Load Required modules

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

Load the required dataset

In [2]:
```
df = pd.read_csv("C:/Users/ADMIN/Desktop/Data Science/Datasets/Bank Customer Churn
```

View the dataset

In [3]:
```
df.head(10)
```

Out[3]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number
0	15634602	619	France	Female	42	2	0.00	1
1	15647311	608	Spain	Female	41	1	83807.86	1
2	15619304	502	France	Female	42	8	159660.80	3
3	15701354	699	France	Female	39	1	0.00	2
4	15737888	850	Spain	Female	43	2	125510.82	1
5	15574012	645	Spain	Male	44	8	113755.78	2
6	15592531	822	France	Male	50	7	0.00	2
7	15656148	376	Germany	Female	29	4	115046.74	4
8	15792365	501	France	Male	44	4	142051.07	2
9	15592389	684	France	Male	27	2	134603.88	1

Structure of the dataset

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customer_id       10000 non-null  int64  
 1   credit_score      10000 non-null  int64  
 2   country           10000 non-null  object 
 3   gender            10000 non-null  object 
 4   age               10000 non-null  int64  
 5   tenure            10000 non-null  int64  
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64  
 8   credit_card       10000 non-null  int64  
 9   active_member     10000 non-null  int64  
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64  
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

Check for missing values

In [5]:
```python
df.isnull().sum()
```

Out[5]:
```
customer_id         0
credit_score        0
country             0
gender              0
age                 0
tenure              0
balance             0
products_number     0
credit_card         0
active_member       0
estimated_salary    0
churn               0
dtype: int64
```

Check for duplicates

In [6]:
```python
df.duplicated().sum()
```

Out[6]:
```
0
```

Data Preprocessing

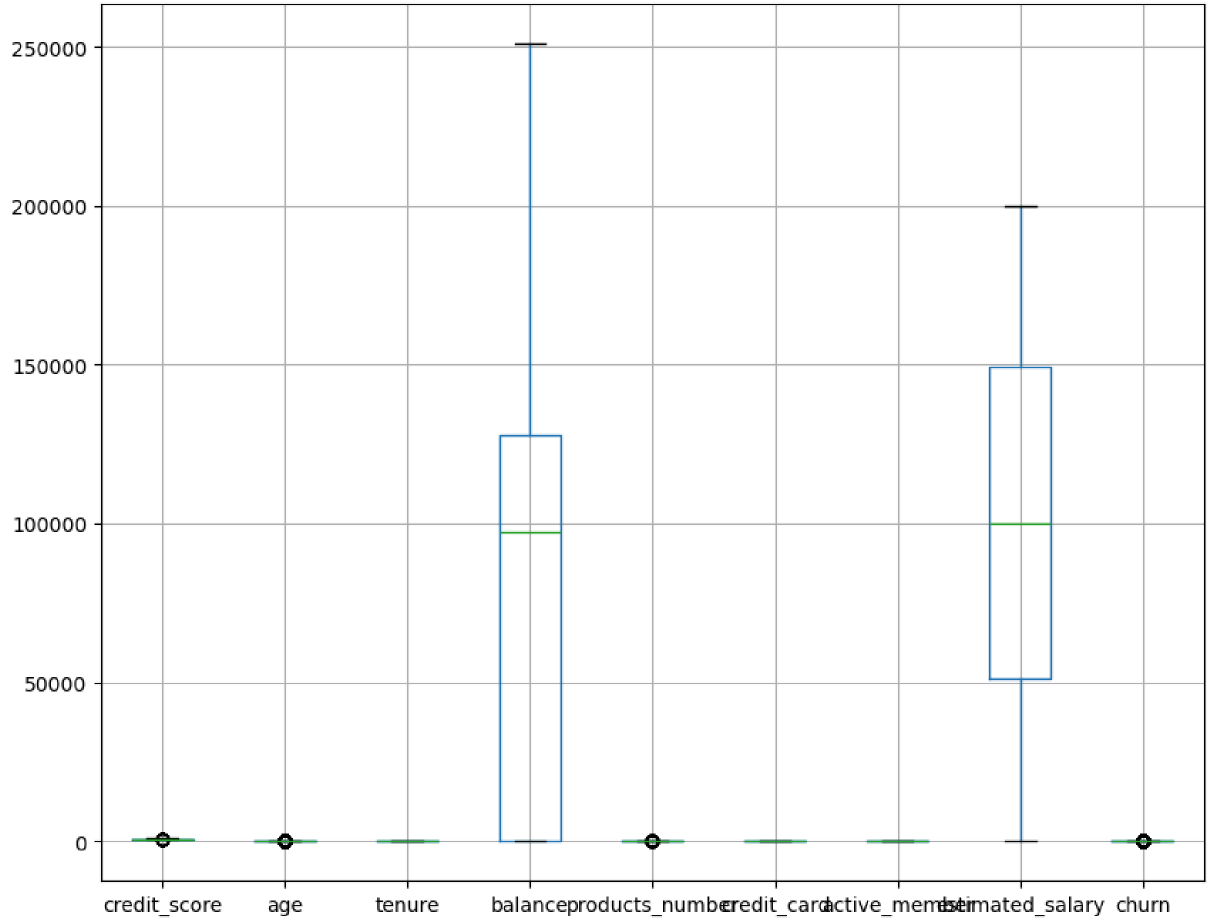
Removing unnecessary columns

In [7]:
```python
df = df.drop(columns = ["customer_id"])
```

Checking for Outliers

In [8]:
```python
numeric_cols = df.select_dtypes(include = ["float64", "int64"])
numeric_cols.boxplot(figsize = (10, 8))
```

```
plt.show()
```



One hot encoding

In [18]:
```
## Load the required module
from sklearn.preprocessing import LabelEncoder

## Select categorical columns
categorical_cols = df.select_dtypes(include = ["object"]).columns

## Initialize the label encoder
label_encoder = LabelEncoder()

## Apply label encooding to selected columns
for col in categorical_cols:
    df[col] = label_encoder.fit_transform(df[col])
```

Exploratory Data Analysis

Summary Statistics

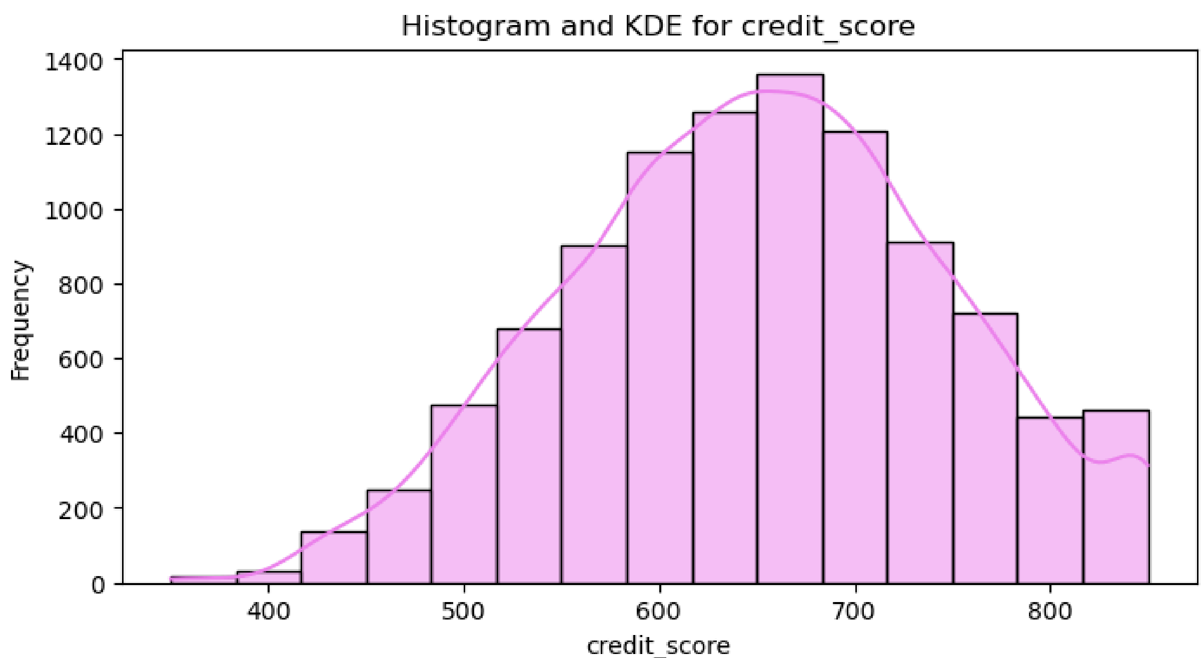
In [20]:
```
df.describe()
```

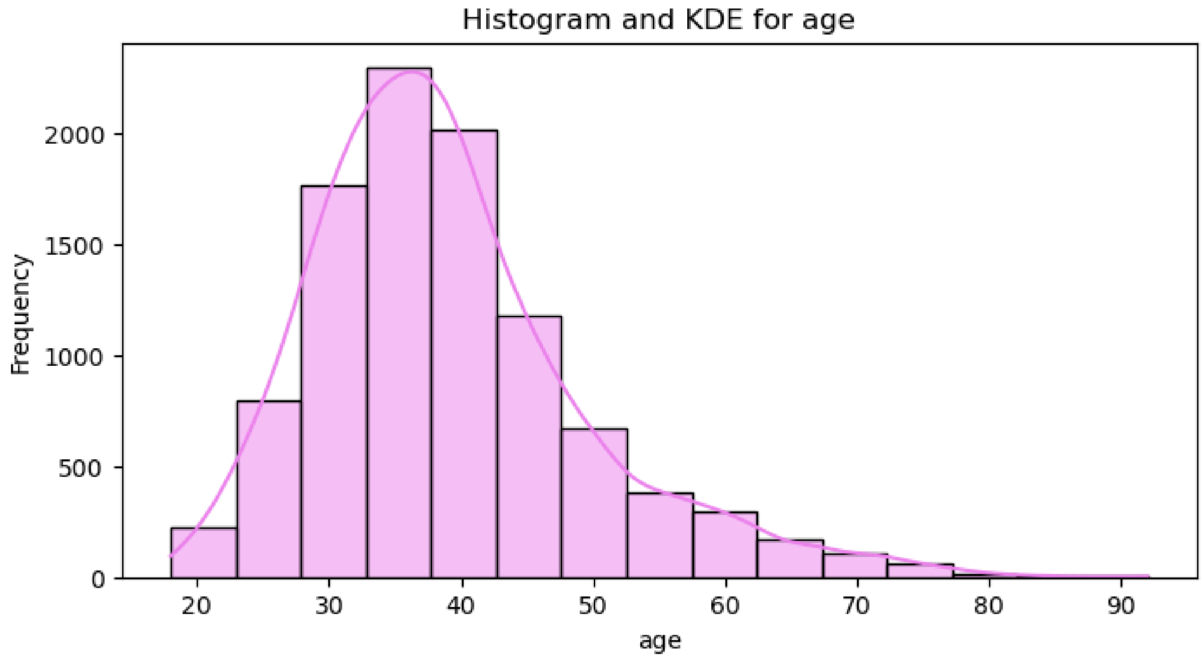
Out[20]:

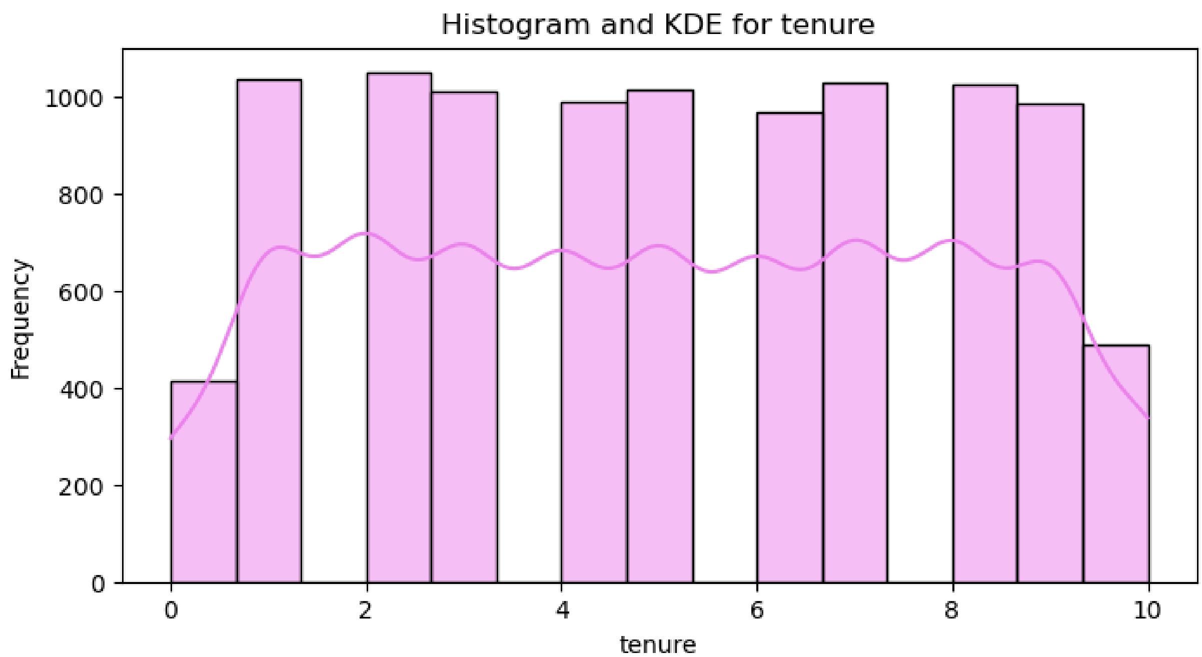
	credit_score	country	gender	age	tenure	balanc
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000
mean	650.528800	0.746300	0.545700	38.921800	5.012800	76485.88928
std	96.653299	0.827529	0.497932	10.487806	2.892174	62397.40520
min	350.000000	0.000000	0.000000	18.000000	0.000000	0.00000
25%	584.000000	0.000000	0.000000	32.000000	3.000000	0.00000
50%	652.000000	0.000000	1.000000	37.000000	5.000000	97198.54000
75%	718.000000	1.000000	1.000000	44.000000	7.000000	127644.24000
max	850.000000	2.000000	1.000000	92.000000	10.000000	250898.09000

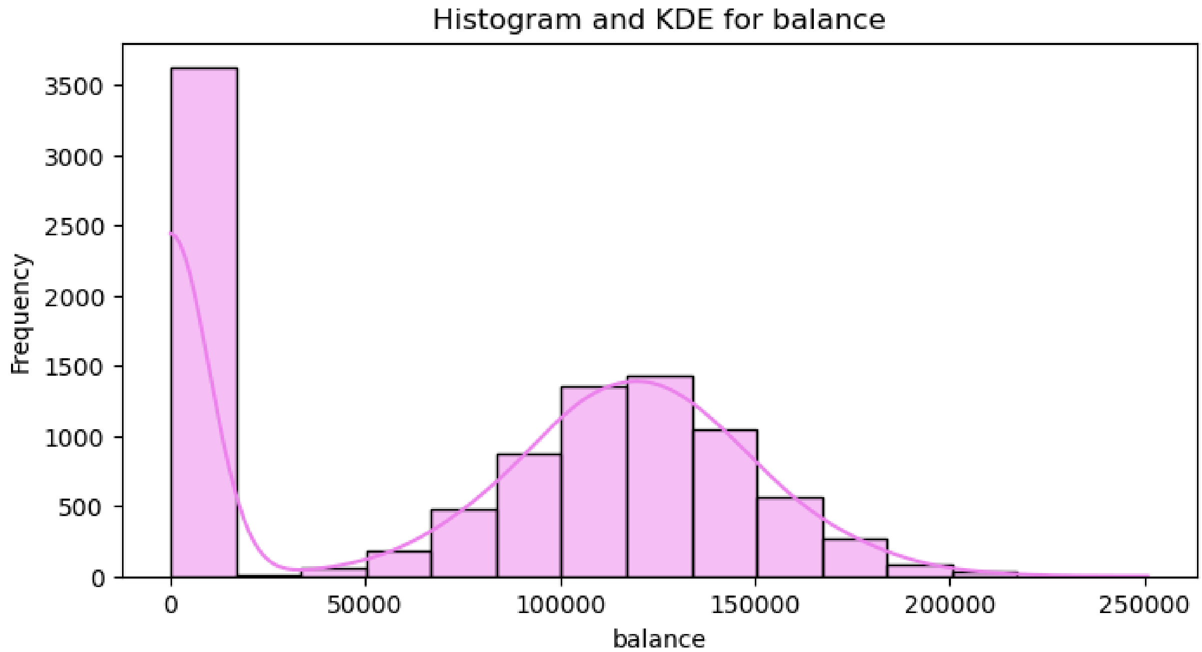
Plotting histograms and kde plots for numeric variables

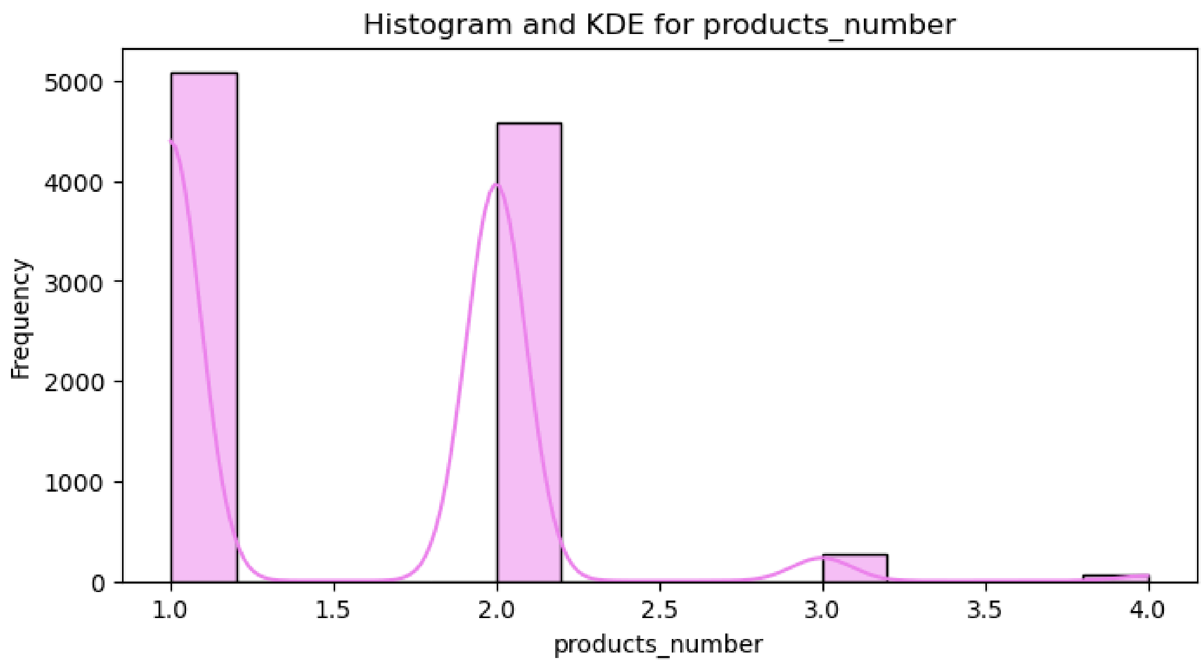
In [21]:
```
numeric_col = df.select_dtypes(include = ["float64", "int64"])
for col in numeric_col:
    plt.figure(figsize = (8, 4))
    sns.histplot(df[col], kde = True, bins = 15, color = "violet")
    plt.title(f'Histogram and KDE for {col}')
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()
```

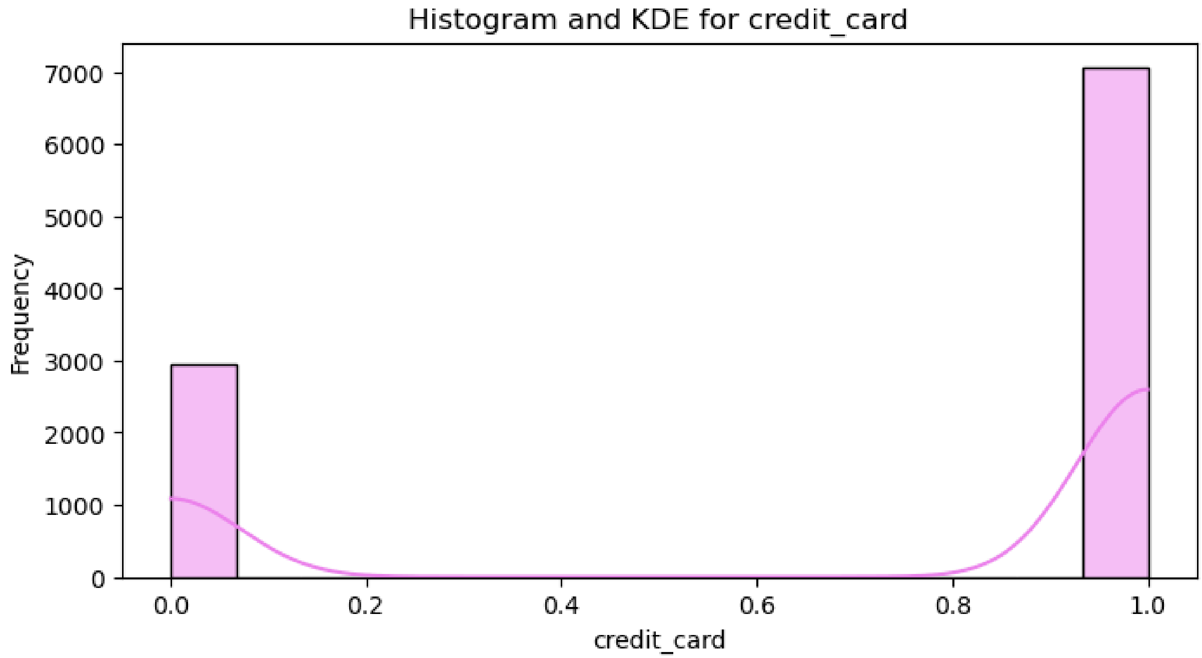


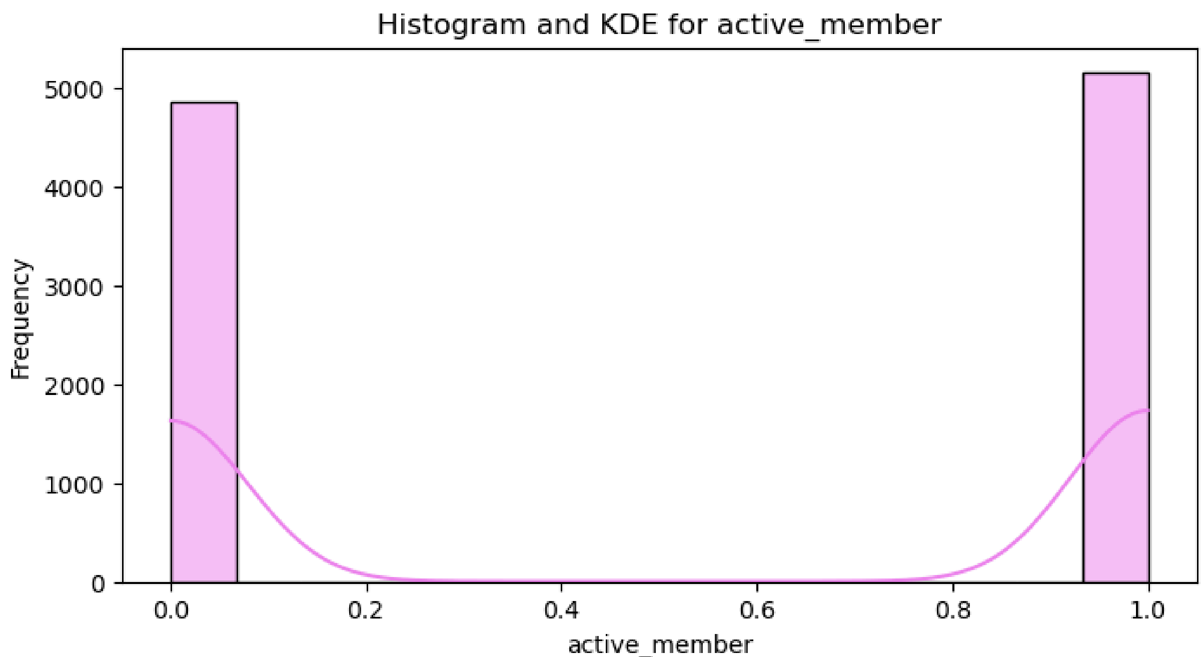

Histogram and KDE for age
Frequency
0
500
1000
1500
2000
20
30
40
50
60
70
80
90
age

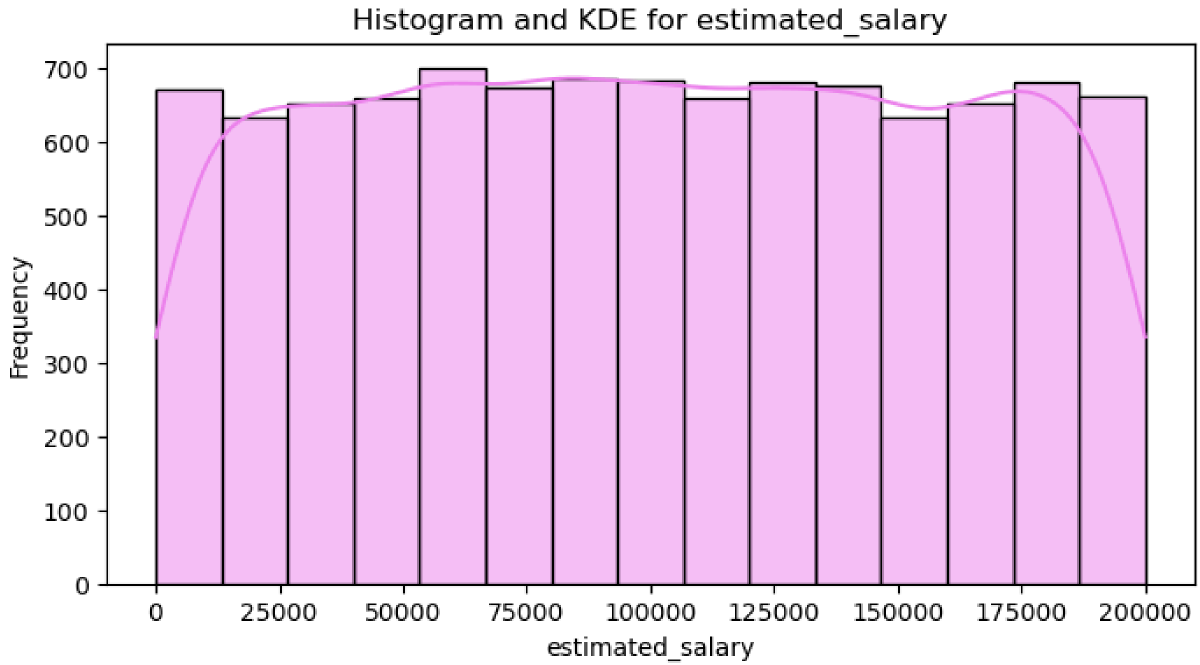

Histogram and KDE for tenure
Frequency
0
200
400
600
800
1000
0
2
4
6
8
10
tenure

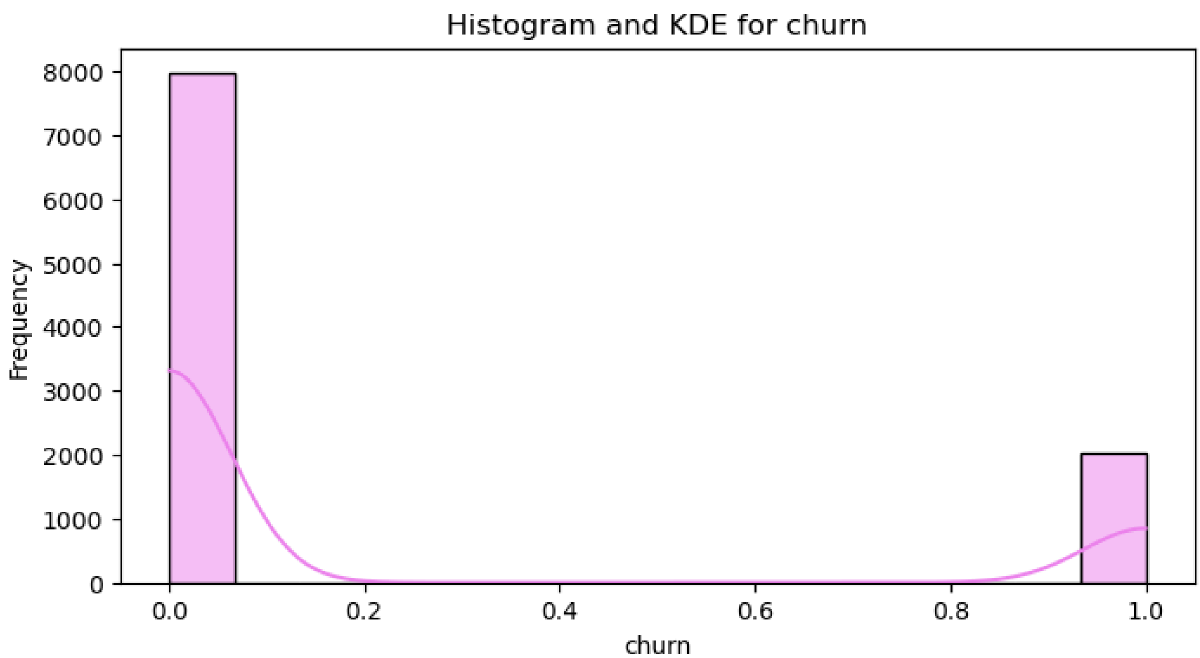

Histogram and KDE for balance
Frequency
0
500
1000
1500
2000
2500
3000
3500
0
50000
100000
150000
200000
250000
balance


Histogram and KDE for products_number
Frequency
0
1000
2000
3000
4000
5000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
products_number


Histogram and KDE for credit_card
Frequency
7000
6000
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
credit_card

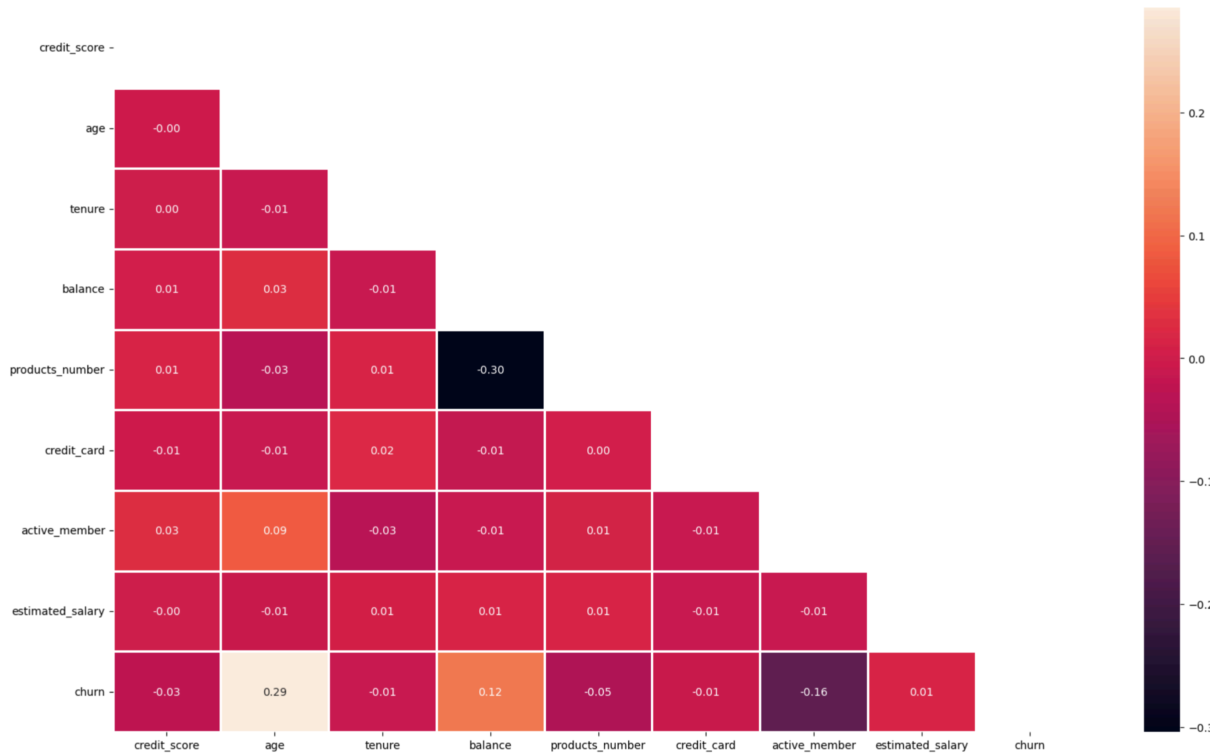

Histogram and KDE for active_member
Frequency
5000
4000
3000
2000
1000
0
0.0
0.2
0.4
0.6
0.8
1.0
active_member





Correlation Analysis

In [22]:
```
plt.figure(figsize = (20, 12))
corr = numeric_col.corr()
mask = np.triu(np.ones_like(corr, dtype = bool))
sns.heatmap(corr, mask = mask, linewidths = 1, annot = True, fmt = ".2f")
plt.show()
```

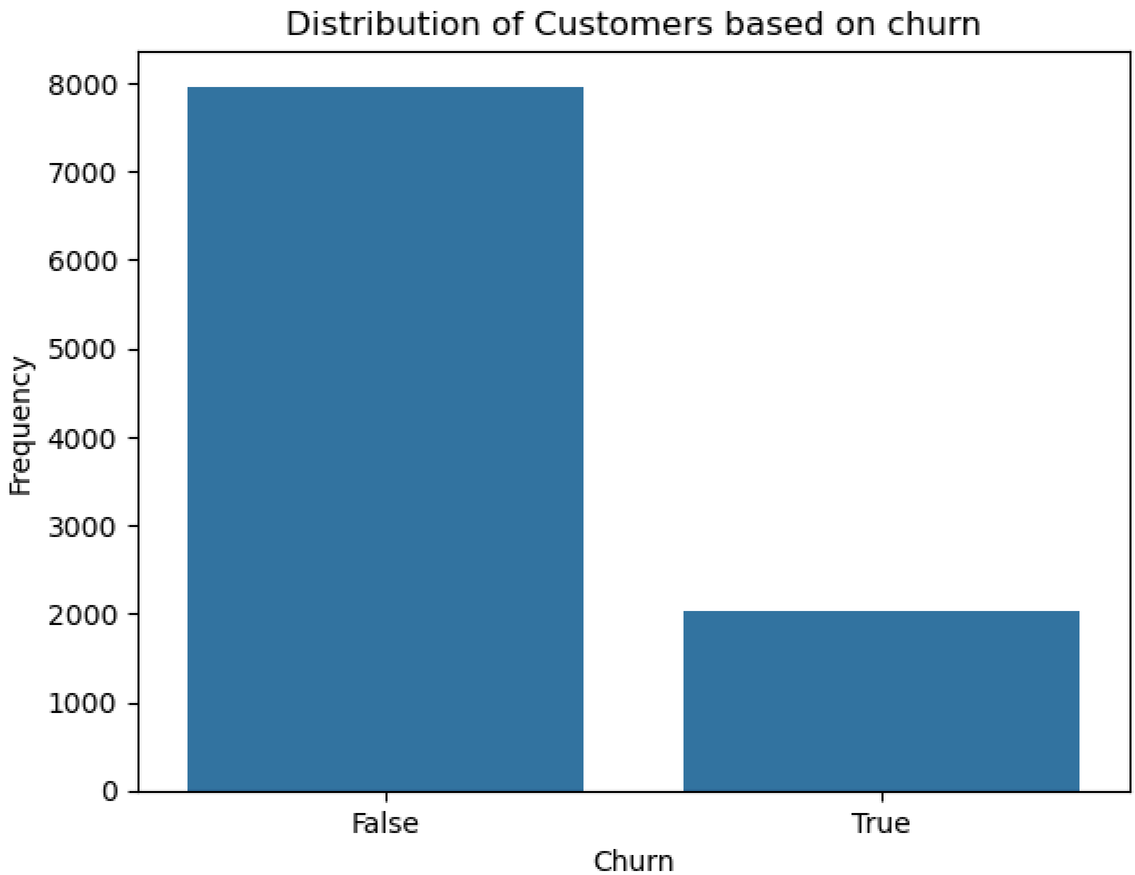


Distribution of the study variable

In [23]:
```
freq_table = df['churn'].value_counts()
percent_table = df['churn'].value_counts(normalize=True) * 100
result = pd.DataFrame({'Frequency': freq_table, 'Percentage': percent_table.round(2
print(result)
```

```
       Frequency  Percentage
churn
0           7963       79.63
1           2037       20.37
```

In [24]:
```
sns.countplot(x = "churn", data = df)
plt.title('Distribution of Customers based on churn')
plt.ylabel("Frequency")
plt.xlabel("Churn")
plt.xticks([0, 1], labels = ["False", "True"])
plt.show()
```



Define the X and y features

In [25]:
```
X = df.drop(columns = ["churn"])
y = df["churn"]
```

Handling Class imbalance

Oversampling of the minority class

In [26]:
```
## Load the required module
from imblearn.over_sampling import RandomOverSampler

## Initialize the RandomOverSampler
ros = RandomOverSampler(random_state = 42)

## Apply the RandomOverSampler
X_resampled, y_resampled = ros.fit_resample(X, y)

## Print the oversampled data
dict(zip(*np.unique(y_resampled, return_counts = True)))
```

Out[26]:
```
{0: 7963, 1: 7963}
```

Splitting the data into training and testing sets

In [27]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_
```

Feature Scaling

In [28]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1. Logistic Regression

In [29]:
```
## Load the required modules
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score

## Initialize the model
reg = LogisticRegression()

## Fit the model
reg.fit(X_train, y_train)

## Make predictions
lr_pred = reg.predict(X_test)

## Print the evaluation metrics
print("Classification Report is:\n",classification_report(y_test,lr_pred))
print("\n F1:\n",f1_score(y_test,lr_pred))
print("\n Precision score is:\n",precision_score(y_test,lr_pred))
print("\n Recall score is:\n",recall_score(y_test,lr_pred))
print("\n Confusion Matrix:\n")

## Print the accuracy score
reg_score = accuracy_score(y_test, lr_pred)
```

```
Classification Report is:
               precision    recall  f1-score   support

           0       0.70      0.70      0.70      1633
           1       0.68      0.68      0.68      1553

    accuracy                           0.69      3186
   macro avg       0.69      0.69      0.69      3186
weighted avg       0.69      0.69      0.69      3186


 F1:
 0.6812479897073014

 Precision score is:
 0.6805912596401028

 Recall score is:
 0.68190598840953

 Confusion Matrix:
```

2. Random Forest

In [30]:

```
## Load the required modules
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

## Initialize the model
RF = RandomForestClassifier()

## Define the hyperparameters
n_estimators = [1800]
max_features = ['sqrt', 'log2']

## Define grid search
grid = dict(n_estimators=n_estimators,max_features=max_features)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=RF, param_grid=grid, n_jobs=-1, cv=cv, scoring

## Fit the model using grid search
best_model = grid_search.fit(X_train, y_train)

## Make predictions
rf_pred = best_model.predict(X_test)

## Print the accuracy score
RF_score= accuracy_score(y_test, rf_pred)

## Print the evaluation matrix
print("Classification Report is:\n",classification_report(y_test,rf_pred))
print("\n F1:\n",f1_score(y_test,rf_pred))
```

```
print("\n Precision score is:\n",precision_score(y_test,rf_pred))
print("\n Recall score is:\n",recall_score(y_test,rf_pred))
```

```
Classification Report is:
               precision    recall  f1-score   support

           0       0.98      0.92      0.95      1633
           1       0.92      0.98      0.95      1553

    accuracy                           0.95      3186
   macro avg       0.95      0.95      0.95      3186
weighted avg       0.95      0.95      0.95      3186


 F1:
 0.950625

 Precision score is:
 0.9234972677595629

 Recall score is:
 0.9793947198969736
```

3. Support Vector Machines

In [31]:
```
## Load the required modules
from sklearn.model_selection import RepeatedStratifiedKFold, GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, f1_score, prec
import seaborn as sns
import matplotlib.pyplot as plt

## Define model and parameter grid
svm = SVC()
kernel = ['poly', 'rbf']
C = [50, 10, 1.0, 0.1, 0.01]
gamma = ['scale']
grid = dict(kernel=kernel, C=C, gamma=gamma)

## Setup cross-validation and GridSearch
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=svm, param_grid=grid, n_jobs=-1, cv=cv, scorin

## Fit the model
grid_result = grid_search.fit(X_train, y_train)

## Predict class labels on test data
svm_pred = grid_result.predict(X_test)
svm_score = accuracy_score(y_test, svm_pred)

## Evaluate performance
print("Classification Report:\n", classification_report(y_test, svm_pred))
print("F1 Score:", f1_score(y_test, svm_pred, average='macro'))
print("Precision:", precision_score(y_test, svm_pred, average='macro'))
print("Recall:", recall_score(y_test, svm_pred, average='macro'))
print("Accuracy:", svm_score)
```

```
Best Parameters: {'C': 50, 'gamma': 'scale', 'kernel': 'rbf'}
Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.79      0.82      1633
           1       0.79      0.85      0.82      1553

    accuracy                           0.82      3186
   macro avg       0.82      0.82      0.82      3186
weighted avg       0.82      0.82      0.82      3186

F1 Score: 0.8204362283717523
Precision: 0.8219068926051922
Recall: 0.8212976168835855
Accuracy: 0.8204645323289391
```

4. XG Boost Classifier

In [32]:
```python
## Load the required module
from xgboost import XGBClassifier

## Intialize the model
xgb = XGBClassifier()

## Fit the model
xgb.fit(X_train, y_train)

## Make predictions
xgb_pred = xgb.predict(X_test)

## Print the accuracy score
xgb_score  = xgb.score(X_test, y_test)

## Evaluate performance
print("Classification Report:\n", classification_report(y_test, xgb_pred))
print("F1 Score:", f1_score(y_test, xgb_pred))
print("Precision:", precision_score(y_test, xgb_pred))
print("Recall:", recall_score(y_test, xgb_pred))
print("Accuracy:", accuracy_score(y_test, xgb_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.87      0.90      1633
           1       0.87      0.93      0.90      1553

    accuracy                           0.90      3186
   macro avg       0.90      0.90      0.90      3186
weighted avg       0.90      0.90      0.90      3186

F1 Score: 0.8987577639751553
Precision: 0.8680263947210558
Recall: 0.9317450096587251
Accuracy: 0.8976773383553045
```

5. K Nearest Neighbors

In [33]:
```
## Load the required libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score
from sklearn.model_selection import GridSearchCV

## List Hyperparameters to tune
knn= KNeighborsClassifier()
n_neighbors =range(15,25)
weights = ['uniform', 'distance']
metric = ['euclidean', 'manhattan']

## convert to dictionary
hyperparameters = dict(n_neighbors=n_neighbors, weights=weights, metric=metric)

## Making model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
grid_search = GridSearchCV(estimator=knn, param_grid=hyperparameters, n_jobs=-1, cv

best_model = grid_search.fit(X_train, y_train)

## Making Predictions
knn_pred = best_model.predict(X_test)

## Print the evaluation metrics
Knn_score = accuracy_score(y_test, knn_pred)
print("Classification Report is:\n",classification_report(y_test,knn_pred))
print("\n F1:\n",f1_score(y_test,knn_pred))
print("\n Precision score is:\n",precision_score(y_test,knn_pred))
print("\n Recall score is:\n",recall_score(y_test,knn_pred))
```

```
Classification Report is:
               precision    recall  f1-score   support

           0       0.98      0.76      0.86      1633
           1       0.80      0.99      0.88      1553

    accuracy                           0.87      3186
   macro avg       0.89      0.88      0.87      3186
weighted avg       0.89      0.87      0.87      3186


 F1:
 0.8828102505038872

 Precision score is:
 0.7984375

 Recall score is:
 0.9871216999356085
```

6. Gradient Boosting Machines

In [34]:
```
## Load the required modules
from sklearn.ensemble import GradientBoostingClassifier

## Initialize the softwares
gbc = GradientBoostingClassifier()

## Define the hyperparameters
parameters = {
    'loss': ['deviance', 'exponential'],
    'learning_rate': [0.001, 0.1, 1, 10],
    'n_estimators': [100, 150, 180, 200]
}

## Fit the model with the best hyperparameters
grid_search_gbc = GridSearchCV(gbc, parameters, cv = 5, n_jobs = -1, verbose = 1)
grid_search_gbc.fit(X_train, y_train)

## Make predictions
gbc_pred = grid_search_gbc.predict(X_test)

## Evaluate performance
gbc_score = accuracy_score(y_test, gbc_pred)
print("Classification Report:\n", classification_report(y_test, gbc_pred))
print("F1 Score:", f1_score(y_test, gbc_pred))
print("Precision:", precision_score(y_test, gbc_pred))
print("Recall:", recall_score(y_test, gbc_pred))
print("Accuracy:", accuracy_score(y_test, gbc_pred))
```

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
Classification Report:
               precision    recall  f1-score   support

           0       0.90      0.83      0.87      1633
           1       0.84      0.91      0.87      1553

    accuracy                           0.87      3186
   macro avg       0.87      0.87      0.87      3186
weighted avg       0.87      0.87      0.87      3186

F1 Score: 0.8714462299134734
Precision: 0.8377896613190731
Recall: 0.9079201545396007
Accuracy: 0.869428750784683
```

7. Ada Boost Classifier

In [35]:
```
## Load the required modules
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

## Define the model
base_estimator = DecisionTreeClassifier(max_depth = 1)
ada = AdaBoostClassifier(estimator = base_estimator, n_estimators=180, learning_rat

## Fit the model
```

```
ada.fit(X_train, y_train)

## Make predictions
ada_pred = ada.predict(X_test)

## Evaluate performance
ada_score = accuracy_score(y_test, ada_pred)
print("Classification Report:\n", classification_report(y_test, ada_pred))
print("F1 Score:", f1_score(y_test, ada_pred))
print("Precision:", precision_score(y_test, ada_pred))
print("Recall:", recall_score(y_test, ada_pred))
print("Accuracy:", accuracy_score(y_test, ada_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.78      0.79      0.79      1633
           1       0.77      0.77      0.77      1553

    accuracy                           0.78      3186
   macro avg       0.78      0.78      0.78      3186
weighted avg       0.78      0.78      0.78      3186

F1 Score: 0.7736943907156673
Precision: 0.7746933505487411
Recall: 0.77269800386349
Accuracy: 0.7796610169491526
```

8. Voting Classifier

In [36]:

```
## Load the required module
from sklearn.ensemble import VotingClassifier

## Define the base classifiers
classifiers = [('Logistic Regression', reg), ('K Nearest Neighbours', knn), ('Suppo

## Initialize the model
vc = VotingClassifier(estimators = classifiers)

## Fit the model
vc.fit(X_train, y_train)

## Make predictions
vc_pred = vc.predict(X_test)

## Evaluate performance
vc_score = accuracy_score(y_test, vc_pred)
print("Classification Report:\n", classification_report(y_test, vc_pred))
print("F1 Score:", f1_score(y_test, vc_pred))
print("Precision:", precision_score(y_test, vc_pred))
print("Recall:", recall_score(y_test, vc_pred))
print("Accuracy:", accuracy_score(y_test, vc_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.76      0.79      1633
           1       0.77      0.81      0.79      1553

    accuracy                           0.79      3186
   macro avg       0.79      0.79      0.79      3186
weighted avg       0.79      0.79      0.79      3186

F1 Score: 0.7878787878787878
Precision: 0.7651699029126213
Recall: 0.8119768190598841
Accuracy: 0.7868801004394225
```

Model Comparison

In [37]:
```
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'KNN', 'SVM', 'Random Forest Classifier', 'Vot
              'Gradient Boosting Classifier', 'xgboost'],
    'Score': [reg_score, Knn_score, svm_score, RF_score, vc_score, ada_score, gbc_s
})

models.sort_values(by = 'Score', ascending = False)
```

Out[37]:

	Model	Score
3	Random Forest Classifier	0.950408
7	xgboost	0.897677
1	KNN	0.872254
6	Gradient Boosting Classifier	0.869429
2	SVM	0.820465
4	Voting Classifier	0.786880
5	Ada Boost Classifier	0.779661
0	Logistic Regression	0.688952

In [38]:
```
## Lets save our model using pickle
import pickle as pkl
pkl.dump(grid_search, open("bank_customer_chrun.sav","wb"))
```