PHASE 1 - GOLDEN TASK - Breast Cancer Wisconsin (Diagnostic) CODERSCAVE

Objective

- The objective is to classify whether the breast cancer is benign or malignant and to achieve this I will be using 8 Supervised machine learning algorithms.

Machine learning models applied

- Logistic Regression
- Decision Tree
- K Nearest Neighbors
- Random Forest
- Naive Bayes
- Support vector machines (SVMs)
- XGBoost
- Voting classifier

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

Project outline

- Importing Libraries & Dataset.
- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Data splitting into dependent & independent variable
- Feature scaling
- Splitting Data for Model Training
- Model Creation, Training and Evaluation
- Confusion Matrix Analysis
- Cross validation of selected models
- Model Training and Tuning

Import the required Libraries

In [2]:

```
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 & Read the dataset - Breast Cancer (Wisconsin)

In [114]:

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

Out[114]:

	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 [115]:

```
df.shape  #check shape
```

Out[115]:

```
(569, 32)
```

- Dataset has 569 rows and 32 columns

In [116]:

```
df.columns  #check column names
```

Out[116]:

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mea
n',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_s
e',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_s
e',
       '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'],
      dtype='object')
```

In [117]:

```
df.duplicated().sum()  #check duplicates
```

Out[117]:

```
0
```

- No duplicates present

In [118]:

```
df.isnull().sum()   #check null values
```

Out[118]:

```
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
```

- No null values present

In [119]:

```
df.info()   #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
```

- All are numerical columns except diagnosis

In [120]:

```
df.describe().T.style.background_gradient(cmap='Blues')   #Statistical Analysis on Numer
```

Out[120]:

	count	mean	std	min	
id	569.000000	30371831.432337	125020585.612224	8670.000000	86921
radius_mean	569.000000	14.127292	3.524049	6.981000	1
texture_mean	569.000000	19.289649	4.301036	9.710000	1
perimeter_mean	569.000000	91.969033	24.298981	43.790000	7
area_mean	569.000000	654.889104	351.914129	143.500000	42
smoothness_mean	569.000000	0.096360	0.014064	0.052630	
compactness_mean	569.000000	0.104341	0.052813	0.019380	
concavity_mean	569.000000	0.088799	0.079720	0.000000	
concave points_mean	569.000000	0.048919	0.038803	0.000000	
symmetry_mean	569.000000	0.181162	0.027414	0.106000	
fractal_dimension_mean	569.000000	0.062798	0.007060	0.049960	
radius_se	569.000000	0.405172	0.277313	0.111500	
texture_se	569.000000	1.216853	0.551648	0.360200	
perimeter_se	569.000000	2.866059	2.021855	0.757000	
area_se	569.000000	40.337079	45.491006	6.802000	1
smoothness_se	569.000000	0.007041	0.003003	0.001713	
compactness_se	569.000000	0.025478	0.017908	0.002252	
concavity_se	569.000000	0.031894	0.030186	0.000000	
concave points_se	569.000000	0.011796	0.006170	0.000000	
symmetry_se	569.000000	0.020542	0.008266	0.007882	
fractal_dimension_se	569.000000	0.003795	0.002646	0.000895	
radius_worst	569.000000	16.269190	4.833242	7.930000	1
texture_worst	569.000000	25.677223	6.146258	12.020000	2
perimeter_worst	569.000000	107.261213	33.602542	50.410000	8
area_worst	569.000000	880.583128	569.356993	185.200000	51
smoothness_worst	569.000000	0.132369	0.022832	0.071170	
compactness_worst	569.000000	0.254265	0.157336	0.027290	
concavity_worst	569.000000	0.272188	0.208624	0.000000	
concave points_worst	569.000000	0.114606	0.065732	0.000000	
symmetry_worst	569.000000	0.290076	0.061867	0.156500	
fractal_dimension_worst	569.000000	0.083946	0.018061	0.055040	

Dropping non significant variables

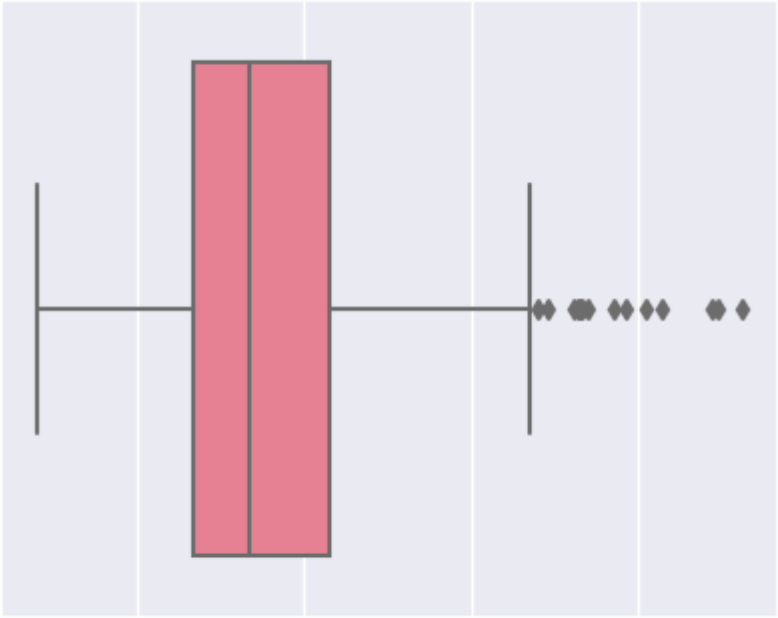
In [121]:

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

Checking for outliers

In [122]:

```
def boxplots(col):
    plt.figure(figsize=(5,4))
    sns.boxplot(df,x=col,palette='husl')
    plt.show()

for i in list(df.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```



Check balance of data

In [123]:

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

Out[123]:

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

Label encoder for diagnosis variable to convert it from categorical variable to numerical variable.

In [16]:

```
df['diagnosis']=df['diagnosis'].astype('category')
df['diagnosis']=df['diagnosis'].cat.codes
```

Exploratory Data Analysis

In [14]:

```
from dataprep.eda import create_report
report = create_report(df, title='Data Report')
report
```

```
  0%|
| 0/7133 [00:00<…
```

Out[14]:

Data Report

Data Report Overview

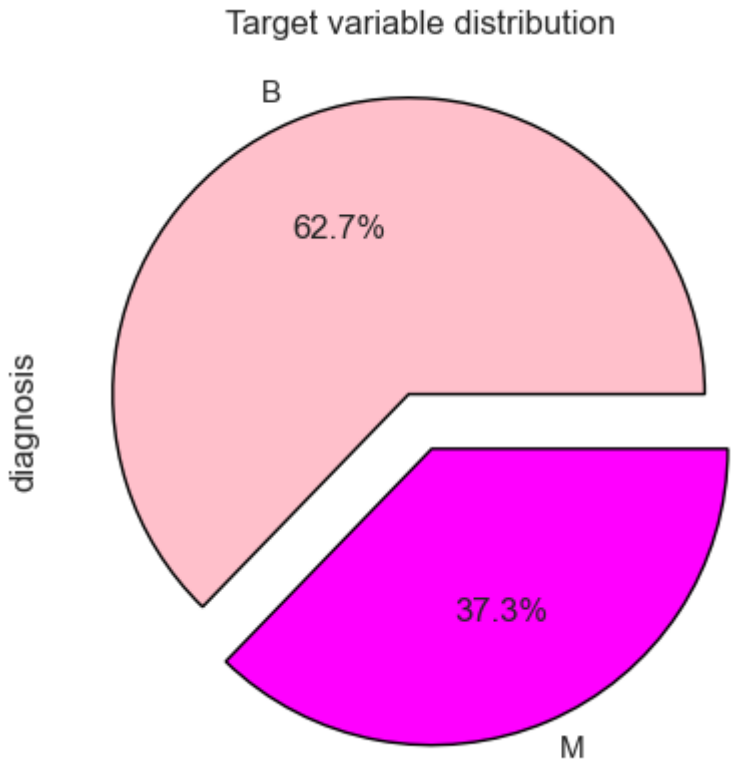
Variables ≡

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

Interactions Correlations Missing Values

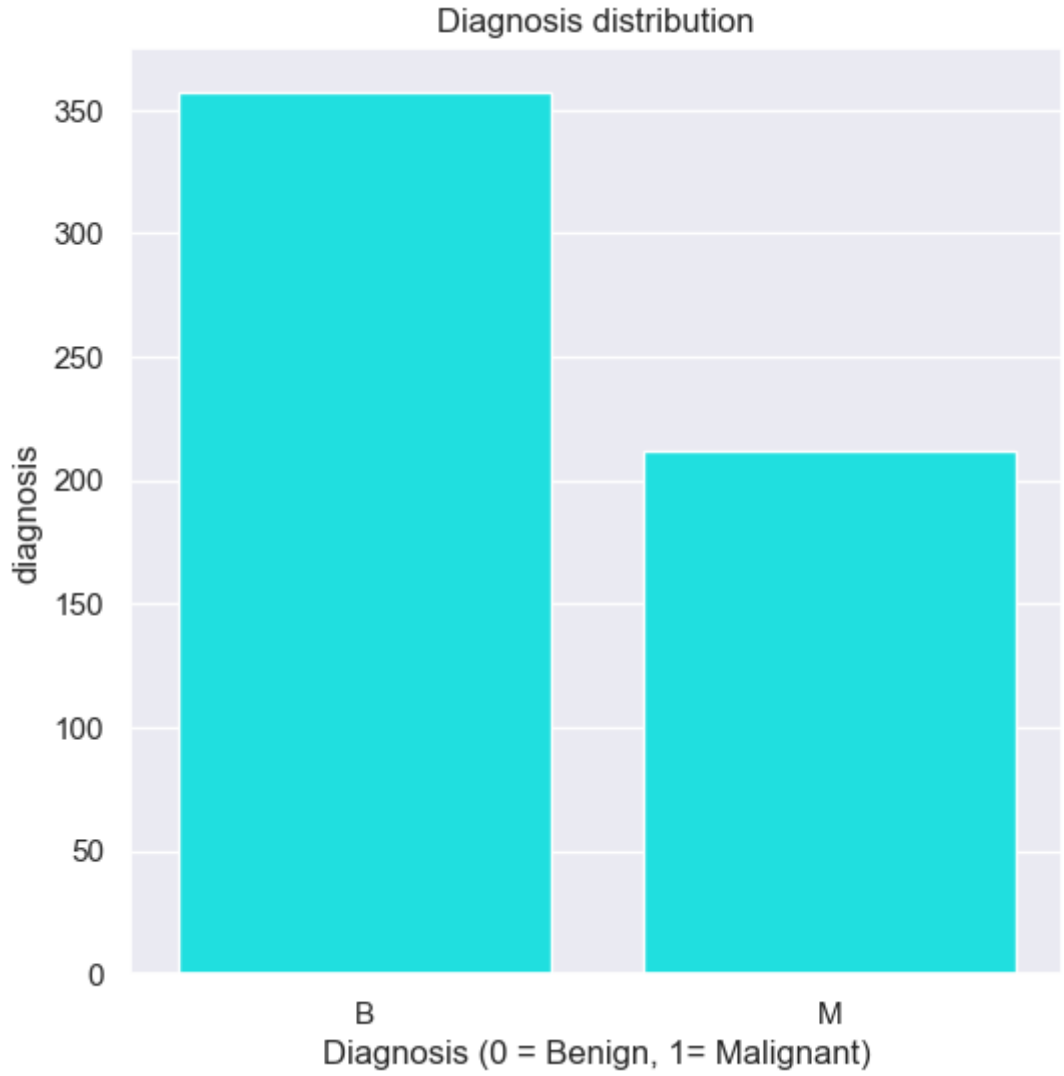
In [125]:

```
df['diagnosis'].value_counts().plot(kind='pie',explode=[0.1,0.1],autopct='%0.1f%%',
                                    colors=('pink','fuchsia'),wedgeprops={'edgecolor': 'bla
plt.title('Target variable distribution')
plt.show()
```



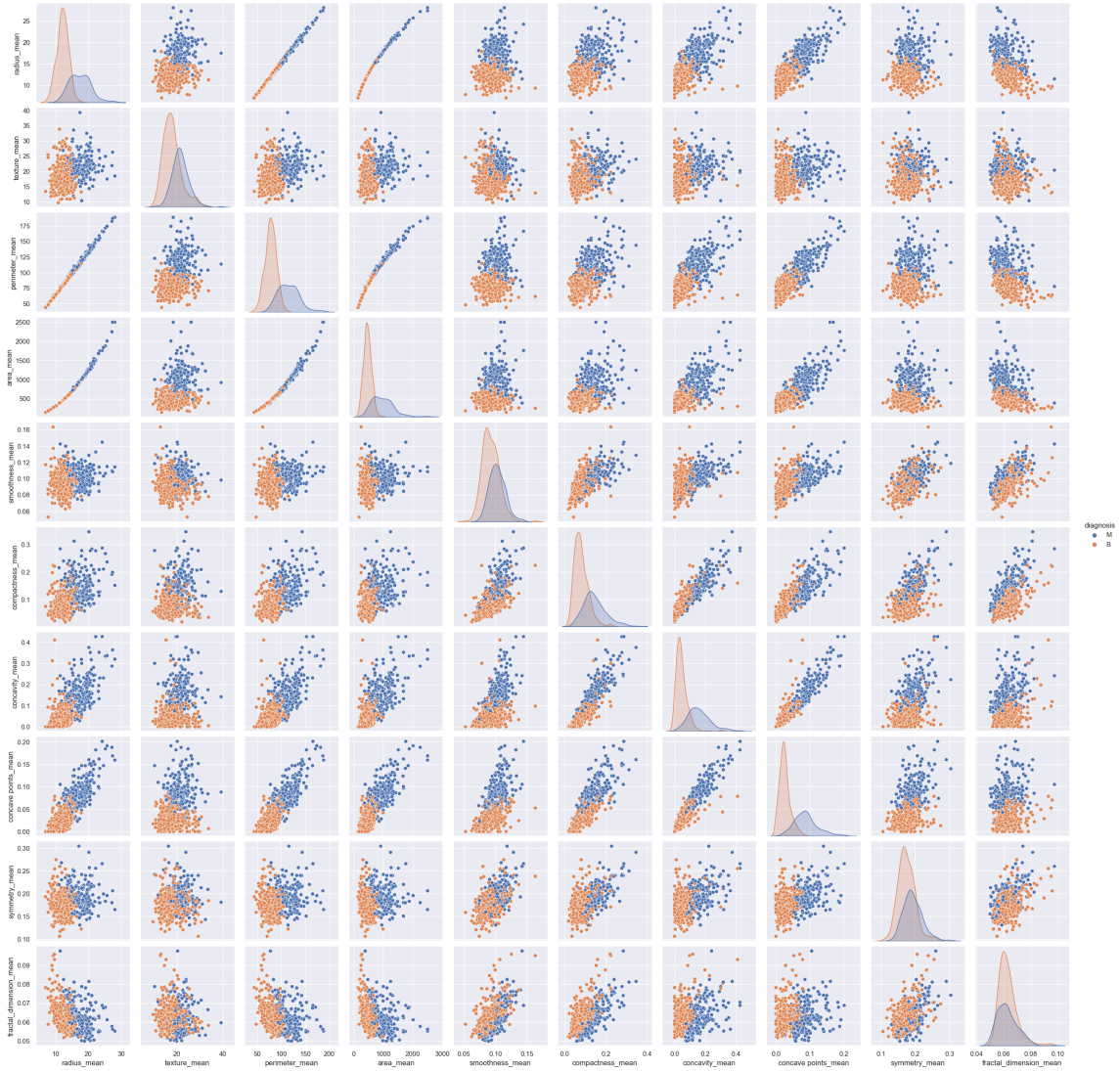
In [134]:

```
plt.figure(figsize=(6,6))
sns.barplot(x=df['diagnosis'].value_counts().index, y=df['diagnosis'].value_counts(), co
plt.title('Diagnosis distribution')
plt.xlabel("Diagnosis (0 = Benign, 1= Malignant)")
plt.show()
```



In [126]:

```
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()
```

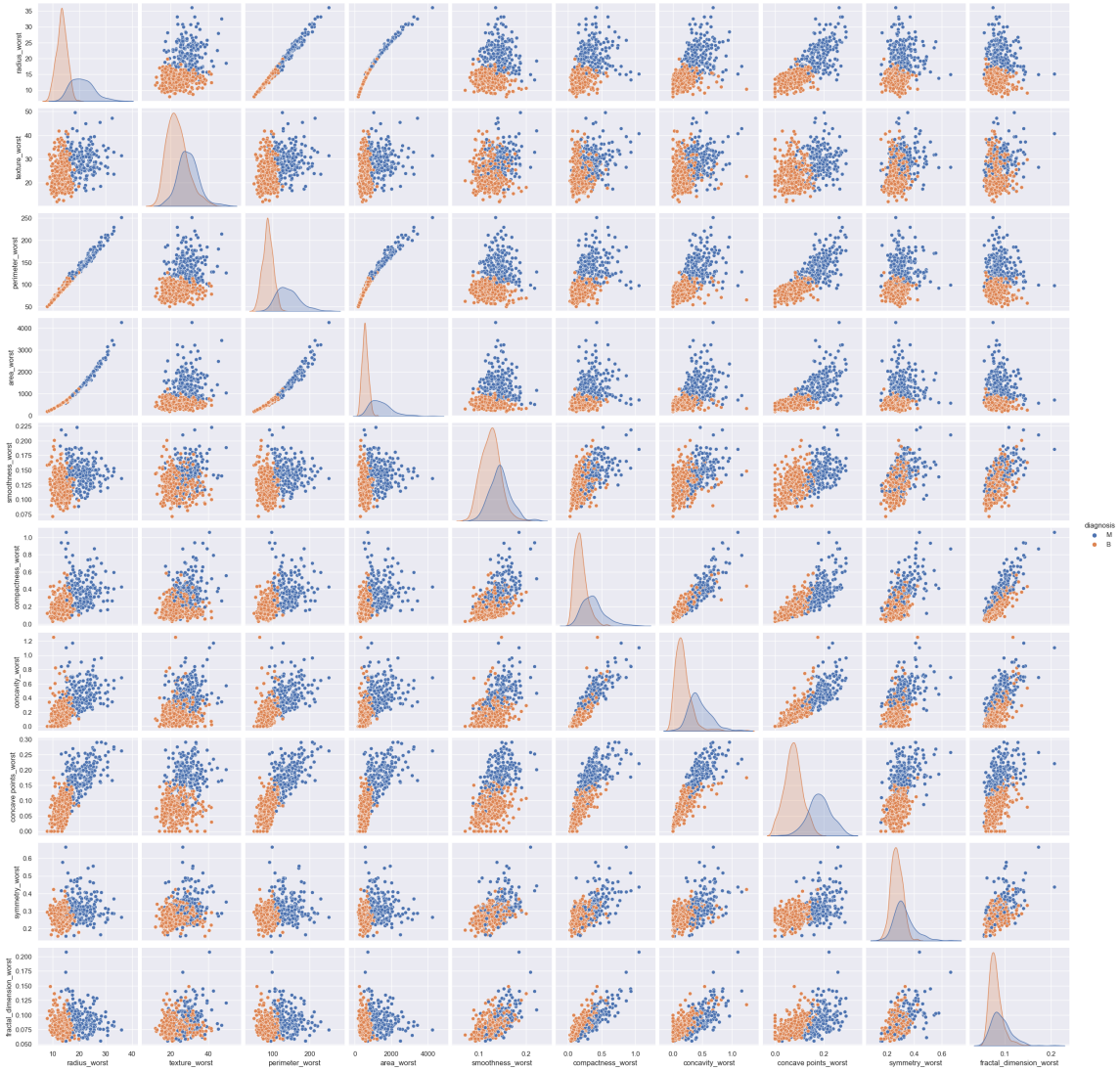


In [127]:

```
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()
```

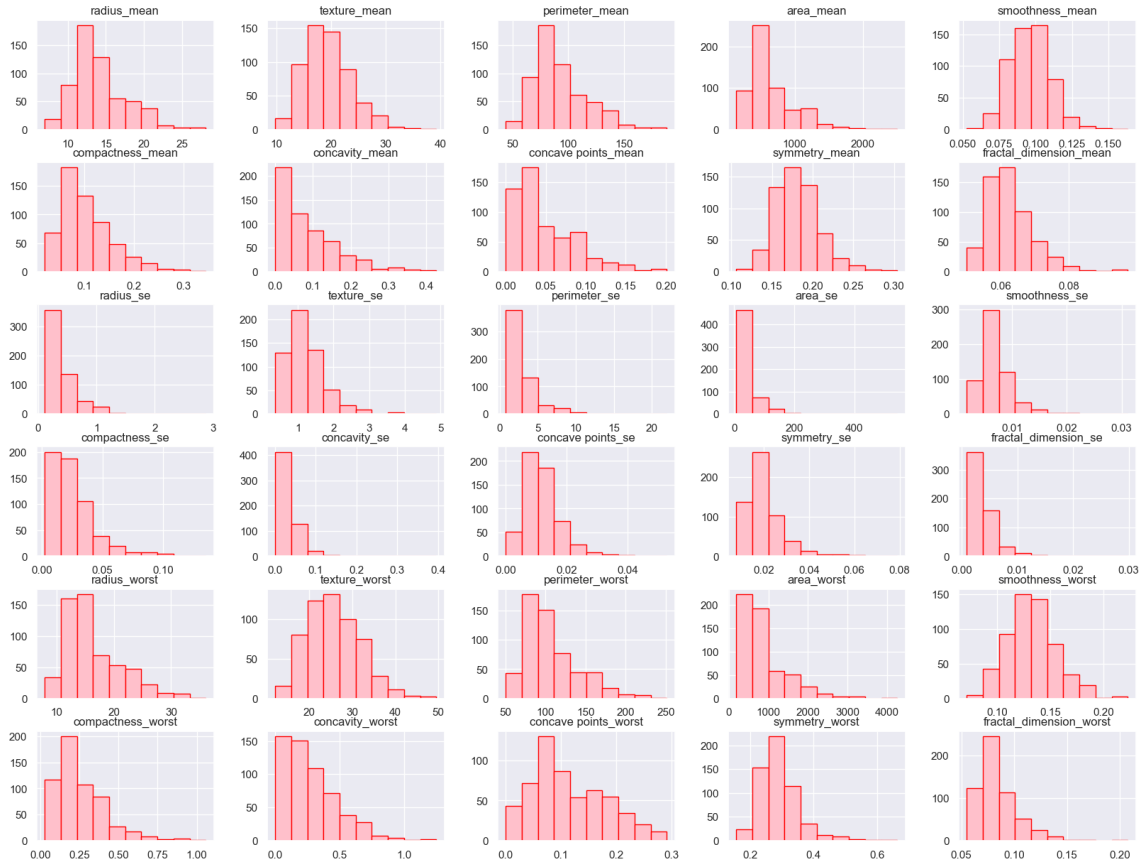
In [128]:

```
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()
```



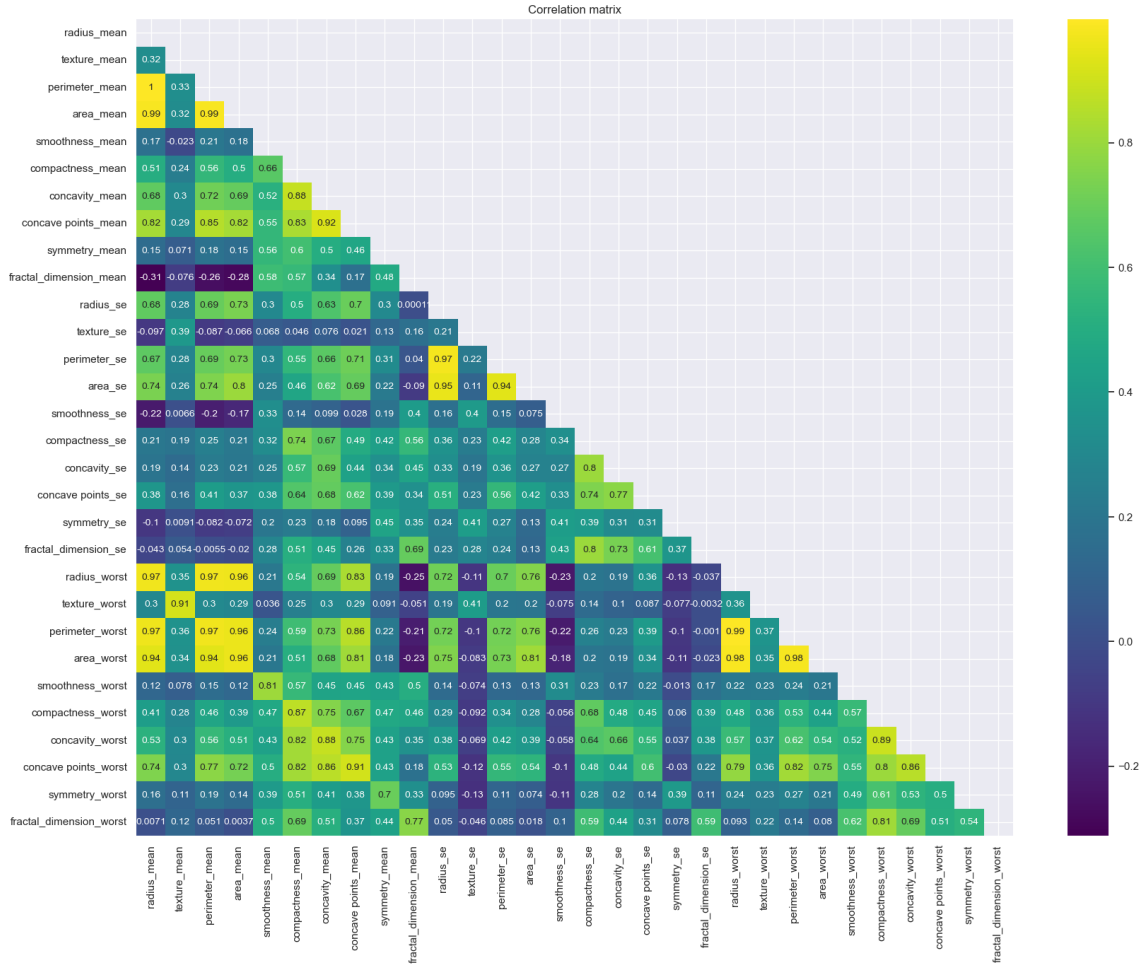
In [133]:

```
df.hist(bins=10, figsize=(20,15),color='pink', edgecolor='red')
plt.show()
```



In [131]:

```
mask = np.zeros_like(df.corr(), dtype=float)
mask[np.triu_indices_from(mask)]=True
plt.figure(figsize=(20,15))
sns.heatmap(df.corr(),annot=True,cmap='viridis', annot_kws={'size':10}, mask=mask)
plt.title('Correlation matrix')
plt.show()
```



Split the data into Independent (x) & Dependent(y) variables

In [18]:

```
x = df.drop(['diagnosis'],axis=1)
y = df[['diagnosis']]
```

In [19]:

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

Out[19]:

	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 [20]:

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

Out[20]:

	diagnosis
0	1
1	1

Feature Scaling

In [21]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x1=sc.fit_transform(x)
pd.DataFrame(x1).head()
```

Out[21]:

	0	1	2	3	4	5	6	7	
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.283515	2.652874	2.532475	2.21
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.487072	-0.023846	0.548144	0.00
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.052926	1.363478	2.037231	0.93
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.402909	1.915897	1.451707	2.86
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.539340	1.371011	1.428493	-0.00

5 rows × 30 columns

Split the data into Train and Test

In [22]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x1,y,test_size=0.2,random_state=0)
```

In [25]:

```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[25]:

```
((455, 30), (114, 30), (455, 1), (114, 1))
```

Model Building

Model No. 1 - Logistic Regression

- It is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class.

In [26]:

```
# Model building
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression(random_state=100)
log=logit.fit(x_train, y_train)
# Predict
y_pred_train_log = logit.predict(x_train)
y_pred_test_log = logit.predict(x_test)
# Evaluate
from sklearn.metrics import confusion_matrix,classification_report, accuracy_score
accuracy_log_test=accuracy_score(y_test,y_pred_test_log)
accuracy_log_train=accuracy_score(y_train,y_pred_train_log)
print('Logistic regression Train accuracy:', accuracy_score(y_train, y_pred_train_log))
print('------'*10)
print('Logistic regression Test accuracy:', accuracy_score(y_test, y_pred_test_log))
```

```
Logistic regression Train accuracy: 0.989010989010989
------------------------------------------------------------
Logistic regression Test accuracy: 0.9649122807017544
```

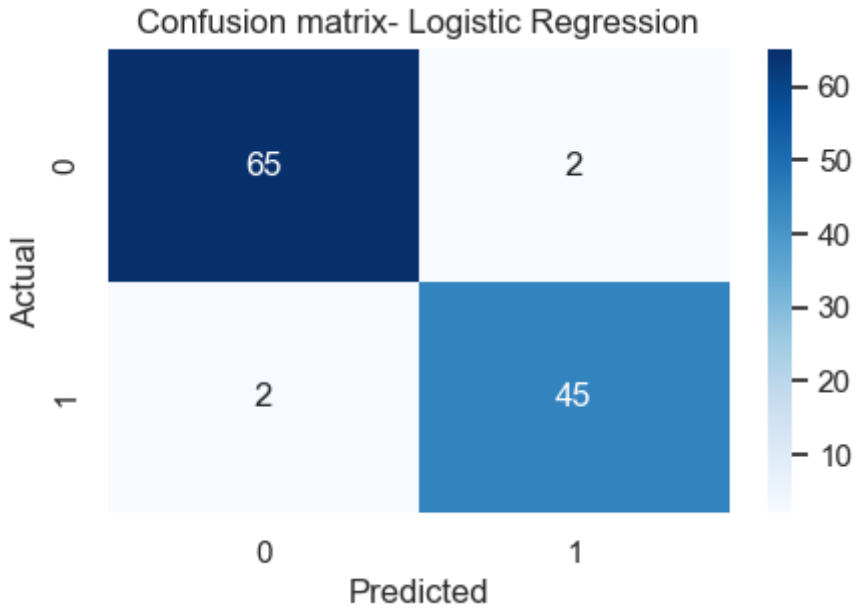
In [100]:

```
train_accuracy_log = cross_val_score(log,x_train, y_train, cv=10)
crossval_train_log=train_accuracy_log.mean()
test_accuracy_log = cross_val_score(log,x_test, y_test, cv=10)
crossval_test_log=test_accuracy_log.mean()
print('Logistic regression Train accuracy after Cross validation:', crossval_train_log)
print('-------'*5)
print('Logistic regression Test accuracy after Cross validation:', crossval_test_log)
```

```
Logistic regression Train accuracy after Cross validation: 0.9780676328502
416
-----------------------------------
Logistic regression Test accuracy after Cross validation: 0.98181818181818
17
```

In [27]:

```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_log),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Logistic Regression")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



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 [38]:

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

Out[38]:

```
0.9637980311209908
```

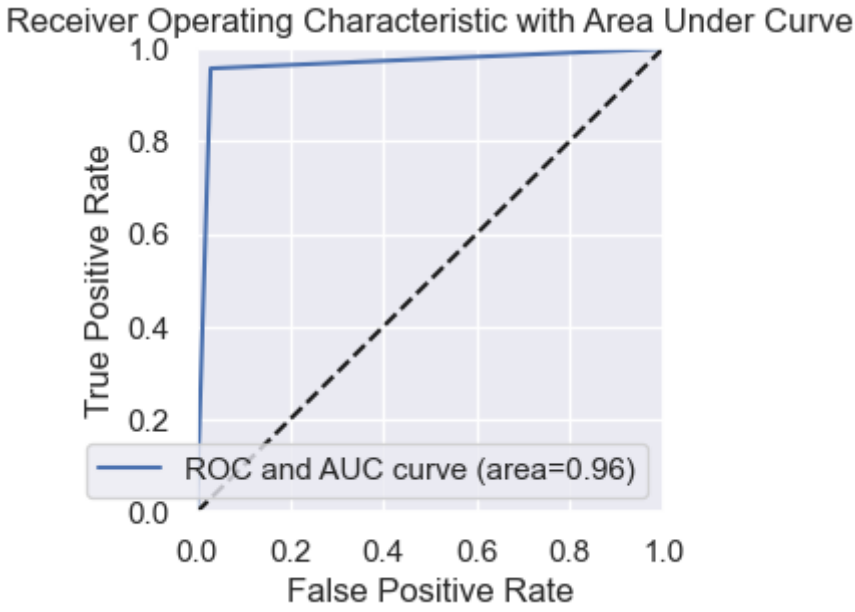
In [36]:

```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test_log)
```

Plotting ROC and AUC curve

In [40]:

```
plt.figure(figsize=(3,3))
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()
```



Model No. 2 - Decision Tree

- A decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree

In [41]:

```
# Model building
from sklearn.tree import DecisionTreeClassifier,plot_tree
dtree= DecisionTreeClassifier()
dtree.fit(x_train,y_train)
#Predict
y_pred_train_dtree=dtree.predict(x_train)
y_pred_test_dtree=dtree.predict(x_test)
#Evaluate
accuracy_dtree_test=accuracy_score(y_test,y_pred_test_dtree)
accuracy_dtree_train=accuracy_score(y_train,y_pred_train_dtree)
print('Decision Tree - Train accuracy:', accuracy_score(y_train, y_pred_train_dtree))
print('-------'*10)
print('Decision Tree - Test accuracy:', accuracy_score(y_test, y_pred_test_dtree))
```

```
Decision Tree - Train accuracy: 1.0
----------------------------------------------------------------------
Decision Tree - Test accuracy: 0.9122807017543859
```

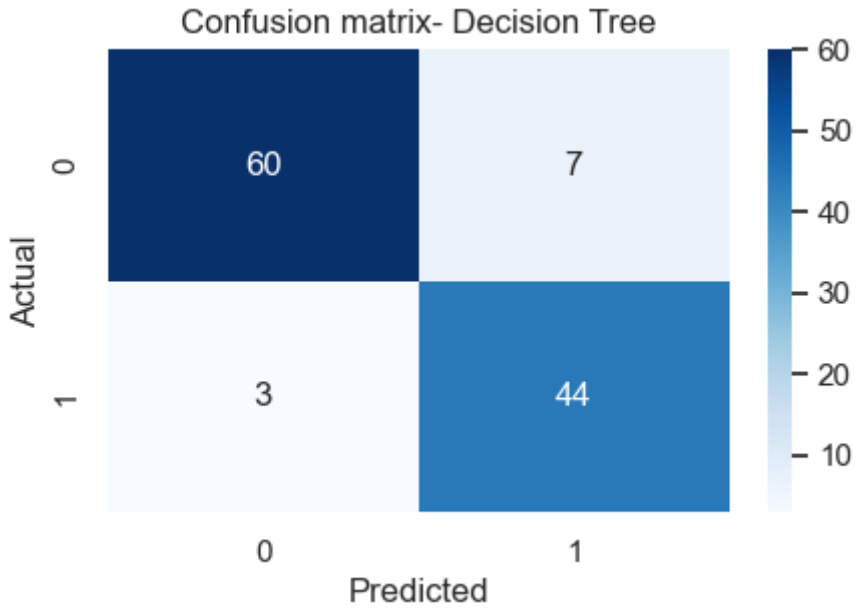
In [101]:

```
train_accuracy_dtree = cross_val_score(dtree,x_train, y_train, cv=10)
crossval_train_dtree=train_accuracy_dtree.mean()
test_accuracy_dtree = cross_val_score(dtree,x_test, y_test, cv=10)
crossval_test_dtree=test_accuracy_dtree.mean()
print('Decision Tree Train accuracy after Cross validation:', crossval_train_dtree)
print('-------'*10)
print('Decision Tree Test accuracy after Cross validation:', crossval_test_dtree)
```

```
Decision Tree Train accuracy after Cross validation: 0.9297101449275363
----------------------------------------------------------------------
Decision Tree Test accuracy after Cross validation: 0.9037878787878787
```

In [42]:

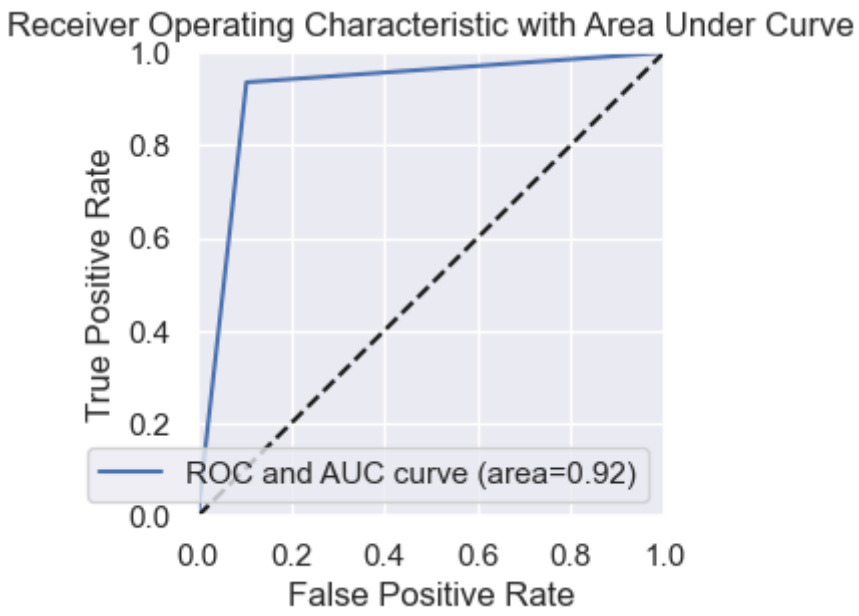
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_dtree),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Decision Tree")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [55]:

```
dtree_roc_auc = roc_auc_score(y_test, y_pred_test_dtree)
print(dtree_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % dtree_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()
```

```
0.9158463004128296
```



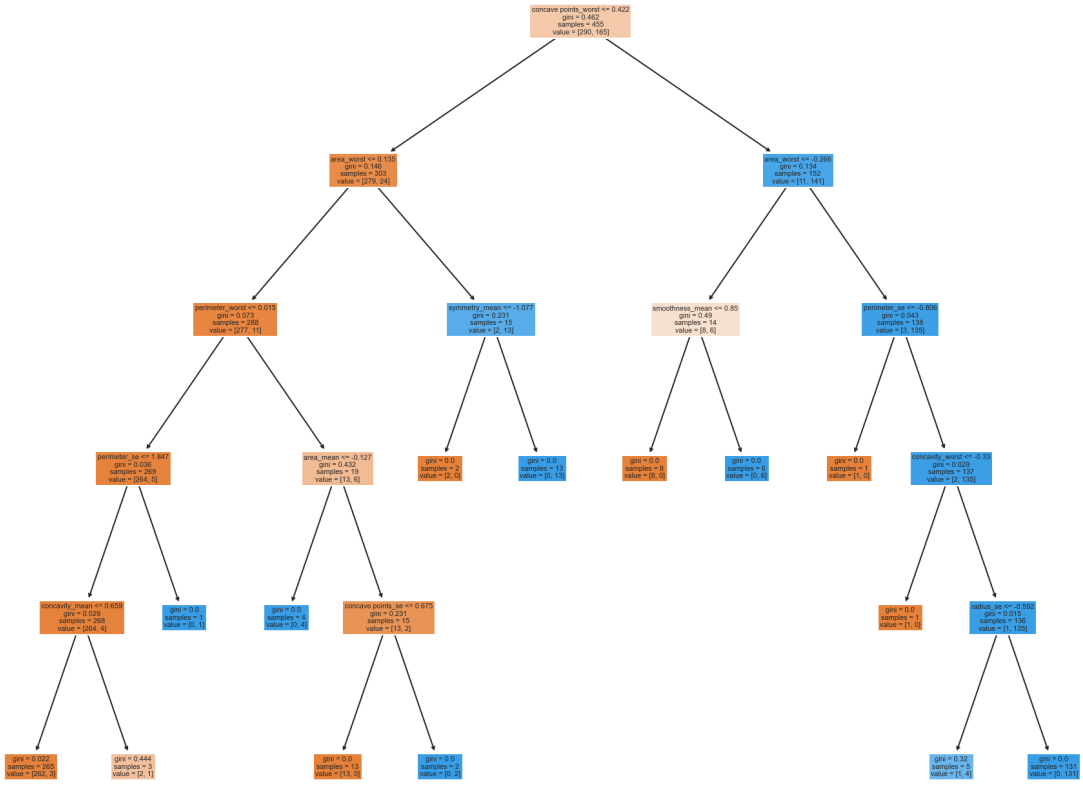
In [65]:

```
# Using Post prunning method to handle overfitting probelm
def dtree_model(model):
    model_preds=model.predict(x_test)
    print(classification_report(y_test,model_preds))
    print('\n')
    plt.figure(figsize=(15,12),dpi=150)
    plot_tree(model,filled=True,feature_names=x.columns)
plt.show()
```

In [66]:

```
# max depth at 5
prunned_dtree=DecisionTreeClassifier(max_depth=5)
prunned_dtree.fit(x_train,y_train)
dtree_model(prunned_dtree)
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        67
           1       0.94      0.94      0.94        47

    accuracy                           0.95       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.95      0.95      0.95       114
```



In [67]:

```
# Predict
y_pred_prunned_train=prunned_dtree.predict(x_train)
y_pred_prunned_test=prunned_dtree.predict(x_test)
# Evaluate
print('Decision Tree post prunning- Train accuracy:',accuracy_score(y_train,y_pred_prunn
print('-------'*10)
print('Decision Tree post prunning- Test accuracy:', accuracy_score(y_test,y_pred_prunne
```

```
Decision Tree post prunning- Train accuracy: 0.989010989010989
----------------------------------------------------------------------
Decision Tree post prunning- Test accuracy: 0.9473684210526315
```

Model No. 3 - K Nearest Neighbors (KNN)

- The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

In [47]:

```
# Model building with K point as 3
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
# Predict
y_pred_train_knn = knn.predict(x_train)
y_pred_test_knn = knn.predict(x_test)
#Evaluate
accuracy_knn_test=accuracy_score(y_test,y_pred_test_knn)
accuracy_knn_train=accuracy_score(y_train,y_pred_train_knn)
print('K nearest neighbor - Train accuracy:', accuracy_score(y_train, y_pred_train_knn))
print('-------'*10)
print('K nearest neighbor - Test accuracy:', accuracy_score(y_test, y_pred_test_knn))
```

```
K nearest neighbor - Train accuracy: 0.9802197802197802
----------------------------------------------------------------------
K nearest neighbor - Test accuracy: 0.956140350877193
```

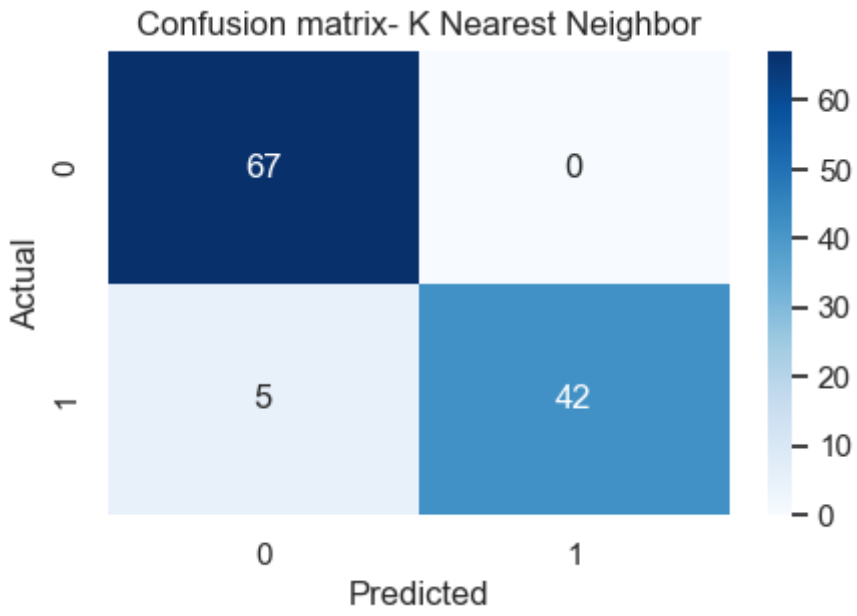
In [99]:

```
train_accuracy_knn = cross_val_score(knn,x_train, y_train, cv=10)
crossval_train_knn=train_accuracy_knn.mean()
test_accuracy_knn = cross_val_score(knn,x_test, y_test, cv=10)
crossval_test_knn=test_accuracy_knn.mean()
print('K Nearest Neighbor after Cross validation Train accuracy:', crossval_train_knn)
print('---------------'*5)
print('K Nearest Neighbor after Cross validation Test accuracy:', crossval_test_knn)
```

```
K Nearest Neighbor after Cross validation Train accuracy: 0.95400966183574
88
---------------------------------------------------------------------------
-
K Nearest Neighbor after Cross validation Test accuracy: 0.939393939393939
2
```

In [48]:

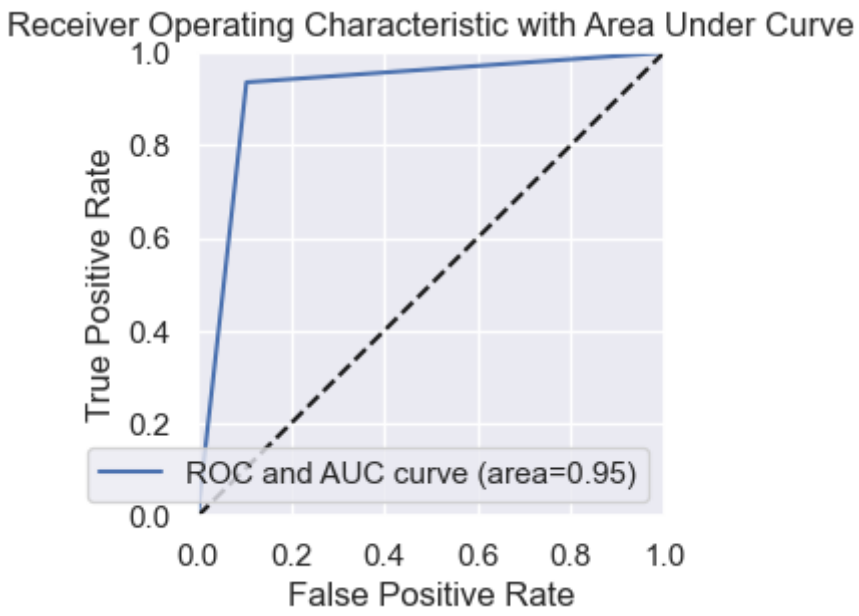
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_knn),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- K Nearest Neighbor")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [54]:

```
knn_roc_auc = roc_auc_score(y_test, y_pred_test_knn)
print(knn_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % knn_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()
```

```
0.9468085106382979
```



In [69]:

```
from sklearn.model_selection import cross_val_score
```

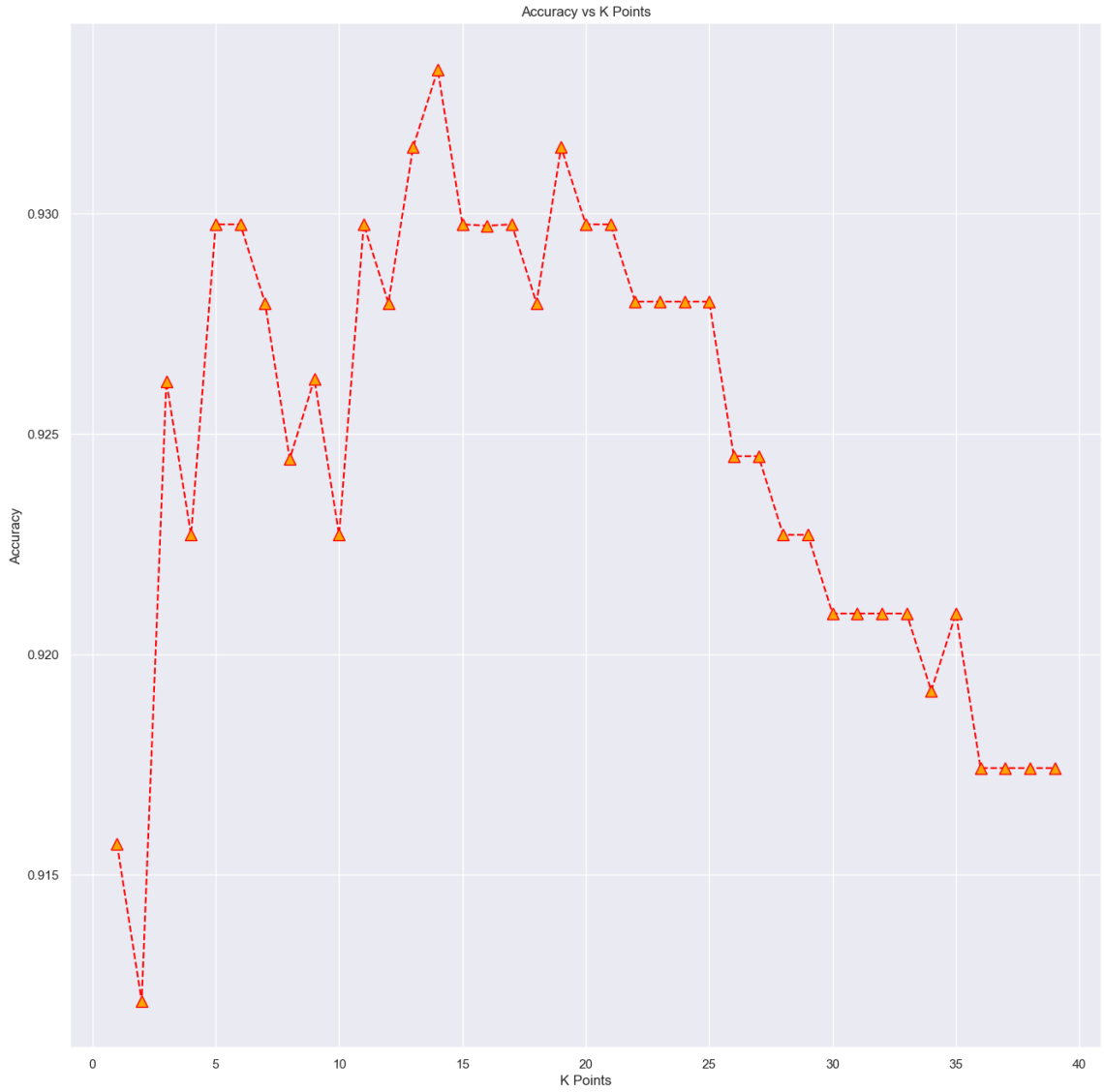
In [70]:

```
accuracy=[]
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,x,y,cv=10)
    accuracy.append(score.mean())
```

Plotting the graph

In [71]:

```
plt.figure(figsize=(16,16))
plt.plot(range(1,40), accuracy, color='red', linestyle='dashed', marker='^',
         markerfacecolor='orange', markersize=10)
plt.title('Accuracy vs K Points')
plt.xlabel('K Points')
plt.ylabel('Accuracy')
plt.show()
```



Model No. 4 - Random Forest

- A random forest is a machine learning technique that utilizes ensemble learning - which is a technique that combines many classifiers to provide solutions to complex problems. It establishes the outcome based on the predictions of the decision trees and employs the bagging method to generate the required prediction. It eradicates the biggest limitation of decision tree - overfitting of dataset and increases precision. The main difference between the decision tree algorithm and the random forest algorithm is that establishing root nodes and segregating nodes is done randomly in the latter

In [51]:

```
# Model building
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=200,oob_score=False)
rf.fit(x_train,y_train)
# Predict
y_pred_train_rf=rf.predict(x_train)
y_pred_test_rf=rf.predict(x_test)
# Evaluate
accuracy_rf_test=accuracy_score(y_test,y_pred_test_rf)
accuracy_rf_train=accuracy_score(y_train,y_pred_train_rf)
print('Random Forest - Train accuracy:', accuracy_score(y_train, y_pred_train_rf))
print('-------'*10)
print('Random Forest - Test accuracy:', accuracy_score(y_test, y_pred_test_rf))
```

```
Random Forest - Train accuracy: 1.0
----------------------------------------------------------------------
Random Forest - Test accuracy: 0.956140350877193
```

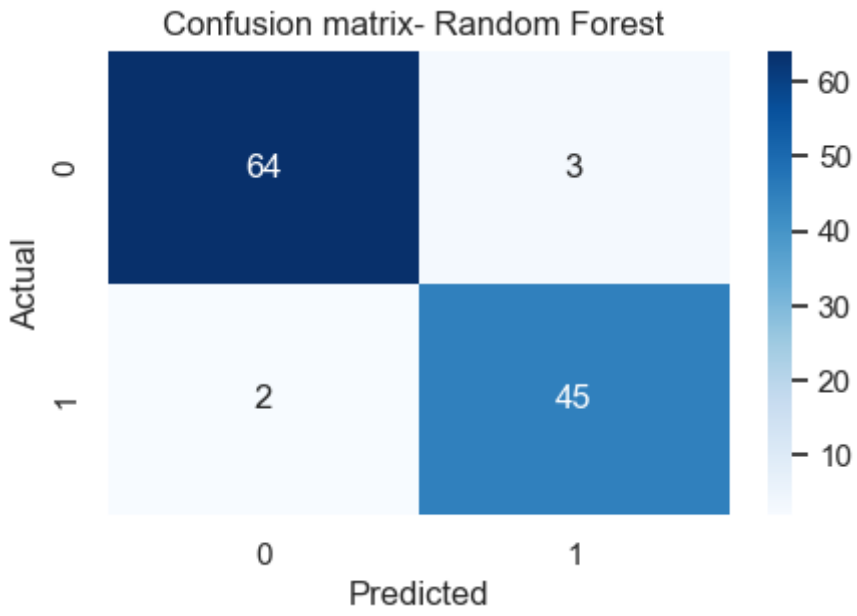
In [98]:

```
train_accuracy_rf = cross_val_score(rf,x_train, y_train, cv=10)
crossval_train_rf=train_accuracy_rf.mean()
test_accuracy_rf = cross_val_score(rf,x_test, y_test, cv=10)
crossval_test_rf=test_accuracy_rf.mean()
print('Random forest after Cross validation Train accuracy:', crossval_train_rf)
print('-------'*10)
print('Random forest  after Cross validation Test accuracy:', crossval_test_rf)
```

```
Random forest after Cross validation Train accuracy: 0.9582608695652175
----------------------------------------------------------------------
Random forest  after Cross validation Test accuracy: 0.931060606060606
```

In [53]:

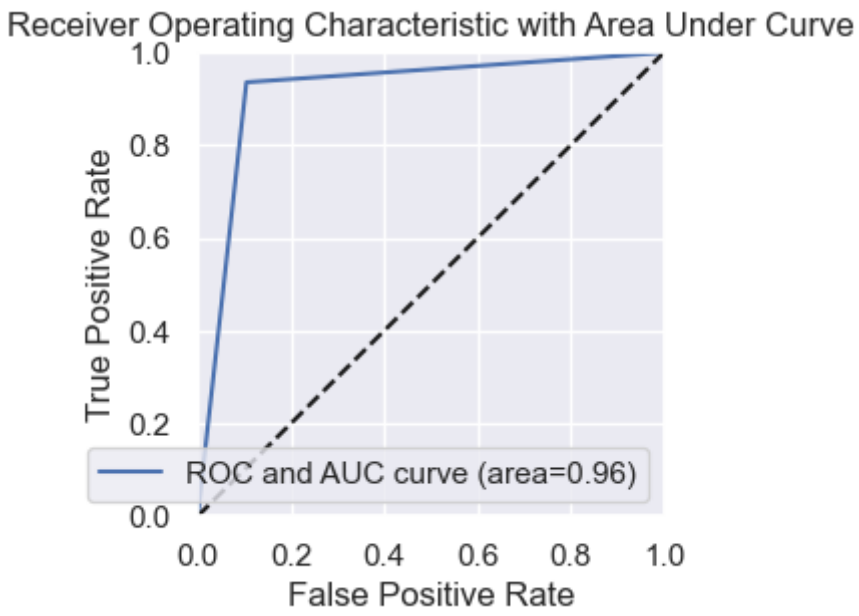
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_rf),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Random Forest ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [52]:

```
rf_roc_auc = roc_auc_score(y_test, y_pred_test_rf)
print(rf_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % rf_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()
```

```
0.9563353445538266
```



Model No. 5 - Naives Bayes

- The Naive Bayes algorithm is comprised of Naive and Bayes. It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features and it is called Bayes because it depends on the principle of Bayes' Theorem.

In [72]:

```
# Model building
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
# Predict the model
y_pred_train_nb = nb.predict(x_train)
y_pred_test_nb = nb.predict(x_test)
# Evaluate
accuracy_nb_test=accuracy_score(y_test,y_pred_test_nb)
accuracy_nb_train=accuracy_score(y_train,y_pred_train_nb)
print('Naive Bayes -Train accuracy:', accuracy_score(y_train, y_pred_train_nb))
print('-------'*10)
print('Naive Bayes -Test accuracy:', accuracy_score(y_test, y_pred_test_nb))
```

```
Naive Bayes -Train accuracy: 0.9472527472527472
----------------------------------------------------------------------
Naive Bayes -Test accuracy: 0.9035087719298246
```

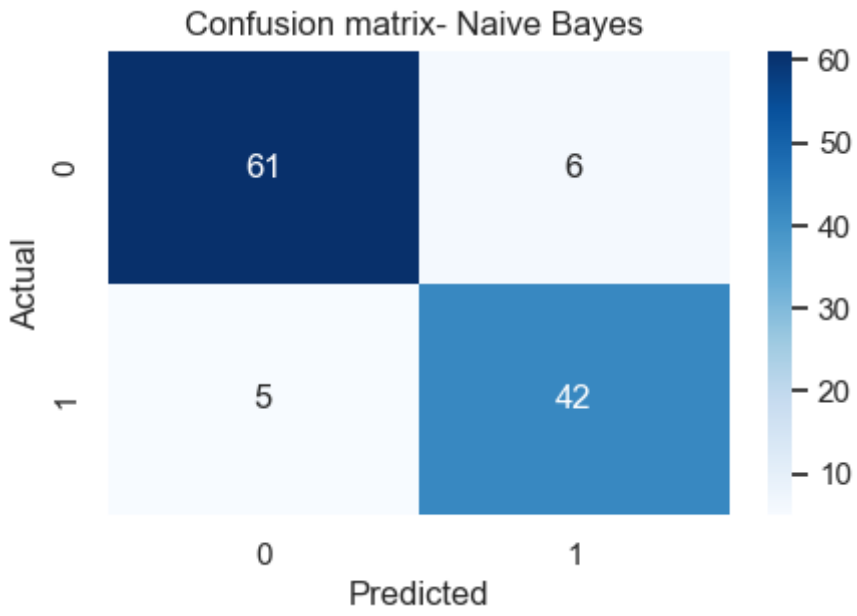
In [97]:

```
train_accuracy_nb = cross_val_score(nb,x_train, y_train, cv=10)
crossval_train_nb=train_accuracy_nb.mean()
test_accuracy_nb = cross_val_score(nb,x_test, y_test, cv=10)
crossval_test_nb=test_accuracy_nb.mean()
print('Naives Bayes after Cross validation Train accuracy:', crossval_train_nb)
print('---------------'*5)
print('Naives Bayes after Cross validation Test accuracy:', crossval_test_nb)
```

```
Naives Bayes after Cross validation Train accuracy: 0.9495652173913044
---------------------------------------------------------------------------
-
Naives Bayes after Cross validation Test accuracy: 0.8871212121212121
```

In [73]:

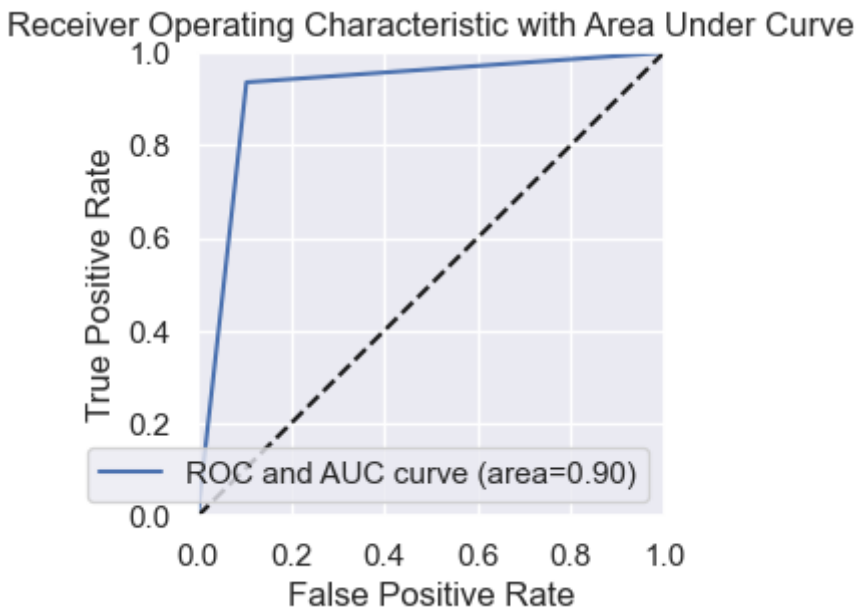
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_nb),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Naive Bayes ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [74]:

```
nb_roc_auc = roc_auc_score(y_test, y_pred_test_nb)
print(nb_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % nb_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()
```

```
0.9020323912353128
```



Model No. 6 - Support Vector Machine Model¶

- Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression.They are extremely popular because of their ability to handle multiple continuous and categorical variables.The objective of the support vector machine algorithm is to find a maximum marginal hyperplane.

In [56]:

```
# Radial Basis Function Kernel (RBF) - (Defaut SVM) Model building
from sklearn.svm import SVC
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
#Predict
y_pred_train_rbf = svm_rbf.predict(x_train)
y_pred_test_rbf = svm_rbf.predict(x_test)
#Evaluate
accuracy_rbf_test=accuracy_score(y_test,y_pred_test_rbf)
accuracy_rbf_train=accuracy_score(y_train,y_pred_train_rbf)
print('Rbf - SVM - Train accuracy:', accuracy_score(y_train, y_pred_train_rbf))
print('-------'*10)
print('Rbf - SVM - Test accuracy:', accuracy_score(y_test, y_pred_test_rbf))
```

```
Rbf - SVM - Train accuracy: 0.9846153846153847
----------------------------------------------------------------------
Rbf - SVM - Test accuracy: 0.9736842105263158
```

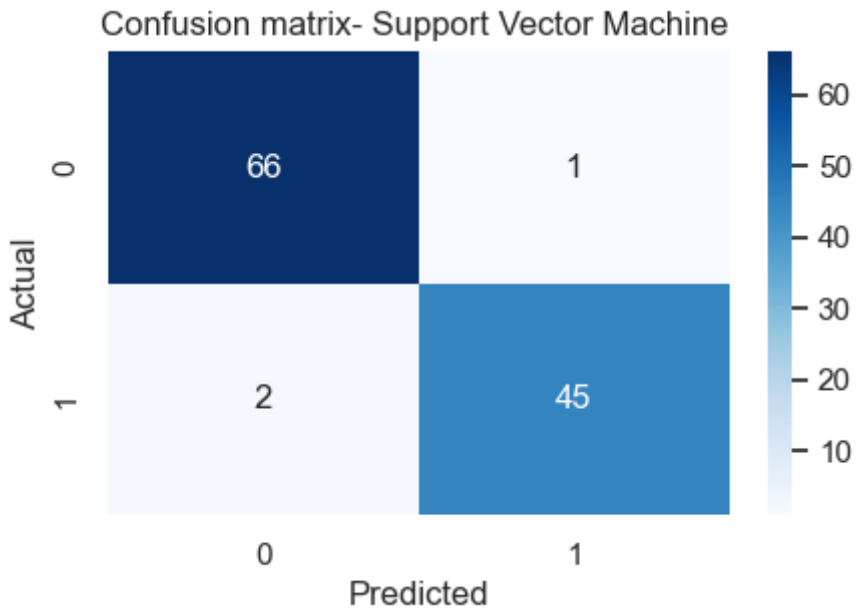
In [96]:

```
train_accuracy_rbf = cross_val_score(svm_rbf,x_train, y_train, cv=10)
crossval_train_rbf=train_accuracy_rbf.mean()
test_accuracy_rbf = cross_val_score(svm_rbf,x_test, y_test, cv=10)
crossval_test_rbf=test_accuracy_rbf.mean()
print('Rbf- SVM after Cross validation Train accuracy:', crossval_train_rbf)
print('---------------'*5)
print('Rbf- SVM after Cross validation Test accuracy:', crossval_test_rbf)
```

```
Rbf- SVM after Cross validation Train accuracy: 0.9758937198067633
----------------------------------------------------------------------
-
Rbf- SVM after Cross validation Test accuracy: 0.956060606060606
```

In [57]:

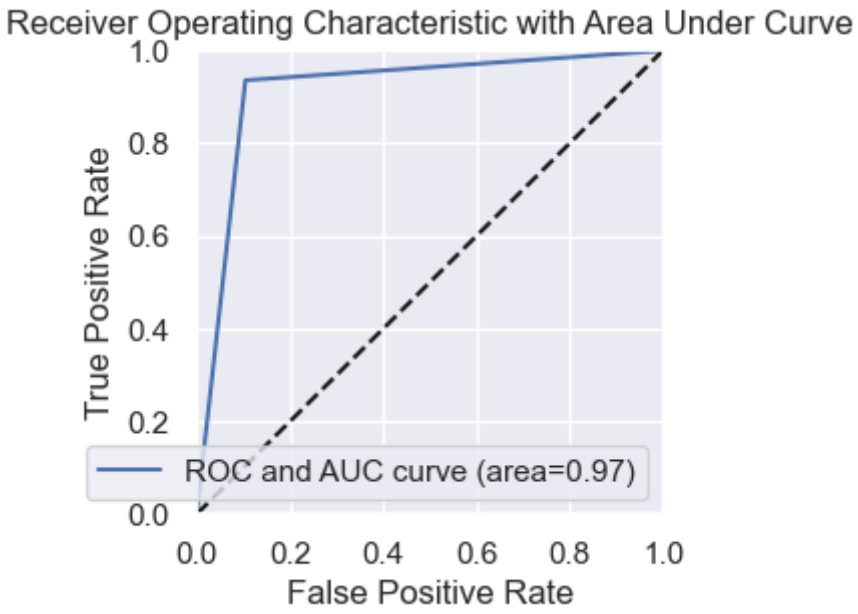
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_rbf),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Support Vector Machine ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [59]:

```
svm_roc_auc = roc_auc_score(y_test, y_pred_test_rbf)
print(svm_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % svm_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()
```

```
0.971260717688155
```



Model No. 7 - Extreme Gradient Boosting or XGBoost¶

- XGBoost is an implementation of gradient boosting that's designed for computational speed and scale. It leverages multiple cores on the CPU, allowing for learning to occur in parallel during training

In [75]:

```
# Model building
from xgboost import XGBClassifier
xgb = XGBClassifier()
xg=xgb.fit(x_train, y_train)
# Predict
y_pred_xg = xgb.predict(x_test)
y_pred_xg_train = xgb.predict(x_train)
# Evaluate
accuracy_xg_test=accuracy_score(y_test,y_pred_xg)
accuracy_xg_train=accuracy_score(y_train,y_pred_xg_train)
print('XGBoost Train accuracy:', accuracy_score(y_train, y_pred_xg_train))
print('-------'*5)
print('XGBoost Test accuracy:', accuracy_score(y_test, y_pred_xg))
```

```
XGBoost Train accuracy: 1.0
-----------------------------------
XGBoost Test accuracy: 0.9824561403508771
```

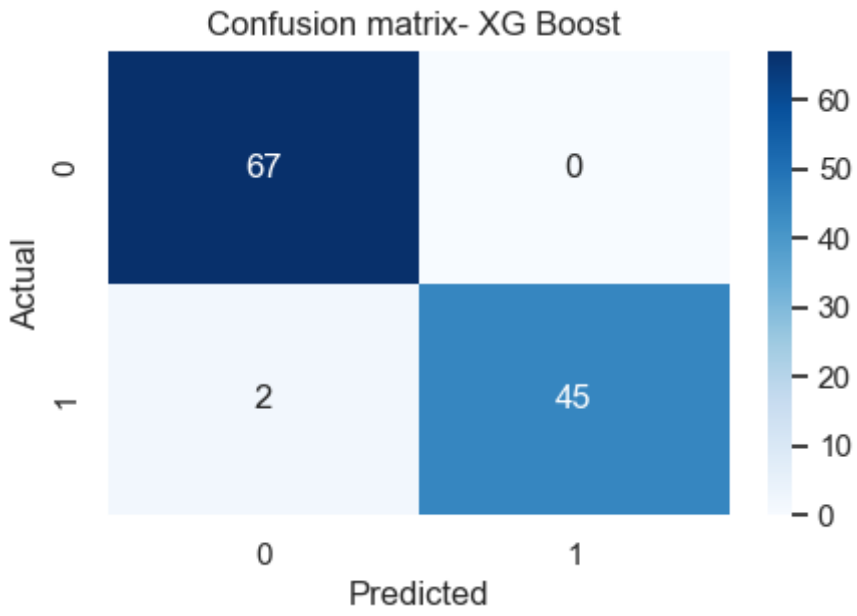
In [95]:

```
train_accuracy_xg= cross_val_score(xgb,x_train, y_train, cv=10)
crossval_train_xg=train_accuracy_xg.mean()
test_accuracy_xg = cross_val_score(xgb,x_test, y_test, cv=10)
crossval_test_xg=test_accuracy_xg.mean()
print('XGBoost Train accuracy after Cross validation:', crossval_train_xg)
print('---------------'*5)
print('XGBoost Test accuracy after Cross validation:', crossval_test_xg)
```

```
XGBoost Train accuracy after Cross validation: 0.9604347826086956
---------------------------------------------------------------------------
-
XGBoost Test accuracy after Cross validation: 0.9136363636363635
```

In [77]:

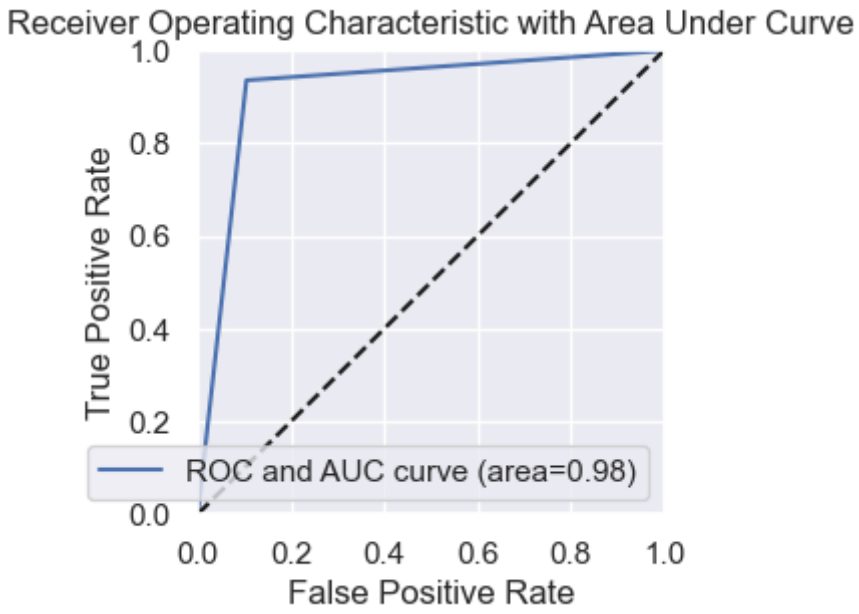
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_xg),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- XG Boost ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [76]:

```
xg_roc_auc = roc_auc_score(y_test, y_pred_xg)
print(xg_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % xg_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()
```

```
0.9787234042553192
```



Combining all models in Tabular format for better understanding

In [103]:

```
Models=['Logistic','Decision_tree','KNN','Random_forest','Naive_bayes','SVM','XGboost']
Trainacc=[accuracy_log_train,accuracy_dtree_train,accuracy_knn_train,accuracy_rf_train,a
          accuracy_rbf_train,accuracy_xg_train]
Testacc=[accuracy_log_test,accuracy_dtree_test,accuracy_knn_test,accuracy_rf_test,accura
         accuracy_rbf_test, accuracy_xg_test]
roc_auc_score=[logit_roc_auc,dtree_roc_auc,knn_roc_auc,rf_roc_auc,nb_roc_auc,svm_roc_auc
Cross_val_train=[crossval_train_log,crossval_train_dtree,crossval_train_knn, crossval_tr
                 crossval_train_rbf,crossval_train_xg]
Cross_val_test=[crossval_test_log,crossval_test_dtree,crossval_test_knn,crossval_test_rf
                crossval_test_rbf,crossval_test_xg]
```

In [106]:

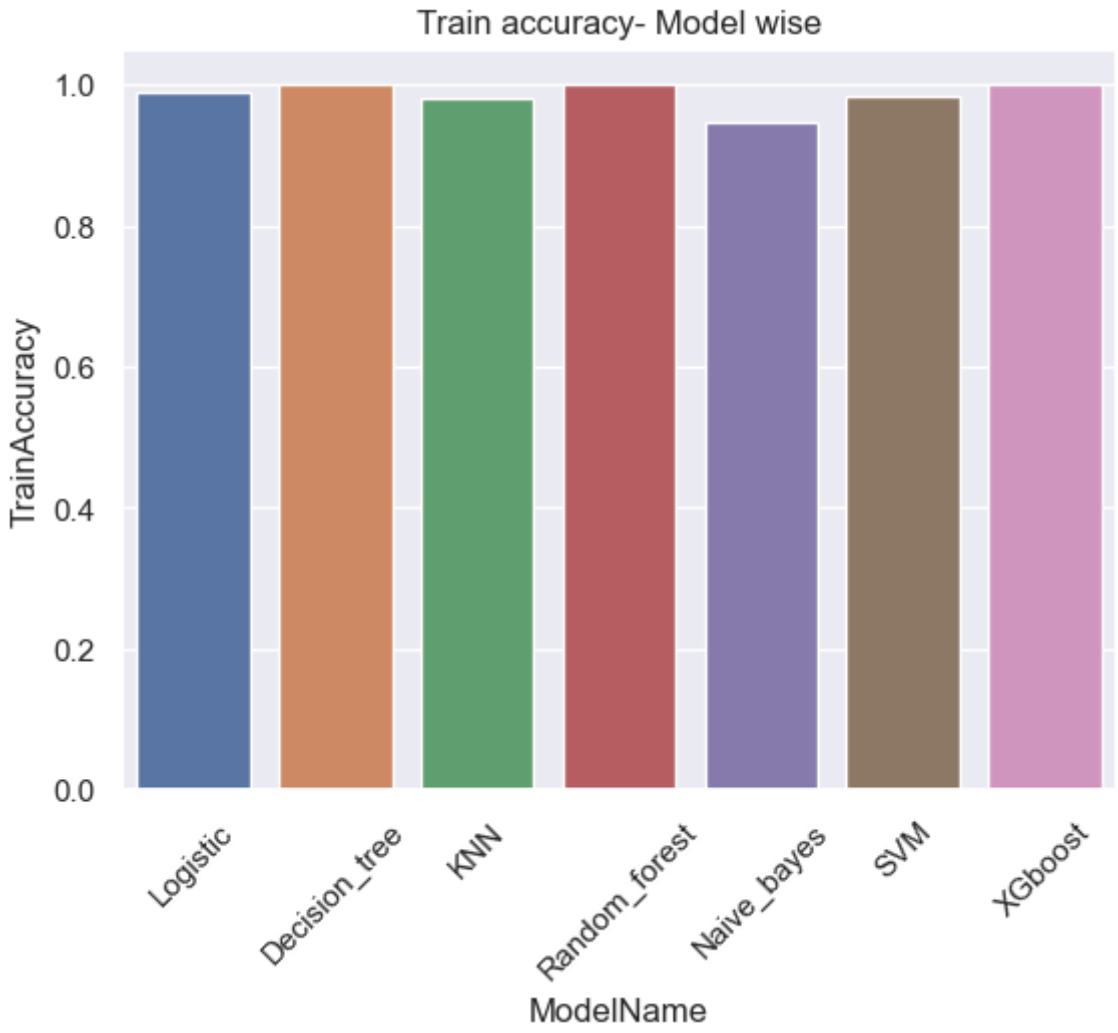
```
Combined_accuracy=pd.DataFrame({'ModelName':Models,'TrainAccuracy':Trainacc,'TestAccurac
                                'ROC-AUCScore':roc_auc_score,'TrainCrossval':Cross_val_t
print(Combined_accuracy)
```

```
       ModelName  TrainAccuracy  TestAccuracy  ROC-AUCScore  TrainCrossval
\
0       Logistic       0.989011      0.964912      0.963798       0.978068
1  Decision_tree       1.000000      0.912281      0.915846       0.929710
2            KNN       0.980220      0.956140      0.946809       0.954010
3  Random_forest       1.000000      0.956140      0.956335       0.958261
4    Naive_bayes       0.947253      0.903509      0.902032       0.949565
5            SVM       0.984615      0.973684      0.971261       0.975894
6        XGboost       1.000000      0.982456      0.978723       0.960435

   TestCrossval
0      0.981818
1      0.903788
2      0.939394
3      0.931061
4      0.887121
5      0.956061
6      0.913636
```

Accuracy Visualization

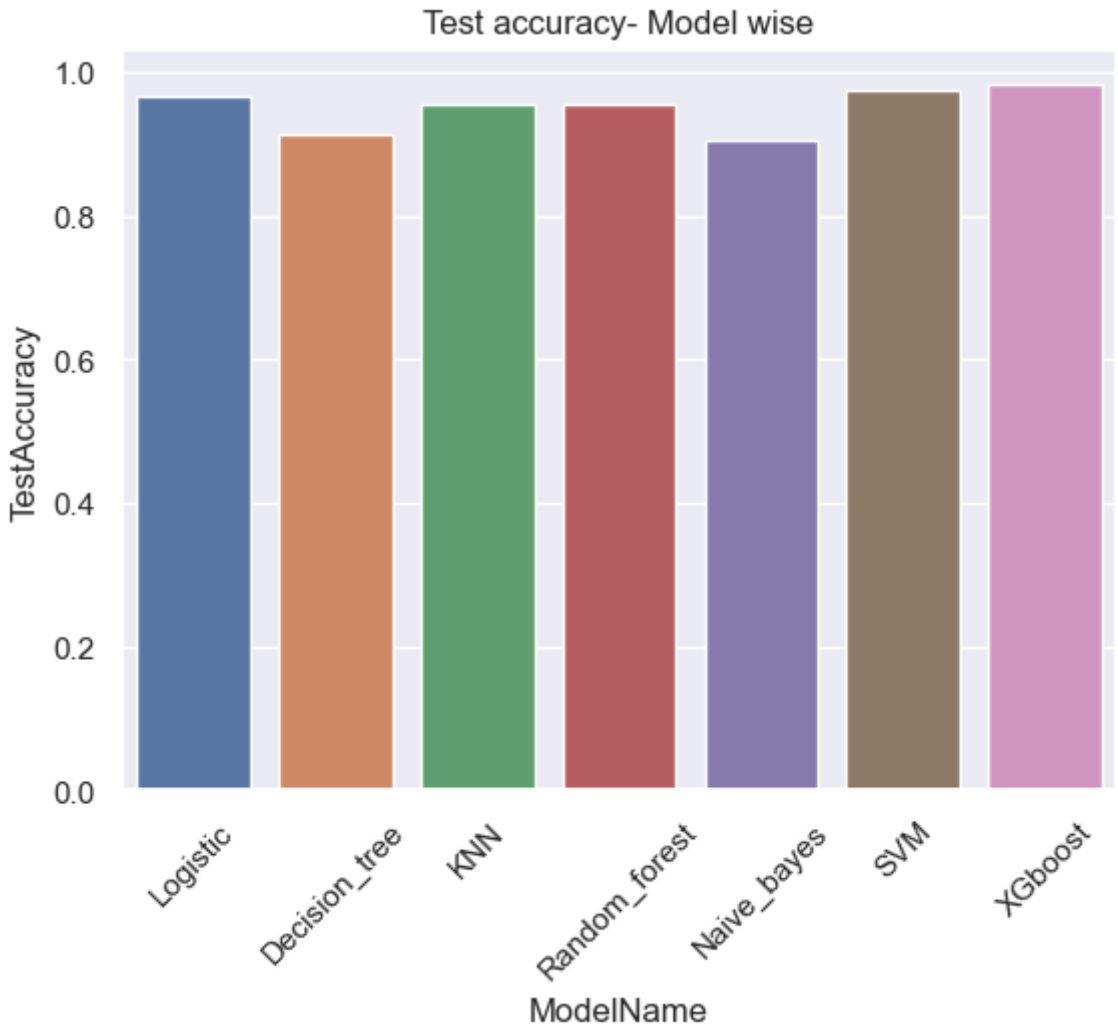
In [112]:

```
sns.barplot(x='ModelName',y='TrainAccuracy',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Train accuracy- Model wise')
plt.show()
```



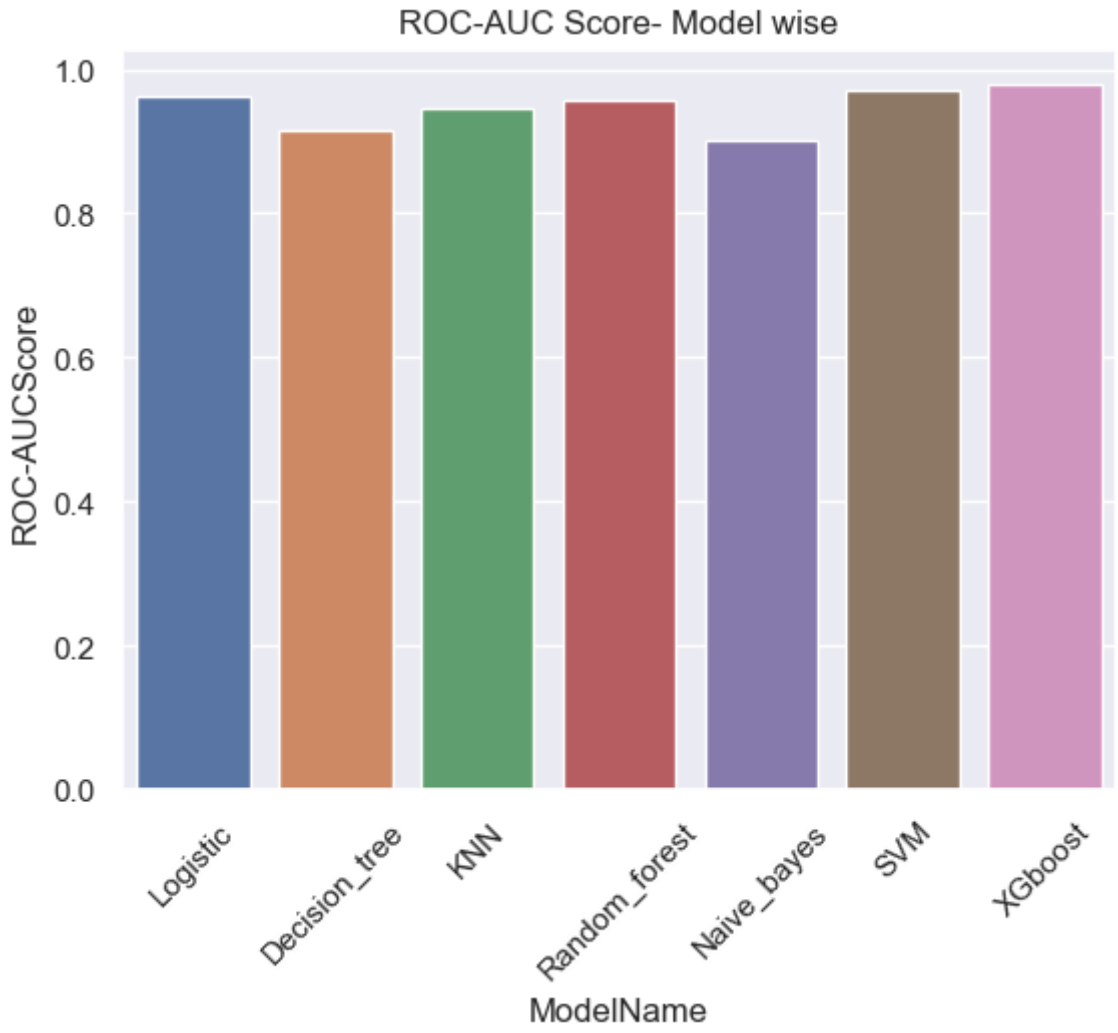
In [109]:

```
sns.barplot(x='ModelName',y='TestAccuracy',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Test accuracy- Model wise')
plt.show()
```



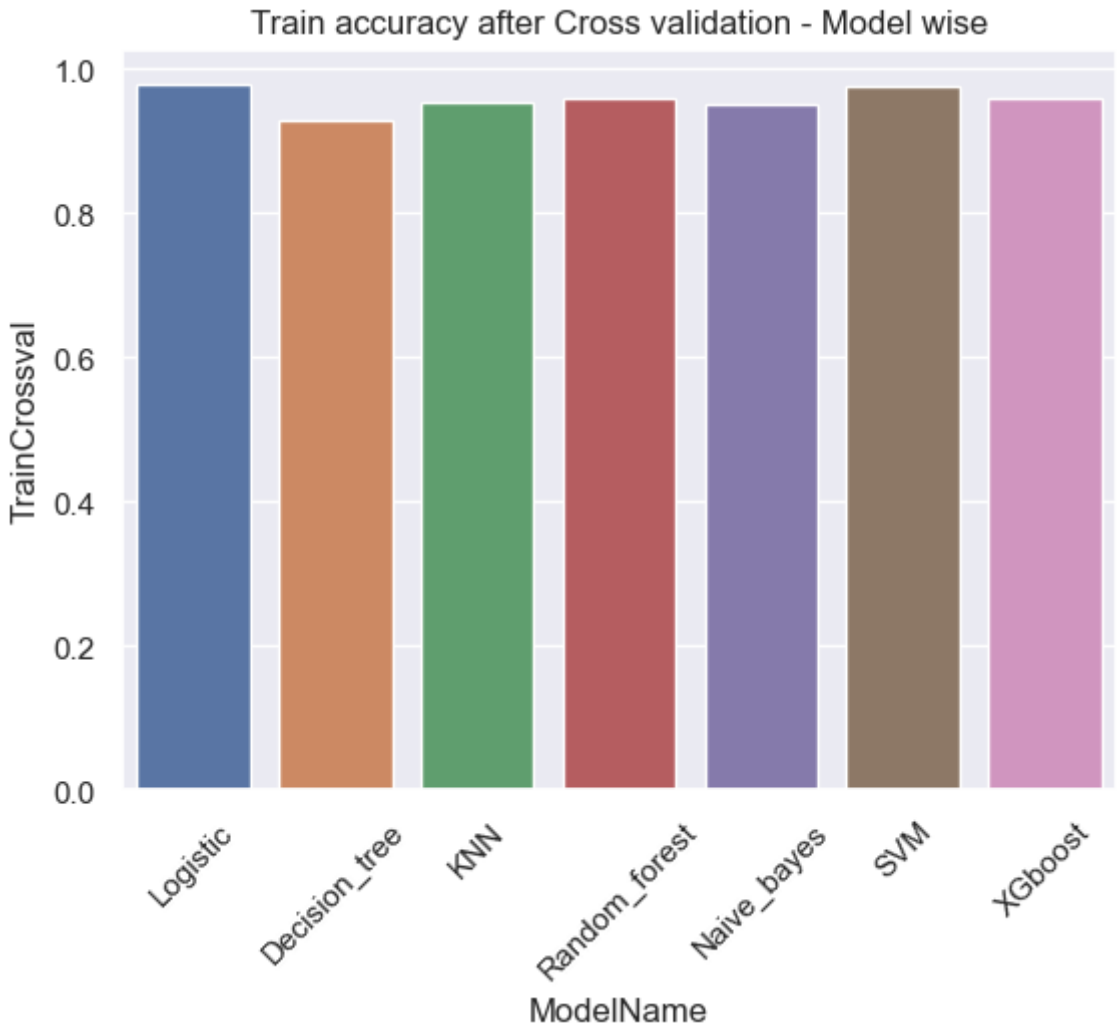
In [110]:

```
sns.barplot(x='ModelName',y='ROC-AUCScore',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('ROC-AUC Score- Model wise')
plt.show()
```



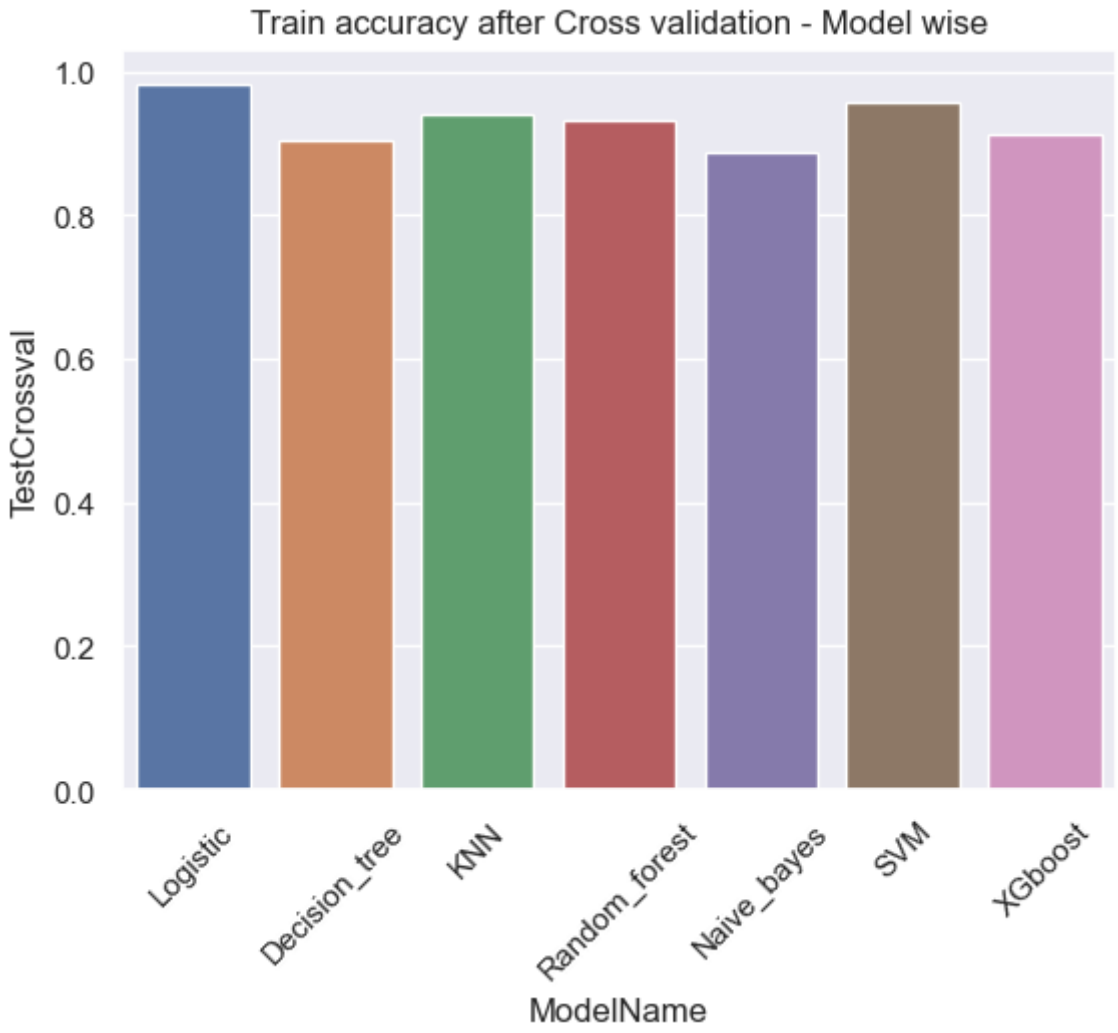
In [111]:

```
sns.barplot(x='ModelName',y='TrainCrossval',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Train accuracy after Cross validation - Model wise')
plt.show()
```



In [113]:

```
sns.barplot(x='ModelName',y='TestCrossval',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Train accuracy after Cross validation - Model wise')
plt.show()
```



Model No. 8 Voting ensemble¶

- Voting is an ensemble method that combines the performances of multiple models to make predictions.

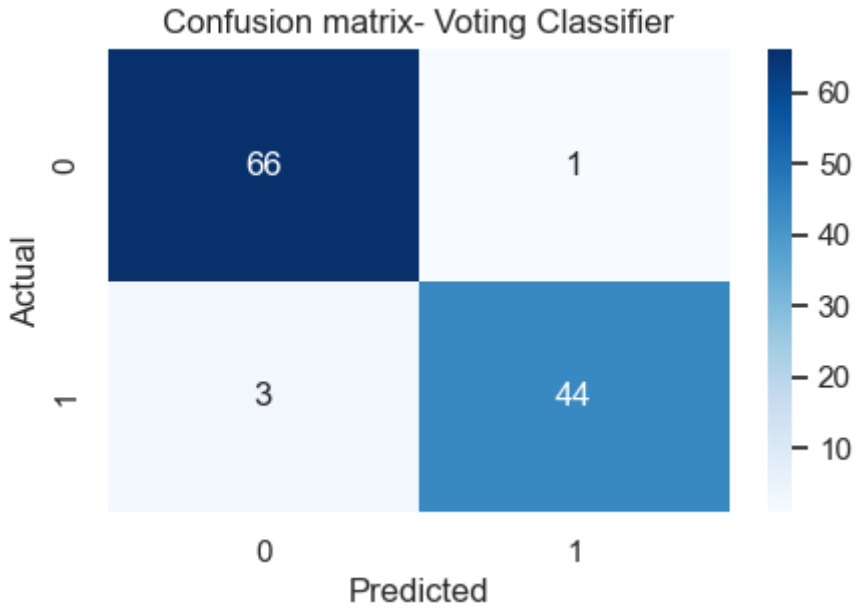
In [78]:

```
from sklearn.ensemble import VotingClassifier
evc=VotingClassifier(estimators = [('Logistic',logit),('Decision_tree', dtree),('KNN',kn
                                   ('NaiveBayes',nb), ('SVM_RBF' ,svm_rbf),('Extra_Gradi
evc_model=evc.fit(x_train,y_train)
evc_pred=evc.predict(x_test)
evc_pred_train=evc.predict(x_train)

accuracy_evc=accuracy_score(y_test,evc_pred)
accuracy_evc_train=accuracy_score(y_train,evc_pred_train)

print('Voting ensemble train accuracy:', accuracy_score(y_train, evc_pred_train))
print('---------------'*5)
print('Voting ensemble train accuracy:', accuracy_score(y_test, evc_pred))
```

```
Voting ensemble train accuracy: 0.9912087912087912
--------------------------------------------------------------------------------
-
Voting ensemble train accuracy: 0.9649122807017544
```

In [79]:

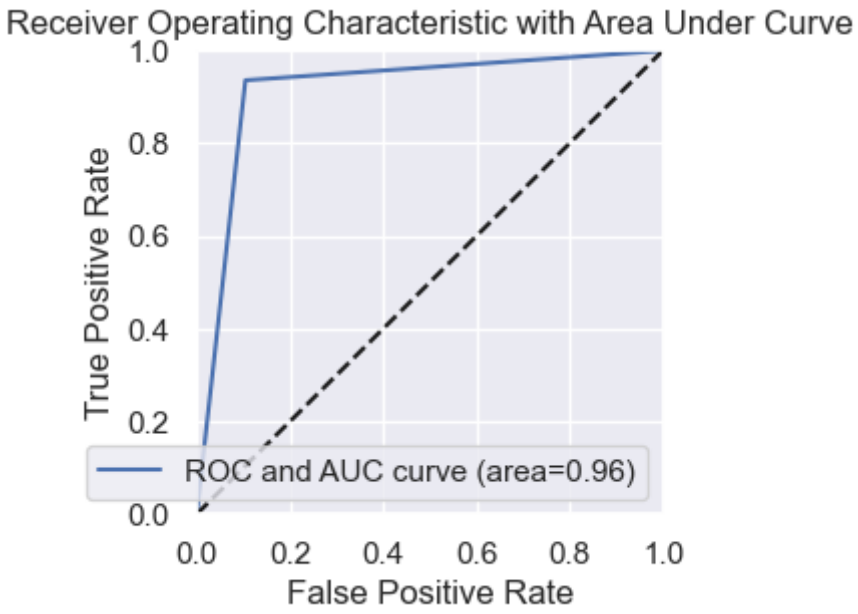
```
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,evc_pred),cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Voting Classifier")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



In [80]:

```
evc_roc_auc = roc_auc_score(y_test, evc_pred)
print(evc_roc_auc)
plt.figure(figsize=(3,3))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % evc_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()
```

```
0.9606224198158145
```



Conclusion

- I used 8 Supervised machine learning models to solve the classification problem performed on 569 observations of the dataset. The objective was to build machine learning model to classify whether the breast cancer is benign or malignant.
- For Evaluation I used cross validation, accuracy score, roc-auc score & confusion matrix.
- All the models yielded more than 90% accuracy.
- We can safely say that all the models are good to predict the classification in this datset.
- At last combined all models and performed Voting ensemble method which yielded Train accuracy of 99% & test accuracy of 96%.

In [ ]:

In [ ]:

```

```

In [ ]:

```

```