In [1]:
```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/dock
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list a

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
# You can also write temporary files to /kaggle/temp/, but they won't be saved ou
```

In [3]:
```python
df=pd.read_csv("/kaggle/input/bank-customer-churn-prediction/Churn_Modelling.csv"
```

In [6]:
```python
df.shape
```

Out[6]: (10000, 14)

In [7]:
```python
df.sample(5)
```

Out[7]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
5613	5614	15689412	Christie	604	France	Female	32	7	127849.38
5278	5279	15799300	Kao	510	Germany	Male	31	0	113688.63
5864	5865	15803840	Forbes	729	France	Female	32	9	0.00
4532	4533	15739194	Manfrin	548	Spain	Male	38	0	178056.54
6615	6616	15792934	Carruthers	661	France	Male	26	8	0.00

In [8]:
```python
df.drop(columns=["RowNumber", "CustomerId", "Surname"], axis=1, inplace=True)
df.sample(5)
```

Out[8]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiv
3750	629	France	Male	39	2	129669.32	2	1	
9879	486	Germany	Male	62	9	118356.89	2	1	
1891	584	France	Female	37	1	0.00	2	1	
9732	724	Spain	Male	39	3	0.00	2	0	
1174	705	Spain	Female	40	5	203715.15	1	1	

In [9]:
```python
df.duplicated().sum()
```

Out[9]: 0

In [10]:
```
df.isna().sum()
```

Out[10]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

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

Out[11]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAc
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	1C
mean	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	
std	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	
min	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	
25%	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	
50%	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	
75%	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	
max	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	

In [12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CreditScore      10000 non-null  int64  
 1   Geography        10000 non-null  object 
 2   Gender           10000 non-null  object 
 3   Age              10000 non-null  int64  
 4   Tenure           10000 non-null  int64  
 5   Balance          10000 non-null  float64
 6   NumOfProducts    10000 non-null  int64  
 7   HasCrCard        10000 non-null  int64  
 8   IsActiveMember   10000 non-null  int64  
 9   EstimatedSalary  10000 non-null  float64
 10  Exited           10000 non-null  int64  
dtypes: float64(2), int64(7), object(2)
memory usage: 859.5+ KB
```

In [13]:
```
df["Exited"].value_counts()
```

Out[13]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

In [14]:
```python
df['Geography'].value_counts()
```

Out[14]:
```
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
```

In [15]:
```python
df.Gender.value_counts()
```

Out[15]:
```
Gender
Male      5457
Female    4543
Name: count, dtype: int64
```

In [16]:
```python
df.columns
```

Out[16]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [17]:
```python
df=pd.get_dummies(df, ["Geography","Gender" ], drop_first=True, dtype='int')
df.sample(5)
```

Out[17]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedS
3473	682	42	0	0.00	1	0	1	919
1377	768	44	6	60603.40	1	1	1	1780
7989	645	39	8	0.00	2	0	0	968
1488	596	30	6	121345.88	4	1	0	419
1021	485	32	6	102238.01	2	1	1	1940

In [18]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

In [19]:
```python
# def mask_outliers_iqr(df, columns): # Not required
#     df_out = df.copy()
#     for col in columns:
#         if not np.issubdtype(df_out[col].dtype, np.number):
#             continue  # Skip non-numeric columns

#         Q1 = df_out[col].quantile(0.25)
#         Q3 = df_out[col].quantile(0.75)
#         IQR = Q3 - Q1
#         lower_bound = Q1 - 1.5 * IQR
#         upper_bound = Q3 + 1.5 * IQR

#         # Mask outliers with NaN
#         df_out[col] = df_out[col].mask((df_out[col] < lower_bound) | (df_out[co

#     return df_out
```

In [80]:
```
X,y= df.drop(["Exited"], axis=1), df[["Exited"]]
```

In [81]:
```
X.head()
```

Out[81]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSala
0	619	42	2	0.00	1	1	1	101348.8
1	608	41	1	83807.86	1	0	1	112542.5
2	502	42	8	159660.80	3	1	0	113931.5
3	699	39	1	0.00	2	0	0	93826.6
4	850	43	2	125510.82	1	1	1	79084.1

In [82]:
```
col=['CreditScore',
 'Age',

 'NumOfProducts',
]
```

In [83]:
```
# X=mask_outliers_iqr(X, col)
```

In [84]:
```
import seaborn as sns, matplotlib.pyplot as plt
```

In [85]:
```
x_train, X_test, y_train, Y_test = train_test_split(X, y , stratify=y, test_size=
```

In [86]:
```
scaller=StandardScaler()
```

In [87]:
```
x_train_scaled= scaller.fit_transform(x_train)
```

In [88]:
```
X_test_scaled=scaller.transform(X_test)
```

In [90]:
```
import tensorflow
```

```
from tensorflow import keras
```

In [91]:
```
from tensorflow.keras import Sequential
```

In [92]:
```
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
```

In [93]:
```
df.shape[1]
```

Out[93]: 12

In [94]:
```
model = Sequential()
model.add(Dense(units=32,activation="relu", kernel_regularizer=l2(0.01), input_di
model.add(Dense(units=12,activation="relu"))
model.add(Dense(units=12,activation="relu", kernel_regularizer=l2(0.01)))
model.add(Dense(units=12,activation="relu", kernel_regularizer=l2(0.01)))
model.add(Dropout(0.2))
model.add(Dense(units=6,activation="relu",kernel_regularizer=l2(0.01)))
model.add(Dense(units=3,activation="relu", kernel_regularizer=l2(0.01)))
model.add(Dropout(0.2))
model.add(Dense(units=1,activation ='sigmoid'))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserW
arning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in t
he model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [95]:
```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Pa
dense_7 (Dense)	(None, 32)	
dense_8 (Dense)	(None, 12)	
dense_9 (Dense)	(None, 12)	
dense_10 (Dense)	(None, 12)	
dropout_2 (Dropout)	(None, 12)	
dense_11 (Dense)	(None, 6)	
dense_12 (Dense)	(None, 3)	
dropout_3 (Dropout)	(None, 3)	
dense_13 (Dense)	(None, 1)	

Total params: 1,195 (4.67 KB)

Trainable params: 1,195 (4.67 KB)

Non-trainable params: 0 (0.00 B)

In [96]:
```
print(np.unique(y_train))
y_train.info()
```

```
[0 1]
<class 'pandas.core.frame.DataFrame'>
Index: 8000 entries, 2515 to 2304
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Exited  8000 non-null   int64
dtypes: int64(1)
memory usage: 125.0 KB
```

In [97]:
```
from tensorflow.keras.metrics import Recall, Precision, Accuracy

model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=[Recall(), Precision(), Accuracy()]
)
```

In [99]:
```
from tensorflow.keras.callbacks import EarlyStopping

early_stop = EarlyStopping(
    monitor='val_recall',
    mode='max',              # Because higher recall is better for churn customer
    patience=10,              # Number of epochs with no improvement before stopp
    restore_best_weights=True,
    min_delta=0.0001
)
```

In [100]:
```
history=model.fit(x_train_scaled, y_train, epochs=100, validation_split=.2, \
                  callbacks=[early_stop],class_weight={0: 1, 1: 10})
```

```
Epoch 1/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 6s 9ms/step - accuracy: 0.0000e+00 - loss: 2.1716
- precision_1: 0.2024 - recall_1: 0.9935 - val_accuracy: 0.0000e+00 - val_los
s: 1.2140 - val_precision_1: 0.2050 - val_recall_1: 1.0000
Epoch 2/100
 57/200 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.0000e+00 - loss: 1.8722
- precision_1: 0.2013 - recall_1: 1.0000

/usr/local/lib/python3.11/dist-packages/keras/src/callbacks/early_stopping.py:
153: UserWarning: Early stopping conditioned on metric `val_recall` which is n
ot available. Available metrics are: accuracy,loss,precision_1,recall_1,val_ac
curacy,val_loss,val_precision_1,val_recall_1
  current = self.get_monitor_value(logs)

200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.0000e+00 - loss: 1.8375
- precision_1: 0.2051 - recall_1: 1.0000 - val_accuracy: 0.0000e+00 - val_los
s: 1.1105 - val_precision_1: 0.2050 - val_recall_1: 1.0000
Epoch 3/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.0000e+00 - loss: 1.6502
- precision_1: 0.2011 - recall_1: 1.0000 - val_accuracy: 0.0000e+00 - val_los
```

In [101]:
```python
w,b=model.layers[0].get_weights()
b
```

Out[101]:
```
array([-0.18660632,  0.04815373, -0.24157415,  0.00117472,  0.02556655,
        0.17147207, -0.09948713, -0.3484123 , -0.32381824,  0.08960334,
       -0.08077571, -0.03213134,  0.05704772, -0.01498128,  0.0861932 ,
       -0.5179489 , -0.1961962 ,  0.36160317, -0.0240429 , -0.05631261,
        0.09280947,  0.09838948,  0.00956321,  0.01863383, -0.0962879 ,
        0.00358349,  0.02820406,  0.28084746, -0.0657841 , -0.00603615,
        0.01156903,  0.07417452], dtype=float32)
```

In [102]:
```python
test_pred_proba=model.predict(X_test_scaled)
test_pred=np.where(test_pred_proba>=0.5, 1,0)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```

In [103]:
```python
from sklearn.metrics import accuracy_score
```

In [104]:
```python
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sco
```

In [105]:
```python
Y_test.value_counts()
```

Out[105]:
```
Exited
0    1593
1     407
Name: count, dtype: int64
```

In [106]:
```python
accuracy_score( Y_test, test_pred)
```

Out[106]:
```
0.7145
```

In [107]:
```python
train_pred_proba=model.predict(x_train_scaled)
train_pred=np.where(train_pred_proba>=0.5, 1,0)
```

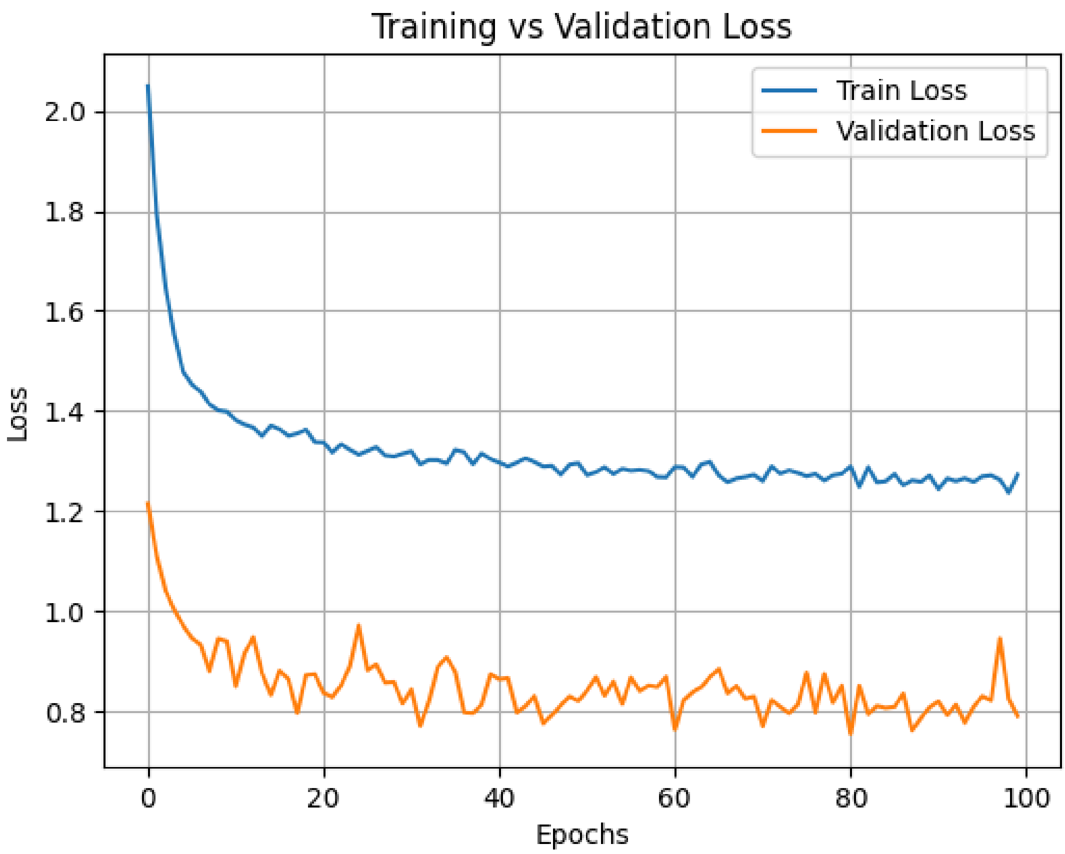
```
250/250 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

In [108]:
```python
print(classification_report(y_train, train_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.69      0.80      6370
           1       0.42      0.88      0.57      1630

    accuracy                           0.73      8000
   macro avg       0.69      0.79      0.69      8000
weighted avg       0.85      0.73      0.76      8000
```

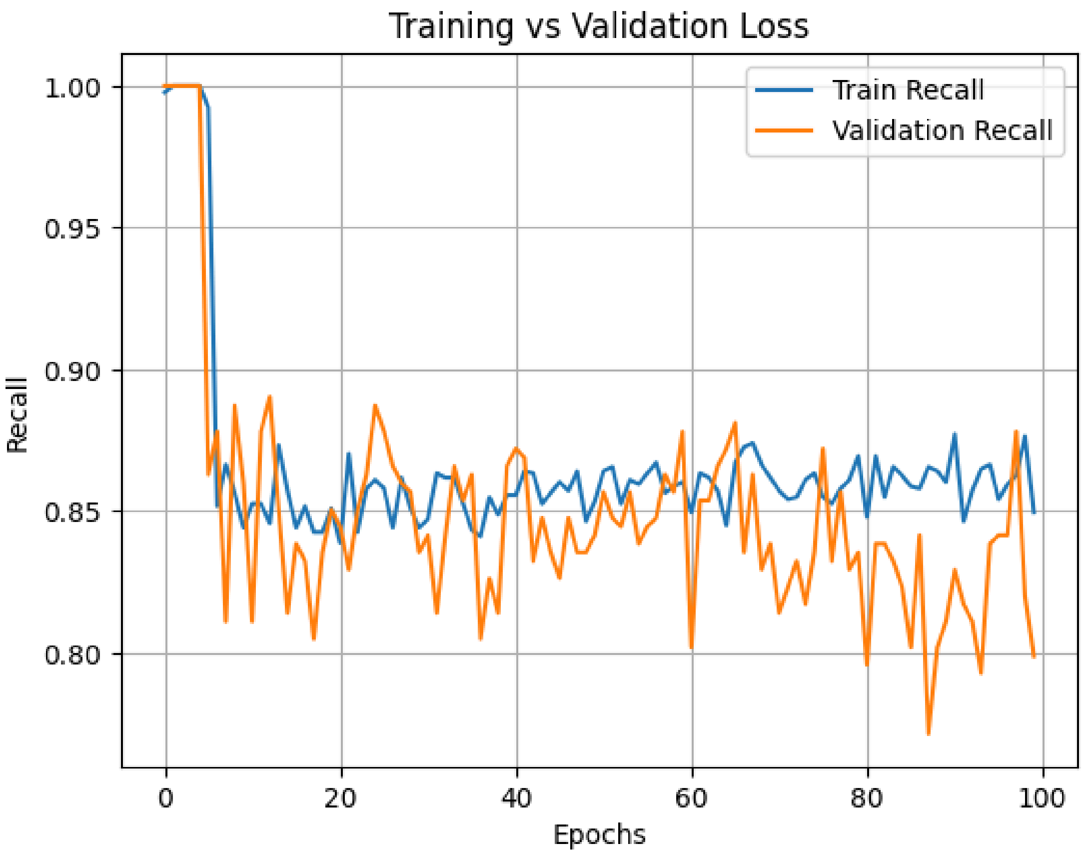
In [109]:
```python
print(classification_report(Y_test, test_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.68      0.79      1593
           1       0.40      0.83      0.54       407

    accuracy                           0.71      2000
   macro avg       0.67      0.76      0.67      2000
weighted avg       0.83      0.71      0.74      2000
```

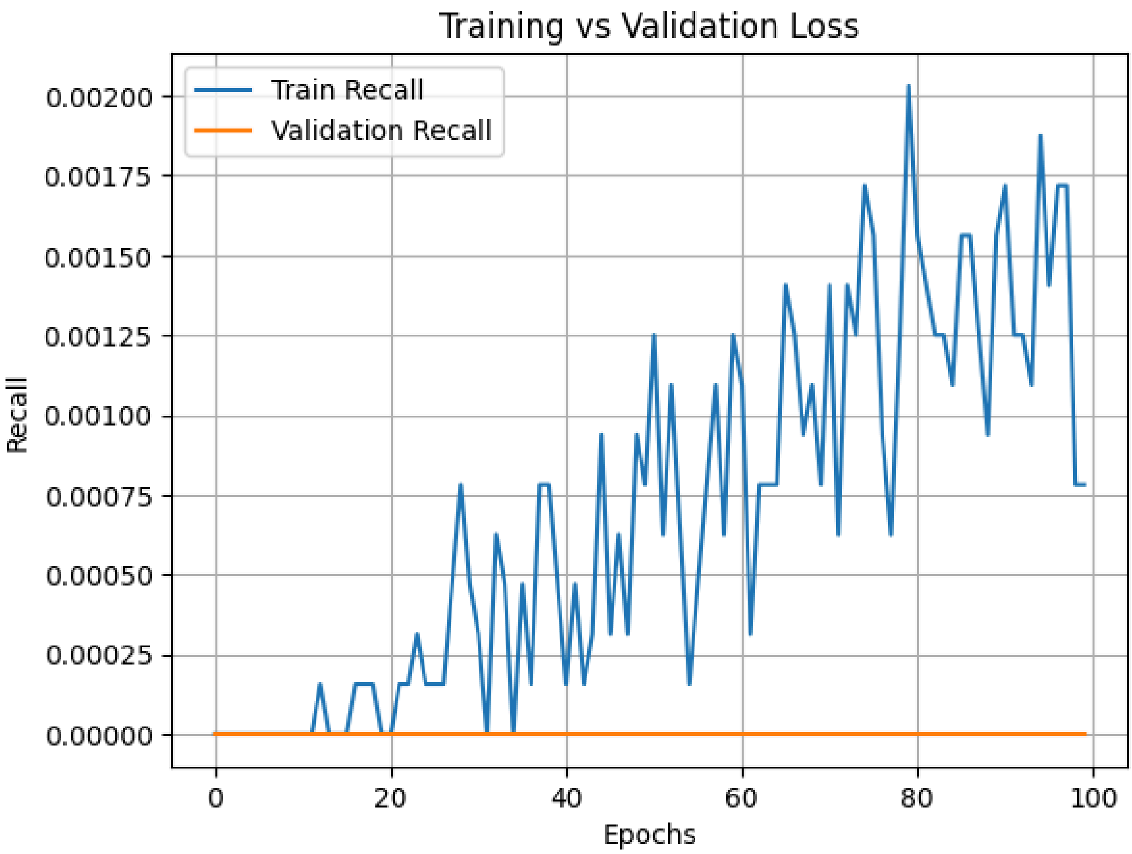
In [110]:
```
type(history.history)
```
Out[110]:
```
dict
```
In [111]:
```
import matplotlib.pyplot as plt
```
In [112]:
```
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True)
plt.show()
```


In [113]:
```
list(history.history.keys())
```
Out[113]:
```
['accuracy',
 'loss',
 'precision_1',
 'recall_1',
 'val_accuracy',
 'val_loss',
 'val_precision_1',
 'val_recall_1']
```

In [114]:
```
plt.plot(history.history["recall_1"], label="Train Recall")
plt.plot(history.history["val_recall_1"], label="Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True)
plt.show()
```



In [116]:
```
plt.plot(history.history["accuracy"], label="Train Recall")
plt.plot(history.history["val_accuracy"], label="Validation Recall")
plt.xlabel("Epochs")
plt.ylabel("Recall")
plt.title("Training vs Validation Loss")
plt.legend()
plt.grid(True)
plt.show()
```



Make the App

In [4]:
```
import ipywidgets as widgets
from IPython.display import display
```

In [5]:
```
# Create widgets for each feature
credit_score = widgets.IntSlider(description='CreditScore', min=300, max=900, val
age = widgets.IntSlider(description='Age', min=18, max=100, value=40)
tenure = widgets.IntSlider(description='Tenure', min=0, max=20, value=5)
balance = widgets.FloatText(description='Balance', value=50000.0)
products = widgets.Dropdown(description='Products', options=[1,2,3,4], value=1)
has_card = widgets.ToggleButtons(description='HasCrCard', options=[0,1], value=1)
is_active = widgets.ToggleButtons(description='IsActive', options=[0,1], value=1)
salary = widgets.FloatText(description='Salary', value=100000.0)

gender = widgets.Dropdown(description='Gender', options=['Male', 'Female'], value
geography = widgets.Dropdown(description='Geography', options=['Germany', 'Spain'

predict_btn = widgets.Button(description='Predict Churn', button_style='success')
output = widgets.Output()
```

In [6]:
```
def make_prediction(b):
    data = {
        'CreditScore': credit_score.value,
        'Age': age.value,
        'Tenure': tenure.value,
        'Balance': balance.value,
        'NumOfProducts': products.value,
        'HasCrCard': has_card.value,
        'IsActiveMember': is_active.value,
        'EstimatedSalary': salary.value,
        'Geography_Germany': 1 if geography.value == 'Germany' else 0,
        'Geography_Spain': 1 if geography.value == 'Spain' else 0,
        'Gender_Male': 1 if gender.value == 'Male' else 0
    }

    input_df = pd.DataFrame([data])
    input_scaled = scaller.transform(input_df)  # Use the same scaler
    proba = model.predict(input_scaled)[0][0]
    prediction = 1 if proba >= 0.5 else 0

    with output:
        output.clear_output()
        print(f"🔍 Predicted Probability: {proba:.2f}")
        print("🚨 Prediction:", "Will Churn (Exited = 1)" if prediction == 1 else
```

In [7]:
```
predict_btn.on_click(make_prediction)
```

In [9]:
```
form_items = widgets.VBox([
    credit_score, age, tenure, balance, products,
    has_card, is_active, salary, geography, gender,
    predict_btn, output
])
display(form_items)
```

CreditScore 600

Age 40

Tenure 5

Balance 50000

Products 1

HasCrCard

0 1

IsActive

0 1

Salary 100000

Geography France

Gender Male

Predict Churn

In [ ]: