INTRODUCTION

- Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.
- That means it identifies pattern or structure after analysing the data. Based on all these findings we formulated the data to make prediction of new obervartion.
- Breast cancer is considered one of the most common cancers in women caused by various clinical, lifestyle, social, and economic factors.In this project,we will be using a Breast Cancer Dataset to predict whether the cancer type is Malignant(cancerous) or Benign(non-cancerous) by using various machine learning models like Decision Tree Classifier,Logistic Regression,Supervised Vector Machine(SVM),K-nearest neighbour(KNN) to determine which algorithm is best for this particular problem.
- Machine learning has the potential to predict breast cancer based on features hidden in data as a modeling approach, represents the process of extracting knowledge from data and discovering hidden relationships , widely used in healthcare in recent years to predict different diseases

PROBLEM STATEMENT

-The problem statement for this machine learning model to predict whether the breast cancer type is Maligant(cancerous) or Bening(non-cancerous).

Importing Essentail Python Libraries

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

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

Exploratory Data Analysis (EDA)

Importing DataSet:

```
Dataset Name = Breast Cancer Dataset
```

In [2]:
```python
df = pd.read_csv(r'C:\Users\Hp\Downloads\breast-cancer.csv')
df
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.
565	926682	M	20.13	28.25	131.20	1261.0	0.0
566	926954	M	16.60	28.08	108.30	858.1	0.0
567	927241	M	20.60	29.33	140.10	1265.0	0.
568	92751	B	7.76	24.54	47.92	181.0	0.0

569 rows × 32 columns

In [3]:
```python
df.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
0	842302	M	17.99	10.38	122.80	1001.0	0.118
1	842517	M	20.57	17.77	132.90	1326.0	0.084
2	84300903	M	19.69	21.25	130.00	1203.0	0.109
3	84348301	M	11.42	20.38	77.58	386.1	0.142
4	84358402	M	20.29	14.34	135.10	1297.0	0.100

5 rows × 32 columns

In [4]:
```
df.tail()
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
564	926424	M	21.56	22.39	142.00	1479.0	0.11
565	926682	M	20.13	28.25	131.20	1261.0	0.09
566	926954	M	16.60	28.08	108.30	858.1	0.08
567	927241	M	20.60	29.33	140.10	1265.0	0.11
568	92751	B	7.76	24.54	47.92	181.0	0.05

5 rows × 32 columns

Handling Missing Values

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 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
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
dtype: int64
```

In [7]:
```
df.head().T
```

Out[7]:

	0	1	2	3	4
id	842302	842517	84300903	84348301	84358402
diagnosis	M	M	M	M	M
radius_mean	17.99	20.57	19.69	11.42	20.29
texture_mean	10.38	17.77	21.25	20.38	14.34
perimeter_mean	122.8	132.9	130.0	77.58	135.1
area_mean	1001.0	1326.0	1203.0	386.1	1297.0
smoothness_mean	0.1184	0.08474	0.1096	0.1425	0.1003
compactness_mean	0.2776	0.07864	0.1599	0.2839	0.1328
concavity_mean	0.3001	0.0869	0.1974	0.2414	0.198
concave points_mean	0.1471	0.07017	0.1279	0.1052	0.1043
symmetry_mean	0.2419	0.1812	0.2069	0.2597	0.1809
fractal_dimension_mean	0.07871	0.05667	0.05999	0.09744	0.05883
radius_se	1.095	0.5435	0.7456	0.4956	0.7572
texture_se	0.9053	0.7339	0.7869	1.156	0.7813
perimeter_se	8.589	3.398	4.585	3.445	5.438
area_se	153.4	74.08	94.03	27.23	94.44
smoothness_se	0.006399	0.005225	0.00615	0.00911	0.01149
compactness_se	0.04904	0.01308	0.04006	0.07458	0.02461
concavity_se	0.05373	0.0186	0.03832	0.05661	0.05688
concave points_se	0.01587	0.0134	0.02058	0.01867	0.01885
symmetry_se	0.03003	0.01389	0.0225	0.05963	0.01756
fractal_dimension_se	0.006193	0.003532	0.004571	0.009208	0.005115
radius_worst	25.38	24.99	23.57	14.91	22.54
texture_worst	17.33	23.41	25.53	26.5	16.67
perimeter_worst	184.6	158.8	152.5	98.87	152.2
area_worst	2019.0	1956.0	1709.0	567.7	1575.0
smoothness_worst	0.1622	0.1238	0.1444	0.2098	0.1374
compactness_worst	0.6656	0.1866	0.4245	0.8663	0.205
concavity_worst	0.7119	0.2416	0.4504	0.6869	0.4
concave points_worst	0.2654	0.186	0.243	0.2575	0.1625
symmetry_worst	0.4601	0.275	0.3613	0.6638	0.2364
fractal_dimension_worst	0.1189	0.08902	0.08758	0.173	0.07678

column 'id' is not useful for this problem so we will drop the

In [8]:
```
df.drop('id',axis=1,inplace=True)
```

In [9]:
```
df
```

Out[9]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	
565	M	20.13	28.25	131.20	1261.0	0.09780	
566	M	16.60	28.08	108.30	858.1	0.08455	
567	M	20.60	29.33	140.10	1265.0	0.11780	
568	B	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

In [10]:
```
df.describe()
```

Out[10]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactnes
count	569.000000	569.000000	569.000000	569.000000	569.000000	569
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0
std	3.524049	4.301036	24.298981	351.914129	0.014064	0
min	6.981000	9.710000	43.790000	143.500000	0.052630	0
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0

8 rows × 30 columns

Now we will build some insights from the data through various graphical representation

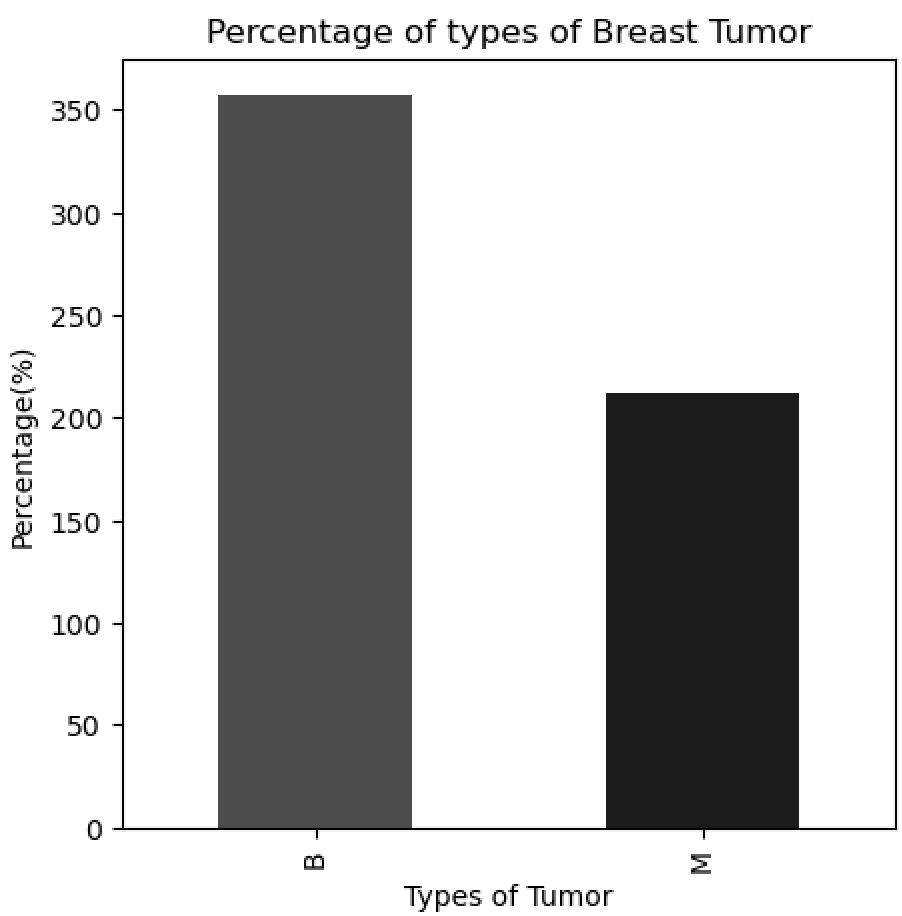
In [11]:
```
df['diagnosis'].value_counts()
```

Out[11]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

Bar Graph

In [12]:
```
plt.figure(figsize=(5,5))
df['diagnosis'].value_counts().plot(kind = 'bar',color=['red','blue'])
plt.xlabel('Types of Tumor')
plt.ylabel('Percentage(%)')
plt.title('Percentage of types of Breast Tumor')
plt.show
```

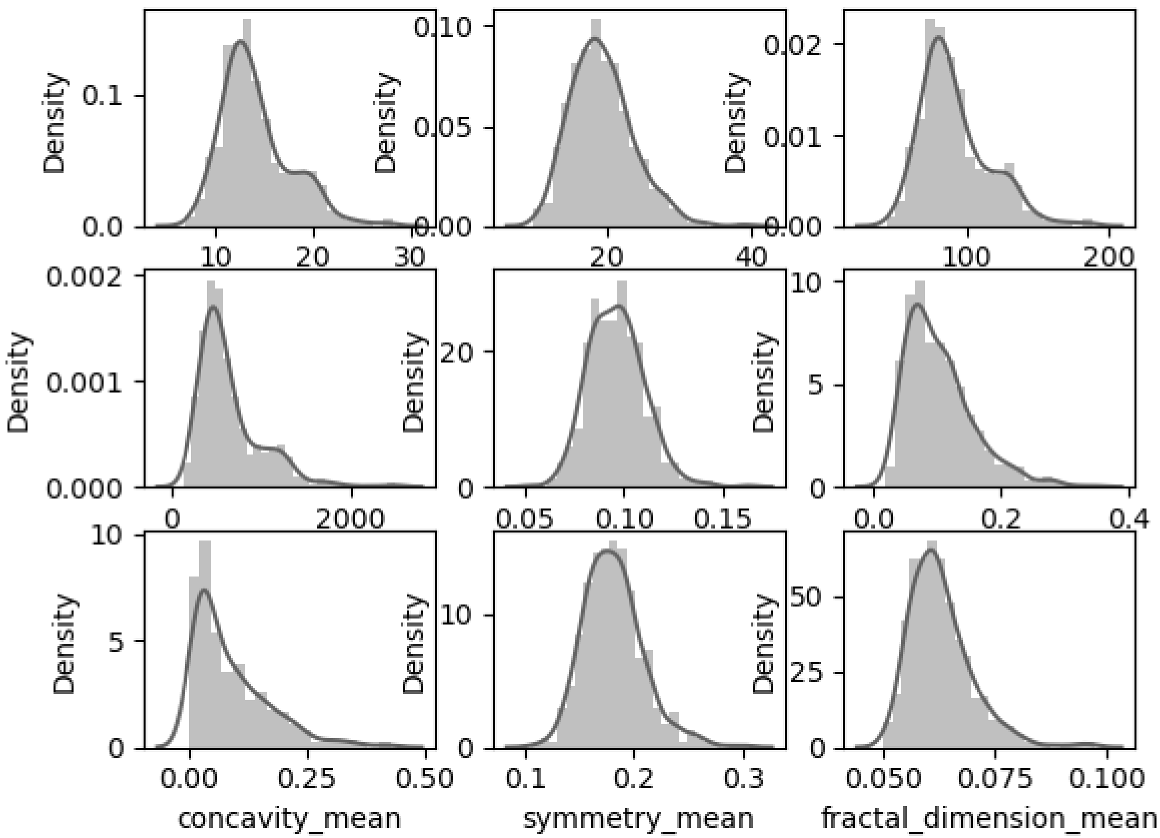
Out[12]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Visualisation multiple graphs (density distribution)

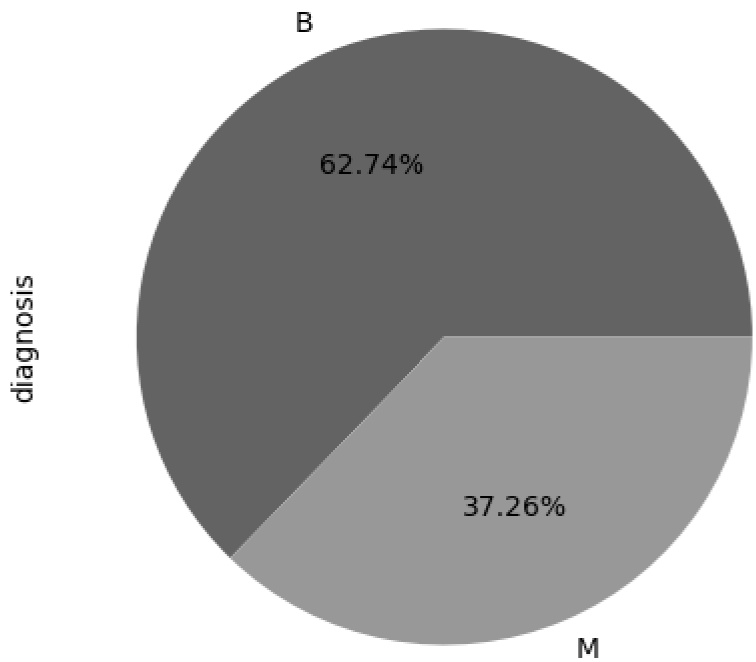
In [13]:
```
plt.subplot(3,3,1)
sns.distplot(df.radius_mean)

plt.subplot(3,3,2)
sns.distplot(df.texture_mean)

plt.subplot(3,3,3)
sns.distplot(df.perimeter_mean)

plt.subplot(3,3,4)
sns.distplot(df.area_mean)

plt.subplot(3,3,5)
sns.distplot(df.smoothness_mean)

plt.subplot(3,3,6)
sns.distplot(df.compactness_mean)

plt.subplot(3,3,7)
sns.distplot(df.concavity_mean)


plt.subplot(3,3,8)
sns.distplot(df.symmetry_mean)

plt.subplot(3,3,9)
sns.distplot(df.fractal_dimension_mean)
```

Out[13]:
```
<AxesSubplot:xlabel='fractal_dimension_mean', ylabel='Density'>
```


Density
concavity_mean
symmetry_mean
fractal_dimension_mean

Pie chart

In [14]:
```
plt.figure(figsize=(5,5))
df['diagnosis'].value_counts().plot(kind = 'pie',autopct = '%1.2f%%')
plt.show()
```



In [15]:
```
df.corr()
```

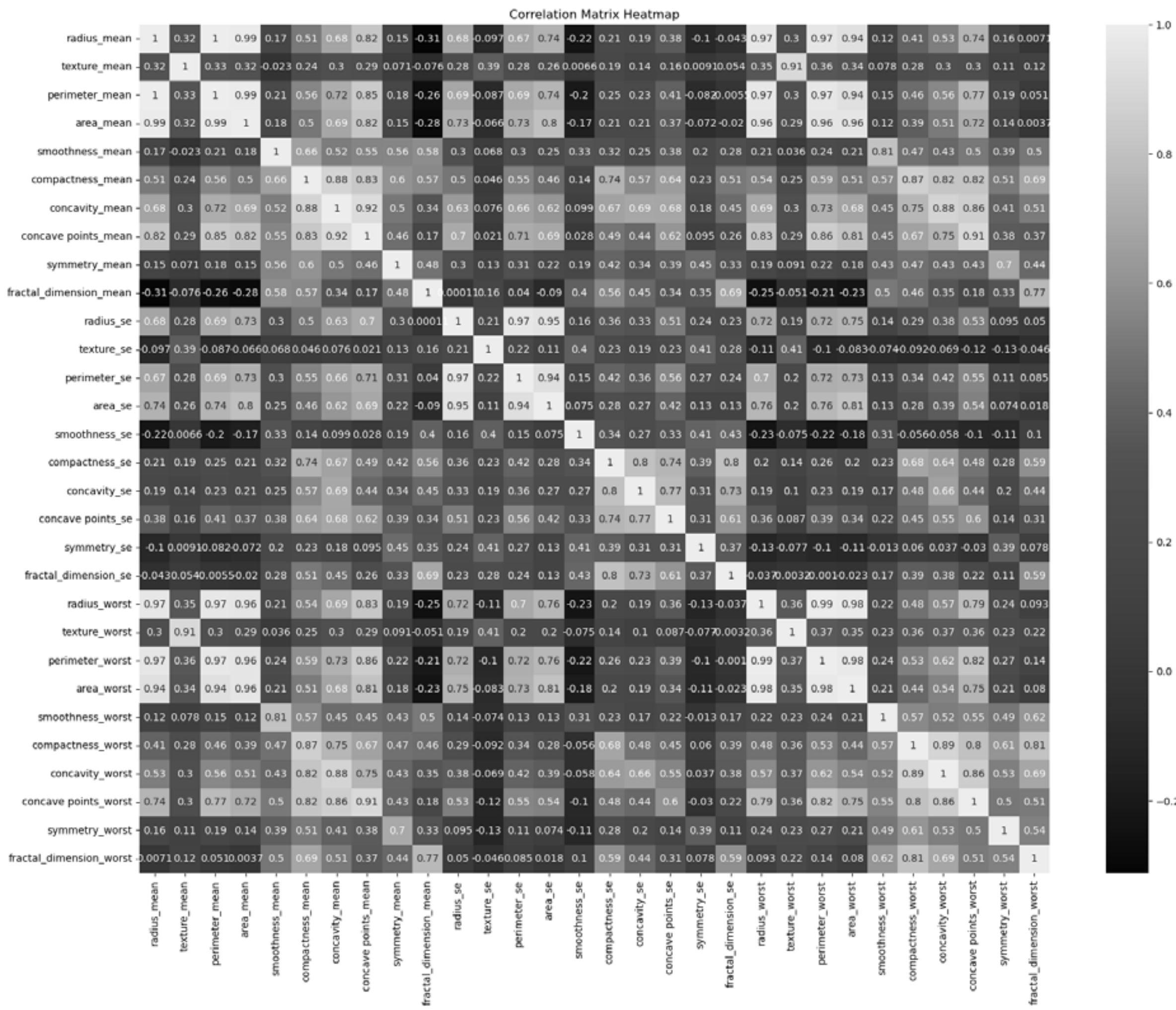
Out[15]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207
area_mean	0.987357	0.321086	0.986507	1.000000	0.177
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584
radius_se	0.679090	0.275869	0.691765	0.732562	0.301
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296
area_se	0.735864	0.259845	0.744983	0.800086	0.246
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238
area_worst	0.941082	0.343546	0.941550	0.959213	0.206
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499

30 rows × 30 columns

In [16]:
```
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True)
plt.title('Correlation Matrix Heatmap')
plt.show
```

Out[16]: `<function matplotlib.pyplot.show(close=None, block=None)>`



TRAINING THE MODEL

Training Process

Now we will splits features(x) and target(y)

In [17]:
```
x = df.iloc[:,1:]
x
```

Out[17]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_m
0	17.99	10.38	122.80	1001.0	0.11840	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.1
...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.0

569 rows × 30 columns

In [18]:
```
y = df.iloc[:,0]
y
```

Out[18]:
```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

now we will perform Train Test Split

In [19]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state
```

In the above code i have just imported the train_test_split model and applied the model to the dataset in 70:30 ratio so that machine will take learning from 70% of the data and make a testing on 30% of the data and hence make a optimum conclusion

Import Machine Learning Algorithm

In [20]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

In [21]:
```
#now we will assign variable to those algorithms
logreg = LogisticRegression()
knn = KNeighborsClassifier()
dt = DecisionTreeClassifier()
svm = SVC()
```

In [22]:
```
#for accuracy we will import classification report and accuracy score
from sklearn.metrics import classification_report,accuracy_score
```

Build a function mymodel so that the accuracies can be easily predicted

In [28]:
```
def mymodel(model):
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    print('Accuracy Score : ',accuracy_score(y_test,y_pred))
    print(classification_report(y_test,y_pred))

    train = model.score(X_train,y_train)
    test = model.score(X_test,y_test)
    print('Training Performance : ',train)
    print('Testing Performance : ',test)
```

Applying different algorithm on the class function to check accuracy of each and finding the best and most accurate algorithm for prediction

Logistic Regression

In [30]:
```
mymodel(logreg)
```

```
Accuracy Score :  0.9707602339181286
              precision    recall  f1-score   support

           B       0.96      0.99      0.98       108
           M       0.98      0.94      0.96        63

    accuracy                           0.97       171
   macro avg       0.97      0.96      0.97       171
weighted avg       0.97      0.97      0.97       171

Training Performance :  0.9396984924623115
Testing Performance :  0.9707602339181286
```

KNeighbors Classifier

In [31]:
```
mymodel(knn)
```

```
Accuracy Score :  0.9590643274853801
              precision    recall  f1-score   support

           B       0.95      0.99      0.97       108
           M       0.98      0.90      0.94        63

    accuracy                           0.96       171
   macro avg       0.96      0.95      0.96       171
weighted avg       0.96      0.96      0.96       171

Training Performance :  0.9221105527638191
Testing Performance :  0.9590643274853801
```

Support Vector Machine (SVM)

In [32]:
```
mymodel(svm)
```

```
Accuracy Score :  0.935672514619883
              precision    recall  f1-score   support

           B       0.91      1.00      0.95       108
           M       1.00      0.83      0.90        63

    accuracy                           0.94       171
   macro avg       0.95      0.91      0.93       171
weighted avg       0.94      0.94      0.93       171

Training Performance :  0.8994974874371859
Testing Performance :  0.935672514619883
```

Decision Tree Classifier

In [33]:
```
mymodel(dt)
```

```
Accuracy Score :  0.9298245614035088
              precision    recall  f1-score   support

           B       0.97      0.92      0.94       108
           M       0.87      0.95      0.91        63

    accuracy                           0.93       171
   macro avg       0.92      0.93      0.93       171
weighted avg       0.93      0.93      0.93       171

Training Performance :  1.0
Testing Performance :  0.9298245614035088
```

Hypertunning

In [35]:
```
for i in range(1,51):
    dt1 = DecisionTreeClassifier(max_depth=i)
    dt1.fit(X_train,y_train)
    y_pred = dt1.predict(X_test)
    ac = accuracy_score(y_test,y_pred)
    print(f'max depth={i}    accuracy = {ac}')
```

```
max depth=1    accuracy = 0.8947368421052632
max depth=2    accuracy = 0.9298245614035088
max depth=3    accuracy = 0.9590643274853801
max depth=4    accuracy = 0.9532163742690059
max depth=5    accuracy = 0.9532163742690059
max depth=6    accuracy = 0.9415204678362573
max depth=7    accuracy = 0.9298245614035088
max depth=8    accuracy = 0.9298245614035088
max depth=9    accuracy = 0.9181286549707602
max depth=10    accuracy = 0.9298245614035088
max depth=11    accuracy = 0.9415204678362573
max depth=12    accuracy = 0.9239766081871345
max depth=13    accuracy = 0.9298245614035088
max depth=14    accuracy = 0.9298245614035088
max depth=15    accuracy = 0.9239766081871345
max depth=16    accuracy = 0.9298245614035088
max depth=17    accuracy = 0.935672514619883
max depth=18    accuracy = 0.935672514619883
max depth=19    accuracy = 0.9415204678362573
max depth=20    accuracy = 0.9181286549707602
max depth=21    accuracy = 0.935672514619883
max depth=22    accuracy = 0.9239766081871345
max depth=23    accuracy = 0.9298245614035088
max depth=24    accuracy = 0.9298245614035088
max depth=25    accuracy = 0.9239766081871345
max depth=26    accuracy = 0.9298245614035088
max depth=27    accuracy = 0.9239766081871345
max depth=28    accuracy = 0.9298245614035088
max depth=29    accuracy = 0.9298245614035088
max depth=30    accuracy = 0.935672514619883
max depth=31    accuracy = 0.935672514619883
max depth=32    accuracy = 0.9415204678362573
max depth=33    accuracy = 0.9122807017543859
max depth=34    accuracy = 0.9298245614035088
max depth=35    accuracy = 0.9298245614035088
max depth=36    accuracy = 0.9239766081871345
max depth=37    accuracy = 0.935672514619883
max depth=38    accuracy = 0.935672514619883
max depth=39    accuracy = 0.9298245614035088
max depth=40    accuracy = 0.935672514619883
max depth=41    accuracy = 0.9239766081871345
max depth=42    accuracy = 0.9239766081871345
max depth=43    accuracy = 0.935672514619883
max depth=44    accuracy = 0.9181286549707602
max depth=45    accuracy = 0.9415204678362573
max depth=46    accuracy = 0.9239766081871345
max depth=47    accuracy = 0.935672514619883
max depth=48    accuracy = 0.935672514619883
max depth=49    accuracy = 0.935672514619883
max depth=50    accuracy = 0.9298245614035088
```

In [37]:
```
for i in range(2,51):
    dt1 = DecisionTreeClassifier(min_samples_split=i)
    dt1.fit(X_train,y_train)
    y_pred = dt1.predict(X_test)
    ac = accuracy_score(y_test,y_pred)
    print(f'min sample split={i}    accuracy = {ac}')
```

```
min sample split=2    accuracy = 0.9415204678362573
min sample split=3    accuracy = 0.9239766081871345
min sample split=4    accuracy = 0.9239766081871345
min sample split=5    accuracy = 0.9064327485380117
min sample split=6    accuracy = 0.9415204678362573
min sample split=7    accuracy = 0.9239766081871345
min sample split=8    accuracy = 0.9298245614035088
min sample split=9    accuracy = 0.935672514619883
min sample split=10    accuracy = 0.9181286549707602
min sample split=11    accuracy = 0.935672514619883
min sample split=12    accuracy = 0.935672514619883
min sample split=13    accuracy = 0.935672514619883
min sample split=14    accuracy = 0.9590643274853801
min sample split=15    accuracy = 0.935672514619883
min sample split=16    accuracy = 0.9298245614035088
min sample split=17    accuracy = 0.9298245614035088
min sample split=18    accuracy = 0.9298245614035088
min sample split=19    accuracy = 0.935672514619883
min sample split=20    accuracy = 0.935672514619883
min sample split=21    accuracy = 0.9298245614035088
min sample split=22    accuracy = 0.935672514619883
min sample split=23    accuracy = 0.9298245614035088
min sample split=24    accuracy = 0.935672514619883
min sample split=25    accuracy = 0.935672514619883
min sample split=26    accuracy = 0.935672514619883
min sample split=27    accuracy = 0.935672514619883
min sample split=28    accuracy = 0.935672514619883
min sample split=29    accuracy = 0.935672514619883
min sample split=30    accuracy = 0.935672514619883
min sample split=31    accuracy = 0.9298245614035088
min sample split=32    accuracy = 0.935672514619883
min sample split=33    accuracy = 0.9298245614035088
min sample split=34    accuracy = 0.935672514619883
min sample split=35    accuracy = 0.9298245614035088
min sample split=36    accuracy = 0.9298245614035088
min sample split=37    accuracy = 0.935672514619883
min sample split=38    accuracy = 0.935672514619883
min sample split=39    accuracy = 0.9298245614035088
min sample split=40    accuracy = 0.935672514619883
min sample split=41    accuracy = 0.935672514619883
min sample split=42    accuracy = 0.935672514619883
min sample split=43    accuracy = 0.935672514619883
min sample split=44    accuracy = 0.9298245614035088
min sample split=45    accuracy = 0.9298245614035088
min sample split=46    accuracy = 0.935672514619883
min sample split=47    accuracy = 0.935672514619883
min sample split=48    accuracy = 0.9298245614035088
min sample split=49    accuracy = 0.9298245614035088
min sample split=50    accuracy = 0.9298245614035088
```

In [38]:
```
for i in range(1,51):
    dt1 = DecisionTreeClassifier(min_samples_leaf=i)
    dt1.fit(X_train,y_train)
    y_pred = dt1.predict(X_test)
    ac = accuracy_score(y_test,y_pred)
    print(f'min sample leaf={i}    accuracy = {ac}')
```

```
min sample leaf=1    accuracy = 0.935672514619883
min sample leaf=2    accuracy = 0.9649122807017544
min sample leaf=3    accuracy = 0.9649122807017544
min sample leaf=4    accuracy = 0.9649122807017544
min sample leaf=5    accuracy = 0.9590643274853801
min sample leaf=6    accuracy = 0.9649122807017544
min sample leaf=7    accuracy = 0.9532163742690059
min sample leaf=8    accuracy = 0.9415204678362573
min sample leaf=9    accuracy = 0.9415204678362573
min sample leaf=10    accuracy = 0.9415204678362573
min sample leaf=11    accuracy = 0.9415204678362573
min sample leaf=12    accuracy = 0.9415204678362573
min sample leaf=13    accuracy = 0.9415204678362573
min sample leaf=14    accuracy = 0.9415204678362573
min sample leaf=15    accuracy = 0.9649122807017544
min sample leaf=16    accuracy = 0.9415204678362573
min sample leaf=17    accuracy = 0.9415204678362573
min sample leaf=18    accuracy = 0.9415204678362573
min sample leaf=19    accuracy = 0.9415204678362573
min sample leaf=20    accuracy = 0.8947368421052632
min sample leaf=21    accuracy = 0.8947368421052632
min sample leaf=22    accuracy = 0.8947368421052632
min sample leaf=23    accuracy = 0.8947368421052632
min sample leaf=24    accuracy = 0.8947368421052632
min sample leaf=25    accuracy = 0.8947368421052632
min sample leaf=26    accuracy = 0.8947368421052632
min sample leaf=27    accuracy = 0.8947368421052632
min sample leaf=28    accuracy = 0.8947368421052632
min sample leaf=29    accuracy = 0.8947368421052632
min sample leaf=30    accuracy = 0.8947368421052632
min sample leaf=31    accuracy = 0.8947368421052632
min sample leaf=32    accuracy = 0.8947368421052632
min sample leaf=33    accuracy = 0.8947368421052632
min sample leaf=34    accuracy = 0.8947368421052632
min sample leaf=35    accuracy = 0.8947368421052632
min sample leaf=36    accuracy = 0.8947368421052632
min sample leaf=37    accuracy = 0.8947368421052632
min sample leaf=38    accuracy = 0.8947368421052632
min sample leaf=39    accuracy = 0.8947368421052632
min sample leaf=40    accuracy = 0.8947368421052632
min sample leaf=41    accuracy = 0.8947368421052632
min sample leaf=42    accuracy = 0.8947368421052632
min sample leaf=43    accuracy = 0.8947368421052632
min sample leaf=44    accuracy = 0.8947368421052632
min sample leaf=45    accuracy = 0.8947368421052632
min sample leaf=46    accuracy = 0.8947368421052632
min sample leaf=47    accuracy = 0.8947368421052632
min sample leaf=48    accuracy = 0.8947368421052632
min sample leaf=49    accuracy = 0.8947368421052632
min sample leaf=50    accuracy = 0.8947368421052632
```

In [39]:

```
dt = DecisionTreeClassifier(criterion='entropy',max_depth=3,min_samples_split=2
```

In [40]:
```
mymodel(dt)
```

```
Accuracy Score :  0.9649122807017544
              precision    recall  f1-score   support

           B       0.96      0.98      0.97       108
           M       0.97      0.94      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

Training Performance :  0.9798994974874372
Testing Performance :  0.9649122807017544
```

BEST MODEL

In [42]:
```
mymodel(dt)
```

```
Accuracy Score :  0.9649122807017544
              precision    recall  f1-score   support

           B       0.96      0.98      0.97       108
           M       0.97      0.94      0.95        63

    accuracy                           0.96       171
   macro avg       0.97      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

Training Performance :  0.9798994974874372
Testing Performance :  0.9649122807017544
```

CONCLUSION

- The proposed machine-learning approaches could predict breast cancer as the early detection of this disease could help slow down the progress of the disease and reduce the mortality rate through appropriate therapeutic interventions at the right time.
- In this project we are explored the use of different machine learning algorithms like Logistic regression, support vector machine, K-neighbors,decision tree to predict whether the breast cancer type is Maligant(cancerous) or Bening(non-cancerous).
- Through the our experimentation,we found that the Decision Tree algorithm performed the best highest accuracy score.so this model can be used by healthcare team to identify the potential risk of breast cancer.

In [ ]:
```

```