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

import os
os.environ['KAGGLE_CONFIG_DIR']='/content'

!kaggle datasets download -d blastchar/telco-customer-churn
```

```
Dataset URL: https://www.kaggle.com/datasets/blastchar/telco-customer-
churn
License(s): copyright-authors
Downloading telco-customer-churn.zip to /content
  0% 0.00/172k [00:00<?, ?B/s]
100% 172k/172k [00:00<00:00, 35.3MB/s]
```

```
!unzip \*.zip  && rm *.zip
```

```
Archive:  telco-customer-churn.zip
  inflating: WA_Fn-UseC_-Telco-Customer-Churn.csv
```

```
data = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')
data.head()
```

{"type":"dataframe","variable_name":"data"}

```
data.shape
```

```
(7043, 21)
```

```
data.columns.values
data.dtypes
```

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
```

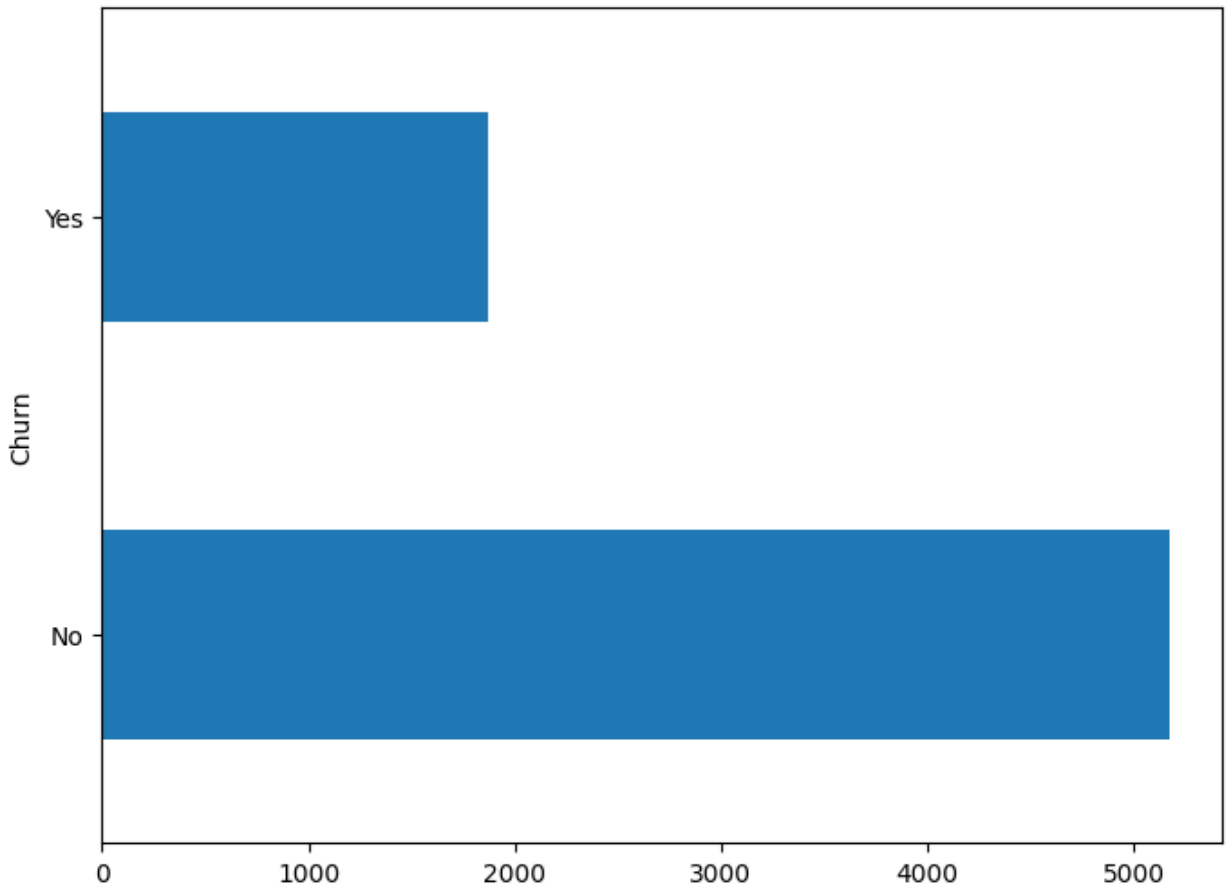
```
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

```
data.describe()
```


```
{"summary":"{\n  \"name\": \"data\",\n  \"rows\": 8,\n  \"fields\": [\
n    {\n      \"column\": \"SeniorCitizen\",\n      \"properties\": {\
n        \"dtype\": \"number\",\n        \"std\": 2489.9992387084,\n
\"min\": 0.0,\n        \"max\": 7043.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.1621468124378816,\n          1.0,\n          0.3686116056100131\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tenure\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
2478.9752758409018,\n        \"min\": 0.0,\n        \"max\": 7043.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
32.37114865824223,\n          29.0,\n          7043.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MonthlyCharges\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2468.7047672837775,\n        \"min\": 18.25,\n        \"max\":
7043.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
64.76169246059918,\n          70.35,\n          7043.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
data['Churn'].value_counts().plot(kind='barh', figsize=(8,6))
```

```
<Axes: ylabel='Churn'>
```




```
(data['Churn'].value_counts()/len(data['Churn']))*100
#imbalanced data

Churn
No     73.463013
Yes    26.536987
Name: count, dtype: float64

data.info(verbose=True)

<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
```

```
miss =
pd.DataFrame((data.isnull().sum())*100/data.shape[0]).reset_index()
print(miss)
#no missing data here
```

```
                index    0
0         customerID  0.0
1             gender  0.0
2      SeniorCitizen  0.0
3            Partner  0.0
4         Dependents  0.0
5             tenure  0.0
6       PhoneService  0.0
7      MultipleLines  0.0
8    InternetService  0.0
9     OnlineSecurity  0.0
10      OnlineBackup  0.0
11  DeviceProtection  0.0
12       TechSupport  0.0
13       StreamingTV  0.0
14   StreamingMovies  0.0
15          Contract  0.0
16  PaperlessBilling  0.0
17     PaymentMethod  0.0
18    MonthlyCharges  0.0
19      TotalCharges  0.0
20             Churn  0.0
```

```
#Data Cleaning
data_copy = data.copy()

data_copy.TotalCharges = pd.to_numeric(data_copy.TotalCharges,
errors='coerce')
data_copy.isnull().sum()
```

```
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         11
Churn                 0
dtype: int64
```

```
(data_copy.isnull().sum()/data_copy.shape[0])*100
```

```
customerID          0.000000
gender              0.000000
SeniorCitizen       0.000000
Partner             0.000000
Dependents          0.000000
tenure              0.000000
PhoneService        0.000000
MultipleLines       0.000000
InternetService     0.000000
OnlineSecurity      0.000000
OnlineBackup        0.000000
DeviceProtection    0.000000
TechSupport         0.000000
StreamingTV         0.000000
StreamingMovies     0.000000
Contract            0.000000
PaperlessBilling    0.000000
PaymentMethod       0.000000
MonthlyCharges      0.000000
TotalCharges        0.156183
Churn               0.000000
dtype: float64
```

```
data_copy.loc[data_copy['TotalCharges'].isnull() == True]
```

```
{"type":"dataframe"}
```

```
data_copy.dropna(how='any', inplace=True)
#now we have 11 less rows

#for tenure column create bins because so many years
data_copy['tenure'].max()
```

```
72
```

```
labels = ["{0} - {1}".format(i, i+11) for i in range(1, 72, 12)]
data_copy['tenure_group'] = pd.cut(data_copy.tenure, range(1, 80, 12),
right=False, labels=labels)
data_copy['tenure_group'].value_counts()
```

```
tenure_group
1 - 12     2175
61 - 72    1407
13 - 24    1024
25 - 36     832
49 - 60     832
37 - 48     762
Name: count, dtype: int64
```

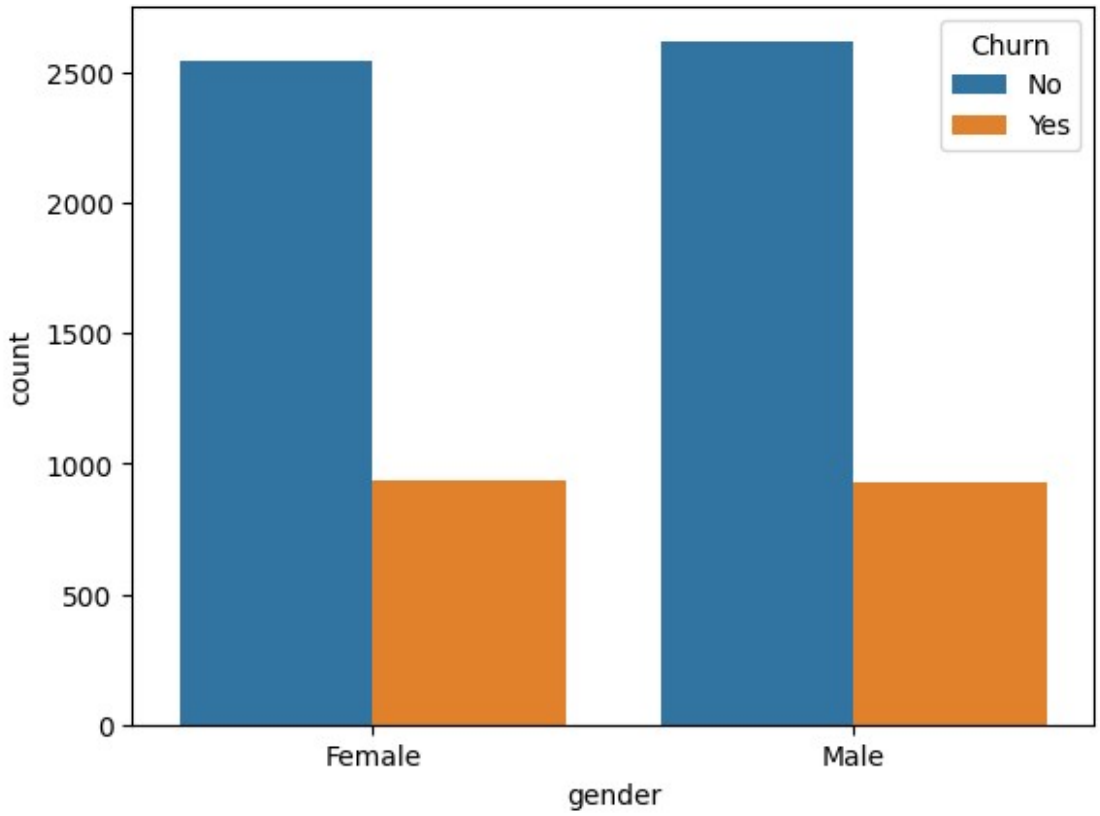
```
data_copy.drop(columns= ['customerID','tenure'], axis=1, inplace=True)
data_copy.head()
```

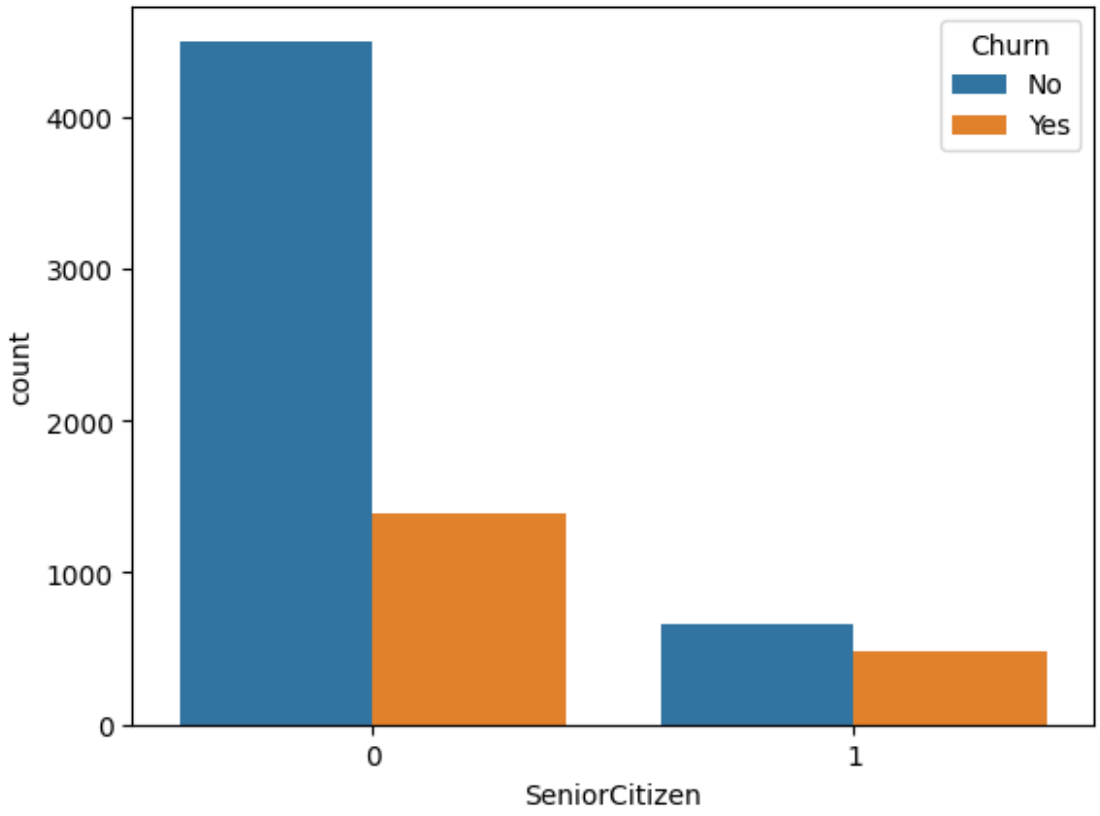

```
{"summary":"{\n  \"name\": \"data_copy\",\n  \"rows\": 7032,\n
\"fields\": [\n    {\n      \"column\": \"gender\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"Male\",\n          \"Female\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SeniorCitizen\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Partner\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dependents\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PhoneService\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Yes\",\n          \"No\"\n        ],\n
```

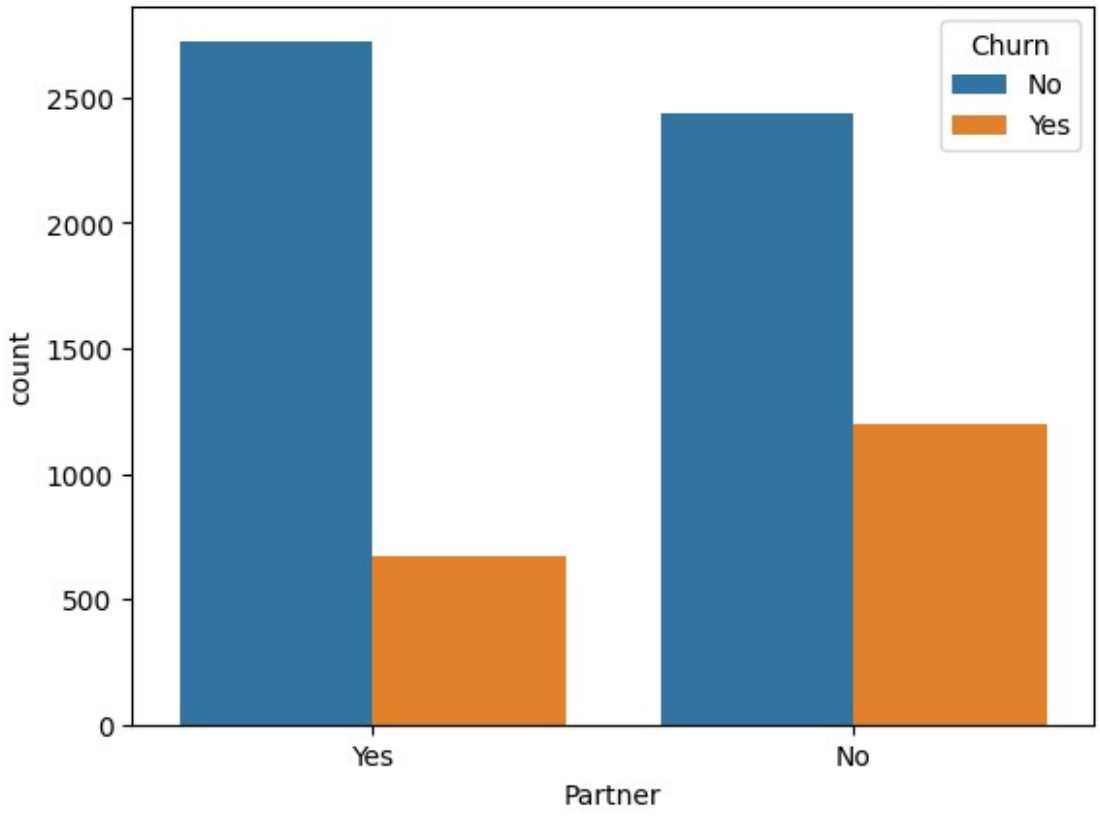
```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MultipleLines\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No
phone service\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InternetService\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"DSL\",\
n          \"Fiber optic\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"OnlineSecurity\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OnlineBackup\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DeviceProtection\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TechSupport\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StreamingTV\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StreamingMovies\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Contract\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Month-to-month\",\n          \"One year\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PaperlessBilling\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PaymentMethod\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
```

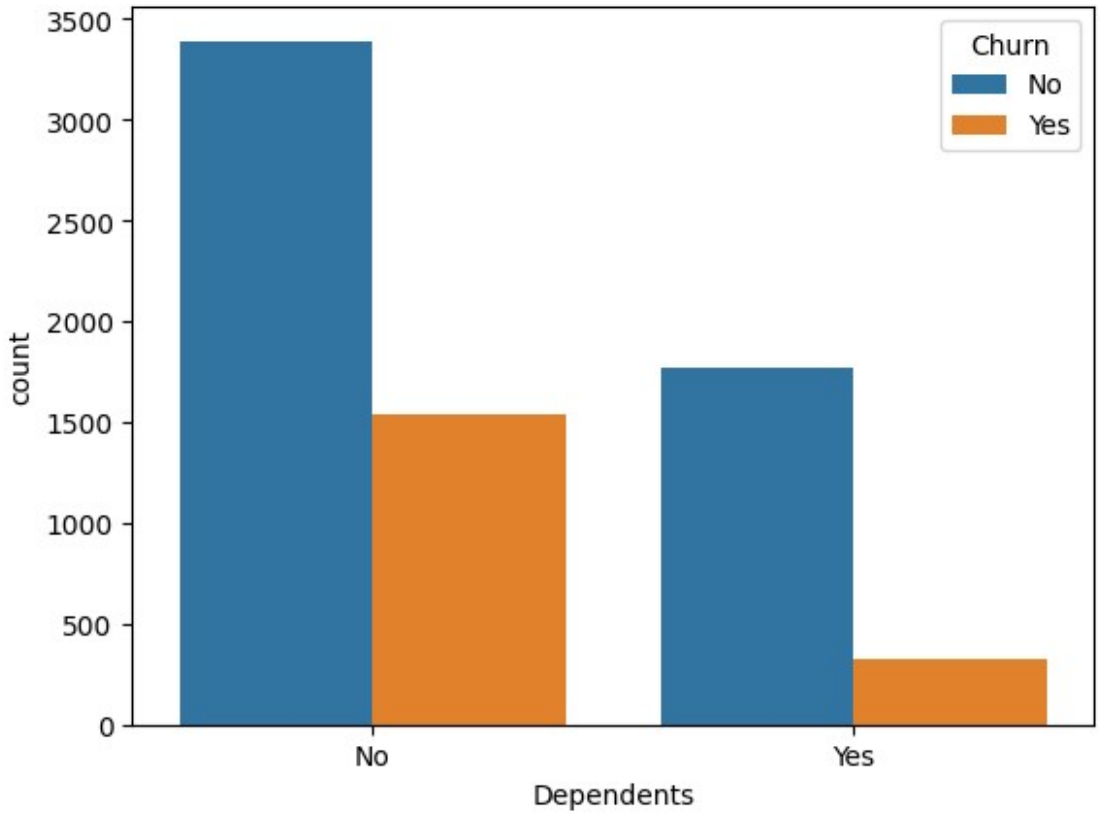
```
[\n          \"Mailed check\",\n          \"Credit card (automatic)\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MonthlyCharges\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 30.085973884049842,\n        \"min\":
18.25,\n        \"max\": 118.75,\n        \"num_unique_values\":
1584,\n        \"samples\": [\n          102.85,\n          20.05\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"TotalCharges\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2266.771361883145,\n        \"min\": 18.8,\n        \"max\": 8684.8,\n
\"num_unique_values\": 6530,\n        \"samples\": [\n
5594.0,\n          6840.95\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Churn\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Yes\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"tenure_group\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 6,\n        \"samples\": [\n          \"1 -
12\",\n          \"25 - 36\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data_copy"}
```

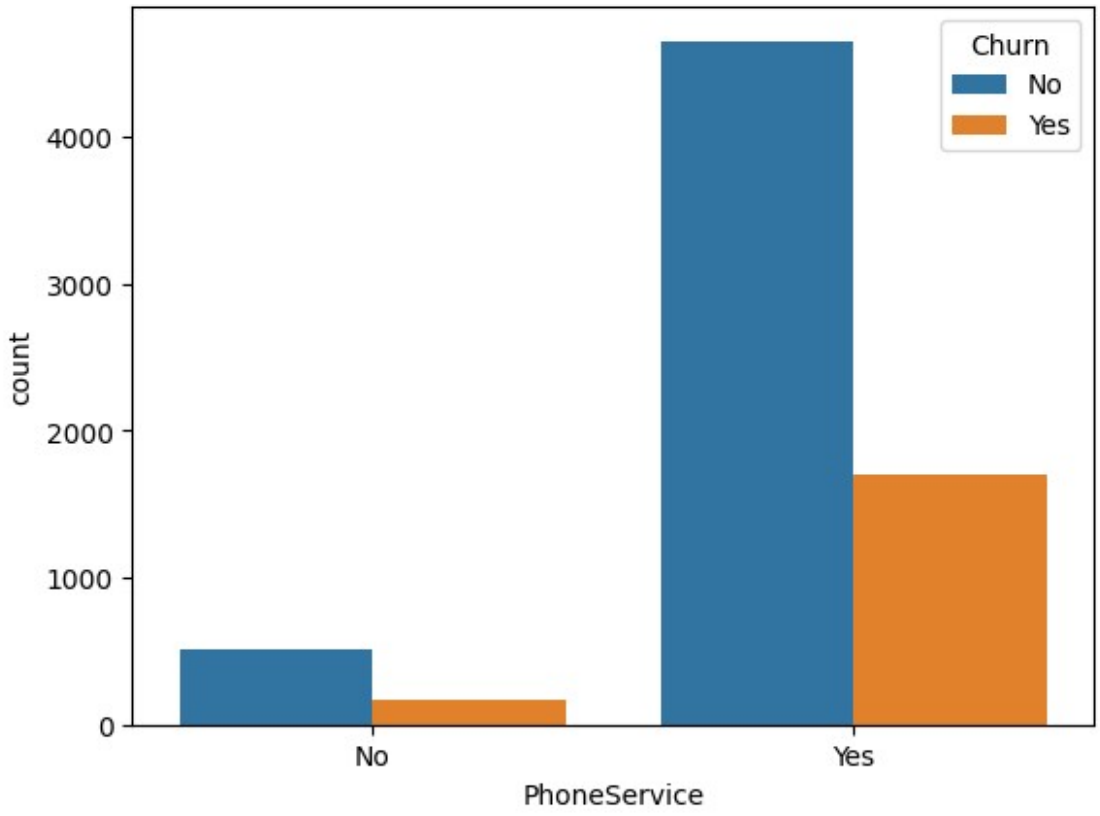

```
for i, predictor in enumerate(data_copy.drop(columns=['Churn',
'TotalCharges', 'MonthlyCharges'])):
    plt.figure(i)
    sns.countplot(data=data_copy, x=predictor, hue='Churn')
```

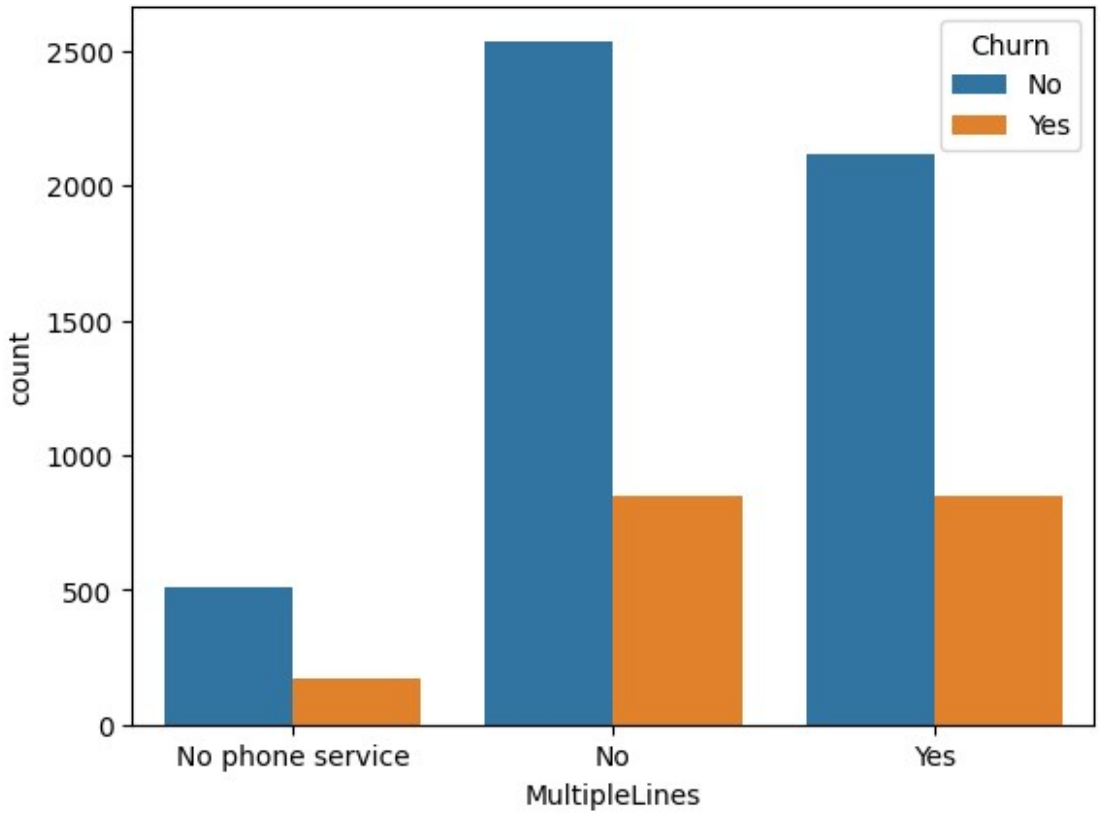

Churn
No
Yes
2500
2000
1500
1000
500
0
count
Female
Male
gender

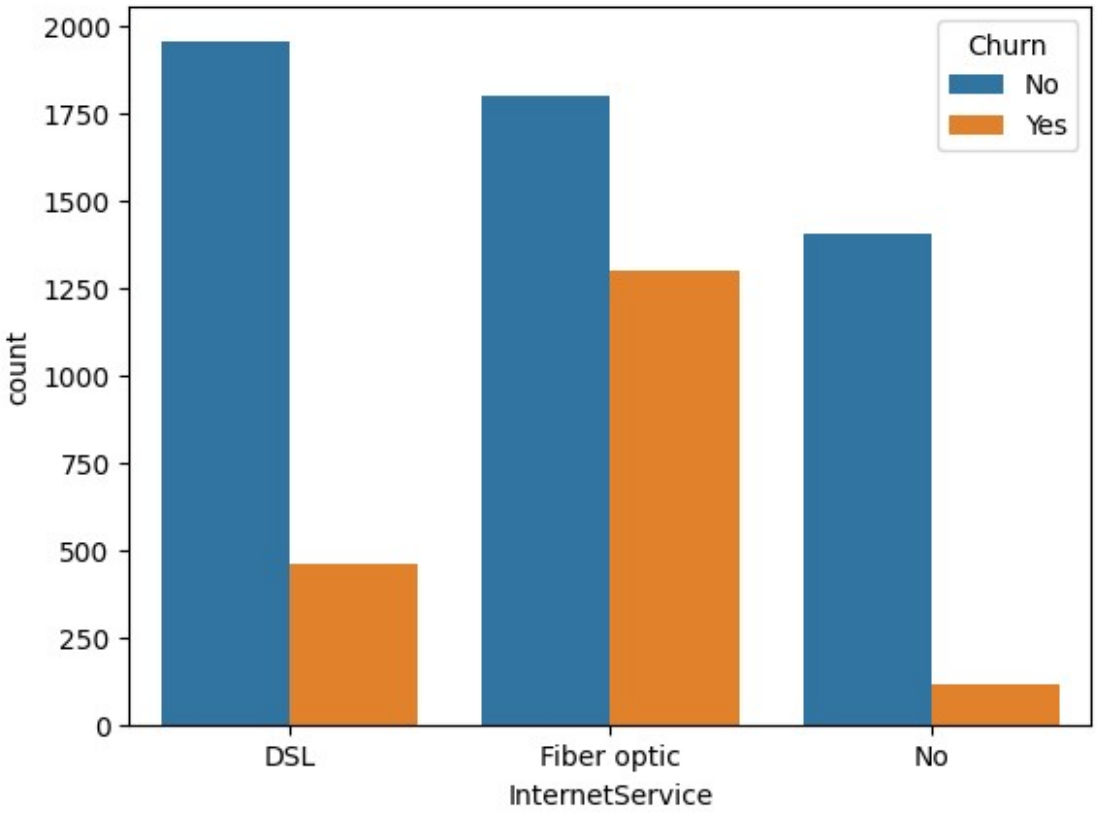

Churn
No
Yes
4000
3000
2000
1000
0
count
0
1
SeniorCitizen

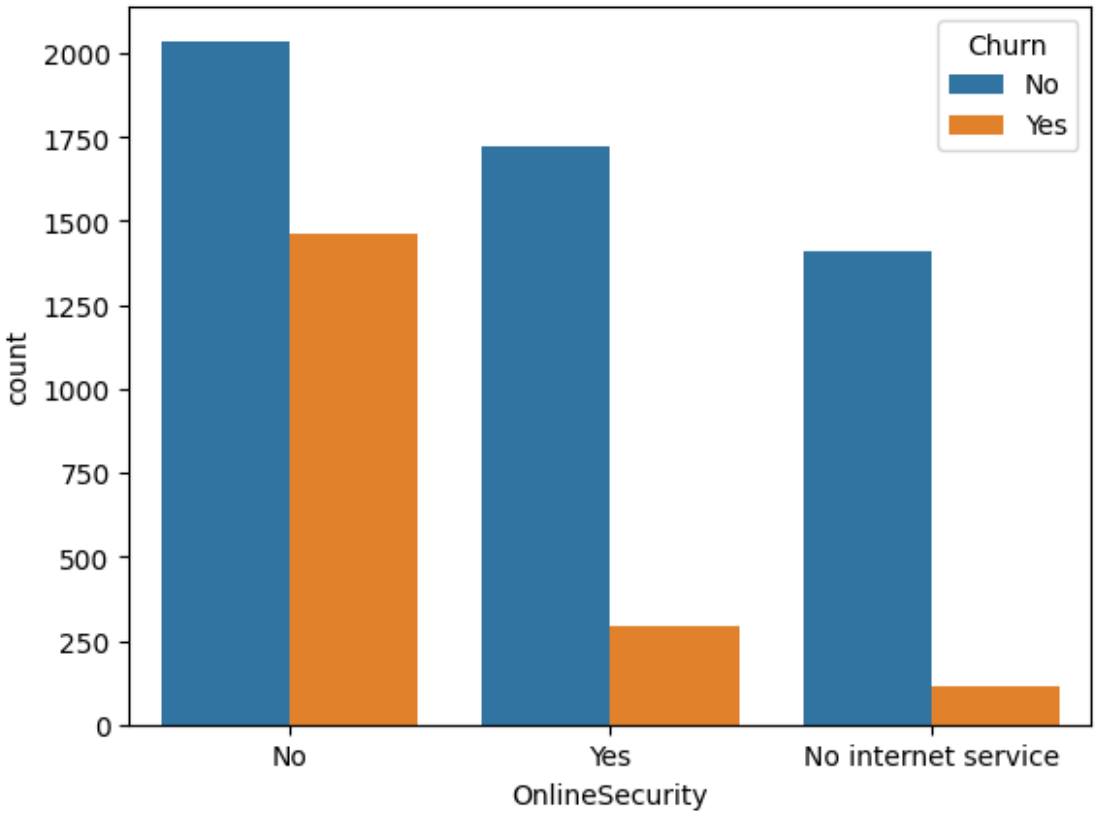

Churn
No
Yes
2500
2000
1500
1000
500
0
count
Yes
No
Partner

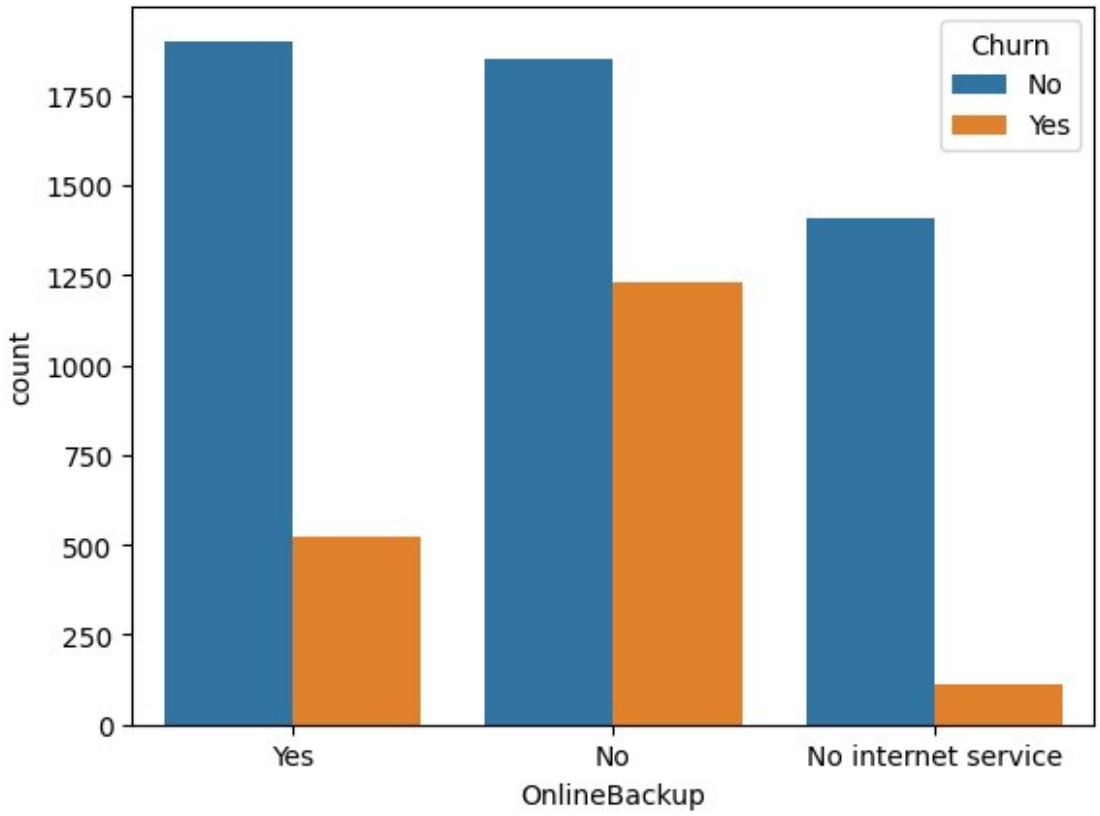

Churn
No
Yes
3500
3000
2500
2000
1500
1000
500
0
count
No
Yes
Dependents

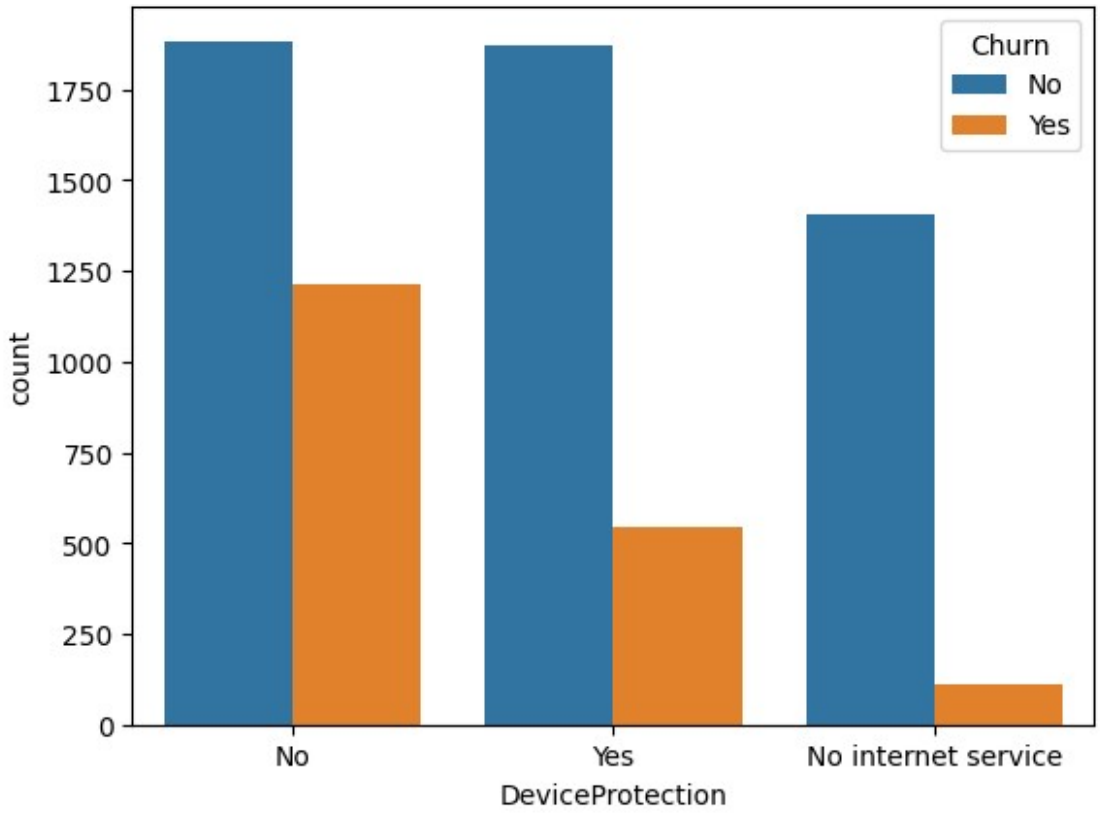

Churn
No
Yes
4000
3000
2000
1000
0
count
No
Yes
PhoneService

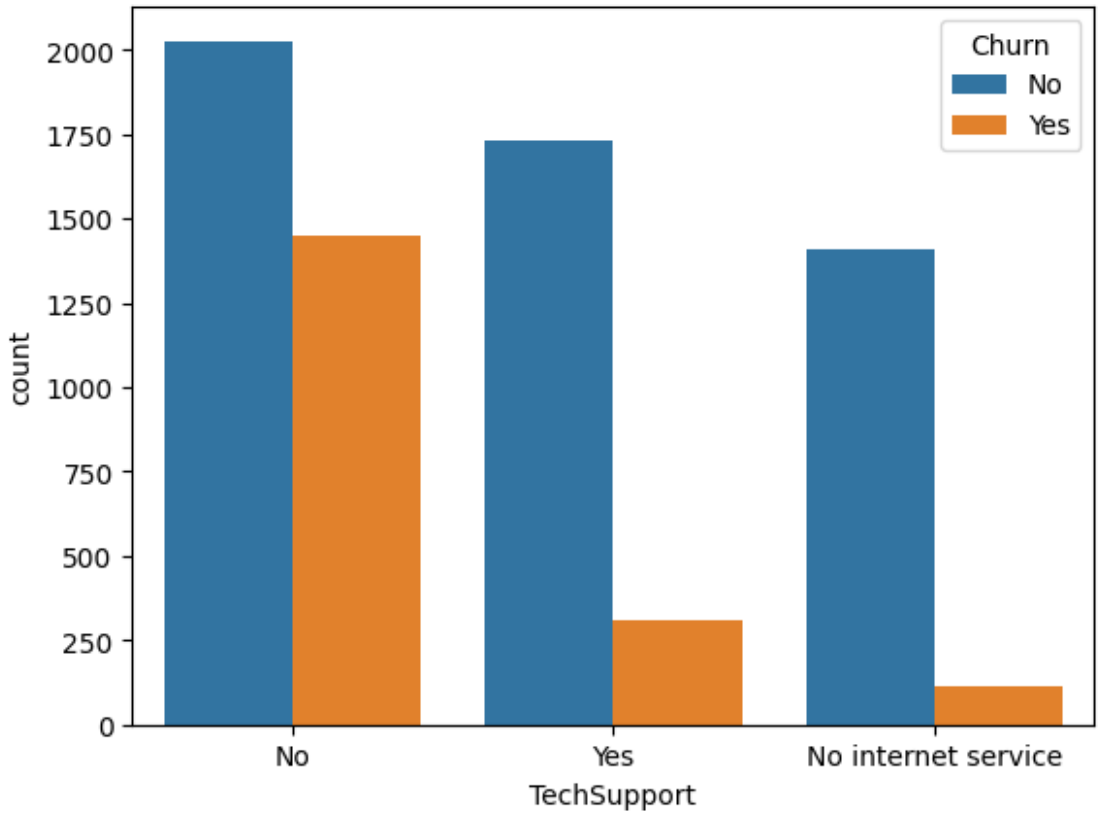

Churn
No
Yes
2500
2000
1500
1000
500
0
count
No phone service
No
Yes
MultipleLines

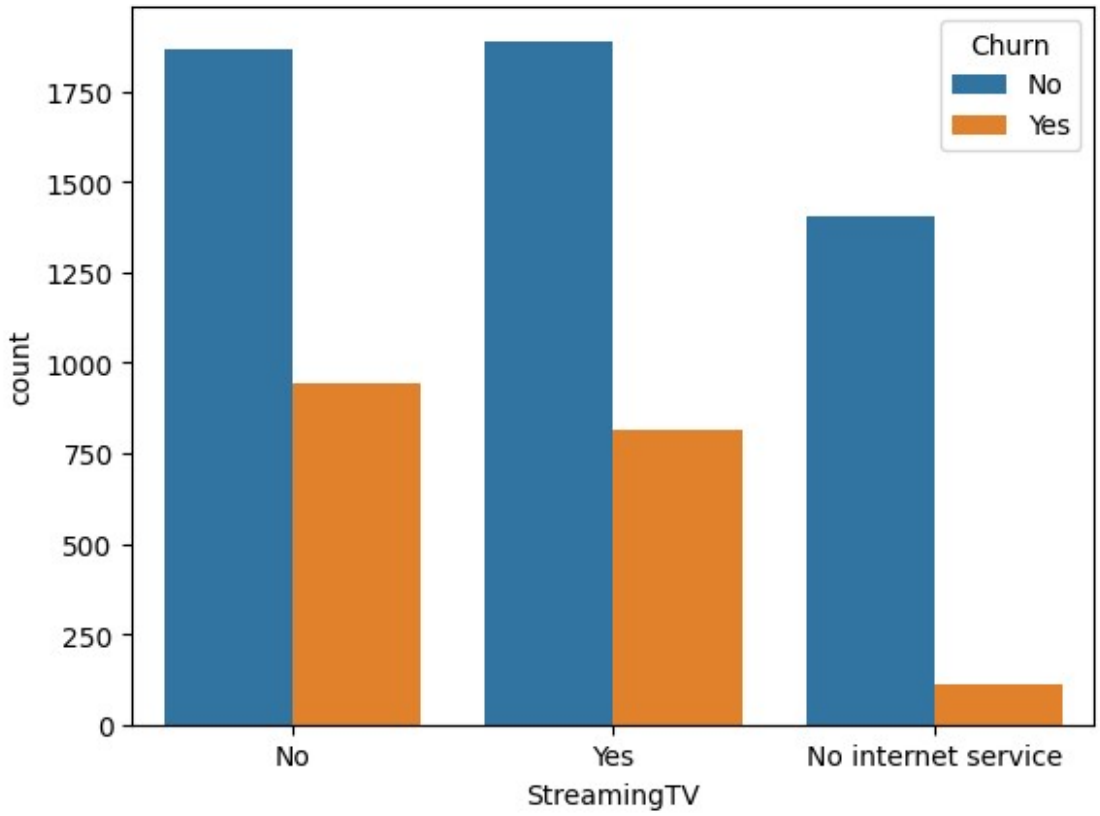

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
DSL
Fiber optic
No
InternetService

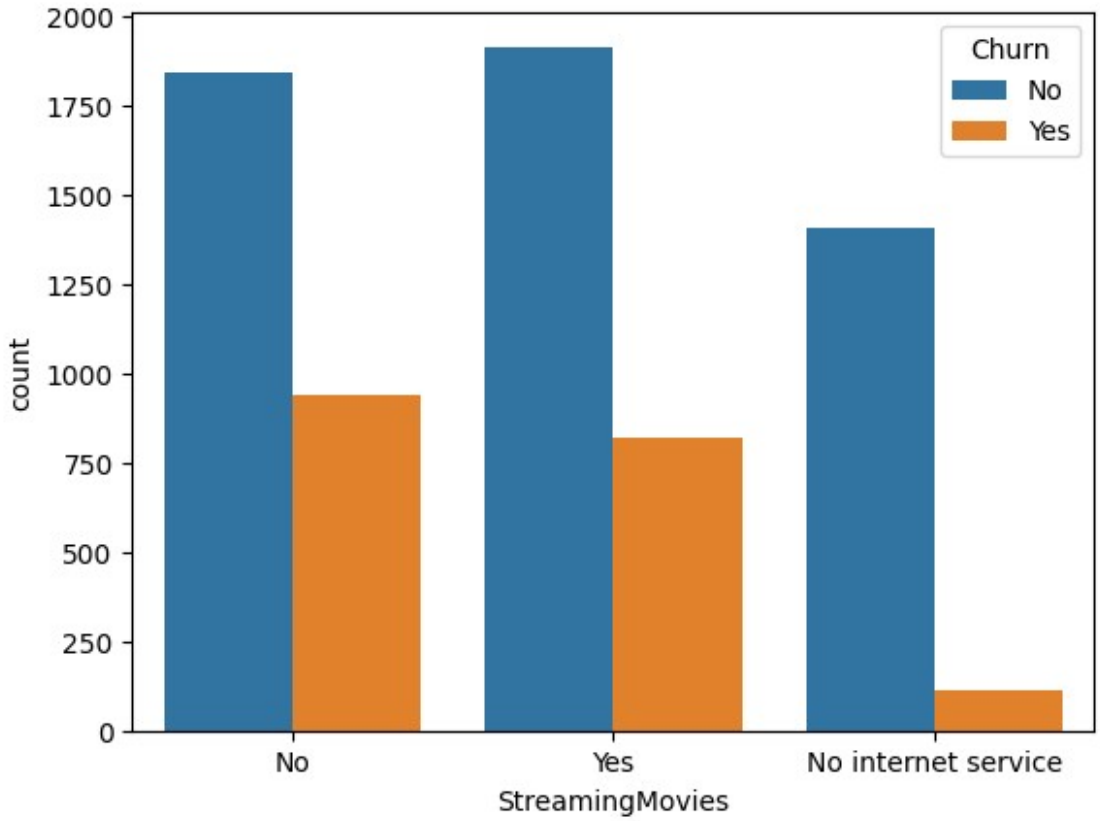

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
OnlineSecurity

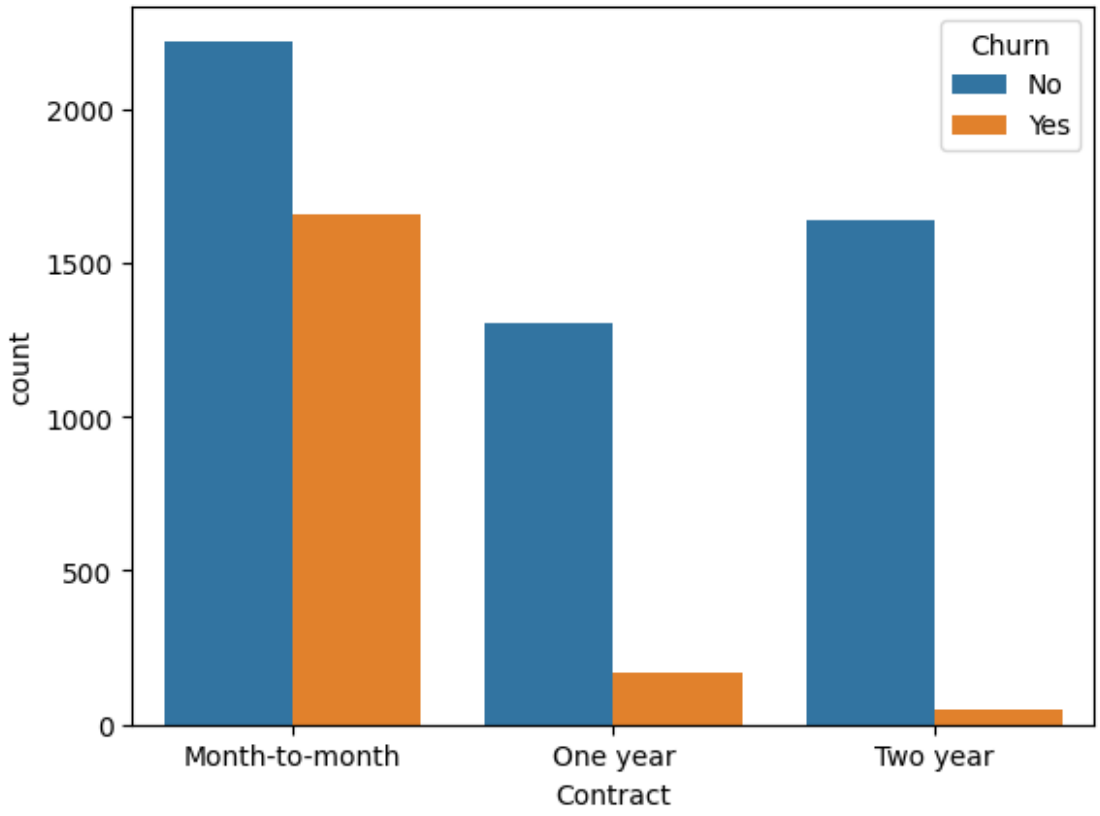

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
Yes
No
No internet service
OnlineBackup

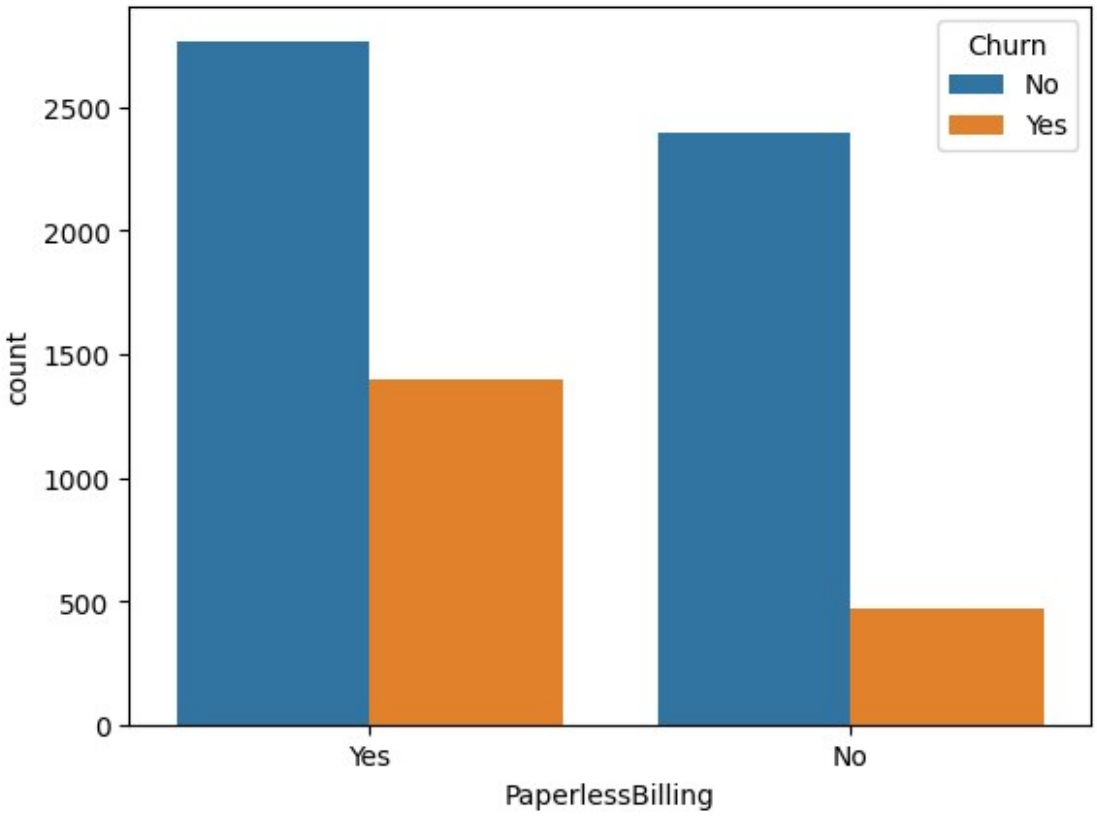

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
DeviceProtection

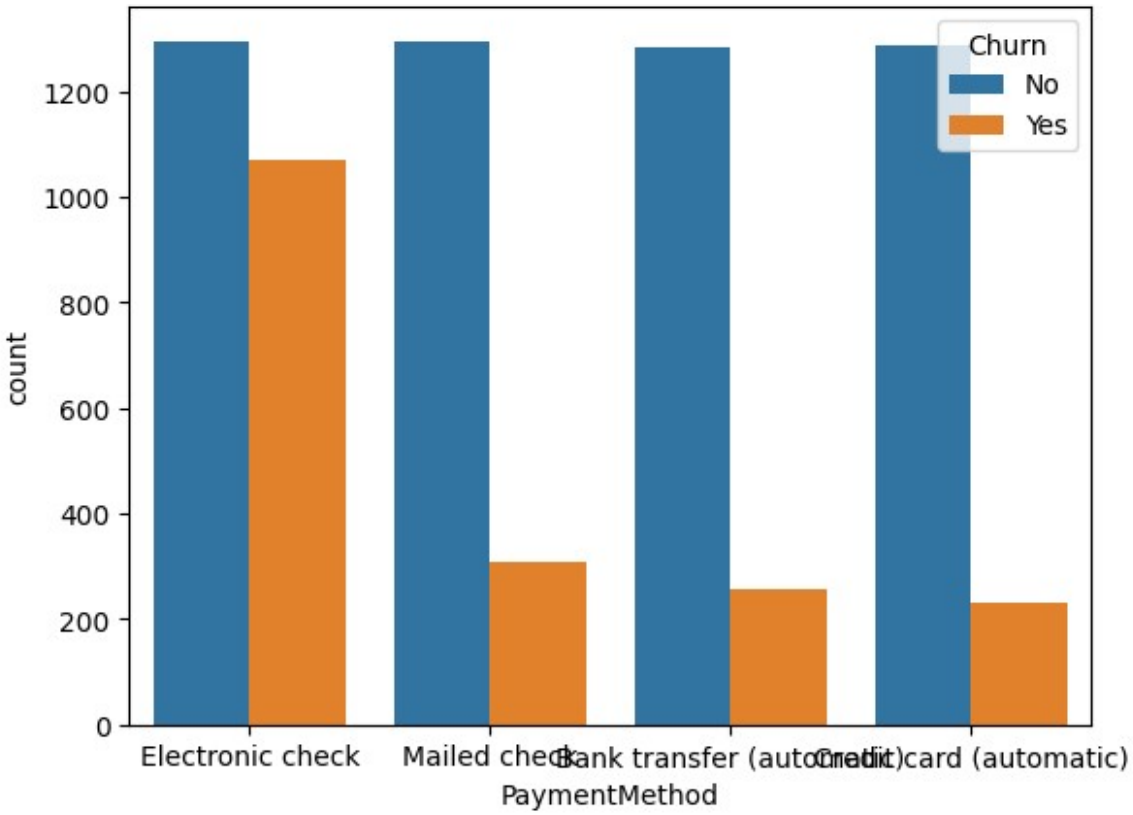

Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
TechSupport

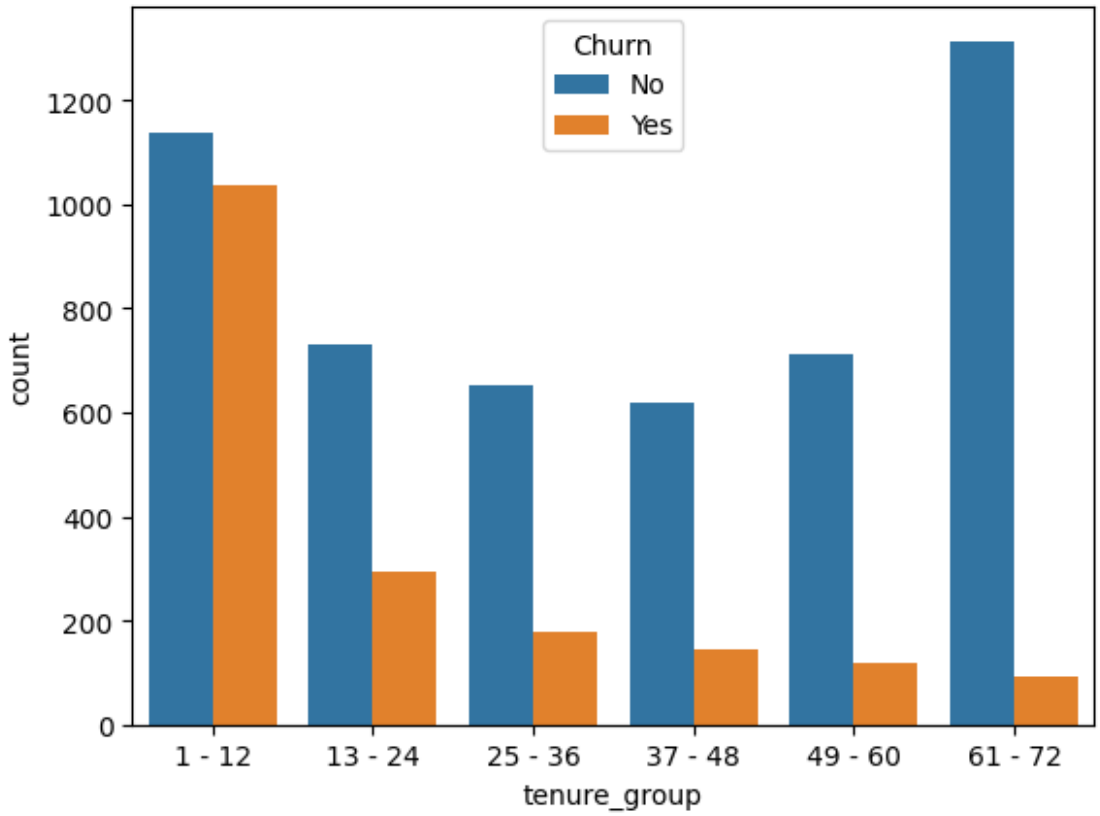

Churn
No
Yes
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
StreamingTV


Churn
No
Yes
2000
1750
1500
1250
1000
750
500
250
0
count
No
Yes
No internet service
StreamingMovies


Churn
No
Yes
2000
1500
1000
500
0
count
Month-to-month
One year
Two year
Contract


Churn
No
Yes
2500
2000
1500
1000
500
0
count
Yes
No
PaperlessBilling


Churn
No
Yes
1200
1000
800
600
400
200
0
count
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod




```
data_copy['Churn'] = np.where(data_copy.Churn == 'Yes',1,0)

data_copy.head()
```


```
{"summary":"{\n  \"name\": \"data_copy\",\n  \"rows\": 7032,\n
\"fields\": [\n    {\n      \"column\": \"gender\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n
\"Male\",\n          \"Female\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"SeniorCitizen\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Partner\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Dependents\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PhoneService\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"Yes\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"MultipleLines\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No
phone service\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InternetService\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"DSL\",\
n          \"Fiber optic\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"OnlineSecurity\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OnlineBackup\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DeviceProtection\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TechSupport\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StreamingTV\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StreamingMovies\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Contract\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Month-to-month\",\n          \"One year\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PaperlessBilling\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"No\",\n
```

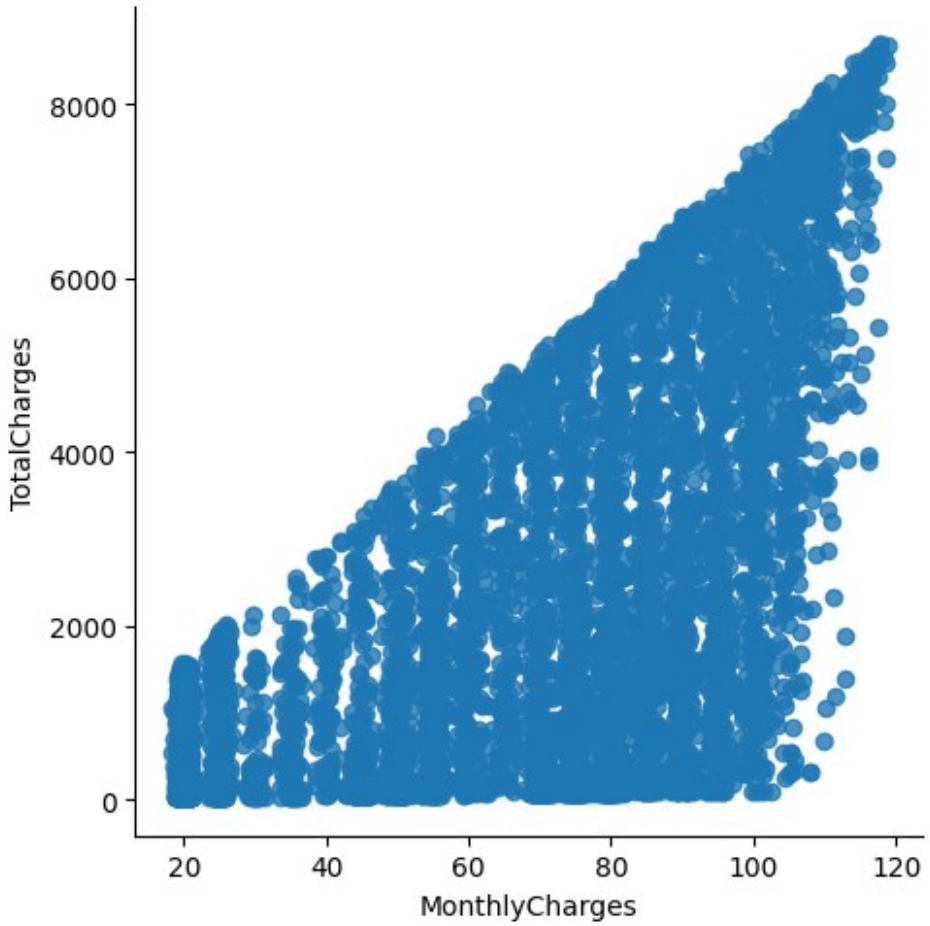
```
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PaymentMethod\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"Mailed check\",\n          \"Credit card (automatic)\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MonthlyCharges\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 30.085973884049842,\n        \"min\":
18.25,\n        \"max\": 118.75,\n        \"num_unique_values\":
1584,\n        \"samples\": [\n          102.85,\n          20.05\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"TotalCharges\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2266.771361883145,\n        \"min\": 18.8,\n        \"max\": 8684.8,\n
\"num_unique_values\": 6530,\n        \"samples\": [\n
5594.0,\n          6840.95\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Churn\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"tenure_group\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 6,\n
\"samples\": [\n          \"1 - 12\",\n          \"25 - 36\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"data_copy"}
```


```
data_copy_dumies = pd.get_dummies(data_copy)
data_copy_dumies.head()
```


```
{"type":"dataframe","variable_name":"data_copy_dumies"}
```


```
sns.lmplot(data=data_copy_dumies, x='MonthlyCharges',
y='TotalCharges', fit_reg=False)
```

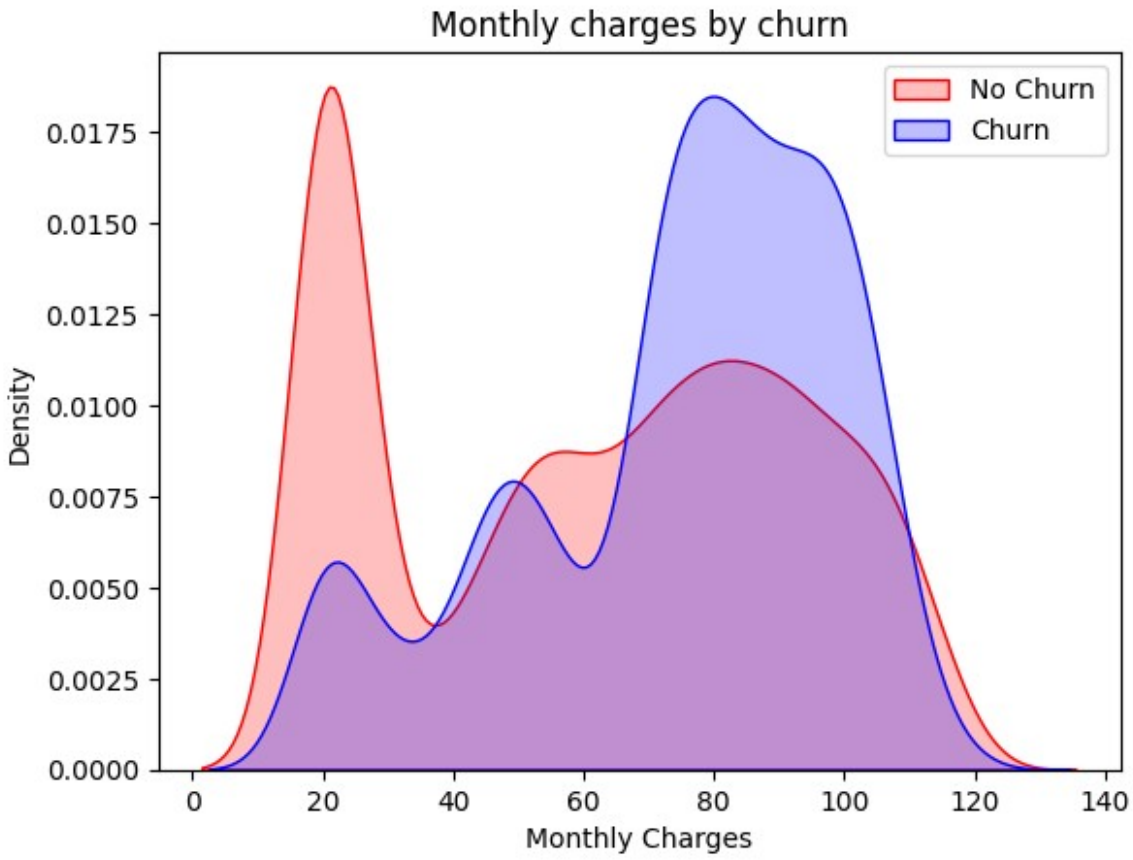
```
<seaborn.axisgrid.FacetGrid at 0x7bd392679b40>
```




```
Mth =
sns.kdeplot(data_copy_dumies.MonthlyCharges[(data_copy_dumies["Churn"]
== 0) ],
                color="Red", shade = True)
Mth =
sns.kdeplot(data_copy_dumies.MonthlyCharges[(data_copy_dumies["Churn"]
== 1) ],
                ax =Mth, color="Blue", shade= True)
Mth.legend(["No Churn","Churn"],loc='upper right')
Mth.set_ylabel('Density')
Mth.set_xlabel('Monthly Charges')
Mth.set_title('Monthly charges by churn')
```

```
<ipython-input-26-f7410b51e9eb>:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  Mth =
sns.kdeplot(data_copy_dumies.MonthlyCharges[(data_copy_dumies["Churn"]
```

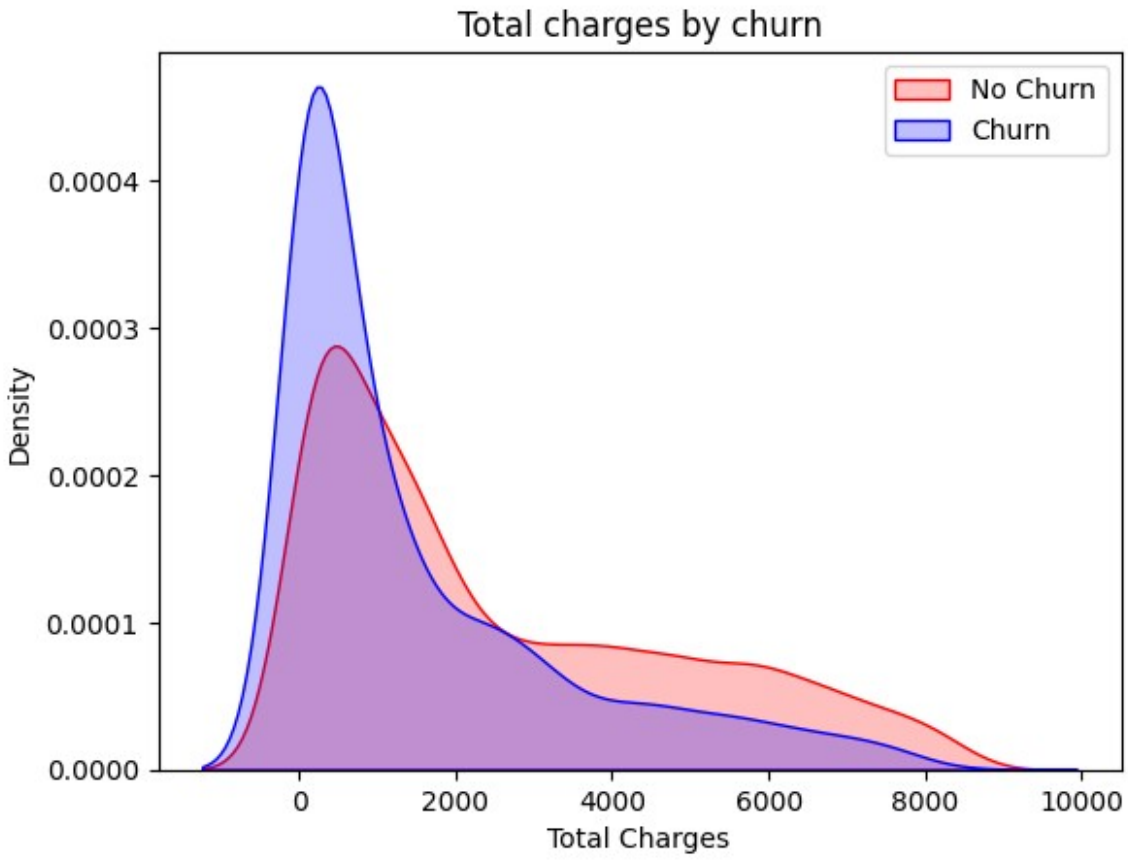
```
== 0) ],
<ipython-input-26-f7410b51e9eb>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  Mth =
sns.kdeplot(data_copy_dumies.MonthlyCharges[(data_copy_dumies["Churn"]
== 1) ],

Text(0.5, 1.0, 'Monthly charges by churn')
```




```
Tot =
sns.kdeplot(data_copy_dumies.TotalCharges[(data_copy_dumies["Churn"]
== 0) ],
                color="Red", shade = True)
Tot =
sns.kdeplot(data_copy_dumies.TotalCharges[(data_copy_dumies["Churn"]
== 1) ],
                ax =Tot, color="Blue", shade= True)
Tot.legend(["No Churn","Churn"],loc='upper right')
Tot.set_ylabel('Density')
```

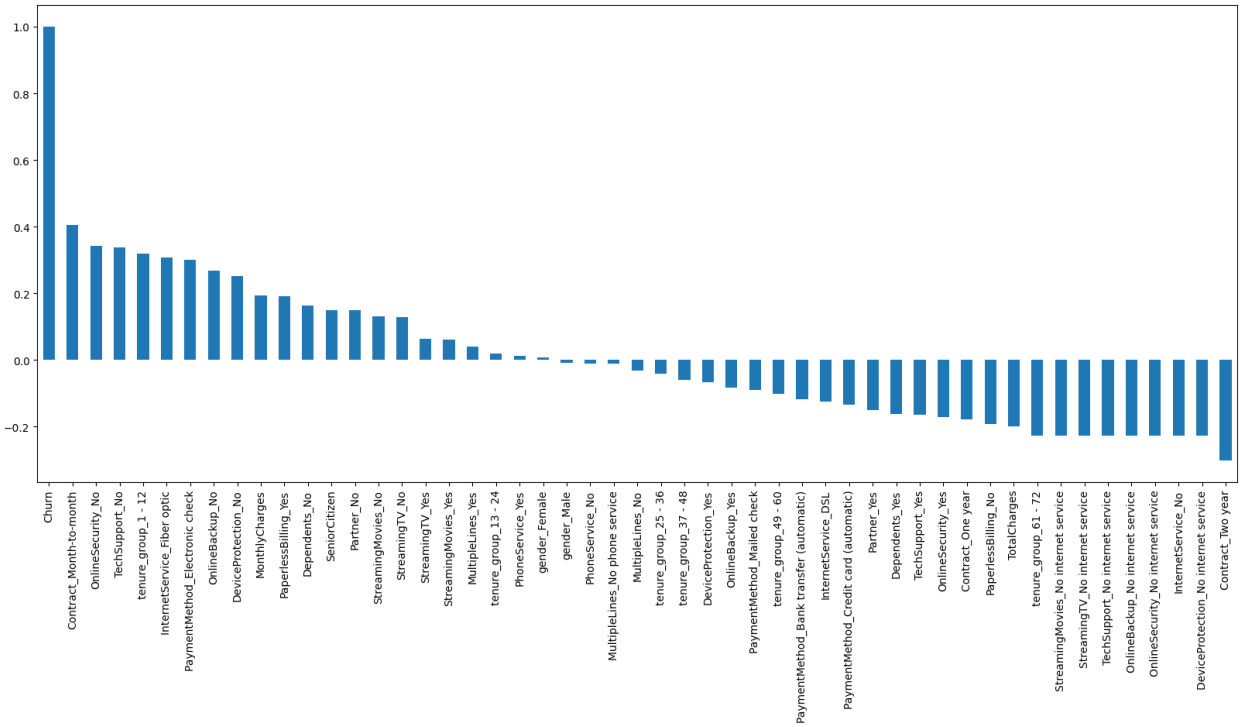
```
Tot.set_xlabel('Total Charges')
Tot.set_title('Total charges by churn')
```

```
<ipython-input-27-a5d2f272417a>:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  Tot =
sns.kdeplot(data_copy_dumies.TotalCharges[(data_copy_dumies["Churn"]
== 0) ],
<ipython-input-27-a5d2f272417a>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  Tot =
sns.kdeplot(data_copy_dumies.TotalCharges[(data_copy_dumies["Churn"]
== 1) ],

Text(0.5, 1.0, 'Total charges by churn')
```



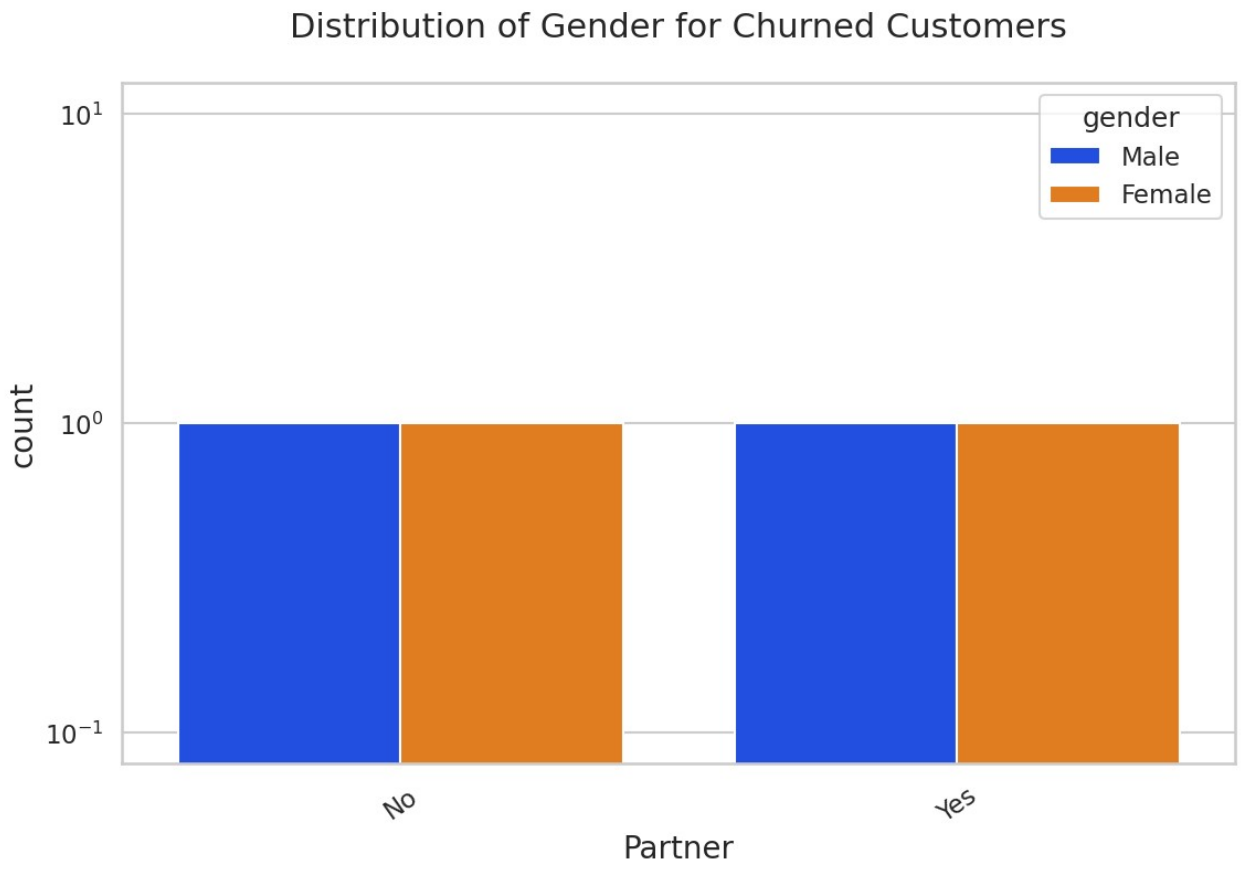
```
plt.figure(figsize=(20,8))
data_copy_dumies.corr()['Churn'].sort_values(ascending =
False).plot(kind='bar')#with all predictors
```

```
<Axes: >
```

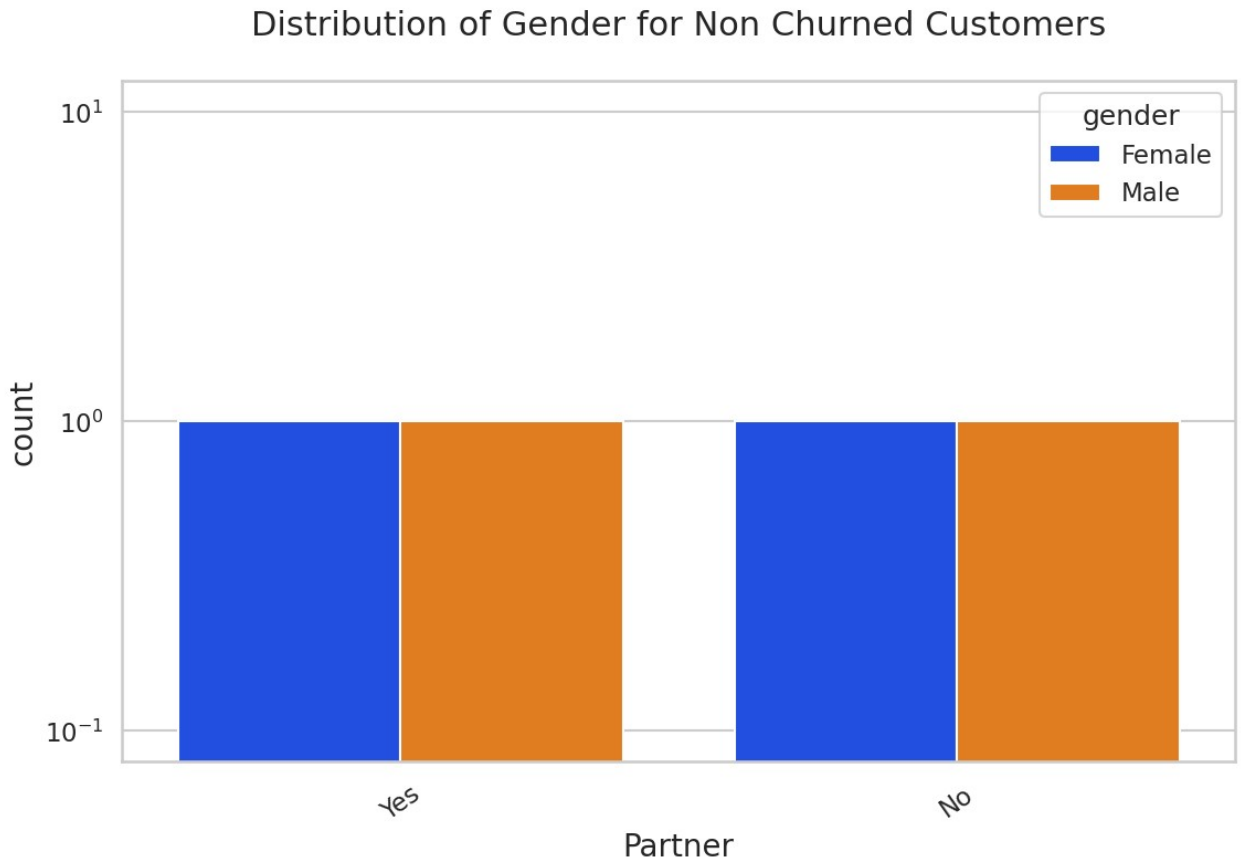



```
#bivariate analysis
nochurners=data_copy.loc[data_copy["Churn"]==0]
churners=data_copy.loc[data_copy["Churn"]==1]

def uniplot(df,col,title,hue =None):

    sns.set_style('whitegrid')
    sns.set_context('talk')
    plt.rcParams["axes.labelsize"] = 20
    plt.rcParams['axes.titlesize'] = 22
    plt.rcParams['axes.titlepad'] = 30


    temp = pd.Series(data = hue)
    fig, ax = plt.subplots()
    width = len(df[col].unique()) + 7 + 4*len(temp.unique())
    fig.set_size_inches(width , 8)
    plt.xticks(rotation=35)
    plt.yscale('log')
    plt.title(title)
    ax = sns.countplot(data = df, x= col,
```

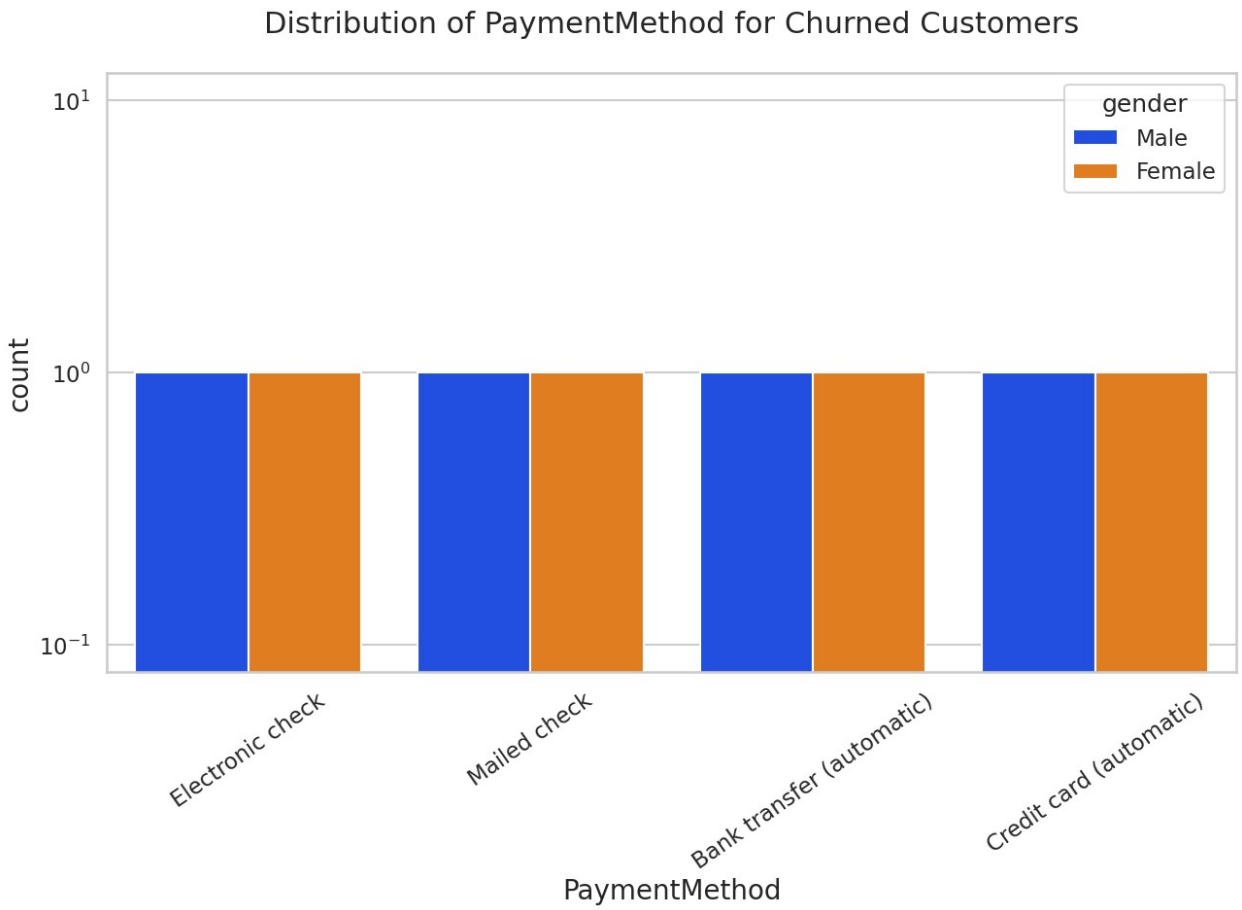
```
order=df[col].value_counts().index,hue = hue,palette='bright')

    plt.show()

uniplot(churners,col='Partner',title='Distribution of Gender for
Churned Customers',hue='gender')
```



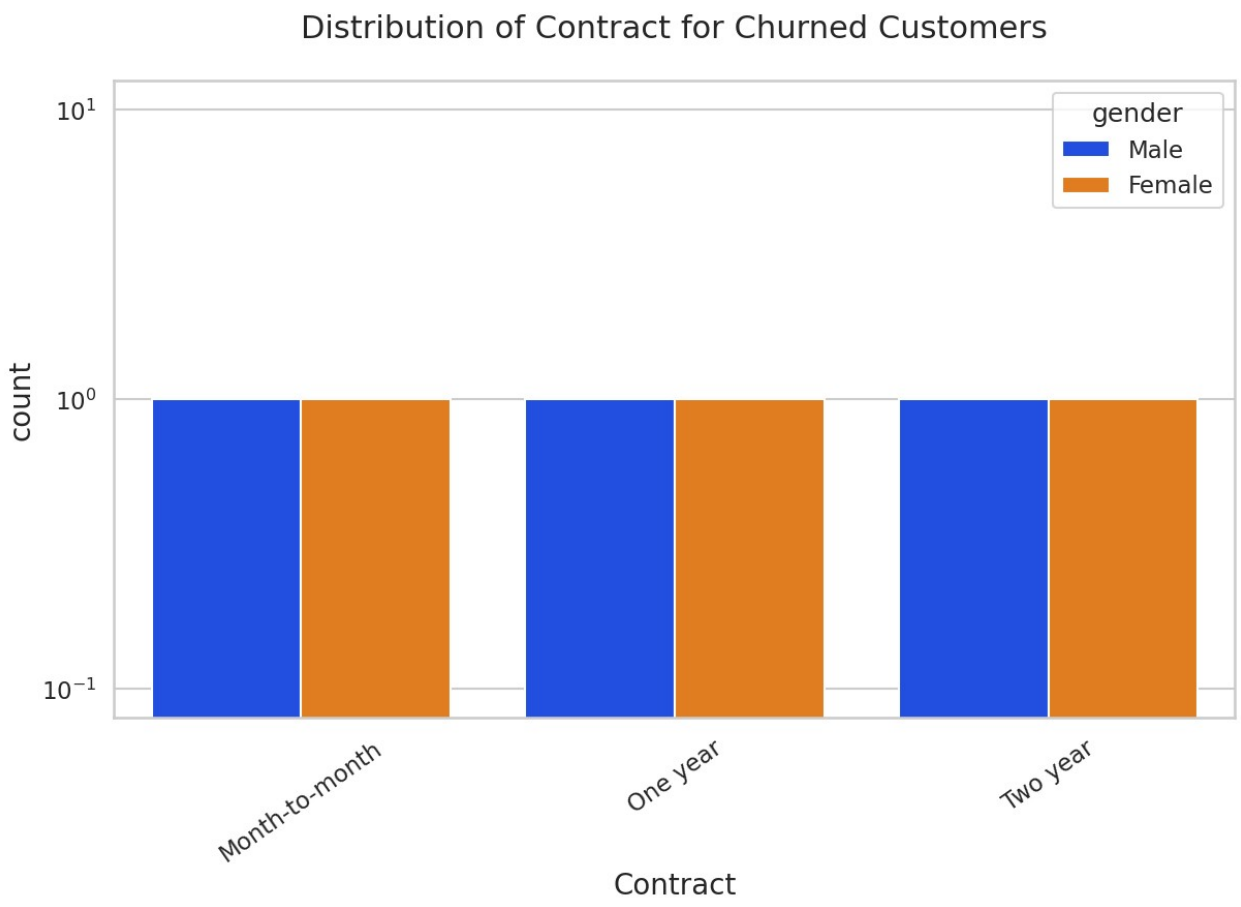

```
uniplot(nochurners,col='Partner',title='Distribution of Gender for Non
Churned Customers',hue='gender')
```



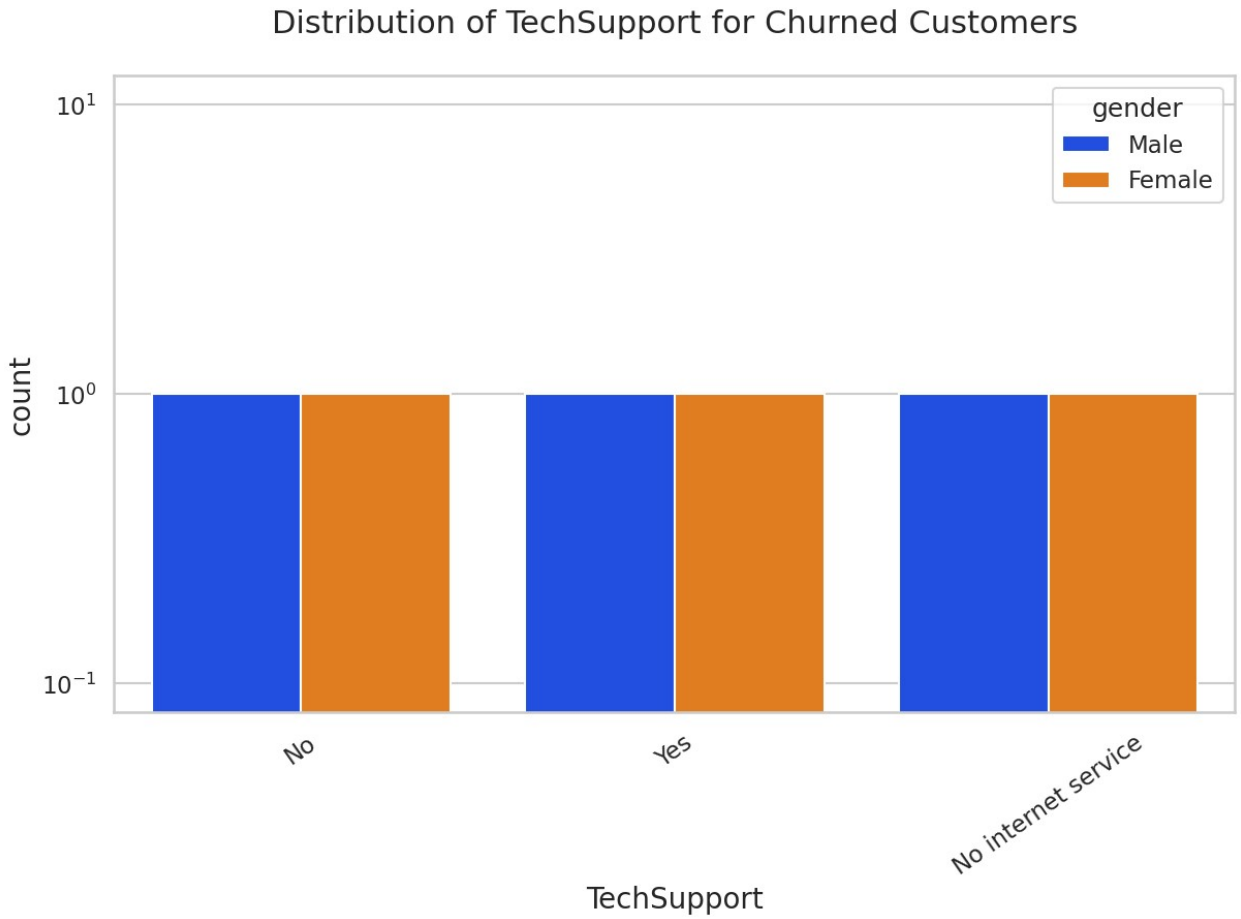

```
uniplot(churners,col='PaymentMethod',title='Distribution of
PaymentMethod for Churned Customers',hue='gender')
```



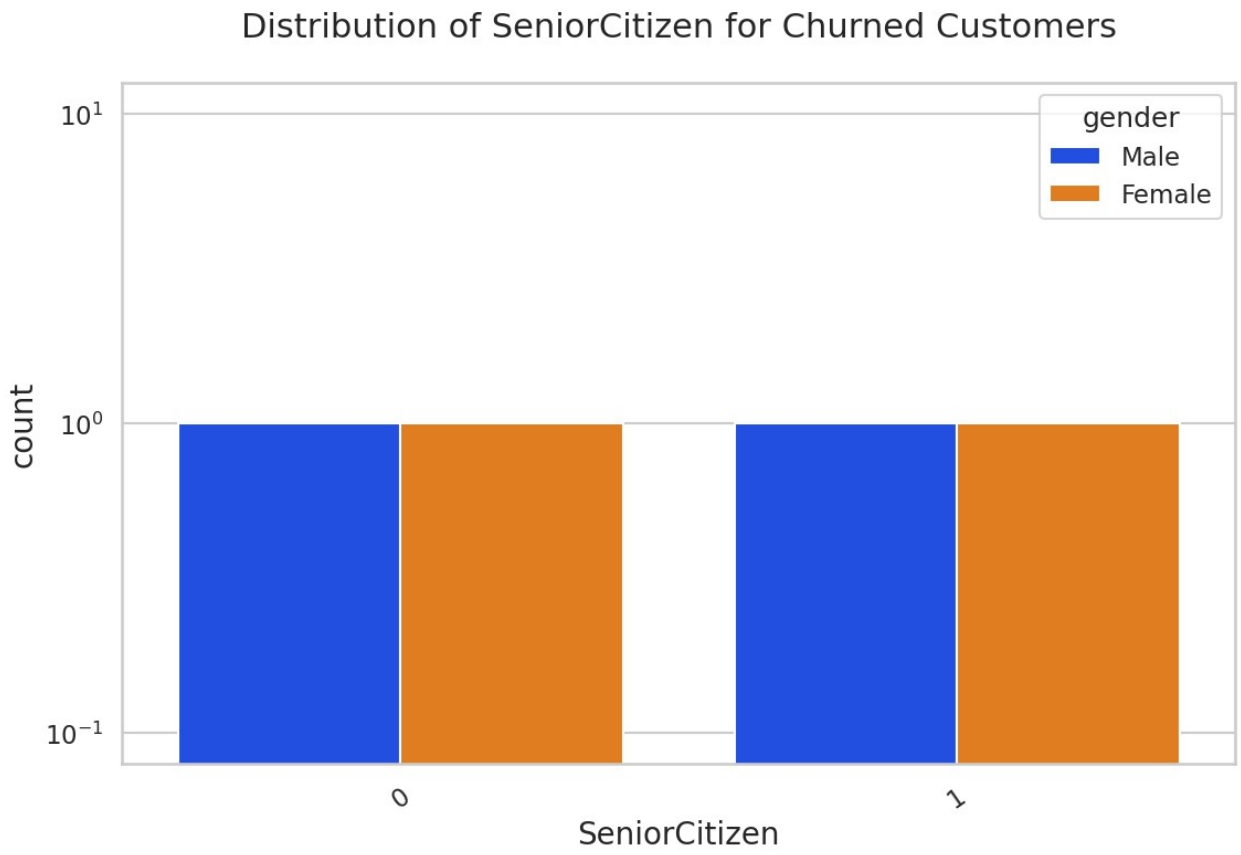

```
uniplot(churners,col='Contract',title='Distribution of Contract for
Churned Customers',hue='gender')
```




```
uniplot(churners,col='TechSupport',title='Distribution of TechSupport
for Churned Customers',hue='gender')
```




```
uniplot(churners,col='SeniorCitizen',title='Distribution of
SeniorCitizen for Churned Customers',hue='gender')
```




```
data_copy_dumies.to_csv('telco_churn_after_preprocess')

df=pd.read_csv("telco_churn_after_preprocess")
df = df.applymap(lambda x: 1 if x is True else 0 if x is False else x)
del df['Unnamed: 0']
df.to_csv('final_feed_csv')
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F

churn_column = df.iloc[:, 3]
features = df.drop(df.columns[3], axis=1)

churn_tensor = torch.tensor(churn_column.values, dtype=torch.int64)
features_tensor = torch.tensor(features.values, dtype=torch.float32)
print(churn_tensor)
print(features_tensor)
```

```
tensor([0, 0, 1,  ..., 0, 1, 0])
tensor([[0.0000e+00, 2.9850e+01, 2.9850e+01,  ..., 0.0000e+00,
0.0000e+00,
```

```
         0.0000e+00],
        [0.0000e+00, 5.6950e+01, 1.8895e+03,  ..., 0.0000e+00,
0.0000e+00,
         0.0000e+00],
        [0.0000e+00, 5.3850e+01, 1.0815e+02,  ..., 0.0000e+00,
0.0000e+00,
         0.0000e+00],
        ...,
        [0.0000e+00, 2.9600e+01, 3.4645e+02,  ..., 0.0000e+00,
0.0000e+00,
         0.0000e+00],
        [1.0000e+00, 7.4400e+01, 3.0660e+02,  ..., 0.0000e+00,
0.0000e+00,
         0.0000e+00],
        [0.0000e+00, 1.0565e+02, 6.8445e+03,  ..., 0.0000e+00,
0.0000e+00,
         1.0000e+00]])

input_size = features_tensor.shape[1]

from torch.utils.data import Dataset, DataLoader
import torch.optim as optim
# Define dataset
class ChurnDataset(Dataset):
    def __init__(self, features, labels):
        self.features = features
        self.labels = labels

    def __len__(self):
        return len(self.features)

    def __getitem__(self, idx):
        return self.features[idx], self.labels[idx]


# Create dataset and dataloader
dataset = ChurnDataset(features_tensor, churn_tensor)

train_loader = DataLoader(dataset, batch_size=32, shuffle=True)


class ChurnNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):

        super(ChurnNN, self).__init__()
        self.finput = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc1_1= nn.Linear(hidden_size, hidden_size)
        self.fc1_2 = nn.Linear(hidden_size, hidden_size * 2)
        self.fc2_3 = nn.Linear(hidden_size * 2 , hidden_size * 3)
```

```
        self.fc3_2 = nn.Linear(hidden_size * 3, hidden_size * 2)
        self.fc2_1 = nn.Linear(hidden_size * 2, hidden_size)
        self.output = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(.05)
        self.batchNorm1 = nn.BatchNorm1d(hidden_size)
        self.batchNorm2 = nn.BatchNorm1d(hidden_size * 2)
        self.batchNorm3 = nn.BatchNorm1d(hidden_size * 3)

    def forward(self, x):
        x = x.float()
        x = self.finput(x)
        # x = self.batchNorm1(x)
        x = self.relu(x)
        x = self.fc1_1(x)
        x = self.relu(x)
        x = self.fc1_1(x)
        x = self.relu(x)
        x = self.fc1_1(x)
        x = self.relu(x)
        x = self.fc1_1(x)
        x = self.relu(x)
        x = self.output(x)



        # print(f"X is now: {x}")
        # output_probs = nn.functional.softmax(x, dim=1)
        output_probs = x
        return output_probs

model = ChurnNN(input_size, 256, 2)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0006)
# 0.0006 - .34
# optimizer = optim.SGD(model.parameters(), lr=.001, momentum=0.9)
def train(model, optimizer, criterion, batch_size):
    # Set model to training mode
    model.train()
    num_epochs = 100

    for epoch in range(num_epochs):
        running_loss = 0.0

        # Iterate over the training dataset
        for inputs, targets in train_loader:
            # Zero the gradients
            optimizer.zero_grad()
```

```
            # Forward pass
            outputs = model(inputs)

            # Compute the loss

            # print(f"Outputs: {outputs}")
            # print(f"targets: {targets}")
            loss = criterion(outputs, targets)
            # print(f"loss: {loss}")

            # Backward pass
            loss.backward()

            # Update model parameters
            optimizer.step()

            # Accumulate the loss
            running_loss += loss.item() * inputs.size(0)

        # Calculate average loss for the epoch
        epoch_loss = running_loss / len(train_loader.dataset)

        # Print epoch statistics
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss:
{epoch_loss:.4f}")

    print("Training complete")

train(model, optimizer, criterion, train_loader)
```

```
Epoch [1/100], Loss: 0.6168
Epoch [2/100], Loss: 0.5350
Epoch [3/100], Loss: 0.5138
Epoch [4/100], Loss: 0.4940
Epoch [5/100], Loss: 0.4826
Epoch [6/100], Loss: 0.4690
Epoch [7/100], Loss: 0.4647
Epoch [8/100], Loss: 0.4538
Epoch [9/100], Loss: 0.4524
Epoch [10/100], Loss: 0.4479
Epoch [11/100], Loss: 0.4413
Epoch [12/100], Loss: 0.4402
Epoch [13/100], Loss: 0.4485
Epoch [14/100], Loss: 0.4366
Epoch [15/100], Loss: 0.4409
Epoch [16/100], Loss: 0.4334
Epoch [17/100], Loss: 0.4320
Epoch [18/100], Loss: 0.4305
Epoch [19/100], Loss: 0.4309
Epoch [20/100], Loss: 0.4245
```

```
Epoch [21/100], Loss: 0.4274
Epoch [22/100], Loss: 0.4243
Epoch [23/100], Loss: 0.4258
Epoch [24/100], Loss: 0.4262
Epoch [25/100], Loss: 0.4260
Epoch [26/100], Loss: 0.4269
Epoch [27/100], Loss: 0.4193
Epoch [28/100], Loss: 0.4296
Epoch [29/100], Loss: 0.4221
Epoch [30/100], Loss: 0.4221
Epoch [31/100], Loss: 0.4248
Epoch [32/100], Loss: 0.4260
Epoch [33/100], Loss: 0.4191
Epoch [34/100], Loss: 0.4193
Epoch [35/100], Loss: 0.4254
Epoch [36/100], Loss: 0.4204
Epoch [37/100], Loss: 0.4176
Epoch [38/100], Loss: 0.4230
Epoch [39/100], Loss: 0.4181
Epoch [40/100], Loss: 0.4188
Epoch [41/100], Loss: 0.4190
Epoch [42/100], Loss: 0.4208
Epoch [43/100], Loss: 0.4161
Epoch [44/100], Loss: 0.4164
Epoch [45/100], Loss: 0.4173
Epoch [46/100], Loss: 0.4213
Epoch [47/100], Loss: 0.4173
Epoch [48/100], Loss: 0.4183
Epoch [49/100], Loss: 0.4149
Epoch [50/100], Loss: 0.4184
Epoch [51/100], Loss: 0.4146
Epoch [52/100], Loss: 0.4160
Epoch [53/100], Loss: 0.4151
Epoch [54/100], Loss: 0.4134
Epoch [55/100], Loss: 0.4126
Epoch [56/100], Loss: 0.4114
Epoch [57/100], Loss: 0.4130
Epoch [58/100], Loss: 0.4134
Epoch [59/100], Loss: 0.4150
Epoch [60/100], Loss: 0.4169
Epoch [61/100], Loss: 0.4120
Epoch [62/100], Loss: 0.4094
Epoch [63/100], Loss: 0.4088
Epoch [64/100], Loss: 0.4089
Epoch [65/100], Loss: 0.4122
Epoch [66/100], Loss: 0.4093
Epoch [67/100], Loss: 0.4103
Epoch [68/100], Loss: 0.4134
Epoch [69/100], Loss: 0.4100
```

```
Epoch [70/100], Loss: 0.4098
Epoch [71/100], Loss: 0.4086
Epoch [72/100], Loss: 0.4101
Epoch [73/100], Loss: 0.4125
Epoch [74/100], Loss: 0.4077
Epoch [75/100], Loss: 0.4069
Epoch [76/100], Loss: 0.4049
Epoch [77/100], Loss: 0.4052
Epoch [78/100], Loss: 0.4047
Epoch [79/100], Loss: 0.4049
Epoch [80/100], Loss: 0.4061
Epoch [81/100], Loss: 0.4035
Epoch [82/100], Loss: 0.4066
Epoch [83/100], Loss: 0.4040
Epoch [84/100], Loss: 0.4044
Epoch [85/100], Loss: 0.4065
Epoch [86/100], Loss: 0.4051
Epoch [87/100], Loss: 0.4054
Epoch [88/100], Loss: 0.4016
Epoch [89/100], Loss: 0.4007
Epoch [90/100], Loss: 0.4028
Epoch [91/100], Loss: 0.4019
Epoch [92/100], Loss: 0.3991
Epoch [93/100], Loss: 0.4009
Epoch [94/100], Loss: 0.4039
Epoch [95/100], Loss: 0.3987
Epoch [96/100], Loss: 0.3994
Epoch [97/100], Loss: 0.4009
Epoch [98/100], Loss: 0.3989
Epoch [99/100], Loss: 0.4015
Epoch [100/100], Loss: 0.3956
Training complete
```

```
input_tensor = torch.tensor(features_tensor, dtype=torch.float32)

model.eval()

with torch.no_grad():
    predicted_outputs = model(input_tensor)

predicted_moves = torch.argmax(predicted_outputs, dim=1)
print(predicted_moves)


print(predicted_outputs)
print(churn_tensor)
```

```
<ipython-input-42-42efed418a43>:1: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
```

```
torch.tensor(sourceTensor).
  input_tensor = torch.tensor(features_tensor, dtype=torch.float32)

tensor([1, 0, 0,  ..., 0, 1, 0])
tensor([[-0.3541,  0.2420],
        [ 1.5643, -1.2819],
        [ 0.0037, -0.0356],
        ...,
        [ 0.6706, -0.5037],
        [-0.3540,  0.2175],
        [ 1.0392, -0.9422]])
tensor([0, 0, 1,  ..., 0, 1, 0])

#testing a new sample (modified from the data)
test_sample_tensor = features_tensor[1,:]
print('Input: ', test_sample_tensor)
print()
model.eval()

with torch.no_grad():
    predicted_outputs = model(test_sample_tensor)

predicted_moves = torch.argmax(predicted_outputs)
print('y_hat: ', predicted_moves)
churn = churn_tensor[1]
print('y: ', churn)

Input:  tensor([0.0000e+00, 5.6950e+01, 1.8895e+03, 0.0000e+00,
1.0000e+00, 1.0000e+00,
        0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
1.0000e+00,
        0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
0.0000e+00,
        0.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
0.0000e+00,
        0.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
1.0000e+00,
        0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
0.0000e+00,
        1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00,
0.0000e+00,
        0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
0.0000e+00,
        0.0000e+00, 0.0000e+00])

y_hat:  tensor(0)
y:  tensor(0)
```