Banking Churn prediction

imports

Install dependecies

In [2]:
```
%%capture
!pip install polars
!pip install folium
!pip install imblearn
!pip install -U seaborn
```

Import dependecies

In [3]:
```
# sagemaker imports
import sagemaker
from sagemaker import get_execution_role
from sagemaker.tuner import IntegerParameter, CategoricalParameter, ContinuousParameter, HyperparameterTuner
from sagemaker.sklearn.estimator import SKLearn
from sagemaker.model import Model
from sagemaker.pipeline import PipelineModel
from sagemaker.predictor import Predictor
from sagemaker.serializers import CSVSerializer
import boto3
```

```
from sklearn.metrics import RocCurveDisplay, auc, roc_curve, accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from time import gmtime, strftime
import polars as pl
import numpy as np
import json
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import folium
import pandas as pd

# constants has all constant variables
from src import constants
```

In [4]:
```
def pie_chart(ax, values, labels, title, space_width=3, startangle=90):
    patches, texts, pcts = ax.pie(
        values, labels=labels, autopct='%.1f%%',
        wedgeprops={'linewidth': space_width, 'edgecolor': 'white'},
        textprops={'size': 'large'},
        startangle=startangle)
    # For each wedge, set the corresponding text label color to the wedge's
    # face color.
    for i, patch in enumerate(patches):
        texts[i].set_color(patch.get_facecolor())
    plt.setp(pcts, color='white')
    plt.setp(texts, fontweight=600)
    ax.set_title(title, fontsize=18)
    plt.tight_layout()
```

Loading data

- i downloaded the dataset from kaggle manualy from here

In [5]:
```
# use polars `read_csv` function to load dataset
bank_chrun_df = pd.read_csv("./datasets/bank_customer_churn.csv")
```

In [ ]:
```
```

In [6]:
```
# preview dataset
bank_chrun_df.head()
```

Out[6]:

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

In [7]:
```
bank_chrun_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
```

In [8]:
```
bank_chrun_df['gender']=bank_chrun_df['gender'].astype('category')
bank_chrun_df['credit_card']=bank_chrun_df['credit_card'].astype('category')
bank_chrun_df['active_member']=bank_chrun_df['active_member'].astype('category')
bank_chrun_df['country']=bank_chrun_df['country'].astype('category')
bank_chrun_df['churn']=bank_chrun_df['churn'].astype('category')
bank_chrun_df['customer_id']=bank_chrun_df['customer_id'].astype('category')
```

In [9]:
```
bank_chrun_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  category
 1   credit_score      10000 non-null  int64   
 2   country           10000 non-null  category
 3   gender            10000 non-null  category
 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  category
 9   active_member     10000 non-null  category
 10  estimated_salary  10000 non-null  float64 
 11  churn             10000 non-null  category
dtypes: category(6), float64(2), int64(4)
memory usage: 874.0 KB
```

- no null values count found

Exploratory data analysis

In [10]:
```
# data describtion
bank_chrun_df.describe()
```

Out[10]:

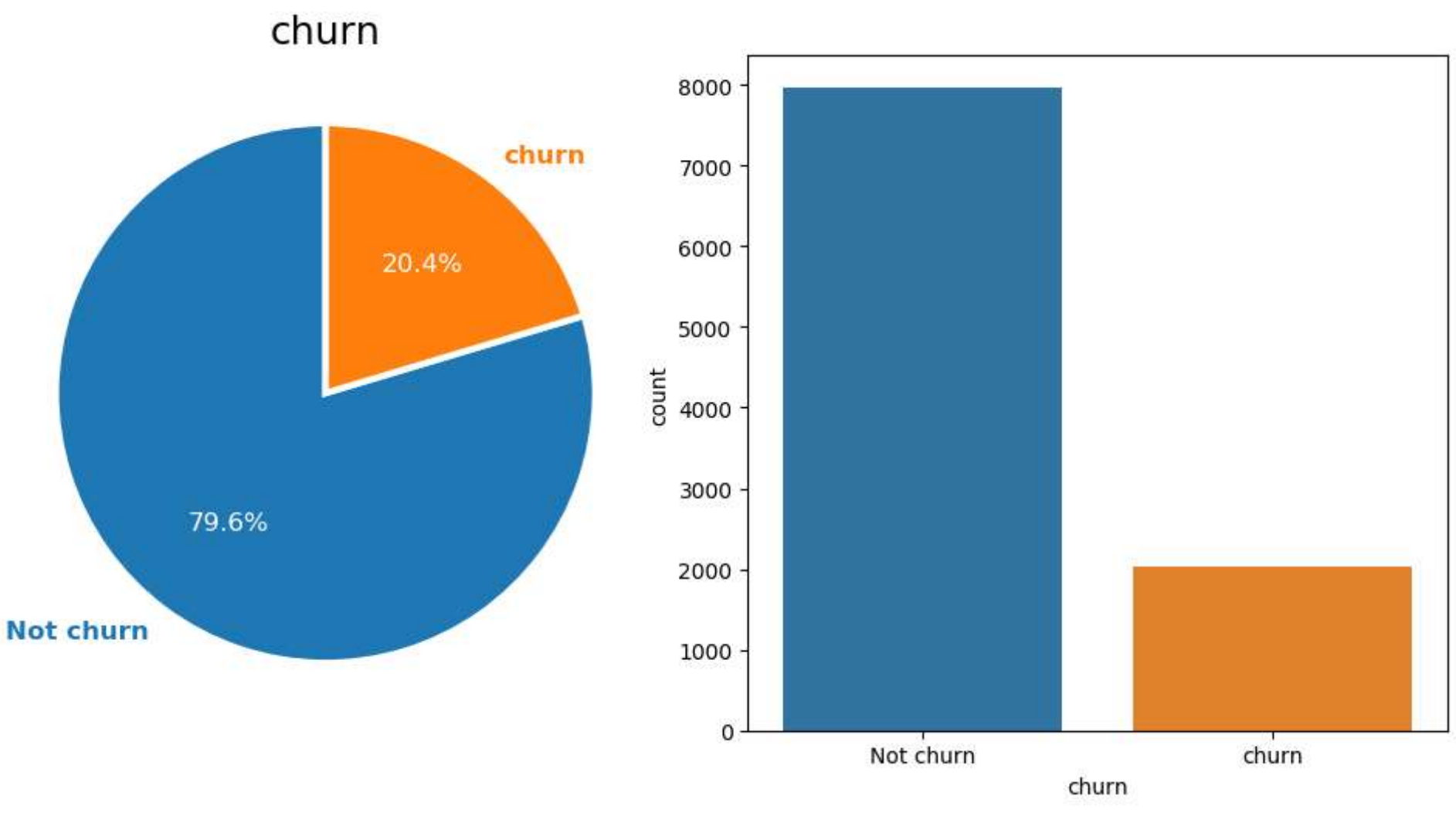
	credit_score	age	tenure	balance	products_number	estimated_salary
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	100090.239881
std	96.653299	10.487806	2.892174	62397.405202	0.581654	57510.492818
min	350.000000	18.000000	0.000000	0.000000	1.000000	11.580000
25%	584.000000	32.000000	3.000000	0.000000	1.000000	51002.110000
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	100193.915000
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	149388.247500
max	850.000000	92.000000	10.000000	250898.090000	4.000000	199992.480000

In [ ]:
```

```

churn column

In [ ]:
```

```

In [11]:
```
fig,ax=plt.subplots(1,2,figsize=(10,5))
values,labels=bank_chrun_df.churn.value_counts(),["Not churn","churn"]
pie_chart(ax[0],values,labels,"churn")
count_plt=sns.countplot(data=bank_chrun_df,x="churn",ax=ax[1])
count_plt.set_xticklabels(labels)


fig.show()
```



In [12]:
```
bank_chrun_df.churn.value_counts()
```

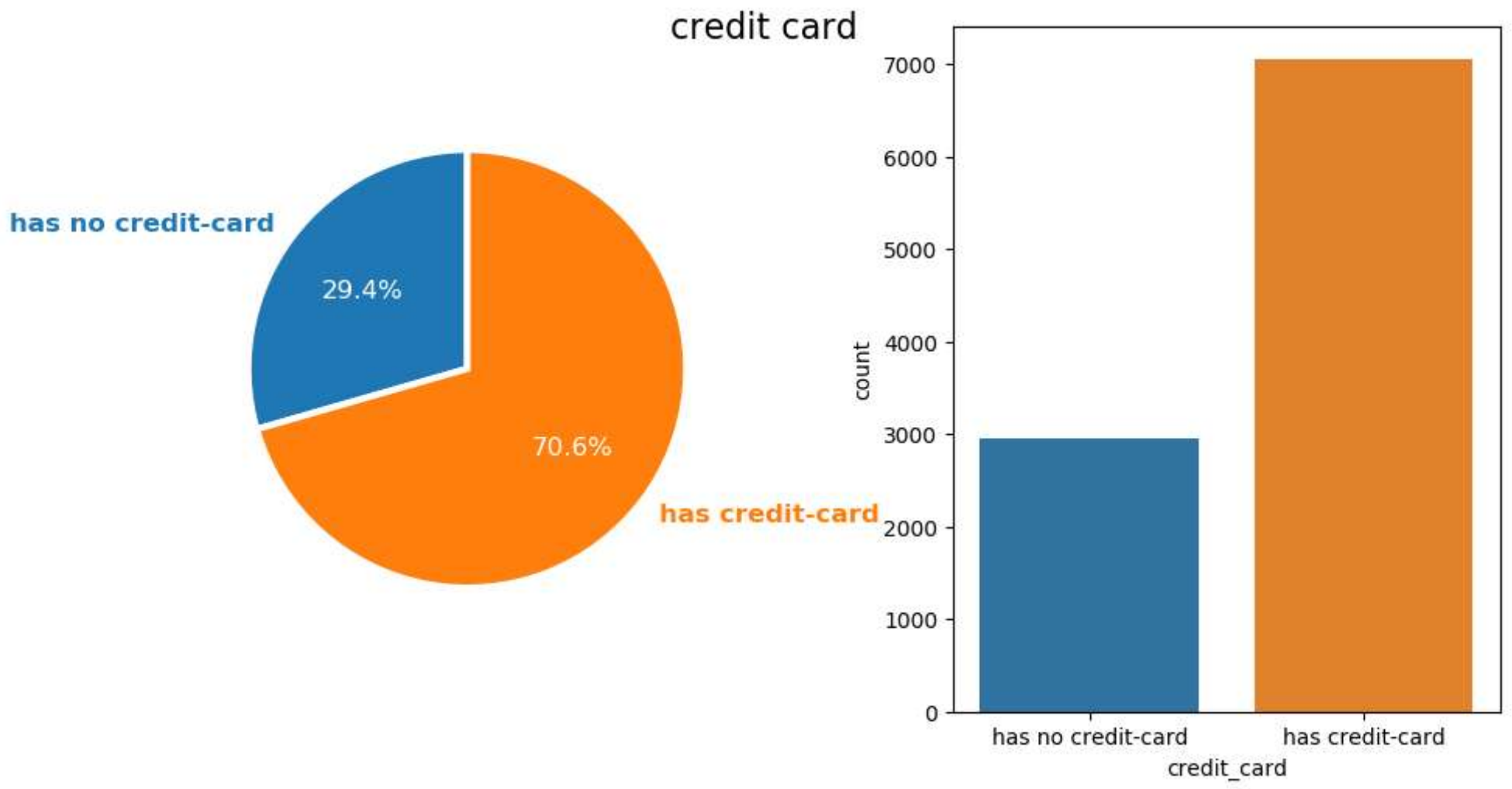
Out[12]:
```
0    7963
1    2037
Name: churn, dtype: int64
```

credit card

In [13]:
```
bank_chrun_df.credit_card.value_counts().sort_values()
```

Out[13]:
```
0    2945
1    7055
Name: credit_card, dtype: int64
```

In [14]:
```
fig,ax=plt.subplots(1,2,figsize=(10,5))

fig.suptitle("credit card", fontsize=16)
values,labels=bank_chrun_df.credit_card.value_counts().sort_values(),["has no credit-card","has credit-card"]
pie_chart(ax[0],values,labels,"")
count_plt=sns.countplot(data=bank_chrun_df,x="credit_card",ax=ax[1])
count_plt.set_xticklabels(labels)


fig.show()
```



continous columns relation with churn and distrbutions

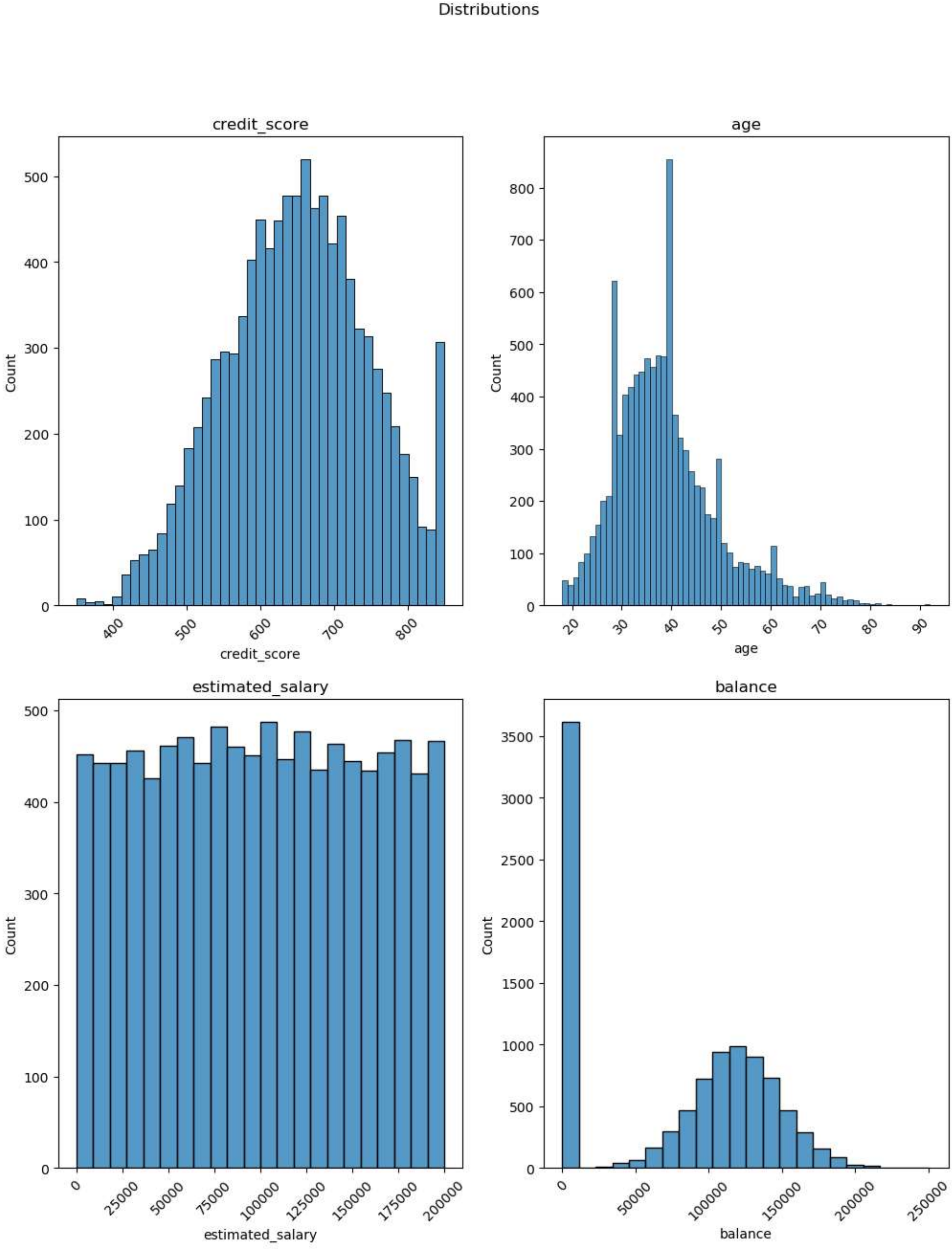
In [15]:
```
churn_mean=bank_chrun_df.groupby("churn",as_index=False).mean()
churn_mean
```

Out[15]:

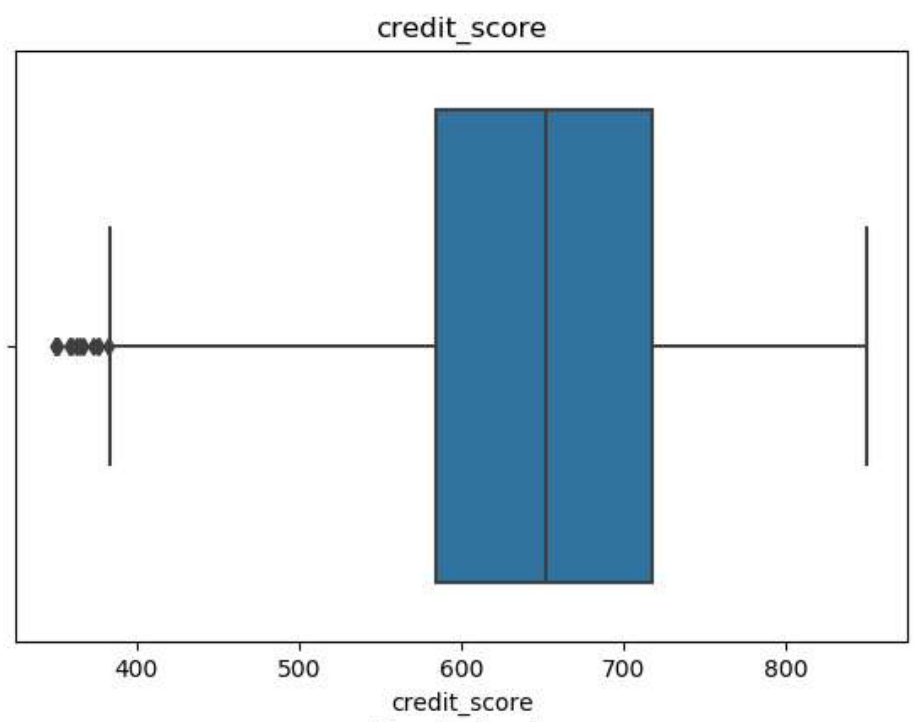
	churn	credit_score	age	tenure	balance	products_number	estimated_salary
0	0	651.853196	37.408389	5.033279	72745.296779	1.544267	99738.391772
1	1	645.351497	44.837997	4.932744	91108.539337	1.475209	101465.677531

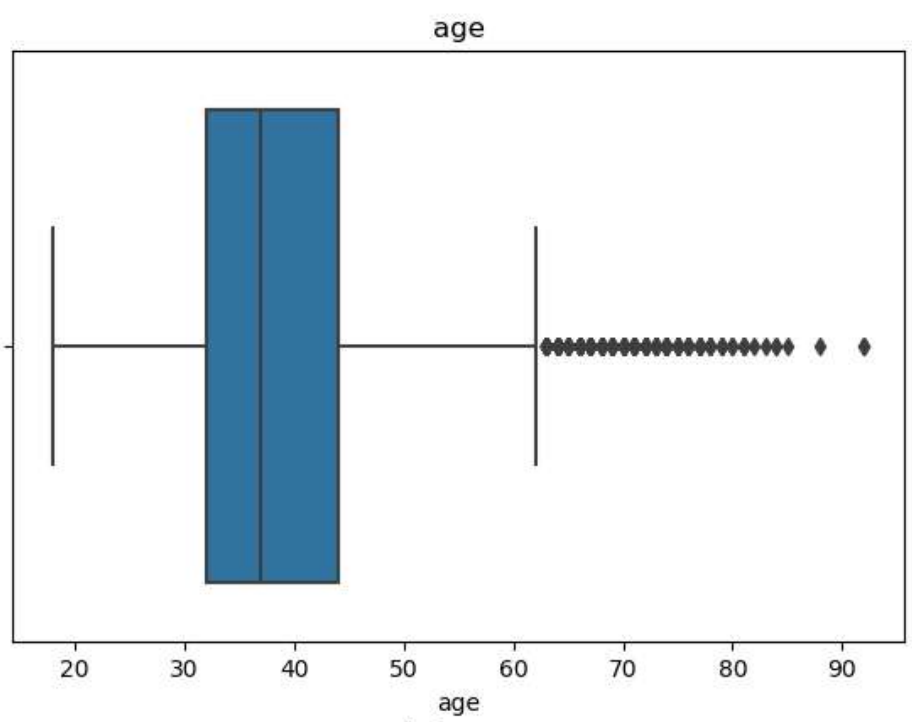
In [16]:

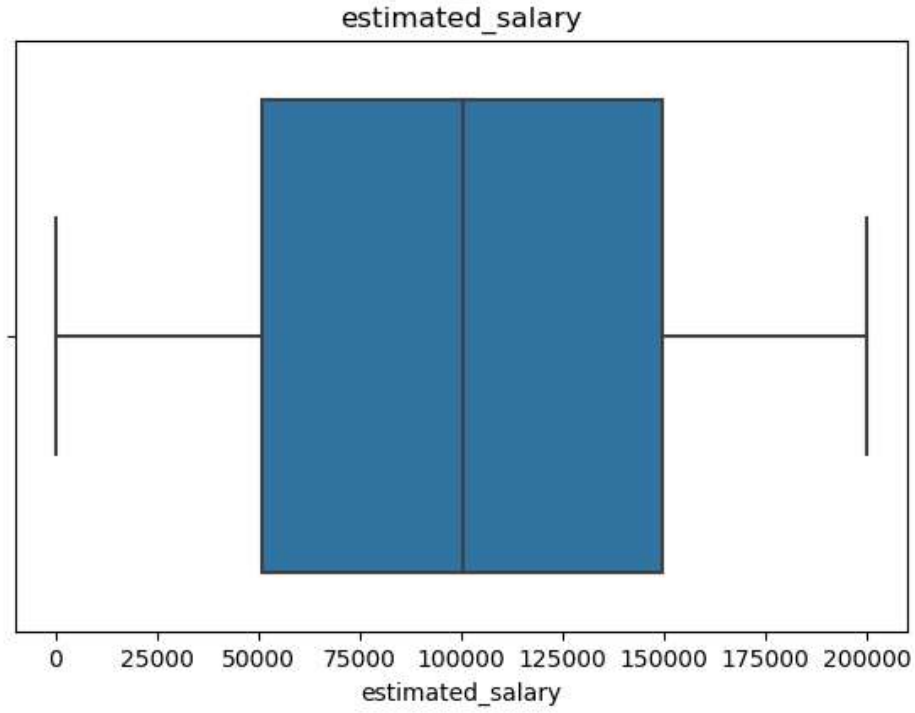
```
fig, axs = plt.subplots(2, 2, figsize=(12,14))

cols=[["credit_score","age"],["estimated_salary","balance"]]


for i in range(2):
    for j in range(2):
        sns.histplot(data=bank_chrun_df,x=cols[i][j],ax=axs[i,j])
        axs[i,j].set_title(cols[i][j])

fig.suptitle('Distributions')
# Rotate x-axis labels on all subplots
for ax in axs.flat:
    ax.tick_params(axis='x', labelrotation=45)

# Show plot
plt.show()
```

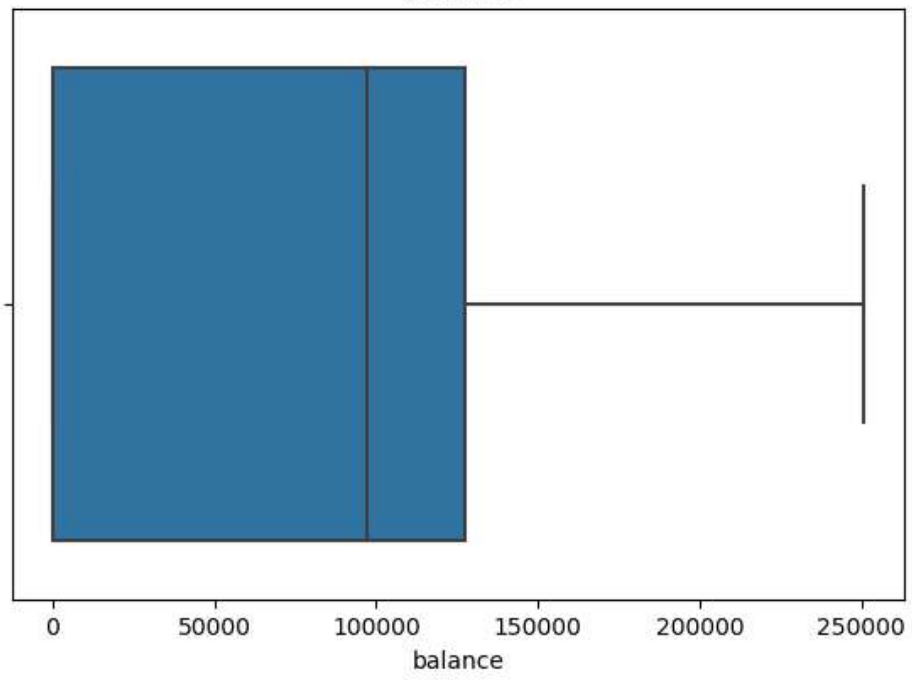


In [17]:
```
cols=[["credit_score","age"],["estimated_salary","balance"]]

fig, axs = plt.subplots(2, 2, figsize=(15,10))
for i in range(2):
    for j in range(2):
        sns.boxplot(data=bank_chrun_df,x=cols[i][j],ax=axs[i,j])
        axs[i,j].set_title(cols[i][j])
plt.show()
```







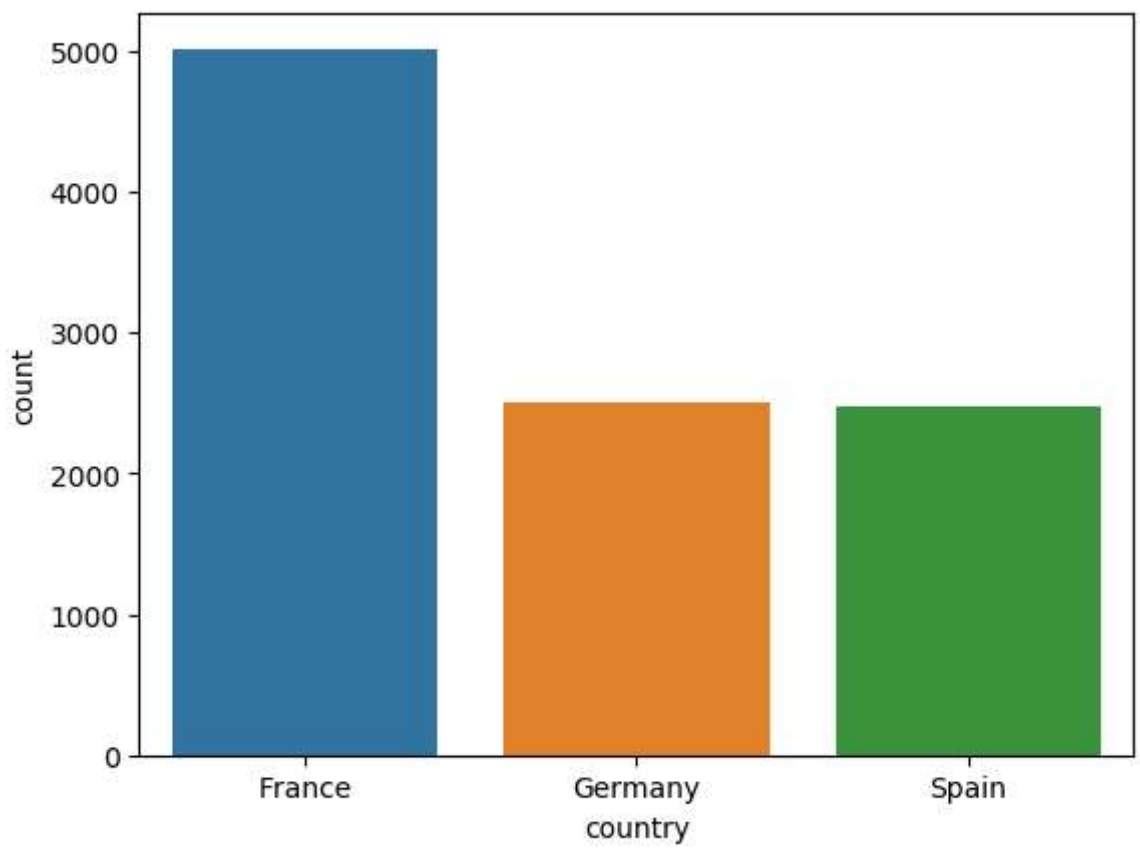


In [18]:
```
#  the number of customers by country
df_churn_by_country = bank_chrun_df.groupby("country").count()['customer_id']
df_churn_by_country
```

Out[18]:
```
country
France     5014
Germany    2509
Spain      2477
Name: customer_id, dtype: int64
```

In [19]:
```
sns.countplot(data=bank_chrun_df,x="country")
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f186060fe50>
```



train data preparation

In [20]:
```
train_data_df=bank_chrun_df.copy(deep=True)
train_data_df.drop('customer_id',inplace=True,axis=1)
```

In [21]:
```
train_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   credit_score      10000 non-null  int64
 1   country           10000 non-null  category
 2   gender            10000 non-null  category
 3   age               10000 non-null  int64
 4   tenure            10000 non-null  int64
 5   balance           10000 non-null  float64
 6   products_number   10000 non-null  int64
 7   credit_card       10000 non-null  category
 8   active_member     10000 non-null  category
 9   estimated_salary  10000 non-null  float64
 10  churn             10000 non-null  category
dtypes: category(5), float64(2), int64(4)
memory usage: 518.3 KB
```

hot-encoding

In [22]:
```
categorical_colums=['country','gender']
dummies=pd.get_dummies(train_data_df[categorical_colums])

train_data_df.drop(categorical_colums,axis=1,inplace=True)

train_data_df=pd.concat([train_data_df,dummies],axis=1)
```

In [23]:
```
train_data_df.head()
```

Out[23]:

	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn	country_France	country_Germany	country_Spain	ge
0	619	42	2	0.00	1	1	1	101348.88	1	1	0	0	
1	608	41	1	83807.86	1	0	1	112542.58	0	0	0	1	
2	502	42	8	159660.80	3	1	0	113931.57	1	1	0	0	
3	699	39	1	0.00	2	0	0	93826.63	0	1	0	0	
4	850	43	2	125510.82	1	1	1	79084.10	0	0	0	1	

Upload dataset to S3

In [24]:
```
#%load_ext autoreload
#%autoreload 2

from sklearn.model_selection import train_test_split

data=train_data_df.drop('churn',axis=1)
target=train_data_df['churn']
# use_smote = True will balance datasets using SMOTE
X_train, X_test, y_train, y_test,  = train_test_split(data,target, test_size=0.1)

X_train,X_val,y_train,y_val= train_test_split(X_train,y_train, test_size=0.18)

# Add the "churn" target variable
X_train["churn"] = y_train
X_test["churn"] = y_test
X_val["churn"] = y_val

# Save sets as a CSV files in the "datasets" folder
X_train.to_csv("datasets/train.csv", index=False)
X_test.to_csv("datasets/test.csv", index=False)
X_val.to_csv("datasets/validation.csv", index=False)
```

In [25]:
```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7380 entries, 6484 to 2315
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   credit_score      7380 non-null   int64
 1   age               7380 non-null   int64
 2   tenure            7380 non-null   int64
 3   balance           7380 non-null   float64
 4   products_number   7380 non-null   int64
 5   credit_card       7380 non-null   category
 6   active_member     7380 non-null   category
 7   estimated_salary  7380 non-null   float64
 8   country_France    7380 non-null   uint8
 9   country_Germany   7380 non-null   uint8
 10  country_Spain     7380 non-null   uint8
 11  gender_Female     7380 non-null   uint8
 12  gender_Male       7380 non-null   uint8
 13  churn             7380 non-null   category
dtypes: category(3), float64(2), int64(4), uint8(5)
memory usage: 461.6 KB
```

In [26]:
```
X_train.columns.tolist()
```

Out[26]:
```
['credit_score',
 'age',
 'tenure',
 'balance',
 'products_number',
 'credit_card',
 'active_member',
 'estimated_salary',
 'country_France',
 'country_Germany',
 'country_Spain',
 'gender_Female',
 'gender_Male',
 'churn']
```

In [27]:
```
columns_name = ['credit_score',
 'age',
 'tenure',
 'balance',
 'products_number',
 'credit_card',
 'active_member',
 'estimated_salary',
 'country_France',
 'country_Germany',
 'country_Spain',
 'gender_Female',
 'gender_Male',
 'churn']
```

In [28]:
```
# Create a SageMaker session to manage interactions with the SageMaker environment
sagemaker_session = sagemaker.Session()

# Get the  ARN of the execution role for the notebook instance
role = get_execution_role()

# Retrieve the default S3 bucket associated with this SageMaker session
bucket = sagemaker_session.default_bucket()

# Set a prefix for the S3 bucket directory to store the data and artifacts
prefix = "churn-prediction"

# Location of the full dataset on s3
dataset_path = sagemaker_session.upload_data(path="datasets/bank_customer_churn.csv", bucket=bucket, key_prefix=prefix + '/datasets')

# Location of the training data on s3
train_path = sagemaker_session.upload_data(path="datasets/train.csv", bucket=bucket, key_prefix=prefix + '/datasets')
# Location of the testing data on s3
test_path = sagemaker_session.upload_data(path="datasets/test.csv", bucket=bucket, key_prefix=prefix + '/datasets')
# Location of the validation data on s3
val_path = sagemaker_session.upload_data(path="datasets/validation.csv", bucket=bucket, key_prefix=prefix + '/datasets')



# import os
# os.environ["SM_CHANNEL_TRAINING"] =f"s3://{bucket}/{prefix}/datasets/"
# os.environ["SM_CHANNEL_TEST"] =f"s3://{bucket}/{prefix}/datasets/"
# os.environ['SM_MODEL_DIR']=f"s3://{bucket}"+"/models/"
# os.environ["SM_OUTPUT_DATA_DIR"]=f"s3://{bucket}/"+"/output/"

# os.environ["c_reg"]="0.1"
# os.environ["penalty"]="l1"
```

In [29]:
```
t = pd.read_csv("./datasets/validation.csv",header=None)
```

In [30]:
```
t.head()
```

Out[30]:

	0	1	2	3	4	5	6	7	8	9	10	11
0	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	country_France	country_Germany	country_Spain	gender_Female	ge
1	809	33	3	0.0	2	0	1	141426.78	0	0	1	1	
2	633	36	6	125130.28	1	0	0	125961.48	1	0	0	1	
3	667	44	5	140406.68	2	0	1	57164.19	0	1	0	1	
4	582	41	6	70349.48	2	0	1	178074.04	0	1	0	0	

Train models

In [59]:
```
dataset_path=f"s3://{bucket}/{prefix}/datasets/"
preprocessed_train=dataset_path
preprocessed_val=dataset_path
preprocessed_test=dataset_path
```

Fit Logistic Regression model

In [60]:
```
# Define an Estimator to run a logistic regression model
FRAMEWORK_VERSION ='0.23-1'
logistic_reg_estimator = SKLearn(
    # Specify the entry point script
```

```
    entry_point="logistic_regression.py",
    source_dir="src",
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge",
    framework_version=FRAMEWORK_VERSION,
    base_job_name="logistic-regression",
    py_version='py3',
    sagemaker_session=sagemaker_session
)
```

In [70]:
```
if constants.TUNING:
    # Define a range of hyperparameters for the model training
    hyperparameter_ranges = {
        'c_reg': ContinuousParameter(0.1, 1.0)
    }
```

In [ ]:
```
```

In [71]:
```
if constants.TUNING:
    # Define the objective (Minimize or Maximize)
    objective_type = "Maximize"

    # Define the metric definitions to retrieve from logs
    metric_definitions = [{ "Name": "Accuracy", "Regex": "Accuracy: ([0-9\\.]+)"}]

    # Define a HyperparameterTuner for the logistic regression estimator
    tuner = HyperparameterTuner(
        # Specify the estimator to use for tuning
        estimator=logistic_reg_estimator,
        # Specify the name of the objective metric to optimize for
        objective_metric_name='Accuracy',
        # Specify custom metrics to use for evaluation
        metric_definitions=metric_definitions,
        # Specify the ranges of hyperparameters
        hyperparameter_ranges=hyperparameter_ranges,
        # Specify the maximum number of training jobs to run
        max_jobs=10,
        # Specify the maximum number of training jobs to run in parallel
        max_parallel_jobs=2,
        # Specify a base job name for the tuning job
        base_tuning_job_name="logistic-regression-tunning",
        random_seed=42
    )
```

In [72]:
```
preprocessed_train
```

Out[72]:
```
's3://sagemaker-us-east-1-457600505329/churn-prediction/datasets/'
```

In [ ]:
```
if constants.TUNING:
    # Define the input channels for the training and testing data
    channels = {"train": preprocessed_train, "test": preprocessed_val}

    # Fit the tuner to the input data channels
    tuner.fit(inputs=channels, logs=False)
```

```
No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config
.............................................................!
```

In [79]:
```
if constants.TUNING:
    # Get the best training job
    best = tuner.best_training_job()
    print(best)

    # Get the best estimator
    best_estimator = tuner.best_estimator()
    print(best_estimator)

    # Get the hyperparameters of the best trained model
    hypers = best_estimator.hyperparameters()
    print(json.dumps(hypers, indent=4))



    # Define model hyperparameters
    hyperparameters = {"c_reg": hypers["c_reg"]}
    print(hyperparameters)
```

```
logistic-regression--230417-1322-001-6679c195

2023-04-17 13:24:56 Starting - Preparing the instances for training
2023-04-17 13:24:56 Downloading - Downloading input data
2023-04-17 13:24:56 Training - Training image download completed. Training in progress.
2023-04-17 13:24:56 Uploading - Uploading generated training model
2023-04-17 13:24:56 Completed - Resource reused by training job: logistic-regression--230417-1322-004-e5c5cc86
<sagemaker.sklearn.estimator.SKLearn object at 0x7f1856b1c9d0>
{
    "_tuning_objective_metric": "\"Accuracy\"",
    "c_reg": "0.509230946999235",
    "sagemaker_container_log_level": "20",
    "sagemaker_estimator_class_name": "\"SKLearn\"",
    "sagemaker_estimator_module": "\"sagemaker.sklearn.estimator\"",
    "sagemaker_job_name": "\"logistic-regression-2023-04-17-13-22-22-308\"",
    "sagemaker_program": "\"logistic_regression.py\"",
    "sagemaker_region": "\"us-east-1\"",
    "sagemaker_submit_directory": "\"s3://sagemaker-us-east-1-457600505329/logistic-regression-2023-04-17-13-22-22-308/source/sourcedir.tar.gz\""
}
{'c_reg': '0.509230946999235'}
```

In [80]:
```python
if constants.TUNING:
    # Define an Estimator
    logistic_reg_estimator = SKLearn(
        entry_point="logistic_regression.py",
        source_dir="src",
        role=role,
        instance_count=1,
        instance_type="ml.m5.xlarge",
        framework_version=FRAMEWORK_VERSION,
        base_job_name="logistic-regression",
        sagemaker_session=sagemaker_session,
        hyperparameters=hyperparameters)
```

In [81]:
```python
# Fit the estimator with preprocessed training and validation data
logistic_reg_estimator.fit({'train': preprocessed_train, 'test':  preprocessed_val}, logs=False)
print("S3 model path: {} ".format(logistic_reg_estimator.latest_training_job.job_name))
```

```
INFO:sagemaker:Creating training-job with name: logistic-regression-2023-04-17-13-34-56-517
2023-04-17 13:34:56 Starting - Starting the training job.....
2023-04-17 13:35:28 Starting - Preparing the instances for training........
2023-04-17 13:36:16 Downloading - Downloading input data.....
2023-04-17 13:36:46 Training - Downloading the training image.
2023-04-17 13:36:56 Training - Training image download completed. Training in progress.........
2023-04-17 13:37:38 Uploading - Uploading generated training model.
2023-04-17 13:37:50 Completed - Training job completed
S3 model path: logistic-regression-2023-04-17-13-34-56-517
```

Inference Logistic Regression

In [82]:
```python
# Define a timestamp prefix
timestamp_prefix = strftime("%Y-%m-%d-%H-%M-%S", gmtime())

logistic_reg_model = logistic_reg_estimator.create_model()

# Set the model and endpoint names
model_name = "churn-prediction-model-lr-" + timestamp_prefix
endpoint_name = "churn-prediction-endpoint-lr-" + timestamp_prefix

# Create the pipeline model
sm_model = PipelineModel(
    name=model_name,
    role=role,
    models=[logistic_reg_model]
)

# Deploy the model to an endpoint
sm_model.deploy(initial_instance_count=1, instance_type="ml.m5.xlarge", endpoint_name=endpoint_name)
```

```
INFO:sagemaker:Creating model with name: churn-prediction-model-lr-2023-04-17-13-37-53
INFO:sagemaker:Creating endpoint-config with name churn-prediction-endpoint-lr-2023-04-17-13-37-53
INFO:sagemaker:Creating endpoint with name churn-prediction-endpoint-lr-2023-04-17-13-37-53
----!
```

Prediction request - Logistic Regression

In [83]:
```python
# Create a client for accessing the SageMaker runtime API
runtime = boto3.client("sagemaker-runtime")

# Create a predictor object using the endpoint name
predictor = Predictor(
    endpoint_name=endpoint_name, sagemaker_session=sagemaker_session,
)

# Invoke the endpoint
response = runtime.invoke_endpoint(
    EndpointName=predictor.endpoint,
    Body=X_test.to_csv(header=True, index=False).encode("utf-8"),
    ContentType="text/csv",
)
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

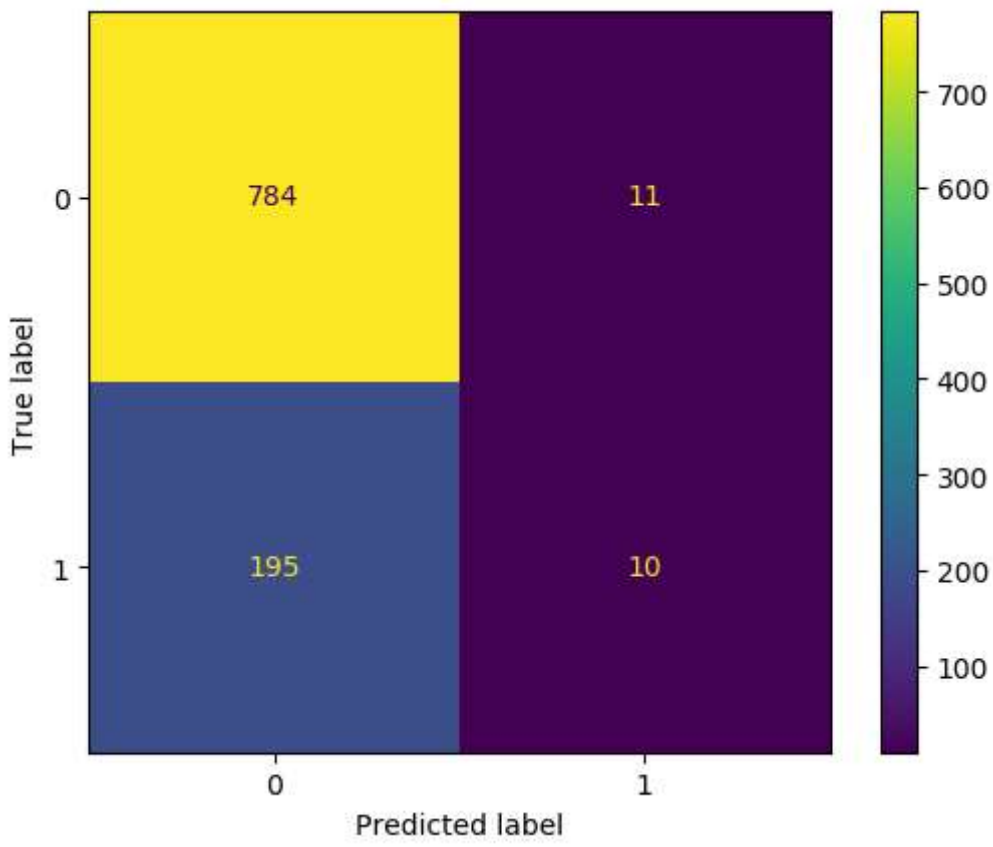
Accuracy score for Logistic Regression model

In [84]:
```
# Convert the test labels to a NumPy array
y_true = np.array(y_test)

# Load the predicted labels from the response body
y_pred = np.array(json.load(response["Body"]))

# Compute the accuracy of the model's predictions
accuracy = accuracy_score(y_true, y_pred)

print(accuracy)
```

```
0.794
```

Confusion Matrix - Logistic Regression

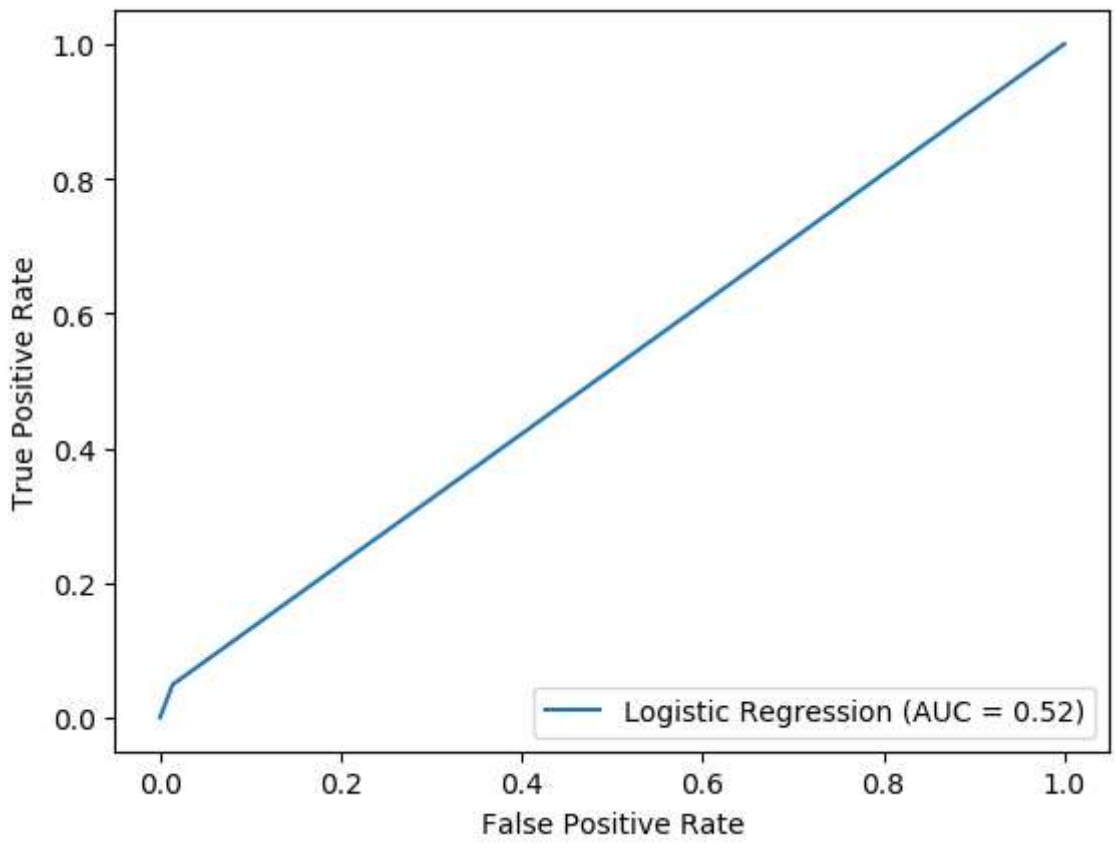
In [87]:
```
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.savefig(f"lr_cm_tuning_{str(constants.TUNING)}_smote_{str(constants.USE_SMOTE)}.png")
```



ROC curve for Logistic Regression model

In [88]:
```
# Compute the false positive rate, true positive rate, and decision thresholds for the logistic regression model's predictions
lr_fpr, lr_tpr, lr_thresholds = roc_curve(y_true, y_pred)

# Compute the area under the ROC curve for the logistic regression model
lr_roc_auc = auc(lr_fpr, lr_tpr)

# Plot the ROC curve for the logistic regression model
display = RocCurveDisplay(fpr=lr_fpr, tpr=lr_tpr, roc_auc=lr_roc_auc, estimator_name='Logistic Regression')
display.plot()
```

Out[88]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f18554ee710>
```



Delete Endpoint

In [53]:
```
sm_client = sagemaker_session.boto_session.client("sagemaker")

# Delete model and endpoint
if constants.DELETE_ENDPOINTS:
    sm_client.delete_endpoint(EndpointName=endpoint_name)
if constants.DELETE_MODELS:
    predictor.delete_model()
```

```
INFO:sagemaker:Deleting model with name: churn-prediction-model-lr-2023-04-07-14-39-04
```

Fit Decision Tree model with preprocessed data

In [94]:
```
# Define an Estimator to train a decision tree model
decision_tree_estimator = SKLearn(
    entry_point="decision_tree.py",
    source_dir="src",
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge",
    framework_version=FRAMEWORK_VERSION,
    base_job_name="decision-tree",
    sagemaker_session=sagemaker_session)
```

In [95]:
```
if constants.TUNING:
    # Define a range of hyperparameters for the model training
    hyperparameter_ranges = {
        'max_depth': IntegerParameter(1, 10),
        'min_samples_split': IntegerParameter(2, 10),
        'min_samples_leaf': IntegerParameter(1, 10)
    }
```

In [96]:
```
if constants.TUNING:
    # Define the objective (Minimize or Maximize)
    objective_type = "Maximize"

    # Define the metric definitions
    metric_definitions = [{ "Name": "Accuracy", "Regex": "Accuracy: ([0-9\\.]+)"}]
```

In [97]:
```
if constants.TUNING:
    # Define a HyperparameterTuner for the decision tree estimator
    tuner = HyperparameterTuner(decision_tree_estimator,
                                objective_metric_name='Accuracy',
                                metric_definitions=metric_definitions,
                                hyperparameter_ranges=hyperparameter_ranges,
                                # Number of training jobs to run
                                max_jobs=10,
                                # Number of training jobs to run in parallel
                                max_parallel_jobs=2,
                                base_tuning_job_name="decision-tree-tunning",
                                random_seed=42)
```

In [98]:
```
if constants.TUNING:
    # Define the input channels for the training and testing data
    channels = {"train": preprocessed_train, "test": preprocessed_val}

    # Fit the tuner to the input data channels
    tuner.fit(inputs=channels, logs=False)
```

```
WARNING:sagemaker.estimator:No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config
INFO:sagemaker:Creating hyperparameter tuning job with name: decision-tree-tunnin-230417-1409
.....................................................................!
```

In [99]:
```
if constants.TUNING:
    # Get the best training job
    best = tuner.best_training_job()
    print(best)

   # Get the best estimator
    best_estimator = tuner.best_estimator()
    print(best_estimator)

    # Get the hyperparameters of the best trained model
    hypers = best_estimator.hyperparameters()
    print(json.dumps(hypers, indent=4))

    # Define model hyperparameters
    hyperparameters = {
        "max_depth": hypers["max_depth"],
        "min_samples_split": hypers["min_samples_split"],
        "min_samples_leaf": hypers["min_samples_leaf"]
    }
    print(hyperparameters)
```

```
decision-tree-tunnin-230417-1409-008-1ff2dc5b

2023-04-17 14:14:31 Starting - Found matching resource for reuse
2023-04-17 14:14:31 Downloading - Downloading input data
2023-04-17 14:14:31 Training - Training image download completed. Training in progress.
2023-04-17 14:14:31 Uploading - Uploading generated training model
2023-04-17 14:14:31 Completed - Resource retained for reuse
<sagemaker.sklearn.estimator.SKLearn object at 0x7f18550e9e10>
{
    "_tuning_objective_metric": "\"Accuracy\"",
    "max_depth": "4",
    "min_samples_leaf": "5",
    "min_samples_split": "4",
    "sagemaker_container_log_level": "20",
    "sagemaker_estimator_class_name": "\"SKLearn\"",
    "sagemaker_estimator_module": "\"sagemaker.sklearn.estimator\"",
    "sagemaker_job_name": "\"decision-tree-2023-04-17-14-09-02-504\"",
    "sagemaker_program": "\"decision_tree.py\"",
    "sagemaker_region": "\"us-east-1\"",
    "sagemaker_submit_directory": "\"s3://sagemaker-us-east-1-457600505329/decision-tree-2023-04-17-14-09-02-504/source/sourcedir.tar.gz\""
}
{'max_depth': '4', 'min_samples_split': '4', 'min_samples_leaf': '5'}
```

In [102…

```
if constants.TUNING:
    # Define an Estimator with the best hyperparameters
    decision_tree_estimator = SKLearn(
        entry_point="decision_tree.py",
        source_dir="src",
        role=role,
        instance_count=1,
        dependencies=["requirements.txt"],
        instance_type="ml.m5.xlarge",
        framework_version=FRAMEWORK_VERSION,
        base_job_name="decision-tree",
        sagemaker_session=sagemaker_session,
       # Set hyperparameters for training job
        hyperparameters=hyperparameters)
```

In [103…

```
# Fit the estimator with preprocessed training and validation data
decision_tree_estimator.fit({'train': preprocessed_train, 'test':  preprocessed_val}, logs=False)
print("S3 model path: {} ".format(decision_tree_estimator.latest_training_job.job_name))
```

```
INFO:sagemaker:Creating training-job with name: decision-tree-2023-04-17-14-16-36-868
2023-04-17 14:16:37 Starting - Starting the training job...
2023-04-17 14:16:58 Starting - Preparing the instances for training........
2023-04-17 14:17:46 Downloading - Downloading input data.....
2023-04-17 14:18:16 Training - Downloading the training image.
2023-04-17 14:18:27 Training - Training image download completed. Training in progress...
2023-04-17 14:18:42 Uploading - Uploading generated training model.
2023-04-17 14:18:53 Completed - Training job completed
S3 model path: decision-tree-2023-04-17-14-16-36-868
```

Inference - Decision Tree

In [105…

```
timestamp_prefix = strftime("%Y-%m-%d-%H-%M-%S", gmtime())

decision_tree_model = decision_tree_estimator.create_model()

model_name = "churn-prediction-model-dt-" + timestamp_prefix
endpoint_name = "churn-prediction-endpoint-dt-" + timestamp_prefix

sm_model = PipelineModel(
    name = model_name,
    role=role, models=[decision_tree_model]
)

sm_model.deploy(initial_instance_count=1, instance_type="ml.m5.xlarge", endpoint_name=endpoint_name)
```

```
INFO:sagemaker:Creating model with name: churn-prediction-model-dt-2023-04-17-14-19-12
INFO:sagemaker:Creating endpoint-config with name churn-prediction-endpoint-dt-2023-04-17-14-19-12
INFO:sagemaker:Creating endpoint with name churn-prediction-endpoint-dt-2023-04-17-14-19-12
----!
```

Prediction - Decision Tree

In [106…

```
predictor = Predictor(
    endpoint_name=endpoint_name, sagemaker_session=sagemaker_session,
)

response = runtime.invoke_endpoint(
    EndpointName=predictor.endpoint,
    Body=X_test.to_csv(header=True, index=False).encode("utf-8"),
    ContentType="text/csv",
)
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

Accuracy score for Decision Tree model

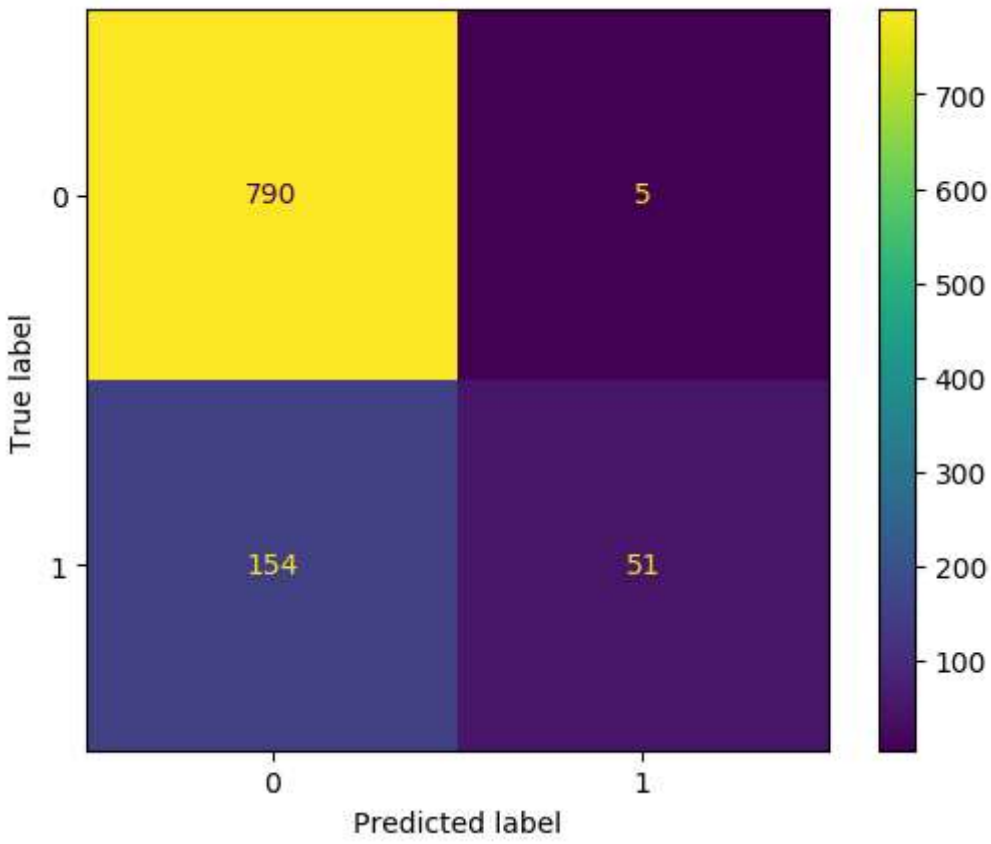
In [107…

```
y_true = np.array(y_test)
y_pred = np.array(json.load(response["Body"]))
print(accuracy_score(y_true, y_pred))
```

0.841

Confusion Matrix - Decision Tree

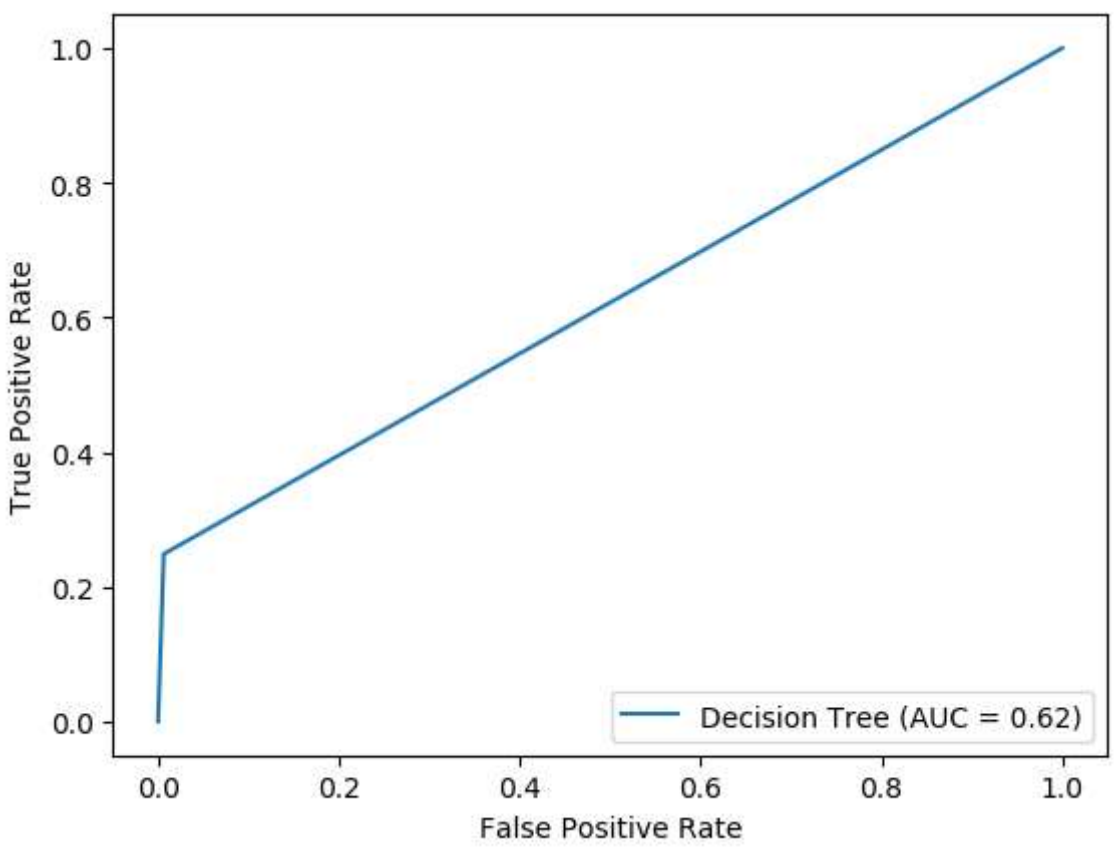
In [108…
```
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.savefig(f"dt_cm_tuning_{str(constants.TUNING)}_smote_{str(constants.USE_SMOTE)}.png")
```



ROC curve for Decision Tree model

In [109…
```
dt_fpr, dt_tpr, dt_thresholds = roc_curve(y_true, y_pred)
dt_roc_auc = auc(dt_fpr, dt_tpr)
display = RocCurveDisplay(fpr=dt_fpr, tpr=dt_tpr, roc_auc=dt_roc_auc, estimator_name='Decision Tree')

display.plot()
```

Out[109]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1854b03590>



Delete Endpoint

In [70]:
```
if constans.DELETE_ENDPOINTS:
    sm_client.delete_endpoint(EndpointName=endpoint_name)
if constans.DELETE_MODELS:
    predictor.delete_model()
```

```
INFO:sagemaker:Deleting model with name: churn-prediction-model-dt-2023-04-07-14-50-53
```

Fit RandomForest model

In [115…
```
# Define an Estimator to train a random forest model
random_forest_estimator = SKLearn(
    entry_point="random_forest.py",
    source_dir="src",
    role=role,
    instance_count=1,
    instance_type="ml.m5.xlarge",
```

```
    framework_version=FRAMEWORK_VERSION,
    base_job_name="random-forest",
    sagemaker_session=sagemaker_session)
```

In [116…

```
if constants.TUNING:
    # Define a range of hyperparameters for the model training
    hyperparameter_ranges = {
        'n_estimators': IntegerParameter(50, 500),
        'max_depth': IntegerParameter(5, 15),
        'min_samples_split': IntegerParameter(2, 10),
        'min_samples_leaf': IntegerParameter(1, 5),
        'criterion': CategoricalParameter(['gini', 'entropy'])
    }
```

In [117…

```
if constants.TUNING:
    # Define the objective (Minimize or Maximize)
    objective_type = "Maximize"

    # Define the metric definitions
    metric_definitions = [{ "Name": "Accuracy", "Regex": "Accuracy: ([0-9\\.]+)"}]
```

In [118…

```
if constants.TUNING:
    # Define a HyperparameterTuner for the random forest estimator
    tuner = HyperparameterTuner(random_forest_estimator,
                                objective_metric_name='Accuracy',
                                metric_definitions=metric_definitions,
                                hyperparameter_ranges=hyperparameter_ranges,
                                max_jobs=10,
                                max_parallel_jobs=2,
                                base_tuning_job_name="random-forest-tunning",
                                random_seed=42)
```

In [119…

```
if constants.TUNING:
    # Define the input channels for the training and testing data
    channels = {"train": preprocessed_train, "test": preprocessed_val}

    # Fit the tuner to the input data channels
    tuner.fit(inputs=channels, logs=False)
```

```
WARNING:sagemaker.estimator:No finished training job found associated with this estimator. Please make sure this estimator is only used for building workflow config
INFO:sagemaker:Creating hyperparameter tuning job with name: random-forest-tunnin-230417-1427
.....................................................................................!
```

In [120…

```
if constants.TUNING:
    # Get the best training job
    best = tuner.best_training_job()
    print(best)

    # Get the best estimator
    best_estimator = tuner.best_estimator()
    print(best_estimator)

    # Get the hyperparameters of the best trained model
    hypers = best_estimator.hyperparameters()
    print(json.dumps(hypers, indent=4))
```

```
random-forest-tunnin-230417-1427-008-edf87aae

2023-04-17 14:34:55 Starting - Preparing the instances for training
2023-04-17 14:34:55 Downloading - Downloading input data
2023-04-17 14:34:55 Training - Training image download completed. Training in progress.
2023-04-17 14:34:55 Uploading - Uploading generated training model
2023-04-17 14:34:55 Completed - Resource retained for reuse
<sagemaker.sklearn.estimator.SKLearn object at 0x7f1856874410>
{
    "_tuning_objective_metric": "\"Accuracy\"",
    "criterion": "\"entropy\"",
    "max_depth": "12",
    "min_samples_leaf": "2",
    "min_samples_split": "4",
    "n_estimators": "397",
    "sagemaker_container_log_level": "20",
    "sagemaker_estimator_class_name": "\"SKLearn\"",
    "sagemaker_estimator_module": "\"sagemaker.sklearn.estimator\"",
    "sagemaker_job_name": "\"random-forest-2023-04-17-14-27-32-200\"",
    "sagemaker_program": "\"random_forest.py\"",
    "sagemaker_region": "\"us-east-1\"",
    "sagemaker_submit_directory": "\"s3://sagemaker-us-east-1-457600505329/random-forest-2023-04-17-14-27-32-200/source/sourcedir.tar.gz\""
}
```

In [121…

```
if constants.TUNING:
    # Define model hyperparameters
    hyperparameters = {
        'n_estimators': hypers["n_estimators"],
        'max_depth': hypers["max_depth"],
        'min_samples_split': hypers["min_samples_split"],
        'min_samples_leaf': hypers["min_samples_leaf"],
        'criterion': hypers["criterion"].strip('"')}
    print(hyperparameters)
```

```
{'n_estimators': '397', 'max_depth': '12', 'min_samples_split': '4', 'min_samples_leaf': '2', 'criterion': 'entropy'}
```

In [122…

```
if constants.TUNING:
    # Define an Estimator with the best hyperparameters
    random_forest_estimator = SKLearn(
```

```
        entry_point="random_forest.py",
        source_dir="src",
        role=role,
        instance_count=1,
        instance_type="ml.m5.xlarge",
        framework_version=FRAMEWORK_VERSION,
        base_job_name="logistic-regression",
        sagemaker_session=sagemaker_session,
       # Set hyperparameters for training job
        hyperparameters=hyperparameters)
```

In [123…

```
# Fit the estimator with preprocessed training and validation data
random_forest_estimator.fit({'train': preprocessed_train, 'test':  preprocessed_val}, logs=False)
print("S3 model path: {} ".format(random_forest_estimator.latest_training_job.job_name))
```

```
INFO:sagemaker:Creating training-job with name: logistic-regression-2023-04-17-14-35-05-360
2023-04-17 14:35:07 Starting - Starting the training job..
2023-04-17 14:35:23 Starting - Preparing the instances for training........
2023-04-17 14:36:10 Downloading - Downloading input data.....
2023-04-17 14:36:38 Training - Downloading the training image.
2023-04-17 14:36:48 Training - Training image download completed. Training in progress........
2023-04-17 14:37:29 Uploading - Uploading generated training model.
2023-04-17 14:37:40 Completed - Training job completed
S3 model path: logistic-regression-2023-04-17-14-35-05-360
```

Inference pipeline - RandomForest

In [124…

```
timestamp_prefix = strftime("%Y-%m-%d-%H-%M-%S", gmtime())

random_forest_model = random_forest_estimator.create_model()

model_name = "churn-prediction-model-rf-" + timestamp_prefix
endpoint_name = "churn-prediction-endpoint-rf-" + timestamp_prefix

sm_model = PipelineModel(
    name = model_name,
    role=role, models=[random_forest_model]
)

sm_model.deploy(initial_instance_count=1, instance_type="ml.m5.xlarge", endpoint_name=endpoint_name)
```

```
INFO:sagemaker:Creating model with name: churn-prediction-model-rf-2023-04-17-14-37-42
INFO:sagemaker:Creating endpoint-config with name churn-prediction-endpoint-rf-2023-04-17-14-37-42
INFO:sagemaker:Creating endpoint with name churn-prediction-endpoint-rf-2023-04-17-14-37-42
----!
```

Prediction request - RandomForest

In [125…

```
predictor = Predictor(
    endpoint_name=endpoint_name, sagemaker_session=sagemaker_session,
)

response = runtime.invoke_endpoint(
    EndpointName=predictor.endpoint,
    Body=X_test.to_csv(header=True, index=False).encode("utf-8"),
    ContentType="text/csv",
)
```

```
WARNING:sagemaker.deprecations:The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

Accuracy score for RandomForest model

In [126…

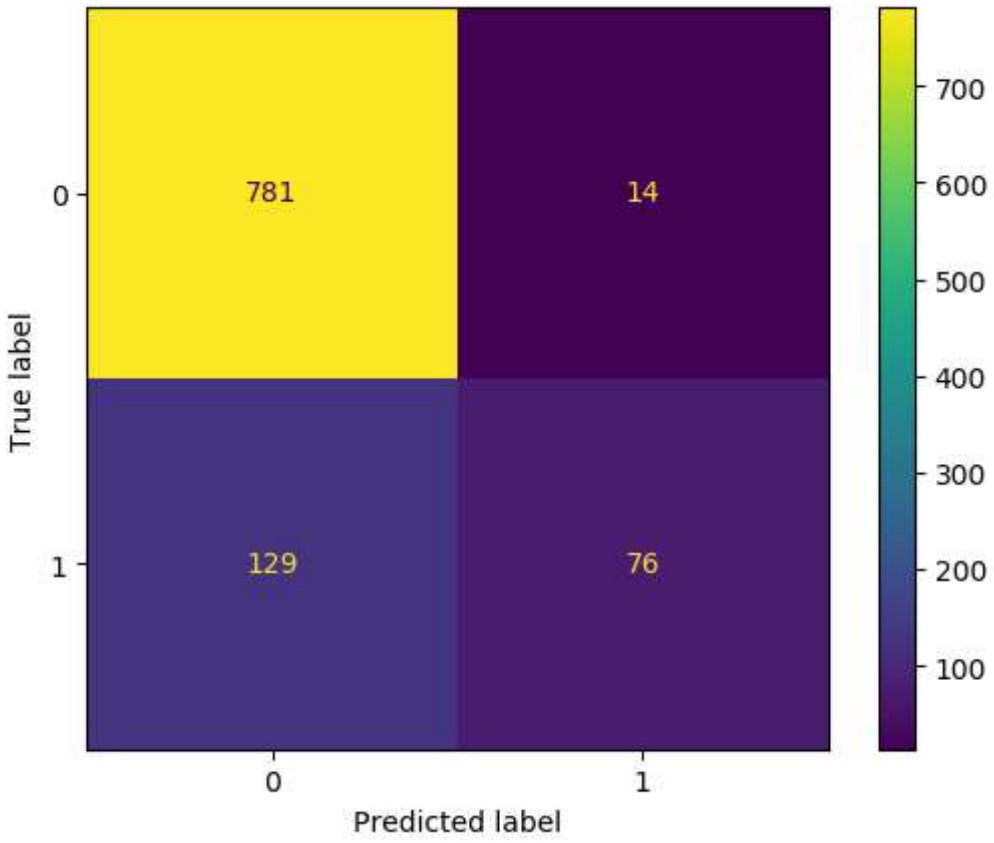
```
y_true = np.array(y_test)
y_pred = np.array(json.load(response["Body"]))
print(accuracy_score(y_true, y_pred))
```

```
0.857
```

Confusion Matrix - RandomForest

In [129…

```
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.savefig(f"rf_cm_tuning_{str(constants.TUNING)}_smote_{str(constants.USE_SMOTE)}.png")
```

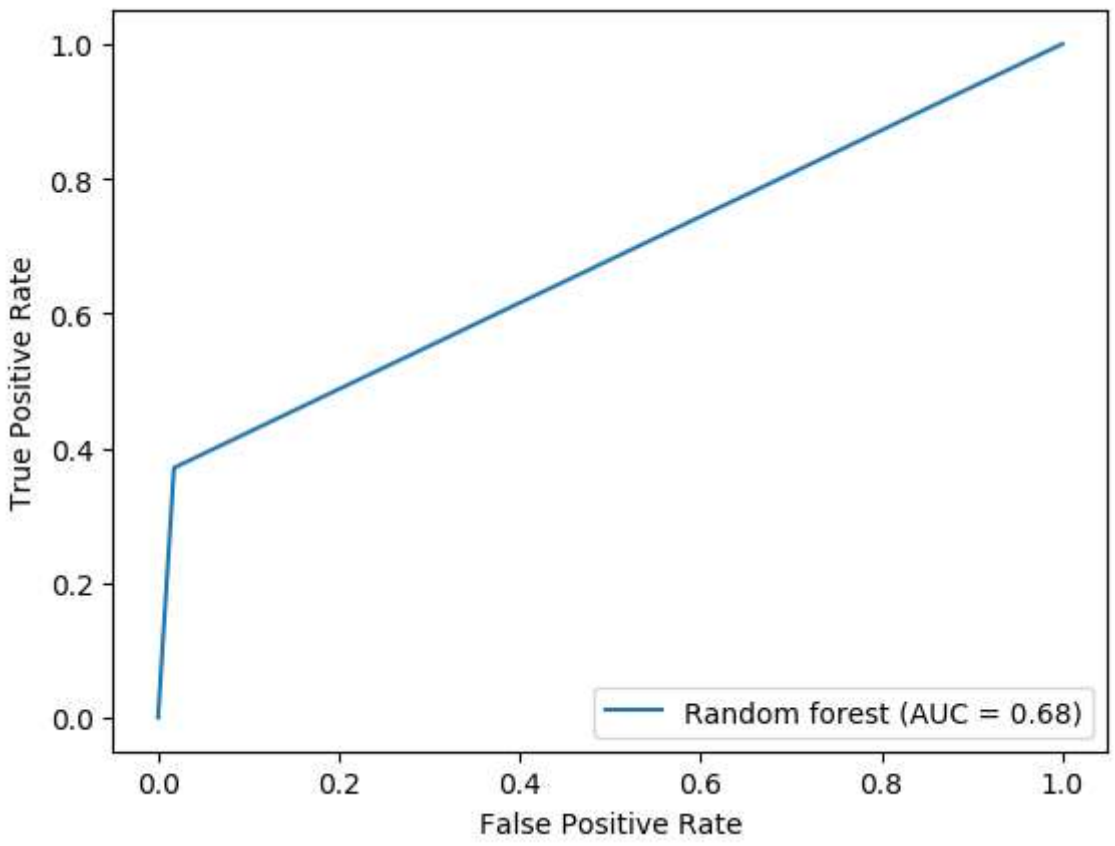


ROC curve for RandomForest model

In [130…

```
rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_true, y_pred)
rf_roc_auc = auc(rf_fpr, rf_tpr)
display = RocCurveDisplay(fpr=rf_fpr, tpr=rf_tpr, roc_auc=rf_roc_auc, estimator_name='Random forest')

display.plot()
```

Out[130]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1854e4a490>



Delete Endpoint

In [87]:

```
# if constants.DELETE_ENDPOINTS:
#     sm_client.delete_endpoint(EndpointName=endpoint_name)
# if constants.DELETE_MODELS:
#     predictor.delete_model()
```

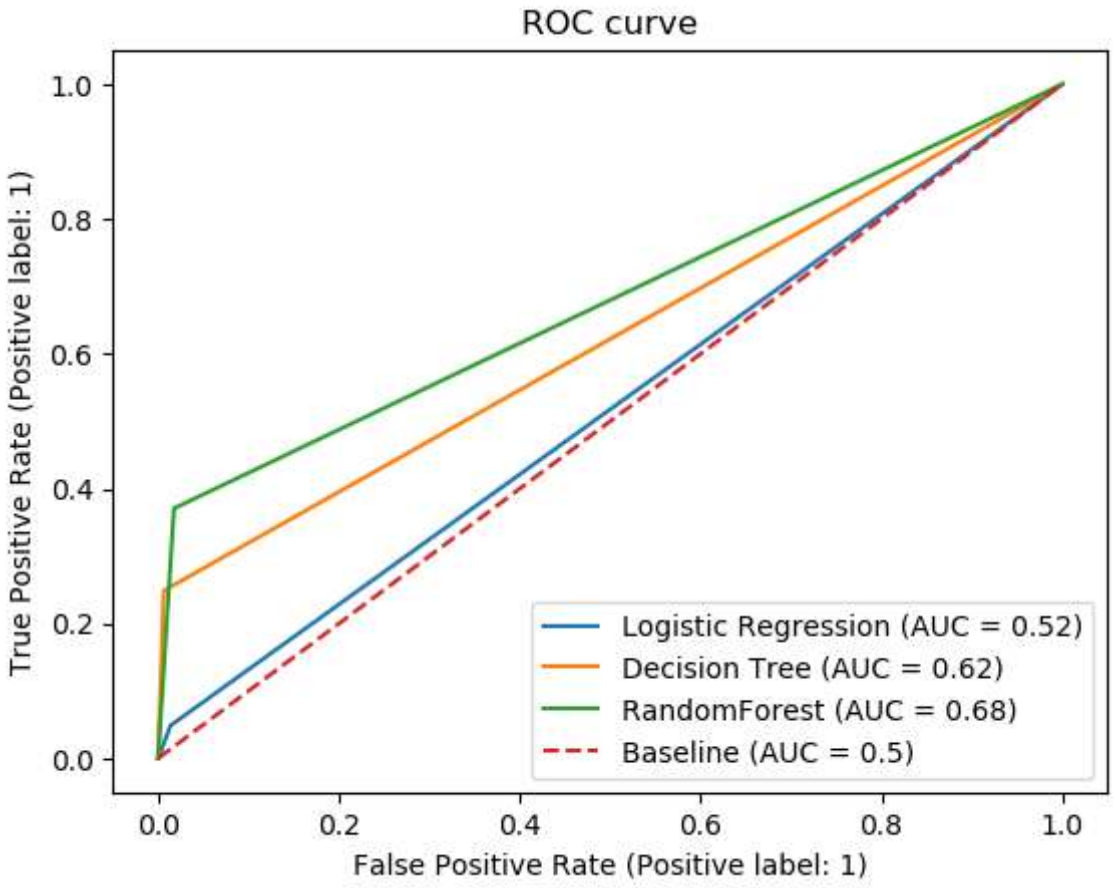
```
INFO:sagemaker:Deleting model with name: churn-prediction-model-rf-2023-04-07-15-03-34
```

Comparing ROC curves

In [132…

```
# Create arrays of false positive rates and true positive rates for a random classifier
fpr_random = np.linspace(0, 1.0, num=len(y_test))
tpr_random = np.linspace(0, 1.0, num=len(y_test))

# Plot the ROC curves for the logistic regression, decision tree, and random forest models, as well as the random classifier
plt.plot(lr_fpr, lr_tpr, label='Logistic Regression (AUC = %0.2f)' % lr_roc_auc)
plt.plot(dt_fpr, dt_tpr, label='Decision Tree (AUC = %0.2f)' % dt_roc_auc)
plt.plot(rf_fpr, rf_tpr, label='RandomForest (AUC = %0.2f)' % rf_roc_auc)
plt.plot(fpr_random, tpr_random, label = 'Baseline (AUC = 0.5)', linestyle='--')

# Add a legend and axis labels to the plot
plt.legend(loc='lower right')
plt.xlabel('False Positive Rate (Positive label: 1)')
plt.ylabel('True Positive Rate (Positive label: 1)')
plt.title('ROC curve')

# Save the plot as an image
```

```
plt.savefig('model_tuning_{}_smote_{}.png'.format(str(constants.TUNING), str(constants.USE_SMOTE)))

plt.show()
```



In [ ]: