Machine Learning Lab 4 - Predicting Breast Cancer

Submitted By
Name: **Jortin Paul**
Register Number: **19112010**
Class: **5 BSc Data Science**

---

1. Lab Overview

Objectives

- To Predict the Breast Cancer
- To Compare and Contrast the Differences in Classification Result among Logistic Regression, K-Nearest Neighbors and Decision
  Trees
- To Demonstrate various evaluation metrices
- To Check the effect of classification with respect to change in train-test dataset, classification parameters, hyper parameters
- To

Problem Definition

As all the ML libraries were installed and verified, and also with the exprience in Exploratory Data Analysis that has got during previous lab session we can move forward to new machine algorithm like Logistic Regression Regression, K-Nearest Neighbors and Decision Trees concept. To achieve the objectives such as analyse,and predict the breast cancer we can use python and the libraries such as pandas,matplotlib,seaborn and sklearn.

Approach

The Breast Cancer data which is already provided can be analysed using python through jupyter notebook with the libraries that are already intsalled.A statistical approach has been used to find the hidden features and predict the cancer.

Using pandas to import the Dataset and visualizations can be done through matplotlib,seaborn libraries and for Logistic regression,K-nearest neighbour,Descision Tree use sklearn libraries

Sections

References

1. https://www.javatpoint.com/logistic-regression-in-machine-learning
2. https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_neares
3. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

---

2. Theoretical Background

A. Logistic Regression

- Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable.
- It predicts the output of a categorical dependent variable.
  - Therefore the outcome must be a categorical or discrete value.
  - It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- Logistic Regression is much similar to the Linear Regression except that how they are used.
  - Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the **classification problems**.
- Equation of Logistic Regression: $\log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$

B. K-Nearest Neighbors

- K-nearest neighbors (KNN) algorithm is a supervised ML algorithm which can be used for both classification as well as regression predictive problems.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity which means when a new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- Distance Functions:

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

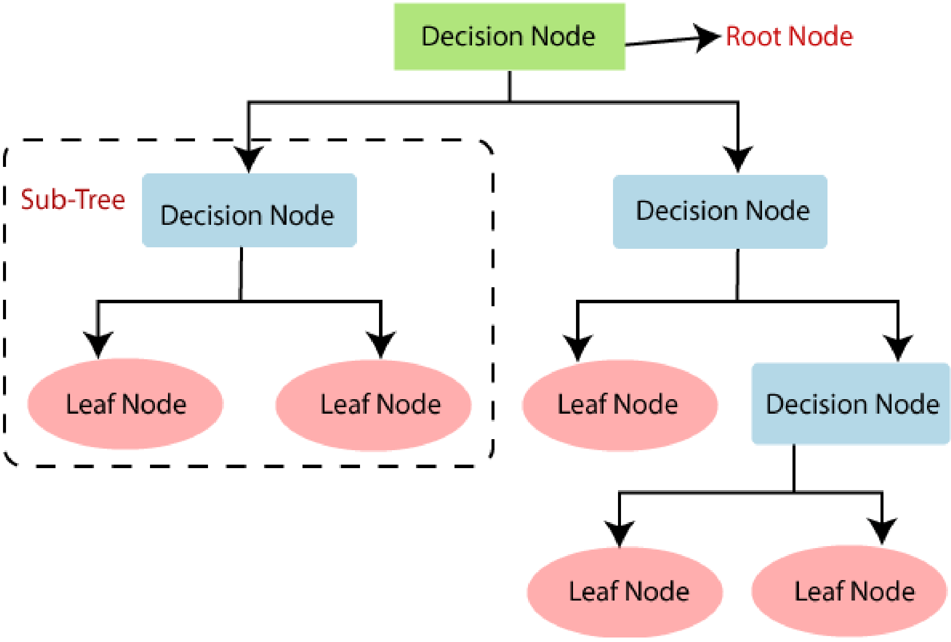
Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

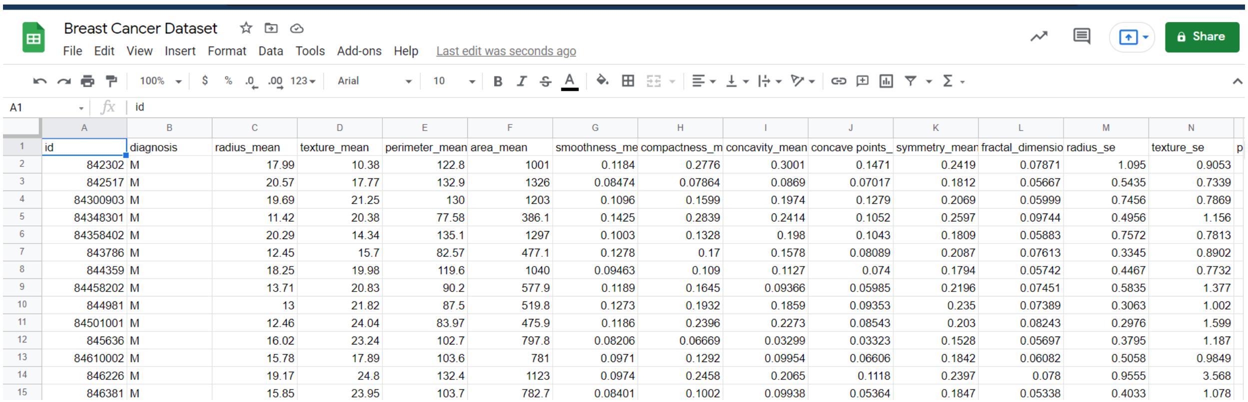
C. Decision Trees

- Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

- Structre of Decision Tree:



3. Data Overview



Breast Cancer Dataset
File Edit View Insert Format Data Tools Add-ons Help Last edit was seconds ago
Share

A1 id

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me	compactness_m	concavity_mean	concave points_	symmetry_mean	fractal_dimensio	radius_se	texture_se	p
2	842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871	1.095	0.9053	
3	842517	M	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	
4	84300903	M	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999	0.7456	0.7869	
5	84348301	M	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744	0.4956	1.156	
6	84358402	M	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883	0.7572	0.7813	
7	843786	M	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613	0.3345	0.8902	
8	844359	M	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742	0.4467	0.7732	
9	84458202	M	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451	0.5835	1.377	
10	844981	M	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389	0.3063	1.002	
11	84501001	M	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243	0.2976	1.599	
12	845636	M	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697	0.3795	1.187	
13	84610002	M	15.78	17.89	103.6	781	0.0971	0.1292	0.09954	0.06606	0.1842	0.06082	0.5058	0.9849	
14	846226	M	19.17	24.8	132.4	1123	0.0974	0.2458	0.2065	0.1118	0.2397	0.078	0.9555	3.568	
15	846381	M	15.85	23.95	103.7	782.7	0.08401	0.1002	0.09938	0.05364	0.1847	0.05338	0.4033	1.078	

Breast Cancer Dataset is a dataset of size **122 kb** that contains data of **569** people that has breast cancer

The dataset has 33 attributes that gives information of cancer such as:

- ID number
- Diagnosis (M = malignant, B = benign)
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter^2 / area - 1.0)
- concavity (severity of concave portions of the contour)
- concave points (number of concave portions of the contour)
- symmetry
- fractal dimension ("coastline approximation" - 1)

4. Exploratory Data Analysis

A. Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn import preprocessing
```

```python
import warnings
warnings.filterwarnings('ignore')
```

B. Load the Data

In [2]:
```python
# Assigning the data set a dataframe
df= pd.read_csv('C:/Users/JORTIN PAUL/Documents/PROJECTS/SEM 5/Machine Learning/Lab 4/Breast Cancer Dataset.c
```

C. Understand the Data

In [3]:
```python
# Print first 5 rows of the dataframe
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavit
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	

5 rows × 33 columns

In [4]:
```python
# Print last 5 rows of the dataframe
df.tail()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavit
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	

5 rows × 33 columns

In [5]:
```python
# Shape of the dataframe
df.shape
```

Out[5]:
```
(569, 33)
```

we have 569 rows and 33 columns

In [6]:
```python
# Summary of dataframe
df.info()
```

```
<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
```

In [7]:
```
# Columns of the dataframe
df.columns
```

Out[7]:
```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst', 'Unnamed: 32'],
      dtype='object')
```

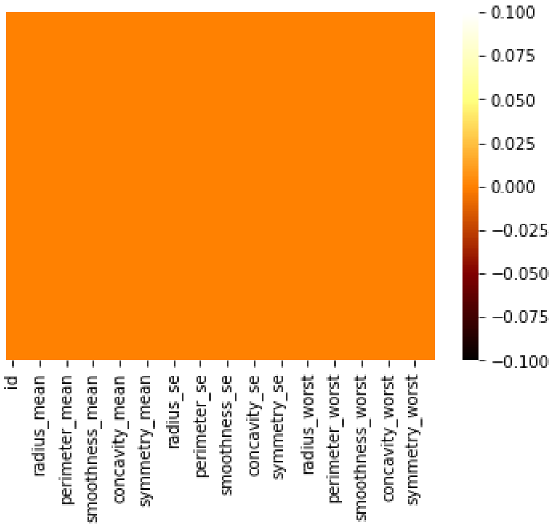
In [8]:
```
# Check for null values for Data Cleaning
df.isnull().sum()
```

Out[8]:
```
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
```

In [9]:
```
# delete the Column that has Null Values
df.drop('Unnamed: 32', axis = 1, inplace = True)
```

In [10]:
```
# Checking for null values graphicaly
sns.heatmap(df.isnull(),yticklabels=False,cmap='afmhot')
```

Out[10]: <AxesSubplot:>



As we don't have any null values in our data set we can proceed with the Analysis

In [11]:

```
# Statistical summary of the dataframe
df.describe()
```

Out[11]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.0
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.0
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.0
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.0
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.0
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.0
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.1
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.4

8 rows × 31 columns

In [12]:

```
# Unique values of each columns
df.nunique()
```

Out[12]:

```
id                         569
diagnosis                    2
radius_mean                456
texture_mean               479
perimeter_mean             522
area_mean                  539
smoothness_mean            474
compactness_mean           537
concavity_mean             537
concave points_mean        542
symmetry_mean              432
fractal_dimension_mean     499
radius_se                  540
texture_se                 519
perimeter_se               533
area_se                    528
smoothness_se              547
compactness_se             541
concavity_se               533
concave points_se          507
symmetry_se                498
fractal_dimension_se       545
radius_worst               457
texture_worst              511
perimeter_worst            514
area_worst                 544
```

```
smoothness_worst           411
compactness_worst          529
concavity_worst            539
concave points_worst       492
symmetry_worst             500
fractal_dimension_worst    535
dtype: int64
```

D. Descriptive Statistics

Correlation

In [13]:

```python
df.corr()
```

Out[13]:

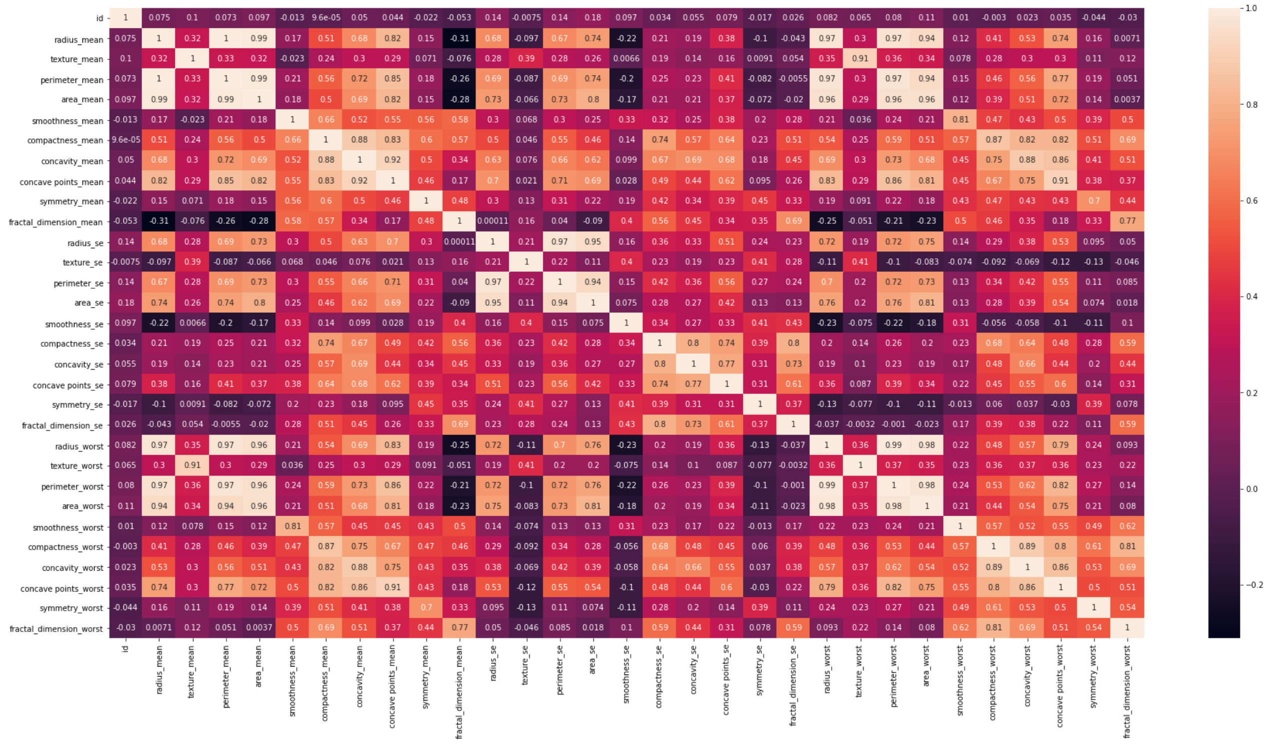
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mear
id	1.000000	0.074626	0.099770	0.073159	0.096893	-0.012968	0.00009
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	0.170581	0.50612
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	-0.023389	0.23670
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	0.207278	0.55693
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	0.177028	0.49850
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	1.000000	0.65912
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	0.659123	1.00000
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	0.521984	0.88312
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	0.553695	0.83113
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	0.557775	0.60264
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.56536
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.49747
texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.04620
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	0.296092	0.54890
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	0.246552	0.45565
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.13529
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	0.318943	0.73872
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	0.248396	0.57051
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	0.380676	0.64226
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.22997
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.50731
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	0.213120	0.53531
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	0.036072	0.24813
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	0.238853	0.59021
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	0.206718	0.50960
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	0.805324	0.56554
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	0.472468	0.86580
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	0.434926	0.81627
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	0.503053	0.81557
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	0.394309	0.51022
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	0.499316	0.68738

31 rows × 31 columns

In [14]:

```python
plt.figure(figsize = (30, 15))
sns.heatmap(df.corr(),annot=True)
```

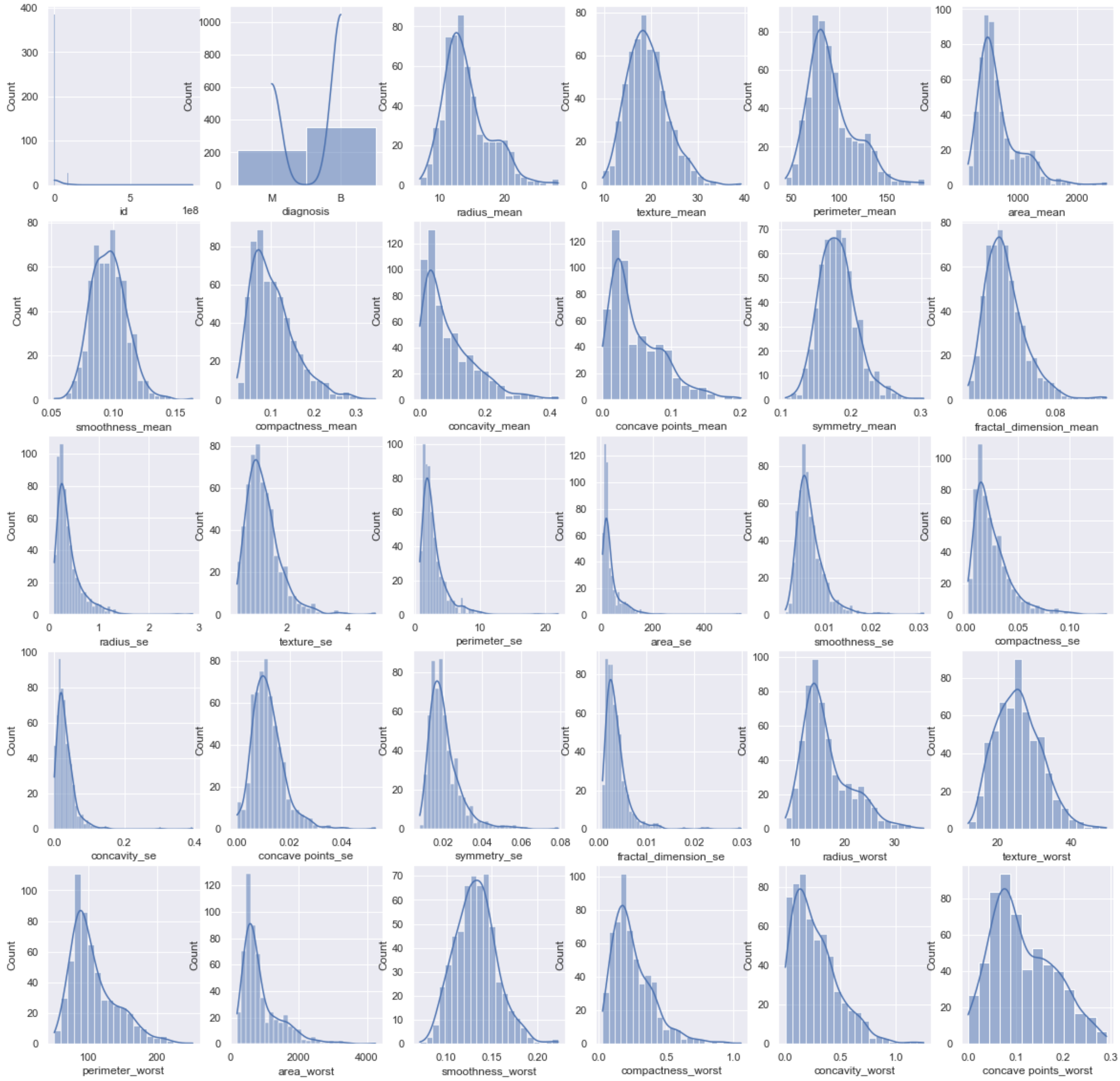
Out[14]: <AxesSubplot:>



Ditribution of Data

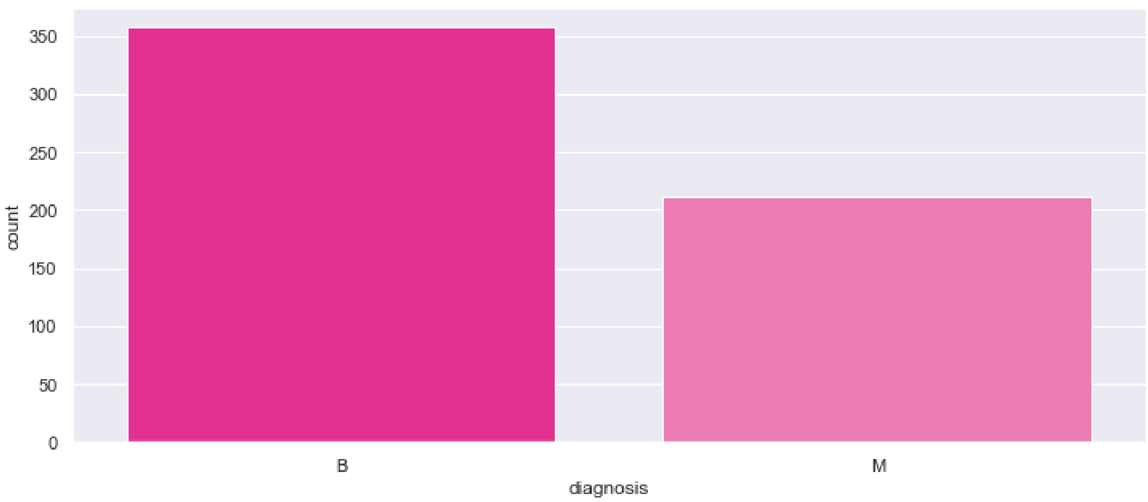
In [15]:

```
plt.figure(figsize = (20, 20))
sns.set(style="darkgrid")
plotnumber = 1

for column in df:
    if plotnumber <= 30:
        ax = plt.subplot(5, 6, plotnumber)
        sns.histplot(df[column],kde=True)
        plt.xlabel(column)

    plotnumber += 1
plt.show()
```



Diagnosis

In [16]:
```
plt.figure(figsize = (12, 5))
sns.countplot(df['diagnosis'], label="Count", palette=sns.color_palette(['#FF1493', '#FF69B4']),
              order=pd.value_counts(df['diagnosis']).iloc[:17].index)
plt.show()
```

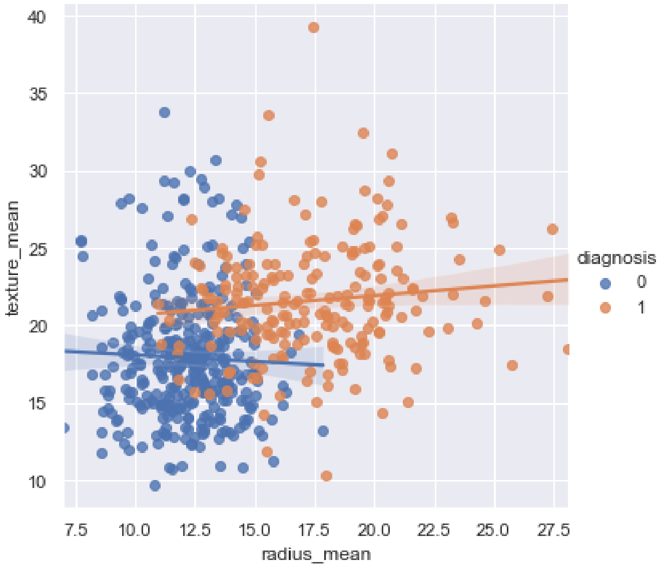


- B:Benign
- M:Malignant

In [17]:
```
def diagnosis_value(diagnosis):
    if diagnosis == 'M':
        return 1
    else:
        return 0

df['diagnosis'] = df['diagnosis'].apply(diagnosis_value)
```

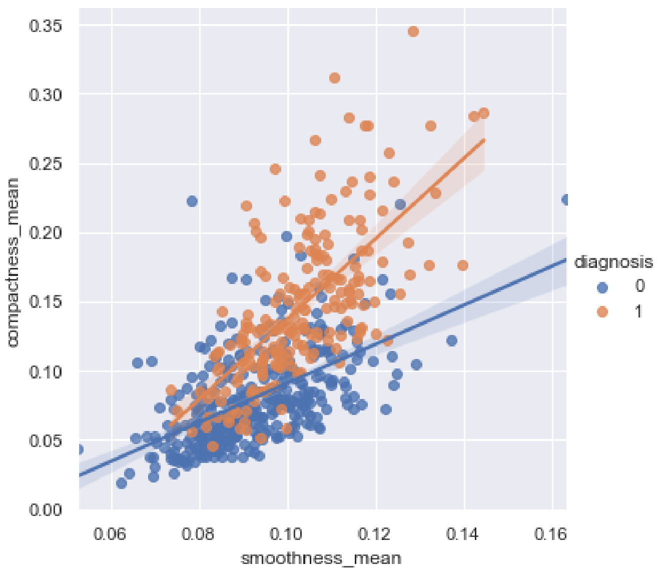
In [18]:
```
sns.lmplot(x = 'radius_mean', y = 'texture_mean', hue = 'diagnosis', data = df)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x198fbdcc7c0>



In [19]:
```
sns.lmplot(x ='smoothness_mean', y = 'compactness_mean',
           data = df, hue = 'diagnosis')
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x198fc390c40>



Modify the Dataset

In [20]:
```
df.drop('id', axis = 1, inplace = True)
```

5. Logistic Regression

In [21]:
```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [22]:
```
X=df.drop('diagnosis',axis=1)
y=df['diagnosis']
```

In [23]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.30,random_state =8)
```

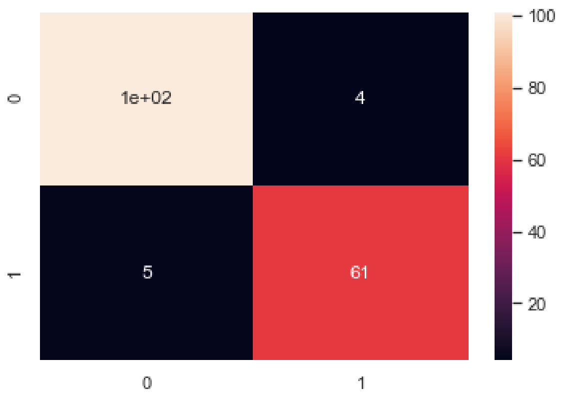
In [24]:
```
from sklearn.linear_model import LogisticRegression
l=LogisticRegression()
l.fit(X_train,y_train)
pred1=l.predict(X_test)
```

In [25]:
```
accuracy_score(pred1,y_test)
```

Out[25]:
```
0.9473684210526315
```

In [26]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,pred1)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[26]:
```
<AxesSubplot:>
```



In [27]:
```
# Classification Report
from sklearn.metrics import classification_report
class_report = classification_report(y_test, pred1)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       105
           1       0.94      0.92      0.93        66

    accuracy                           0.95       171
   macro avg       0.95      0.94      0.94       171
weighted avg       0.95      0.95      0.95       171
```

6. Decision Tree Classifier

In [28]:
```
from sklearn.tree import DecisionTreeClassifier
d=DecisionTreeClassifier()
d.fit(X_train,y_train)
pred2=d.predict(X_test)
```

In [29]:
```
accuracy_score(pred2,y_test)
```

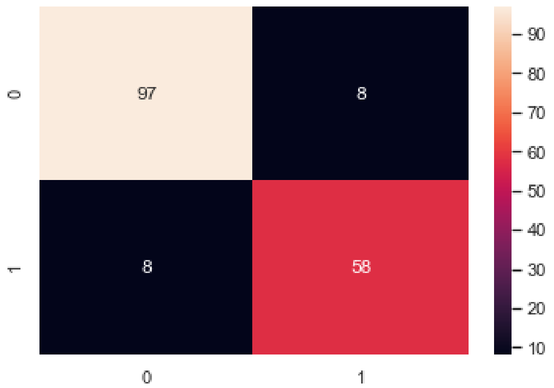
Out[29]:
```
0.9064327485380117
```

In [30]:
```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,pred2)
```

```
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[30]:

```
<AxesSubplot:>
```



In [31]:

```
# Classification Report
from sklearn.metrics import classification_report
class_report = classification_report(y_test, pred2)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92       105
           1       0.88      0.88      0.88        66

    accuracy                           0.91       171
   macro avg       0.90      0.90      0.90       171
weighted avg       0.91      0.91      0.91       171
```

7. K-Nearest Neighbours

In [32]:

```
knc =KNeighborsClassifier(n_neighbors=2)
knc.fit(X_train, y_train)
pred3=knc.predict(X_test)
```

In [33]:

```
accuracy_score(pred3,y_test)
```

Out[33]:

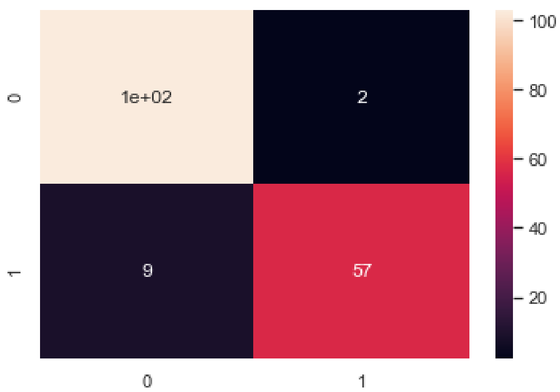
```
0.935672514619883
```

In [34]:

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,pred3)
dataframe_conf_matrix = conf_matrix
sns.heatmap(dataframe_conf_matrix, annot=True)
```

Out[34]:

```
<AxesSubplot:>
```



In [35]:

```
# Classification Report
from sklearn.metrics import classification_report
class_report = classification_report(y_test, pred3)
print(class_report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       105
           1       0.97      0.86      0.91        66

    accuracy                           0.94       171
   macro avg       0.94      0.92      0.93       171
weighted avg       0.94      0.94      0.93       171
```

8. Evalutaion on Different Test Size, Random States and Models

In [36]:

```
def KNN(X, y, test_size = 0.20, randomstate = 8,nn=5 ):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = test_size, random_state =randomstat
    cls1 =KNeighborsClassifier(n_neighbors=nn)
    cls1.fit(X_train, Y_train)
    pred1=cls1.predict(X_test)
    acc_score1 = accuracy_score(pred1,Y_test)
    return acc_score1
```

In [37]:

```
def DC(X, y, test_size = 0.20, randomstate = 8,c='gini',mf='auto'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = test_size, random_state =randomstat
    cls2=DecisionTreeClassifier(criterion=c,max_features=mf)
    cls2.fit(X_train,Y_train)
    pred2=cls2.predict(X_test)
    acc_score2 = accuracy_score(pred2,Y_test)
    return acc_score2
```

In [38]:

```
def LR(X, y, test_size = 0.20, randomstate = 8 ,penalty='l2',solver='lbfgs'):
    X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = test_size, random_state =randomstat
    cls3=LogisticRegression(penalty=penalty,solver=solver)
    cls3.fit(X_train,Y_train)
    pred3=cls3.predict(X_test)
    acc_score3 = accuracy_score(pred3,Y_test)
    return acc_score3
```

In [39]:

```
df3 = pd.DataFrame(columns = ['Test Size', 'Random States','Decision Tree Accuracy','Logistic regression Accu
```

In [40]:

```
test_size = [0.30, 0.25, 0.20,0.10]
random_states = [8, 27, 42]
n_neighbours = [2,3,4,5]

criterions=['gini', 'entropy']
maxfeatures=['auto', 'sqrt', 'log2']
penalties = [ 'l1', 'elasticnet','none', 'l2']
solvers = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
```

In [41]:

```
df4 = pd.DataFrame(columns = ['Test Size', 'Random States','Number of neighbours','K-nearest neighbour Accura
df5 = pd.DataFrame(columns = ['Test Size', 'Random States','Decision Tree Accuracy','Criterions','Max feature
df6 = pd.DataFrame(columns = ['Test Size', 'Random States','Solvers','Logistic regression Accuracy','Penalty'
```

In [42]:

```
for t_size in test_size:
    for r_state in random_states:
        for neigh in n_neighbours:
            a1 = KNN(X, y, t_size, r_state,neigh)
            I1 = {}
            I1['Test Size'] = t_size
            I1['Random States'] = r_state
            I1['Number of neighbours'] = neigh
            I1['K-nearest neighbour Accuracy'] = a1

            df4 = df4.append(I1, ignore_index = True)


df4
```

Out[42]:

	Test Size	Random States	Number of neighbours	K-nearest neighbour Accuracy
0	0.30	8.0	2.0	0.935673
1	0.30	8.0	3.0	0.941520

	Test Size	Random States	Number of neighbours	K-nearest neighbour Accuracy
2	0.30	8.0	4.0	0.947368
3	0.30	8.0	5.0	0.941520
4	0.30	27.0	2.0	0.906433
5	0.30	27.0	3.0	0.923977
6	0.30	27.0	4.0	0.906433
7	0.30	27.0	5.0	0.912281
8	0.30	42.0	2.0	0.941520
9	0.30	42.0	3.0	0.941520
10	0.30	42.0	4.0	0.953216
11	0.30	42.0	5.0	0.959064
12	0.25	8.0	2.0	0.923077
13	0.25	8.0	3.0	0.944056
14	0.25	8.0	4.0	0.930070
15	0.25	8.0	5.0	0.930070
16	0.25	27.0	2.0	0.895105
17	0.25	27.0	3.0	0.916084
18	0.25	27.0	4.0	0.909091
19	0.25	27.0	5.0	0.923077
20	0.25	42.0	2.0	0.944056
21	0.25	42.0	3.0	0.930070
22	0.25	42.0	4.0	0.951049
23	0.25	42.0	5.0	0.965035
24	0.20	8.0	2.0	0.964912
25	0.20	8.0	3.0	0.964912
26	0.20	8.0	4.0	0.964912
27	0.20	8.0	5.0	0.947368
28	0.20	27.0	2.0	0.885965
29	0.20	27.0	3.0	0.938596
30	0.20	27.0	4.0	0.912281
31	0.20	27.0	5.0	0.929825
32	0.20	42.0	2.0	0.938596
33	0.20	42.0	3.0	0.929825
34	0.20	42.0	4.0	0.947368
35	0.20	42.0	5.0	0.956140
36	0.10	8.0	2.0	0.964912
37	0.10	8.0	3.0	0.964912
38	0.10	8.0	4.0	0.947368
39	0.10	8.0	5.0	0.964912
40	0.10	27.0	2.0	0.912281
41	0.10	27.0	3.0	0.964912
42	0.10	27.0	4.0	0.929825
43	0.10	27.0	5.0	0.947368
44	0.10	42.0	2.0	0.947368
45	0.10	42.0	3.0	0.964912
46	0.10	42.0	4.0	0.964912

	Test Size	Random States	Number of neighbours	K-nearest neighbour Accuracy
47	0.10	42.0	5.0	0.964912

In [43]:

```
for t_size in test_size:
    for r_state in random_states:
        for crs in criterions:
            for mfs in maxfeatures:

                a2 = DC(X, y, t_size, r_state,crs,mfs)
                I2 = {}
                I2['Test Size'] = t_size
                I2['Random States'] = r_state
                I2['Decision Tree Accuracy'] = a2
                I2['Criterions'] = crs
                I2['Max features'] = mfs

                df5 = df5.append(I2, ignore_index = True)

df5
```

Out[43]:

	Test Size	Random States	Decision Tree Accuracy	Criterions	Max features
0	0.3	8	0.953216	gini	auto
1	0.3	8	0.918129	gini	sqrt
2	0.3	8	0.888889	gini	log2
3	0.3	8	0.953216	entropy	auto
4	0.3	8	0.912281	entropy	sqrt
...	...	...	...	...	...
67	0.1	42	0.947368	gini	sqrt
68	0.1	42	0.894737	gini	log2
69	0.1	42	0.964912	entropy	auto
70	0.1	42	0.964912	entropy	sqrt
71	0.1	42	0.947368	entropy	log2

72 rows × 5 columns

In [44]:

```
for t_size in test_size:
    for r_state in random_states:
        for penalty in penalties:
            for solver in solvers:
                a3 = LR(X, y, t_size, r_state)
                I3 = {}
                I3['Test Size'] = t_size
                I3['Random States'] = r_state
                I3['Solvers'] = solver
                I3['Penalty'] = penalty
                I3['Logistic regression Accuracy'] = a3


                df6 = df6.append(I3, ignore_index = True)


df6
```

Out[44]:

	Test Size	Random States	Solvers	Logistic regression Accuracy	Penalty
0	0.3	8	newton-cg	0.947368	l1
1	0.3	8	lbfgs	0.947368	l1
2	0.3	8	liblinear	0.947368	l1
3	0.3	8	sag	0.947368	l1
4	0.3	8	saga	0.947368	l1
...	...	...	...	...	...
235	0.1	42	newton-cg	0.982456	l2

	Test Size	Random States	Solvers	Logistic regression Accuracy	Penalty
236	0.1	42	lbfgs	0.982456	l2
237	0.1	42	liblinear	0.982456	l2
238	0.1	42	sag	0.982456	l2
239	0.1	42	saga	0.982456	l2

240 rows × 5 columns

In [45]:

```
for t_size in test_size:
    for r_state in random_states:
        a1 = KNN(X, y, t_size, r_state)
        a2 = DC(X, y, t_size, r_state)
        a3 = LR(X, y, t_size, r_state)
        I = {}
        I['Test Size'] = t_size
        I['Random States'] = r_state
        I['Decision Tree Accuracy'] = a2
        I['Logistic regression Accuracy'] = a3
        I['K-nearest neighbour Accuracy'] = a1

        df3 = df3.append(I, ignore_index = True)

df3
```

Out[45]:

	Test Size	Random States	Decision Tree Accuracy	Logistic regression Accuracy	K-nearest neighbour Accuracy
0	0.30	8.0	0.912281	0.947368	0.941520
1	0.30	27.0	0.912281	0.929825	0.912281
2	0.30	42.0	0.918129	0.970760	0.959064
3	0.25	8.0	0.944056	0.944056	0.930070
4	0.25	27.0	0.923077	0.923077	0.923077
5	0.25	42.0	0.937063	0.972028	0.965035
6	0.20	8.0	0.947368	0.956140	0.947368
7	0.20	27.0	0.929825	0.947368	0.929825
8	0.20	42.0	0.947368	0.956140	0.956140
9	0.10	8.0	0.894737	0.964912	0.964912
10	0.10	27.0	0.947368	0.929825	0.947368
11	0.10	42.0	0.964912	0.982456	0.964912

9. Conclusion

Through the Predicting Breast Cancer lab,we could know more regarding the Logistic Regresssion, Decision Tree Classifier and K-Nearest Neighbours and how to use it in a real life sitauation and also got exposure on how to do evaluation Metrices.

Finally could understand which algorithm is good under which random state and test size for this situation.

10. Future Enhancement

Could Enhance the Lab session using more Visulization