Breast Cancer Detection (Kaggle Dataset)

Name: Prashanth B

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image and the 3-dimensional space is that described in [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

This database is also available through the UW CS ftp server: ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Import Relevant Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Data Preprocessing

```
data = pd.read_csv('data_breast_cancer.csv')
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean
0	842302	M	17.99	10.38	122.80
1	842517	M	20.57	17.77	132.90
2	84300903	M	19.69	21.25	130.00
3	84348301	M	11.42	20.38	77.58
4	84358402	M	20.29	14.34	135.10

5 rows × 33 columns

Data Exploration

```
data.shape
```

```
(569, 33)
```

```
data.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
```

```
data.select_dtypes(include='object').columns
```

```
Index(['diagnosis'], dtype='object')
```

```
len(data.select_dtypes(include='object').columns)
```

```
1
```

```
data.select_dtypes(include=['float64','int64']).columns
```

```
Index(['id', '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')
```

```
len(data.select_dtypes(include=['float64','int64']).columns)
```

```
32
```

```
#statistical summary
data.describe()
```

	id	radius_mean	texture_mean	perimeter_mean	a
count	5.690000e+02	569.000000	569.000000	569.000000	56
mean	3.037183e+07	14.127292	19.289649	91.969033	65
std	1.250206e+08	3.524049	4.301036	24.298981	35
min	8.670000e+03	6.981000	9.710000	43.790000	14
25%	8.692180e+05	11.700000	16.170000	75.170000	42
50%	9.060240e+05	13.370000	18.840000	86.240000	55
75%	8.813129e+06	15.780000	21.800000	104.100000	78
max	9.113205e+08	28.110000	39.280000	188.500000	250

8 rows × 32 columns

```
data.columns
```

```
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')
```

Dealing with missing values

```
data.isnull().values.any()
```

```
True
```

```
data.isnull().values.sum()
```

```
569
```

```
data.columns[data.isnull().any()]
```

```
Index(['Unnamed: 32'], dtype='object')
```

```
len(data.columns[data.isnull().any()])
```

```
1
```

```
data['Unnamed: 32'].count()
```

```
0
```

```
data = data.drop(columns = 'Unnamed: 32')
```

```
data.shape
```

```
(569, 32)
```

```
data.isnull().values.any()
```

```
False
```

Dealing with categorical data

```
data.select_dtypes(include='object').columns
```

```
Index(['diagnosis'], dtype='object')
```

```
data['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

```
data['diagnosis'].nunique()
```

```
2
```

```
#One hot encoding
data = pd.get_dummies(data=data, drop_first=True)
```

```
data.head()
```

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

5 rows × 32 columns

```
# B Values
(data.diagnosis_M == 0).sum()
```

```
357
```

```
# M Values
(data.diagnosis_M == 1).sum()
```

```
212
```

Correlation matrix and Heatmap

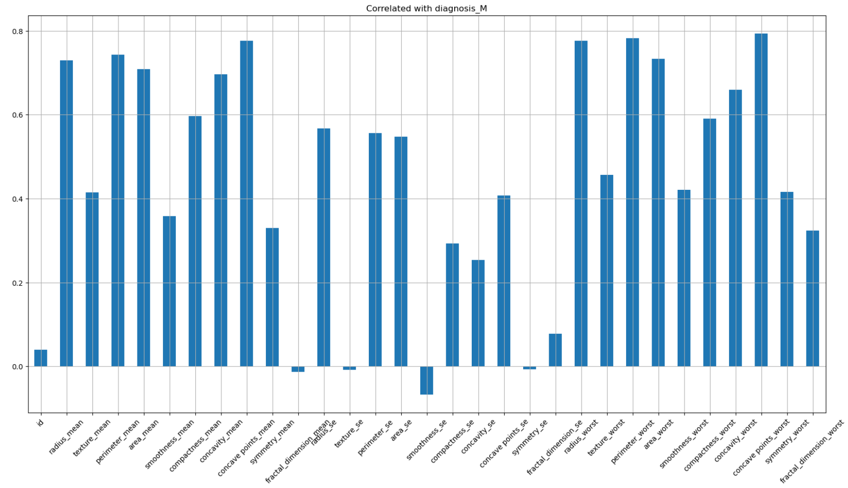
```
data_1 = data.drop(columns='diagnosis_M')
```

```
data_1.head()
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean
0	842302	17.99	10.38	122.80	1001.0
1	842517	20.57	17.77	132.90	1326.0
2	84300903	19.69	21.25	130.00	1203.0

```
data_1.corrwith(data['diagnosis_M']).plot.bar(
    figsize=(20,10), title='Correlated with diagnosis_M', rot=45, grid=True
)
```

```
<Axes: title={'center': 'Correlated with diagnosis_M'}>
```

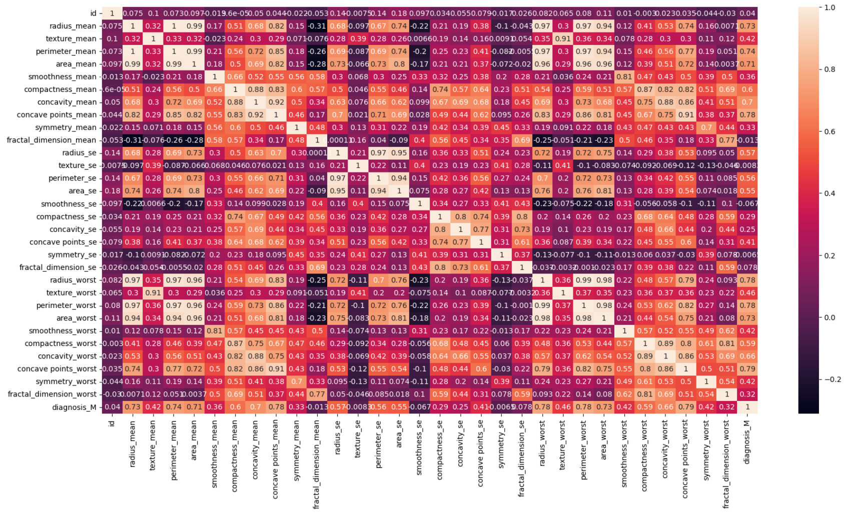



```
#Correlation matrix
corr = data.corr()
corr
```

	id	radius_mean	texture_mean	perime
id	1.000000	0.074626	0.099770	
radius_mean	0.074626	1.000000	0.323782	
texture_mean	0.099770	0.323782	1.000000	
perimeter_mean	0.073159	0.997855	0.329533	
area_mean	0.096893	0.987357	0.321086	
smoothness_mean	-0.012968	0.170581	-0.023389	
compactness_mean	0.000096	0.506124	0.236702	
concavity_mean	0.050080	0.676764	0.302418	
concave points_mean	0.044158	0.822529	0.293464	
symmetry_mean	-0.022114	0.147741	0.071401	
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	
radius_se	0.143048	0.679090	0.275869	
texture_se	-0.007526	-0.097317	0.386358	
perimeter_se	0.137331	0.674172	0.281673	
area_se	0.177742	0.735864	0.259845	
smoothness_se	0.096781	-0.222600	0.006614	
compactness_se	0.033961	0.206000	0.191975	
concavity_se	0.055239	0.194204	0.143293	
concave points_se	0.078768	0.376169	0.163851	
symmetry_se	-0.017306	-0.104321	0.009127	
fractal_dimension_se	0.025725	-0.042641	0.054458	
radius_worst	0.082405	0.969539	0.352573	
texture_worst	0.064720	0.297008	0.912045	
perimeter_worst	0.079986	0.965137	0.358040	
area_worst	0.107187	0.941082	0.343546	
smoothness_worst	0.010338	0.119616	0.077503	
compactness_worst	-0.002968	0.413463	0.277830	
concavity_worst	0.023203	0.526911	0.301025	

```
#Heatmap
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True)
```

```
<Axes: >
```



Splitting the data to train and test

```
data.head()
```

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

5 rows × 32 columns

```
#Matrix of features or independent variables
x = data.iloc[:, 1:-1].values
x.shape
```

```
(569, 30)
```

```
# target variable / dependent variable
y = data.iloc[:, -1].values
y.shape
```

```
(569,)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
x_train.shape
```

```
(455, 30)
```

```
x_test.shape
```

```
(114, 30)
```

```
y_train.shape
```

```
(455,)
```

```
y_test.shape
```

```
(114,)
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

```
x_train
```

```
array([[-1.15036482, -0.39064196, -1.12855021, ..., -0.75798367,
        -0.01614761, -0.38503402],
       [-0.93798972,  0.68051405, -0.94820146, ..., -0.60687023,
         0.09669004, -0.38615797],
       [ 0.574121  , -1.03333557,  0.51394098, ..., -0.02371948,
        -0.20050207, -0.75144254],
       ...,
       [-1.32422924, -0.20048168, -1.31754581, ..., -0.97974953,
        -0.71542314, -0.11978123],
       [-1.24380987, -0.2245526 , -1.28007609, ..., -1.75401433,
        -1.58157125, -1.00601779],
       [-0.73694129,  1.14989702, -0.71226578, ..., -0.27460457,
        -1.25895095,  0.21515662]])
```

```
x_test
```

```
array([[-0.20175604,  0.3290786 , -0.13086754, ...,  1.3893291 ,
         1.08203284,  1.54029664],
       [-0.25555773,  1.46763319, -0.31780437, ..., -0.83369364,
        -0.73131577, -0.87732522],
       [-0.02619262, -0.8407682 , -0.09175081, ..., -0.49483785,
        -1.22080864, -0.92115937],
       ...,
       [ 1.71811488,  0.09318356,  1.7286186 , ...,  1.57630515,
         0.20317063, -0.15406178],
       [ 1.18859296,  0.34352115,  1.19333694, ...,  0.56019755,
         0.26991966, -0.27320074],
       [ 0.26263752, -0.58080224,  0.28459338, ..., -0.19383705,
        -1.15564888,  0.11231497]])
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(random_state=0)
classifier_lr.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=0)
```

```
y_pred = classifier_lr.predict(x_test)
y_pred
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0], dtype=uint8)
```

```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, confusion_matrix
acc = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)

results = pd.DataFrame([['Logistic Regression', acc, f1, prec, rec]],
                       columns=['Model','Accuracy','f1 score','Precision','Recall'])
results
```

	Model	Accuracy	f1 score	Precision	Recall
0	Logistic Regression	0.964912	0.957447	0.957447	0.957447

```
#confusion matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[65  2]
 [ 2 45]]
```

cross validation

```
from sklearn.model_selection import cross_val_score
accuracy1 = cross_val_score(estimator=classifier_lr, X=x_train, y=y_train, cv=10)
print(accuracy1)
```

```
[0.97826087 0.97826087 0.97826087 0.97826087 0.95652174 0.93333333
 1.         1.         0.97777778 1.        ]
```

```
print("Accuracy is {:.2f} %".format(accuracy1.mean()*100))
print("Standard Deviation is {:.2f} %".format(accuracy1.std()*100))
```

```
Accuracy is 97.81 %
Standard Deviation is 1.98 %
```

Thank you