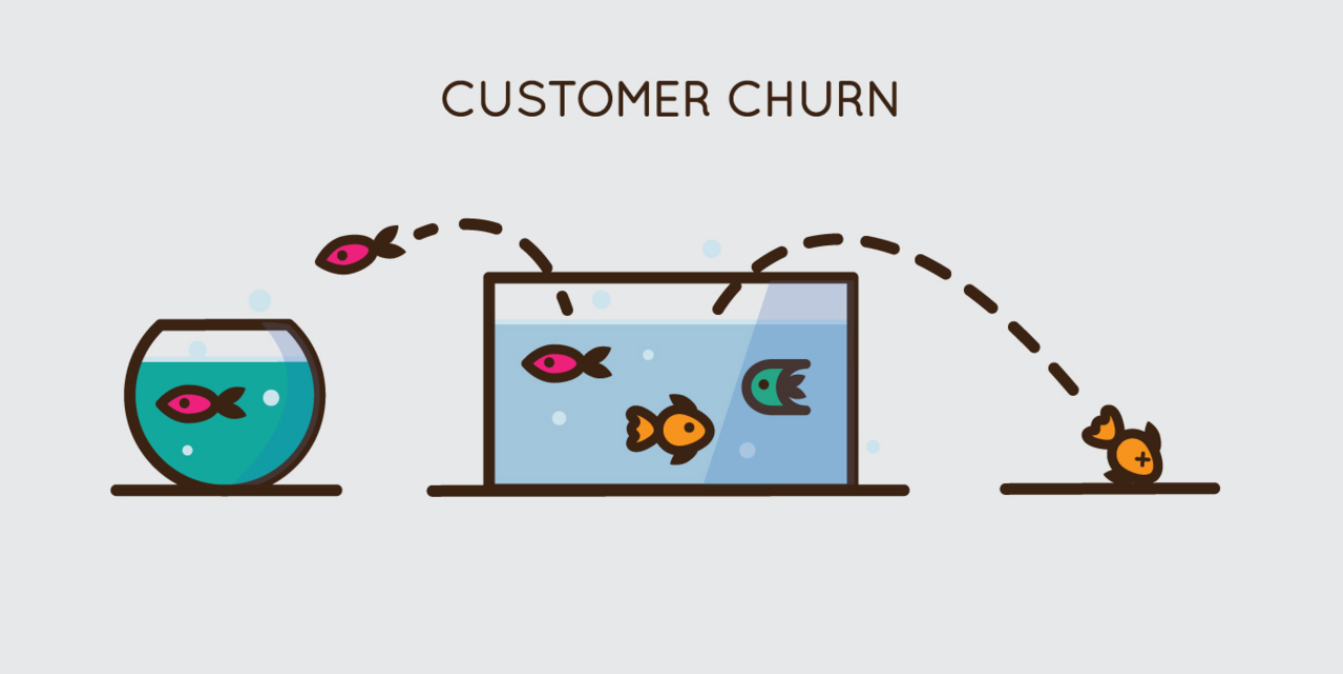
TASK 3- CUSTOMER CHURN PREDICTION DATASET



Objective

- The objective is to develop a model to predict customer churn. For this I will be using 11 Supervised machine learning algorithms.

Machine learning models applied

- Logistic Regression
- Decision Tree
- K Nearest Neighbors
- Bagging (or Bootstrap aggregating)
- Random Forest
- Naive Bayes
- Support vector machines (SVMs)
- Adaptive boosting or AdaBoost
- Gradient boosting
- XGBoost
- Voting classifier

Dataset source & brief

- The dataset has been sourced from kaggle and it contains details of a bank's customers and the target variable is a binary variable reflecting the fact whether the customer left the bank (closed his account) or he continues to be a customer.

Project outline

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

Import the basic 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 & Read the dataset

In [2]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Codsoft\3. Churn_Modelling.csv")
df.head()
```

Out[2]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bala
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8380
2	3	15619304	Onio	502	France	Female	42	8	15966
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12551

Basic info about the dataset

In [3]:

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

Out[3]:

```
(10000, 14)
```

- Dataset has 100000 rows and 14 columns

In [4]:

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

Out[4]:

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [5]:

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

Out[5]:

```
0
```

- No duplicates present

In [6]:

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

Out[6]:

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

- No null values present

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

- All are numerical columns except Surname, Geography & Gender

In [8]:

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

Out[8]:

	count	mean	std	min	
RowNumber	10000.000000	5000.500000	2886.895680	1.000000	2500.75
CustomerId	10000.000000	15690940.569400	71936.186123	15565701.000000	15628528.25
CreditScore	10000.000000	650.528800	96.653299	350.000000	584.00
Age	10000.000000	38.921800	10.487806	18.000000	32.00
Tenure	10000.000000	5.012800	2.892174	0.000000	3.00
Balance	10000.000000	76485.889288	62397.405202	0.000000	0.00
NumOfProducts	10000.000000	1.530200	0.581654	1.000000	1.00
HasCrCard	10000.000000	0.705500	0.455840	0.000000	0.00
IsActiveMember	10000.000000	0.515100	0.499797	0.000000	0.00
EstimatedSalary	10000.000000	100090.239881	57510.492818	11.580000	51002.11
Exited	10000.000000	0.203700	0.402769	0.000000	0.00

In [9]:

```
df.describe(include='object').T  # Analysis on Categorical Columns
```

Out[9]:

	count	unique	top	freq
Surname	10000	2932	Smith	32
Geography	10000	3	France	5014
Gender	10000	2	Male	5457

Dropping non significant variables

In [10]:

```
df.drop(['RowNumber','CustomerId','Surname'],axis=1, inplace=True)
```

In [11]:

```
df.shape
```

Out[11]:

```
(10000, 11)
```

Segregation of Numerical and Categorical Variables/Columns

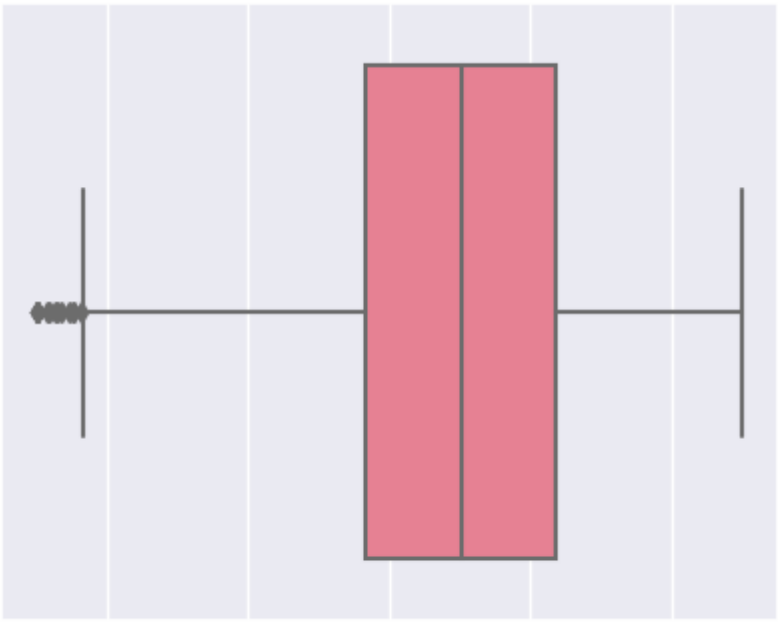
In [12]:

```
categorical_col = df.select_dtypes(include = ['object']).columns
numerical_col = df.select_dtypes(exclude = ['object']).columns
```

Checking for Outliers

In [13]:

```
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)
```



Exploratory Data Analysis

In [14]:

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

