Customer churn also known as customer defection is the loss of clients or customers. Telephone service companies often use customer churn analysis and customer churning rates as one of their key business metrics. For this project, we will be exploring the dataset of a telecom company and try to predict the customer churn. Using a neuron network binary classification, classify whether or not the customer will churn.

In [1]:

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

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler
```

In [2]:

```
df=pd.read_csv("telecom_churn.csv")
```

In [3]:

```
df
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	7590-VHVEG	Female	0	Yes	No	1	No	No ph ser
1	5575-GNVDE	Male	0	No	No	34	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	No ph ser
4	9237-HQITU	Female	0	No	No	2	Yes	
...	...	...	...	...	...	...	...	
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No ph ser
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	
7042	3186-AJIEK	Male	0	No	No	66	Yes	

7043 rows × 21 columns

In [4]:

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

Out[4]:

```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [6]:

```
df.drop(["customerID"],axis=1, inplace=True)
```

In [7]:

```
df.insert(0, 'ID', range(1, 1 + len(df)))
df.set_index("ID",inplace=True)
```

In [8]:

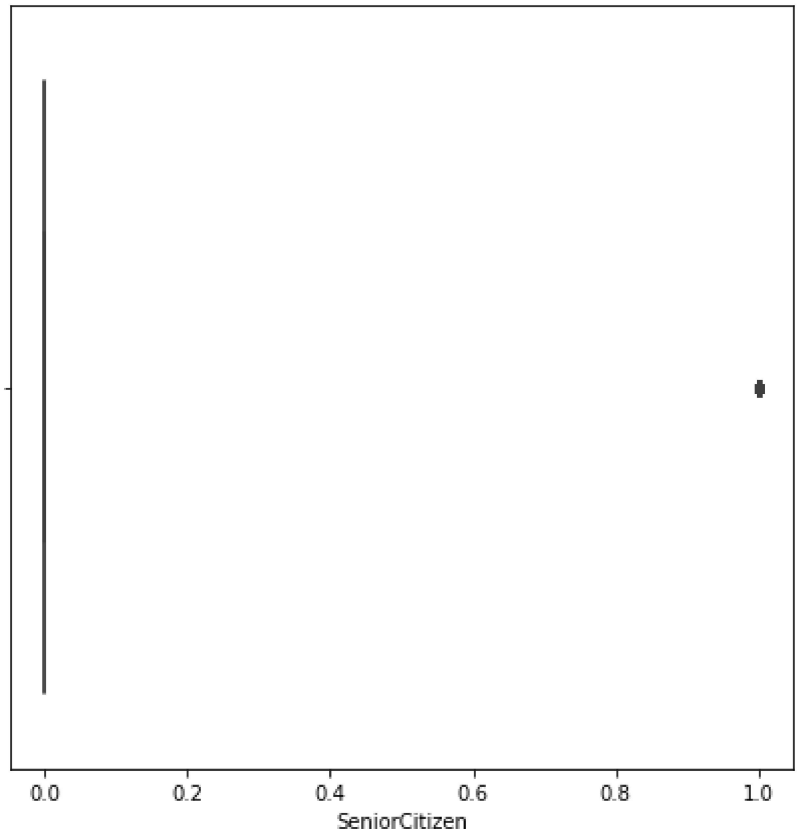
```
df_num=df.select_dtypes(['int64','float64'])
```

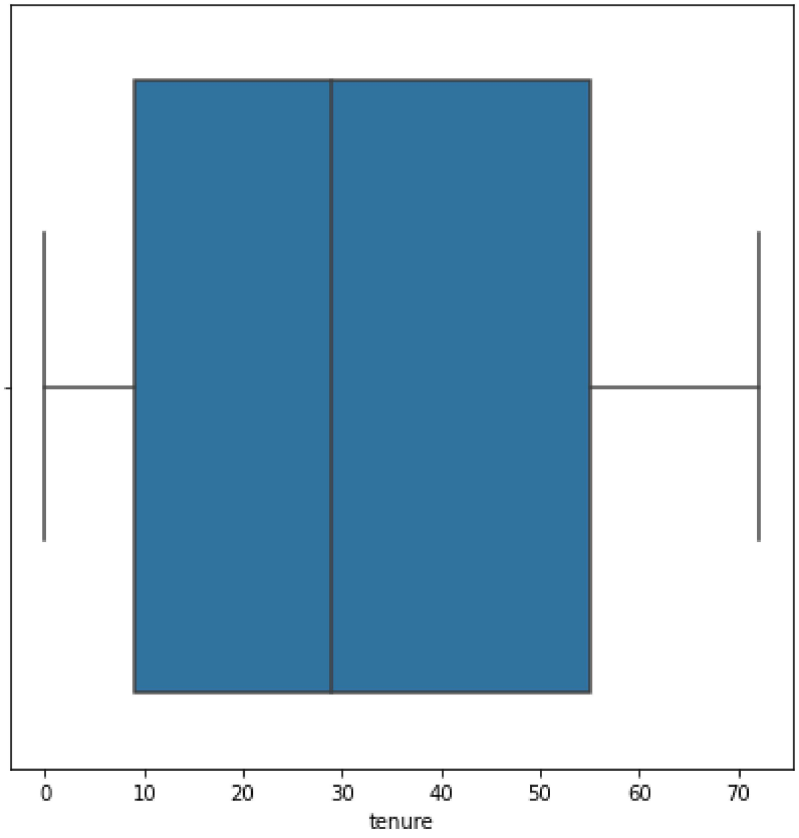
In [9]:

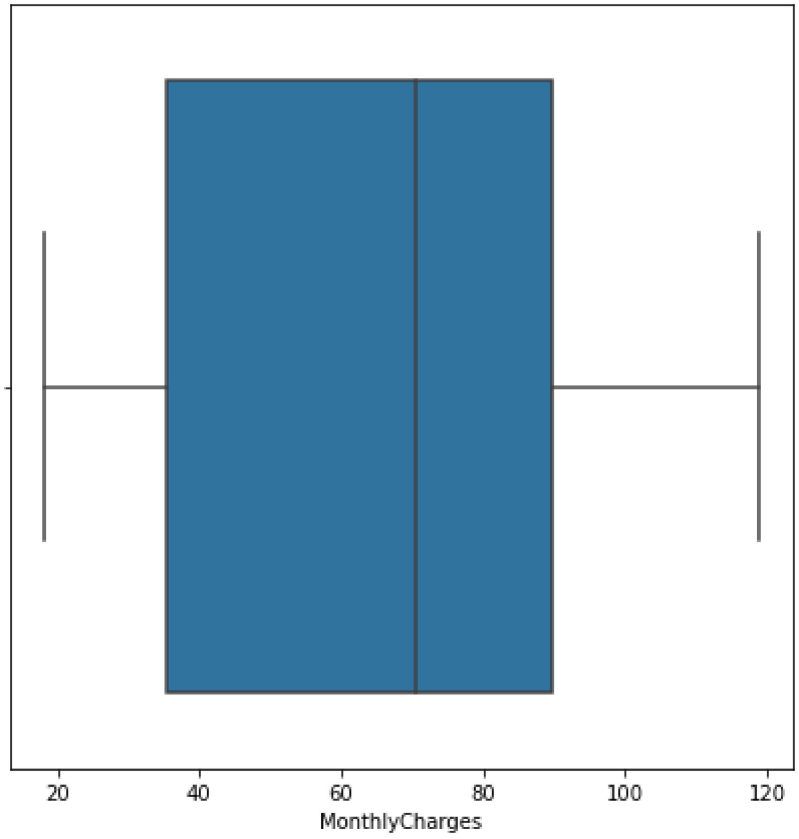
```
df_cat=df.select_dtypes(['object'])
```

In [10]:

```
for i in df_num:
    plt.figure(figsize=(7,7))
    sns.boxplot(data=df_num,x=i,whis=3)
    # upper whisker = q3+1.5*IQR
    # lower whisker = q1 - 1.5*IQR
    # boxplot will calculate upper whisker and lower whisker by it's own and the nit will p
    plt.show()
```




0
10
20
30
40
50
60
70
tenure


20
40
60
80
100
120
MonthlyCharges

In [11]:

```
from sklearn.preprocessing import LabelEncoder
```

In [12]:

```
le=LabelEncoder()
for col in df_cat:
    df_cat[col]=le.fit_transform(df_cat[col])
```

In [13]:

```
df_new=pd.merge(df_num,df_cat,on="ID")
```

In [14]:

```
df_new
```

Out[14]:

ID	SeniorCitizen	tenure	MonthlyCharges	gender	Partner	Dependents	PhoneService	Multi
1	0	1	29.85	0	1	0	0	
2	0	34	56.95	1	0	0	1	
3	0	2	53.85	1	0	0	1	
4	0	45	42.30	1	0	0	0	
5	0	2	70.70	0	0	0	1	
...	...	...	...	...	...	...	...	
7039	0	24	84.80	1	1	1	1	
7040	0	72	103.20	0	1	1	1	
7041	0	11	29.60	0	1	1	0	
7042	1	4	74.40	1	1	0	1	
7043	0	66	105.65	1	0	0	1	

7043 rows × 20 columns

In [15]:

```
x=df_new.drop("Churn",axis=1)
y=df_new["Churn"]
```

In [16]:

```
ss=StandardScaler()
x = ss.fit_transform(x)
```

In [17]:

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

In [18]:

```
model = tf.keras.Sequential(
    [tf.keras.layers.Dense(2, activation="relu", input_shape=(x.shape[1],)),
     tf.keras.layers.Dense(3, activation="relu"),
     tf.keras.layers.Dense(1, activation="sigmoid")]
)
```

In [19]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 2)                 40
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 9
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 4
=================================================================
Total params: 53
Trainable params: 53
Non-trainable params: 0
_________________________________________________________________
```

In [20]:

```
model.compile(optimizer="sgd", loss="binary_crossentropy")
```

In [21]:

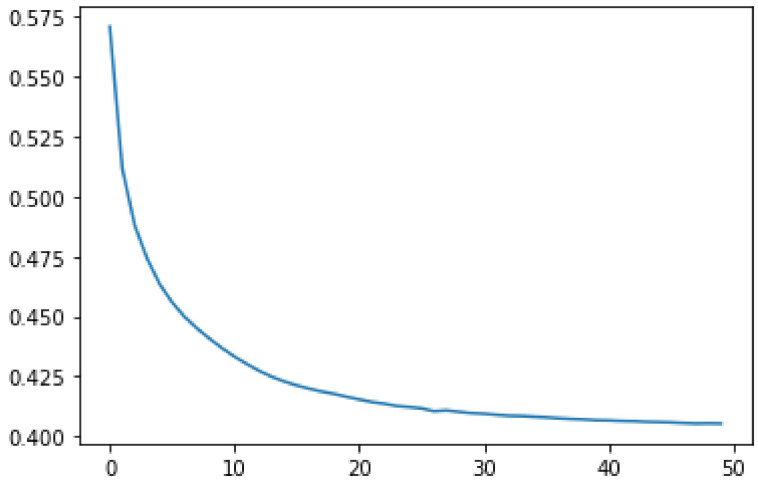
```
trained_model = model.fit(x_train, y_train, epochs=50, batch_size=20)
```

```
Epoch 1/50
247/247 [==============================] - 1s 523us/step - loss: 0.6019
Epoch 2/50
247/247 [==============================] - 0s 523us/step - loss: 0.5238
Epoch 3/50
247/247 [==============================] - 0s 503us/step - loss: 0.4849
Epoch 4/50
247/247 [==============================] - 0s 503us/step - loss: 0.4763
Epoch 5/50
247/247 [==============================] - 0s 555us/step - loss: 0.4623
Epoch 6/50
247/247 [==============================] - 0s 515us/step - loss: 0.4526
Epoch 7/50
247/247 [==============================] - 0s 503us/step - loss: 0.4500
Epoch 8/50
247/247 [==============================] - 0s 491us/step - loss: 0.4513
Epoch 9/50
247/247 [==============================] - 0s 507us/step - loss: 0.4386
Epoch 10/50
247/247 [==============================] - 0s 503us/step - loss: 0.4332
Epoch 11/50
247/247 [==============================] - 0s 499us/step - loss: 0.4234
Epoch 12/50
247/247 [==============================] - 0s 499us/step - loss: 0.4318
Epoch 13/50
247/247 [==============================] - 0s 543us/step - loss: 0.4213
Epoch 14/50
247/247 [==============================] - 0s 515us/step - loss: 0.4254
Epoch 15/50
247/247 [==============================] - 0s 503us/step - loss: 0.4195
Epoch 16/50
247/247 [==============================] - 0s 503us/step - loss: 0.4120
Epoch 17/50
247/247 [==============================] - 0s 511us/step - loss: 0.4237
Epoch 18/50
247/247 [==============================] - 0s 499us/step - loss: 0.4121
Epoch 19/50
247/247 [==============================] - 0s 499us/step - loss: 0.4139
Epoch 20/50
247/247 [==============================] - 0s 503us/step - loss: 0.4249
Epoch 21/50
247/247 [==============================] - 0s 551us/step - loss: 0.4087
Epoch 22/50
247/247 [==============================] - 0s 503us/step - loss: 0.4215
Epoch 23/50
247/247 [==============================] - 0s 507us/step - loss: 0.4136
Epoch 24/50
247/247 [==============================] - 0s 507us/step - loss: 0.4114
Epoch 25/50
247/247 [==============================] - 0s 503us/step - loss: 0.4184
Epoch 26/50
247/247 [==============================] - 0s 503us/step - loss: 0.4007
Epoch 27/50
247/247 [==============================] - 0s 503us/step - loss: 0.3962
Epoch 28/50
247/247 [==============================] - 0s 539us/step - loss: 0.4121
```

```
Epoch 29/50
247/247 [==============================] - 0s 689us/step - loss: 0.4074
Epoch 30/50
247/247 [==============================] - 0s 596us/step - loss: 0.4105
Epoch 31/50
247/247 [==============================] - 0s 584us/step - loss: 0.3965
Epoch 32/50
247/247 [==============================] - 0s 531us/step - loss: 0.4155
Epoch 33/50
247/247 [==============================] - 0s 600us/step - loss: 0.4010
Epoch 34/50
247/247 [==============================] - 0s 474us/step - loss: 0.4112
Epoch 35/50
247/247 [==============================] - 0s 482us/step - loss: 0.4113
Epoch 36/50
247/247 [==============================] - 0s 499us/step - loss: 0.4041
Epoch 37/50
247/247 [==============================] - 0s 657us/step - loss: 0.4109
Epoch 38/50
247/247 [==============================] - 0s 539us/step - loss: 0.4087
Epoch 39/50
247/247 [==============================] - 0s 482us/step - loss: 0.4075
Epoch 40/50
247/247 [==============================] - 0s 462us/step - loss: 0.3934
Epoch 41/50
247/247 [==============================] - 0s 507us/step - loss: 0.4028
Epoch 42/50
247/247 [==============================] - 0s 458us/step - loss: 0.4124
Epoch 43/50
247/247 [==============================] - 0s 487us/step - loss: 0.4068
Epoch 44/50
247/247 [==============================] - 0s 568us/step - loss: 0.4235
Epoch 45/50
247/247 [==============================] - 0s 515us/step - loss: 0.3954
Epoch 46/50
247/247 [==============================] - 0s 539us/step - loss: 0.4050
Epoch 47/50
247/247 [==============================] - 0s 497us/step - loss: 0.4079
Epoch 48/50
247/247 [==============================] - 0s 456us/step - loss: 0.4108
Epoch 49/50
247/247 [==============================] - 0s 454us/step - loss: 0.3955
Epoch 50/50
247/247 [==============================] - 0s 470us/step - loss: 0.4057
```

In [22]:

```
plt.plot(trained_model.history["loss"])
plt.show()
```



In [23]:

```
y_hat = model.predict(x_test)
```

In [24]:

```
y_hat
```

Out[24]:

```
array([[0.09079063],
       [0.17288956],
       [0.01384917],
       ...,
       [0.1909835 ],
       [0.5906654 ],
       [0.46951127]], dtype=float32)
```

In [25]:

```
y_hat1 = np.where(y_hat >= 0.5, 1, 0)
```

In [26]:

```
y_hat1.flatten()
```

Out[26]:

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

In [27]:

```
roc_auc_score(y_test, y_hat1)
```

Out[27]:

```
0.7056512059784311
```

In [ ]: