In [1]:
```
#import liabraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Lets read the data and create a df
df = pd.read_csv("netflix_customer_churn.csv")
df.head()
```

Out[2]:

	customer_id	age	gender	subscription_type	watch
0	a9b75100-82a8-427a-a208-72f24052884a	51	Other	Basic	
1	49a5dfd9-7e69-4022-a6ad-0a1b9767fb5b	47	Other	Standard	
2	4d71f6ce-fca9-4ff7-8afa-197ac24de14b	27	Female	Standard	
3	d3c72c38-631b-4f9e-8a0e-de103cad1a7d	53	Other	Premium	
4	4e265c34-103a-4dbb-9553-76c9aa47e946	56	Other	Standard	

In [3]:
```
#data frame information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   customer_id             5000 non-null   object 
 1   age                     5000 non-null   int64  
 2   gender                  5000 non-null   object 
 3   subscription_type       5000 non-null   object 
 4   watch_hours             5000 non-null   float64
 5   last_login_days         5000 non-null   int64  
 6   region                  5000 non-null   object 
 7   device                  5000 non-null   object 
 8   monthly_fee             5000 non-null   float64
 9   churned                 5000 non-null   int64  
 10  payment_method          5000 non-null   object 
 11  number_of_profiles      5000 non-null   int64  
 12  avg_watch_time_per_day  5000 non-null   float64
 13  favorite_genre          5000 non-null   object 
dtypes: float64(3), int64(4), object(7)
memory usage: 547.0+ KB
```

In [4]:
```
#check contents of df col
#df['monthly_fee'].tolist()
df['monthly_fee'].tolist()
```

Out[4]:
```
[8.99,
 13.99,
 13.99,
 17.99,
 13.99,
 13.99,
 8.99,
 8.99,
 8.99,
 17.99,
 17.99,
 13.99,
 13.99,
 8.99,
 8.99,
 17.99,
 17.99,
 13.99,
 17.99,
 13.99,
 8.99,
 17.99,
 13.99,
 13.99,
 8.99,
 13.99,
 17.99,
 17.99,
 8.99,
 13.99,
 13.99,
 13.99,
 17.99,
 8.99,
 17.99,
 13.99,
 13.99,
 13.99,
 17.99,
 13.99,
 17.99,
 17.99,
 17.99,
 8.99,
 17.99,
 13.99,
 8.99,
 13.99,
 17.99,
 8.99,
 17.99,
 13.99,
 13.99,
 13.99,
```

8.99,
17.99,
17.99,
8.99,
13.99,
17.99,
13.99,
17.99,
17.99,
13.99,
8.99,
13.99,
13.99,
13.99,
13.99,
17.99,
13.99,
13.99,
13.99,
17.99,
17.99,
8.99,
17.99,
8.99,
17.99,
13.99,
13.99,
17.99,
13.99,
17.99,
8.99,
17.99,
13.99,
13.99,
8.99,
13.99,
17.99,
17.99,
17.99,
17.99,
17.99,
17.99,
17.99,
8.99,
17.99,
13.99,
17.99,
17.99,
13.99,
8.99,
17.99,
17.99,
8.99,
8.99,

8.99,
13.99,
17.99,
17.99,
8.99,
8.99,
8.99,
17.99,
13.99,
8.99,
17.99,
8.99,
8.99,
8.99,
13.99,
13.99,
8.99,
8.99,
13.99,
17.99,
17.99,
17.99,
8.99,
8.99,
8.99,
17.99,
8.99,
8.99,
17.99,
8.99,
8.99,
13.99,
13.99,
13.99,
17.99,
13.99,
8.99,
13.99,
8.99,
8.99,
8.99,
17.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
17.99,
13.99,
13.99,
17.99,
8.99,
17.99,

13.99,
8.99,
8.99,
8.99,
13.99,
17.99,
17.99,
8.99,
8.99,
17.99,
17.99,
8.99,
17.99,
8.99,
8.99,
13.99,
8.99,
13.99,
8.99,
17.99,
13.99,
13.99,
13.99,
13.99,
8.99,
13.99,
13.99,
17.99,
17.99,
17.99,
13.99,
8.99,
13.99,
17.99,
8.99,
13.99,
8.99,
17.99,
8.99,
17.99,
8.99,
17.99,
8.99,
17.99,
17.99,
13.99,
17.99,
13.99,
8.99,
8.99,
8.99,
13.99,
13.99,
8.99,

13.99,
13.99,
8.99,
17.99,
17.99,
13.99,
13.99,
17.99,
13.99,
17.99,
17.99,
13.99,
17.99,
8.99,
13.99,
13.99,
13.99,
8.99,
17.99,
8.99,
17.99,
17.99,
8.99,
8.99,
17.99,
13.99,
8.99,
8.99,
17.99,
8.99,
17.99,
13.99,
17.99,
13.99,
17.99,
13.99,
8.99,
8.99,
17.99,
17.99,
8.99,
17.99,
17.99,
8.99,
8.99,
13.99,
8.99,
13.99,
8.99,
8.99,
8.99,
17.99,
17.99,
8.99,

8.99,
13.99,
13.99,
13.99,
13.99,
17.99,
17.99,
13.99,
17.99,
13.99,
17.99,
17.99,
13.99,
13.99,
17.99,
8.99,
17.99,
17.99,
13.99,
17.99,
17.99,
13.99,
8.99,
8.99,
8.99,
13.99,
8.99,
17.99,
13.99,
17.99,
17.99,
8.99,
8.99,
8.99,
13.99,
8.99,
8.99,
17.99,
17.99,
17.99,
8.99,
13.99,
8.99,
8.99,
8.99,
13.99,
17.99,
17.99,
8.99,
17.99,
8.99,
13.99,
8.99,
17.99,

8.99,
13.99,
8.99,
8.99,
17.99,
17.99,
8.99,
17.99,
13.99,
8.99,
13.99,
17.99,
13.99,
17.99,
13.99,
17.99,
8.99,
13.99,
13.99,
8.99,
17.99,
17.99,
13.99,
13.99,
13.99,
13.99,
13.99,
8.99,
13.99,
13.99,
8.99,
13.99,
17.99,
13.99,
13.99,
13.99,
8.99,
13.99,
13.99,
17.99,
17.99,
17.99,
13.99,
13.99,
13.99,
13.99,
13.99,
13.99,
8.99,
17.99,
17.99,
8.99,
8.99,
13.99,

8.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
8.99,
17.99,
17.99,
8.99,
17.99,
8.99,
17.99,
8.99,
13.99,
17.99,
8.99,
17.99,
13.99,
13.99,
13.99,
17.99,
13.99,
13.99,
17.99,
8.99,
17.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
17.99,
8.99,
8.99,
17.99,
17.99,
17.99,
17.99,
8.99,
13.99,
13.99,
17.99,
17.99,
8.99,
13.99,
17.99,
13.99,
8.99,
17.99,
8.99,

13.99,
8.99,
17.99,
8.99,
8.99,
13.99,
8.99,
17.99,
13.99,
8.99,
13.99,
17.99,
13.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
17.99,
8.99,
17.99,
13.99,
13.99,
17.99,
13.99,
8.99,
17.99,
13.99,
13.99,
13.99,
13.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
13.99,
8.99,
8.99,
8.99,
17.99,
13.99,
8.99,
8.99,
8.99,
13.99,
8.99,
17.99,
17.99,
8.99,
17.99,
8.99,

8.99,
17.99,
8.99,
17.99,
8.99,
17.99,
17.99,
17.99,
17.99,
17.99,
8.99,
13.99,
13.99,
13.99,
8.99,
17.99,
17.99,
13.99,
13.99,
13.99,
8.99,
8.99,
17.99,
13.99,
8.99,
13.99,
13.99,
13.99,
13.99,
17.99,
17.99,
17.99,
13.99,
8.99,
17.99,
13.99,
13.99,
8.99,
8.99,
17.99,
17.99,
13.99,
17.99,
17.99,
13.99,
17.99,
13.99,
13.99,
8.99,
8.99,
13.99,
13.99,
13.99,
13.99,

8.99,
17.99,
8.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
13.99,
13.99,
13.99,
8.99,
13.99,
13.99,
13.99,
17.99,
17.99,
13.99,
17.99,
17.99,
8.99,
17.99,
17.99,
13.99,
13.99,
8.99,
13.99,
8.99,
13.99,
8.99,
17.99,
13.99,
17.99,
17.99,
13.99,
17.99,
8.99,
8.99,
17.99,
17.99,
13.99,
8.99,
13.99,
8.99,
17.99,
8.99,
8.99,
13.99,
8.99,
17.99,
8.99,
17.99,

17.99,
17.99,
8.99,
8.99,
8.99,
8.99,
13.99,
13.99,
17.99,
13.99,
8.99,
13.99,
8.99,
13.99,
17.99,
17.99,
8.99,
8.99,
13.99,
13.99,
13.99,
13.99,
17.99,
17.99,
8.99,
8.99,
13.99,
17.99,
13.99,
8.99,
13.99,
13.99,
17.99,
13.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
13.99,
13.99,
17.99,
13.99,
13.99,
13.99,
13.99,
13.99,
17.99,
13.99,
17.99,
8.99,
17.99,

8.99,
13.99,
8.99,
13.99,
8.99,
17.99,
8.99,
8.99,
17.99,
8.99,
17.99,
17.99,
17.99,
13.99,
13.99,
8.99,
13.99,
8.99,
17.99,
13.99,
17.99,
17.99,
17.99,
13.99,
13.99,
13.99,
13.99,
8.99,
17.99,
8.99,
17.99,
8.99,
17.99,
8.99,
8.99,
17.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
17.99,
17.99,
17.99,
13.99,
17.99,
13.99,
17.99,
13.99,
13.99,
13.99,
8.99,
13.99,

17.99,
8.99,
17.99,
8.99,
13.99,
17.99,
13.99,
8.99,
8.99,
13.99,
13.99,
8.99,
8.99,
13.99,
17.99,
17.99,
17.99,
8.99,
17.99,
13.99,
13.99,
8.99,
17.99,
17.99,
13.99,
17.99,
13.99,
17.99,
8.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
8.99,
17.99,
8.99,
13.99,
17.99,
17.99,
13.99,
13.99,
8.99,
8.99,
8.99,
17.99,
13.99,
8.99,
17.99,
13.99,
17.99,
17.99,
13.99,

8.99,
13.99,
8.99,
8.99,
13.99,
8.99,
8.99,
8.99,
17.99,
17.99,
8.99,
13.99,
8.99,
8.99,
17.99,
13.99,
13.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
8.99,
13.99,
8.99,
8.99,
17.99,
8.99,
13.99,
8.99,
17.99,
17.99,
13.99,
13.99,
13.99,
8.99,
8.99,
8.99,
17.99,
8.99,
13.99,
13.99,
17.99,
8.99,
8.99,
13.99,
13.99,
13.99,
8.99,
13.99,
17.99,
17.99,
17.99,

17.99,
13.99,
8.99,
8.99,
17.99,
8.99,
17.99,
8.99,
17.99,
13.99,
8.99,
8.99,
13.99,
13.99,
13.99,
8.99,
13.99,
8.99,
13.99,
17.99,
13.99,
13.99,
13.99,
13.99,
17.99,
13.99,
17.99,
13.99,
13.99,
17.99,
8.99,
8.99,
13.99,
8.99,
8.99,
17.99,
13.99,
17.99,
8.99,
8.99,
8.99,
13.99,
17.99,
13.99,
13.99,
13.99,
13.99,
17.99,
8.99,
8.99,
17.99,
17.99,
13.99,
8.99,

17.99,
17.99,
17.99,
8.99,
8.99,
8.99,
13.99,
8.99,
13.99,
13.99,
8.99,
17.99,
17.99,
17.99,
8.99,
13.99,
13.99,
8.99,
13.99,
8.99,
8.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
13.99,
17.99,
17.99,
8.99,
17.99,
13.99,
8.99,
17.99,
17.99,
17.99,
17.99,
8.99,
8.99,
13.99,
17.99,
17.99,
8.99,
8.99,
17.99,
13.99,
8.99,
13.99,
8.99,
8.99,
13.99,
8.99,
17.99,

8.99,
17.99,
13.99,
13.99,
8.99,
13.99,
17.99,
17.99,
8.99,
8.99,
8.99,
8.99,
17.99,
13.99,
17.99,
8.99,
17.99,
13.99,
13.99,
8.99,
17.99,
13.99,
8.99,
8.99,
13.99,
17.99,
8.99,
17.99,
13.99,
13.99,
8.99,
13.99,
13.99,
17.99,
13.99,
13.99,
13.99,
13.99,
8.99,
13.99,
8.99,
13.99,
17.99,
13.99,
17.99,
13.99,
17.99,
13.99,
17.99,
17.99,
13.99,
17.99,
8.99,
13.99,

```
 13.99,
 13.99,
 17.99,
 8.99,
 13.99,
 13.99,
 13.99,
 17.99,
 13.99,
 8.99,
 17.99,
 17.99,
 8.99,
 17.99,
 13.99,
 13.99,
 13.99,
 17.99,
 17.99,
 8.99,
 13.99,
 8.99,
 8.99,
 13.99,
 17.99,
 8.99,
 8.99,
 8.99,
 ...]
```

In [5]:

```
filter_res=df['monthly_fee']!=' '
df= df[filter_res]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   customer_id             5000 non-null   object 
 1   age                     5000 non-null   int64  
 2   gender                  5000 non-null   object 
 3   subscription_type       5000 non-null   object 
 4   watch_hours             5000 non-null   float64
 5   last_login_days         5000 non-null   int64  
 6   region                  5000 non-null   object 
 7   device                  5000 non-null   object 
 8   monthly_fee             5000 non-null   float64
 9   churned                 5000 non-null   int64  
 10  payment_method          5000 non-null   object 
 11  number_of_profiles      5000 non-null   int64  
 12  avg_watch_time_per_day  5000 non-null   float64
 13  favorite_genre          5000 non-null   object 
dtypes: float64(3), int64(4), object(7)
memory usage: 547.0+ KB
```

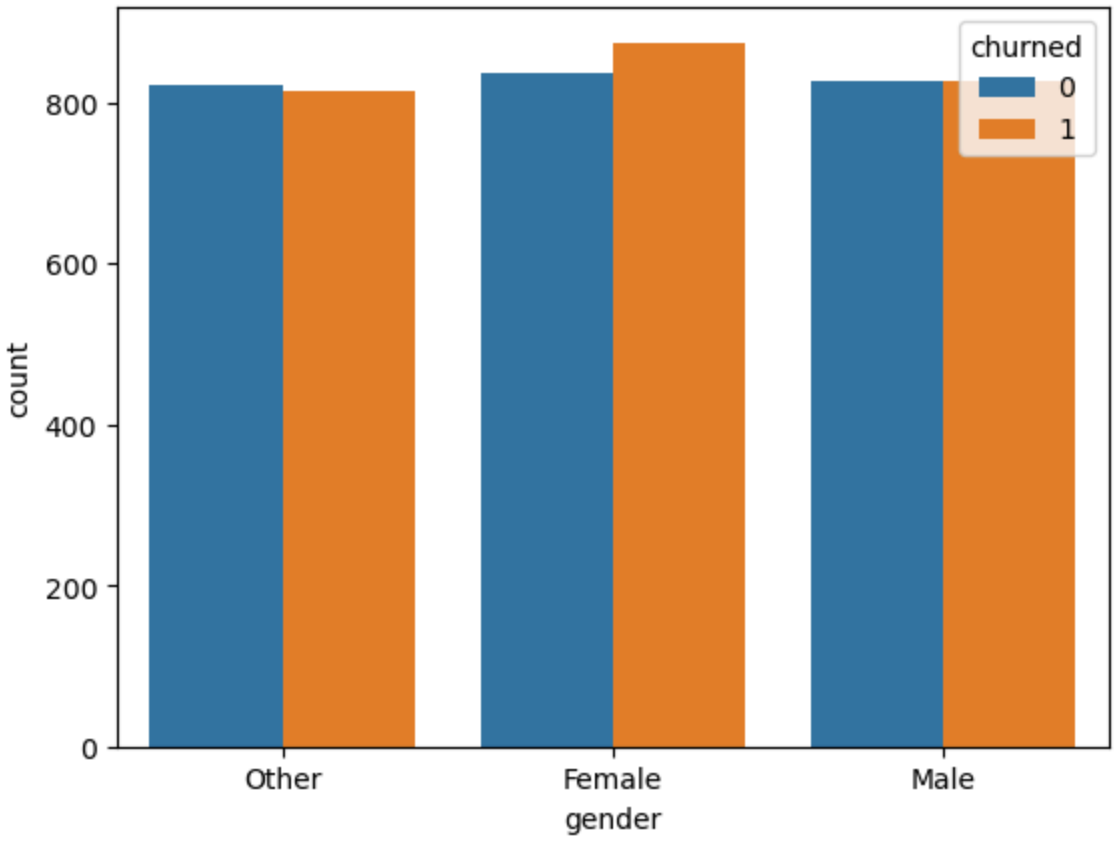
In [6]:
```
df.head(5)
```

Out[6]:

	customer_id	age	gender	subscription_type	watch
0	a9b75100-82a8-427a-a208-72f24052884a	51	Other	Basic	
1	49a5dfd9-7e69-4022-a6ad-0a1b9767fb5b	47	Other	Standard	
2	4d71f6ce-fca9-4ff7-8afa-197ac24de14b	27	Female	Standard	
3	d3c72c38-631b-4f9e-8a0e-de103cad1a7d	53	Other	Premium	
4	4e265c34-103a-4dbb-9553-76c9aa47e946	56	Other	Standard	

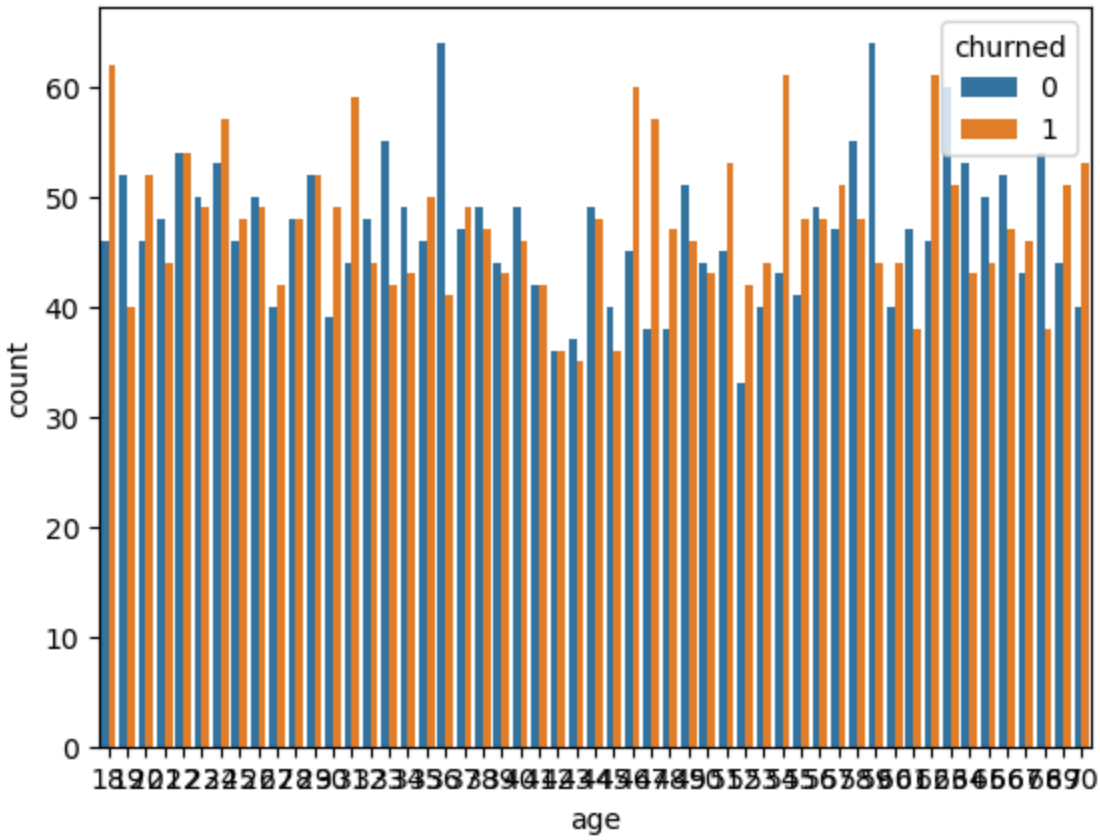
In [7]:
```
#general plotting
sns.countplot(data=df,x='gender',hue='churned')
```

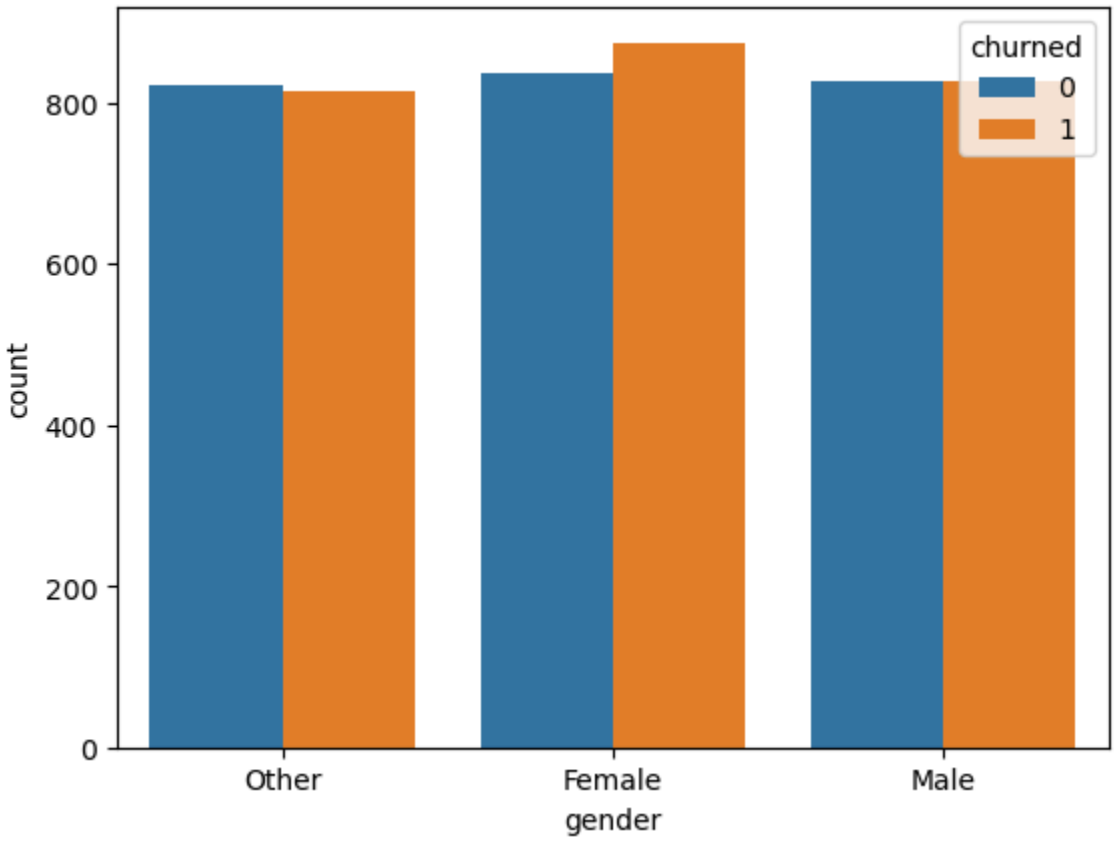
Out[7]:
```
<Axes: xlabel='gender', ylabel='count'>
```

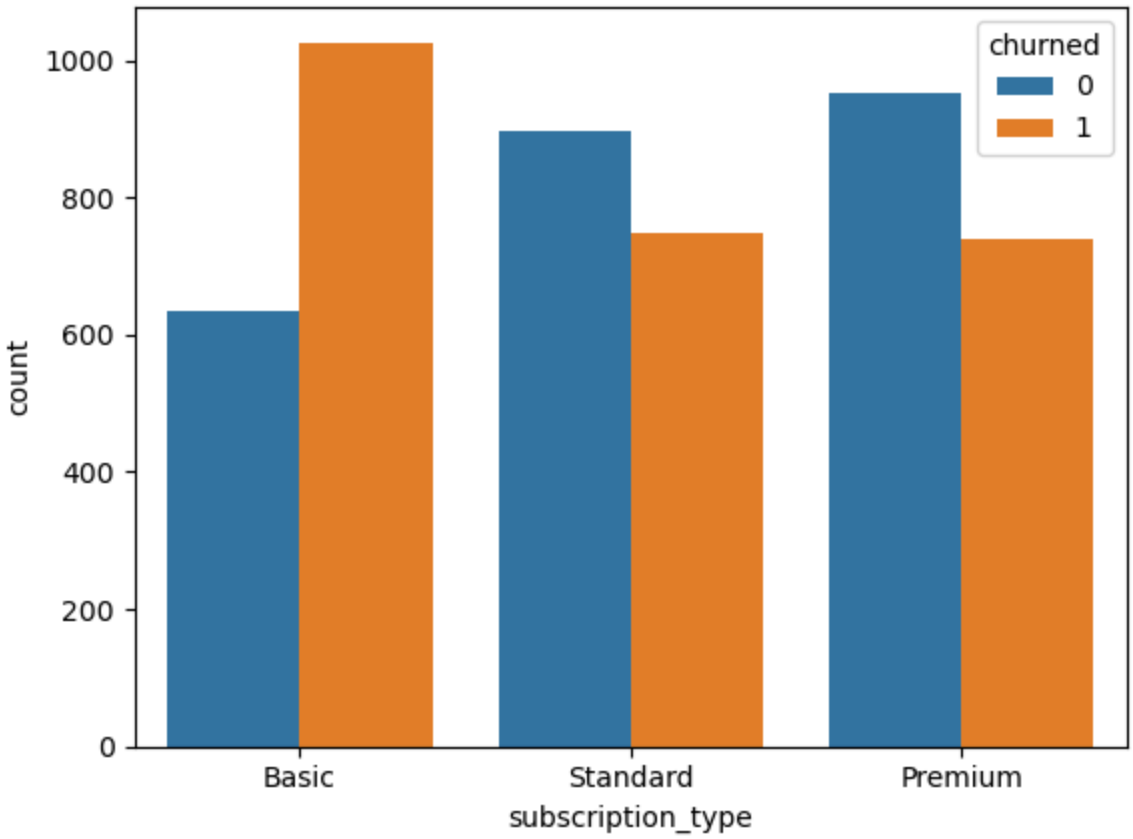


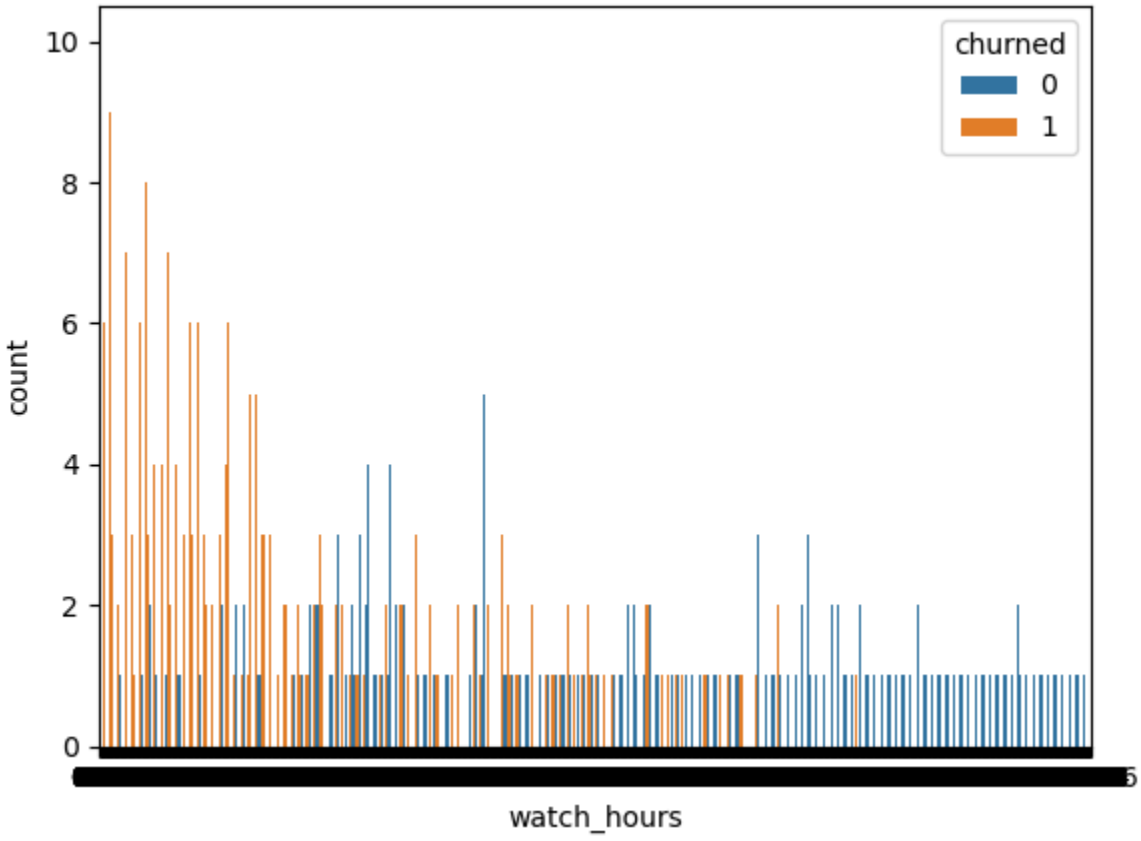
In [8]:
```
df_cols=list(df.columns)
df_cols_short=df_cols[1:5]
```

In [9]:
```
#plot using loops
for col in df_cols_short:
    sns.countplot(data=df,x=col,hue='churned')
    plt.show()
```


churned
0
1
count
age


churned
0
1
count
Other
Female
Male
gender





In [10]:
```
# checking if all columns have unique values
```

```
collist=df.columns.to_list()
for i in collist:
    print(df[i].unique())
```

```
['a9b75100-82a8-427a-a208-72f24052884a'
 '49a5dfd9-7e69-4022-a6ad-0a1b9767fb5b'
 '4d71f6ce-fca9-4ff7-8afa-197ac24de14b' ...
 '3f32e8c5-615b-4a3b-a864-db2688f7834f'
 '7b0ad82d-6571-430e-90f4-906259e0e89c'
 '82aeef39-ddb0-40ad-bae1-5c436e0cf042']
[51 47 27 53 56 58 48 45 32 26 28 49 39 46 68 21 60 70 36 55 24 30 69 67
 23 57 35 22 34 19 25 54 31 42 63 66 38 65 43 64 62 41 18 61 37 20 50 33
 59 29 52 40 44]
['Other' 'Female' 'Male']
['Basic' 'Standard' 'Premium']
[14.73  0.7  16.32 ... 49.17 16.55  9.12]
[29 19 10 12 13 26 20 56 34 36 38 23 33  7 32 40 27  4 57 43 18  6 55 53
 52  0 48 16 35 45  1 15 14 25 50 44 39 24 21 22 11 30 37 31 46 41  2  3
 51  9 54 17 49 42 28 59 58 60 47  5  8]
['Africa' 'Europe' 'Asia' 'Oceania' 'South America' 'North America']
['TV' 'Mobile' 'Laptop' 'Desktop' 'Tablet']
[ 8.99 13.99 17.99]
[1 0]
['Gift Card' 'Crypto' 'Debit Card' 'PayPal' 'Credit Card']
[1 5 2 3 4]
[4.900e-01 3.000e-02 1.480e+00 3.500e-01 1.300e-01 5.100e-01 6.600e-01
 2.500e-01 9.100e-01 6.000e-02 4.200e-01 5.700e-01 3.300e-01 1.500e-01
 1.000e+00 5.120e+00 9.000e-02 1.900e-01 5.500e-01 6.380e+00 1.400e-01
 9.800e-01 1.730e+00 1.290e+00 1.060e+00 2.000e-02 4.500e-01 3.000e-01
 2.700e-01 1.000e-01 3.800e-01 1.560e+00 4.400e-01 1.330e+00 2.800e-01
 2.170e+00 8.200e-01 8.000e-02 6.000e-01 1.240e+00 4.000e-01 6.300e-01
 2.140e+00 4.030e+00 5.200e-01 7.000e-02 2.200e-01 5.000e-02 8.360e+00
 1.220e+00 1.180e+00 3.350e+00 3.600e-01 1.100e+00 6.900e-01 6.090e+00
 2.100e-01 1.600e-01 1.800e-01 2.600e-01 8.300e-01 1.700e-01 3.900e-01
 1.160e+00 1.890e+00 5.800e-01 1.410e+00 4.100e-01 1.100e-01 7.800e-01
 8.700e-01 2.300e+00 3.700e-01 1.370e+00 2.990e+00 6.060e+00 9.300e-01
 7.500e-01 7.000e-01 2.900e-01 5.860e+00 4.000e-02 3.060e+00 5.400e-01
 3.100e-01 3.400e-01 1.200e-01 4.010e+00 7.200e-01 8.900e-01 5.590e+00
 9.700e-01 2.300e-01 9.200e-01 3.200e-01 1.630e+00 7.600e-01 1.830e+00
 2.180e+00 2.240e+00 4.600e-01 8.400e-01 2.400e+00 0.000e+00 1.044e+01
 2.000e-01 2.870e+00 1.770e+00 2.900e+00 3.460e+00 5.900e-01 1.000e-02
 2.120e+00 1.810e+00 7.880e+00 2.400e-01 7.700e-01 6.400e-01 2.500e+00
 5.490e+00 9.000e-01 3.680e+00 5.280e+00 4.300e-01 7.400e-01 9.400e-01
 1.092e+01 1.680e+00 5.000e-01 2.460e+00 2.060e+00 8.800e-01 1.570e+00
 1.650e+00 1.670e+00 1.470e+00 5.300e-01 8.500e-01 3.160e+00 1.440e+00
 1.450e+00 4.700e-01 8.000e-01 8.600e-01 1.260e+00 8.100e-01 6.620e+00
 1.070e+00 1.622e+01 3.210e+00 1.230e+00 2.090e+00 2.130e+00 1.390e+00
 1.840e+00 1.030e+00 2.121e+01 4.000e+00 4.800e-01 6.100e-01 6.700e-01
 7.080e+00 4.190e+00 9.600e-01 7.300e-01 7.900e-01 2.030e+00 6.800e-01
 2.291e+01 1.540e+00 4.290e+00 1.880e+00 8.400e+00 1.175e+01 8.390e+00
 3.880e+00 3.700e+00 3.090e+00 2.420e+00 1.710e+00 1.140e+00 2.710e+00
 9.900e-01 1.080e+00 2.190e+00 1.590e+00 1.640e+00 1.164e+01 1.190e+00
 2.360e+00 1.892e+01 1.270e+00 1.120e+00 1.500e+00 2.700e+00 2.660e+00
 4.120e+00 1.980e+00 1.820e+00 1.200e+00 6.110e+00 4.750e+00 1.090e+00
 1.900e+00 2.470e+00 2.515e+01 4.040e+00 4.220e+00 2.860e+00 1.461e+01
 6.460e+00 1.143e+01 6.500e-01 4.910e+00 6.200e-01 2.680e+00 3.280e+00
 1.750e+00 1.150e+00 2.350e+00 7.520e+00 2.880e+00 2.210e+00 2.340e+00
 5.420e+00 1.130e+00 1.720e+00 1.780e+00 1.960e+00 1.320e+00 4.240e+00
```

```
7.100e-01 2.020e+00 2.480e+00 5.600e-01 1.210e+00 4.320e+00 9.500e-01
2.000e+00 1.010e+00 1.610e+00 2.250e+00 1.520e+00 2.840e+00 3.130e+00
9.740e+00 3.390e+00 3.850e+00 1.020e+00 1.600e+00 1.189e+01 1.875e+01
6.520e+00 1.660e+00 4.420e+00 1.460e+00 1.350e+00 4.410e+00 1.850e+00
1.572e+01 1.061e+01 3.040e+00 4.550e+00 1.910e+00 3.070e+00 1.250e+00
3.420e+00 1.490e+00 1.040e+00 1.411e+01 5.640e+00 5.280e+01 5.260e+00
2.910e+00 1.510e+00 2.890e+00 2.040e+00 1.110e+00 2.590e+00 1.920e+00
4.180e+00 1.930e+00 3.470e+00 7.530e+00 1.400e+00 1.340e+00 2.800e+00
2.080e+00 3.360e+00 1.170e+00 5.050e+00 1.620e+00 2.280e+00 4.270e+00
4.360e+00 6.780e+00 3.410e+00 1.256e+01 7.480e+00 3.150e+00 1.790e+00
9.040e+00 1.050e+00 1.610e+01 3.990e+00 3.890e+00 4.760e+00 2.690e+00
2.100e+00 2.850e+00 1.700e+00 1.360e+00 4.170e+00 1.430e+00 7.840e+00
2.730e+00 1.058e+01 6.580e+00 1.860e+00 6.290e+00 1.380e+00 1.463e+01
1.580e+00 2.010e+00 9.370e+00 3.900e+00 5.770e+00 1.431e+01 2.960e+00
3.600e+00 2.740e+00 2.070e+00 6.420e+00 5.210e+00 2.380e+00 1.990e+00
2.320e+00 2.270e+00 1.870e+00 9.842e+01 9.810e+00 8.190e+00 6.280e+00
2.110e+00 8.970e+00 5.360e+00 3.110e+00 4.470e+00 5.880e+00 1.303e+01
1.008e+01 3.080e+00 1.530e+00 1.760e+00 5.000e+00 2.750e+00 1.259e+01
5.530e+00 2.056e+01 5.320e+00 3.750e+00 1.108e+01 3.100e+00 3.620e+00
6.500e+00 1.970e+00 5.470e+00 3.500e+00 1.950e+00 3.630e+00 3.320e+00
2.810e+00 2.650e+00 3.453e+01 1.054e+01 2.970e+00 2.310e+00 7.310e+00
9.260e+00 1.280e+00 1.550e+00 2.830e+00 4.210e+00 1.665e+01 2.260e+00
1.371e+01 3.570e+00 2.780e+00 7.450e+00 2.450e+00 4.460e+00 2.950e+00
1.690e+00 3.170e+00 2.580e+00 5.710e+00 8.180e+00 9.320e+00 2.770e+00
3.710e+00 2.560e+00 3.980e+00 9.190e+00 4.630e+00 6.370e+00 6.530e+00
3.310e+00 7.680e+00 4.340e+00 2.550e+00 1.300e+00 1.136e+01 4.140e+00
1.443e+01 1.740e+00 5.110e+00 2.640e+00 3.270e+00 4.680e+00 2.430e+00
2.290e+00 1.800e+00 1.378e+01 5.440e+00 1.420e+00 7.030e+00 1.704e+01
5.340e+00 2.790e+00 6.130e+00 2.720e+00 5.430e+00 3.000e+00 4.710e+00
8.050e+00 2.866e+01 2.172e+01 6.800e+00 2.220e+00 4.840e+00 3.790e+00
1.310e+00 4.740e+00 4.090e+00 3.190e+00 7.600e+00 1.436e+01 1.152e+01
5.070e+00 1.272e+01 3.610e+00 3.830e+00 3.250e+00 2.530e+00 4.810e+00
1.276e+01 3.920e+00 7.820e+00 3.281e+01 3.220e+00 9.050e+00 5.970e+00
7.180e+00 1.027e+01 1.160e+01 3.580e+00 3.030e+00 6.710e+00 2.050e+00
9.520e+00 1.167e+01 2.727e+01 5.990e+00 3.810e+00 4.100e+00 4.080e+00
8.710e+00 1.940e+00 3.510e+00 2.760e+00 6.440e+00 2.160e+00 4.310e+00
4.070e+00 1.552e+01 2.200e+00 1.185e+01 3.430e+00 1.799e+01 7.660e+00
3.020e+00 7.780e+00 3.480e+00 6.040e+00 5.460e+00 4.980e+00 3.288e+01
1.426e+01 2.955e+01 7.170e+00 7.760e+00 2.698e+01 1.201e+01 3.550e+00
3.120e+00]
['Action' 'Sci-Fi' 'Drama' 'Horror' 'Romance' 'Comedy' 'Documentary']
```

In [11]:
```
#Import model specific libraries
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.metrics import classification_report, confusion_matrix, Confusion
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [12]:
```
#set y
code ={"Yes" : 1, "No":0}
df["churned"] = df["churned"].map(code)
```

```
y = df["churned"]
```

In [13]:
```
# Set X by dropping ID and target column
x = df.drop(['customer_id', 'churned'], axis=1)

# Create dummy variables for categorical features
x = pd.get_dummies(x, drop_first=True, dtype=int)

x.head()
```

Out[13]:

	age	watch_hours	last_login_days	monthly_fee	number_of_profiles	avg_wat
0	51	14.73	29	8.99	1	
1	47	0.70	19	13.99	5	
2	27	16.32	10	13.99	2	
3	53	4.51	12	17.99	2	
4	56	1.89	13	13.99	2	

5 rows × 29 columns

In [14]:
```
#checking all the new columns (dummy focused)
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 29 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   age                          5000 non-null   int64  
 1   watch_hours                  5000 non-null   float64
 2   last_login_days              5000 non-null   int64  
 3   monthly_fee                  5000 non-null   float64
 4   number_of_profiles           5000 non-null   int64  
 5   avg_watch_time_per_day       5000 non-null   float64
 6   gender_Male                  5000 non-null   int32  
 7   gender_Other                 5000 non-null   int32  
 8   subscription_type_Premium    5000 non-null   int32  
 9   subscription_type_Standard   5000 non-null   int32  
 10  region_Asia                  5000 non-null   int32  
 11  region_Europe                5000 non-null   int32  
 12  region_North America         5000 non-null   int32  
 13  region_Oceania               5000 non-null   int32  
 14  region_South America         5000 non-null   int32  
 15  device_Laptop                5000 non-null   int32  
 16  device_Mobile                5000 non-null   int32  
 17  device_TV                    5000 non-null   int32  
 18  device_Tablet                5000 non-null   int32  
 19  payment_method_Crypto        5000 non-null   int32  
 20  payment_method_Debit Card    5000 non-null   int32  
 21  payment_method_Gift Card     5000 non-null   int32  
 22  payment_method_PayPal        5000 non-null   int32  
 23  favorite_genre_Comedy        5000 non-null   int32  
 24  favorite_genre_Documentary   5000 non-null   int32  
 25  favorite_genre_Drama         5000 non-null   int32  
 26  favorite_genre_Horror        5000 non-null   int32  
 27  favorite_genre_Romance       5000 non-null   int32  
 28  favorite_genre_Sci-Fi        5000 non-null   int32  
dtypes: float64(3), int32(23), int64(3)
memory usage: 683.7 KB
```

In [15]:
```
# Load dataset
df = pd.read_csv("netflix_customer_churn.csv")
```

In [16]:
```
# Drop rows with NaN values (you can also use fillna instead if you prefer)
df = df.dropna()
```

In [17]:
```
# Define features and target
x = df.drop("churned", axis=1)
y = df["churned"]
```

In [18]:
```
# Train test split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, rand
```

In [19]:
```
print(y_test)
```

```
1501    0
2586    1
2653    0
1055    1
705     1
       ..
4711    0
2313    1
3214    0
2732    0
1926    0
Name: churned, Length: 1000, dtype: int64
```

In [20]:
```
# Encode categorical columns (convert text to numbers)
for col in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

In [21]:
```
# Load dataset
df = pd.read_csv("netflix_customer_churn.csv")
```

In [22]:
```
# Drop NaN rows
df = df.dropna()
```

In [23]:
```
# Drop identifier columns (UUID, ID, names, etc.)
if "customerid" in df.columns:
    df = df.drop("customerid", axis=1)
```

In [24]:
```
# Encode categorical variables
for col in df.select_dtypes(include=['object']).columns:
    if col != "Churn":  # don't encode target
        le = LabelEncoder()
        df[col] = le.fit_transform(df[col])
```

In [25]:
```
# # Define features and target
x = df.drop("churned", axis=1)
y = df["churned"]
```

In [26]:
```
# Train-test split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, rand
```

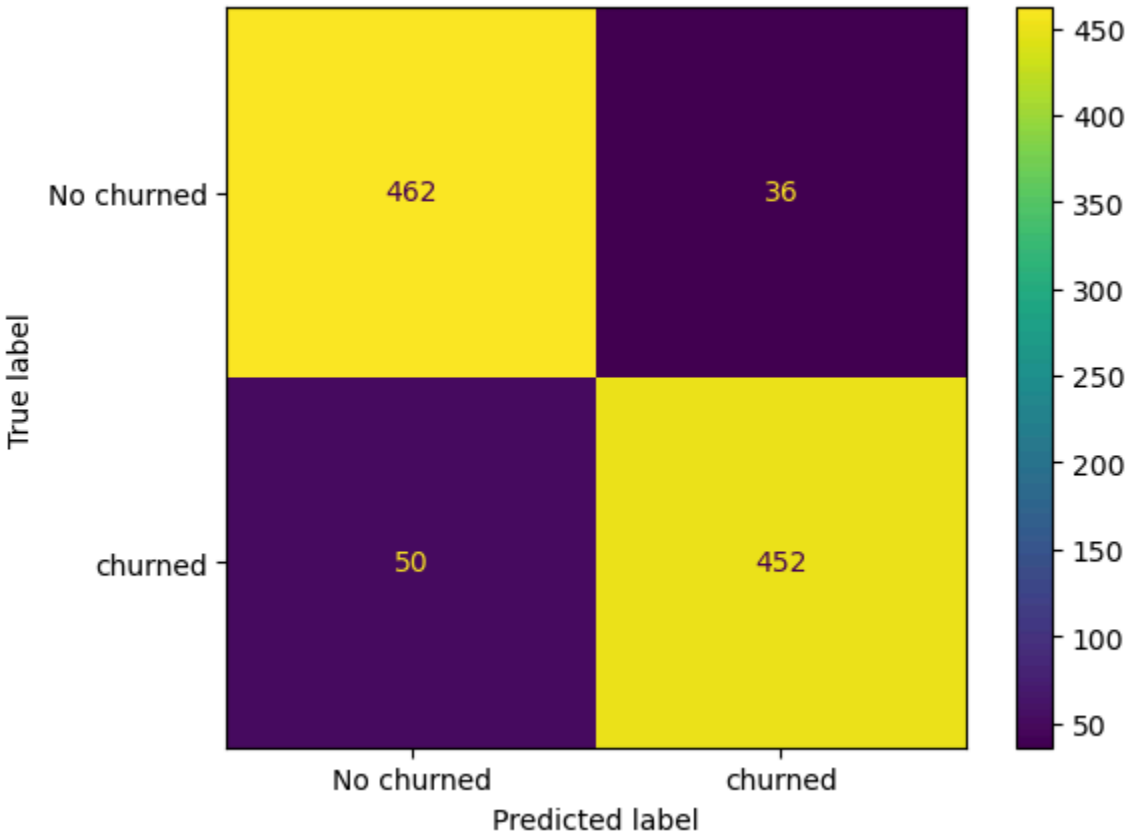
In [27]:
```
#seting up the model
mod1 = DecisionTreeClassifier(max_depth=5, random_state=42)
#run model
mod1.fit(x_train, y_train)
# run the predictions
pred1 = mod1.predict(x_test)
# #evaluate model performance by classification report
print(classification_report(y_test, pred1))
```

```
                precision    recall  f1-score   support

           0       0.90      0.93      0.91       498
           1       0.93      0.90      0.91       502

    accuracy                           0.91      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```

Precision - How many of the positives predicted are actually positive Recall - How many of the actual positives were predicted F1 Score - Harmonic Mean of the precision and recall scores Support - Number of true samples in that class(in this case, the actual number of customers who stayed) Accuracy - Overall percentage of correct predictions: True Posivites + True Negatives /Total Samples

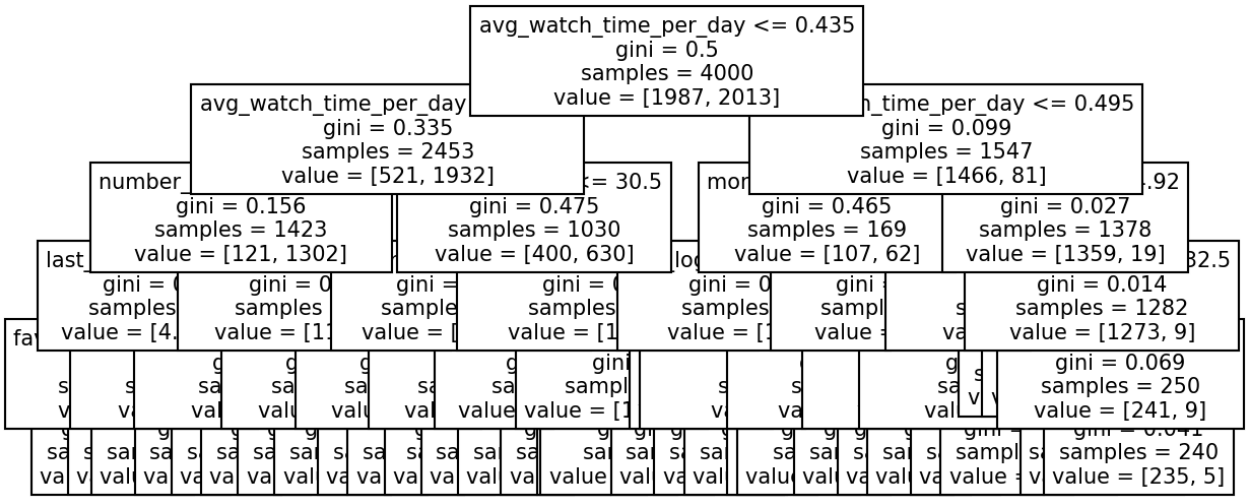
In [32]:
```
#confusion matrix
conf_mat= confusion_matrix(y_test, pred1)

#cm display
ConfusionMatrixDisplay(conf_mat, display_labels=['No churned', 'churned']).plo
```

Out[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x26f49d9d5
80>
```

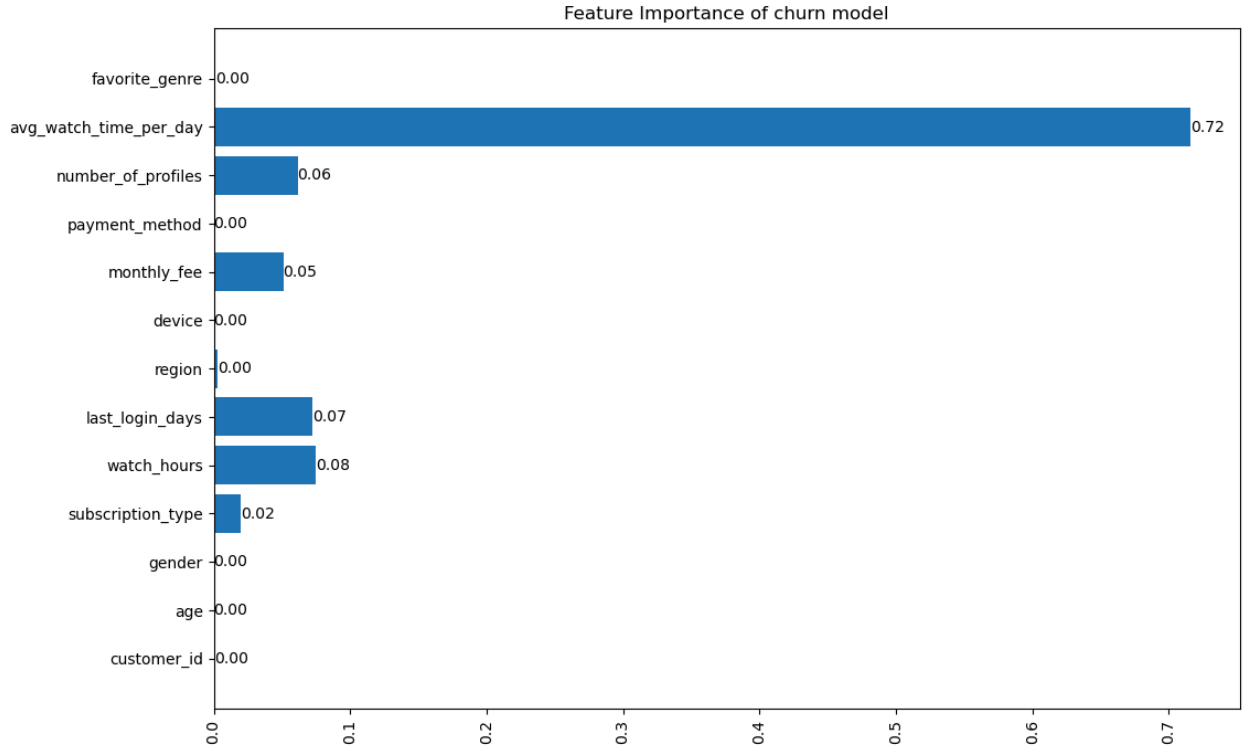


In [33]:
```
#plot the decision tree
```

```
#get column names from x
x_cols=x.columns
plt.figure(figsize=(9,4), dpi=150)
plot_tree(mod1, fontsize=10, feature_names=x_cols);
```



In [34]:
```
#Find which feature is the most important in determining churn
feat_imp= mod1.feature_importances_
plt.figure(figsize= (12,8))
bars = plt.barh(y=x_cols, width=feat_imp);
plt.bar_label(bars, fmt='%1.2f')
plt.xticks(rotation=90)
plt.title("Feature Importance of churn model")
plt.show()
```



In [35]:
```
#balancing the dataset
```

```
#step 1: Split the dataset into 2 classes:
#no churn df:
df_major=df[df['churned']==0]
#churn df:
df_minor = df[df["churned"]==1]
```

In [36]:
```
df_major.shape
```

Out[36]: (2485, 14)

In [37]:
```
df_minor.shape
```

Out[37]: (2515, 14)

In [38]:
```
#Creating down sampled majority data set
df_major_s= df_major.sample(n=len(df_minor),replace=True,random_state=20)
df_major_s.shape
```

Out[38]: (2515, 14)

In [39]:
```
#Merging downsampled majority with minority df
df_ds = pd.concat([df_minor, df_major_s])
df_ds
```

Out[39]:

	customer_id	age	gender	subscription_type	watch_hours	last_login_day
0	3314	51	2	0	14.73	2
1	1498	47	2	2	0.70	1
3	4137	53	2	1	4.51	1
4	1600	56	2	2	1.89	1
7	44	51	1	0	14.30	5
...	...	...	...	...	...	.
3946	4714	45	1	1	18.07	
4291	2769	32	1	1	28.61	2
2083	39	23	0	2	2.63	
2604	1722	47	2	0	5.53	1
725	1897	28	1	1	49.34	3

5030 rows × 14 columns

In [40]:
```
df_ds= df_ds.reset_index(drop=True)
```

In [41]:
```
#Shuffle the data
df_sf= df_ds.sample(frac=1, random_state=41)
df_sf
```

Out[41]:

	customer_id	age	gender	subscription_type	watch_hours	last_login_day
150	1575	52	2	1	11.50	5
4363	3793	46	1	1	27.63	2
1161	4553	20	0	0	1.27	4
544	677	40	2	0	0.86	
1996	3417	20	1	0	3.38	3
...	...	...	...	...	...	.
321	3686	52	0	1	6.28	4
4066	3386	62	1	2	7.31	2
3980	3300	55	1	2	22.74	
931	1999	31	0	0	3.77	1
1984	4194	34	0	1	1.67	2

5030 rows × 14 columns

In [42]:
```
x= df_sf.drop(["churned","customer_id"], axis=1)
x = pd.get_dummies(x, dtype='int')
y= df_sf["churned"]
```

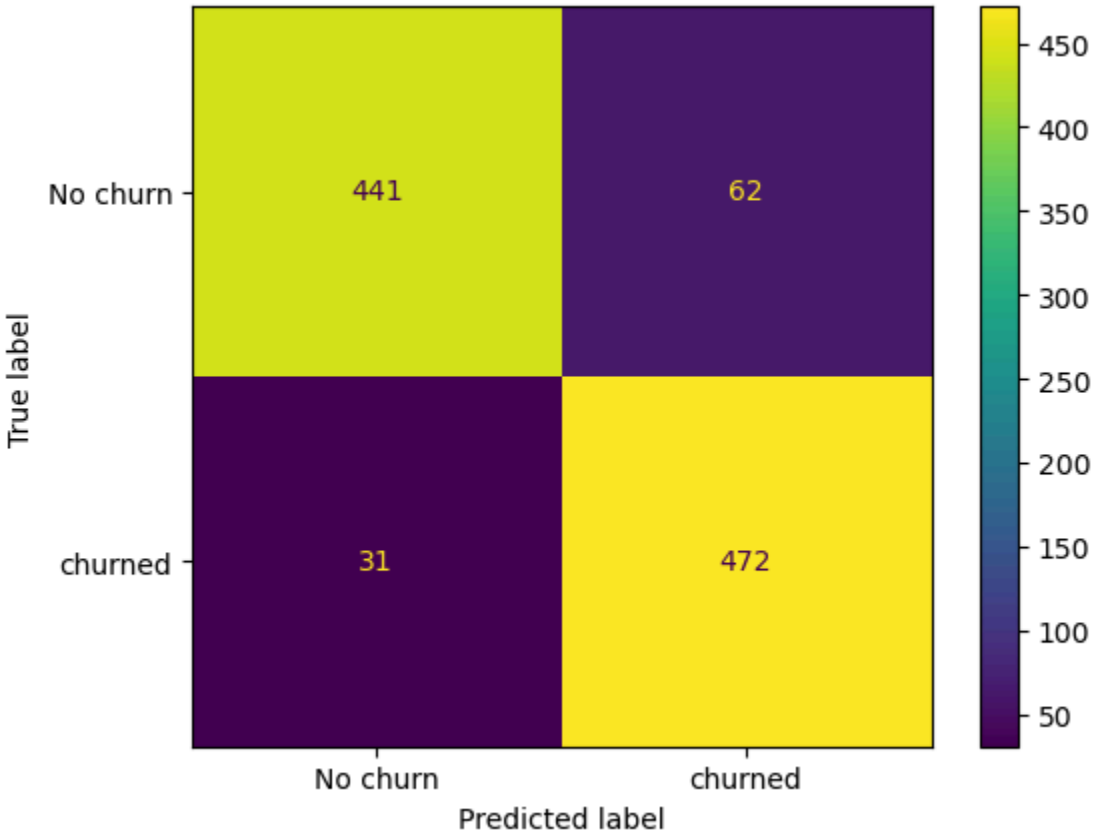
In [43]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= 0.20, ran
```

In [44]:
```
mod2 = DecisionTreeClassifier(max_depth=4)

#run model
mod2.fit(x_train, y_train)
#run predictions
pred2=mod2.predict(x_test)

#evaluate model performance by classification report
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.93      0.88      0.90       503
           1       0.88      0.94      0.91       503

    accuracy                           0.91      1006
   macro avg       0.91      0.91      0.91      1006
weighted avg       0.91      0.91      0.91      1006
```

In [45]:
```
#confusion matrix
conf_mat= confusion_matrix(y_test, pred2)

#cm display
```

```
ConfusionMatrixDisplay(conf_mat, display_labels=['No churn', 'churned']).plot(
```



In [46]:
```
#random forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
```

In [47]:
```
#Mod 3
mod3 = RandomForestClassifier(n_estimators=1000, max_depth=7)
mod3.fit(x_train, y_train)
pred_rfc=mod3.predict(x_test)
print(classification_report(y_test, pred_rfc))
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97       503
           1       0.98      0.97      0.97       503

    accuracy                           0.97      1006
   macro avg       0.97      0.97      0.97      1006
weighted avg       0.97      0.97      0.97      1006
```

K-Neighbours Classifier

In [49]:
```
#Mod4
mod4=KNeighborsClassifier()
mod4.fit(x_train, y_train)
```

```
pred_KN=mod4.predict(x_test)
print(classification_report(y_test,pred_KN))
```

```
              precision    recall  f1-score   support

           0       0.90      0.86      0.88       503
           1       0.87      0.90      0.88       503

    accuracy                           0.88      1006
   macro avg       0.88      0.88      0.88      1006
weighted avg       0.88      0.88      0.88      1006
```

Bayesian Model Tuning

In [51]:
```
pip install scikit-optimize
```

```
Requirement already satisfied: scikit-optimize in c:\users\krishna\anaconda3\li
b\site-packages (0.10.2)
Requirement already satisfied: joblib>=0.11 in c:\users\krishna\anaconda3\lib\s
ite-packages (from scikit-optimize) (1.4.2)
Requirement already satisfied: pyaml>=16.9 in c:\users\krishna\anaconda3\lib\si
te-packages (from scikit-optimize) (25.7.0)
Requirement already satisfied: numpy>=1.20.3 in c:\users\krishna\anaconda3\lib\
site-packages (from scikit-optimize) (1.26.4)
Requirement already satisfied: scipy>=1.1.0 in c:\users\krishna\anaconda3\lib\s
ite-packages (from scikit-optimize) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.0 in c:\users\krishna\anaconda
3\lib\site-packages (from scikit-optimize) (1.4.2)
Requirement already satisfied: packaging>=21.3 in c:\users\krishna\anaconda3\li
b\site-packages (from scikit-optimize) (23.2)
Requirement already satisfied: PyYAML in c:\users\krishna\anaconda3\lib\site-pa
ckages (from pyaml>=16.9->scikit-optimize) (6.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\krishna\anacond
a3\lib\site-packages (from scikit-learn>=1.0.0->scikit-optimize) (2.2.0)
Note: you may need to restart the kernel to use updated packages.
```

In [52]:
```
from skopt import BayesSearchCV
from skopt.space import Real, Categorical, Integer
```

In [53]:
```
#defining the search space
param_space = {
    'max_depth': Integer(1, 20), 'criterion':("gini","entropy","log_loss")}
```

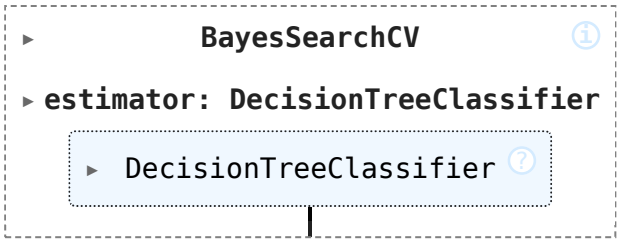
In [54]:
```
#Initialising the classifier
dt_classifier = DecisionTreeClassifier(random_state = 41)
```

In [55]:
```
#Setting the search space
opt = BayesSearchCV(
    dt_classifier,
    param_space,
    n_iter=50,
    cv=3,
```

```
    n_jobs=3
)
```

In [56]:
```
#Fitting the search to the data
opt.fit(x,y)
```

Out[56]:



In [57]:
```
#creating a variable with the best parameters
best_param=opt.best_params_
best_param
```

Out[57]:
```
OrderedDict([('criterion', 'entropy'), ('max_depth', 16)])
```

In [58]:
```
#creating a new model with the best parameters
mod1b = DecisionTreeClassifier(**best_param)
```

In [59]:
```
# fitting the new model with x_train and y_train
mod1b.fit(x_train, y_train)

# running the predictions
pred1b = mod1b.predict(x_test)

# printing the classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred1b))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       503
           1       0.99      1.00      0.99       503

    accuracy                           0.99      1006
   macro avg       0.99      0.99      0.99      1006
weighted avg       0.99      0.99      0.99      1006
```

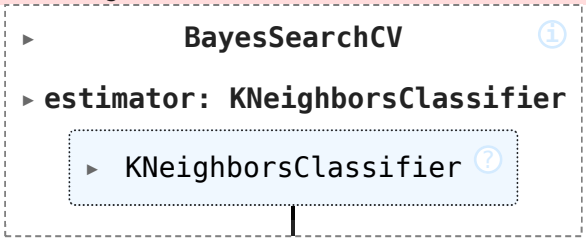
In [60]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
```

In [61]:
```
#setting and intialising the bayesian model tuning
param_space = {
'n_neighbors':(1, 50),
'weights':("uniform","distance"),
'metric':("minkowski","euclidean","manhattan","chebyshev","hamming")}
```

```
kn_classifier= KNeighborsClassifier()

opt3 = BayesSearchCV(
    kn_classifier,
    param_space,
    n_iter = 32,
    cv=3,
    n_jobs=3)

opt3.fit(x,y)
```

```
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['minkowski', 50, 'dista
nce'] before, using random point ['minkowski', 37, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['euclidean', 50, 'dista
nce'] before, using random point ['hamming', 30, 'distance']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['euclidean', 50, 'dista
nce'] before, using random point ['manhattan', 35, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['euclidean', 50, 'dista
nce'] before, using random point ['manhattan', 32, 'distance']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['manhattan', 50, 'dista
nce'] before, using random point ['hamming', 12, 'distance']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['manhattan', 50, 'dista
nce'] before, using random point ['euclidean', 18, 'distance']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['euclidean', 39, 'dista
nce'] before, using random point ['minkowski', 49, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['euclidean', 39, 'dista
nce'] before, using random point ['manhattan', 36, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['manhattan', 50, 'dista
nce'] before, using random point ['minkowski', 39, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['manhattan', 50, 'dista
nce'] before, using random point ['euclidean', 46, 'uniform']
  warnings.warn(
C:\Users\Krishna\anaconda3\Lib\site-packages\skopt\optimizer\optimizer.py:517:
UserWarning: The objective has been evaluated at point ['manhattan', 50, 'dista
nce'] before, using random point ['manhattan', 30, 'uniform']
  warnings.warn(
```

Out[61]:



In [62]:
```
best_param3 = opt3.best_params_
print(best_param3)
mod3b = KNeighborsClassifier(**best_param3)
mod3b.fit(x_train, y_train)
pred_knc_b = mod3b.predict(x_test)
print(classification_report(y_test, pred_knc_b))
```

```
OrderedDict({'metric': 'manhattan', 'n_neighbors': 31, 'weights': 'distance'})
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       503
           1       0.94      0.94      0.94       503

    accuracy                           0.94      1006
   macro avg       0.94      0.94      0.94      1006
weighted avg       0.94      0.94      0.94      1006
```

In [ ]: