Deep Learning Customer Churn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

```
df=pd.read_excel("Churn_Modelling.xlsx")
```

```
df.head()
```

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

```
df.info()
```

```
Frame'>
 9999
):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
```

Saving...

```
 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
```

```
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estim
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	1(
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	10(
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	5
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	5
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	10(
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	14
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	19


```
df.isnull().sum()
```

```
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
```

```
df.drop(columns=["RowNumber","CustomerId","Surname"],inplace=True)
```

```
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Target Column

Saving...

Exited

```
df['Exited'].describe()
```

```
count    10000.000000
mean         0.203700
std          0.402769
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: Exited, dtype: float64
```

```
df['Exited'].value_counts(normalize=True)
```

```
0    0.7963
1    0.2037
Name: Exited, dtype: float64
```

```
df['Exited'].value_counts().plot(kind='pie',autopct="%1.2f%%",labels=['Retained','Exited'],explode=[0,0.2],shadow=True)
plt.title("Proportion of customer churned and retained", size = 20)
plt.show()
```

Categorical Features

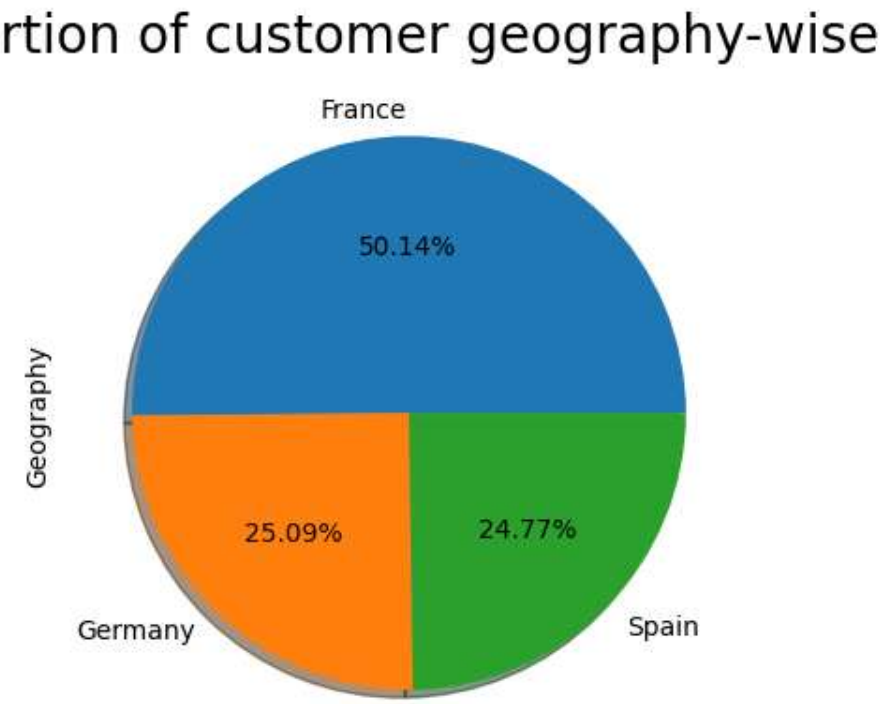
```
#function is created to explore Feature of datatype=Object

def feature_info(feature):
    print(df[feature].value_counts())
    print('\n')
    print(df.groupby(feature)['Exited'].value_counts(normalize=True)*100)

    ax=sns.countplot(data=df,x=feature,hue='Exited')
    plt.title(f"{feature} wise numbers of customer churned and retained",size = 20)
    plt.legend(labels=['0 : Reatained','1 : Churned'])
    for container in ax.containers:
        ax.bar_label(container)

    return feature
```

Geography

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

     France     5014
```

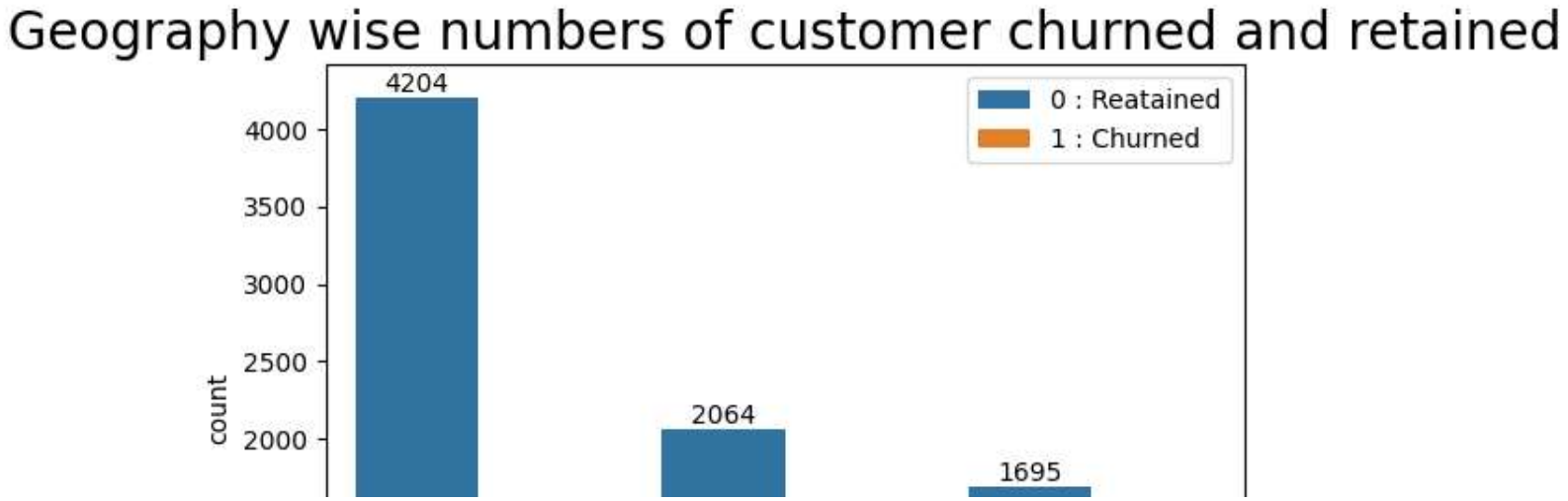
Saving...

```
df['Geography'].value_counts().plot(kind='pie',autopct="%1.2f%%",shadow=True)
plt.title("Portion of customer geography-wise", size = 20)
plt.show()
```




```
feature_info("Geography")
```

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


Geography  Exited
France     0         83.845233
           1         16.154767
Germany    0         67.556796
           1         32.443204
Spain      0         83.326605
           1         16.673395
Name: Exited, dtype: float64
'Geography'
```



Gender

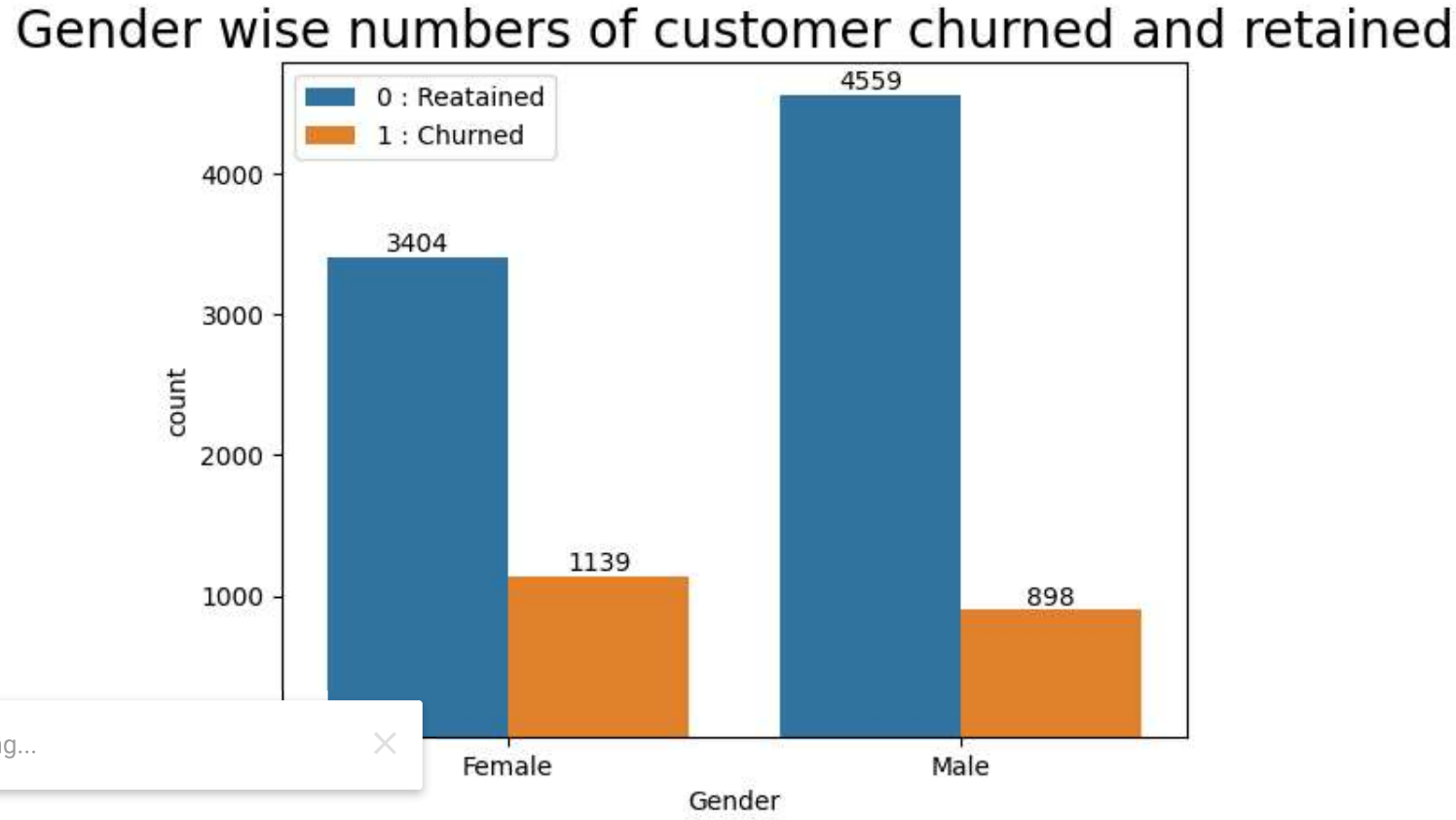
```
df['Gender'].describe()
```

```
top        Male
freq       5457
Name: Gender, dtype: object
```

```
feature_info('Gender')
```

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

```
Gender  Exited
Female  0         74.928461
        1         25.071539
Male    0         83.544072
        1         16.455928
Name: Exited, dtype: float64
'Gender'
```



Saving...

Number of **Product**

```
feature_info('NumOfProducts')
```

```
1    5084
2    4590
3     266
4      60
Name: NumOfProducts, dtype: int64
```

```
NumOfProducts  Exited
```

Has Credit Card or Not

0: No Credit Card

1: Has Credit Card

```
Name: Exited, dtype: float64
```

```
df['HasCrCard'].value_counts().plot(kind='pie',autopct="%1.2f%%",labels=['Has Credit Card','No Credit Card'],shadow=True)
plt.title("Proportion of customer have credit or Not", size = 20)
plt.legend()
plt.show()
```

Saving...


```
feature_info('HasCrCard')
```

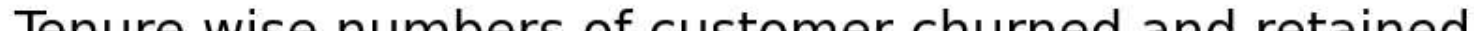
```
1    7055
```

Tenure

```
     HasCrCard  Exited
```

```
feature_info('Tenure')
```

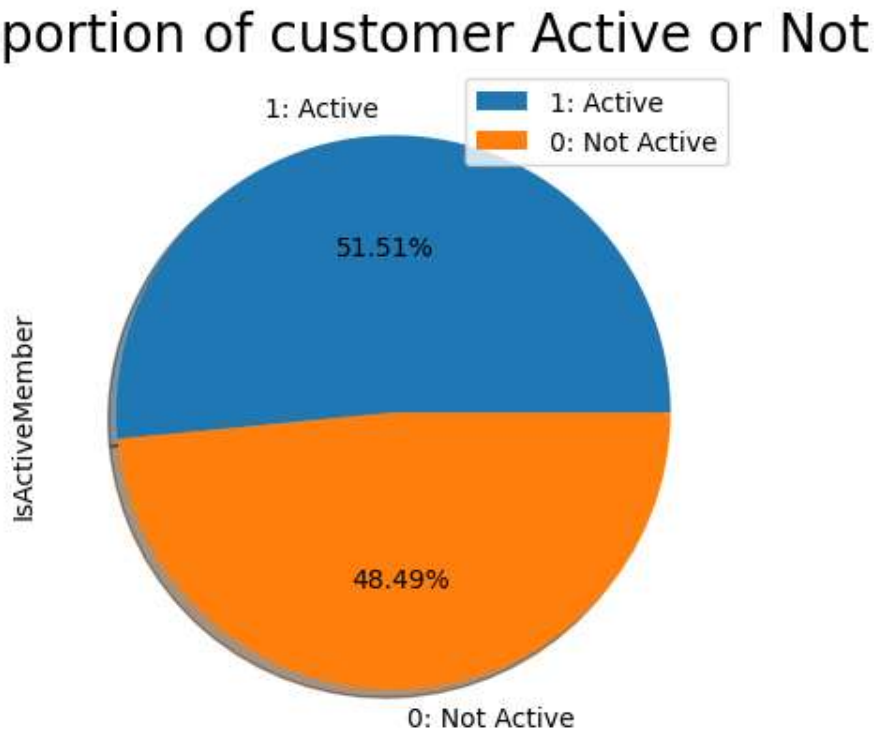
```
10     490
0      413
Name: Tenure, dtype: int64


Tenure  Exited
0       0         76.997579
        1         23.002421
1       0         77.584541
        1         22.415459
2       0         80.820611
        1         19.179389
3       0         78.889990
        1         21.110010
4       0         79.474216
        1         20.525784
5       0         79.347826
        1         20.652174
6       0         79.731127
        1         20.268873
7       0         82.782101
        1         17.217899
8       0         80.780488
        1         19.219512
9       0         78.353659
        1         21.646341
10      0         79.387755
        1         20.612245
Name: Exited, dtype: float64
'Tenure'
```



Customer Active Or Not

```
df['IsActiveMember'].value_counts().plot(kind='pie',autopct="%1.2f%%",labels=['1: Active','0: Not Active'],shadow=True)
                                         ctive or Not", size = 20)
plt.show()
```

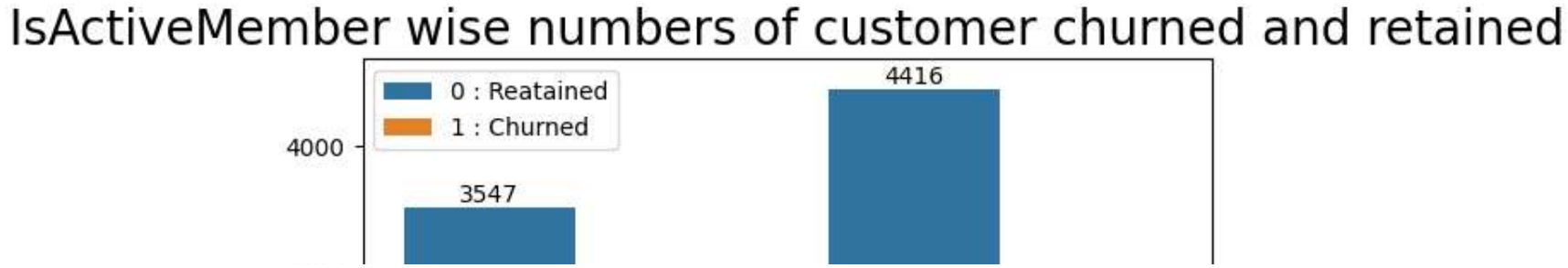
Saving...




```
feature_info('IsActiveMember')
```

Saving...

```
1    5151
0    4849
Name: IsActiveMember, dtype: int64


IsActiveMember  Exited
0               0         73.149103
                1         26.850897
1               0         85.730926
                1         14.269074
Name: Exited, dtype: float64
'IsActiveMember'
```



Numerical Feature

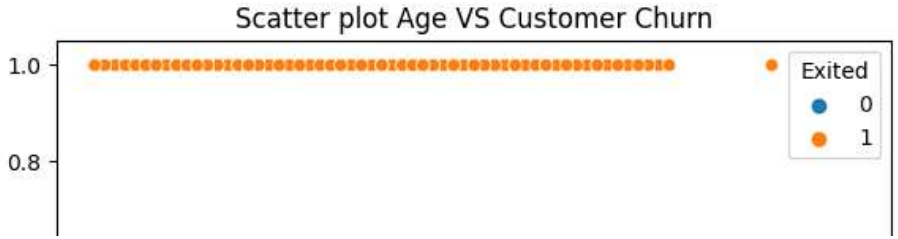
```
def Num_feature(feature):
  print(df[feature].describe())
  fig, axes = plt.subplots(2, 2, figsize=(15, 10))
  sns.scatterplot(x=df[feature], y=df['Exited'],hue=df['Exited'],ax=axes[0][0])
  axes[0][0].set_title((f'Scatter plot {feature} VS Customer Churn'))
  sns.boxplot(y=df[feature],x=df['Exited'],ax=axes[0][1])
  axes[0][1].set_title((f'Distribution of {feature} VS Customer Churn'))
  #axes[0][1].legend(labels=['0 : Reatained','1 : Churned'],loc='center')

  sns.distplot(x=df[feature],kde=True,ax=axes[1][0])
                                    ion of {feature}'))

  sns.boxplot(y=df[feature],ax=axes[1][1])
  axes[1][1].set_title((f'Distribution of {feature} using boxPlot'))
  fig.subplots_adjust(wspace=0.2,hspace=0.3,)
```

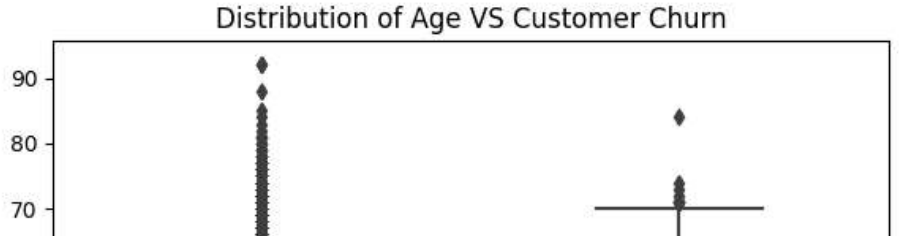
Saving...

Age

```
Num_feature('Age')
```

```
count    10000.000000
mean        38.921800
std         10.487806
min         18.000000
25%         32.000000
50%         37.000000
75%         44.000000
max         92.000000
Name: Age, dtype: float64
```

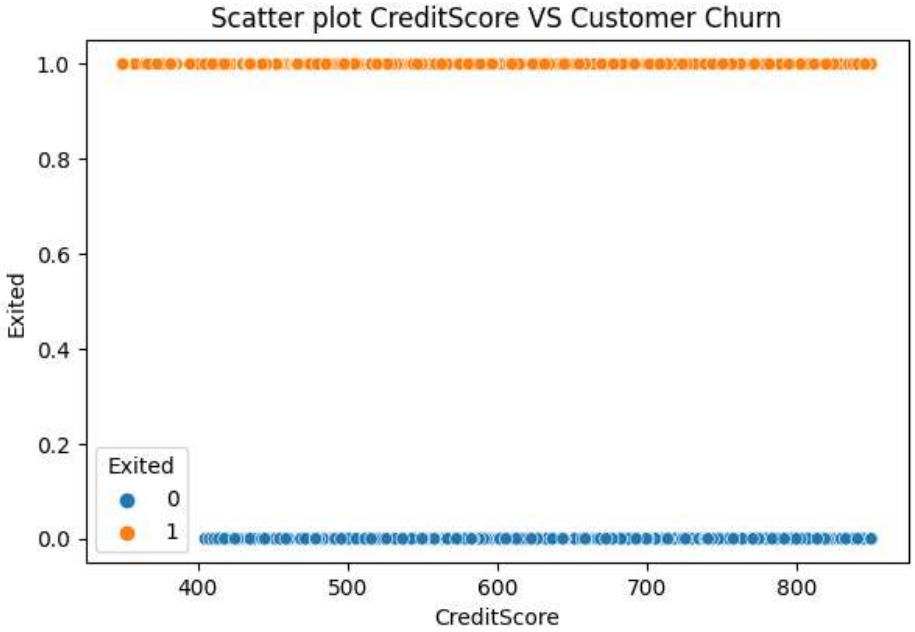


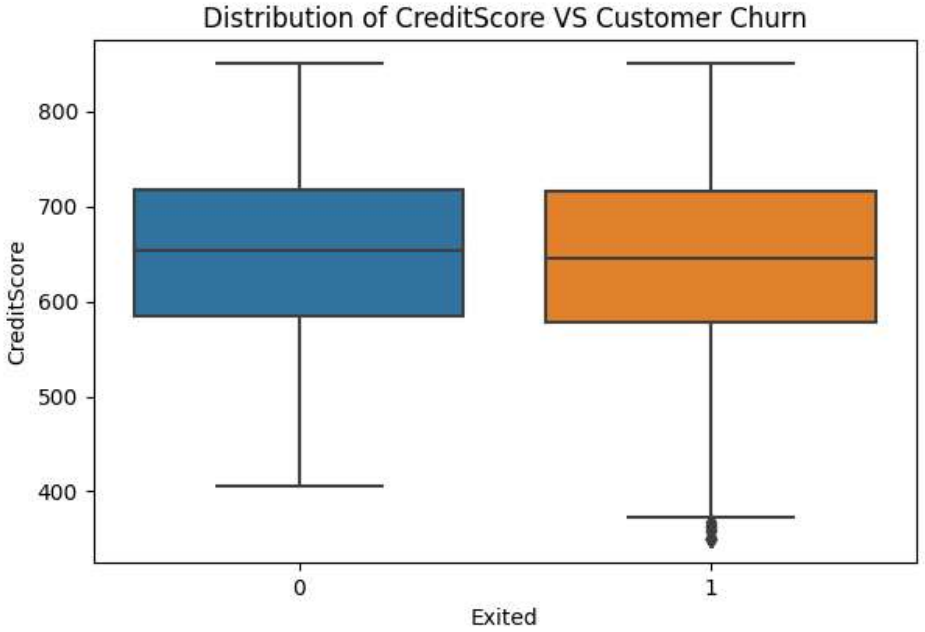


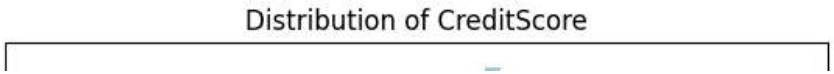
Credit Score

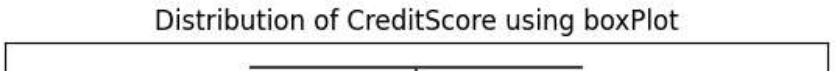
```
Num_feature('CreditScore')
```

```
count    10000.000000
mean       650.528800
std         96.653299
min        350.000000
25%        584.000000
50%        652.000000
75%        718.000000
max        850.000000
Name: CreditScore, dtype: float64
```





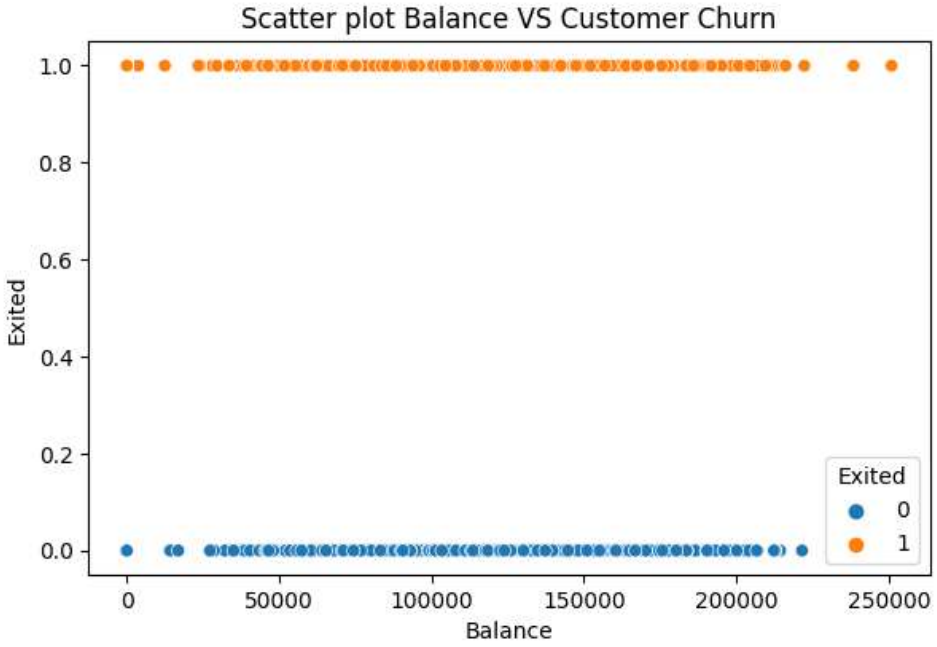


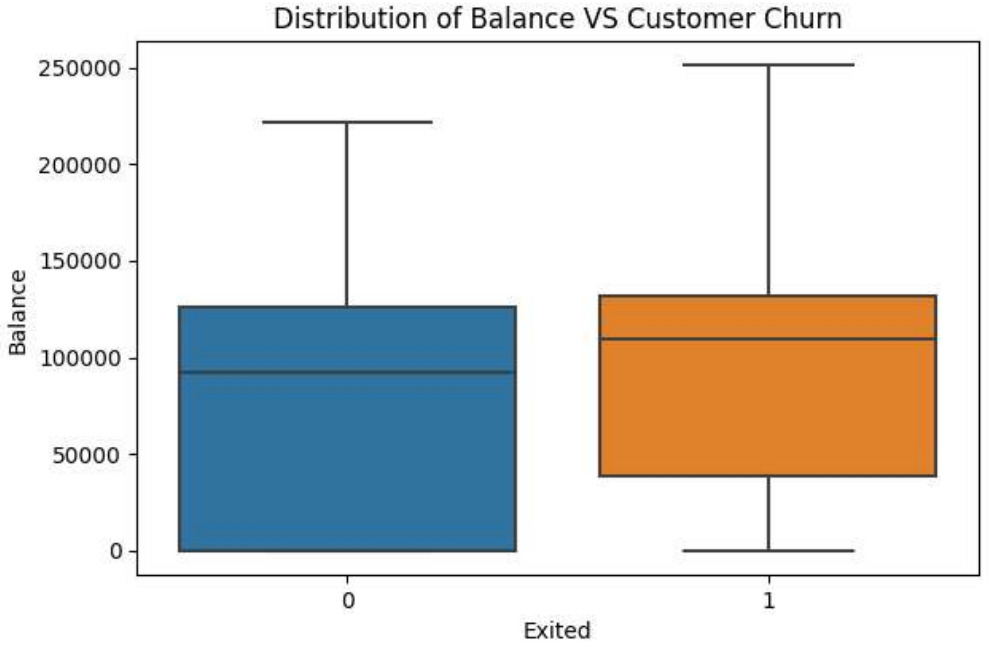


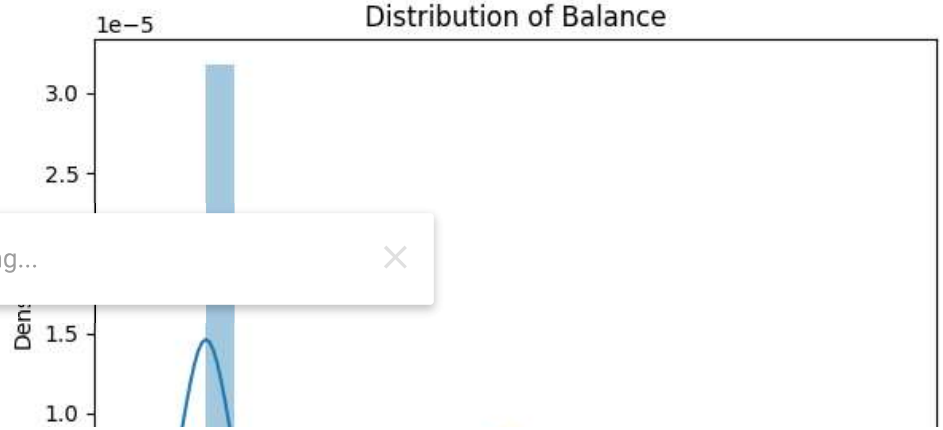
Balance

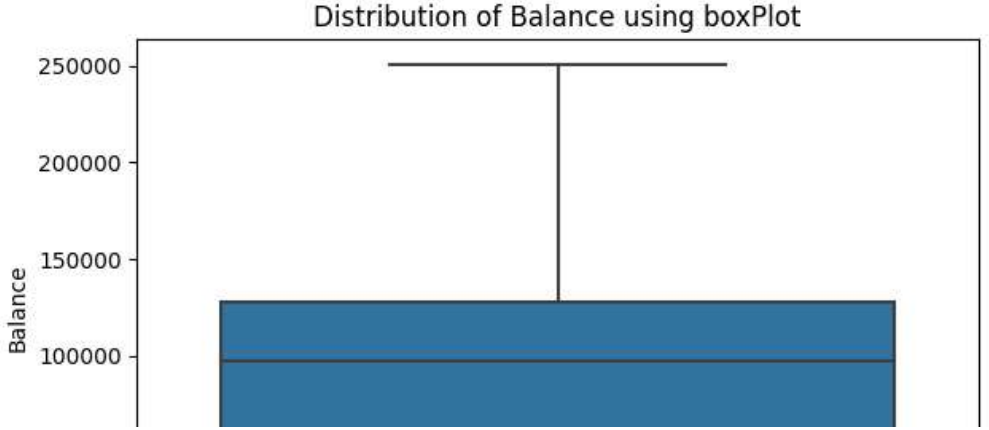
Saving...

```
count     10000.000000
mean      76485.889288
std       62397.405202
min           0.000000
25%           0.000000
50%       97198.540000
75%      127644.240000
max      250898.090000
Name: Balance, dtype: float64
```

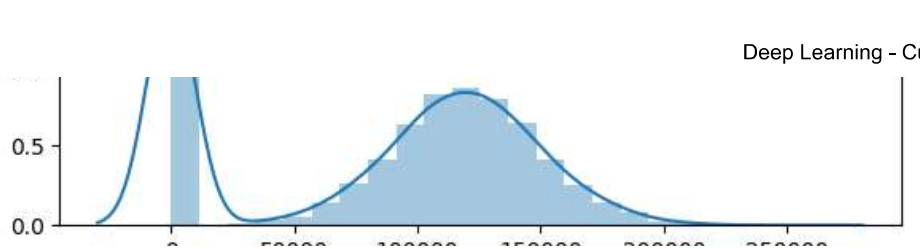


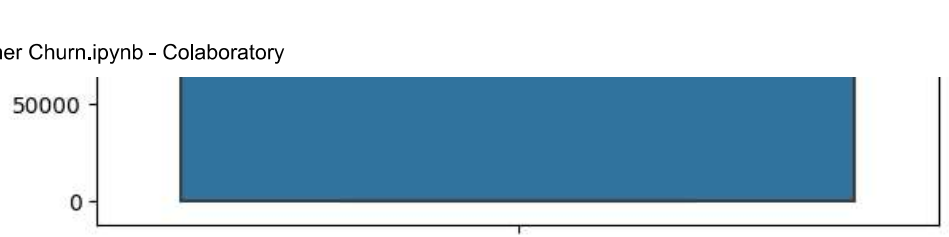






Saving...



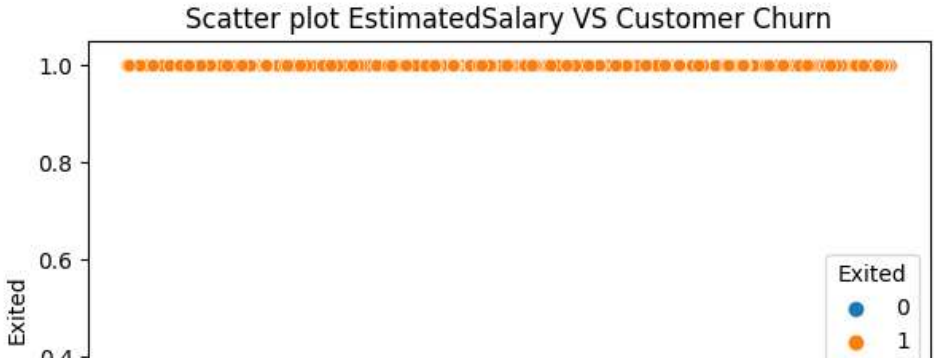


EstimatedSalary

```
Num_feature('EstimatedSalary')
```

Saving...

```
count     10000.000000
mean     100090.239881
std       57510.492818
min          11.580000
25%       51002.110000
50%      100193.915000
75%      149388.247500
max      199992.480000
Name: EstimatedSalary, dtype: float64
```

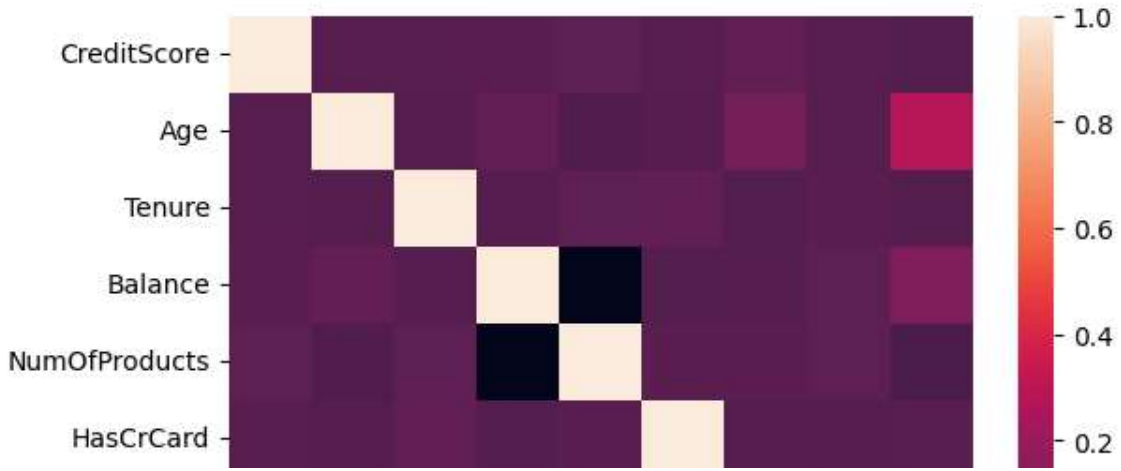


Correlation

```
sns.heatmap(df.corr())
```

Saving...

```
<Axes: >
```

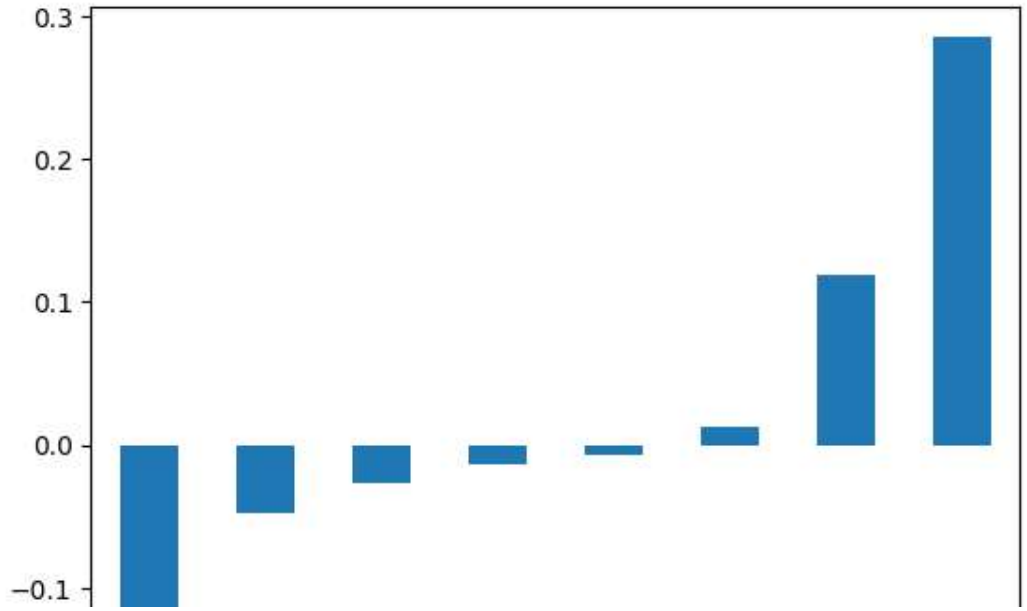



```
df.corr()['Exited'].sort_values()

IsActiveMember    -0.156128
NumOfProducts     -0.047820
CreditScore       -0.027094
Tenure            -0.014001
HasCrCard         -0.007138
EstimatedSalary    0.012097
Balance            0.118533
Age                0.285323
Exited             1.000000
Name: Exited, dtype: float64
```

```
df.corr()['Exited'][:-1].sort_values().plot(kind='bar')
```

Saving...

```
<Axes: >
```



Encoding

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df["Geography"]=le.fit_transform(df["Geography"])
le1=LabelEncoder()
df["Gender"]=le1.fit_transform(df["Gender"])
```

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

Saving...

```
from sklearn.preprocessing import StandardScaler

std=StandardScaler()


x=std.fit_transform(x)
x
```

```
array([[-0.32622142, -0.90188624, -1.09598752, ...,  0.64609167,
         0.97024255,  0.02188649],
       [-0.44003595,  1.51506738, -1.09598752, ..., -1.54776799,
         0.97024255,  0.21653375],
       [-1.53679418, -0.90188624, -1.09598752, ...,  0.64609167,
        -1.03067011,  0.2406869 ],
       ...,
       [ 0.60498839, -0.90188624, -1.09598752, ..., -1.54776799,
         0.97024255, -1.00864308],
       [ 1.25683526,  0.30659057,  0.91241915, ...,  0.64609167,
        -1.03067011, -0.12523071],
       [ 1.46377078, -0.90188624, -1.09598752, ...,  0.64609167,
        -1.03067011, -1.07636976]])
```

Train Test Split

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.30,random_state=42,stratify=y)
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
                              Dense
                            ication_report,confusion_matrix
```

Saving...

```
ann=Sequential()
ann.add(Dense(units=20,activation="relu"))
ann.add(Dense(units=20,activation="relu"))
ann.add(Dense(units=1,activation="sigmoid"))


ann.compile(optimizer="adam",loss="binary_crossentropy")


ann.fit(xtrain,ytrain,epochs=200,batch_size=32,validation_data=(xtest, ytest))
```

```
Epoch 1/200
219/219 [==============================] - 7s 4ms/step - loss: 0.6295 - val_loss: 0.4669
Epoch 2/200
219/219 [==============================] - 1s 4ms/step - loss: 0.4438 - val_loss: 0.4113
Epoch 3/200
219/219 [==============================] - 1s 5ms/step - loss: 0.4142 - val_loss: 0.3884
Epoch 4/200
219/219 [==============================] - 1s 6ms/step - loss: 0.3942 - val_loss: 0.3710
Epoch 5/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3800 - val_loss: 0.3644
Epoch 6/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3703 - val_loss: 0.3572
Epoch 7/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3637 - val_loss: 0.3517
Epoch 8/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3587 - val_loss: 0.3510
Epoch 9/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3558 - val_loss: 0.3457
Epoch 10/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3532 - val_loss: 0.3429
Epoch 11/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3507 - val_loss: 0.3418
Epoch 12/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3486 - val_loss: 0.3407
Epoch 13/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3472 - val_loss: 0.3401
Epoch 14/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3454 - val_loss: 0.3406
Epoch 15/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3444 - val_loss: 0.3377
Epoch 16/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3425 - val_loss: 0.3374
```

Saving...

```
Epoch 17/200
219/219 [==============================] - 1s 6ms/step - loss: 0.3417 - val_loss: 0.3370
Epoch 18/200
219/219 [==============================] - 1s 5ms/step - loss: 0.3405 - val_loss: 0.3340
Epoch 19/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3395 - val_loss: 0.3382
Epoch 20/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3388 - val_loss: 0.3381
Epoch 21/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3371 - val_loss: 0.3359
Epoch 22/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3368 - val_loss: 0.3345
Epoch 23/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3361 - val_loss: 0.3381
Epoch 24/200
219/219 [==============================] - 1s 5ms/step - loss: 0.3345 - val_loss: 0.3340
Epoch 25/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3339 - val_loss: 0.3328
Epoch 26/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3329 - val_loss: 0.3359
Epoch 27/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3322 - val_loss: 0.3342
Epoch 28/200
219/219 [==============================] - 1s 4ms/step - loss: 0.3312 - val_loss: 0.3323
Epoch 29/200
```

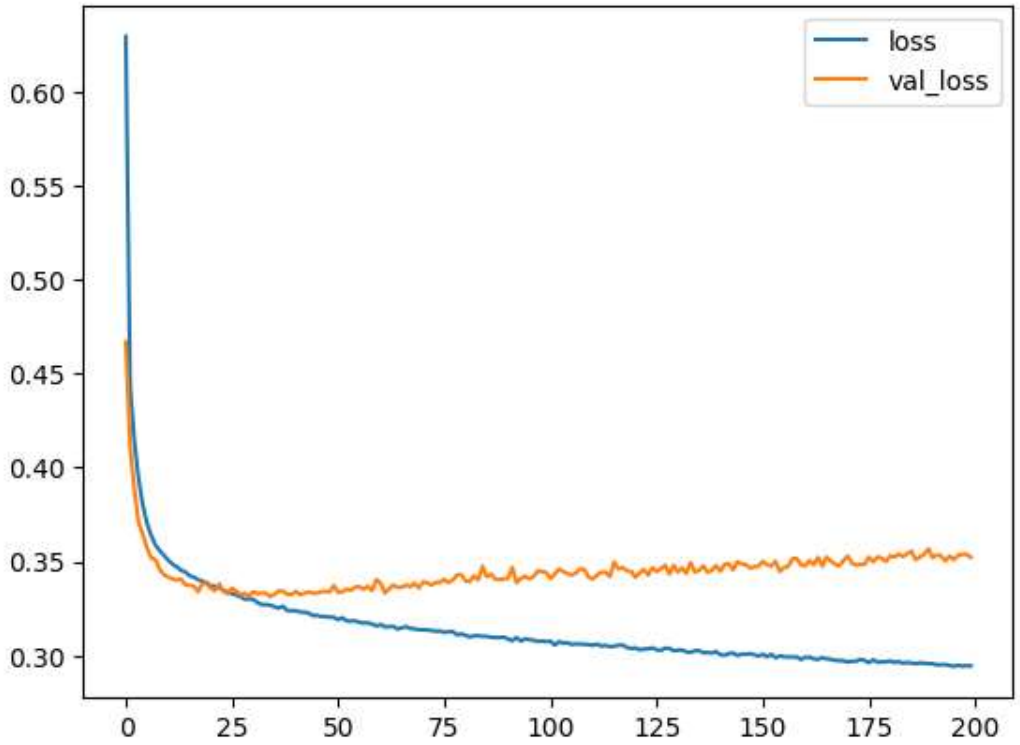
```
#ann.history.history


model_loss = pd.DataFrame(ann.history.history)
model_loss.head()
```

	loss	val_loss
0	0.629473	0.466923
1	0.443821	0.411328
3	0.394162	0.371012
4	0.379991	0.364356

Saving...

```
model_loss.plot()
```

```
<Axes: >
```




```
ypred=ann.predict(xtest)
```

```
94/94 [==============================] - 0s 2ms/step
```

```
ypred
```



```
[0.05692097],
[0.02869534],
...,
```

```
            [0.5670314 ],
            [0.31318736],
            [0.00165451]], dtype=float32)
```

```
ypred=np.where(ypred>0.5,1,0)
ypred
```

```
     array([[0],
            [0],
            [0],
            ...,
            [1],
            [0],
            [0]])
```

```
print(classification_report(ytest,ypred))
```

```
                  precision    recall  f1-score   support

               0       0.89      0.94      0.91      2389
               1       0.71      0.53      0.60       611

        accuracy                           0.86      3000
       macro avg       0.80      0.74      0.76      3000
    weighted avg       0.85      0.86      0.85      3000
```

```
print(confusion_matrix(ytest,ypred))
```

```
     [[2257  132]
      [ 289  322]]
```

Saving...

```
ann=Sequential()
ann.add(Dense(units=20,activation="relu"))
ann.add(Dense(units=20,activation="relu"))
ann.add(Dense(units=1,activation="sigmoid"))
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=['accuracy'])


from tensorflow.keras.callbacks import EarlyStopping


help(EarlyStopping)
```

```
|  on_train_batch_end(self, batch, logs=None)
|      Called at the end of a training batch in `fit` methods.
|
|      Subclasses should override for any actions to run.
|
|      Note that if the `steps_per_execution` argument to `compile` in
|      `tf.keras.Model` is set to `N`, this method will only be called every
|      `N` batches.
|
|      Args:
|          batch: Integer, index of batch within the current epoch.
|          logs: Dict. Aggregated metric results up until this batch.
|
|  set_model(self, model)
|
|  set_params(self, params)
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from Callback:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```

```
early_stop = EarlyStopping(monitor='val_loss', mode='min', patience=25)
```

```
ann.fit(xtrain,ytrain,epochs=200,batch_size=32,validation_data=(xtest, ytest),callbacks=[early_stop])
```

Saving...

```
Epoch 28/200
```

Saving...

```
Epoch 50/200
219/219 [==============================] - 1s 6ms/step - loss: 0.3229 - accuracy: 0.8637 - val_loss: 0.3337 - val_accuracy: 0.8643
<keras.callbacks.History at 0x7fbab83b5240>
```

```
#help(ann.fit)
```

```
model_loss = pd.DataFrame(ann.history.history)
model_loss.head()
```

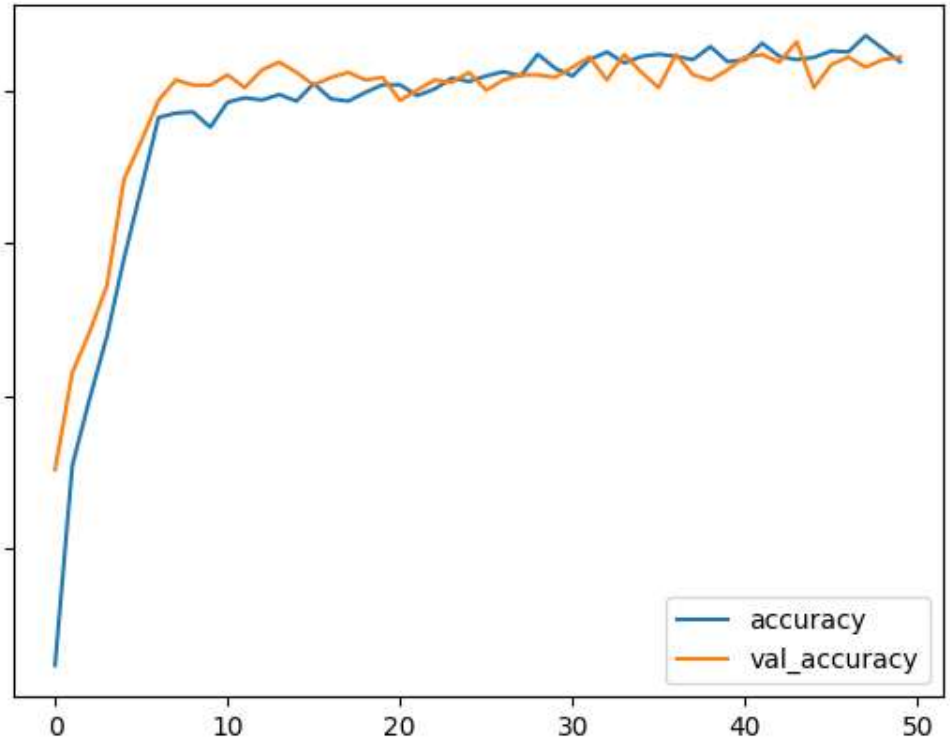
	loss	accuracy	val_loss	val_accuracy
0	0.523735	0.784714	0.447092	0.810333
1	0.449816	0.810857	0.414659	0.823000
2	0.429820	0.819571	0.401808	0.828333
3	0.414360	0.827714	0.387696	0.834333
4	0.395488	0.838000	0.369148	0.848333

```
model_loss[['accuracy','val_accuracy']].plot()
```

```
<Axes: >
```


```
model_loss[['accuracy','val_accuracy']].plot()
```

```
<Axes: >
```




```
predictions = ann.predict(xtest)
```

Saving...

```
=======] - 0s 1ms/step
```

```
predictions
```

```
array([[0.004102  ],
       [0.11235598],
       [0.05570493],
       ...,
       [0.46079794],
       [0.47469172],
       [0.01805007]], dtype=float32)
```

Adding in DropOut Layers

```
from tensorflow.keras.layers import Dropout


ann=Sequential()
ann.add(Dense(units=20,activation="relu"))
ann.add((Dropout(0.5)))

ann.add(Dense(units=20,activation="relu"))
ann.add((Dropout(0.5)))

ann.add(Dense(units=1,activation="sigmoid"))
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=['accuracy'])


ann.fit(xtrain,ytrain,epochs=200,batch_size=32,validation_data=(xtest, ytest),callbacks=[early_stop])
```

Saving...

```
<keras.callbacks.History at 0x7f0a782b9960>
```

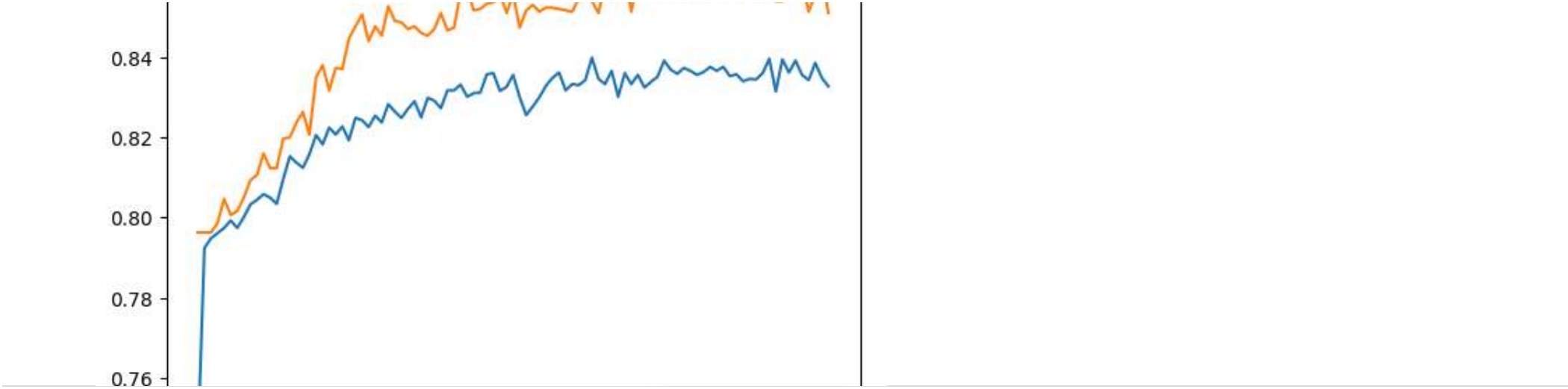
```
model_loss = pd.DataFrame(ann.history.history)
model_loss.head()
```

	loss	accuracy	val_loss	val_accuracy
0	0.585261	0.743857	0.490537	0.796333
1	0.527671	0.792429	0.457221	0.796333
2	0.497498	0.794857	0.439290	0.796333
3	0.489286	0.796143	0.431156	0.798667
4	0.471733	0.797429	0.423902	0.804667

```
model_loss[['accuracy','val_accuracy']].plot()
```

Saving...

```
<Axes: >
```



✓ 0s completed at 4:02 PM

Saving...