Breast Cancer Wisconsin (Diagnostic) - Logistic Regression Model

Objective of the Project

- The objective is to classify whether the breast cancer is benign or malignant and to achieve this I have used Logistics Regression algorithm.

Brief about Logistics Regression

- Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. These binary outcomes allow straightforward decisions between two alternatives. It allows algorithms used in machine learning applications to classify incoming data based on historical data. As additional relevant data comes in, the algorithms get better at predicting classifications within data sets.
- Logistic regression 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.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

Logistic regression assumptions

- Binary logistic regression requires the dependent variable to be binary and ordinal logistic regression requires the dependent variable to be ordinal.
- It requires the observations to be independent of each other.
- It requires there to be little or no multicollinearity among the independent variables.
- It assumes linearity of independent variables and log odds.
- It requires a large sample size.

Dataset source & brief

- Breast Cancer Wisconsin (Diagnostic) Data Set is taken from Kaggle. Its 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. It has following attributes:
- ID number, Diagnosis (M = malignant, B = benign
- Ten real-valued features are computed for each cell nucleus:
- 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)
- The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features.

Outline

- 1. Import Libraries & Dataset
- 2. Basic information about the dataset
- 3. Data Pre Processing & EDA
- 4. Split data into Dependent and Independent variables & then into Train and test
- 5. Build Logistics Regression model
- 6. Predict the model
- 7. Evaluate the model
- 8. Conclude

Import the required Libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

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

Load the dataset - Breast Cancer (Wisconsin)

In [2]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Sundaram Sir\Dataset\Breast
df.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
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	

5 rows × 32 columns

Check basic information

In [3]:

```
# Shape
df.shape
```

Out[3]:

```
(569, 32)
```

In [4]:

```
#info
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 [5]:

```
#Drop non significant variable
df.drop(['id'],axis=1,inplace=True)
```

Data Pre Processing

In [6]:

```
# Check for duplicate values
df.duplicated().sum()     # no duplicate values
```

Out[6]:

```
0
```

In [7]:

```
# Check for missing values
df.isnull().sum()        # no missing values
```

Out[7]:

```
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 [8]:

```
# Check Imbalance data
df['diagnosis'].value_counts()   # data is balanced
```

Out[8]:

```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [9]:

```
# Label Encoder for 'diagnosis' variable
df['diagnosis']=df['diagnosis'].astype('category')
df['diagnosis']=df['diagnosis'].cat.codes
```

In [10]:

```
# Statistical summary
df.describe().T.style.background_gradient(cmap='Greens')
```

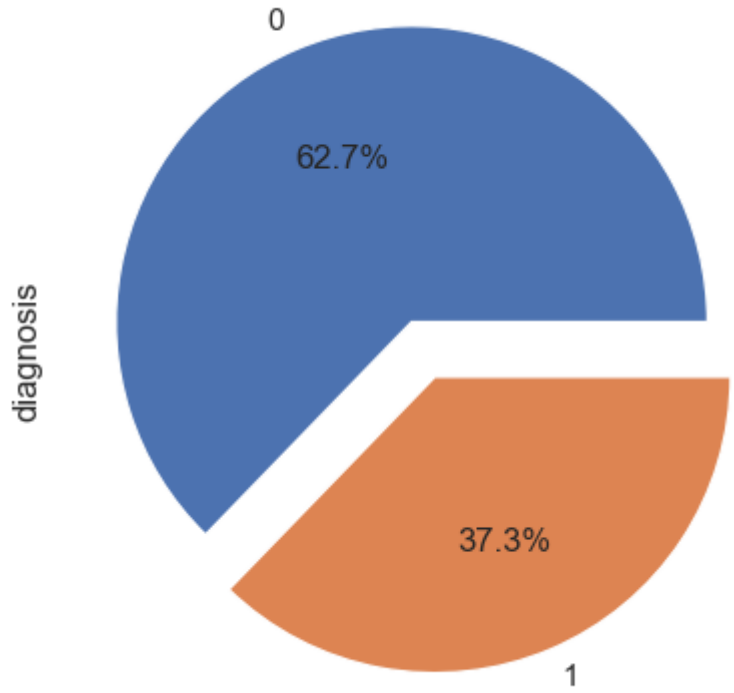
Out[10]:

	count	mean	std	min	25%	
diagnosis	569.000000	0.372583	0.483918	0.000000	0.000000	0.000
radius_mean	569.000000	14.127292	3.524049	6.981000	11.700000	13.370
texture_mean	569.000000	19.289649	4.301036	9.710000	16.170000	18.840
perimeter_mean	569.000000	91.969033	24.298981	43.790000	75.170000	86.240
area_mean	569.000000	654.889104	351.914129	143.500000	420.300000	551.100
smoothness_mean	569.000000	0.096360	0.014064	0.052630	0.086370	0.095
compactness_mean	569.000000	0.104341	0.052813	0.019380	0.064920	0.092
concavity_mean	569.000000	0.088799	0.079720	0.000000	0.029560	0.061
concave points_mean	569.000000	0.048919	0.038803	0.000000	0.020310	0.033
symmetry_mean	569.000000	0.181162	0.027414	0.106000	0.161900	0.179
fractal_dimension_mean	569.000000	0.062798	0.007060	0.049960	0.057700	0.061
radius_se	569.000000	0.405172	0.277313	0.111500	0.232400	0.324
texture_se	569.000000	1.216853	0.551648	0.360200	0.833900	1.108
perimeter_se	569.000000	2.866059	2.021855	0.757000	1.606000	2.287
area_se	569.000000	40.337079	45.491006	6.802000	17.850000	24.530
smoothness_se	569.000000	0.007041	0.003003	0.001713	0.005169	0.006
compactness_se	569.000000	0.025478	0.017908	0.002252	0.013080	0.020
concavity_se	569.000000	0.031894	0.030186	0.000000	0.015090	0.025
concave points_se	569.000000	0.011796	0.006170	0.000000	0.007638	0.010
symmetry_se	569.000000	0.020542	0.008266	0.007882	0.015160	0.018
fractal_dimension_se	569.000000	0.003795	0.002646	0.000895	0.002248	0.003
radius_worst	569.000000	16.269190	4.833242	7.930000	13.010000	14.970
texture_worst	569.000000	25.677223	6.146258	12.020000	21.080000	25.410
perimeter_worst	569.000000	107.261213	33.602542	50.410000	84.110000	97.660
area_worst	569.000000	880.583128	569.356993	185.200000	515.300000	686.500
smoothness_worst	569.000000	0.132369	0.022832	0.071170	0.116600	0.131
compactness_worst	569.000000	0.254265	0.157336	0.027290	0.147200	0.211
concavity_worst	569.000000	0.272188	0.208624	0.000000	0.114500	0.226
concave points_worst	569.000000	0.114606	0.065732	0.000000	0.064930	0.099
symmetry_worst	569.000000	0.290076	0.061867	0.156500	0.250400	0.282
fractal_dimension_worst	569.000000	0.083946	0.018061	0.055040	0.071460	0.080

Exploratory Data Analysis

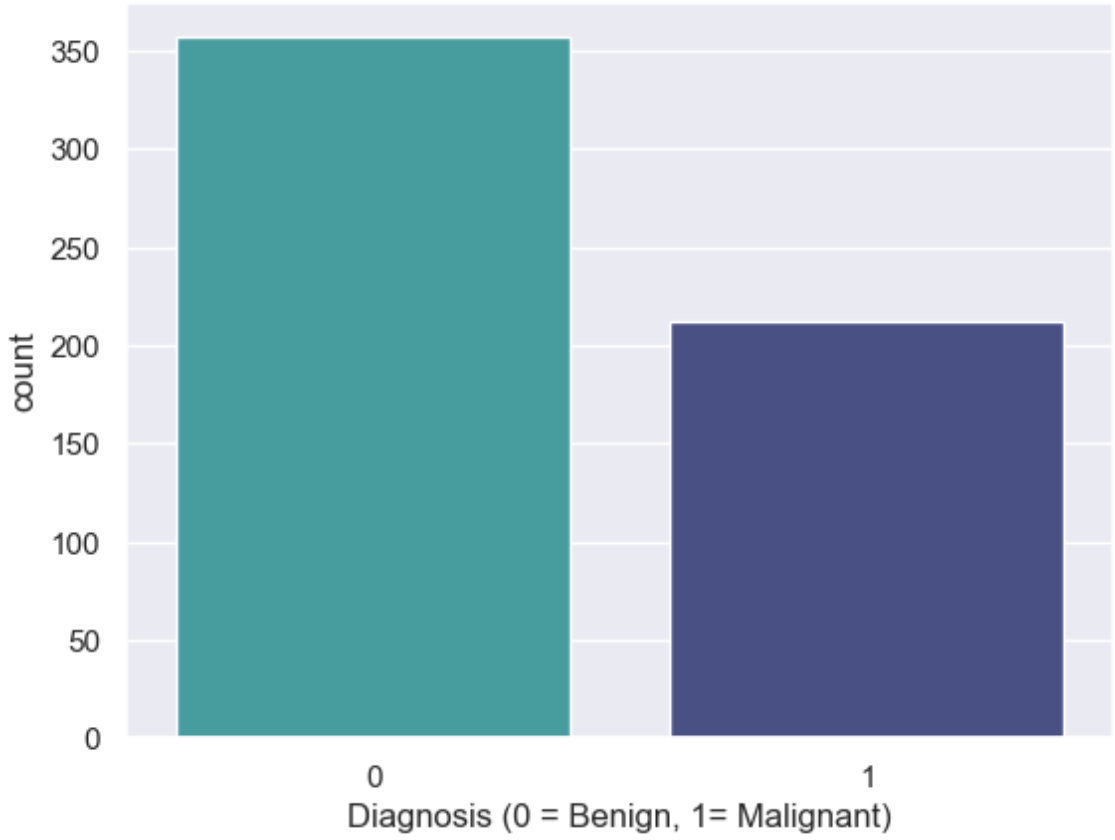
In [12]:

```
df['diagnosis'].value_counts().plot(kind='pie',explode=[0.1,0.1],autopct='%0.1f%%')
plt.show()
```



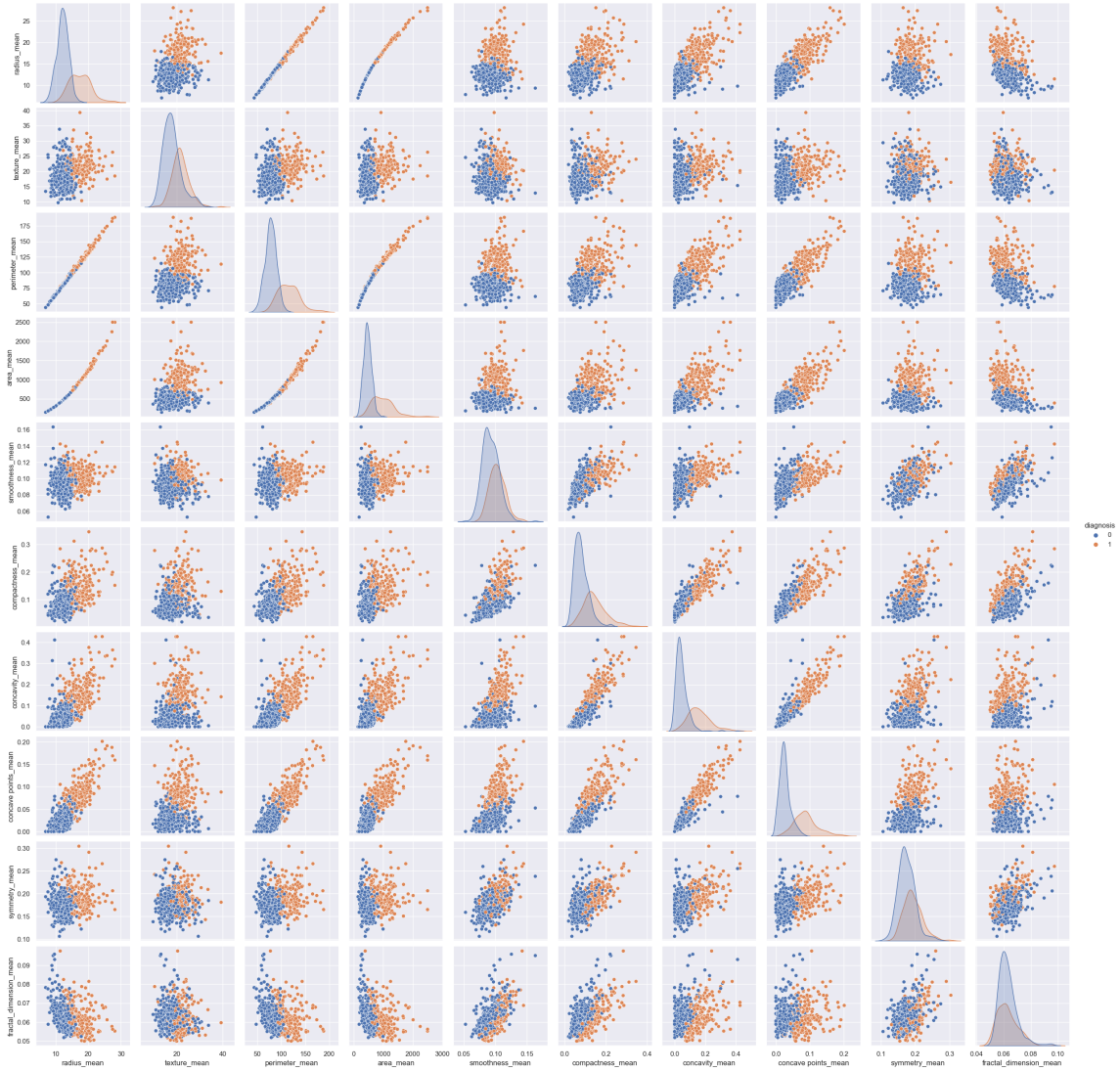
In [13]:

```
sns.countplot(x='diagnosis', data=df, palette="mako_r")
plt.xlabel("Diagnosis (0 = Benign, 1= Malignant)")
plt.show()
```



In [17]:

```
sns.pairplot(df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean', 'diagnosis']],
plt.show()
```

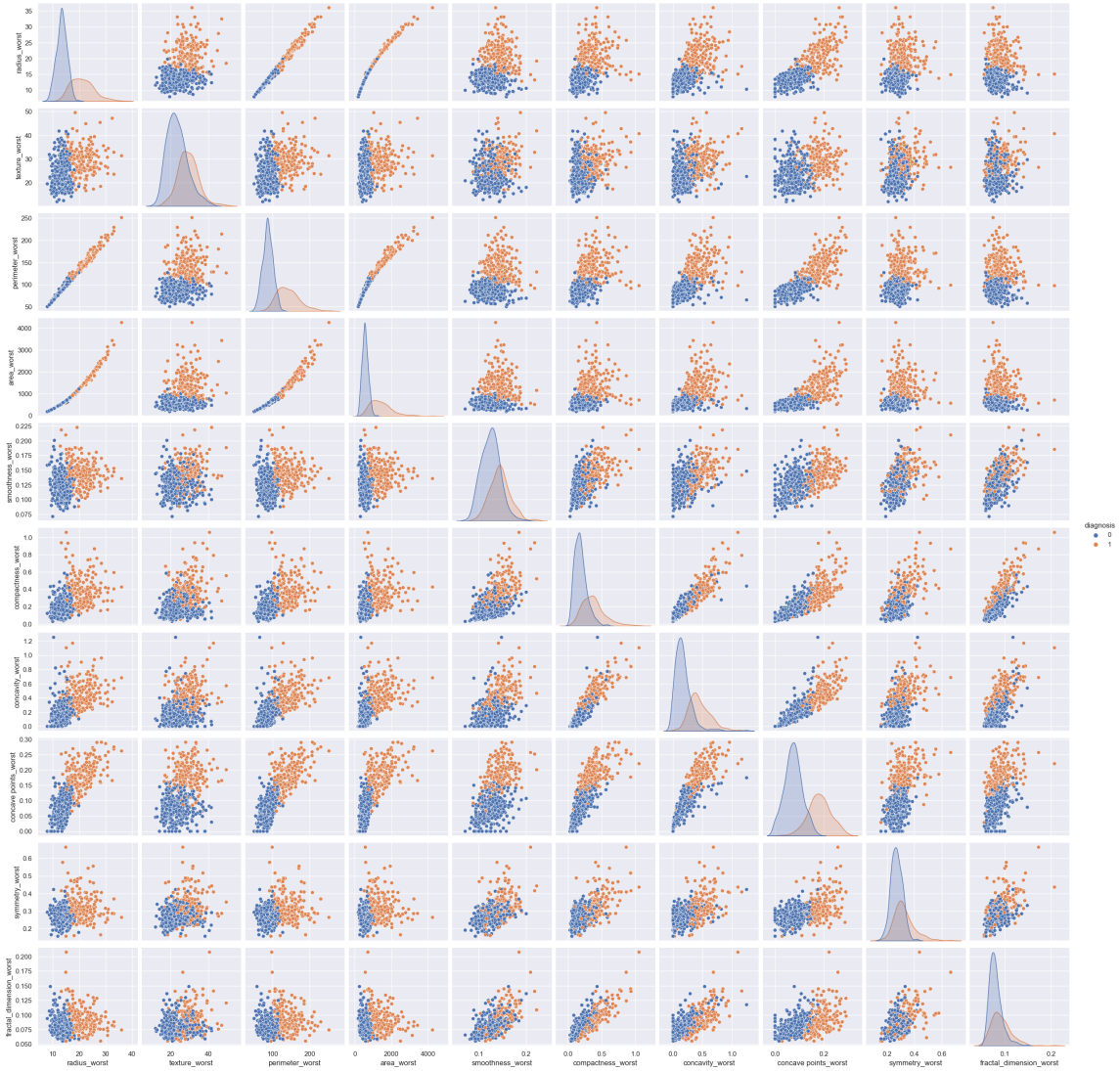


In [25]:

```
sns.pairplot(df[['radius_se', 'texture_se', 'perimeter_se', 'area_se',
                 'smoothness_se', 'compactness_se', 'concavity_se',
                 'concave points_se', 'symmetry_se', 'fractal_dimension_se', 'diagnosis'
plt.show()
```

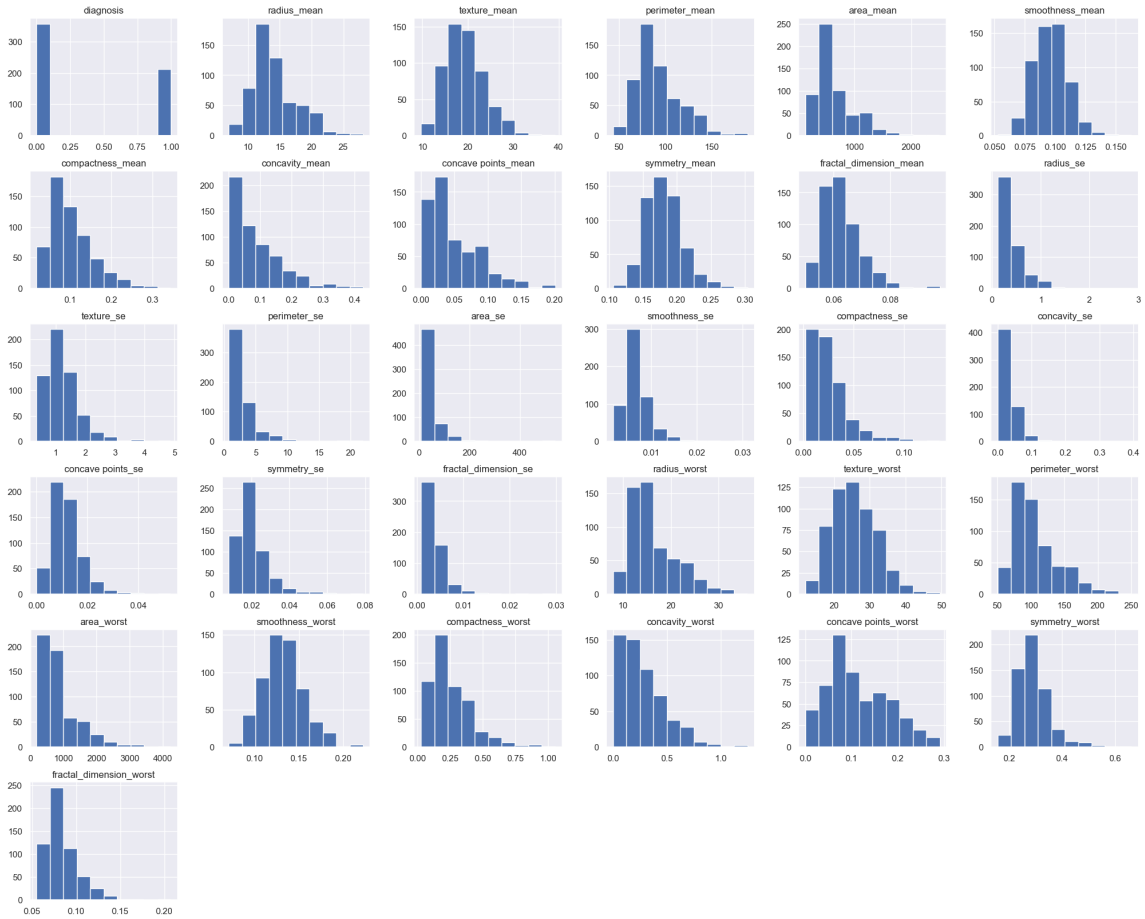
In [28]:

```
sns.pairplot(df[['radius_worst', 'texture_worst', 'perimeter_worst', 'area_worst',
       'smoothness_worst', 'compactness_worst', 'concavity_worst',
       'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst', 'diagnosis']
plt.show()
```



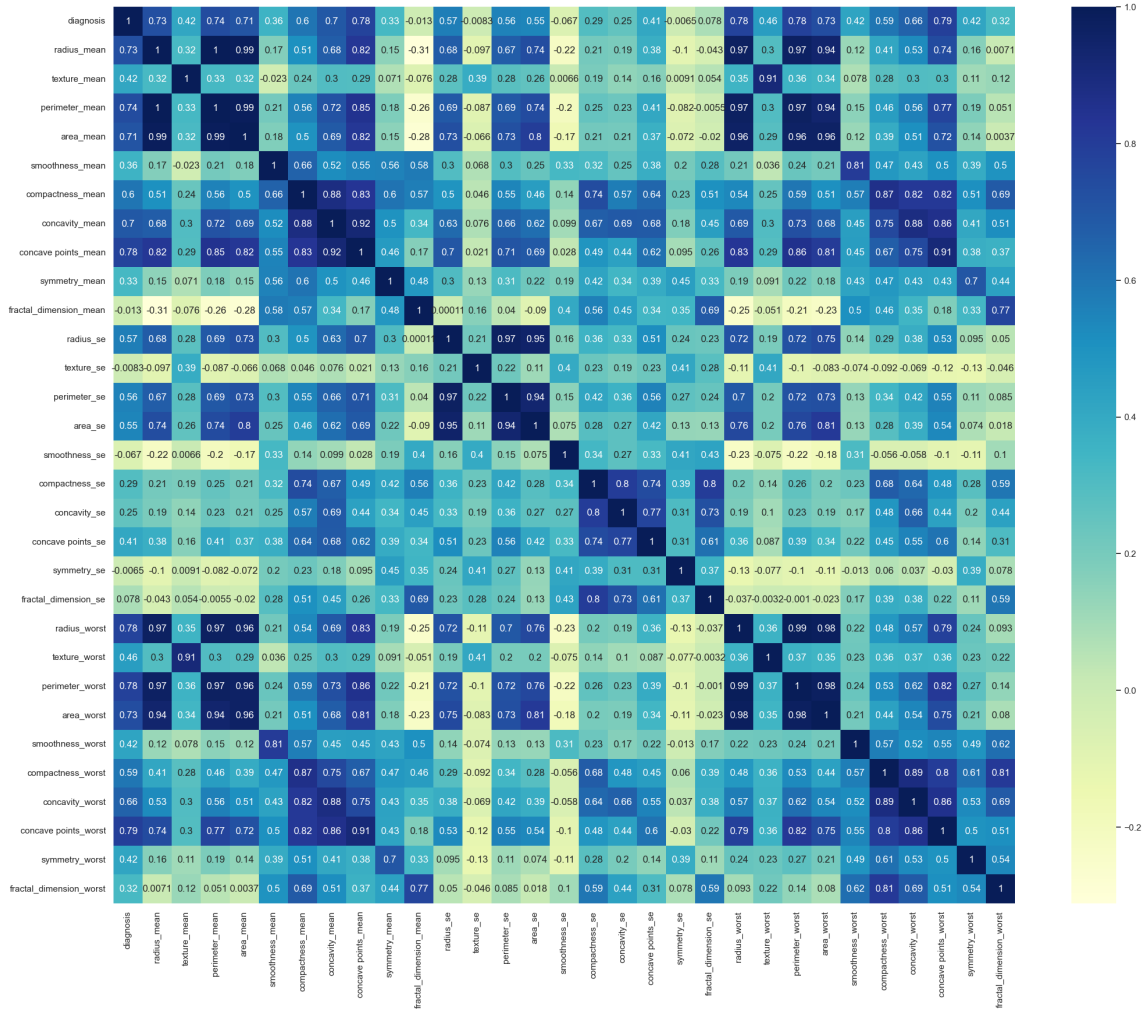
In [26]:

```
df.hist(figsize=(25,20))
plt.show()
```



In [15]:

```
# Find the Correlation by using Heatmap
plt.figure(figsize=(25,20))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
plt.show()
```



Data Splitting

In [38]:

```
# Split data into independent and Dependent variables
x = df.drop(['diagnosis'],axis=1)
y = df[['diagnosis']]
```

In [39]:

```
x.head(2)
```

Out[39]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
0	17.99	10.38	122.8	1001.0	0.11840	0
1	20.57	17.77	132.9	1326.0	0.08474	0

2 rows × 30 columns

In [40]:

```
y.head(2)
```

Out[40]:

	diagnosis
0	1
1	1

In [41]:

```
# Split the data into Training and Testing
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)
```

In [42]:

```
x_train.shape, x_test.shape
```

Out[42]:

```
((455, 30), (114, 30))
```

In [43]:

```
y_train.shape, y_test.shape
```

Out[43]:

```
((455, 1), (114, 1))
```

Build Model

In [44]:

```
# Logistic Regression Method
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression(random_state=0)
logit.fit(x_train, y_train)
```

Out[44]:

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

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:

```
# Predict
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
```

In [128]:

```
# Evaluate test data accuracy
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

acc_test = accuracy_score(y_test, y_pred_test)
f1_test = f1_score(y_test, y_pred_test)
prec_test = precision_score(y_test, y_pred_test)
rec_test = recall_score(y_test, y_pred_test)
conf_mat_test= confusion_matrix(y_test, y_pred_test)
test_accuracy= pd.DataFrame([['Test_data', acc_test, f1_test, prec_test, rec_test,conf_m
                      columns = ['Logistic_Regression', 'accuracy_score', 'f1', 'precis
```

In [129]:

```
# Evaluate train data accuracy
acc_train = accuracy_score(y_train, y_pred_train)
f1_train = f1_score(y_train, y_pred_train)
prec_train = precision_score(y_train, y_pred_train)
rec_train = recall_score(y_train, y_pred_train)
conf_mat_train= confusion_matrix(y_train, y_pred_train)
train_accuracy = pd.DataFrame([['Train_data', acc_train, f1_train, prec_train, rec_train
                      columns = ['Logistic_Regression', 'accuracy_score', 'f1', 'precis
```

In [130]:

```
Accuracy={"Test_data_accuracy":[acc_test, f1_test, prec_test, rec_test,conf_mat_test],
      "Train_data_accuracy":[acc_train, f1_train, prec_train, rec_train,conf_mat_train]}
pd.DataFrame(Accuracy,index=['accuracy_score', 'f1', 'precision', 'recall','confusion_ma
```

Out[130]:

	Test_data_accuracy	Train_data_accuracy
accuracy_score	0.947368	0.951648
f1	0.9375	0.932099
precision	0.918367	0.949686
recall	0.957447	0.915152
confusion_matrix	[[63, 4], [2, 45]]	[[282, 8], [14, 151]]

AUC (Area under the curve) & ROC (Receiver operating characteristics)

- It is one of the most important evaluation metrics for checking classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics)
- ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.

In [111]:

```
from sklearn.metrics import roc_auc_score
logit_roc_auc = roc_auc_score(y_test, y_pred_test)
logit_roc_auc
```

Out[111]:

```
0.9488726579866625
```

In [112]:

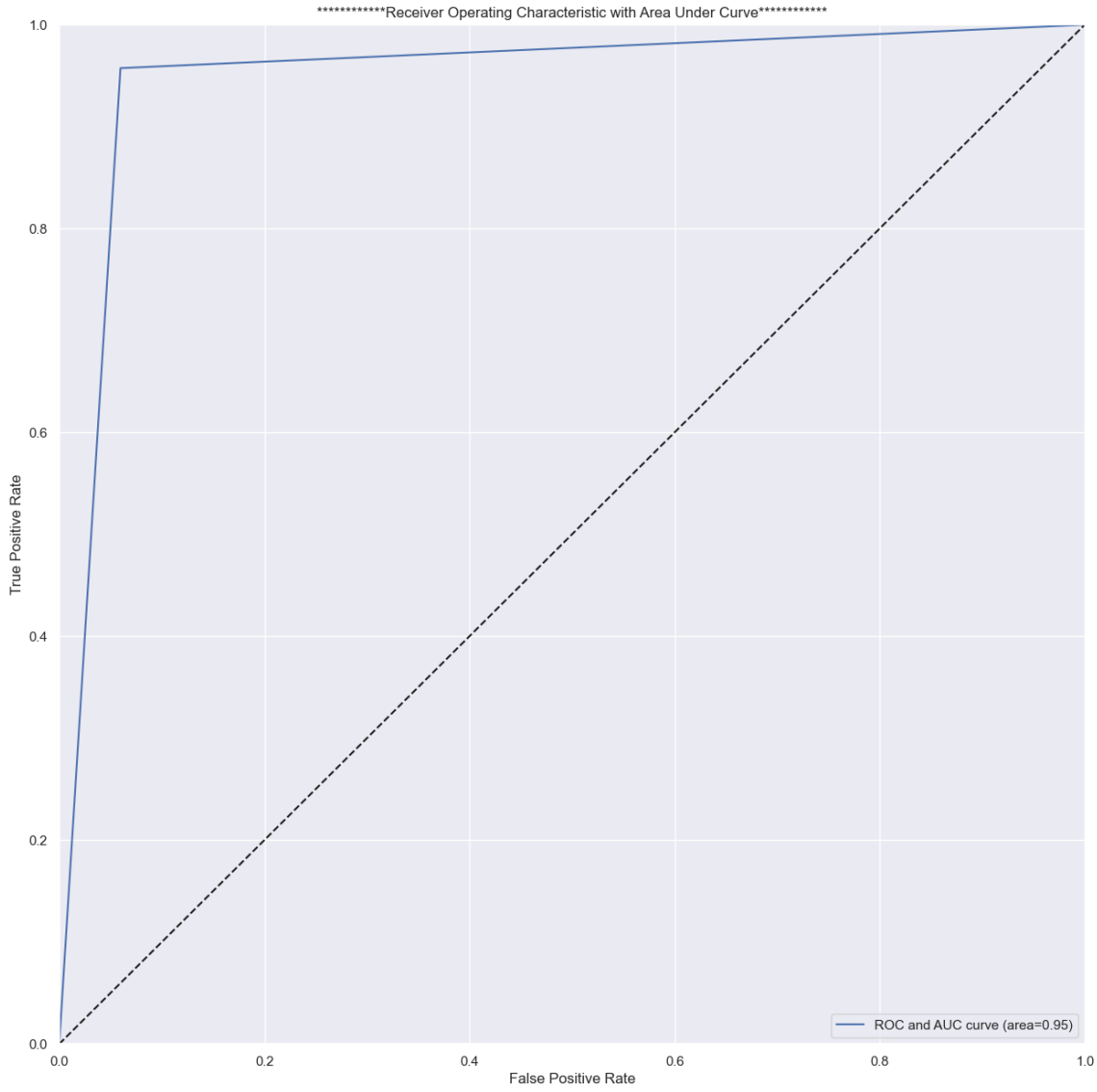
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
display(fpr[:10])
display(tpr[:10])
display(thresholds[:10])
```

```
array([0.        , 0.05970149, 1.        ])

array([0.        , 0.95744681, 1.        ])

array([2, 1, 0], dtype=int8)
```

Plotting ROC and AUC curve

In [113]:

```
plt.figure(figsize=(15,15))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % logit_roc_auc)
plt.plot([0,1],[0,1], 'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("************Receiver Operating Characteristic with Area Under Curve**********
plt.legend(loc='lower right')
plt.show()
```



Cross validation

- Cross-validation is a technique for validating the model efficiency by training it on the subset of input data and testing on previously unseen subset of the input data.

K - fold cross validation approach

- K-fold cross-validation is a technique for evaluating predictive models. The dataset is divided into k subsets or folds. The model is trained and evaluated k times, using a different fold as the validation set each time. Performance metrics from each fold are averaged to estimate the model's generalization performance. This method aids in model assessment, selection, and hyperparameter tuning, providing a more reliable measure of a model's effectiveness.

In [114]:

```
# Cross Validation approach by K-Fold Method
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(logit, x_train, y_train, cv=10)
test_accuracy = cross_val_score(logit, x_test, y_test, cv=10)
print(training_accuracy)
print()
print(test_accuracy)
print()
print("Training Average Accuracy", training_accuracy.mean())
print()
print("Test Average Accuracy", test_accuracy.mean())
```

```
[0.97826087 0.97826087 0.89130435 0.91304348 0.86956522 0.93333333
 0.91111111 0.97777778 0.97777778 0.95555556]

[1.         0.83333333 1.         0.91666667 0.81818182 1.
 0.90909091 0.90909091 0.90909091 0.90909091]

Training Average Accuracy 0.9385990338164252

Test Average Accuracy 0.9204545454545453
```

Conclusion

- Both test and train data accuracy is more than 90% which is over the commonly taken threshold value of 75%.
- Accuracy by K fold cross validation approach is also more than 90% accuracy
- There is less than 10% percent variation in train and test data.

In [ ]:

```

```

In [ ]:

```

```