CUSTOMER CHURN PREDICTION

Problem Statement:

Develop a machine learning model to predict customer churn for a U.S. bank using historical customer data. The dataset includes customer demographics, account details, and usage behavior. The goal is to identify potential churners and help the bank take proactive measures to retain customers.

Import libraries

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Import dataset

In [6]:
```
df = pd.read_csv("Churn_Modelling.csv")
df.head()
```

Out[6]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

Exploratery Data Analysis

In [8]:
```
df.shape
```

Out[8]: (10000, 14)

In [9]:
```
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
```

Summary statistics for numerical features

In [11]:
```
df.describe()
```

Out[11]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiv
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	100(
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	

Check for missing values

In [13]:

```
df.isna().sum()
```

Out[13]:

```
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
```

Check for duplicate records

In [15]:

```
df.duplicated().sum()
```

Out[15]:

```
0
```

In [16]:

```
df.rename(columns={'Exited': 'Churn'}, inplace=True)
```

In [17]:

```
df.columns
```

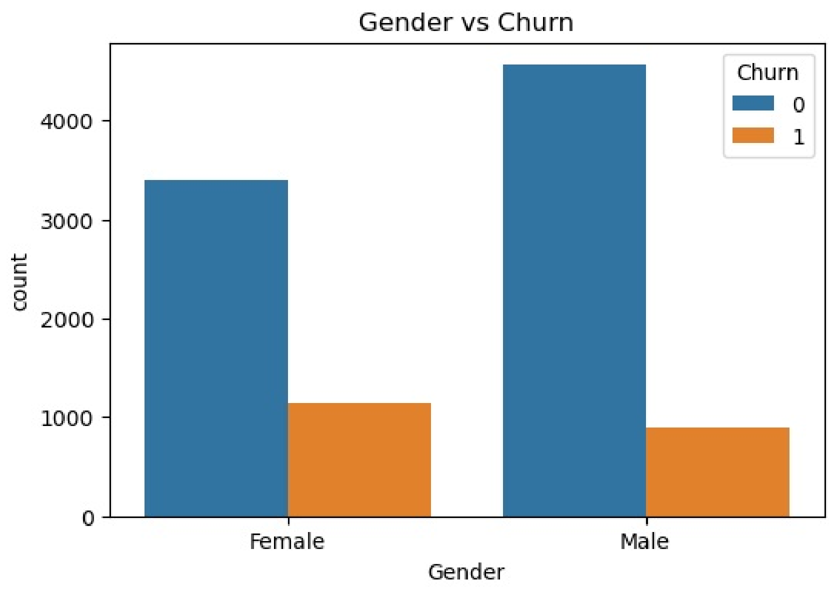
Out[17]:

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

Gender & Churn Distribution

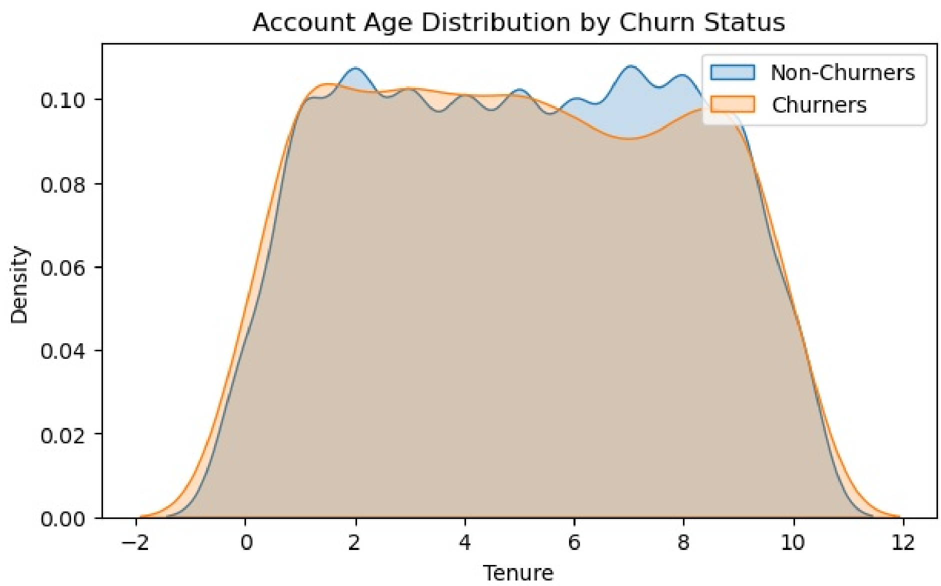
In [19]:

```
plt.figure(figsize=(6,4))
sns.countplot(data=df, x='Gender', hue='Churn')#, palette='viridis'
plt.title("Gender vs Churn")
plt.show()
```



Relationship Between Tenure & Churn

In [21]:
```
plt.figure(figsize=(7,4))
sns.kdeplot(df[df['Churn'] == 0]['Tenure'], label="Non-Churners", shade=True)
sns.kdeplot(df[df['Churn'] == 1]['Tenure'], label="Churners", shade=True)
plt.legend()
plt.title("Account Age Distribution by Churn Status")
plt.show()
```



Numerical Dataframe

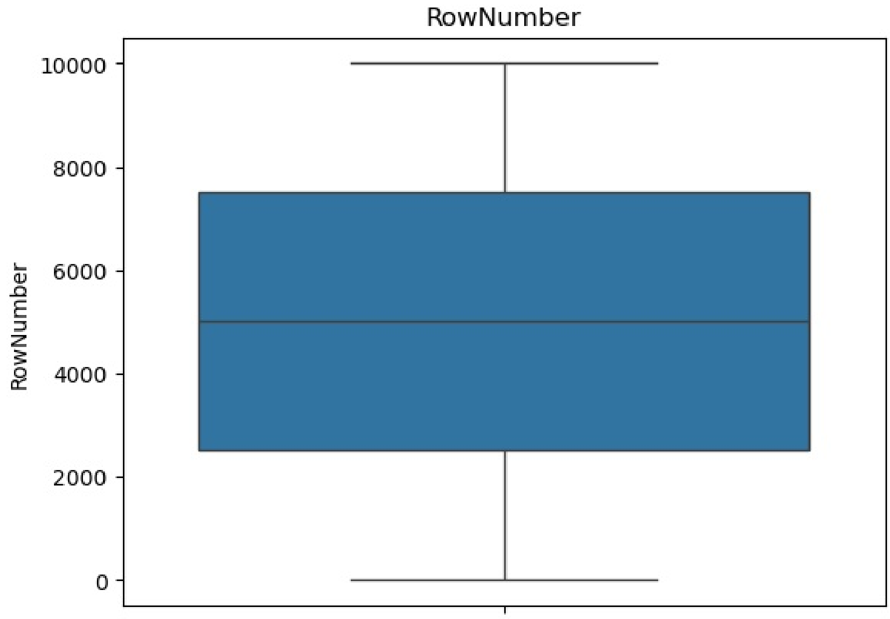
In [23]:
```
num_df = df.select_dtypes(exclude="object")
num_df.head(1)
```

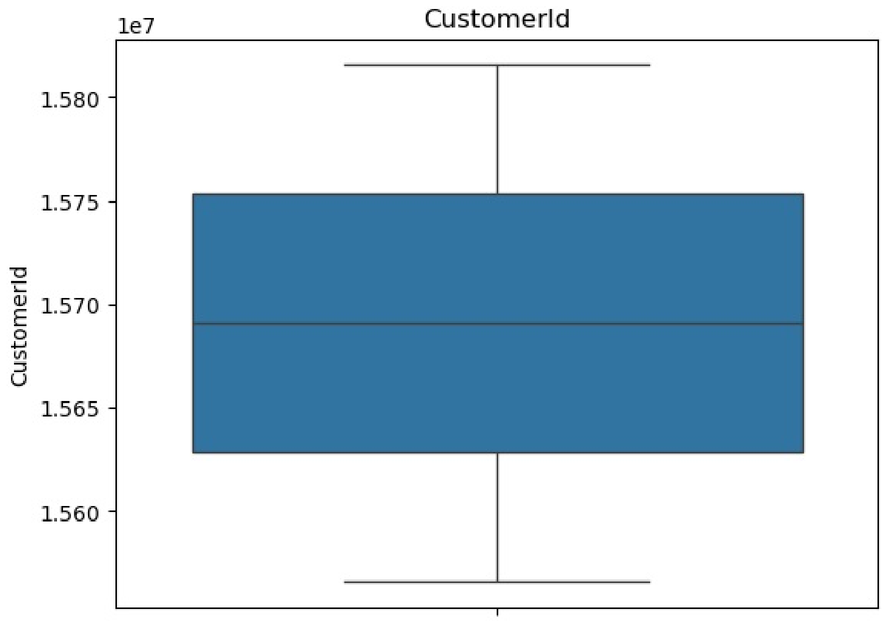
Out[23]:

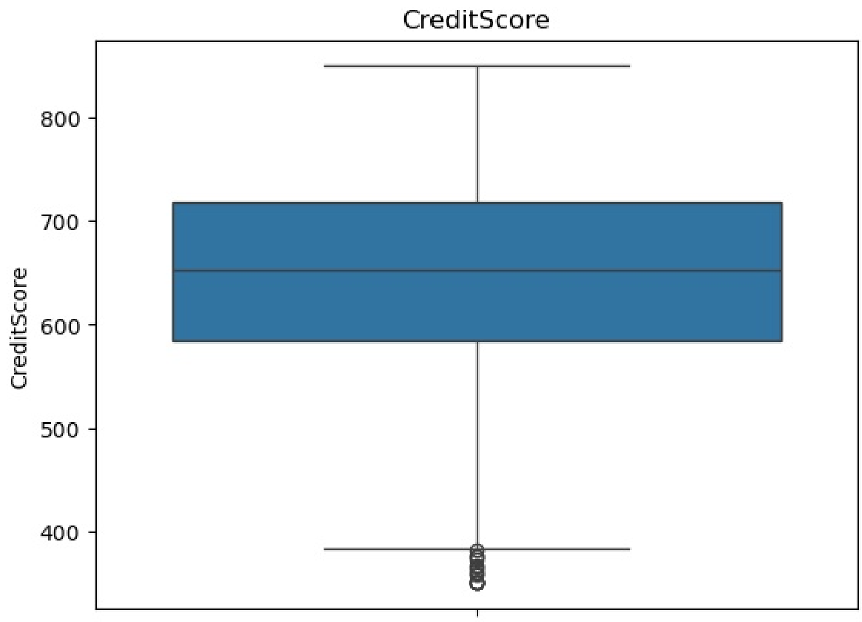
	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Cl
0	1	15634602	619	42	2	0.0	1	1	1	101348.88	

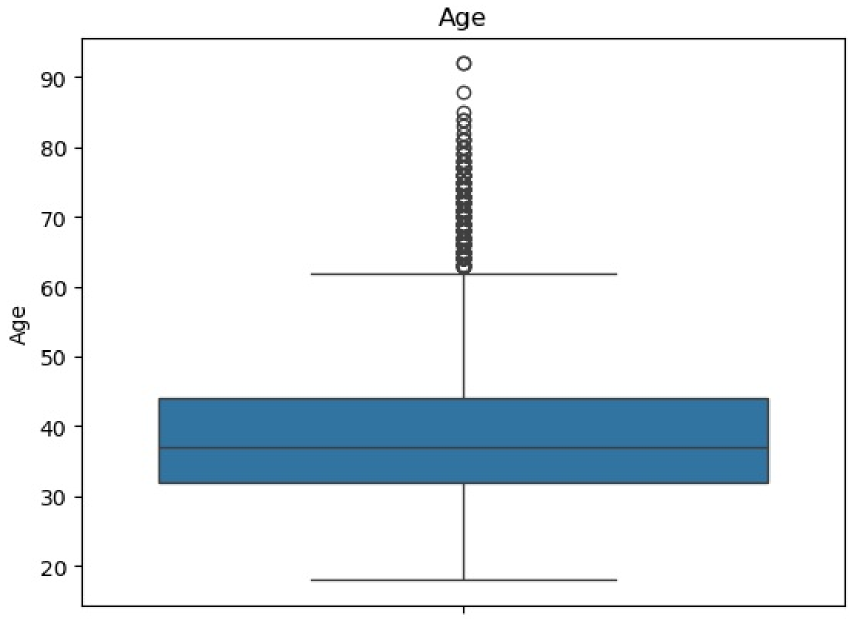
Check for outliers

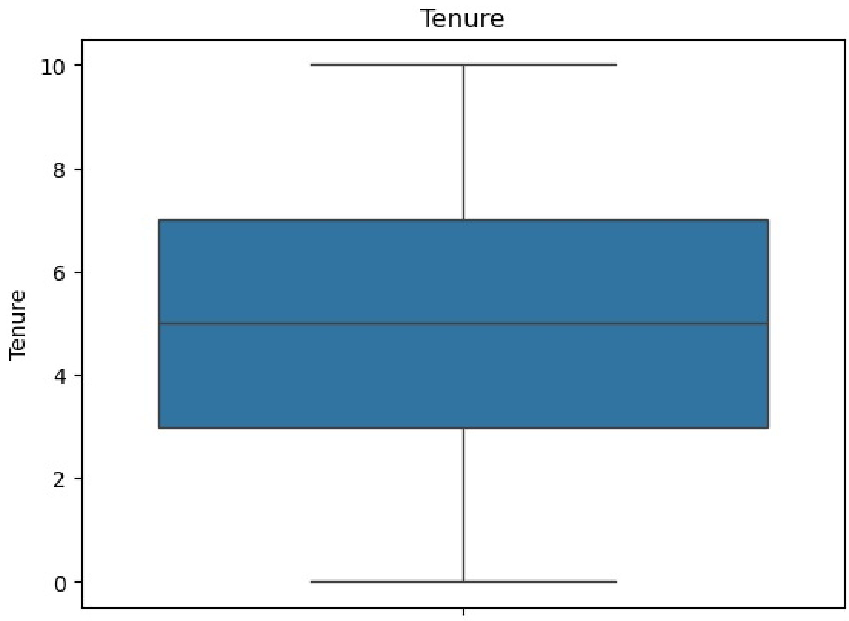
In [25]:
```
for col in num_df:
    sns.boxplot(num_df[col])
    plt.title(col)
    plt.show()
```

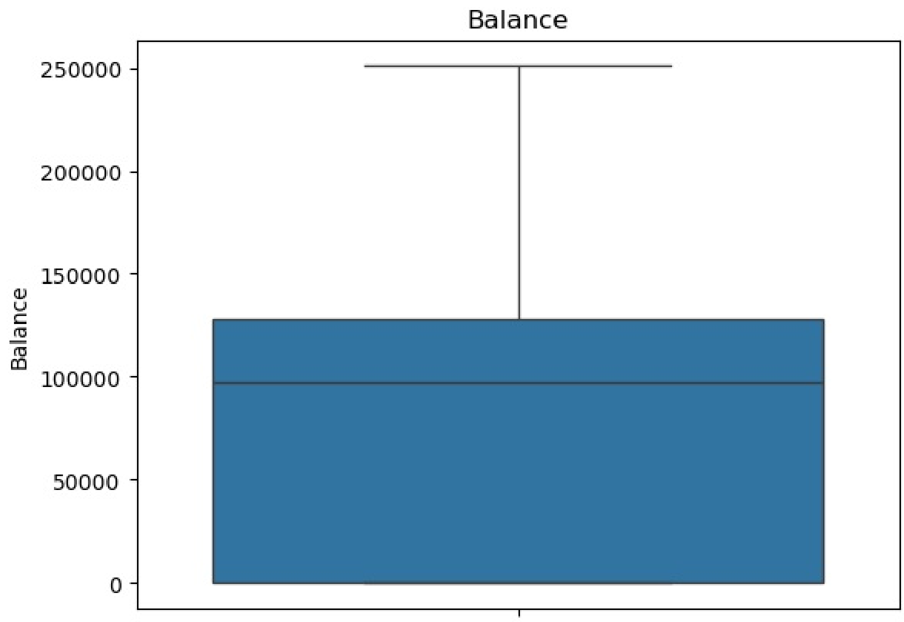

RowNumber
10000
8000
6000
4000
2000
0
RowNumber

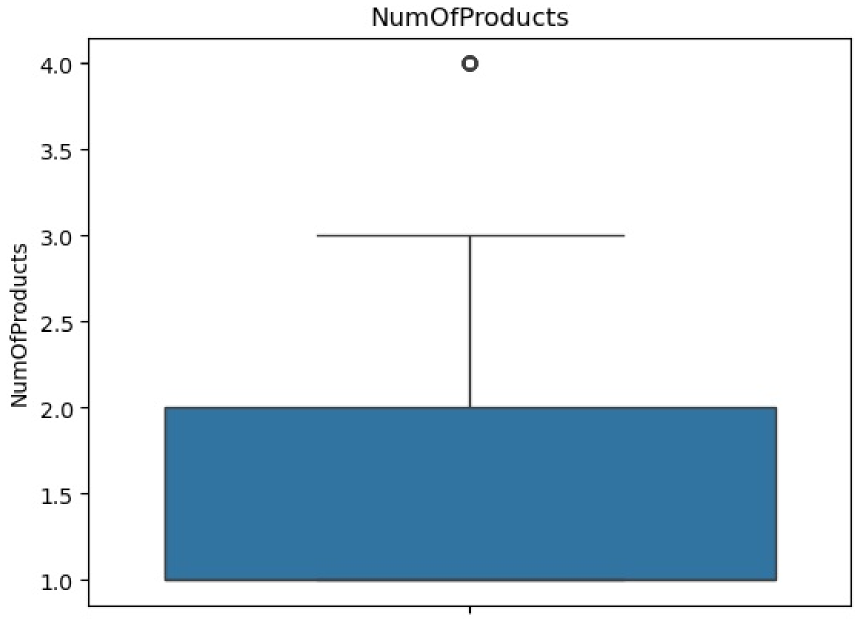

CustomerId
1e7
1.580
1.575
1.570
1.565
1.560
CustomerId

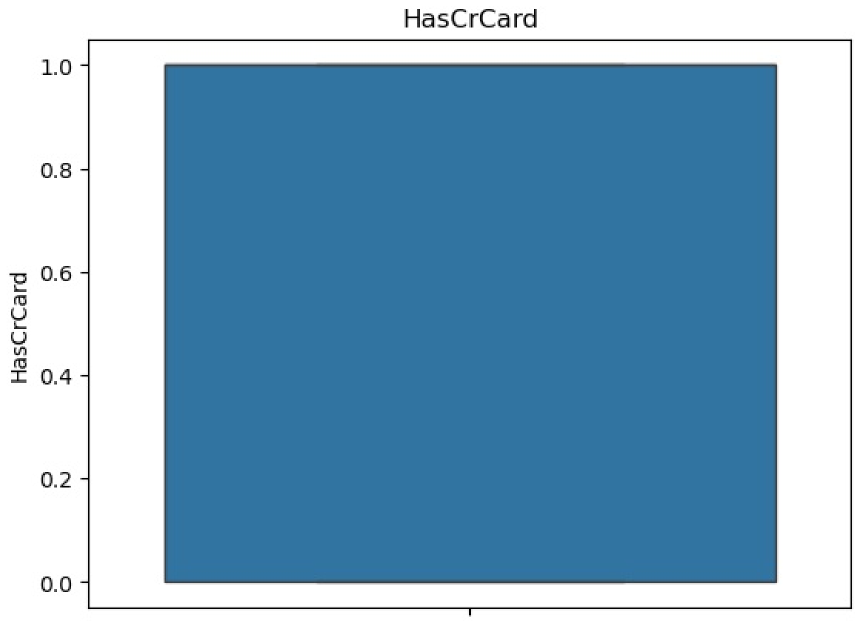

CreditScore
800
700
600
500
400
CreditScore

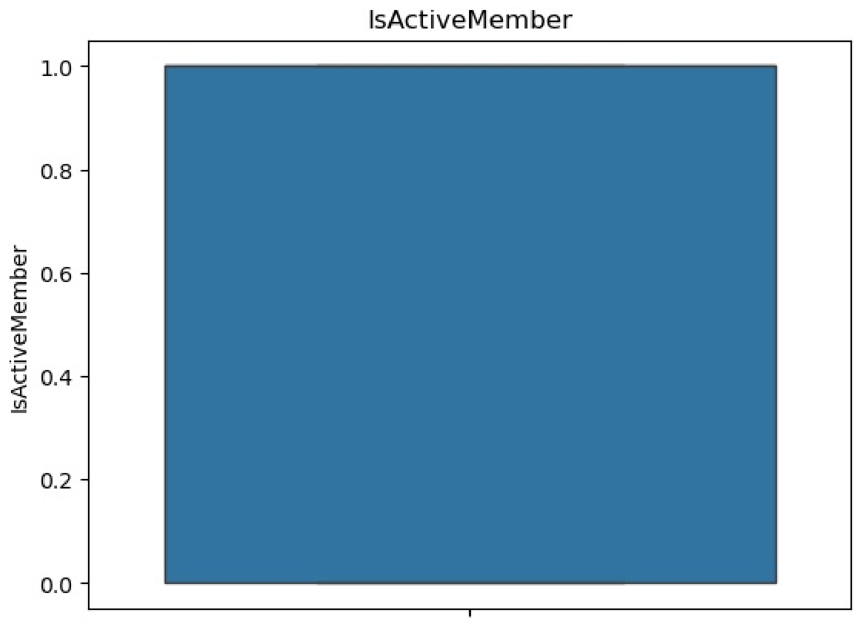

Age
90
80
70
60
50
40
30
20
Age

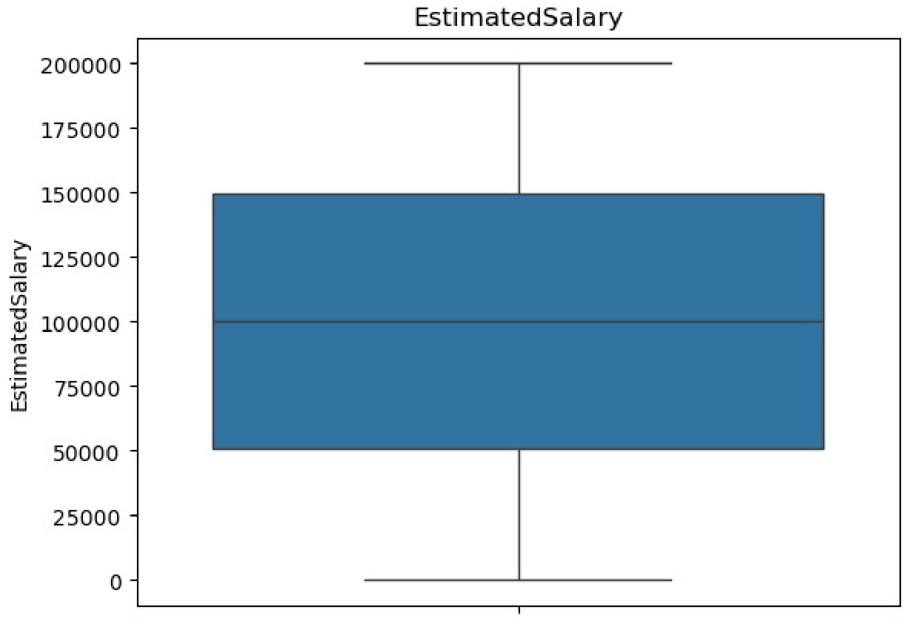

Tenure
10
8
6
4
2
0
Tenure

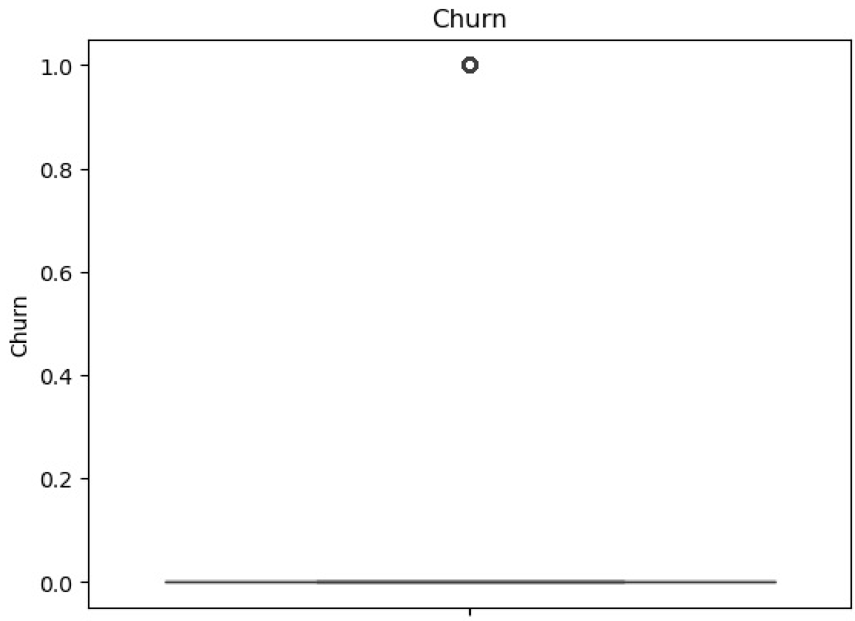

Balance
250000
200000
150000
100000
50000
0
Balance


NumOfProducts
4.0
3.5
3.0
2.5
2.0
1.5
1.0
NumOfProducts


HasCrCard
1.0
0.8
0.6
0.4
0.2
0.0
HasCrCard

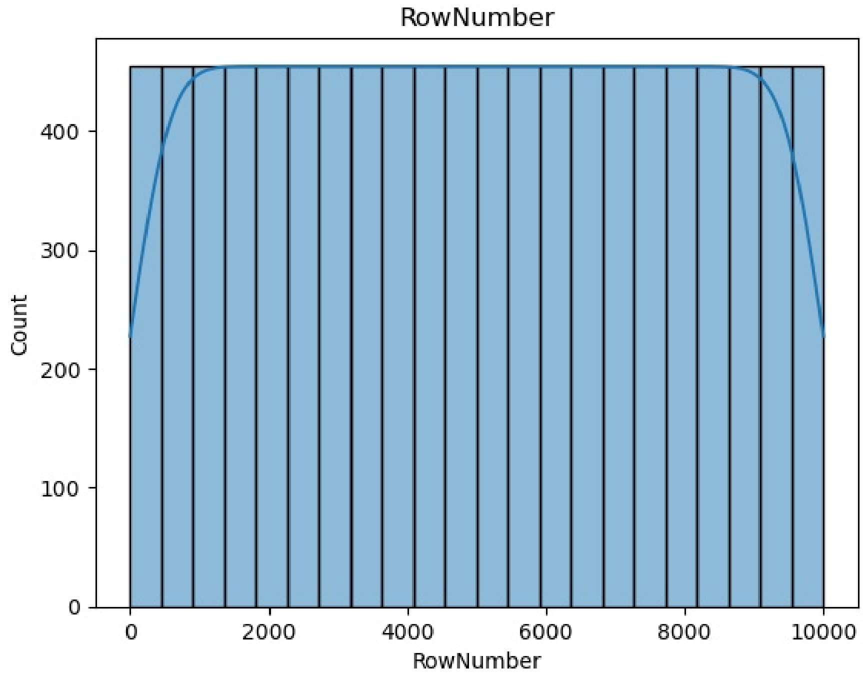

IsActiveMember
1.0
0.8
0.6
0.4
0.2
0.0
IsActiveMember

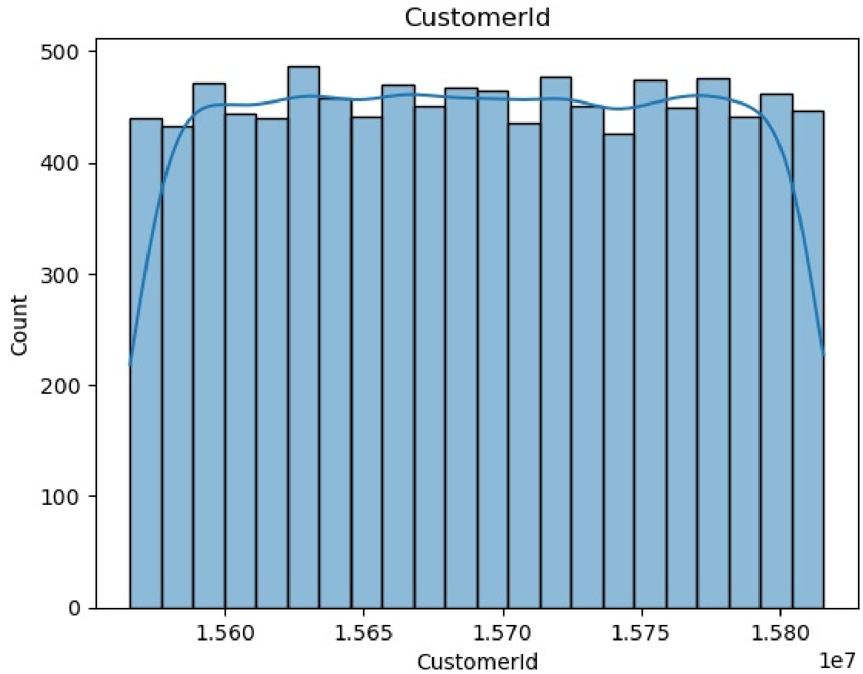


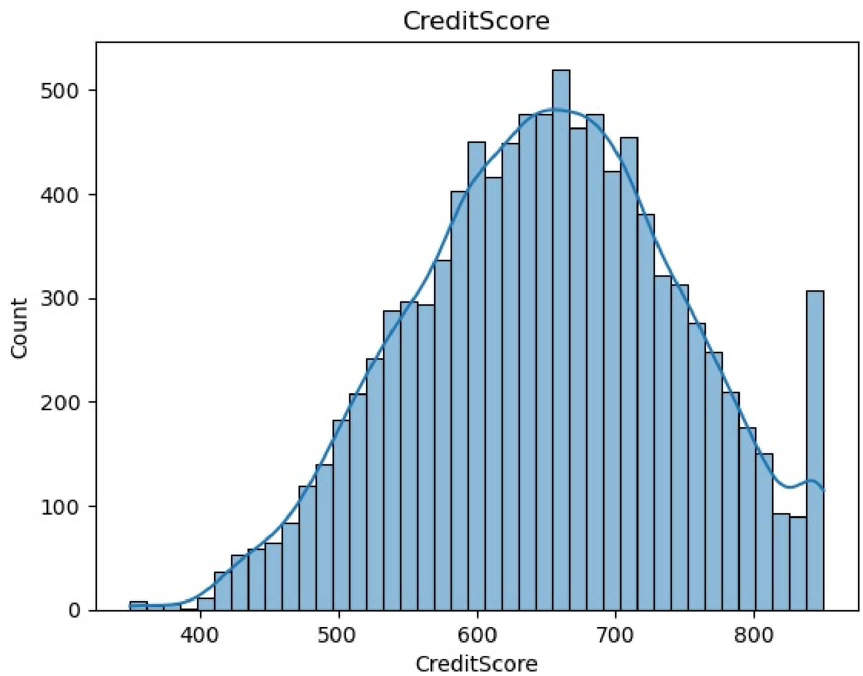


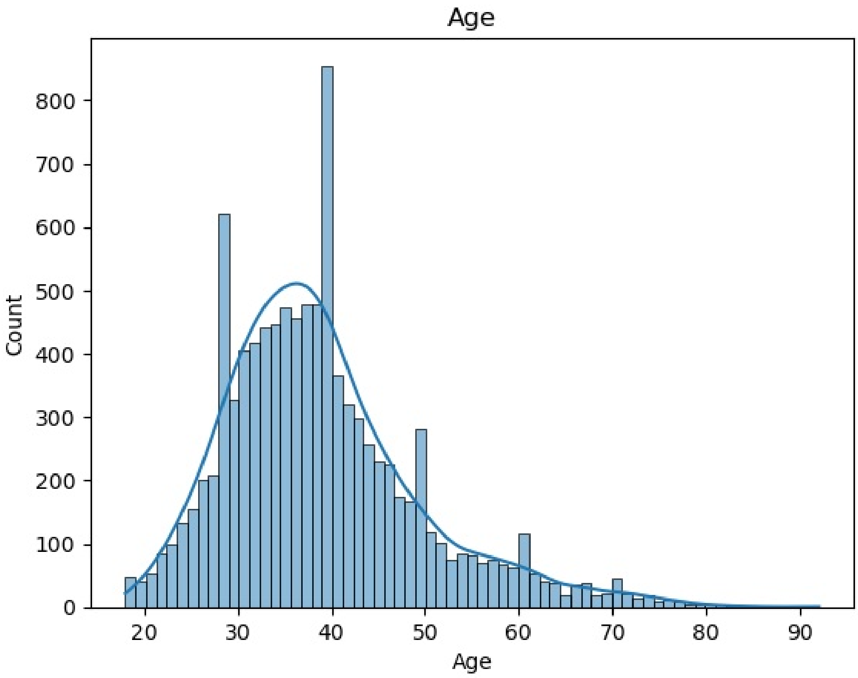
Check for skewness

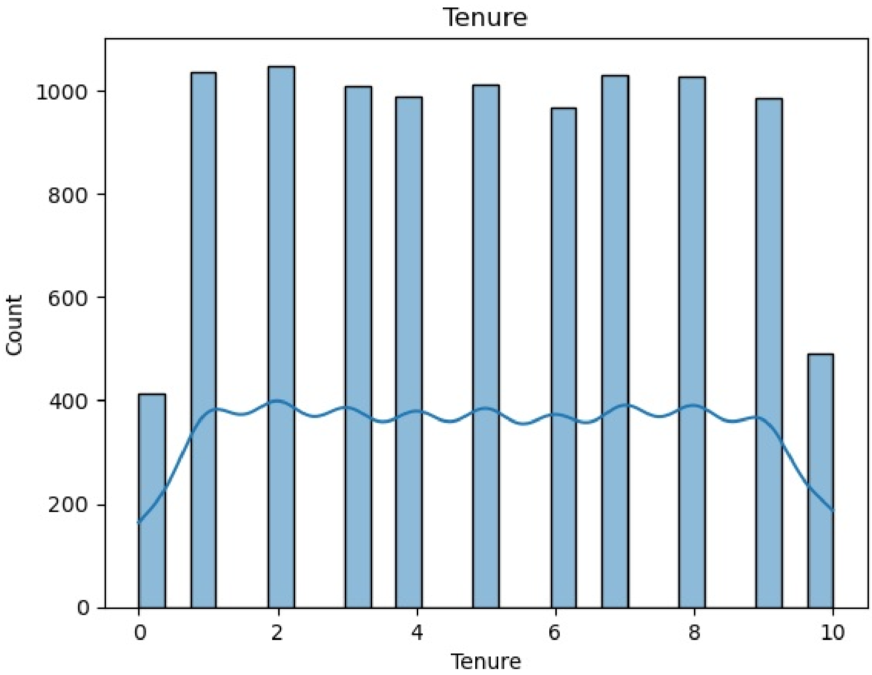
In [27]:
```
for col in num_df:
    sns.histplot(num_df[col], kde=True)
    plt.title(col)
    plt.show()
```

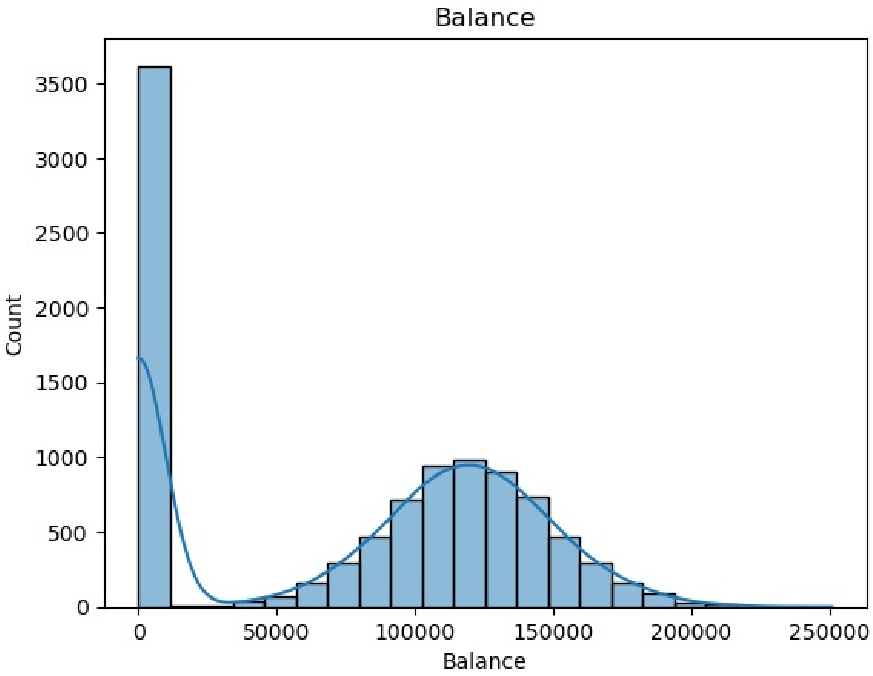

RowNumber
Count
RowNumber
0
100
200
300
400
0
2000
4000
6000
8000
10000

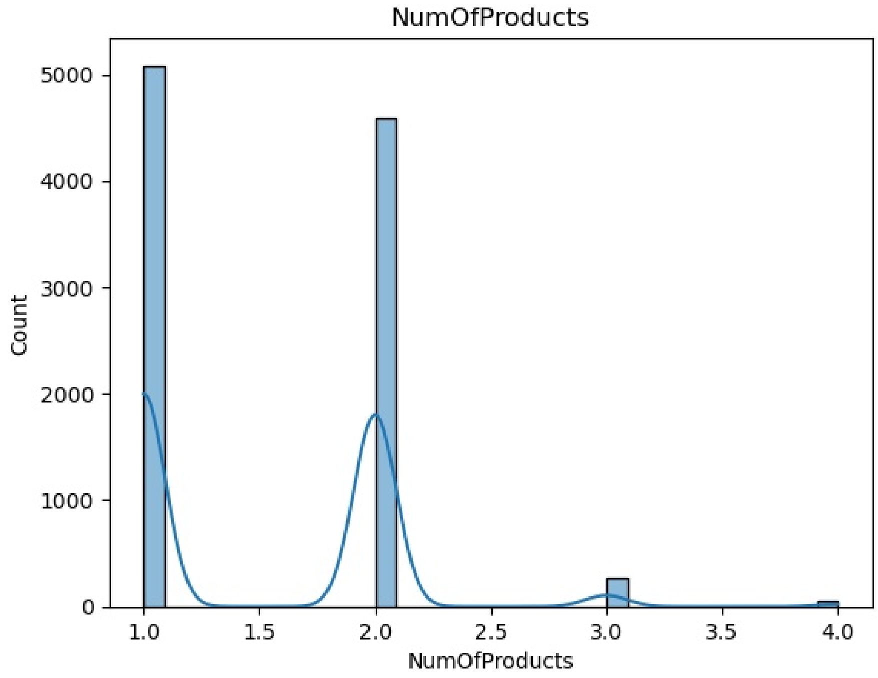

CustomerId
Count
CustomerId
0
100
200
300
400
500
1.560
1.565
1.570
1.575
1.580
1e7

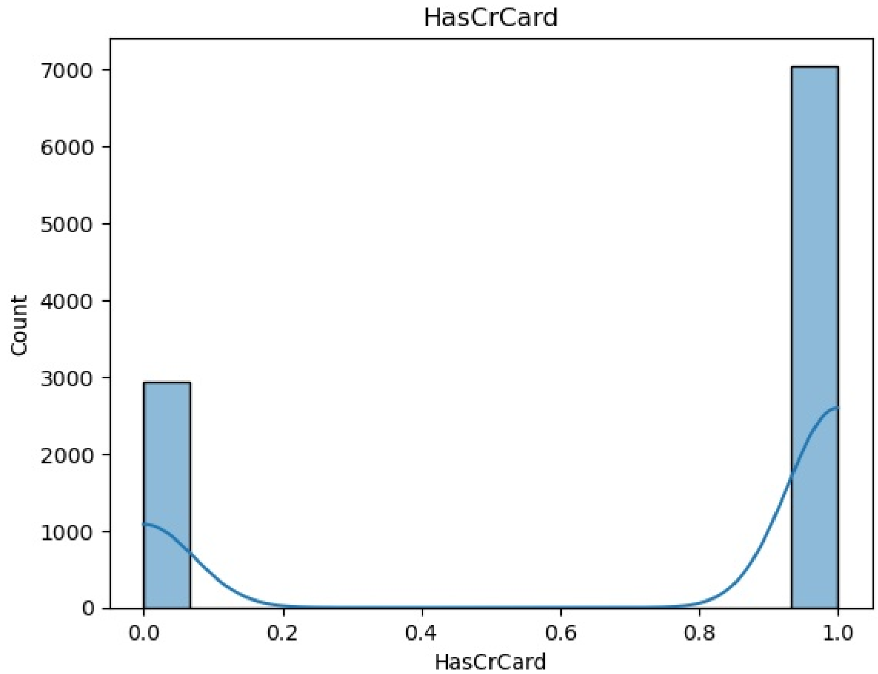

CreditScore
Count
CreditScore
0
100
200
300
400
500
400
500
600
700
800

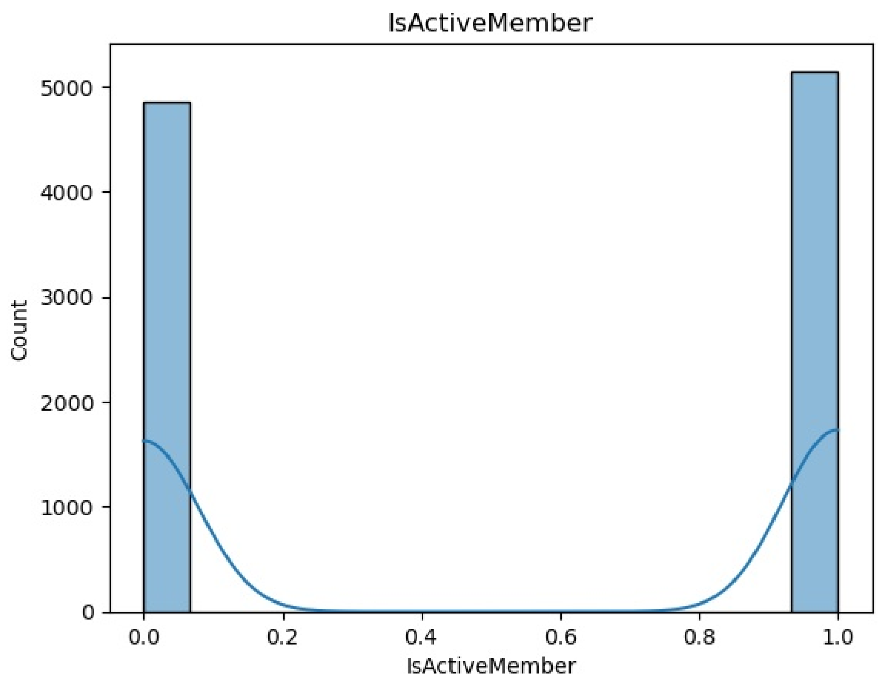

Age
Count
0
100
200
300
400
500
600
700
800
20
30
40
50
60
70
80
90
Age

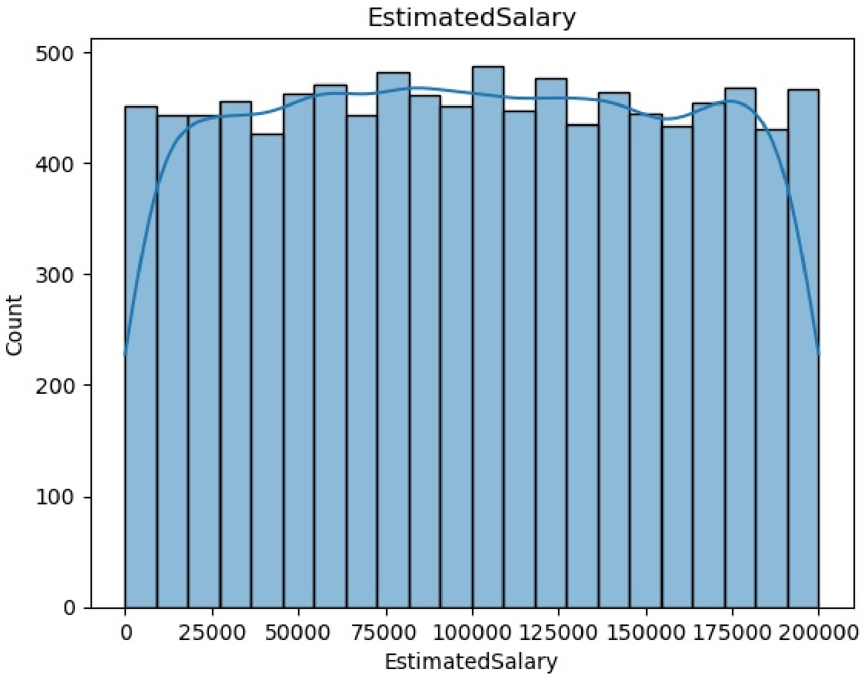

Tenure
Count
0
200
400
600
800
1000
0
2
4
6
8
10
Tenure

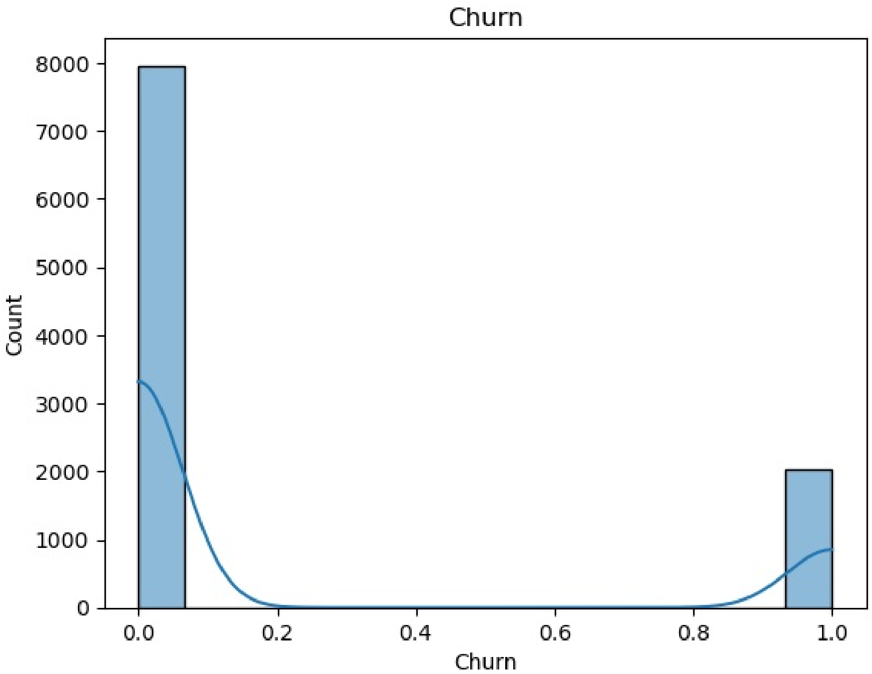

Balance
Count
0
500
1000
1500
2000
2500
3000
3500
0
50000
100000
150000
200000
250000
Balance


NumOfProducts
Count
0
1000
2000
3000
4000
5000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
NumOfProducts


HasCrCard
Count
0
1000
2000
3000
4000
5000
6000
7000
0.0
0.2
0.4
0.6
0.8
1.0
HasCrCard


IsActiveMember
Count
0
1000
2000
3000
4000
5000
0.0
0.2
0.4
0.6
0.8
1.0
IsActiveMember





In [28]:
```
from scipy.stats import skew
for col in num_df:
    print(f"{col} --> {num_df[col].skew()}")
```

```
RowNumber --> 0.0
CustomerId --> 0.001149145900554239
CreditScore --> -0.07160660820092675
Age --> 1.0113202630234552
Tenure --> 0.01099145797717904
Balance --> -0.14110871094154384
NumOfProducts --> 0.7455678882823168
HasCrCard --> -0.9018115952400578
IsActiveMember --> -0.06043662833499078
EstimatedSalary --> 0.0020853576615585162
Churn --> 1.4716106649378211
```

Categorical columns

In [30]:
```
obj_df = df.select_dtypes(include="object")
obj_df.head(1)
```

Out[30]:

	Surname	Geography	Gender
0	Hargrave	France	Female

In [31]:
```
for col in obj_df:
    print(obj_df[col].value_counts())
    print("="*40)
```

```
Surname
Smith       32
Scott       29
Martin      29
Walker      28
Brown       26
            ..
Izmailov     1
Bold         1
Bonham       1
Poninski     1
Burbidge     1
Name: count, Length: 2932, dtype: int64
========================================
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
========================================
Gender
Male      5457
Female    4543
Name: count, dtype: int64
========================================
```

In [32]:
```
from sklearn.preprocessing import LabelEncoder
```

In [33]:
```
lb = LabelEncoder()
```

In [34]:
```
#Label Encoding
obj_df['Geography'] = lb.fit_transform(obj_df['Geography'])
obj_df['Gender'] = lb.fit_transform(obj_df['Gender'])
```

In [35]:
```
obj_df.head()
```

Out[35]:

	Surname	Geography	Gender
0	Hargrave	0	0
1	Hill	2	0
2	Onio	0	0
3	Boni	0	0
4	Mitchell	2	0

Scaling numerical columns

In [36]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [37]:
```
Scaler = MinMaxScaler()
```

In [38]:
```
columns = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']
```

In [39]:
```
for col in columns:
    num_df[col] = Scaler.fit_transform(num_df[[col]])
```

In [40]:
```
num_df.head(2)
```

Out[40]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSala
0	1	15634602	0.538	0.324324	0.2	0.000000	0.0	1	1	0.50673
1	2	15647311	0.516	0.310811	0.1	0.334031	0.0	0	1	0.56270

In [41]:
```
#Concat numerical and categorical dataframes
data = pd.concat([obj_df, num_df], axis=1)
data
```

Out[41]:

	Surname	Geography	Gender	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCa
0	Hargrave	0	0	1	15634602	0.538	0.324324	0.2	0.000000	0.000000	
1	Hill	2	0	2	15647311	0.516	0.310811	0.1	0.334031	0.000000	
2	Onio	0	0	3	15619304	0.304	0.324324	0.8	0.636357	0.666667	
3	Boni	0	0	4	15701354	0.698	0.283784	0.1	0.000000	0.333333	
4	Mitchell	2	0	5	15737888	1.000	0.337838	0.2	0.500246	0.000000	
...	...	...	...	...	...	...	...	...	...	...	
9995	Obijiaku	0	1	9996	15606229	0.842	0.283784	0.5	0.000000	0.333333	
9996	Johnstone	0	1	9997	15569892	0.332	0.229730	1.0	0.228657	0.000000	
9997	Liu	0	0	9998	15584532	0.718	0.243243	0.7	0.000000	0.000000	
9998	Sabbatini	1	1	9999	15682355	0.844	0.324324	0.3	0.299226	0.333333	
9999	Walker	0	0	10000	15628319	0.884	0.135135	0.4	0.518708	0.000000	

10000 rows × 14 columns

In [42]:
```
#drop columns
data.drop(columns=['Surname', 'RowNumber', 'CustomerId'],axis=1,inplace=True)
```

These columns (Surname, RowNumber, CustomerId) contain unique or irrelevant values that do not contribute to patterns in churn prediction. Keeping them may introduce noise, so dropping them helps the model focus on meaningful features and improve performance.

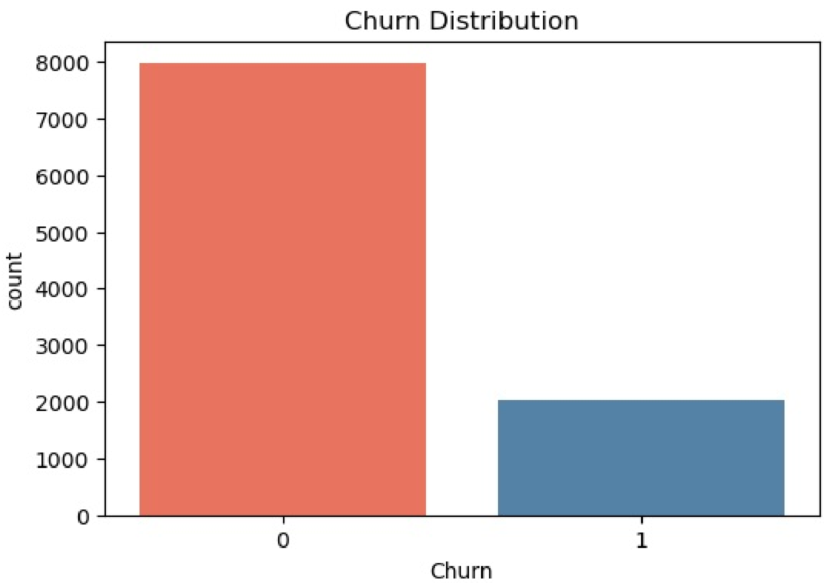
In [44]:
```
data.head()
```

Out[44]:

	Geography	Gender	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Cl
0	0	0	0.538	0.324324	0.2	0.000000	0.000000	1	1	0.506735	
1	2	0	0.516	0.310811	0.1	0.334031	0.000000	0	1	0.562709	
2	0	0	0.304	0.324324	0.8	0.636357	0.666667	1	0	0.569654	
3	0	0	0.698	0.283784	0.1	0.000000	0.333333	0	0	0.469120	
4	2	0	1.000	0.337838	0.2	0.500246	0.000000	1	1	0.395400	

Churn Distribution

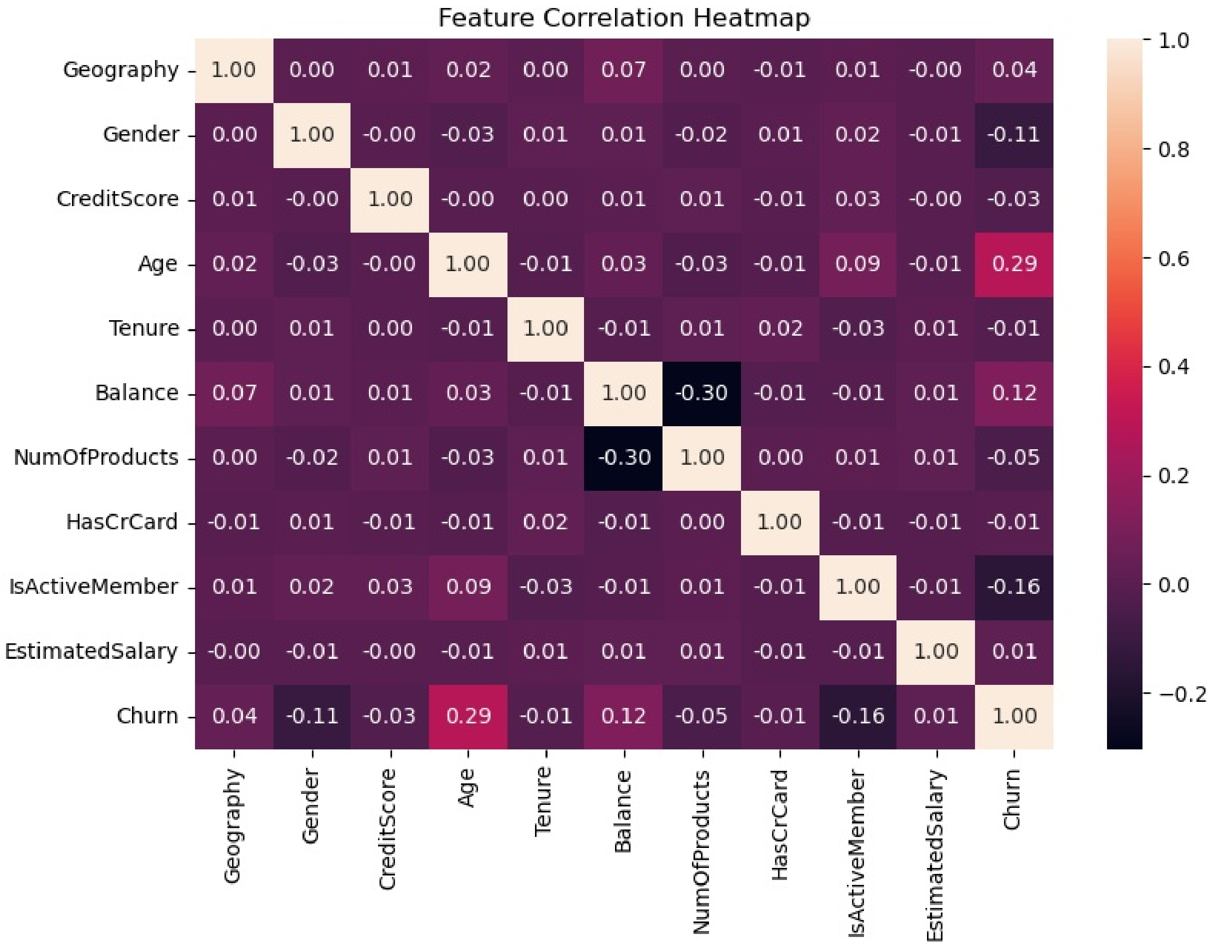
In [46]:
```
plt.figure(figsize=(6,4))
sns.countplot(data=data, x='Churn', palette=['#FF6347', '#4682B4'])  # Customize colors
plt.title("Churn Distribution")
plt.show()
```



In [47]:
```
# Percentage of churners
churn_rate = data["Churn"].value_counts(normalize=True) * 100
print("Churn Rate:\n", churn_rate)
```

```
Churn Rate:
 Churn
0    79.63
1    20.37
Name: proportion, dtype: float64
```

In [48]:
```python
plt.figure(figsize=(9,6))
sns.heatmap(data.corr(), fmt=".2f", annot=True)
plt.title("Feature Correlation Heatmap")
plt.show()
```



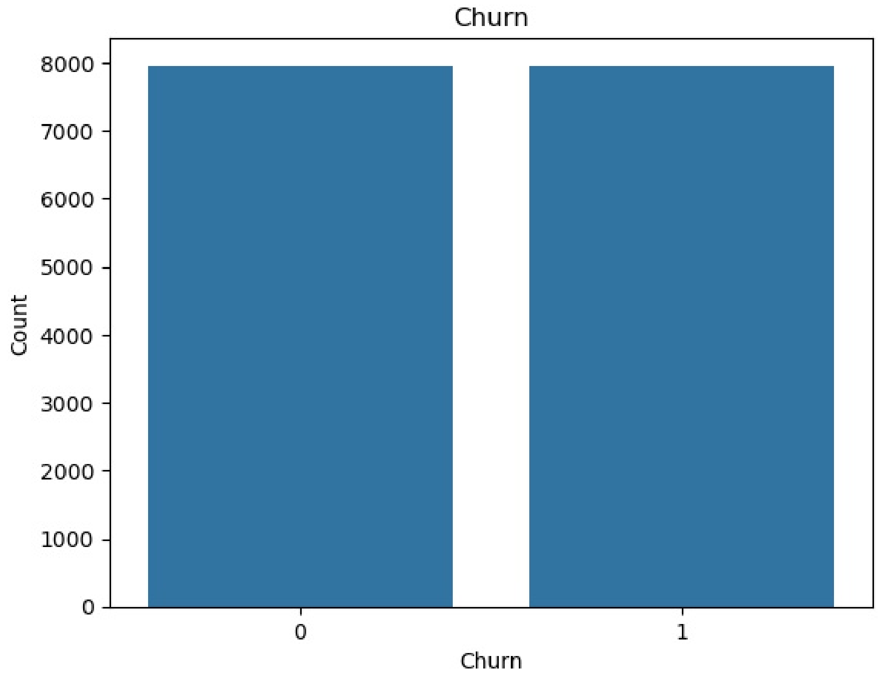
Handling imbalanced data

In [50]:
```python
from imblearn.over_sampling import SMOTE
x = data.drop('Churn', axis=1)
y = data['Churn']

smt = SMOTE(random_state = 10)
x_sample, y_sample = smt.fit_resample(x,y)

x = x_sample
y = y_sample
```

In [51]:
```python
sns.countplot(x=y)
plt.title('Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



In [54]:
```
#Train-Test-Split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=11)
```

Logistic Regression Model

In [57]:
```
from sklearn.linear_model import LogisticRegression
```

In [59]:
```
lr_model = LogisticRegression()
```

In [61]:
```
lr_model.fit(x_train, y_train)
```

Out[61]:
```
▾ LogisticRegression
LogisticRegression()
```

In [75]:
```
from sklearn.metrics import accuracy_score, classification_report
```

In [77]:
```
y_pred_lr = lr_model.predict(x_test)
lr_accuracy_testing = accuracy_score(y_test, y_pred_lr)*100
print("Accuracy score testing:",lr_accuracy_testing)
```

```
Accuracy score testing: 69.36597614563716
```

In [79]:
```
y_pred_lrt = lr_model.predict(x_train)
lr_accuracy_training = accuracy_score(y_train, y_pred_lrt)*100
print("Accuracy score traning:",lr_accuracy_training)
```

```
Accuracy score traning: 70.25902668759811
```

In [81]:
```
report_lr = classification_report(y_test,y_pred_lr)
print(report_lr)
```

```
              precision    recall  f1-score   support

           0       0.70      0.70      0.70      1628
           1       0.69      0.69      0.69      1558

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

Random Forest Model

In [84]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [86]:
```
rf_model = RandomForestClassifier(n_estimators=100,max_depth=10,min_samples_split=10,min_samples_leaf=5,max_fea
rf_model.fit(x_train,y_train)
```

Out[86]:

```
RandomForestClassifier
RandomForestClassifier(max_depth=10, min_samples_leaf=5, min_samples_split=10,
                       random_state=42)
```

In [87]:

```
y_pred_rf = rf_model.predict(x_test)
rf_accuracy_testing = accuracy_score(y_test,y_pred_rf)*100
print("Accuracy score testing:",rf_accuracy_testing)
```

```
Accuracy score testing: 84.02385436283741
```

In [90]:

```
y_pred_rft = rf_model.predict(x_train)
rf_accuracy_training = accuracy_score(y_train,y_pred_rft)*100
print("Accuracy score traning:",rf_accuracy_training)
```

```
Accuracy score traning: 88.30455259026687
```

In [92]:

```
report_rf = classification_report(y_test, y_pred_rf)
print(report_rf)
```

```
              precision    recall  f1-score   support

           0       0.85      0.84      0.84      1628
           1       0.83      0.84      0.84      1558

    accuracy                           0.84      3186
   macro avg       0.84      0.84      0.84      3186
weighted avg       0.84      0.84      0.84      3186
```

GdBoost Model

In [95]:

```
from sklearn.ensemble import GradientBoostingClassifier
gdb_classify=GradientBoostingClassifier(n_estimators=200, learning_rate=0.05, max_depth=4)
gdb_classify.fit(x_sample,y_sample)
```

Out[95]:

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.05, max_depth=4, n_estimators=200)
```

In [96]:

```
y_pred_gdbt = gdb_classify.predict(x_train)
gdb_accuracy_training = accuracy_score(y_train, y_pred_gdbt)*100

print("Accuracy score traning:",gdb_accuracy_training)
```

```
Accuracy score traning: 90.02354788069073
```

In [99]:

```
y_pred_gdb = gdb_classify.predict(x_test)
gdb_accuracy_testing = accuracy_score(y_test, y_pred_gdb)*100

print("Accuracy score testing:",gdb_accuracy_testing)
```

```
Accuracy score testing: 88.54362837413686
```

In [101…

```
report_gdb = classification_report(y_test, y_pred_gdb)
print(report_gdb)
```

```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89      1628
           1       0.89      0.87      0.88      1558

    accuracy                           0.89      3186
   macro avg       0.89      0.89      0.89      3186
weighted avg       0.89      0.89      0.89      3186
```

In [103…

```
Dict={'Model':['Logistic_Regression', 'Random_Forest', 'Gradient_Boosting'],
      'Accuracy':[lr_accuracy_testing, rf_accuracy_testing, gdb_accuracy_testing]}
```

In [105…

```
Data=pd.DataFrame(Dict,columns=['Model','Accuracy'])
Data
```

Out[105…

	Model	Accuracy
0	Logistic_Regression	69.365976
1	Random_Forest	84.023854
2	Gradient_Boosting	88.543628