Start coding or generate with AI.

```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings('ignore')
```

Start coding or generate with AI.

```
data=pd.read_csv('data.csv')
```

```
data.shape
```

```
(2000, 16)
```

```
data.head()
```

	year	customer_id	phone_no	gender	age	no_of_days_subscribed	multi_screen	mail_subscribed	weekly_mins_watched	minimum_daily_mi
0	2015	100198	409-8743	Female	36	62	no	no	148.35	12
1	2015	100643	340-5930	Female	39	149	no	no	294.45	7
2	2015	100756	372-3750	Female	65	126	no	no	87.30	11
3	2015	101595	331-4902	Female	24	131	no	yes	321.30	9
4	2015	101653	351-8398	Female	40	191	no	no	243.00	10

```
data.describe()
```

	year	customer_id	age	no_of_days_subscribed	weekly_mins_watched	minimum_daily_mins	maximum_daily_mins	weekly_max_
count	2000.0	2000.000000	2000.00000	2000.000000	2000.000000	2000.000000	2000.000000	2
mean	2015.0	554887.157500	38.69050	99.750000	270.178425	10.198700	30.620780	
std	0.0	261033.690318	10.20641	39.755386	80.551627	2.785519	9.129165	
min	2015.0	100198.000000	18.00000	1.000000	0.000000	0.000000	0.000000	
25%	2015.0	328634.750000	32.00000	73.000000	218.212500	8.400000	24.735000	
50%	2015.0	567957.500000	37.00000	99.000000	269.925000	10.200000	30.590000	
75%	2015.0	773280.250000	44.00000	127.000000	324.675000	12.000000	36.797500	
max	2015.0	999961.000000	82.00000	243.000000	526.200000	20.000000	59.640000	

```
data.isnull().sum()
```

```
year                      0
customer_id               0
phone_no                  0
gender                   24
age                       0
no_of_days_subscribed     0
multi_screen              0
mail_subscribed           0
weekly_mins_watched       0
minimum_daily_mins        0
maximum_daily_mins        0
weekly_max_night_mins     0
videos_watched            0
maximum_days_inactive    28
customer_support_calls    0
churn                    35
dtype: int64
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   year                    2000 non-null   int64
 1   customer_id             2000 non-null   int64
 2   phone_no                2000 non-null   object
 3   gender                  1976 non-null   object
 4   age                     2000 non-null   int64
 5   no_of_days_subscribed   2000 non-null   int64
 6   multi_screen            2000 non-null   object
 7   mail_subscribed         2000 non-null   object
 8   weekly_mins_watched     2000 non-null   float64
 9   minimum_daily_mins      2000 non-null   float64
 10  maximum_daily_mins      2000 non-null   float64
 11  weekly_max_night_mins   2000 non-null   int64
 12  videos_watched          2000 non-null   int64
 13  maximum_days_inactive   1972 non-null   float64
 14  customer_support_calls  2000 non-null   int64
 15  churn                   1965 non-null   float64
dtypes: float64(5), int64(7), object(4)
memory usage: 250.1+ KB
```

```
data['gender'].value_counts()
```

```
Male      1053
Female     923
Name: gender, dtype: int64
```

```
data['multi_screen'].value_counts()
```

```
no     1802
yes     198
Name: multi_screen, dtype: int64
```

```
data['mail_subscribed'].value_counts()
```

```
no     1430
yes     570
Name: mail_subscribed, dtype: int64
```

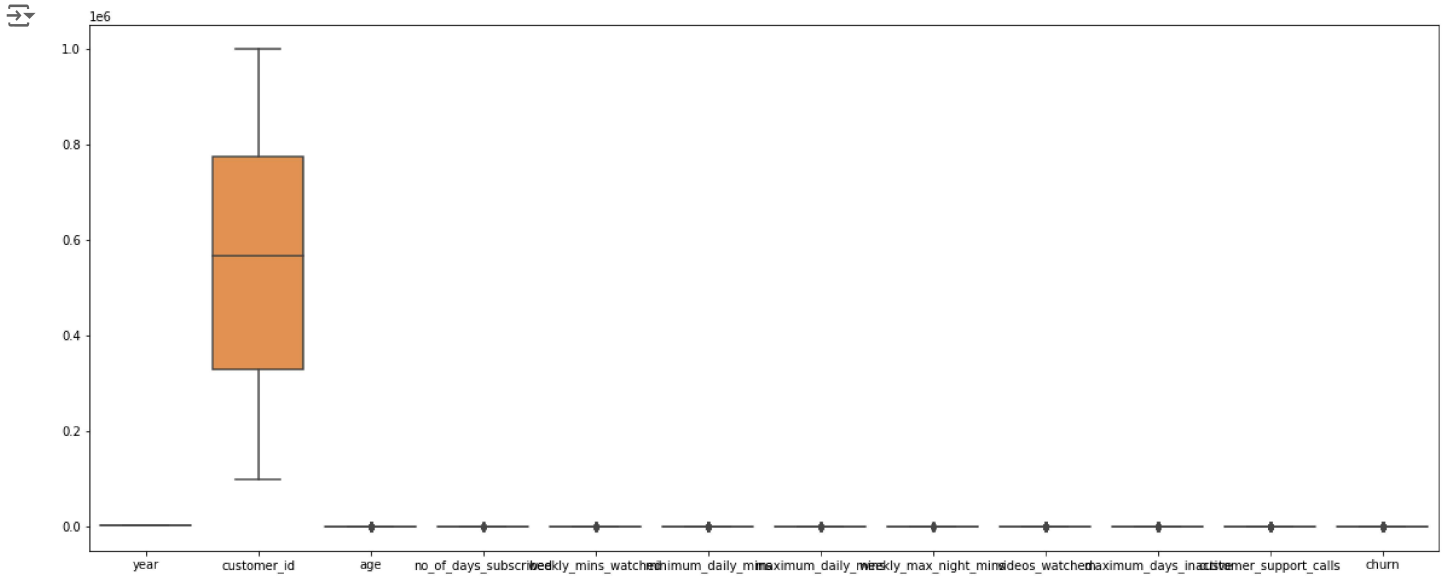
```
data['phone_no'].value_counts()
```

```
409-8743    1
419-5505    1
418-9385    1
347-1914    1
360-6309    1
           ..
330-8142    1
357-5801    1
420-5990    1
390-2891    1
414-1496    1
Name: phone_no, Length: 2000, dtype: int64
```

```
#FEATURE ENGINEERING
```

```
#OUTLIERS DETECTION
```

```
plt.figure(figsize=(20,8))
sns.boxplot(data=data,width=0.8)
plt.show()
```




```
#HANDLING MISSING VALUES
```

```
missing_list=list(data.isnull().sum()[data.isnull().sum()>0].index)
```

```
missing_list
```

```
['gender', 'maximum_days_inactive', 'churn']
```

```
#let's replace gender with mode
mode=data['gender'].mode()[0]
data['gender'].fillna(mode,inplace=True)

#let's replace maximum_days_inactive with median
median=data['maximum_days_inactive'].median()
data['maximum_days_inactive'].fillna(median,inplace=True)

#let's replace churn with mode
mode_2=data['churn'].mode()[0]
data['churn'].fillna(mode_2,inplace=True)
```

```
#Dropping the column phone_no
data.drop(['phone_no'],axis=1,inplace=True)
```

```
data.isnull().sum()
```

```
year                      0
customer_id               0
gender                    0
age                       0
no_of_days_subscribed     0
multi_screen              0
mail_subscribed           0
weekly_mins_watched       0
minimum_daily_mins        0
maximum_daily_mins        0
weekly_max_night_mins     0
videos_watched            0
maximum_days_inactive     0
customer_support_calls    0
churn                     0
dtype: int64
```

```
list_cat_cols=['gender','multi_screen','mail_subscribed']
x=data.drop(['churn'],axis=1)
y=data['churn']
```

```
x.shape,y.shape
```

```
((2000, 14), (2000,))
```

```
#train test split
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,stratify=y)
# i want to encode now
from sklearn.preprocessing import LabelEncoder
for col in list_cat_cols:
  le=LabelEncoder()
  x_train[col]=le.fit_transform(x_train[col])
  x_test[col]=le.transform(x_test[col])
```

```
x_train.isnull().sum()
```

```
year                      0
customer_id               0
gender                    0
age                       0
no_of_days_subscribed     0
multi_screen              0
mail_subscribed           0
weekly_mins_watched       0
minimum_daily_mins        0
maximum_daily_mins        0
weekly_max_night_mins     0
videos_watched            0
maximum_days_inactive     0
customer_support_calls    0
dtype: int64
```

```
x_test.isnull().sum()
```

```
year                      0
customer_id               0
gender                    0
age                       0
no_of_days_subscribed     0
multi_screen              0
mail_subscribed           0
weekly_mins_watched       0
minimum_daily_mins        0
maximum_daily_mins        0
weekly_max_night_mins     0
videos_watched            0
maximum_days_inactive     0
customer_support_calls    0
dtype: int64
```

```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1600 entries, 1851 to 1849
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   year                    1600 non-null   int64
 1   customer_id             1600 non-null   int64
 2   gender                  1600 non-null   int64
 3   age                     1600 non-null   int64
 4   no_of_days_subscribed   1600 non-null   int64
 5   multi_screen            1600 non-null   int64
 6   mail_subscribed         1600 non-null   int64
 7   weekly_mins_watched     1600 non-null   float64
 8   minimum_daily_mins      1600 non-null   float64
```

```
 9   maximum_daily_mins      1600 non-null   float64
 10  weekly_max_night_mins   1600 non-null   int64
 11  videos_watched          1600 non-null   int64
 12  maximum_days_inactive   1600 non-null   float64
 13  customer_support_calls  1600 non-null   int64
dtypes: float64(4), int64(10)
memory usage: 187.5 KB
```

```
x_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 400 entries, 645 to 734
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   year                    400 non-null    int64
 1   customer_id             400 non-null    int64
 2   gender                  400 non-null    int64
 3   age                     400 non-null    int64
 4   no_of_days_subscribed   400 non-null    int64
 5   multi_screen            400 non-null    int64
 6   mail_subscribed         400 non-null    int64
 7   weekly_mins_watched     400 non-null    float64
 8   minimum_daily_mins      400 non-null    float64
 9   maximum_daily_mins      400 non-null    float64
 10  weekly_max_night_mins   400 non-null    int64
 11  videos_watched          400 non-null    int64
 12  maximum_days_inactive   400 non-null    float64
 13  customer_support_calls  400 non-null    int64
dtypes: float64(4), int64(10)
memory usage: 46.9 KB
```

```
#MODEL BUILDING
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
```

```
ss=StandardScaler()
x_train_scaled=ss.fit_transform(x_train)
x_test_scaled=ss.transform(x_test)
```

```
log=LogisticRegression()
log.fit(x_train,y_train)
pred=log.predict(x_test)
accuracy=accuracy_score(y_test,pred)
print(accuracy)
```

```
0.87
```

```
svc=SVC()
svc.fit(x_train,y_train)
pred=svc.predict(x_test)
```

```
accuracy=accuracy_score(y_test,pred)
print(accuracy)
```

```
0.87
```

```
confusion_matrix(y_test,pred)
```

```
array([[348,   0],
       [ 52,   0]])
```

```
precision_score(y_test,pred,average='binary')
```

```
0.0
```

```
recall_score(y_test,pred,average='binary')
```

```
0.0
```

```
from sklearn.model_selection import cross_validate

cv=cross_validate(svc,x_train,y_train,scoring='accuracy',cv=5)
cv['test_score']
```

```
array([0.86875, 0.86875, 0.86875, 0.86875, 0.86875])
```

```
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)

train_pred=dt.predict(x_train)
pred=dt.predict(x_test)

train_acc=accuracy_score(y_train,train_pred)
acc=accuracy_score(y_test,pred)
```

```
train_acc,acc
```

```
(1.0, 0.895)
```

```
cv=cross_validate(dt,x_train,y_train,scoring='accuracy',cv=5)
cv['test_score']
```

```
array([0.86875 , 0.878125, 0.890625, 0.871875, 0.865625])
```

```
confusion_matrix(y_train,train_pred)
```

```
array([[1390,    0],
       [   0,  210]])
```

```
confusion_matrix(y_test,pred)
```

```
array([[326,  22],
       [ 20,  32]])
```

```
#my model is overfitting
#i will choose hyper parameter tuning with RandomSearchCV
```

```
from sklearn.model_selection import RandomizedSearchCV
params={'max_depth':range(2,25),
        'min_samples_split':(2,15),
        'criterion':['gini','entropy']
        }

tree_class=DecisionTreeClassifier(random_state=1024)
clf=RandomizedSearchCV(tree_class,params,random_state=0,n_iter=200,scoring='accuracy',cv=4,n_jobs=-1)
clf.fit(x_train,y_train)
```

```
RandomizedSearchCV(cv=4, estimator=DecisionTreeClassifier(random_state=1024),
                   n_iter=200, n_jobs=-1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': range(2, 25),
                                        'min_samples_split': (2, 15)},
                   random_state=0, scoring='accuracy')
```

```
clf.best_params_
```

```
{'min_samples_split': 2, 'max_depth': 5, 'criterion': 'entropy'}
```

```
dtc=DecisionTreeClassifier(random_state=1024,max_depth=5,min_samples_split=2,criterion='entropy')
```

```
dtc.fit(x_train,y_train)

train_pred=dtc.predict(x_train)
```

```
pred=dtc.predict(x_test)

train_acc=accuracy_score(y_train,train_pred)
acc=accuracy_score(y_test,pred)

train_acc,acc
```

```
(0.934375, 0.915)
```

```
cv=cross_validate(dtc,x_train,y_train,scoring='accuracy',cv=5)
cv['test_score']
```

```
array([0.89375 , 0.890625, 0.91875 , 0.921875, 0.909375])
```

```
confusion_matrix(y_test,pred)
```

```
array([[339,   9],
       [ 25,  27]])
```

```
#now my model is not much overfitting compared to untune decisiontree
```

```
precision_score(y_test,pred,average='binary')
```

```
0.75
```

```
recall_score(y_test,pred,average='binary')
```

```
0.5192307692307693
```