```
  0%|
| 0/1519 [00:00<…
```

Out[14]:

Data Report

Data Report Overview

Variables ≡

CreditScore Geography Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember EstimatedSalary Exited

Interactions Correlations Missing Values

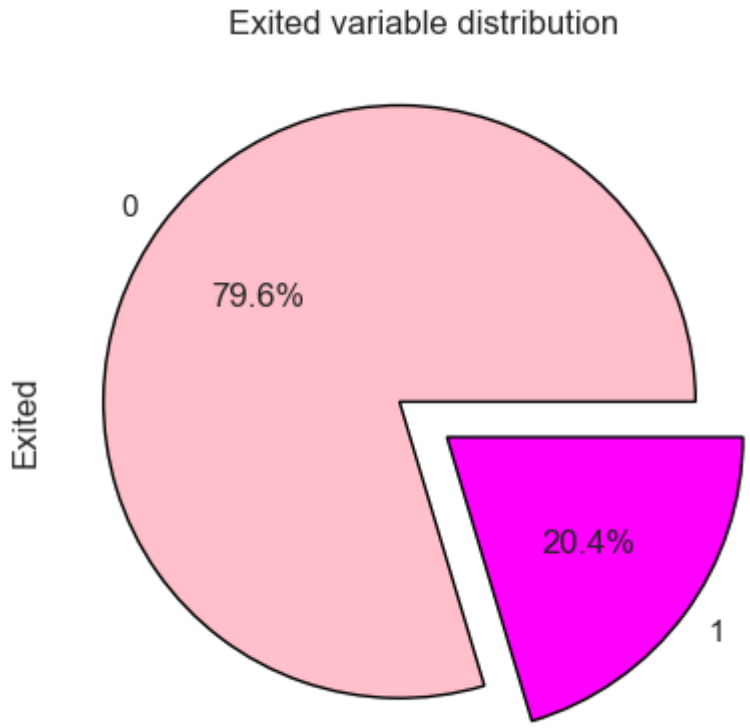
Overview

Dataset Statistics

Number of Variables	11
Number of Rows	10000

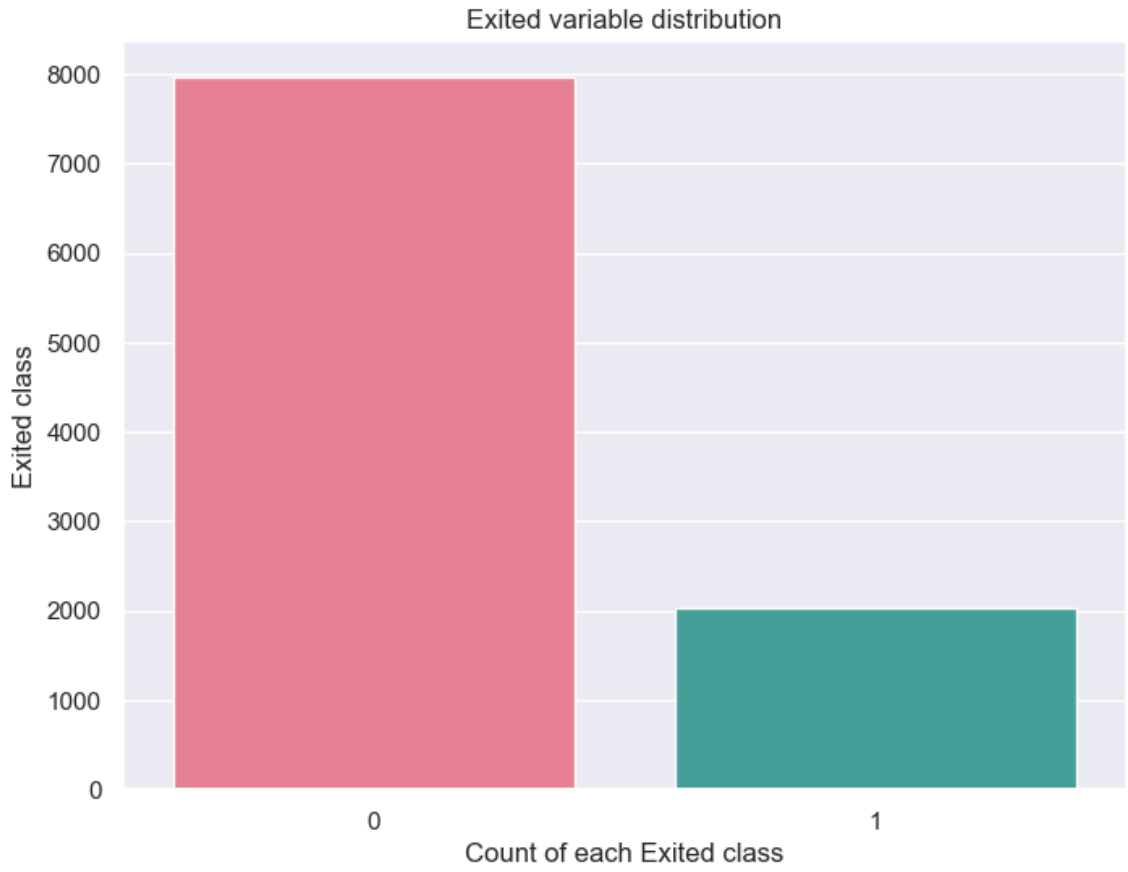
In [15]:

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



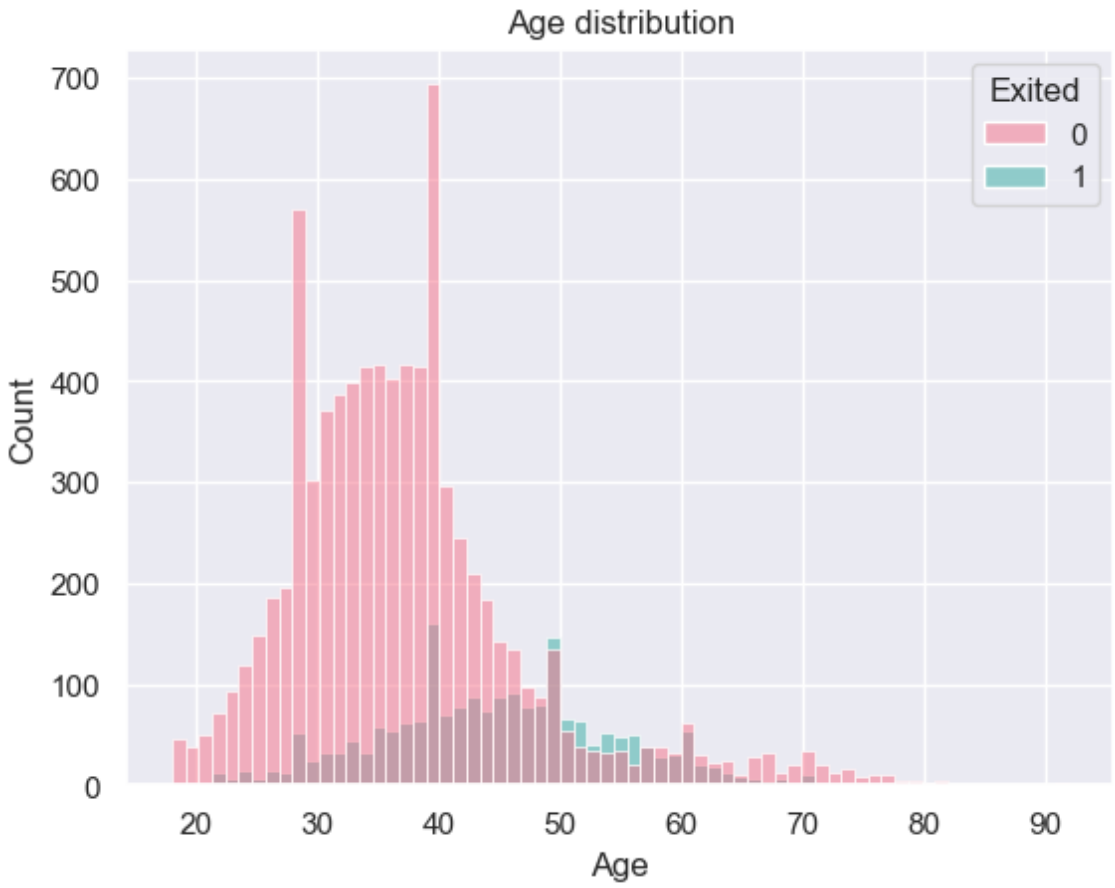
In [16]:

```
plt.figure(figsize=(8,6))
sns.countplot(x='Exited',data=df, palette="husl")
plt.xlabel("Count of each Exited class")
plt.ylabel("Exited class")
plt.title('Exited variable distribution')
plt.show()
```



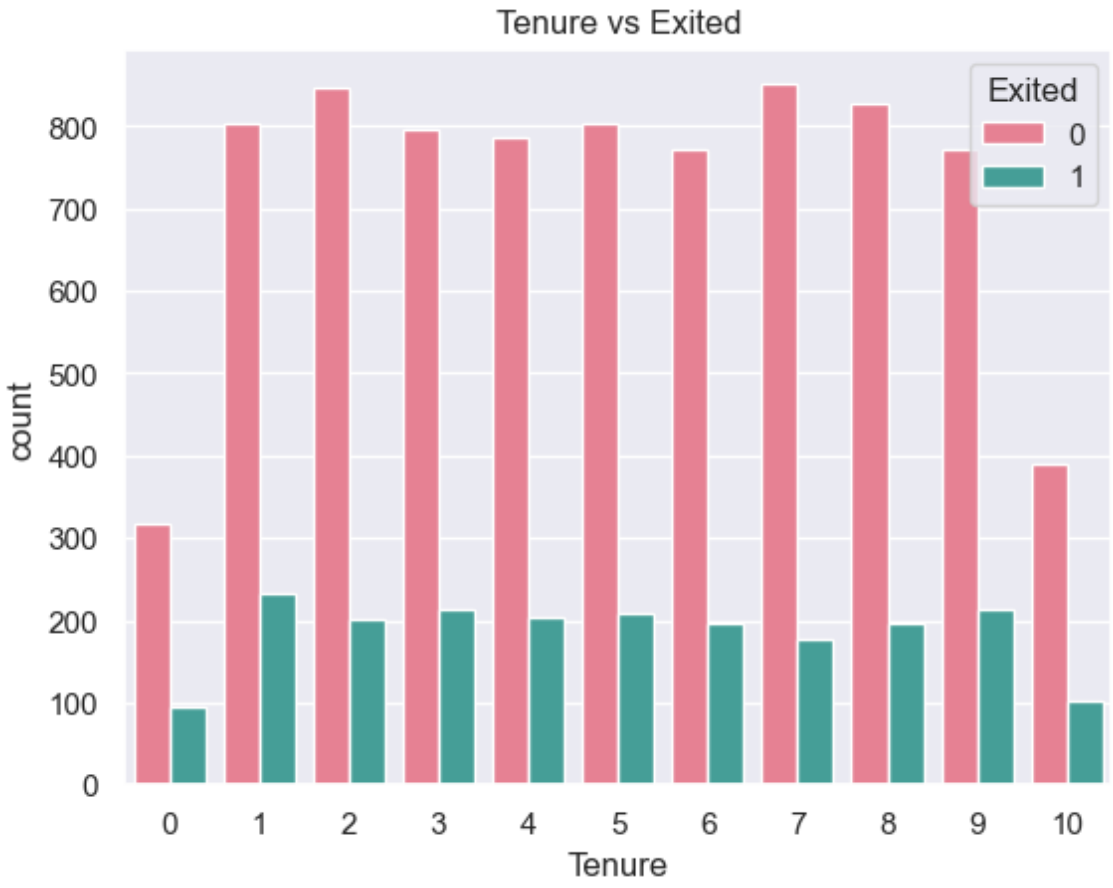
In [17]:

```
sns.histplot(x='Age',hue='Exited', data=df, palette='husl')
plt.title('Age distribution')
plt.show()
```



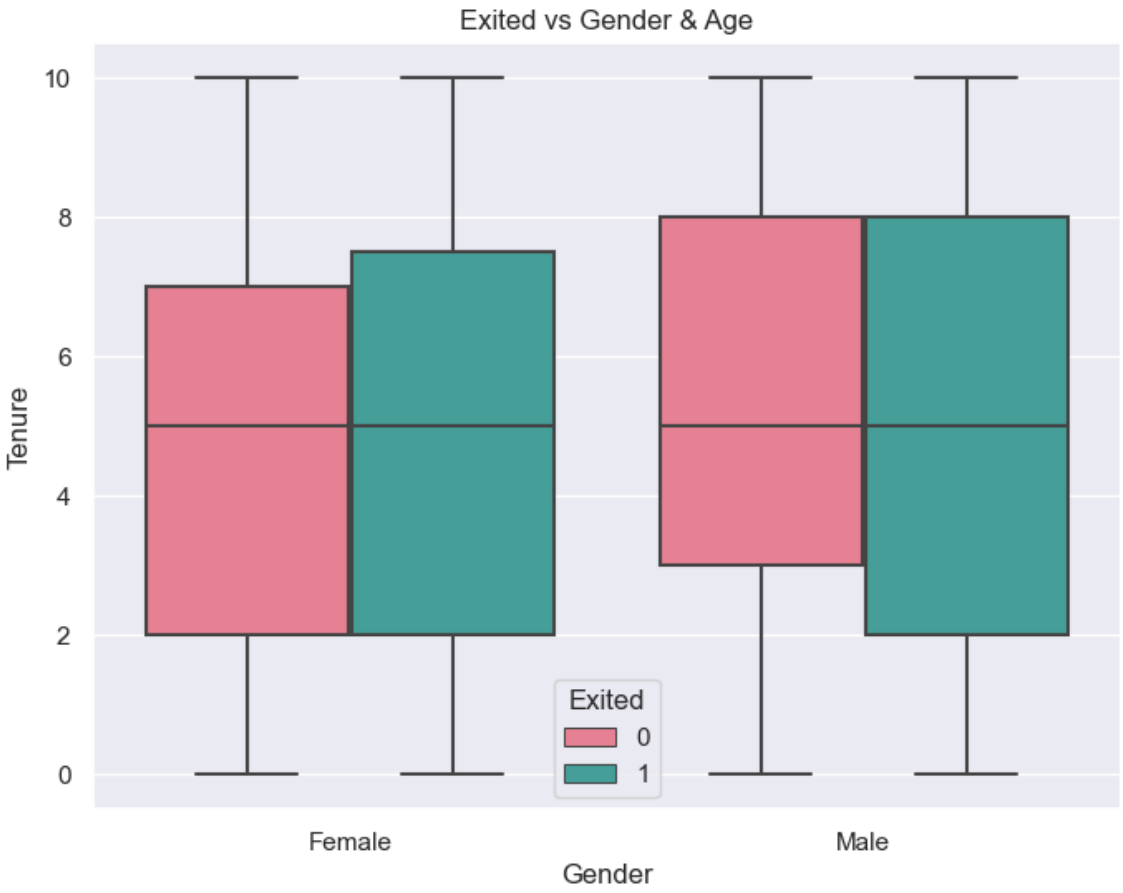
In [18]:

```
sns.countplot(x='Tenure',hue='Exited', data=df, palette='husl')
plt.title('Tenure vs Exited')
plt.show()
```



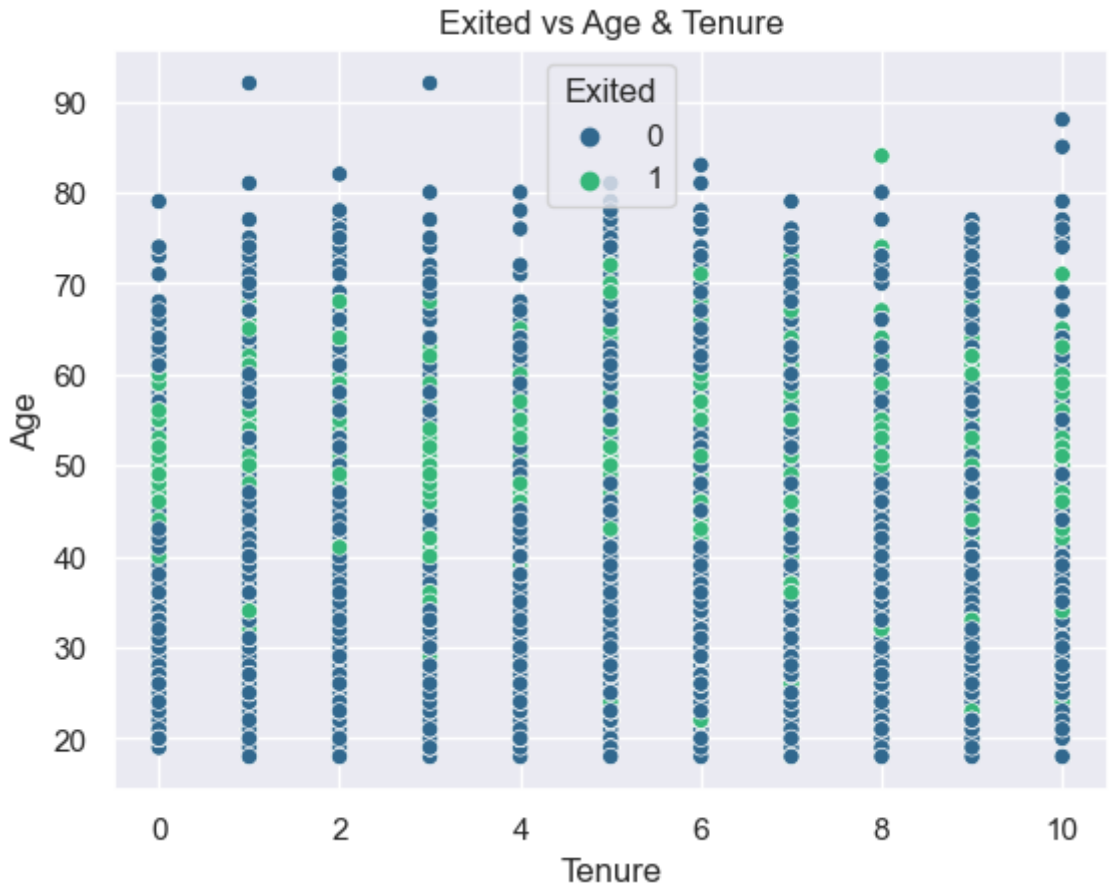
In [83]:

```
plt.figure(figsize=(8,6))
sns.boxplot(x='Gender',y='Tenure',hue='Exited',data=df ,palette='husl')
plt.title('Exited vs Gender & Age')
plt.show()
```



In [84]:

```
sns.scatterplot(data=df, x='Tenure', y='Age', hue = 'Exited', palette= 'viridis')
plt.title('Exited vs Age & Tenure')
plt.show()
```

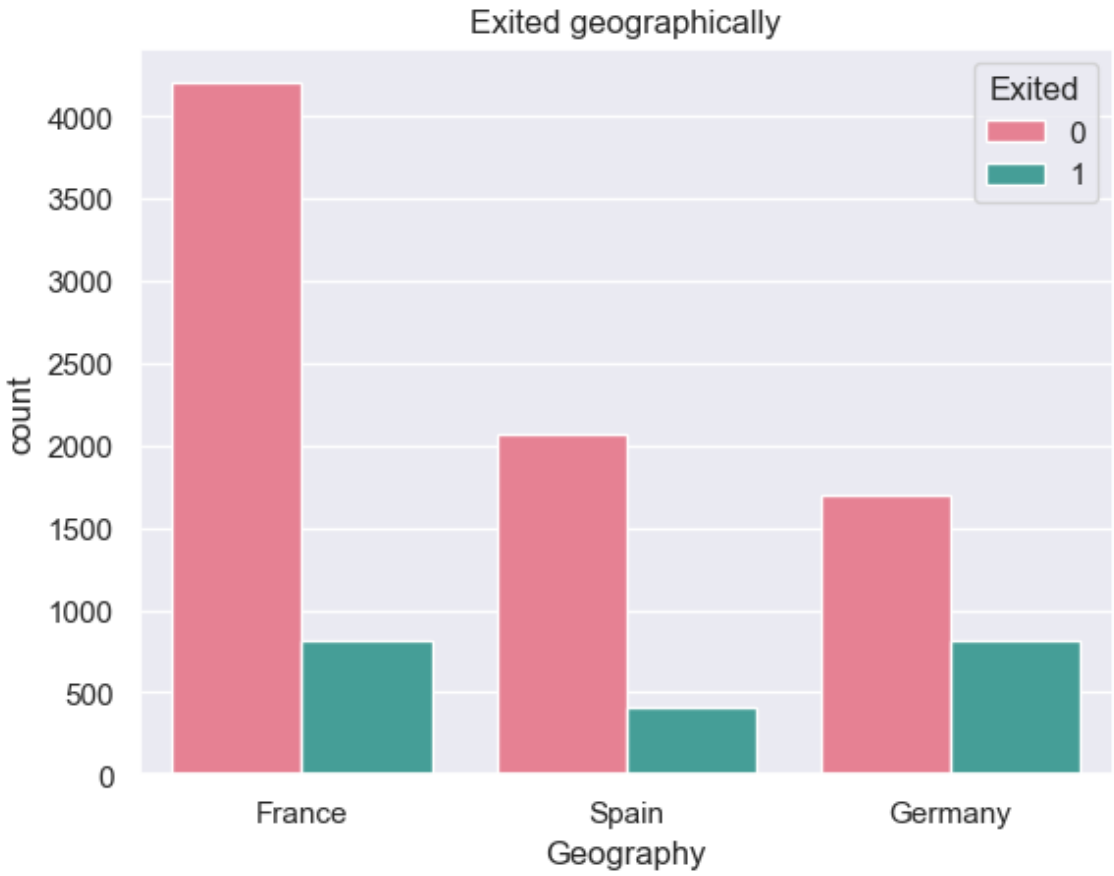


In [88]:

```
plt.figure(figsize=(8,6))
sns.swarmplot(data=df, x='Gender', y='Age', hue = 'Exited', palette='husl')
plt.title('Exited vs Gender')
plt.show()
```

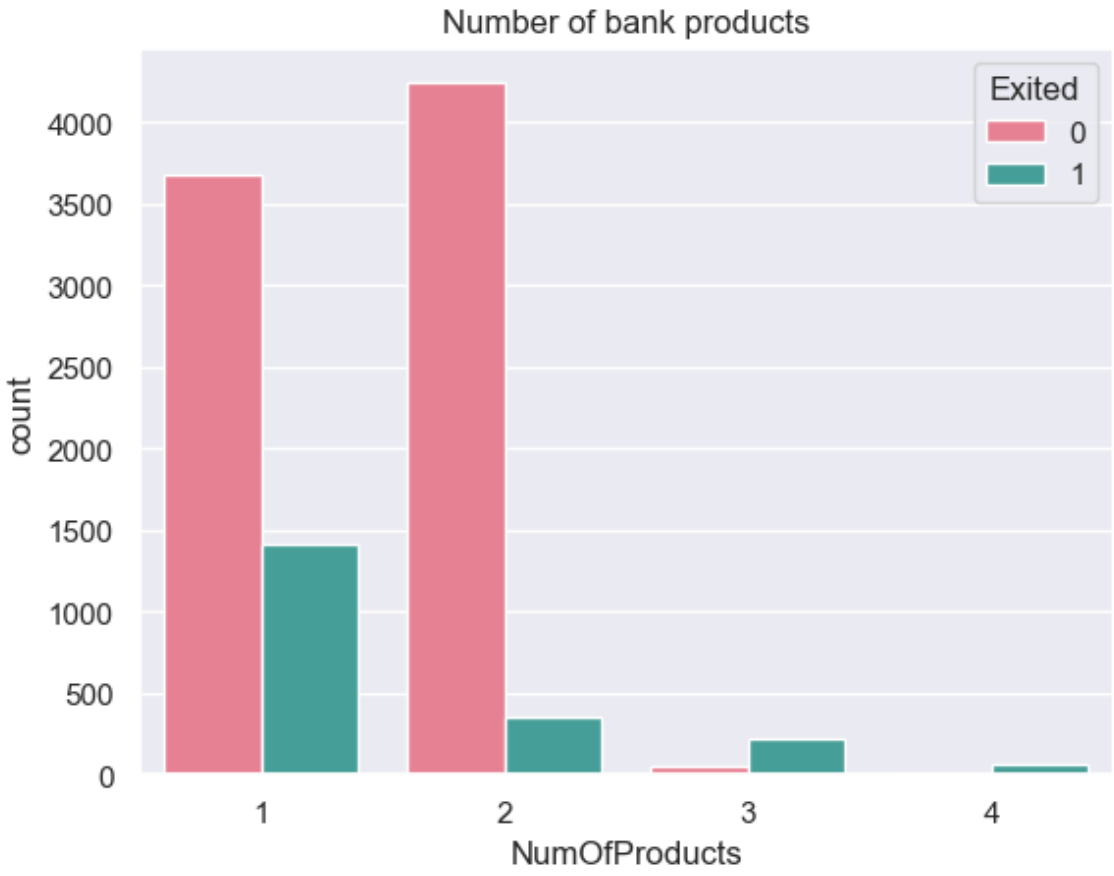
In [93]:

```
sns.countplot(x='Geography',hue='Exited', data=df,palette='husl')
plt.title('Exited geographically')
plt.show()
```



In [104]:

```
sns.countplot(x='NumOfProducts',hue='Exited', data=df,palette='husl')
plt.title('Number of bank products')
plt.show()
```

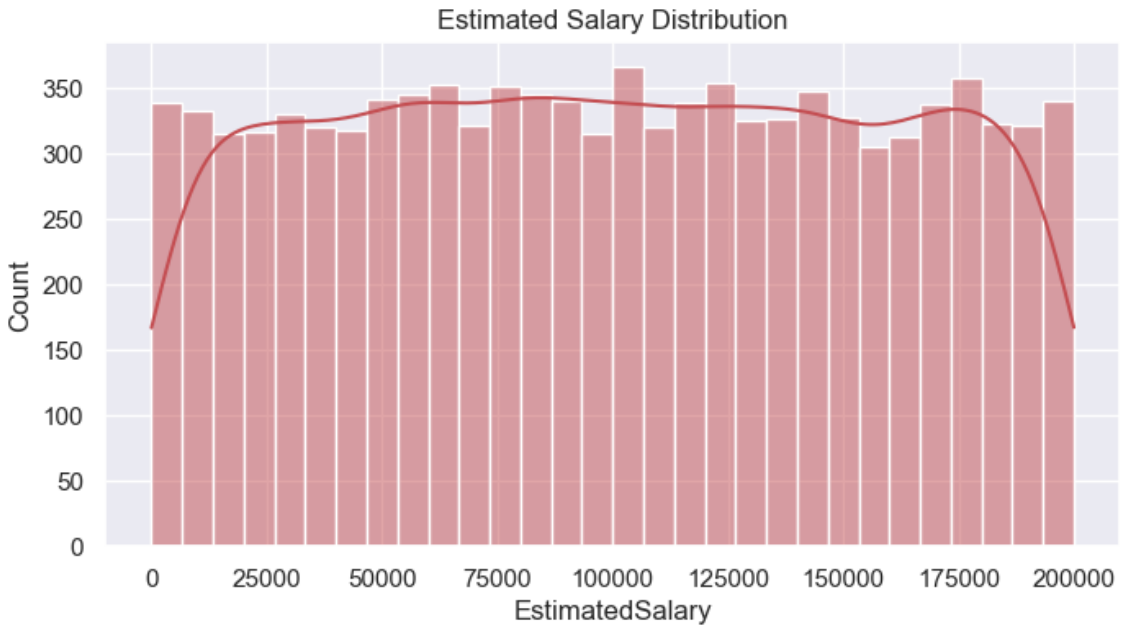


In [108]:

```
sns.barplot(y='HasCrCard',x='IsActiveMember',hue='Exited',data=df, palette='husl')
plt.title('Active membership & Credit Card access')
plt.show()
```

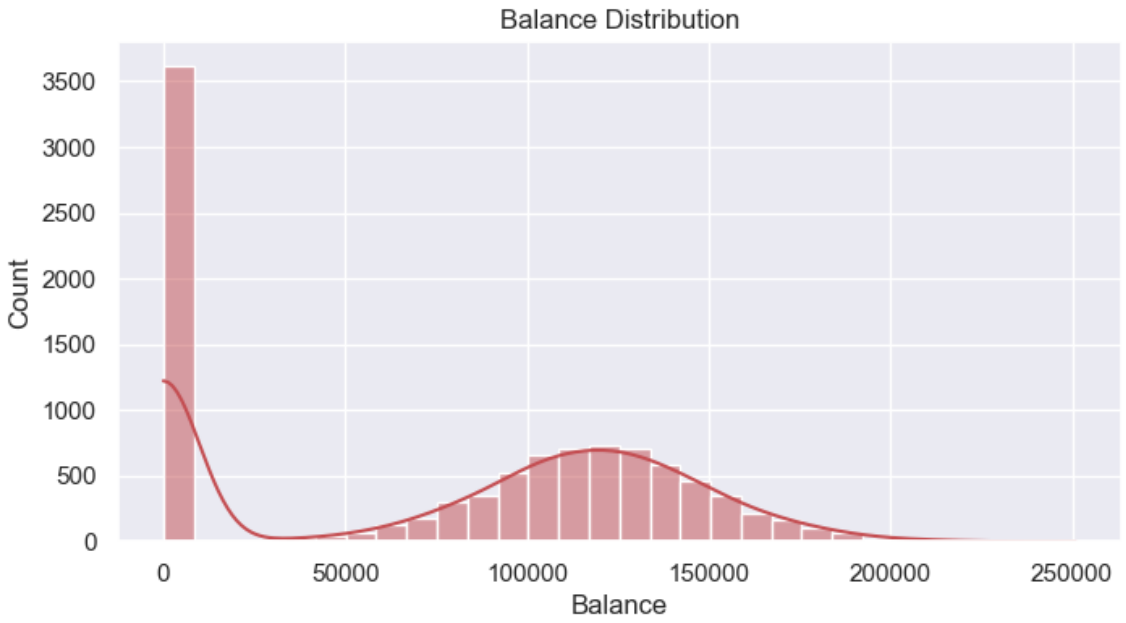
In [111]:

```
plt.figure(figsize=[8,4])
sns.histplot(df.EstimatedSalary, color='r', bins=30, kde=True)
plt.title('Estimated Salary Distribution')
plt.show()
```



In [115]:

```
plt.figure(figsize=[8,4])
sns.histplot(df.Balance, color='r', bins=30, kde=True)
plt.title('Balance Distribution')
plt.show()
```

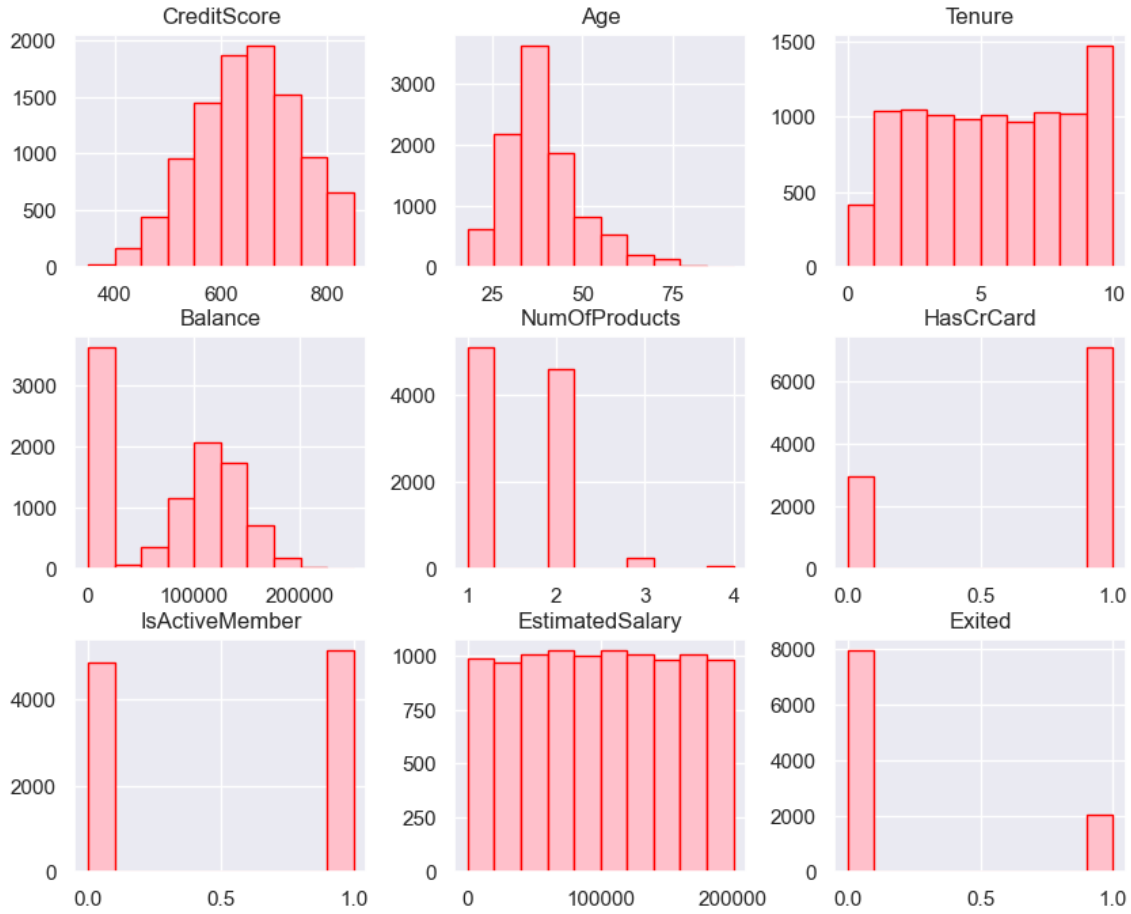


In [116]:

```
sns.scatterplot(data=df, x='EstimatedSalary', y='Balance', hue = 'Exited', palette= 'hus
plt.title('Exited vs Salary & Balance')
plt.show()
```

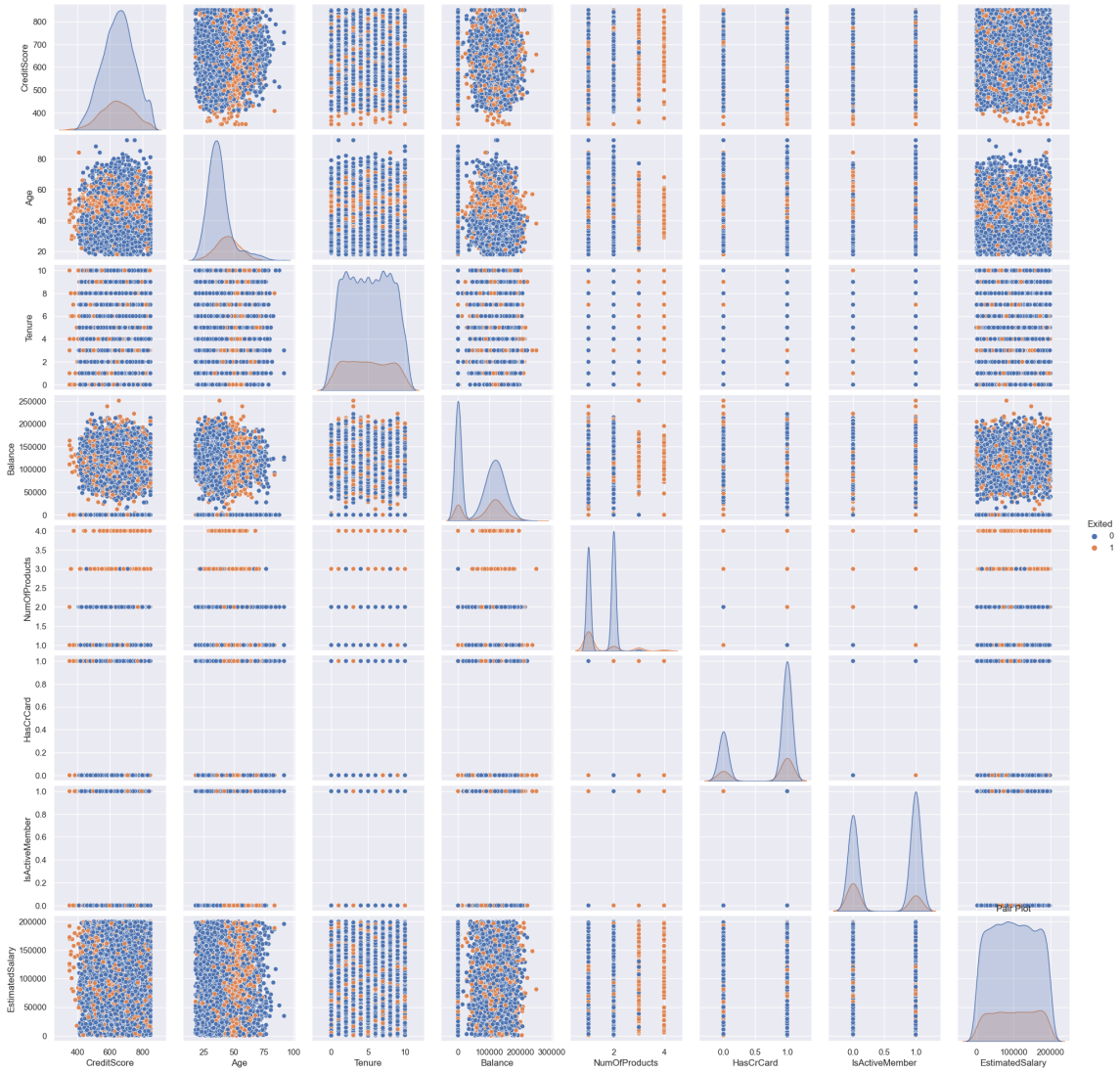
In [98]:

```
df.hist(bins=10, figsize=(10,8),color='pink', edgecolor='red')
plt.show()
```



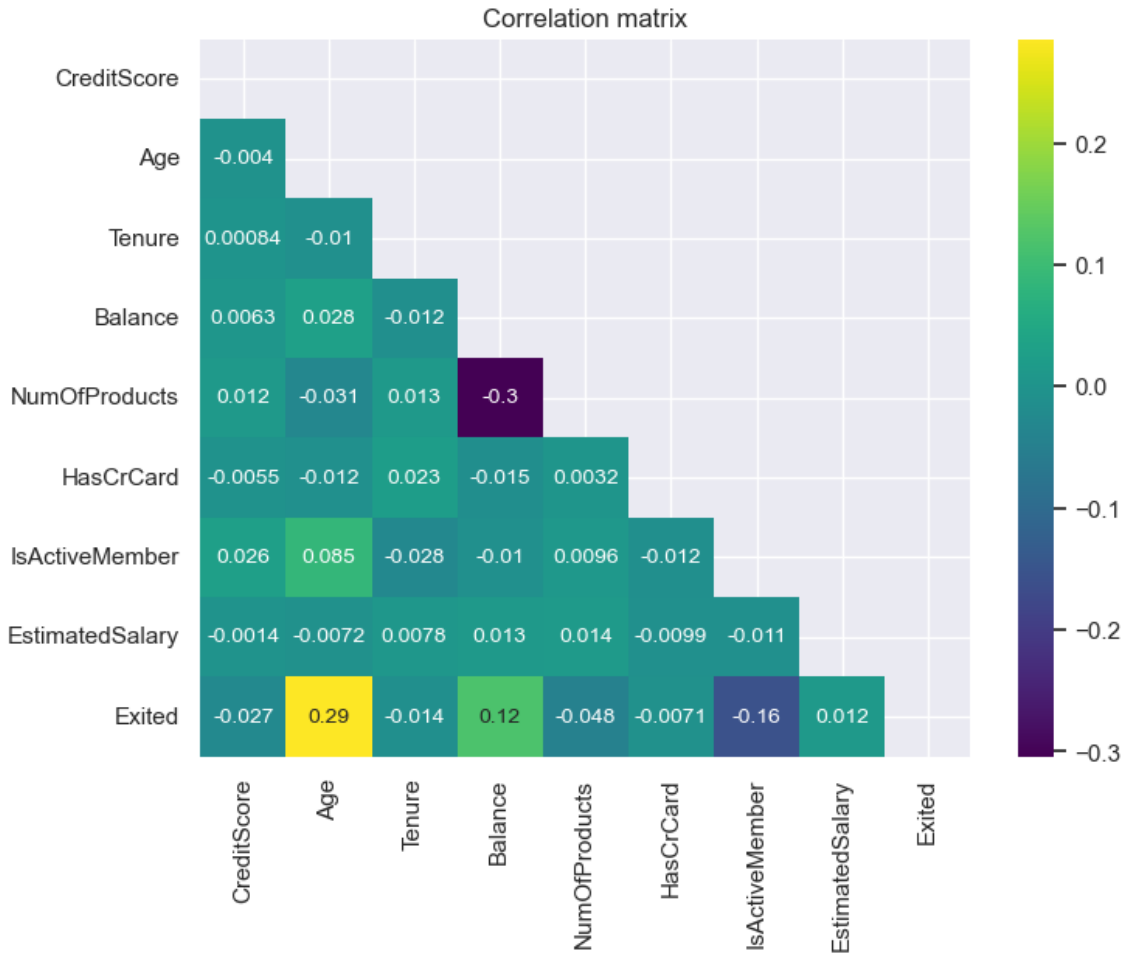
In [121]:

```
sns.pairplot(data=df, hue='Exited')
plt.title('Pair Plot')
plt.show()
```



In [120]:

```
mask = np.zeros_like(df.corr(), dtype=float)
mask[np.triu_indices_from(mask)]=True
plt.figure(figsize=(8,6))
sns.heatmap(df.corr(),annot=True,cmap='viridis', annot_kws={'size':10}, mask=mask)
plt.title('Correlation matrix')
plt.show()
```



Encoding

- One hot encoder for Geography variable to convert it from categorical variable to numerical variable.

In [15]:

```
df['Geography'].value_counts()
```

Out[15]:

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [16]:

```
df=pd.get_dummies(df,columns=['Geography'])
```

In [17]:

```
df.head(2)
```

Out[17]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	619	Female	42	2	0.00	1	1	1
1	608	Female	41	1	83807.86	1	0	1

- Drop dummy variable Geography_France

In [18]:

```
df=df.drop(['Geography_France'],axis=1)
```

- Label encoder for Gender variable to convert it from categorical variable to numerical variable.

In [19]:

```
df['Gender'].value_counts()
```

Out[19]:

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

In [20]:

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

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

In [21]:

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

In [22]:

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

Out[22]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
0	619	0	42	2	0.00	1	1	1
1	608	0	41	1	83807.86	1	0	1

In [23]:

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

Out[23]:

	Exited
0	1
1	0

Feature Scaling

In [25]:

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

Out[25]:

	0	1	2	3	4	5	6	7	
0	-0.326221	-1.095988	0.293517	-1.041760	-1.225848	-0.911583	0.646092	0.970243	0.02
1	-0.440036	-1.095988	0.198164	-1.387538	0.117350	-0.911583	-1.547768	0.970243	0.21
2	-1.536794	-1.095988	0.293517	1.032908	1.333053	2.527057	0.646092	-1.030670	0.24
3	0.501521	-1.095988	0.007457	-1.387538	-1.225848	0.807737	-1.547768	-1.030670	-0.10
4	2.063884	-1.095988	0.388871	-1.041760	0.785728	-0.911583	0.646092	0.970243	-0.36

Check Target variable data balance

In [26]:

```
y.value_counts()    # target variable data is imbalance
```

Out[26]:

```
Exited
0         7963
1         2037
dtype: int64
```

Handle imbalance data

In [27]:

```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler()
x_sam,y_sam=ros.fit_resample(x1,y)
print(x_sam.shape,y_sam.shape,y.shape)
```

```
(15926, 11) (15926, 1) (10000, 1)
```

In [28]:

```
y_sam.value_counts()
```

Out[28]:

```
Exited
0         7963
1         7963
dtype: int64
```

Split the data into train and test data

In [29]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_sam,y_sam,test_size=0.25,random_state=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 [31]:

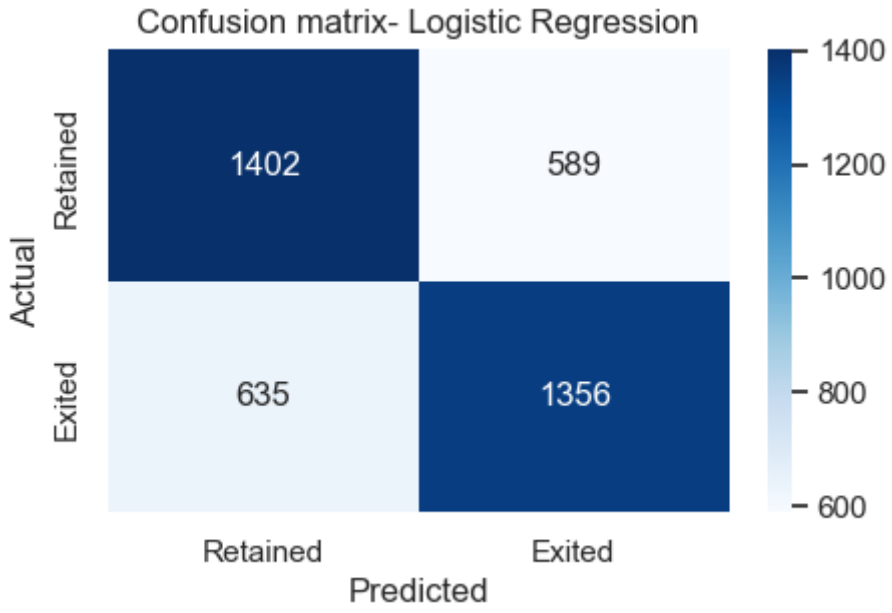
```
# Model building
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression(random_state=100)
log=logit.fit(x_train, y_train)
# Predict
y_pred_train_log = logit.predict(x_train)
y_pred_test_log = logit.predict(x_test)
# Evaluate
from sklearn.metrics import confusion_matrix,classification_report, 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: 0.7049564634963161
------------------------------------------------------------
Logistic regression Test accuracy: 0.6926167754897037
```

Confusion matrix- Logistic Regression

In [32]:

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



Model No. 2 - Decision Tree

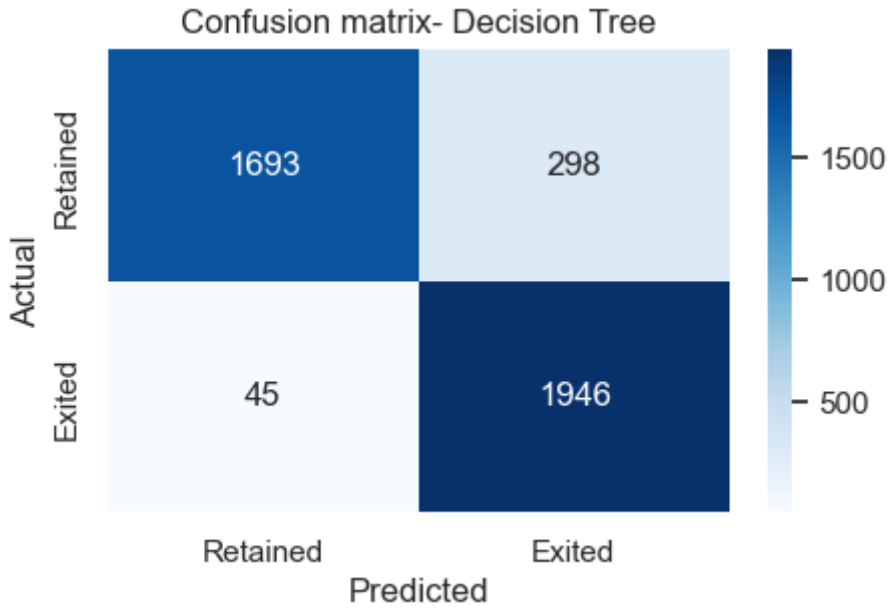
In [33]:

```
# 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: 0.9138623807132095
```

In [34]:

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



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.

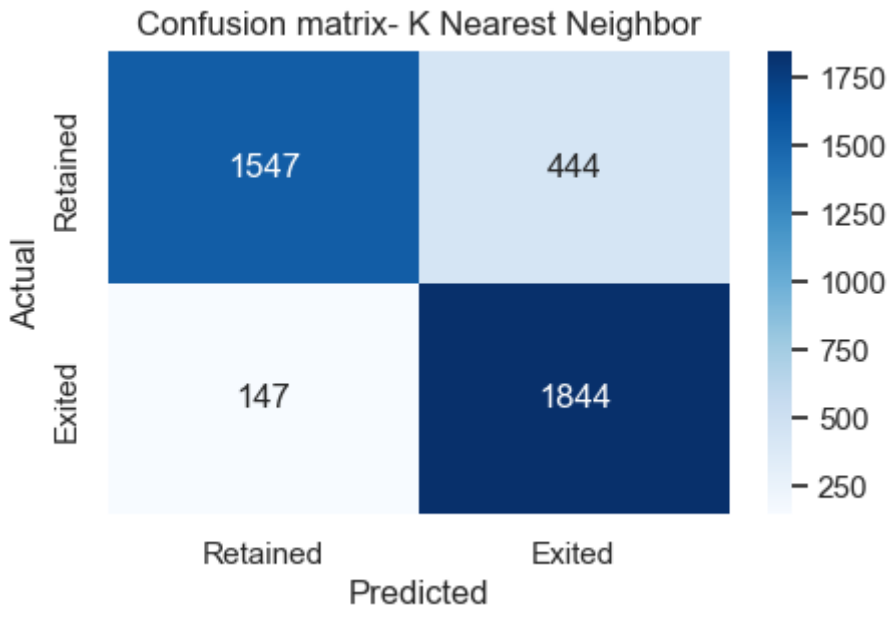
In [35]:

```
# 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: 0.9260716677829873
----------------------------------------------------------------------
K nearest neighbor - Test accuracy: 0.8515821195379206
```

In [36]:

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



Model No. 4 - Bagging model

- Bagging (or Bootstrap aggregating) is a type of ensemble learning in which multiple base models are trained independently in parallel on different subsets of the training data.

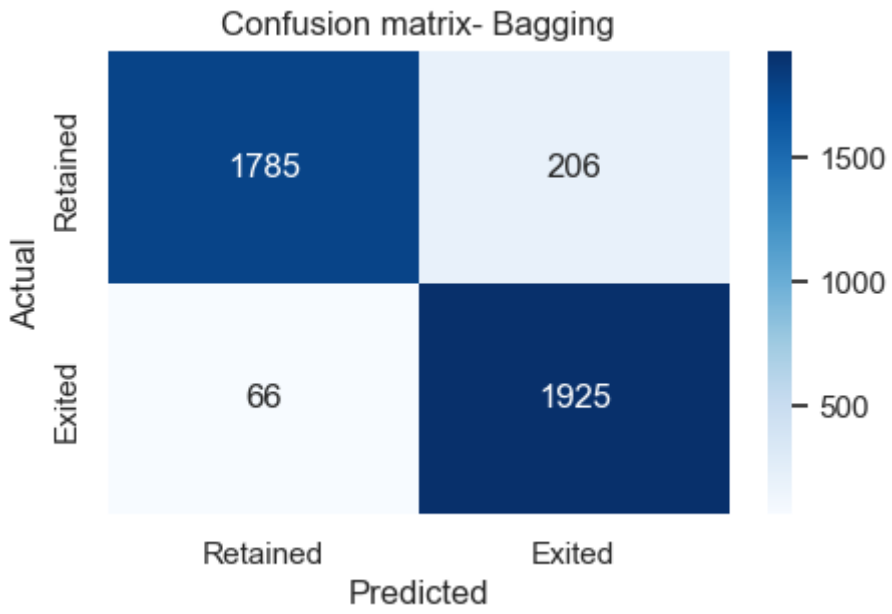
In [37]:

```
# Model building
from sklearn.ensemble import BaggingClassifier
bagging=BaggingClassifier()
bagging.fit(x_train,y_train)# Predict
#Predict
y_pred_train_bag=bagging.predict(x_train)
y_pred_test_bag=bagging.predict(x_test)
# Evaluate
accuracy_bag_test=accuracy_score(y_test,y_pred_test_bag)
accuracy_bag_train=accuracy_score(y_train,y_pred_train_bag)
print('Bagging - Train accuracy:', accuracy_score(y_train, y_pred_train_bag))
print('-------'*10)
print('Bagging - Test accuracy:', accuracy_score(y_test, y_pred_test_bag))
```

```
Bagging - Train accuracy: 0.9960649698593436
----------------------------------------------------------------------
Bagging - Test accuracy: 0.9316926167754898
```

In [38]:

```
Labels = ['Retained','Exited']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_bag),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Bagging")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 5 - 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

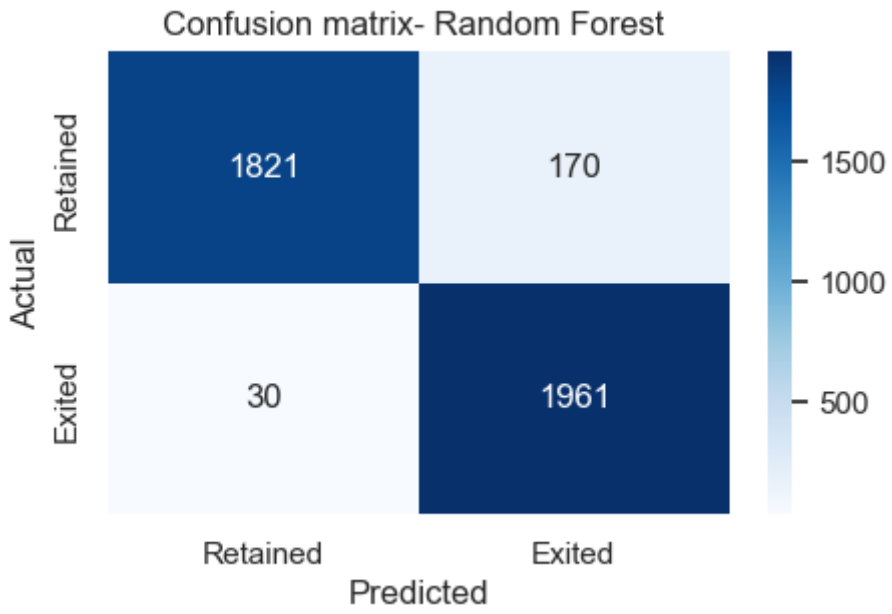
In [33]:

```
# 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: 0.9497739829231542
```

In [34]:

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



Model No. 6 - 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.

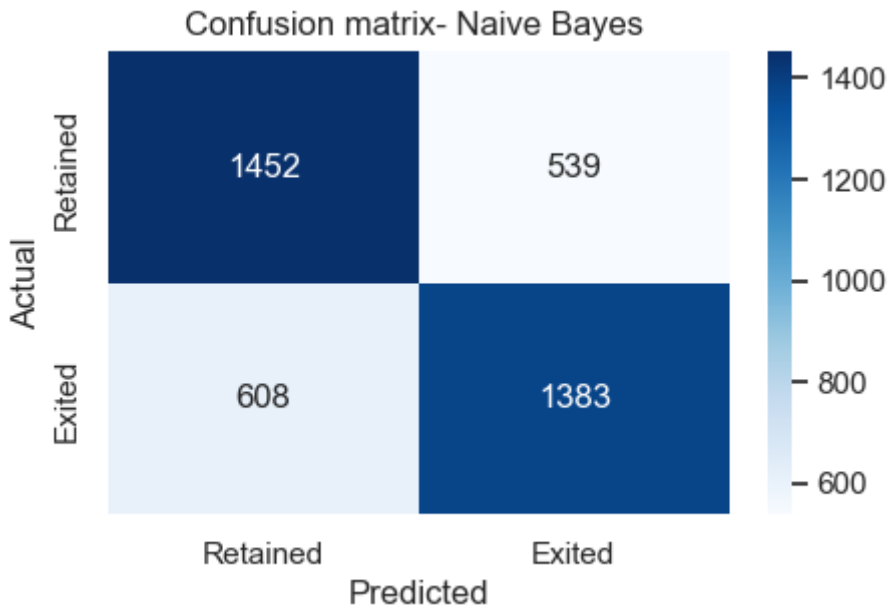
In [41]:

```
# 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: 0.7145847287340924
----------------------------------------------------------------------
Naive Bayes -Test accuracy: 0.7119537920642893
```

In [42]:

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



Model No. 7 - 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.

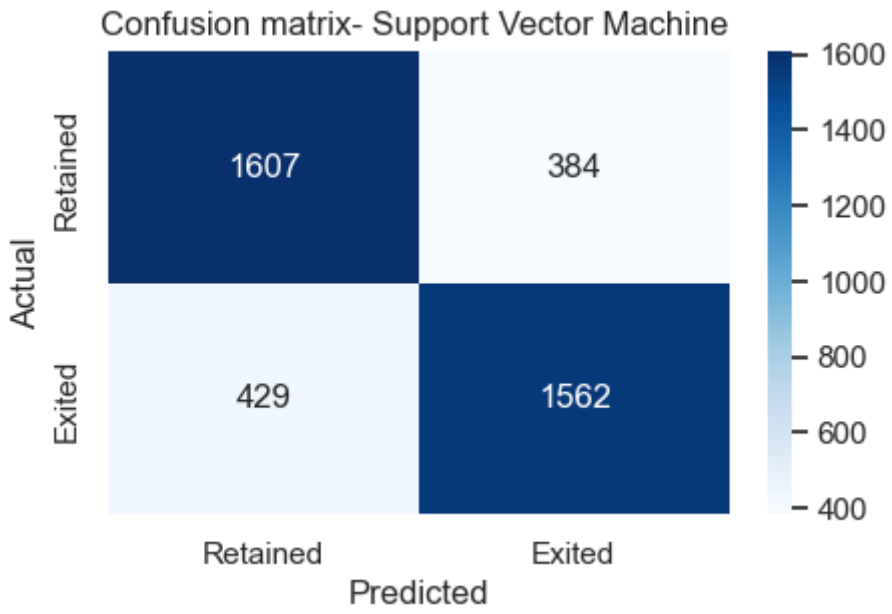
In [43]:

```
# 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: 0.8119557937039518
----------------------------------------------------------------------
Rbf - SVM - Test accuracy: 0.7958312405826218
```

In [47]:

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



Model No. 8 - Adaptive boosting or AdaBoost

- Yoav Freund and Robert Schapire are credited with the creation of the AdaBoost algorithm. This method operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error. The model continues optimize in a sequential fashion until it yields the strongest predictor.

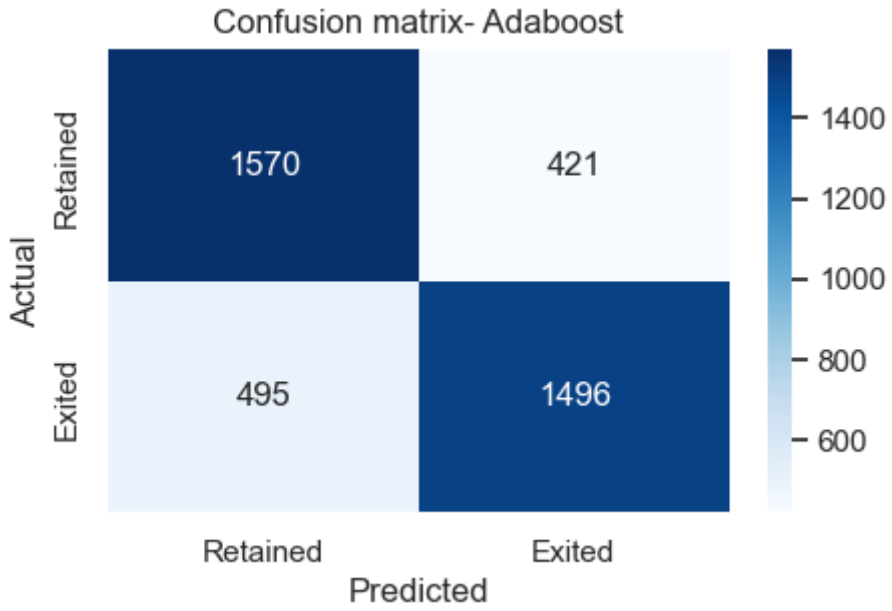
In [48]:

```
# Model building
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ad=ada.fit(x_train, y_train)
# Predict
y_pred_ad = ada.predict(x_test)
y_pred_ad_train = ada.predict(x_train)
# Evaluate
from sklearn.metrics import confusion_matrix,classification_report, accuracy_score
accuracy_ad_test=accuracy_score(y_test,y_pred_ad)
accuracy_ad_train=accuracy_score(y_train,y_pred_ad_train)
print('AdaBoost Train accuracy:', accuracy_score(y_train, y_pred_ad_train))
print('-------'*5)
print('AdaBoost Test accuracy:', accuracy_score(y_test, y_pred_ad))
```

```
AdaBoost Train accuracy: 0.7780475552578701
-----------------------------------
AdaBoost Test accuracy: 0.7699648417880463
```

In [49]:

```
Labels = ['Retained','Exited']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_ad),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Adaboost")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 9 - Gradient Boosting

- Jerome H. Friedman developed gradient boosting, which works by sequentially adding predictors to an ensemble with each one. However, instead of changing weights of data points like AdaBoost, the gradient boosting trains on the residual errors of the previous predictor. The name, gradient boosting, is used since it combines the gradient descent algorithm and boosting method.

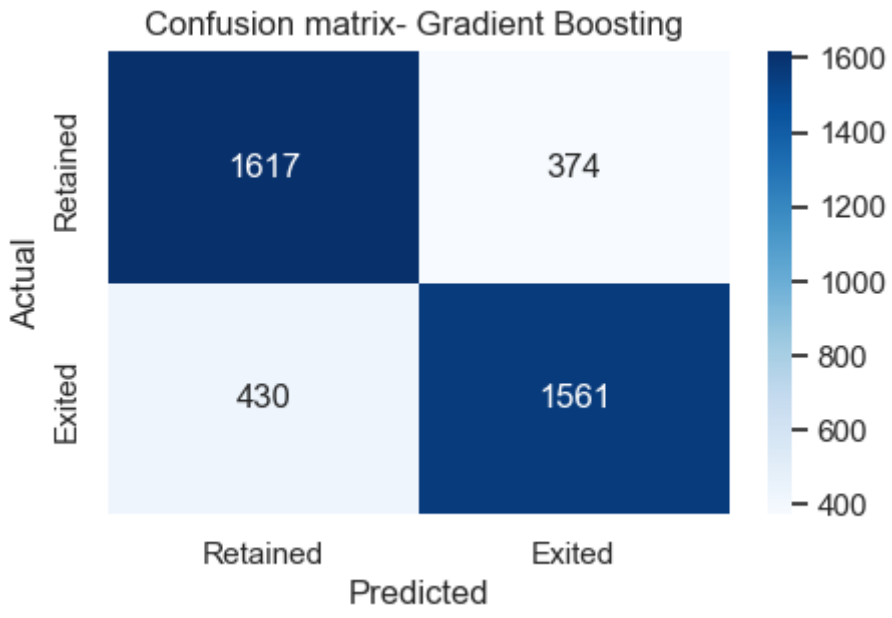
In [50]:

```
# Model building
from sklearn.ensemble import GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd=gdb.fit(x_train, y_train)
# Predict
y_pred_gd = gdb.predict(x_test)
y_pred_gd_train = gdb.predict(x_train)
# Evaluate
accuracy_gd_test=accuracy_score(y_test,y_pred_gd)
accuracy_gd_train=accuracy_score(y_train,y_pred_gd_train)
print('GradientBoosting Train accuracy:', accuracy_score(y_train, y_pred_gd_train))
print('-------'*5)
print('GradientBoosting Test accuracy:', accuracy_score(y_test, y_pred_gd))
```

```
GradientBoosting Train accuracy: 0.8132953784326858
-----------------------------------
GradientBoosting Test accuracy: 0.7980914113510799
```

In [51]:

```
Labels = ['Retained','Exited']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_gd),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Gradient Boosting ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 10 - 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

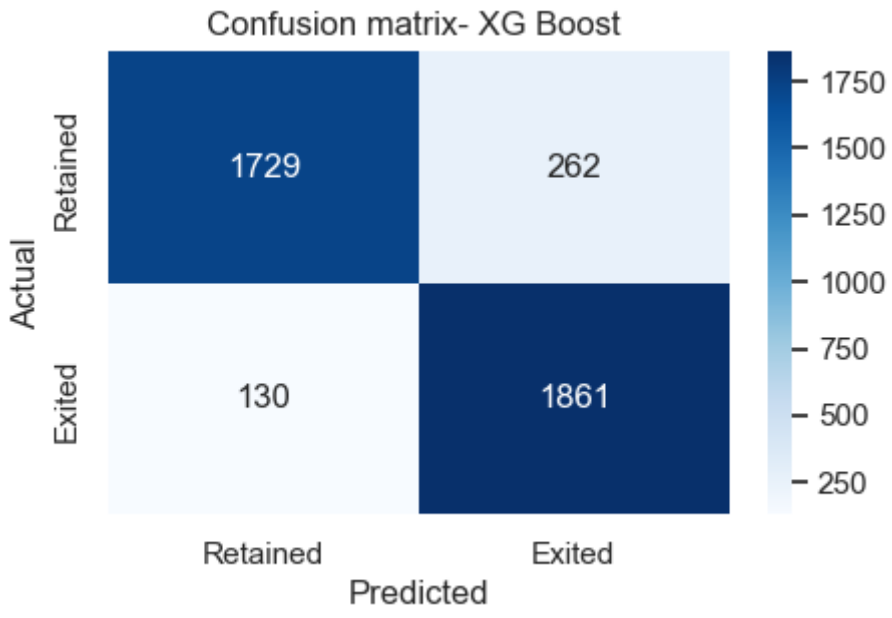
In [52]:

```
# 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: 0.9607334226389819
-----------------------------------
XGBoost Test accuracy: 0.9015570065293822
```

In [53]:

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



Model No. 11 Voting ensemble¶

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

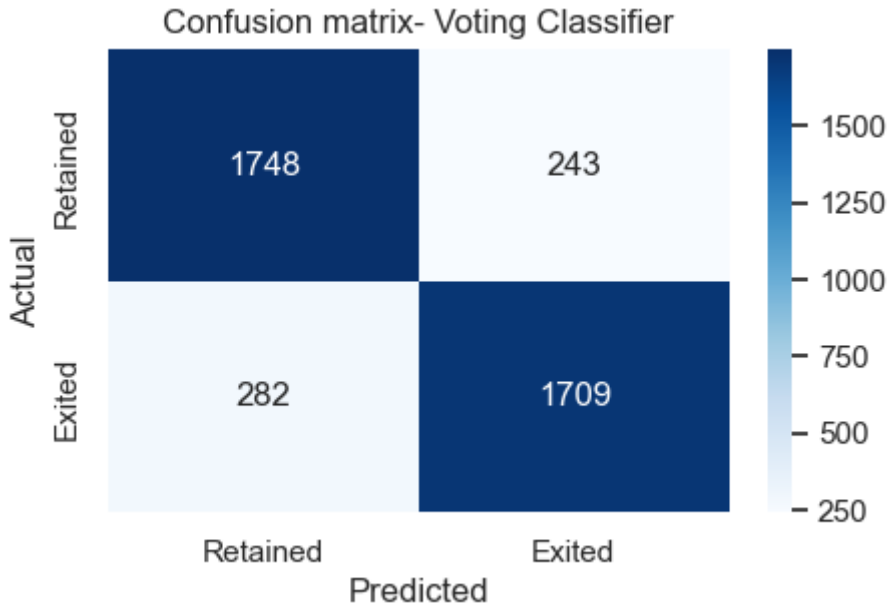
In [54]:

```
from sklearn.ensemble import VotingClassifier
evc=VotingClassifier(estimators = [('Logistic',logit),('Decision_tree', dtree),('KNN',kn
                                   ('Randomforest',rf),('NaiveBayes',nb), ('SVM_RBF' ,sv
                                   ('Gradient_boosting',gd),('Extra_Gradient_booting',xg
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: 0.9120897521768252
--------------------------------------------------------------------------------
-
Voting ensemble train accuracy: 0.8681567051732798
```

In [55]:

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



Combining all models in Tabular format for better understanding

In [56]:

```
Models=['Logistic','Decision_tree','KNN','Bagging','Random_forest','Naive_bayes','SVM',
        'Adaboost','GradientBoosting','XGboost','Voting_Classifier']
Trainacc=[accuracy_log_train,accuracy_dtree_train,accuracy_knn_train,accuracy_bag_train,
          accuracy_rbf_train,accuracy_ad_train,accuracy_gd_train, accuracy_xg_train,accu
Testacc=[accuracy_log_test,accuracy_dtree_test,accuracy_knn_test,accuracy_bag_test,accur
         accuracy_rbf_test, accuracy_ad_test,accuracy_gd_test, accuracy_xg_test,accuracy
```

In [57]:

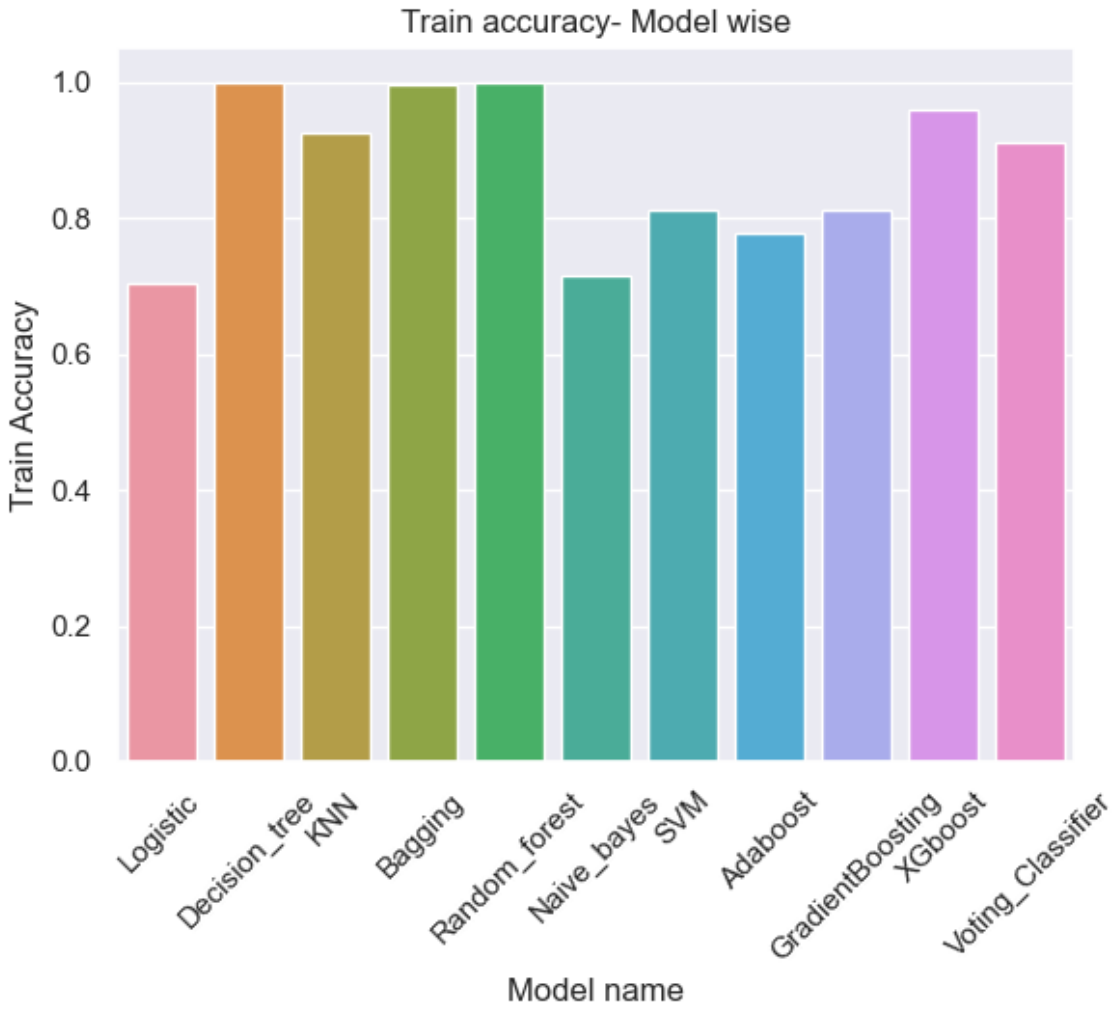
```
Combined_accuracy=pd.DataFrame({'Model name':Models,'Train Accuracy':Trainacc,
                                'Test Accuracy':Testacc})
print(Combined_accuracy)
```

```
           Model name  Train Accuracy  Test Accuracy
0            Logistic        0.703784       0.694626
1       Decision_tree        1.000000       0.913862
2                 KNN        0.926072       0.851582
3             Bagging        0.996065       0.931693
4       Random_forest        1.000000       0.948518
5         Naive_bayes        0.714585       0.711954
6                 SVM        0.811956       0.795831
7            Adaboost        0.778048       0.769965
8    GradientBoosting        0.813295       0.798091
9             XGboost        0.960733       0.901557
10  Voting_Classifier        0.912090       0.868157
```

Accuracy visualization

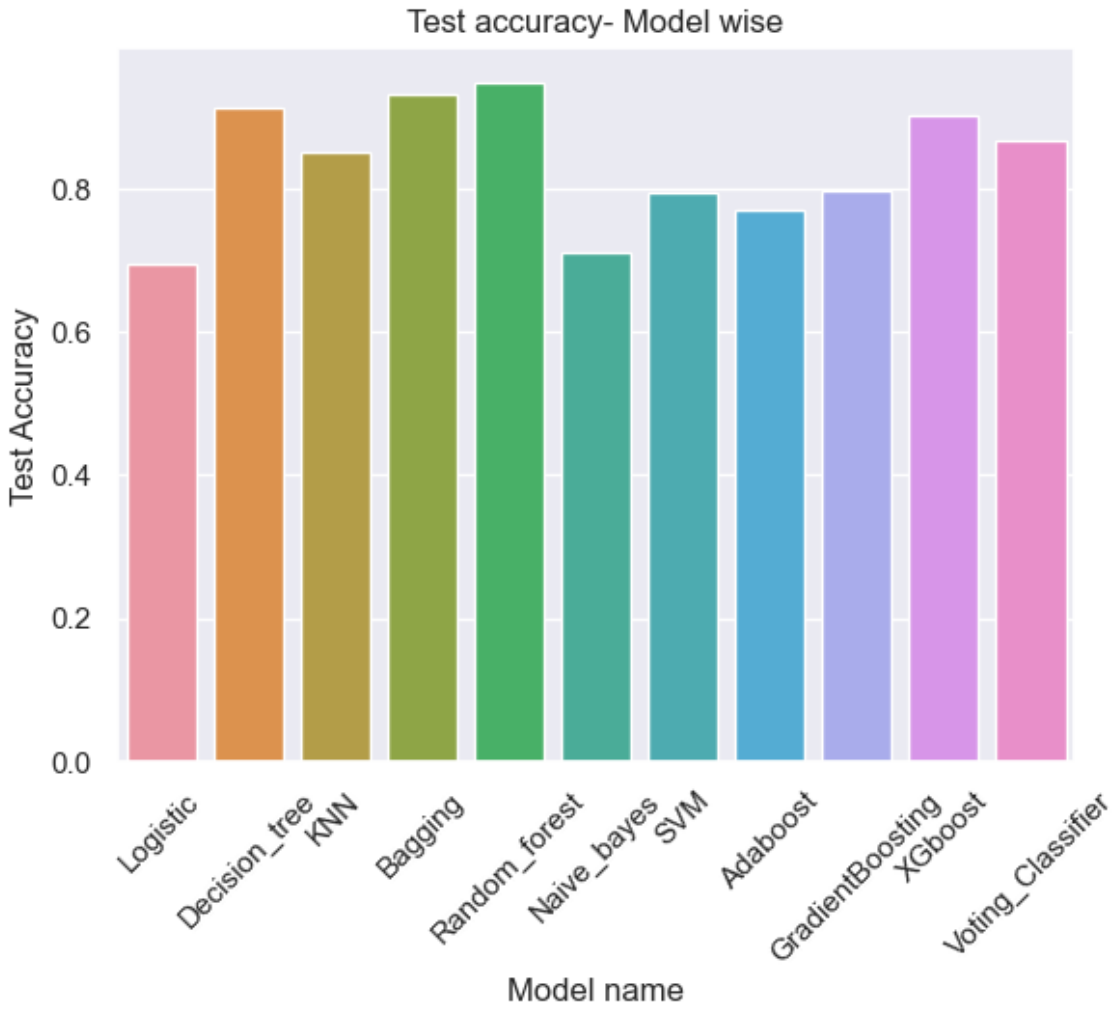
In [58]:

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



In [59]:

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



Cross validation of Top 3 highest accuracy score

- Decision Tree Classifier
- Bagging Classifier
- Random Forest Classifier

In [36]:

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

In [61]:

```
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: 0.8981901838340937
----------------------------------------------------------------------
Decision Tree Test accuracy after Cross validation: 0.7980938527222579
```

In [62]:

```
train_accuracy_bag = cross_val_score(bagging,x_train, y_train, cv=10)
crossval_train_bag=train_accuracy_bag.mean()
test_accuracy_bag = cross_val_score(bagging,x_test, y_test, cv=10)
crossval_test_bag=test_accuracy_bag.mean()
print('Bagging after Cross validation Train accuracy:', crossval_train_bag)
print('---------------'*5)
print('Bagging after Cross validation Test accuracy:', crossval_test_bag)
```

```
Bagging after Cross validation Train accuracy: 0.9232237197143318
-----------------------------------------------------------------------
-
Bagging after Cross validation Test accuracy: 0.833003362678052
```

In [37]:

```
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: 0.9400527042464766
----------------------------------------------------------------------
Random forest  after Cross validation Test accuracy: 0.8556057228498382
```

Conclusion

- I used various Supervised machine learning models to solve the classification problem performed on 10,000 observations of the churn dataset. The objective was to predict customer churn.
- For Evaluation I used accuracy score & confusion matrix.
- Three highest accuracy models were Random Forest Classifier, Bagging Classifier & Decision Tree Classifier with both Train & Test data accuracy crossing over 90%.
- Random Forest classifier with Train accuracy at 100% and Test accuracy at 94% gives the highest accuracy amongst all the models but it has overfitting issue.
- So after cross validation to deal with overfitting issue of Random Forest model train accuracy is 94% and Test accuracy is 85% making it the best model to solve the classification problem for this dataset.
- Bagging follows next in accuracy with train accuracy of 99% and Test accuracy of 93%.
- Logistic Regression performed the worst with 71% accuracy for Train data & 70% for Test data.

In [ ]: