Project Name: Breast Cancer Classifier

Goal: To predict the type of tumour of patient.

Dataset: Kaggle Breast Cancer Dataset (https://www.kaggle.com/datasets/yasserh/breast-cancer-dataset)

About the Project: Cancer diagnosis, particularly distinguishing between malignant (cancerous) and benign (non-cancerous) tumors, is crucial for effective treatment and patient outcomes. Traditional diagnostic methods, such as imaging and biopsies, are often invasive and costly.

Our goal is to support medical practitioners in improving patient outcomes and enhancing overall community health. This tool will streamline the diagnostic process, potentially reducing the need for invasive procedures and lowering healthcare costs.

This project aims to develop a Machine Learning model to predict whether a patient has a malignant or benign tumor using patient data such as tumor size, cell shape, and other relevant medical information. By leveraging historical data and advanced algorithms, our model seeks to assist healthcare professionals in making accurate and timely diagnoses.lth.

Section 1: Collecting the data

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

In [2]:
```
data = pd.read_csv(r"C:\Users\goura\Desktop\Data Science\Datasets\Breast Cancer.
data
```

Out[2]:

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

569 rows × 32 columns

Section 2: Data Manipulation/Cleaning

Lets figure out the data types of columns in our dataset

In [3]:
```
columns = [ i for i in data.columns ]
numeric = data.select_dtypes(include=['float64', 'int64']).columns.tolist()
categoric = [ i for i in data.columns if i not in numeric ]
```

In [4]:
```
categoric
```

Out[4]:
```
['diagnosis']
```

Looks like we've got one categorical column in our dataset. In such situations, we can either

- Drop the column if not relevant
- Encode the column values

Since Diagnosis is an important parameter (the parameter we'll be predicting), it is should be encoded instead of dropping

In [5]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data["diagnosis"] = encoder.fit_transform(data["diagnosis"])
data
```

Out[5]:

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

569 rows × 32 columns

We can observe the following:
Diagnosis: 1 = M = Malignant
Diagnosis: 0 = B = Benign

In [6]:
```
d={
    1:"Malignant",
    0:"Benign"
}
```

We'd also not need the 'ID' column, since it has no actual correlation to the type of tumout.
So we will remove it from the dataset so that it doesn't disturb the prediction quality.

In [7]:
```
features = [ i for i in columns if i not in ('diagnosis','id') ]
label = ['diagnosis']
X = data[features]
Y = data[label]
X
```

Out[7]:

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

569 rows × 30 columns

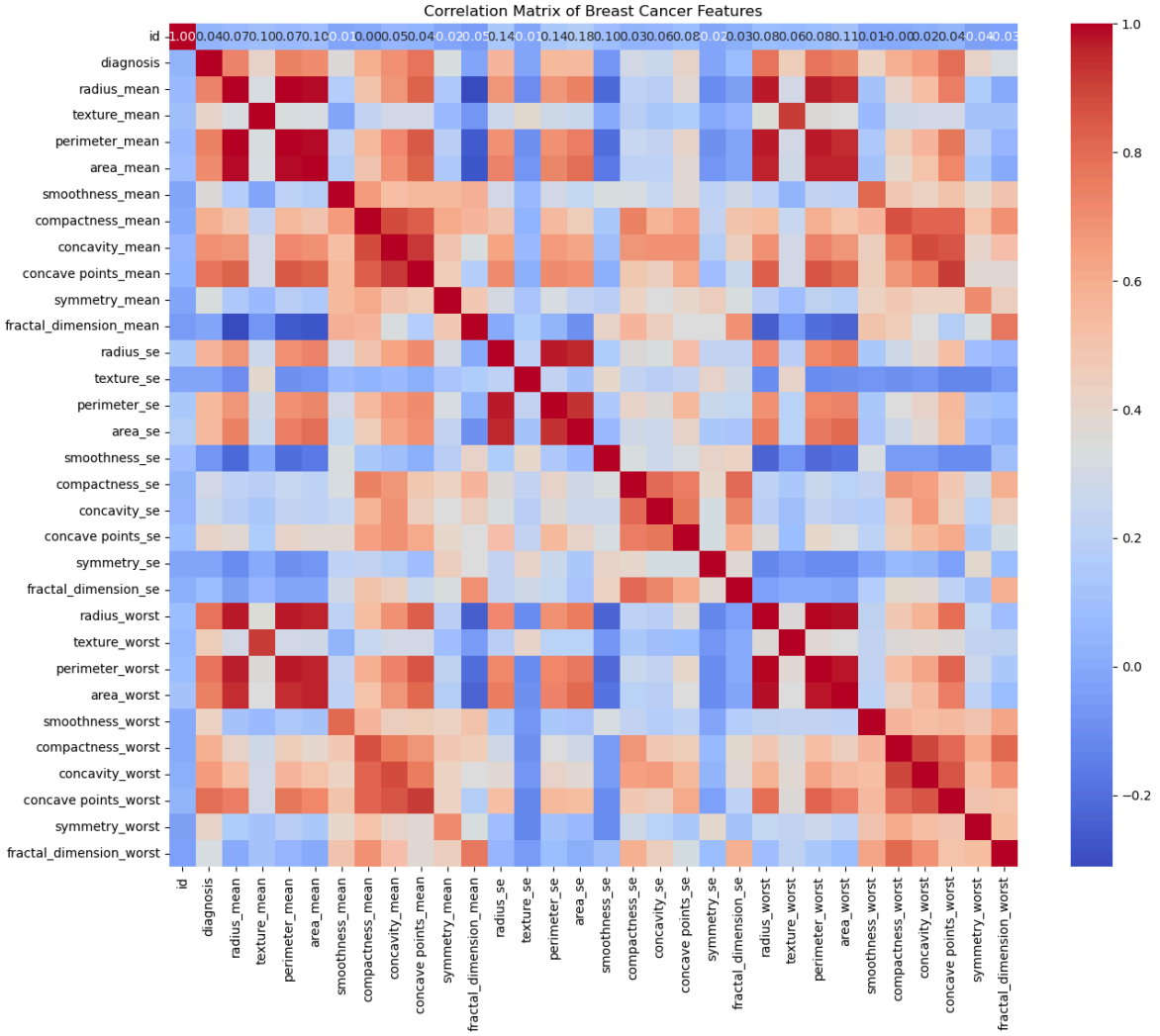
In [8]:
```
corr_matrix = data.corr()
corr_matrix
```

Out[8]:

	id	diagnosis	radius_mean	texture_mean	perimeter_me
id	1.000000	0.039769	0.074626	0.099770	0.0731
diagnosis	0.039769	1.000000	0.730029	0.415185	0.7426
radius_mean	0.074626	0.730029	1.000000	0.323782	0.9978
texture_mean	0.099770	0.415185	0.323782	1.000000	0.3295
perimeter_mean	0.073159	0.742636	0.997855	0.329533	1.0000
area_mean	0.096893	0.708984	0.987357	0.321086	0.9865
smoothness_mean	-0.012968	0.358560	0.170581	-0.023389	0.2072
compactness_mean	0.000096	0.596534	0.506124	0.236702	0.5569
concavity_mean	0.050080	0.696360	0.676764	0.302418	0.7161
concave points_mean	0.044158	0.776614	0.822529	0.293464	0.8509
symmetry_mean	-0.022114	0.330499	0.147741	0.071401	0.1830
fractal_dimension_mean	-0.052511	-0.012838	-0.311631	-0.076437	-0.2614
radius_se	0.143048	0.567134	0.679090	0.275869	0.6917
texture_se	-0.007526	-0.008303	-0.097317	0.386358	-0.0867
perimeter_se	0.137331	0.556141	0.674172	0.281673	0.6931
area_se	0.177742	0.548236	0.735864	0.259845	0.7449
smoothness_se	0.096781	-0.067016	-0.222600	0.006614	-0.2026
compactness_se	0.033961	0.292999	0.206000	0.191975	0.2507
concavity_se	0.055239	0.253730	0.194204	0.143293	0.2280
concave points_se	0.078768	0.408042	0.376169	0.163851	0.4072
symmetry_se	-0.017306	-0.006522	-0.104321	0.009127	-0.0816
fractal_dimension_se	0.025725	0.077972	-0.042641	0.054458	-0.0055
radius_worst	0.082405	0.776454	0.969539	0.352573	0.9694
texture_worst	0.064720	0.456903	0.297008	0.912045	0.3030
perimeter_worst	0.079986	0.782914	0.965137	0.358040	0.9703
area_worst	0.107187	0.733825	0.941082	0.343546	0.9415
smoothness_worst	0.010338	0.421465	0.119616	0.077503	0.1505
compactness_worst	-0.002968	0.590998	0.413463	0.277830	0.4557
concavity_worst	0.023203	0.659610	0.526911	0.301025	0.5638
concave points_worst	0.035174	0.793566	0.744214	0.295316	0.7712
symmetry_worst	-0.044224	0.416294	0.163953	0.105008	0.1891
fractal_dimension_worst	-0.029866	0.323872	0.007066	0.119205	0.0510

32 rows × 32 columns

In [9]:
```
plt.figure(figsize=(15, 12))
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Matrix of Breast Cancer Features')
plt.show()
```



As it can be seen, all of the features except 'ID' have a significant correlation to 'diagnosis'.
So, we retain all such features.

In [10]:
```
for feature in features:
    null_count = data[feature].isnull().sum()
    print(feature, null_count, (null_count/569)*100)
```

```
radius_mean 0 0.0
texture_mean 0 0.0
perimeter_mean 0 0.0
area_mean 0 0.0
smoothness_mean 0 0.0
compactness_mean 0 0.0
concavity_mean 0 0.0
concave points_mean 0 0.0
symmetry_mean 0 0.0
fractal_dimension_mean 0 0.0
radius_se 0 0.0
texture_se 0 0.0
perimeter_se 0 0.0
area_se 0 0.0
smoothness_se 0 0.0
compactness_se 0 0.0
concavity_se 0 0.0
concave points_se 0 0.0
symmetry_se 0 0.0
fractal_dimension_se 0 0.0
radius_worst 0 0.0
texture_worst 0 0.0
perimeter_worst 0 0.0
area_worst 0 0.0
smoothness_worst 0 0.0
compactness_worst 0 0.0
concavity_worst 0 0.0
concave points_worst 0 0.0
symmetry_worst 0 0.0
fractal_dimension_worst 0 0.0
```

Since our dataset has no Null values, we dont need to worry about that either.

Now that our data is ready for use, we will split it into 4 parts, based on-

- Features, Label [to distinguish the columns used to make predictions and the column to be predicted]
- Train, Test [to distinguish data we will train our model on and data we will test the trained model on]

We will use the **Stratified Shuffle Split**, due to such splits being a representatinve of the entire dataset.

In [11]:
```
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size = 0.15, random_state=42)
for train_index,test_index in split.split(data,data[label]):
    train_set = data.iloc[train_index]
    test_set = data.iloc[test_index]
```

In [12]:
```
train_set
```

Out[12]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sm
357	901028	0	13.870	16.21	88.52	593.7	
352	899987	1	25.730	17.46	174.20	2010.0	
224	8813129	0	13.270	17.02	84.55	546.4	
467	9113514	0	9.668	18.10	61.06	286.3	
2	84300903	1	19.690	21.25	130.00	1203.0	
...	...	...	...	...	...	...	
415	905686	0	11.890	21.17	76.39	433.8	
458	9112594	0	13.000	25.13	82.61	520.2	
476	911654	0	14.200	20.53	92.41	618.4	
72	859717	1	17.200	24.52	114.20	929.4	
265	88995002	1	20.730	31.12	135.70	1419.0	

483 rows × 32 columns

In [13]:
```
X_train = train_set[features]
Y_train = train_set[label]
X_test = test_set[features]
Y_test = test_set[label]
```

Lets have a look at alll the created parts of the dataset.

- X (all rows of feature columns),
- Y (all rows of label column),
- X_train (training rows of feature columns),
- X_test (testing rows of feature columns),
- Y_train (training rows of label column),
- Y_test (testing rows of label column)

Section 3: Model Selection & Training

We proceed with the use of KNN algorhithm for this classification problem.

We set up an algorhithm to give us the optimal number of neighbors for the KNN model, by recognising the least MSE for all number of neighbours in the range [1,51,2]

We use cross-validation as a measure for testing our model.

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

In [15]:
```
neighbors = []
cv_scores = []
```

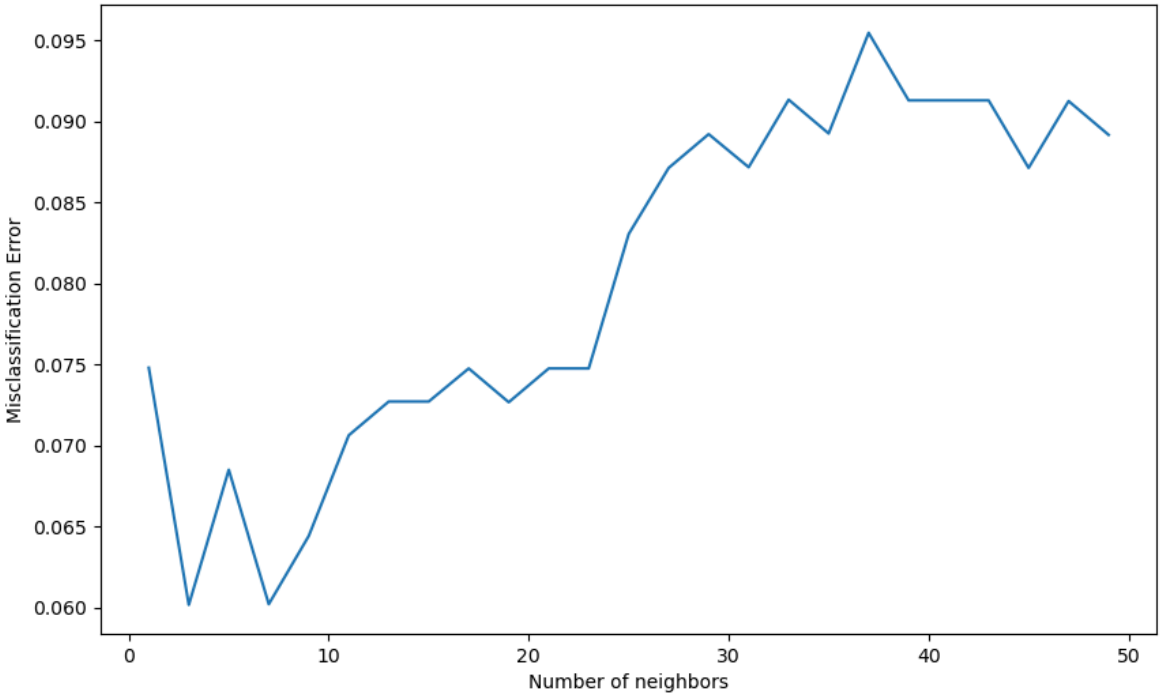
```
Y_train = Y_train.values.ravel()
from sklearn.model_selection import cross_val_score
for k in range(1, 51, 2):
    neighbors.append(k)
    knn = KNeighborsClassifier(n_neighbors = k)
    scores = cross_val_score(
        knn, X_train, Y_train, cv = 10, scoring = 'accuracy')
    cv_scores.append(scores.mean())
```

In [16]:
```
MSE = [1-x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
print('The optimal number of neighbors is % d ' % optimal_k)

plt.figure(figsize = (10, 6))
plt.plot(neighbors, MSE)
plt.xlabel('Number of neighbors')
plt.ylabel('Misclassification Error')
plt.show()
```

```
The optimal number of neighbors is  3
```



As we can see, the optimum number of neighbours for the dataset in question is 3.

So, we prepare our model appropriately.

In [17]:
```
knn_model = KNeighborsClassifier(n_neighbors = 3)
knn_model.fit(X_train,Y_train)
```

Out[17]:
```
KNeighborsClassifier(n_neighbors=3)
```

Lets check for the accuracy of the model using **Accuracy_Score**

In [18]:
```
pred1 = knn_model.predict(X_test)
accuracy_score(Y_test,pred1)
```

Out[18]: 0.9069767441860465

We have acheived an accuracy of 91%.
Our next step is to make this program user interactive, such that the program can provide the disease name as the output for some set of symptoms as input.

Section 4: Creating User-Model interaction space

Since our models make numerical predictions (due to being encoded earlier), we need to convert these numerical output to meaningful output i.e. type of the tumour

Below is a function to receive the values for required attributes, and thus make appropriate changes to make the predictions.

In [19]:
```
Y = Y.values.ravel()
knn_model.fit(X,Y)
```

Out[19]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [21]:
```
c=0
ques = features
while True:
    if c==0:
        print("Do you want to make any predictions?")
    elif c>0:
        print("Do you want to make any more predictions?")
    print("Enter 1 if Yes, else 0")
    a = int(input("Choice:"))
    if a!=0 and a!=1:
        print("Invalid Choice")
    elif a==0:
        print("Choice is No (0)")
        print("Exit Program")
        if c>0:
            print("Thank you for using our services.")
        break
    elif a==1:
        print("Choice is Yes (1)")
        lst = []
        for i in ques:
            lst.append(float(input(f"Enter value of {i}: ")))
        df = pd.DataFrame([lst], columns=ques)
        knn_pred = knn_model.predict(df)
        type = d[knn_pred[0]]
        print("Reported Parameters: ",lst)
        print("The tumour type is predicted to be:",type)
        c+=1
```

```
Do you want to make any predictions?
Enter 1 if Yes, else 0
Choice is Yes (1)
```

```
Reported Parameters:  [3.0, 3.0, 3.0, 3.0, 33.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.
0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.
0, 3.0, 3.0]
The tumour type is predicted to be: Benign
Do you want to make any more predictions?
Enter 1 if Yes, else 0
Choice is No (0)
Exit Program
Thank you for using our services.
```

Since all our models have been defined, and set to actively accept provided data and provide predictions,
our program is ready to use and the project is complete

Conclusion

This marks the end of our program.

We predicted the type of tumour of patients based on various attributes related to the tumour from, by using K Nearest Neiighbors algorhithm to find the final prediction. Our model predictictions have yet provided us with 91% accuracy, we can conclude that our model will provide us with predictions of decent accuracy on unknown & similar datasets.