jupyter Customer Churn Dataset Last Checkpoint: 11 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

What is Customer Churn?

Customer churn refers to the rate at which customers stop doing business with a company over a certain period of time. It is a crucial metric for businesses, especially those in subscription-based models or with recurring revenue streams, as it directly impacts revenue and profitability. Churn can occur for various reasons, including dissatisfaction with the product or service, better offers from competitors, changes in customer needs or circumstances, or simply neglect from the company's end in maintaining customer relationships.

In [1]:
```
import pandas as pd
```

In [2]:
```
#Loading data
df = pd.read_csv(r'C:\Users\VINAY\Downloads\archive (10).zip')
```

Understanding the data

Each row represents a customer, each column contains customer's attributes described on the column Metadata.

jupyter Customer Churn Dataset Last Checkpoint: 11 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [5]:
```
df.head()
```

Out[5]:

RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalar
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.8
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.5
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.5
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.6
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1

The data set includes information about:

- **Customers who left within the last month** – the column is called Churn
- **Services that each customer has signed up for** – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies
- **Customer account information** - how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- **Demographic info about customers** – gender, age range, and if they have partners and dependents

jupyter Customer Churn Dataset Last Checkpoint: 12 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

jupyter Customer Churn Dataset Last Checkpoint: 13 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [9]:
```
df['Exited'].value_counts()
```

Out[9]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

In [10]:
```
df.drop(columns=['RowNumber','CustomerId','Surname'],inplace=True)
```

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

Out[11]:

	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

jupyter Customer Churn Dataset Last Checkpoint: 13 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [12]:
```
df = pd.get_dummies(df,columns=['Geography','Gender'],drop_first=True)
```

In [13]:
```
df
```

Out[13]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography_Germany	Geography_Spain
0	619	42	2	0.00	1	1	1	101348.88	1	False	False
1	608	41	1	83807.86	1	0	1	112542.58	0	False	True
2	502	42	8	159660.80	3	1	0	113931.57	1	False	False
3	699	39	1	0.00	2	0	0	93826.63	0	False	False
4	850	43	2	125510.82	1	1	1	79084.10	0	False	True
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0	False	False
9996	516	35	10	57369.61	1	1	1	101699.77	0	False	False
9997	709	36	7	0.00	1	0	1	42085.58	1	False	False
9998	772	42	3	75075.31	2	1	0	92888.52	1	True	False
9999	792	28	4	130142.79	1	1	0	38190.78	0	False	False

jupyter Customer Churn Dataset Last Checkpoint: 14 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

Data Preprocessing

Splitting the data into train and test sets

In [14]:

```
X = df.drop(columns=['Exited'])
y = df['Exited']
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=1)
```

In [15]:

```
X
```

Out[15]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_Germany	Geography_Spain	Gender_
0	619	42	2	0.00	1	1	1	101348.88	False	False	
1	608	41	1	83807.86	1	0	1	112542.58	False	True	
2	502	42	8	159660.80	3	1	0	113931.57	False	False	
3	699	39	1	0.00	2	0	0	93826.63	False	False	
4	850	43	2	125510.82	1	1	1	79084.10	False	True	
...	...	...	...	...	...	...	...	...	...	...	

jupyter Customer Churn Dataset Last Checkpoint: 14 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

9996	516	35	10	57369.61	1	1	1	101699.77	False	False
9997	709	36	7	0.00	1	0	1	42085.58	False	False
9998	772	42	3	75075.31	2	1	0	92888.52	True	False
9999	792	28	4	130142.79	1	1	0	38190.78	False	False

10000 rows × 11 columns

In [16]:
```
y
```

Out[16]:
```
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
```

jupyter Customer Churn Dataset Last Checkpoint: 14 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

9996	516	35	10	57369.61	1	1	1	101699.77	False	False
9997	709	36	7	0.00	1	0	1	42085.58	False	False
9998	772	42	3	75075.31	2	1	0	92888.52	True	False
9999	792	28	4	130142.79	1	1	0	38190.78	False	False

10000 rows × 11 columns

In [16]:
```
y
```

Out[16]:
```
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
```

jupyter Customer Churn Dataset Last Checkpoint: 15 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [17]:
```
X_train.shape
```

Out[17]:
```
(8000, 11)
```

In [18]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [19]:
```
X_train_scaled
```

Out[19]:
```
array([[-0.23082038, -0.94449979, -0.70174202, ...,  1.71490137,
        -0.57273139,  0.91509065],
       [-0.25150912, -0.94449979, -0.35520275, ..., -0.58312392,
        -0.57273139, -1.09278791],
       [-0.3963303 ,  0.77498705,  0.33787579, ...,  1.71490137,
        -0.57273139, -1.09278791],
       ...,
       [ 0.22433188,  0.58393295,  1.3774936 , ..., -0.58312392,
        -0.57273139, -1.09278791],
       [ 0.13123255,  0.01077067,  1.03095433, ..., -0.58312392,
        -0.57273139, -1.09278791],
```

jupyter Customer Churn Dataset Last Checkpoint: 15 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

```
[ 1.1656695 ,  0.29735181,  0.33787579, ...,  1.71490137,
 -0.57273139,  0.91509065]])
```

In [20]:

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

In [21]:

```
model = Sequential()

model.add(Dense(11,activation='relu',input_dim = 11))
model.add(Dense(11,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
```

```
C:\Users\VINAY\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:85: UserWarning: Do not pass an `input_shape`/`inp
ut_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

In [22]:

```
model.summary()
```

Model: "sequential"

jupyter Customer Churn Dataset Last Checkpoint: 16 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

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 (1.08 KB)

Trainable params: 276 (1.08 KB)

Non-trainable params: 0 (0.00 B)

In [23]:
```
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

In [24]:
```
history = model.fit(X_train_scaled,y_train,epochs=100,validation_split=0.2)
```

```
Epoch 1/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.7162 - loss: 0.5798 - val_accuracy: 0.7987 - val_loss: 0.4625
Epoch 2/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.7908 - loss: 0.4616 - val_accuracy: 0.8075 - val_loss: 0.4369
Epoch 3/100
```

jupyter Customer Churn Dataset Last Checkpoint: 16 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

```
Epoch 97/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8668 - loss: 0.3229 - val_accuracy: 0.8506 - val_loss: 0.3569
Epoch 98/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8641 - loss: 0.3387 - val_accuracy: 0.8500 - val_loss: 0.3542
Epoch 99/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8633 - loss: 0.3223 - val_accuracy: 0.8494 - val_loss: 0.3549
Epoch 100/100
200/200 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8643 - loss: 0.3226 - val_accuracy: 0.8487 - val_loss: 0.3568
```

In [25]:
```
model.layers[2].get_weights()
```

Out[25]:
```
[array([[ 0.84586996],
        [ 1.0174724 ],
        [-0.4084958 ],
        [ 1.1360463 ],
        [-0.0616455 ],
        [-0.6894243 ],
        [-0.65696234],
        [ 1.5234168 ],
        [-0.7272619 ],
        [-0.5272417 ],
        [ 1.0012852 ]], dtype=float32),
 array([-0.33936828], dtype=float32)]
```

jupyter Customer Churn Dataset Last Checkpoint: 16 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

In [26]:
```
y_log = model.predict(X_test_scaled)
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

In [27]:
```
import numpy as np
```

In [28]:
```
y_pred = np.where(y_log>0.5,1,0)
```

In [29]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[29]:
```
0.86
```

In [30]:
```
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [31]:
```
plt.plot(history.history['loss'])
```

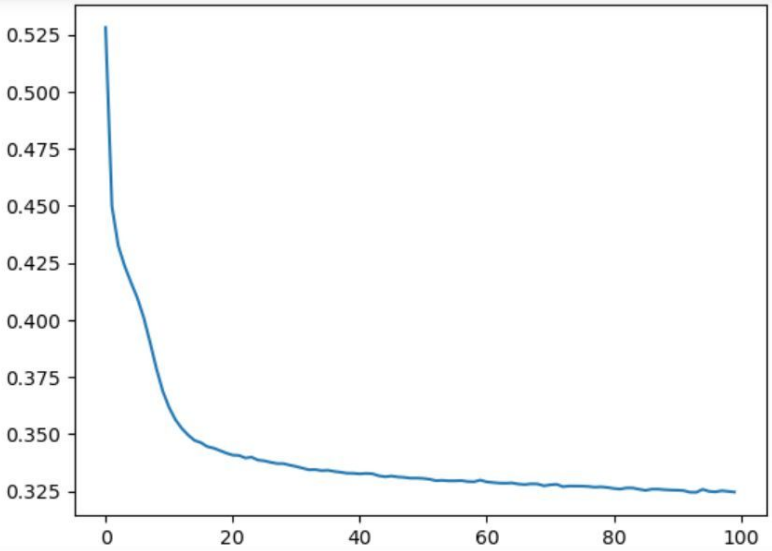
Out[31]:
```
[<matplotlib.lines.Line2D at 0x24cb7b240d0>]
```

jupyter Customer Churn Dataset Last Checkpoint: 17 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Markdown

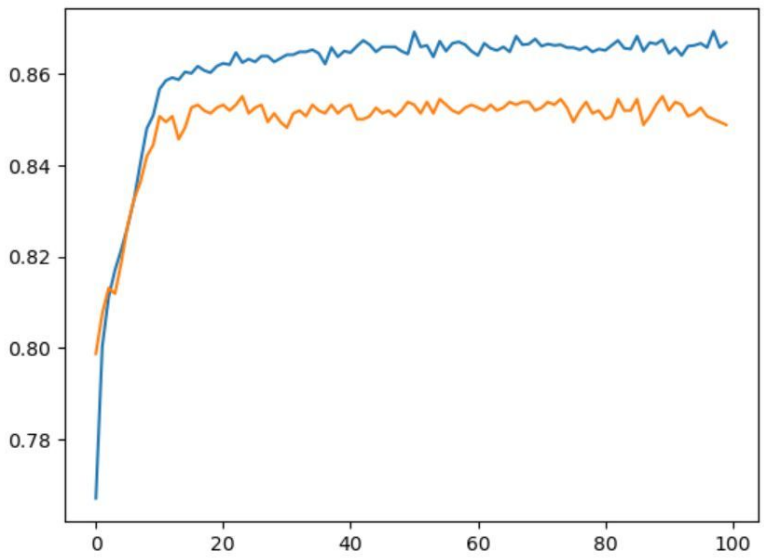


File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

In [33]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x24cbc920b10>]
```



File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

Loading Data

In [34]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, classification_report
```

Machine Learning Model Evaluations and Predictions

KNN

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

In [35]:
```
knn_model = KNeighborsClassifier(n_neighbors = 11)
knn_model.fit(X_train,y_train)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
print("KNN accuracy:",accuracy_knn)
```

```
KNN accuracy: 0.7865
```

In [36]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.80      0.98      0.88      1585
           1       0.36      0.04      0.07       415

    accuracy                           0.79      2000
   macro avg       0.58      0.51      0.47      2000
weighted avg       0.70      0.79      0.71      2000
```

SVC

In [37]:
```
svc_model = SVC(random_state = 1)
svc_model.fit(X_train,y_train)
predict_y = svc_model.predict(X_test)
accuracy_svc = svc_model.score(X_test,y_test)
print("SVM accuracy is :",accuracy_svc)
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

```
SVM accuracy is : 0.7925
```

In [38]:

```
print(classification_report(y_test, predict_y))
```

```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88      1585
           1       0.00      0.00      0.00       415

    accuracy                           0.79      2000
   macro avg       0.40      0.50      0.44      2000
weighted avg       0.63      0.79      0.70      2000
```

Logistic Regression

In [39]:

```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

```
Logistic Regression accuracy is : 0.7855
```

In [40]:

```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

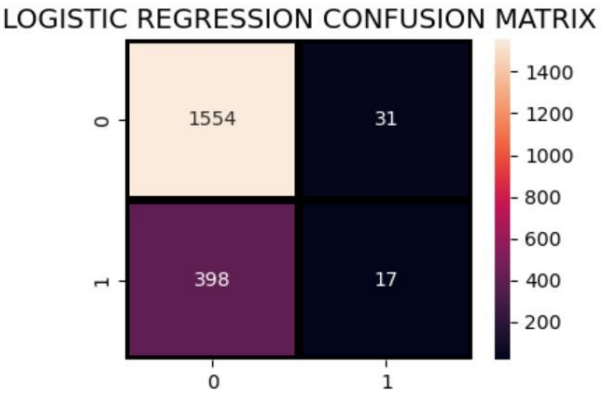
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

In [41]:
```
import seaborn as sns
```

In [42]:
```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, lr_pred),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title("LOGISTIC REGRESSION CONFUSION MATRIX",fontsize=14)
plt.show()
```



File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Markdown

From the confusion matrix we can see that:

There are total 1554+31=1585 actual non-churn values and the algorithm predicts 1554 of them as non churn and 31 of them as churn. While there are 398+17=415 actual churn values and the algorithm predicts 398 of them as non churn values and 17 of them as churn values.

Decision Tree Classifier

In [43]:

```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train,y_train)
predictdt_y = dt_model.predict(X_test)
accuracy_dt = dt_model.score(X_test,y_test)
print("Decision Tree accuracy is :",accuracy_dt)
```

```
Decision Tree accuracy is : 0.795
```

In [44]:

```
print(classification_report(y_test, predictdt_y))
```

```
              precision    recall  f1-score   support

           0       0.87      0.87      0.87      1585
           1       0.51      0.52      0.51       415

    accuracy                           0.80      2000
   macro avg       0.69      0.69      0.69      2000
weighted avg       0.80      0.80      0.80      2000
```