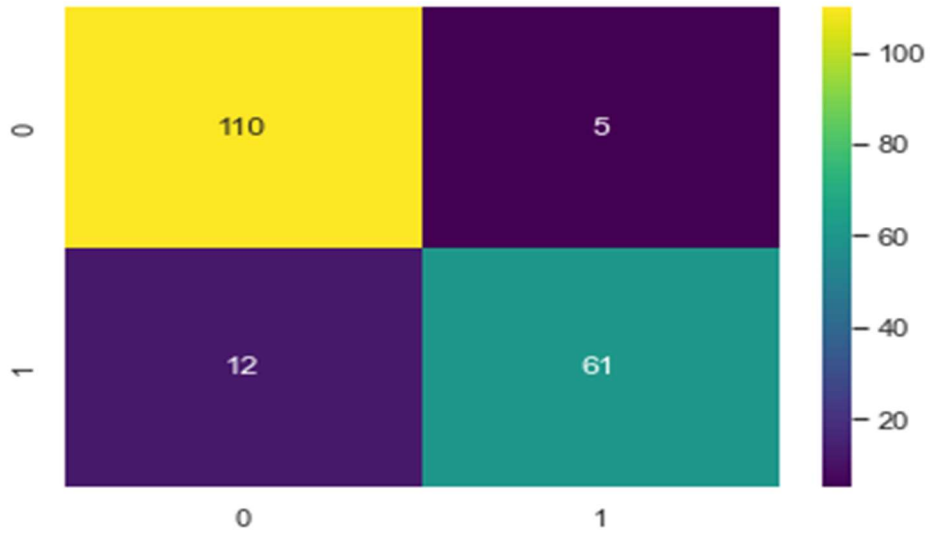
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 110 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 5 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 12. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 61.



C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 100%.

```
Full-Data Accuracy: 1.0
Cross Validation Score of'DecisionTreeClassifier '

Score: 1.0
Score: 1.0
Score: 1.0
Score: 1.0
Score: 1.0
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score , best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators are also shown.

```
# Let's Implement Grid Search Algorithm

# Pick the model
model = DecisionTreeClassifier()

# Tunning Params
param_grid = {'max_features': ['auto', 'sqrt', 'log2'],
              'min_samples_split': [2,3,4,5,6,7,8,9,10],
              'min_samples_leaf':[2,3,4,5,6,7,8,9,10] }


# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=10) # For 10 Cross-Validation

gsc.fit(X_train, y_train) # Model Fitting

print("\n Best Score is ")
print(gsc.best_score_)

print("\n Best Estinator is ")
print(gsc.best_estimator_)

print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9211875843454791

 Best Estinator is
DecisionTreeClassifier(max_features='sqrt', min_samples_leaf=5,
                       min_samples_split=3)

 Best Parametes are
{'max_features': 'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 3}
```

Among the implementation of 3 algorithms, we can see that the accuracy for random forest algorithm is high. For the logistic and Decision Tree classifier the accuracy score is nearly same.

```
dataframe_pred.sort_values('accuracy_score', ascending=False)
```

	model_name	score	accuracy_score	accuracy_percentage
1	RandomForestClassifier	0.992126	0.925532	92.55%
0	LogisticRegression	0.916010	0.909574	90.96%
2	DecisionTreeClassifier	1.000000	0.909574	90.96%

7) **Deploy Model**

We have completed our work thus far. The deployment of our model in the production map is the final phase. Therefore, we must export our model and tie it to an API for a web application. With Pickle, we can export our model and store it in a model.pkl file that we can easily retrieve and use the Web App API to compute customized predictions.

Sample code for using pickle is shown below. The result will give the accuracy of algorithm

```
import pickle
model.fit(X_train, y_train)
# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(model, open(filename, 'wb'))

# some time later...

# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
result = loaded_model.score(X_test, y_test)
print(result)
```

```
0.9095744680851063
```

```
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)

filename = 'logistic_model.pkl'
pickle.dump(logistic_model, open(filename, 'wb'))
```

8) Project Management

Implementation status report

Work completed:

Description: In this project we have implemented the Data preprocessing, EDA and Implementing probability and statical methods, Implemented Decision Tree classification, Random Forest, and Logistic regression.

Responsibility (Task, Person)

Data- preprocessing and EDA: Kota Sai Teja

Probability and Statical methods: Veena Ravuri

Importing data and machine learning implementation: Dinesh Bhogadi

Contributions (members/percentage)

Veena Ravuri: 100%

Kota Sai Teja Jana: 100%

Dinesh Bhogadi: 100%

Work to be completed:

Description: Need to implement K-NN, Naive Bias and Gaussian NB. We are also trying to implement Neural Networks implementation.

Responsibility (Task, Person)

K-NN, Naive Bias: Veena Ravuri

Resampling, cross validation: Kota Sai Teja Jana

Gaussian NB. And Neural networks: Dinesh Bhogadi

Issues or concerns:

We don’t have any issues or concerns.

9) References:

1. Anshuman; Upendra Kumar , "Machine Learning model for detection of Breast Cancer" IEEE 2021

2. Meriem Amrane; Saliha Oukid;"Breast cancer classification using machine learning" IEEE 2018

3. Udit Pratap; Saurabh Chhabra,"Breast Cancer Prediction using Different Machine Learning Algorithms", IEEE 2021

GITHUB link:

https://github.com/Dinesh101010/Project_CSCE5310

Increment 2

Introduction:

In the previous increment we have implemented algorithms like Logistic Regression, Random Forest Classifier, Decision Tree Classifier. In this Increment we are implementing the below algorithms.

- Support vector machine
- K-NN algorithm
- Gaussian NB.
- Neural Networks

Background work:

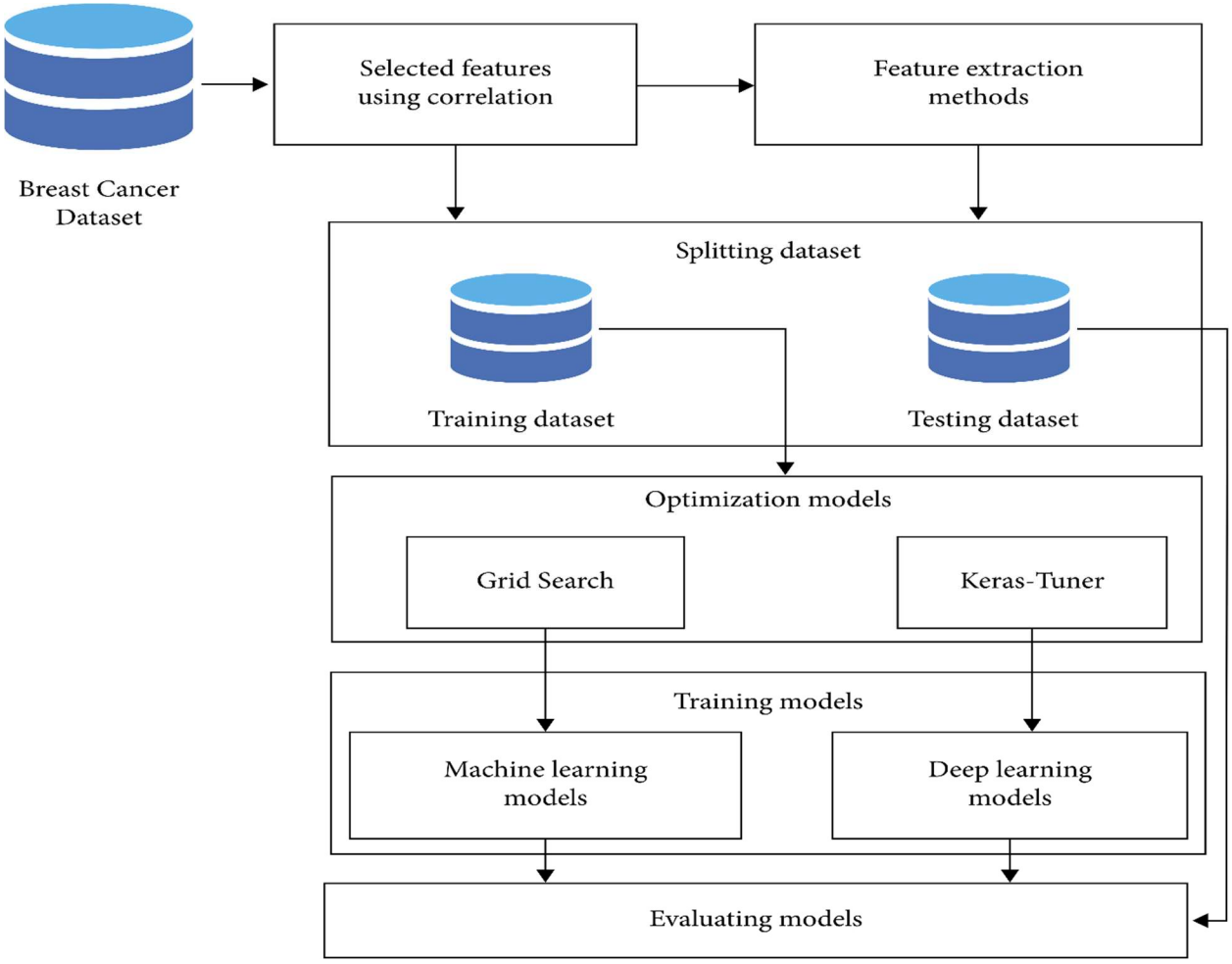
An enormous number of deaths worldwide are brought on by breast cancer. In addition to the conventional cancer detection techniques, the most recent technologies give specialists access to a wide range of adaptive methods for detecting breast cancer in female patients. Numerous data science (DS) techniques work in conjunction with new technologies to aid in the evaluation and collection of data based on cancer in order to forecast this deadly disease. Among these DS technologies, machine learning algorithms have been used to analyze data related to cancer. The effectiveness of these machine learning algorithms in improving diagnostic accuracy was demonstrated, for instance, by research [1]. In the end, a skilled doctor was able to diagnose with an accuracy rate of 79.97%. Machine learning was able to make 91.1% of the right predictions, though.

Machine learning applications have gradually grown in the medical industry over the past two decades. However, the information gathered from the patients and assessment by the doctor are what really matter for a diagnosis. The use of machine learning classifiers has reduced human error and provided quick, in-depth analysis of medical data [2]. For data modeling and prediction, there are a number of machine learning classifiers available; in our work, we used the support vector machine (SVM), logistic regression (LR), k-nearest neighbor (KNN), Gaussian NB, DecisionTreeClassifier, and RandomForestClassifier for breast cancer prediction.

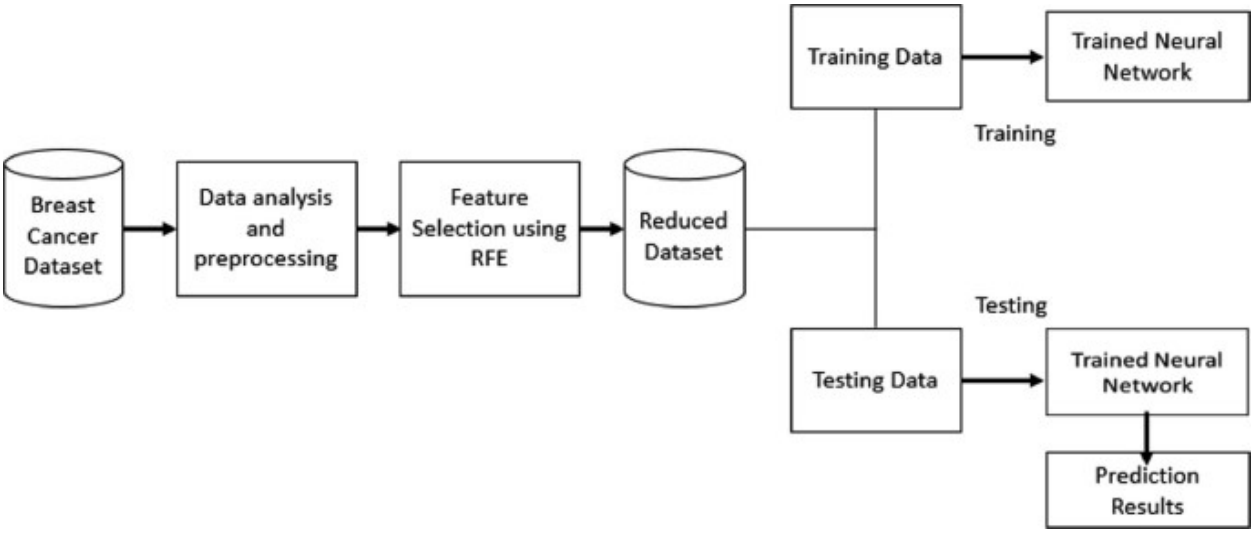
Model Explanation:

Architecture Diagram with explanation

In this step we are explaining step by step Architecture of the model. From the breast cancer dataset, we need to select features on which we are planning to work. These columns can be selected using correlation matrix. Once we have selected the features then we need to split the data. This split then goes through the training and testing phase. Training the dataset and testing the data are important steps while working on the dataset because from these values we will get the accuracy of the model. After we work on the model we are working on the optimization of the model. For the optimization if we are using machine learning models then we are using the Grid search, if we are working on the Deep learning model then we are using Kera's tuner to implement the optimization of the algorithm. Then we will evaluate the models and work on the accuracy of model. The model with high accuracy best fits the algorithm[3].



Workflow diagram with explanation



The workflow diagram of our model is initially we will take the dataset and perform all the required data analysis and data preprocessing of the dataset. This data preprocessing consists of checking Null values, checking for duplicate values, dropping all the unnecessary columns which are not useful. From the correlation matrix we can choose all the fields required for the model preparation. Then we can select the feature set using RFE then we work can work on the reduced dataset. The next step is to Train and Test the dataset. From these models we can predict the accuracy of the models [4].

Dataset

Detailed descriptions:

In this project, we are using Breast Cancer Wisconsin (Diagnostic) Data Set. This is an open-source dataset can be downloaded from Kaggle. Coming to the dataset, the data present in the csv file is from a digital image of a fine needle aspirate (FNA) of a breast mass, features are calculated. They characterize the traits of the visible cell nuclei in the picture. a classification technique that builds several machine learning algorithms using linear programming. An exhaustive search in the domain of 1-4 features and 1-3 separation planes was used to select relevant features. Information on 569 women across 32 different qualities can be found in the dataset. The first group of variables is Mean (3–13), followed by Stranded Error (1–23) and Worst (23–32), each of which has ten parameters. Mean, Standard Error, and Worst all refer to the average of all the cells, respectively. Every instance contains a parameter of cancerous and non-cancerous cells, and we can forecast cancer simply by inputting features. The feature values are presented in a numeric format. The term "Target" refers to the patient who is suffering from either "Benign" or "Malignant" cancer. Malignant denotes the presence of cancer, while benign denotes the

absence of cancer. Features in the dataset have a wide range of units and magnitudes. Therefore, it is necessary to equalize the magnitude of all aspects[5].

Features

In this data set we have 569 rows and 32 columns. Below are some of the features that will be used for the working model. There are total of benign 357 and malignant 212 values.

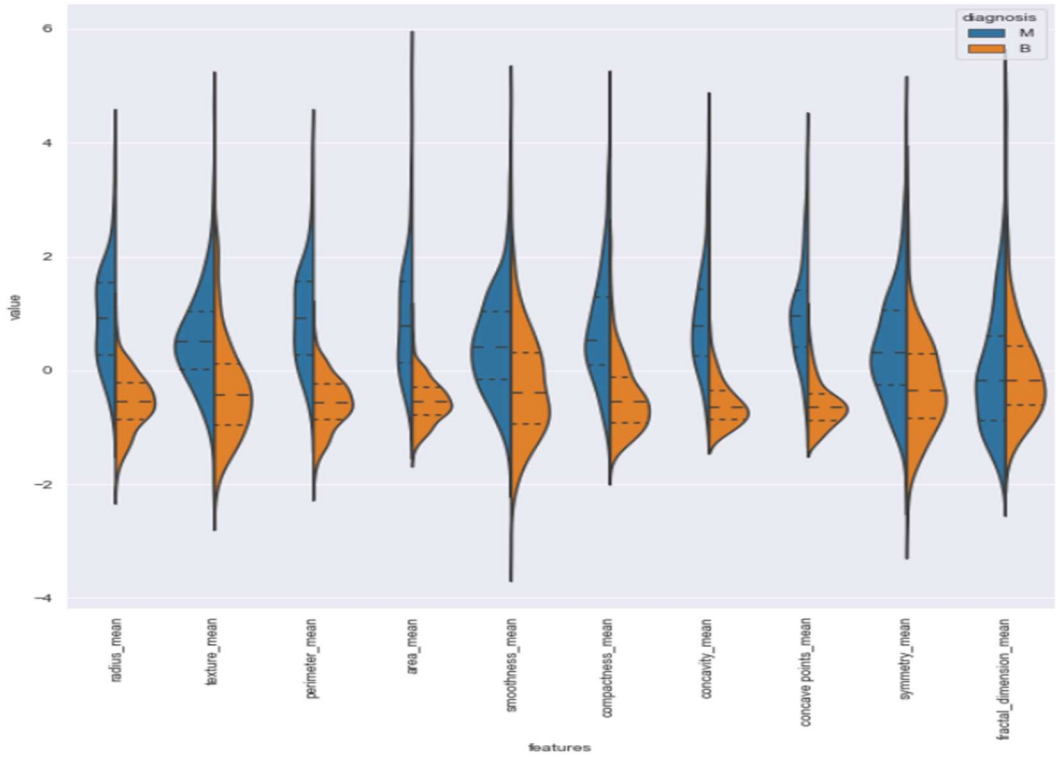
There are some real value features that can be calculated from each cell nuclei

- Radius which is the mean of distances from center to points on the perimeter
- Texture which is the standard deviation of gray-scale values.
- Perimeter which is the perimeter of the cell
- Area is the total area of the contour
- Smoothness which is the local variation in radius lengths
- Compactness can be calculated using ((perimeter)^2/area -1)
- Concavity is the severity of concave portions of the contour
- Concave points are the number of concave portions of the contour
- Symmetry is defined as the symmetry as the contour
- Fractal dimension: A fractal dimension is a ratio that compares how a pattern's level of detail alters depending on the size at which it is measured. This can be calculated by coastline approximation – 1.

Feature selection:

From the correlation matrix we have drawn above some values are strongly connected while some values are not strongly connected. Since area_mean and perimeter_mean are correlated with one another, as shown in the map heat figure radius mean, we will only use area_mean. If you're wondering how we chose area_mean as a feature, there is no right answer. We simply looked at violin plots and area_mean seemed obvious to by seeing the plot, but we can't make an exact distinction between other correlated features without trying. Let's now look for additional correlated features and evaluate accuracy using a RandomForestClassifier. There is a correlation between the three measures of concavity, compactness, and concave points. As a result, we only use concavity_mean. so we can use area_worst. Concavity_worst is what we use because it has the worst concavity, concave points, and compactness. I use concavity se because it correlates with compactness se, concavity se, and concave points se, so we can only use area_mean.

We have drawn the violin plots for all the data by taking 10 at each time for the every data that is present in the dataset. These can be seen in the python file.



Above is the graph. For informational purposes and to provide a variety of plots, we will use seaborn plots to visualize the data. Most often, violin plot and swarm plot are what I use in real life. Remember, we're not choosing features; instead, we're trying to learn information, much like when you look at the drink menu on the bar's door.

We must first normalize or standardize before introducing the violin and swarm plot. because it is very easy to see on a plot that feature values differ greatly. To make it easier to observe, I plot features into three groups, each of which contains ten features.

Data pre-processing:

We have checked the data for the null values, duplicate values, we have dropped the columns which are unnecessary.

Detecting the outliers:

We have used the IQR method to detect the outliers. We have taken the data of 'M' and 'B' fro m the data set and used the 'radius_mean' to check the outliers. After creating the lower bound and upper bound of the we can know the outliers. We can see that outliers in our data is [6.981 16.84 17.85]

Hypothesis Testing:

Let's suppose we are answering to the question? What is the probability of observing an effect by chance given a sample and an apparent effect? Selecting a test statistic is the first step in determining the apparent effect's size. The variance between two groups' means is an obvious choice for the test statistic. The second step entails defining the null hypothesis, which is a model of the system predicated on the notion that the apparent effect is not real. A null hypothesis is a type of statistical hypothesis that asserts that no statistical significance can be found in a given set of observations. A hypothesis that is being challenged is known as the null hypothesis. An alternative hypothesis is a theory that people would like to prove it. The probability of observing the apparent effect if the null hypothesis is true is determined in the third step, which is called the p-value. Let's say that the test is null. Next, the p value is determined. We reject the null hypothesis if the p value falls below or stays below a certain threshold.

When the p-value is small, an effect is considered statistically significant, which means it is unlikely to have happened by chance. As a result, we can conclude that the effect has a higher likelihood of manifesting itself in a larger population. Here's an illustration. The globe is flattening, null. Another possibility is that the world is spherical. The null hypothesis was challenged by a number of scientists. As a result, the alternative hypothesis was eventually accepted, and the null hypothesis was eventually refuted.

Let's now create our example:

I'm curious to know if there is a relationship between area_mean and radius_mean. My null hypothesis is that there is no correlation between radius_mean and area_mean in the population of tumors.

Now, in order to show that the relationship between the radius mean and the area mean exists, we must disprove this null hypothesis. Let's find the p-value (which we already know based on our prior experiences).

Below is the code for finding the statistic and p-value.

```
statistic, p_value = stats.ttest_rel(cancer_data.radius_mean,cancer_data.area_mean)
print('p-value: ',p_value)
```

```
p-value:  1.5253492492559045e-184
```

From the above code we can see that p-value is nearly zero. So we can reject null hypothesis.

Implementation of Ancova:

From the module pingouin we can import the Ancova. Ancova is analysis of co-variance. For calculating the Ancova we need to pass the dataset, deviation, covariance and between parameters. Below is the code for calculating the Ancova. With ancova we can get the behavior of data how independent variables effect the dependent variables. In our data we are using the

perimenter_mean as the dependent variable, while the independent variables are radius_mean and area_mean.

I) ANCOVA

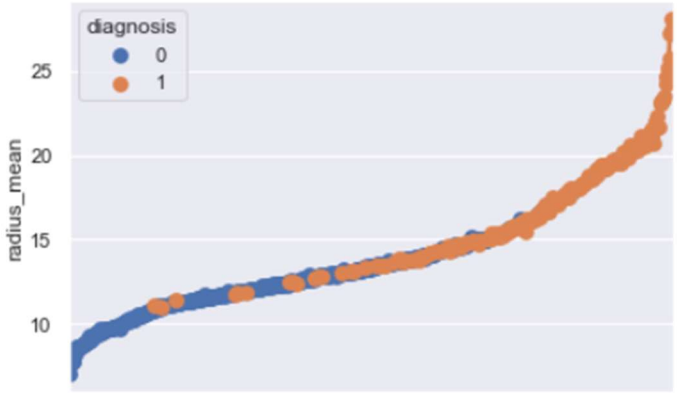
```
from pingouin import ancova
#perform ANCOVA
ancova(data=cancer_data, dv='radius_mean', covar='perimeter_mean', between='area_mean')
```

	Source	SS	DF	F	p-unc	np2
0	area_mean	29.967421	538	6.270936	8.472324e-08	0.991478
1	perimeter_mean	0.047024	1	5.294062	2.878071e-02	0.154373
2	Residual	0.257592	29	NaN	NaN	NaN

Thus, Null hypothesis states that radius_mean are strongly connected with perimeter_mean. Let's work on this. We have passed deviation as 'radius_mean', covariance as 'perimeter_mean', between as 'area_mean'. From the ANCOVA table we could say that the p-value for this data study technique is 8.472324e-08. Since this value is less than 0.05. we can accept the null hypothesis. From the hypothesis we can conclude that radius_mean and perimeter_mean is strongly related.

Implementing Two-way anova:

Two independent variables are studied during an study on one dependent variable. Here the two independent variables are 'Diagnosis' and 'area_mean'. The dependent variable is 'radius_mean'. We have plotted an sns.pointplot here to show how the values are distributed. Below is the graph.



Calculating the sum of squares for the dataset:

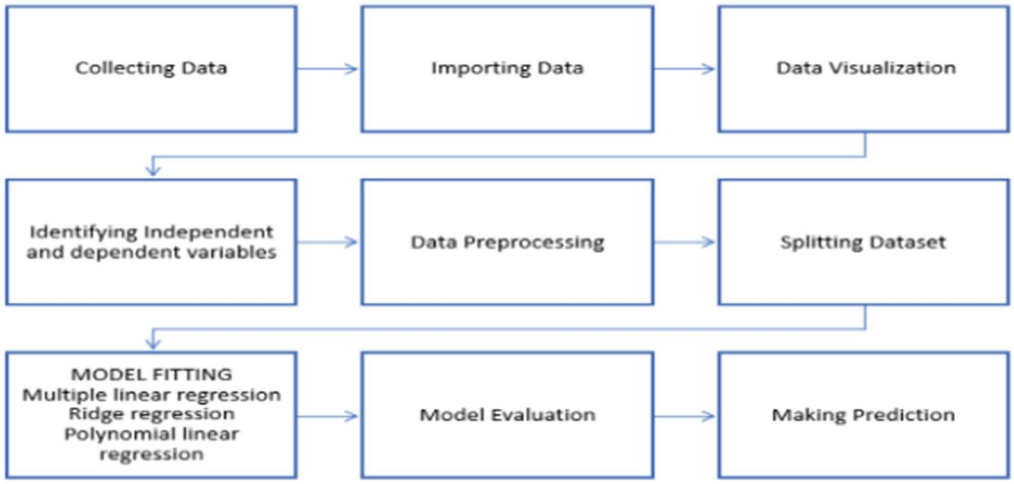
Python makes sum of squares(ss) computations (the variance in the data) quite easy. Getting the required sample size (N) and number of freedoms is where we begin. Later, while calculating the mean square, we shall employ them. We continue to calculate the sum of squares after we have the degree of freedom. The grand mean is necessary to calculate the sum of squares of both A, B, and Total. We can obtain the grand mean by applying the Pandas Data Frame method mean just to the variable which is dependent. The area_mean's grand mean is just the average of all of the scores. Sum of Squares of the data for the factor A is first calculated (diagnosis).

It basically works the same way to calculate the second Sum of Squares of another variable, B (area_mean), but over the levels of that factor. The Sum of Squares of the data Within, also known as error or residual, needs to be calculated next. Since our design is two-way, we must determine the Sum of Squares for the A and B interaction. We continue by determining the mean square for each element, their interactions, and within. Simply dividing the mean square for each effect and interaction by the mean square for inside (error/residual) yields the F-statistic. To determine whether our acquired F-ratios are higher than the threshold number, we can utilize the scipy.stats method f.sf. In order to do that, we must use our F-value along with the degrees of freedom associated with each effect and interaction, as well as the degree of freedom within. Currently, there are numerous variables holding the results. We can create a Data Frame that will house our ANOVA table in order to get a more readable result. Below is the output for the Two way anova.

	sum_sq	df	F	PR(>F)	eta_sq	omega_sq
diagnosis	3759.341799	1	-9651168.336447	NaN	0.532942	0.532942
area_mean	7053.642017	538	-33658.845558	NaN	0.999957	0.999987
diagnosis:area_mean	-3759.235449	538	17938.467118	NaN	-0.532927	-0.532897
Residual	0.198267	-509	NaN	NaN	NaN	NaN

Multiple linear regression:

Workflow diagram of the Multiple Linear regression:



As we are using the same dataset so we can directly start working on identifying the Independent variables and dependent variables in our dataset. So we are using to find the radius_mean of the cancer data so this will be an dependent variable. We will also see about independent variables those are perimeter_mean and area_mean.

First, we need to create a linear relationship between the data. Below is the figure which shows the data after creating the linear form. We can see that p-values of few variables are quite high so we can drop those variables. Before dropping those columns, we need to check the VIF of the variable. While dropping the values using VIF value we generally consider this value as 5. First preference will go to dropping the values based on the p-value followed by VIF value. Please be noted that you can drop only one value at a time.

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.4834	0.064	7.545	0.000	0.358	0.609
perimeter_mean	0.1554	0.001	140.057	0.000	0.153	0.158
area_mean	-0.0003	7.47e-05	-3.670	0.000	-0.000	-0.000
symmetry_mean	0.3721	0.187	1.994	0.047	0.006	0.739
compactness_mean	-4.7739	0.174	-27.468	0.000	-5.115	-4.433
concave points_mean	-0.7902	0.347	-2.274	0.023	-1.473	-0.108

In our model we may need to check the linear relationship exits between the:

Radius_mean (dependent variable) and area_mean (independent variable)

Radius_mean (dependent variable) and perimeter_mean (independent variable)

By fitting the model, we can get the intercept and coefficients.

Radius_mean= (intercept)+ (area_mean) *X1+ (perimeter_mean)*X2

After we plugged in the values the above equation looks like the below

Radius_mean= 0.4834 + 0.1554*X2+ -0.0003*X1.

It is evident that the coefficients collected in this table and highlighted in yellow agree with those produced by sklearn. That indicates success! Applying both sklearn and got us consistent results.

In our data we can see that the symmetry_mean has the highest p-value so we will drop the symmetry_mean and reevaluate the model. We need to repeat this process until all the values of p-value is less than 0.05 and VIF is less than 5. Now we can see that VIF is less than 5 and p value is less than 0.05.

OLS Regression Results

Dep. Variable:	radius_mean	R-squared:	0.975
Model:	OLS	Adj. R-squared:	0.975
Method:	Least Squares	F-statistic:	1.110e+04
Date:	Fri, 02 Dec 2022	Prob (F-statistic):	0.00
Time:	19:53:08	Log-Likelihood:	-472.60
No. Observations:	569	AIC:	951.2
Df Residuals:	566	BIC:	964.2
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	7.5837	0.057	133.404	0.000	7.472	7.695
area_mean	0.0098	7.66e-05	127.922	0.000	0.010	0.010
compactness_mean	1.2364	0.510	2.423	0.016	0.234	2.239

Omnibus:	403.638	Durbin-Watson:	1.910
Prob(Omnibus):	0.000	Jarque-Bera (JB):	7262.244
Skew:	-2.906	Prob(JB):	0.00
Kurtosis:	19.509	Cond. No.	1.63e+04

	Features	VIF
0	area_mean	4.92
1	compactness_mean	4.92

Implementation:

In this implementation we have written a common algorithm for all the machine learning algorithms. In that function we have passed the modelname, X_train, X_test, y_train, y_test.

In this function we are fitting the model depends on the model we have passed initially. Then we are calculating the score of the model. Followed by the predictions and the accuracy score. Depends on the accuracy score we can decide which algorithm best fits the model. This function will return the score, accuracy, and the predictions.

A) Model Building

```
def model_building(model, X_train, X_test, y_train, y_test):
    """

    Model Fitting, Prediction And Other stuff
    return ('score', 'accuracy_score', 'predictions' )
    """

    model.fit(X_train, y_train)
    score = model.score(X_train, y_train)
    predictions = model.predict(X_test)
    accuracy = accuracy_score(predictions, y_test)

    return (score, accuracy, predictions)
```

To reduce the code complexity, we have created a list in which all the models are stored. We are calling all the machine learning models using a single list. These are stored in model_list.

```
models_list = {
    "LogisticRegression" :  LogisticRegression(),
    "RandomForestClassifier" :  RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=5),
    "DecisionTreeClassifier" :  DecisionTreeClassifier(criterion='entropy', random_state=0),
    "SVC" : SVC(),
    "k-NN" : KNeighborsClassifier(),
    "Gaussian" : GaussianNB(),
}
```

Plotting the confusion matrix:

For plotting the confusion matrix we have used sns.heatmap and passing the confusion matrix through the function. Depends on the value of the cmap we will get the desired color of the confusion matrix. We have chosen cmap as viridis. Below is the image of code for the cm_metrix_graph.

C) Confusion Matrix

```
def cm_metrix_graph(cm):

    sns.heatmap(cm,annot=True,fmt="d", cmap="viridis")
    plt.show()
```

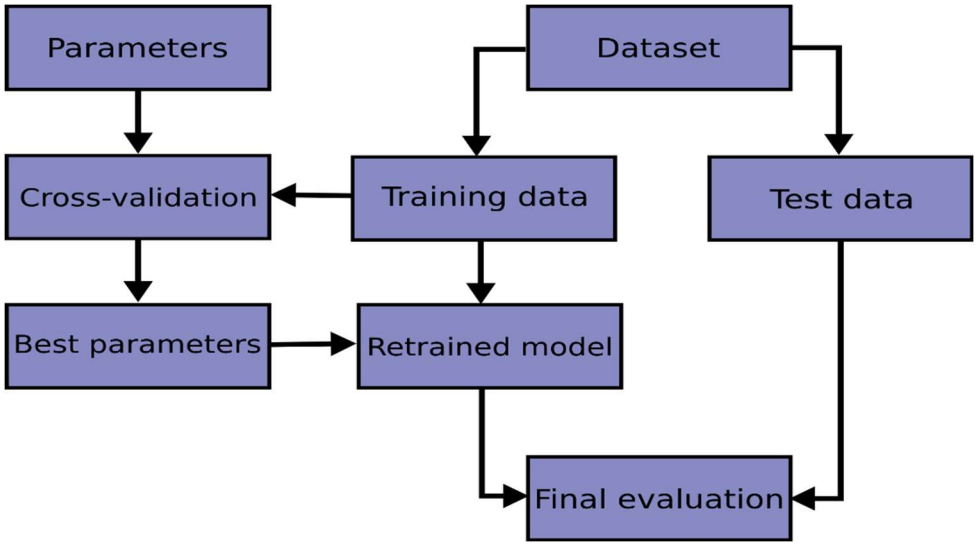
Implementation of ML models:

In this function we are iterating the models in the Dictionary we stored and calculating the accuracy, score and the classification report. This function will run through all the models in the list.

```
df_prediction = []
confusion_matrixs = []
df_prediction_cols = [ 'model_name', 'score', 'accuracy_score' , "accuracy_percentage"]

for name, model in zip(list(models_list.keys()), list(models_list.values())):

    (score, accuracy, predictions) = model_building(model, X_train, X_test, y_train, y_test )

    print("\n\nClassification Report of '"+ str(name), "'\n")

    print(classification_report(y_test, predictions))

    df_prediction.append([name, score, accuracy, "{0:.2%}".format(accuracy)])
        # For Showing Metrics
    confusion_matrixs.append(confusion_matrix(y_test, predictions))


dataframe_pred = pd.DataFrame(df_prediction, columns=df_prediction_cols)
```

Implementing cross validation for the models:

We have applied K-fold for all the models. Below is the flow diagram on how validation is done. First of all, we will choose the parameters that need to be passed through the cross-validation which will return the best parameters for the model. This Can be used for the best evaluation of the model. This is because the estimator's performance can be optimized by adjusting the parameters. As a result, assessment metrics won't accurately reflect generalization performance and knowledge from the test set may "leak" into the model. This problem can be solved by designating a different subset of the dataset as a "validation set." The training set is used for the experiment, the validation set is used for the evaluation, and the test set is used for the final evaluation if it appears that the experiment has been successful.



In our model we have taken the number of folds as 5. Above is the code for performing the cross validation. For each and every model we are performing the cross validation. Here we are splitting the models into 5 different data folds. Then we are choosing the indexes to split the data folds. Then we are again performing the fit of the model and storing all the scores. From this implementation we will get the score for each iteration and overall score. The model with high accuracy score will best fit the model. Furthermore, it is crucial that any data preparation before model fitting take place on the loop's CV-assigned training dataset as opposed to the larger data set. Any tuning of hyperparameters also falls under this. It's possible for data to leak out and for the model's skill to be overestimated if these operations aren't carried out within the loop.

A poorly chosen value for k could give an inaccurate impression of the model's skill, such as a score with a high bias or high variance (which could change significantly depending on the data used to fit the model). While there isn't a set rule, the most common choices for k are 5 or 10, respectively. The size difference between the training set and the subsets used for the resampling decreases as k increases. The bias of the method becomes less pronounced as this difference recedes.

```
def cross_val_scorring(model):

#     (score, accuracy, predictions) = model_building(model, X_train, X_test, y_train, y_test )

    model.fit(cancer_data[prediction_feature], cancer_data[targeted_feature])

    # score = model.score(X_train, y_train)

    predictions = model.predict(cancer_data[prediction_feature])
    accuracy = accuracy_score(predictions, cancer_data[targeted_feature])
    print("\nFull-Data Accuracy:", round(accuracy, 2))
    print("Cross Validation Score of'"+ str(name), "'\n")


    # Initialize K folds.
    kFold = KFold(n_splits=5) # define 5 diffrent data folds

    err = []
    for train_index, test_index in kFold.split(cancer_data):

        # print("TRAIN:", train_index, "TEST:", test_index)

        # Data Spliting via fold indexes
        X_train = cancer_data[prediction_feature].iloc[train_index, :] # train_index = rows and all columns for Prediction_featur
        y_train = cancer_data[targeted_feature].iloc[train_index] # all targeted features trains

        X_test = cancer_data[prediction_feature].iloc[test_index, :] # testing all rows and cols
        y_test = cancer_data[targeted_feature].iloc[test_index] # all targeted tests

        # Again Model Fitting
        model.fit(X_train, y_train)

        err.append(model.score(X_train, y_train))

        print("Score:", round(np.mean(err),  2) )
```

Grid search algorithm for each model:

To find the ideal values for a given model, hyperparameter tuning is done using the Grid Search method. This is important because the values supplied for the hyperparameters determine how well the entire machine learning model performs. Model hyperparameters are characteristics of a model that are external to the model and whose value cannot be deduced from data. The hyperparameter's value must be set before the learning process can begin. On the other hand, the value of a parameter can be calculated from data and is an inherent aspect of the model. The support vectors in SVMs or the beta coefficients of logistic or linear regression. While viewing the model, we can observe the algorithm in action.

Implementation of SVC

A) Classification Report:

The above image shows the classification report for the RandomForestClassifier

The Actual Positives are: 115, F1-score for this is 94%

The Predicted negatives are: 73.

The prediction for this algorithm is nearly 93%

Classification Report of 'SVC '

	precision	recall	f1-score	support
0	0.90	0.97	0.93	115
1	0.94	0.84	0.88	73
accuracy			0.91	188
macro avg	0.92	0.90	0.91	188
weighted avg	0.92	0.91	0.91	188

B) Confusion Matrix:

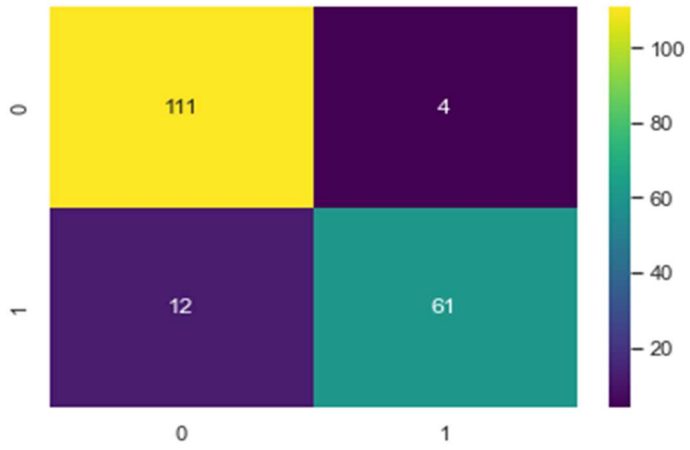
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 111 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 4 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 12. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 61.



C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 89%.

```
Full-Data Accuracy: 0.89
Cross Validation Score of'SVC '

Score: 0.9
Score: 0.89
Score: 0.88
Score: 0.88
Score: 0.88
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score, best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators is also shown. For this model we are passing ‘C’, ‘Kernel’, ‘gamma’ values this has shown that for c value 10, gamma 0.001 and kernel:’RBF ’ we are getting accurate results and best fits the model with an score of 91.8% .

D) Grid search algorithm for SVC

In [107]:
```
# Pick the model
model = SVC()
# Tunning Params
param_grid = [
 {'C': [1, 10, 100, 1000],
 'kernel': ['linear']
 },
 {'C': [1, 10, 100, 1000],
 'gamma': [0.001, 0.0001],
 'kernel': ['rbf']
 }
]

gsc = GridSearchCV(model, param_grid, cv=10) # 10 Cross Validation
# Model Fitting
gsc.fit(X_train, y_train)
print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)
print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9184885290148447

 Best Estinator is
SVC(C=10, gamma=0.001)

 Best Parametes are
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

E) Applying cross validation:

```
# Sample For testing only
cv_score = cross_validate(SVC(), X, y, cv=3,
                          scoring=('r2', 'neg_mean_squared_error'),
                          return_train_score=True)

pd.DataFrame(cv_score).describe().T
```

	count	mean	std	min	25%	50%	75%	max
fit_time	3.0	0.010841	0.012195	0.003119	0.003812	0.004505	0.014703	0.024900
score_time	3.0	0.001179	0.002043	0.000000	0.000000	0.000000	0.001769	0.003538
test_r2	3.0	0.474277	0.217123	0.235412	0.381584	0.527755	0.593709	0.659664
train_r2	3.0	0.485112	0.095093	0.415315	0.430958	0.446600	0.520010	0.593420
test_neg_mean_squared_error	3.0	-0.122946	0.050940	-0.178947	-0.144737	-0.110526	-0.094946	-0.079365
train_neg_mean_squared_error	3.0	-0.120372	0.022307	-0.136842	-0.133065	-0.129288	-0.112137	-0.094987

Here we are taking cv as 3 and taking the scoring factors as ‘r2’ and ‘neg_mean_squared_error’. This function will return the train_score.

Implementation of k-NN

A) Classification Report:

The above image shows the classification report for the RandomForestClassifier

The Actual Positives are: 115, F1-score for this is 94%

The Predicted negatives are: 73.

The prediction for this algorithm is nearly 94%

Classification Report of 'k-NN '

	precision	recall	f1-score	support
0	0.91	0.97	0.94	115
1	0.94	0.85	0.89	73
accuracy			0.92	188
macro avg	0.92	0.91	0.91	188
weighted avg	0.92	0.92	0.92	188

B) Confusion Matrix:

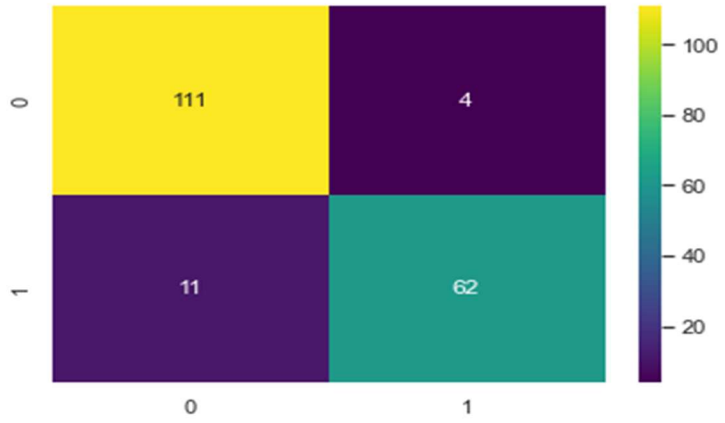
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 111 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 4 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 11. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 62.



C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 91%. We can see that maximum score is 93% but overall full data accuracy is 91%

```
Full-Data Accuracy: 0.91
Cross Validation Score of'k-NN '

Score: 0.93
Score: 0.92
Score: 0.91
Score: 0.91
Score: 0.91
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score , best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators are also shown. For Hyper tuning we are passing n_neighbors, leaf_size and weight. ‘n_neighbors’ the value is from values 1-30 and ‘leaf_size’ from the list of values from 1-30 and the weights are taken as ‘distance ’ and ‘uniform’. The best estimator is with ‘leaf_size’ is 1 and the ‘n_neighbors ’ as 10 and the weights are taken in the ‘uniform’ state. The best score during this implementation is 91.5%

B) Grid search algorithm for Kneighbors

```
model = KNeighborsClassifier()
# Tunning Params
param_grid = {
 'n_neighbors': list(range(1, 30)),
 'leaf_size': list(range(1,30)),
 'weights': [ 'distance', 'uniform' ]
}
# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=10)
# Model Fitting
gsc.fit(X_train, y_train)
print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)
print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9159244264507423

 Best Estinator is
KNeighborsClassifier(leaf_size=1, n_neighbors=10)

 Best Parametes are
{'leaf_size': 1, 'n_neighbors': 10, 'weights': 'uniform'}
```

E) Applying cross validation:

```
# Sample For testing only
cv_score = cross_validate(KNeighborsClassifier(), X, y, cv=3,
                          scoring=('r2', 'neg_mean_squared_error'),
                          return_train_score=True)

pd.DataFrame(cv_score).describe().T
```

	count	mean	std	min	25%	50%	75%	max
fit_time	3.0	0.020059	0.024397	0.003921	0.006026	0.008131	0.028127	0.048124
score_time	3.0	0.029118	0.024337	0.013329	0.015104	0.016879	0.037012	0.057144
test_r2	3.0	0.466379	0.143719	0.302876	0.413203	0.523529	0.548130	0.572731
train_r2	3.0	0.597851	0.065443	0.550243	0.560538	0.570833	0.621655	0.672478
test_neg_mean_squared_error	3.0	-0.124756	0.033718	-0.163158	-0.137135	-0.111111	-0.105556	-0.100000
train_neg_mean_squared_error	3.0	-0.094015	0.015358	-0.105263	-0.102764	-0.100264	-0.088391	-0.076517

Here we are taking cv as 3 and taking the scoring factors as ‘r2’ and ‘neg_mean_squared_error’. This function will return the train_score.

Implementation of Gaussian

A) Classification Report:

The above image shows the classification report for the RandomForestClassifier

The Actual Positives are: 115, F1-score for this is 94%

The Predicted negatives are: 73.

The prediction for this algorithm is nearly 93%

Classification Report of 'Gaussian '

	precision	recall	f1-score	support
0	0.89	0.96	0.92	115
1	0.92	0.82	0.87	73
accuracy			0.90	188
macro avg	0.91	0.89	0.90	188
weighted avg	0.91	0.90	0.90	188

B) Confusion Matrix:

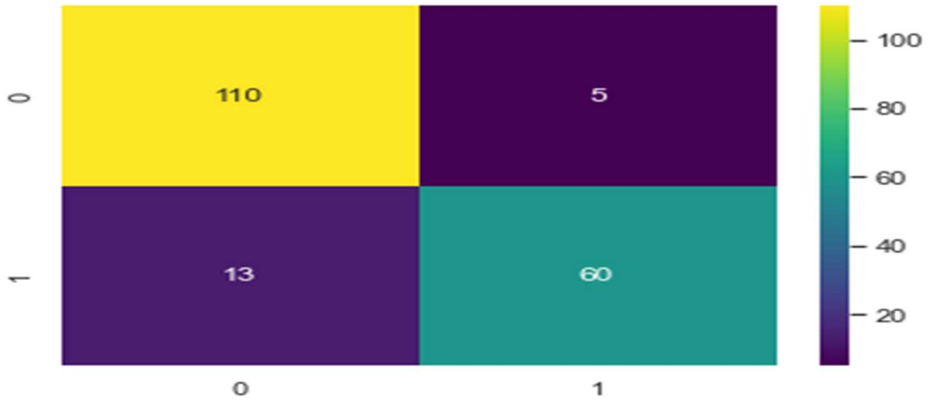
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 110 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 5 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 13. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 60.



C) Applying K-Fold to the data

By applying the K-fold to our algorithm we can test our data with various types of data. We can see that our model has an overall accuracy of 91%.

```
Full-Data Accuracy: 0.91
Cross Validation Score of'Gaussian '

Score: 0.93
Score: 0.92
Score: 0.92
Score: 0.91
Score: 0.91
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score , best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators are also shown. In this we are passing the parameters as 'Var_smoothing'. This var_smoothing has an value of np.logspace(0,-9, num=100). We can see that best var_smoothing value is 0.1. The accuracy of this model is 91.5%

```
model = GaussianNB()
# Tunning Params
param_grid = {
 'var_smoothing': np.logspace(0,-9, num=100)
}
# Implement GridSearchCV
gsc = GridSearchCV(model, param_grid, cv=10)
# Model Fitting
gsc.fit(X_train, y_train)
print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)
print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.9158569500674764

 Best Estinator is
GaussianNB(var_smoothing=0.1)

 Best Parametes are
{'var_smoothing': 0.1}
```

E) Applying cross validation:

```
# Sample For testing only
cv_score = cross_validate(GaussianNB(), X, y, cv=3,
                          scoring=('r2', 'neg_mean_squared_error'),
                          return_train_score=True)

pd.DataFrame(cv_score).describe().T
```

	count	mean	std	min	25%	50%	75%	max
fit_time	3.0	0.004127	0.003857	0.000000	0.002371	0.004741	0.006191	0.007640
score_time	3.0	0.001373	0.002378	0.000000	0.000000	0.000000	0.002060	0.004119
test_r2	3.0	0.616834	0.118011	0.482779	0.572731	0.662682	0.683862	0.705042
train_r2	3.0	0.624120	0.022822	0.606462	0.611235	0.616008	0.632949	0.649890
test_neg_mean_squared_error	3.0	-0.089594	0.027714	-0.121053	-0.100000	-0.078947	-0.073865	-0.068783
train_neg_mean_squared_error	3.0	-0.087870	0.005396	-0.092105	-0.090908	-0.089710	-0.085752	-0.081794

Here we are taking cv as 3 and taking the scoring factors as 'r2' and 'neg_mean_squared_error'. This function will return the train_score.

Implementation of Neutral-Networks using sequential:

A) Accuracy calculation:

We can see the accuracy of this algorithm is 91.48%. We are using the optimizer as 'Adam' and loss as the binary_cross entropy. Then we are fitting the model. After fitting the model we are predicting the y value using the test classification of the model. Later these results are stored to compute the confusion matrix[6].

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from keras.wrappers.scikit_learn import KerasClassifier
from keras.models import Sequential, load_model
from keras.layers import Dense
from sklearn.metrics import confusion_matrix
from keras.layers import Activation, Dense


model = Sequential()
model.add(Dense(units=16, kernel_initializer='uniform', activation='relu', input_dim=6))
model.add(Dense(units=8, kernel_initializer='uniform', activation='relu'))
model.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))
model.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, batch_size=1, epochs=10)
yPred = model.predict(X_test)
yPred = [1 if y > 0.5 else 0 for y in yPred]
matrix = confusion_matrix(y_test, yPred)
accuracy = (matrix[0][0] + matrix[1][1]) / (matrix[0][0] + matrix[0][1] + matrix[1][0] + matrix[1][1])
print("Accuracy: " + str(accuracy * 100) + "%")
```

B) Confusion Matrix:

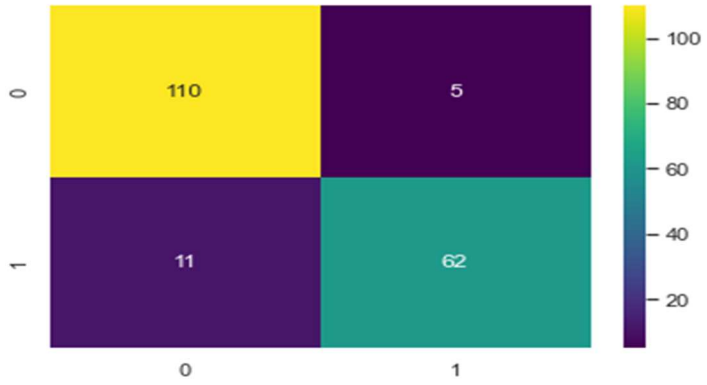
From the above confusion matrix we can classify the TP, TN, FP and FN.

The true positive in this case is 110 which means people who have been identified as malignant nodes, this means they truly has breast cancer.

The True Negative is 5 which means these people are healthy and they don't have cancer, but our model predicted them to have a breast cancer.

The False positive are misclassified with the cancer, but they are healthy the total is 11. This is also known as the Type I – error.

The False Negative is misclassified as healthy even though they have been diagnosed this value is 62.



C) **Calculating the test accuracy:** We can see the score of our model the test accuracy is 92.1 and the test loss is 17.2. This can be done by passing the model.evaluate. This will return a score which is stored in the form of array with loss and accuracy.

```
score=model.evaluate(X_train, y_train)

print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
12/12 [==============================] - 0s 2ms/step - loss: 0.1728 - accuracy: 0.9213
Test loss: 0.1728319376707077
Test accuracy: 0.9212598204612732
```

D) Hyper Tunning the ML model

From this hyper tunning we can get the best score , best fit and best estimator for our algorithm. All the best parameters are shown in the below output. And best estimator with number of estimators are also shown. For the below ANN we can see that are batch_size is 1 and optimizer

is ‘adam’. We can also pass the values from 1-100 but due to space and memory utilization we are limiting this to few values.

G) Grid search algorithm for ANN Keras

```
def classifier1(optimizer):
    model = Sequential()
    model.add(Dense(units=16, kernel_initializer='uniform', activation='relu', input_dim=6))
    model.add(Dense(units=8, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(units=6, kernel_initializer='uniform', activation='relu'))
    model.add(Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model


# Pick the model
model = KerasClassifier(build_fn=classifier1)
param_grid = [
              {'batch_size': [1, 2],'optimizer': ['adam', 'rmsprop']}
]
gsc = GridSearchCV(model,param_grid) # 10 Cross Validation

# Model Fitting
gsc.fit(X_train, y_train)

print("\n Best Score is ")
print(gsc.best_score_)
print("\n Best Estinator is ")
print(gsc.best_estimator_)

print("\n Best Parametes are")
print(gsc.best_params_)
```

```
 Best Score is
0.7974367737770081

 Best Estinator is
<keras.wrappers.scikit_learn.KerasClassifier object at 0x000001EDD9C31AF0>

 Best Parametes are
{'batch_size': 1, 'optimizer': 'adam'}
```

Results:

For comparing the results, we have stored the modelname, test score of the algorithm, accuracy score and the accuracy percentage in a data frame. From the accuracy_score we can see that RandomForestClassifier has higher accuracy when compared to other models, followed by K-NN and SVC. The accuracy score of Logistic regression, DecisionTreeClassifier and ANN are nearly same with an value of 0.909. The Gaussian NB has least accuracy with an accuracy of 90.43%. Below is the image showing results with accuracy score in ascending order [7].

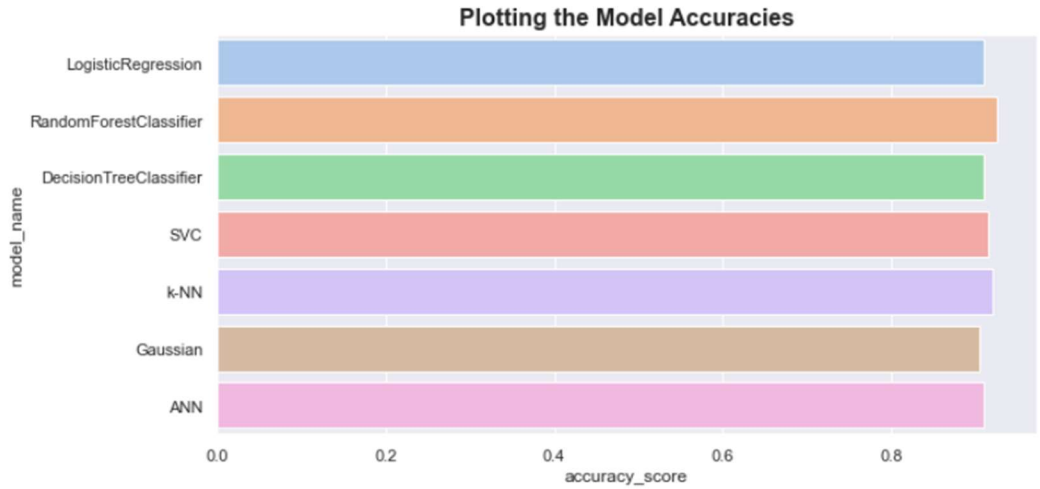
```
dataframe_pred.sort_values('accuracy_score', ascending=False)
```

	model_name	score	accuracy_score	accuracy_percentage
1	RandomForestClassifier	0.992126	0.925532	92.55%
4	k-NN	0.937008	0.920213	92.02%
3	SVC	0.923885	0.914894	91.49%
0	LogisticRegression	0.916010	0.909574	90.96%
2	DecisionTreeClassifier	1.000000	0.909574	90.96%
6	ANN	0.921260	0.909574	90.957447
5	Gaussian	0.918635	0.904255	90.43%

Plotting the bar graph using accuracy score.

```
plt.figure(figsize = (10,5))
sns.barplot(x = dataframe_pred.accuracy_score, y = dataframe_pred.model_name, palette='pastel')
plt.title("Plotting the Model Accuracies", fontsize=16, fontweight="bold")
```

```
Text(0.5, 1.0, 'Plotting the Model Accuracies')
```

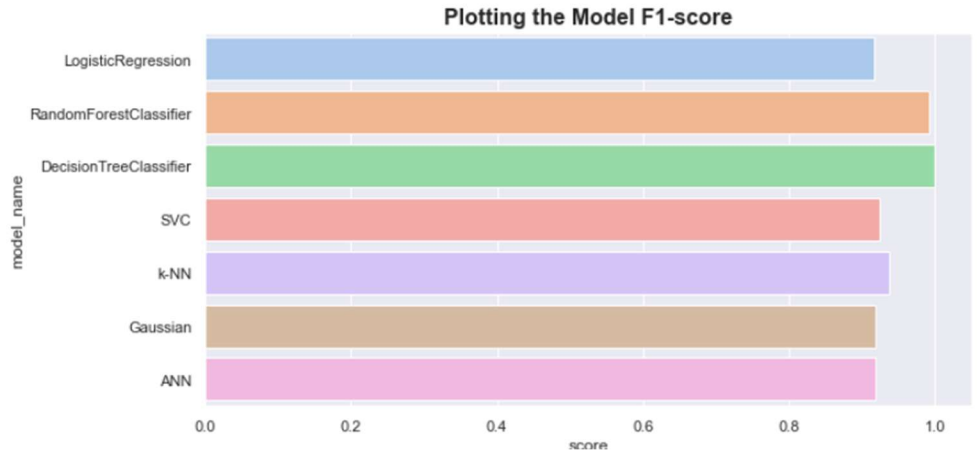


We can see all the algorithms has more than 90% accuracy. With highest being 92.55% for RandomForestClassifier and least being Gaussian NB with a value of 90.43%.

Plotting the graph based on F1-score

```
plt.figure(figsize = (10,5))
sns.barplot(x = dataframe_pred.score, y = dataframe_pred.model_name, palette='pastel')
plt.title("Plotting the Model F1-score", fontsize=16, fontweight="bold")
```

```
Text(0.5, 1.0, 'Plotting the Model F1-score')
```

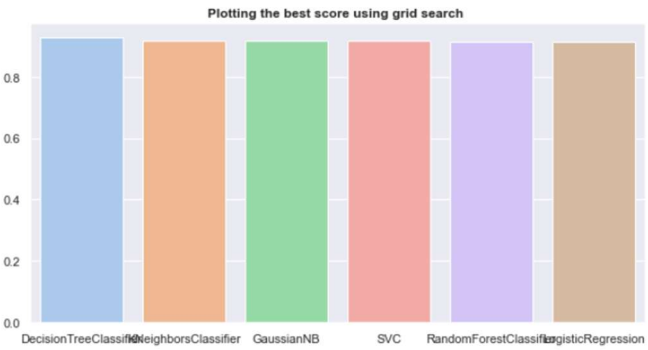


From the above graph we can see that the f1- scores of each model is greater than 0.9. The DecisionTreeClassifier has the high f1-score with an value of 1. Followed by the RandomForestClassifier with an value of 0.99 Then K-NN has the high f1-score with 0.97. Logistic regression and SVC has lowest f1-scores with 0.918 and 0.916 respectively.

Plotting the graph based on grid search score

```
In [163]: plt.figure(figsize = (10,5))
          keys=list(gridsearch_score.keys())
          values=list(gridsearch_score.values())
          sns.barplot(x = keys, y =values, palette='pastel')
          plt.title("Plotting the best score using grid search ", fontsize=12, fontweight="bold")
```

```
Out[163]: Text(0.5, 1.0, 'Plotting the best score using grid search ')
```



The above bar graph we can see the best accuracy scores after applying the grid search algorithm on each of the algorithm. The DecisionTreeClassifier has the highest accuracy of 92.1% remaining all the algorithms has an value around 91.1%. So after applying the gridsearch DecisionTreeClassifier is the best algorithm.

8) Project Management

Implementation status report

Work completed:

Description: In this project overall, we have implemented the Data preprocessing, EDA and Implementing probability and statistical methods, Implemented Decision Tree classification, Random Forest, Logistic regression, K-NN, Gaussian NB, SVC and Neural Networks using sequential. In statistical methods we have covered one-way anova, two way anova, ANCOVA, Type-I and Type -II errors. We have also implemented simple linear regression and Multiple regression. In this increment we have implemented Gaussian NB, SVC, Neural Networks, K-NN are implemented.

Responsibility (Task, Person)

Data- preprocessing and EDA and Multiple regression and results: Kota Sai Teja

Implementation of Ancova and two-way anova and Grid search algorithm: Veena Ravuri

Model Implementation(Model building, confusion Matrix, validation, Implementation): Dinesh Bhogadi

Contributions (members/percentage)

Veena Ravuri: 30%

Kota Sai Teja Jana: 30%

Dinesh Bhogadi: 40%

Issues or concerns:

We don’t have any issues or concerns.

References :

1. Huang, C.L.; Liao, H.C.; Chen, M.C. Prediction model building and feature selection with support vector machines in breast cancer diagnosis. *Expert Syst. Appl.* **2008**, *34*, 578–587.

2. Polat, K.; Güneş, S. Breast cancer diagnosis using least square support vector machine. *Digit. Signal Process.* **2007**, *17*, 694–701.

3. Rajaguru, H. Analysis of decision tree and k-nearest neighbor algorithm in the classification of breast cancer. *Asian Pac. J. Cancer Prev. APJCP* **2019**, *20*, 3777.

4. Mushtaq, Z.; Yaqub, A.; Sani, S.; Khalid, A. Effective K-nearest neighbor classifications for Wisconsin breast cancer data sets. *J. Chin. Inst. Eng.* **2020**, *43*, 80–92

5. American Cancer Society. Cancer facts & figures 2018 . Atlanta: American Cancer Society; 2018.

6. Burnside ES, Rubin DL, Fine JP, Shachter RD, Sisney GA, Leung WK. Bayesian network to predict breast cancer risk of mammographic microcalcifications and reduce number of benign biopsy results: initial experience. Radiology . 2006;240(3):666–73. doi: 10.1148/radiol.2403051096.

7. Yala A, Lehman C, Schuster T, Portnoi T, Barzilay R. A Deep Learning Mammography-based Model for Improved Breast Cancer Risk Prediction. Radiology . 2019;292(1):60–6. doi: 10.1148/radiol.2019182716.

GITHUB LINK : https://github.com/Dinesh101010/Project_CSCE5310