Problem Statement :

Customer churn or customer attrition is a tendency of clients or customers to abandon a brand and stop being a paying client of a particular business or organization. The percentage of customers that discontinue using a company's services or products during a specific period is called a customer churn rate. Several bad experiences (or just one) are enough, and a customer may quit. And if a large chunk of unsatisfied customers churn at a time interval, both material losses and damage to reputation would be enormous.

A reputed bank "ABC BANK" wants to predict the Churn rate. Create a model by using different machine learning approaches that can predict the best result.

Dataset Description :

This is a public dataset, The dataset format is given below.

Inside the dataset, there are 10000 rows and 14 different columns.

The target column here is Exited here.

```
import pandas as pd

df=pd.read_csv("https://raw.githubusercontent.com/sagnikghoshcr7/Bank-Customer-Churn-Prediction/master/Churn_Modelling.csv")

df.head()
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age
\
0          1    15634602  Hargrave          619    France  Female   42

1          2    15647311      Hill          608     Spain  Female   41

2          3    15619304      Onio          502    France  Female   42

3          4    15701354      Boni          699    France  Female   39

4          5    15737888  Mitchell          850     Spain  Female   43


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
```

```
3         93826.63       0
4         79084.10       0

df.tail()

      RowNumber  CustomerId    Surname  CreditScore Geography  Gender
Age  \
9995       9996    15606229   Obijiaku          771    France    Male
39
9996       9997    15569892  Johnstone          516    France    Male
35
9997       9998    15584532        Liu          709    France  Female
36
9998       9999    15682355  Sabbatini          772   Germany    Male
42
9999      10000    15628319     Walker          792    France  Female
28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

df.columns

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

#Dropping Irrelevant Features
df.drop(columns=['RowNumber','CustomerId','Surname'],inplace=True)

---------------------------------------------------------------------
-----
KeyError                                  Traceback (most recent call
last)
<ipython-input-11-075cb218a943> in <cell line: 2>()
      1 #Dropping Irrelevant Features
----> 2
```

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

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in
wrapper(*args, **kwargs)
    329                     stacklevel=find_stack_level(),
    330                 )
--> 331             return func(*args, **kwargs)
    332
    333         # error: "Callable[[VarArg(Any), KwArg(Any)], Any]"
has no

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py in
drop(self, labels, axis, index, columns, level, inplace, errors)
   5397                 weight  1.0     0.8
   5398         """
-> 5399         return super().drop(
   5400             labels=labels,
   5401             axis=axis,

/usr/local/lib/python3.10/dist-packages/pandas/util/_decorators.py in
wrapper(*args, **kwargs)
    329                     stacklevel=find_stack_level(),
    330                 )
--> 331             return func(*args, **kwargs)
    332
    333         # error: "Callable[[VarArg(Any), KwArg(Any)], Any]"
has no

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in
drop(self, labels, axis, index, columns, level, inplace, errors)
   4503         for axis, labels in axes.items():
   4504             if labels is not None:
-> 4505                 obj = obj._drop_axis(labels, axis,
level=level, errors=errors)
   4506
   4507         if inplace:

/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py in
_drop_axis(self, labels, axis, level, errors, only_slice)
   4544                 new_axis = axis.drop(labels, level=level,
errors=errors)
   4545             else:
-> 4546                 new_axis = axis.drop(labels, errors=errors)
   4547             indexer = axis.get_indexer(new_axis)
   4548

/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in
drop(self, labels, errors)
   6932         if mask.any():
   6933             if errors != "ignore":
```

```
-> 6934                 raise KeyError(f"{list(labels[mask])} not
found in axis")
   6935             indexer = indexer[~mask]
   6936         return self.delete(indexer)

KeyError: "['RowNumber', 'CustomerId', 'Surname'] not found in axis"
```

```
df.head()
```

```
   CreditScore Geography  Gender  Age  Tenure    Balance
NumOfProducts  \
0          619    France  Female   42       2       0.00
1
1          608     Spain  Female   41       1   83807.86
1
2          502    France  Female   42       8  159660.80
3
3          699    France  Female   39       1       0.00
2
4          850     Spain  Female   43       2  125510.82
1

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
3          0               0         93826.63       0
4          1               1         79084.10       0
```

```
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
```

```
df.describe()
```

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

         HasCrCard  IsActiveMember  EstimatedSalary        Exited
count  10000.00000    10000.000000     10000.000000  10000.000000
mean       0.70550        0.515100    100090.239881      0.203700
std        0.45584        0.499797     57510.492818      0.402769
min        0.00000        0.000000        11.580000      0.000000
25%        0.00000        0.000000     51002.110000      0.000000
50%        1.00000        1.000000    100193.915000      0.000000
75%        1.00000        1.000000    149388.247500      0.000000
max        1.00000        1.000000    199992.480000      1.000000
```

```
#value counts()
df["Geography"].value_counts()
```

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

```
#value counts()
df["Gender"].value_counts()
```

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

Encoding Categorical Data

```
df=pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)

df.head()
```

```
   CreditScore  Age  Tenure    Balance  NumOfProducts  HasCrCard  \
0          619   42       2       0.00              1          1
1          608   41       1   83807.86              1          0
2          502   42       8  159660.80              3          1
3          699   39       1       0.00              2          0
4          850   43       2  125510.82              1          1

   IsActiveMember  EstimatedSalary  Exited  Geography_Germany  \
0               1        101348.88       1                  0
1               1        112542.58       0                  0
2               0        113931.57       1                  0
3               0         93826.63       0                  0
4               1         79084.10       0                  0

   Geography_Spain  Gender_Male
0                0            0
1                1            0
2                0            0
3                0            0
4                1            0
```

Some insights about the target variable

```
df["Exited"].value_counts()

0    7963
1    2037
Name: Exited, dtype: int64

X=df.drop(['Exited'],axis=1)
y=df["Exited"]

X

      CreditScore  Age  Tenure    Balance  NumOfProducts  HasCrCard  \
0             619   42       2       0.00              1          1
1             608   41       1   83807.86              1          0
2             502   42       8  159660.80              3          1
3             699   39       1       0.00              2          0
4             850   43       2  125510.82              1          1
...           ...  ...     ...        ...            ...        ...
9995          771   39       5       0.00              2          1
9996          516   35      10   57369.61              1          1
9997          709   36       7       0.00              1          0
9998          772   42       3   75075.31              2          1
9999          792   28       4  130142.79              1          1

      IsActiveMember  EstimatedSalary  Geography_Germany
```

```
Geography_Spain  \
0                   1        101348.88                    0
0
1                   1        112542.58                    0
1
2                   0        113931.57                    0
0
3                   0         93826.63                    0
0
4                   1         79084.10                    0
1
...               ...              ...                  ...
...
9995                0         96270.64                    0
0
9996                1        101699.77                    0
0
9997                1         42085.58                    0
0
9998                0         92888.52                    1
0
9999                0         38190.78                    0
0

      Gender_Male
0               0
1               0
2               0
3               0
4               0
...           ...
9995            1
9996            1
9997            0
9998            1
9999            0

[10000 rows x 11 columns]

y

0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
```

```
9998    1
9999    0
Name: Exited, Length: 10000, dtype: int64
```

Keras:

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Keras allows you to switch between different back ends. The frameworks supported by Keras are:

```
Tensorflow

Theano

PlaidML

MXNet

CNTK (Microsoft Cognitive Toolkit )
```

Why Do We Need Keras?

Keras is an API that was made to be easy to learn for people. Keras was made to be simple. It offers consistent & simple APIs, reduces the actions required to implement common code, and explains user error clearly.

Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs.

Languages with a high level of abstraction and inbuilt features are slow and building custom features in then can be hard. But Keras runs on top of TensorFlow and is relatively fast. Keras is also deeply integrated with TensorFlow, so you can create customized workflows with ease.

The research community for Keras is vast and highly developed. The documentation and help available are far more extensive than other deep learning frameworks.

Keras is used commercially by many companies like Netflix, Uber, Square, Yelp, etc which have deployed products in the public domain which are built using Keras.

Apart from this, Keras has features such as :

```
It runs smoothly on both CPU and GPU.

It supports almost all neural network models.

It is modular in nature, which makes it expressive, flexible, and apt
for innovative research.
```

Applications of Keras

```
Keras is used for creating deep models which can be productized on
smartphones.


Keras is also used for distributed training of deep learning models.

Keras is used by companies such as Netflix, Yelp, Uber, etc.

Keras is also extensively used in deep learning competitions to create
and deploy working models, which are fast in a short amount of time.
```

```
import tensorflow
from tensorflow import keras

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

len(X.columns)
```

```
11
```

```
model=Sequential()
model.add(Dense(11,activation="sigmoid",input_dim=11))#input layer
model.add(Dense(11,activation="sigmoid"))#hidden layer
model.add(Dense(1,activation="sigmoid"))#output layer

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 11)                132

 dense_1 (Dense)             (None, 11)                132

 dense_2 (Dense)             (None, 1)                 12

=================================================================
Total params: 276
Trainable params: 276
Non-trainable params: 0
_________________________________________________________________
```

```
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,rando
m_state=0)
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()

X_train_scale=scaler.fit_transform(X_train)
X_test_scale=scaler.transform(X_test)

#The model.compile function in Keras is used to specify the optimizer,
loss function, and evaluation metrics that will be used during the
training of the neural network model.

model.compile(optimizer="Adam",
              loss="binary_crossentropy",
              metrics=["accuracy"]


)

model.fit(X_train_scale,y_train,batch_size=50,epochs=10,verbose=1)
```

```
Epoch 1/10
160/160 [==============================] - 1s 2ms/step - loss: 0.4325
- accuracy: 0.8074
Epoch 2/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4297
- accuracy: 0.8102
Epoch 3/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4279
- accuracy: 0.8105
Epoch 4/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4263
- accuracy: 0.8114
Epoch 5/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4250
- accuracy: 0.8119
Epoch 6/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4235
- accuracy: 0.8155
Epoch 7/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4220
- accuracy: 0.8174
Epoch 8/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4206
- accuracy: 0.8191
Epoch 9/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4193
- accuracy: 0.8207
Epoch 10/10
160/160 [==============================] - 0s 2ms/step - loss: 0.4176
- accuracy: 0.8234

<keras.callbacks.History at 0x78b4377885b0>
```

X_train_scale: Your scaled training data.

y_train: The corresponding target labels for your training data.

batch_size: The number of samples to use in each iteration of gradient descent during training. In this case, you're using a batch size of 50, which means that the model will update its weights after every 50 samples.

epochs: The number of times the training dataset will be passed through the model during training. You're running 10 epochs, so the model will go through the entire training dataset 10 times.

verbose: This controls the verbosity of the training process. Setting it to 1 means that progress bars and logs will be displayed during training, providing you with feedback on the training progress.

```
y_pred=model.predict(X_test_scale)
```

```
63/63 [==============================] - 0s 2ms/step
```

```
y_pred
```

```
array([[0.19777617],
       [0.36974898],
       [0.14303005],
       ...,
       [0.1319361 ],
       [0.09910323],
       [0.14592502]], dtype=float32)
```

```
y_pred=y_pred.argmax(axis=-1)
```

```
from sklearn.metrics import accuracy_score #used to calculate the accuracy of classification models
```

```
accuracy_score(y_test,y_pred)
```

```
0.7975
```