`churn prediction using machine learning`

this was my first live experince doing the project during an interview

steps :

where there are 7043 rows and 21 columns

- seen if null values are there are not
- eda on the data to understand the pattern of data
- splited , scaling , encoding was done on which columns it needed
- then splitted the data and model development using 6 algorithms and 3 evalution score

algorithms :

- `Logistic Regression`
- `Random Forest`
- `K-Nearest Neighbors`
- `Support Vector Classifier`
- `XGBoost`
- `Decision Tree Classifier`

evalution scores :

- `accuracy`
- `f1-score`
- `precision`

problem's faced

- while running the code is there were null values in total charges columns in x train and x test
- filled them with fillna

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

In [3]:
```
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn(in).csv")
```

In [4]:
```
df.head()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Mul
0	7590-VHVEG	Female	0	Yes	No	1	No	
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	
4	9237-HQITU	Female	0	No	No	2	Yes	

5 rows × 21 columns

In [5]:
```
df["tenure"].value_counts()
```

Out[5]:

tenure	count
1	613
72	362
2	238
3	200
4	176
...	...
28	57
39	56
44	51
36	50
0	11

73 rows × 1 columns

dtype: int64

In [6]:
```
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 [64]:
```
# total charges are in object should be changed to float
```

In [65]:
```
df.isnull().sum()
```

Out[65]:

	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

In [66]:
```
df.columns = df.columns.str.lower()
```

In [6]:
```
df["totalcharges"] = pd.to_numeric(df["totalcharges"], errors="coerce")
```

In [12]:
```
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      7032 non-null   float64
 20  churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

In [13]:
```
df["seniorcitizen"].value_counts()
```

Out[13]:

seniorcitizen	count
0	5901
1	1142

dtype: int64

In [7]:
```
df["seniorcitizen"] = df["seniorcitizen"].astype("object")
```

In [9]:
```
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 [ ]:
```
df.describe()
```

Out[ ]:

	tenure	monthlycharges	totalcharges
count	7043.000000	7043.000000	7032.000000
mean	32.371149	64.761692	2283.300441
std	24.559481	30.090047	2266.771362
min	0.000000	18.250000	18.800000
25%	9.000000	35.500000	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.850000	3794.737500
max	72.000000	118.750000	8684.800000

In [ ]:
```
df.columns
```

Out[ ]:
```
Index(['customerid', 'gender', 'seniorcitizen', 'partner', 'dependents',
       'tenure', 'phoneservice', 'multiplelines', 'internetservice',
       'onlinesecurity', 'onlinebackup', 'deviceprotection', 'techsupport',
       'streamingtv', 'streamingmovies', 'contract', 'paperlessbilling',
       'paymentmethod', 'monthlycharges', 'totalcharges', 'churn'],
      dtype='object')
```

In [ ]:
```
set(df["churn"])
```

Out[ ]:
```
{'No', 'Yes'}
```

In [8]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [16]:
```
df["gender"].value_counts()
```

Out[16]:

gender	count
Male	3555
Female	3488

dtype: int64

In [17]:
```
df["tenure"].value_counts()
```

Out[17]:

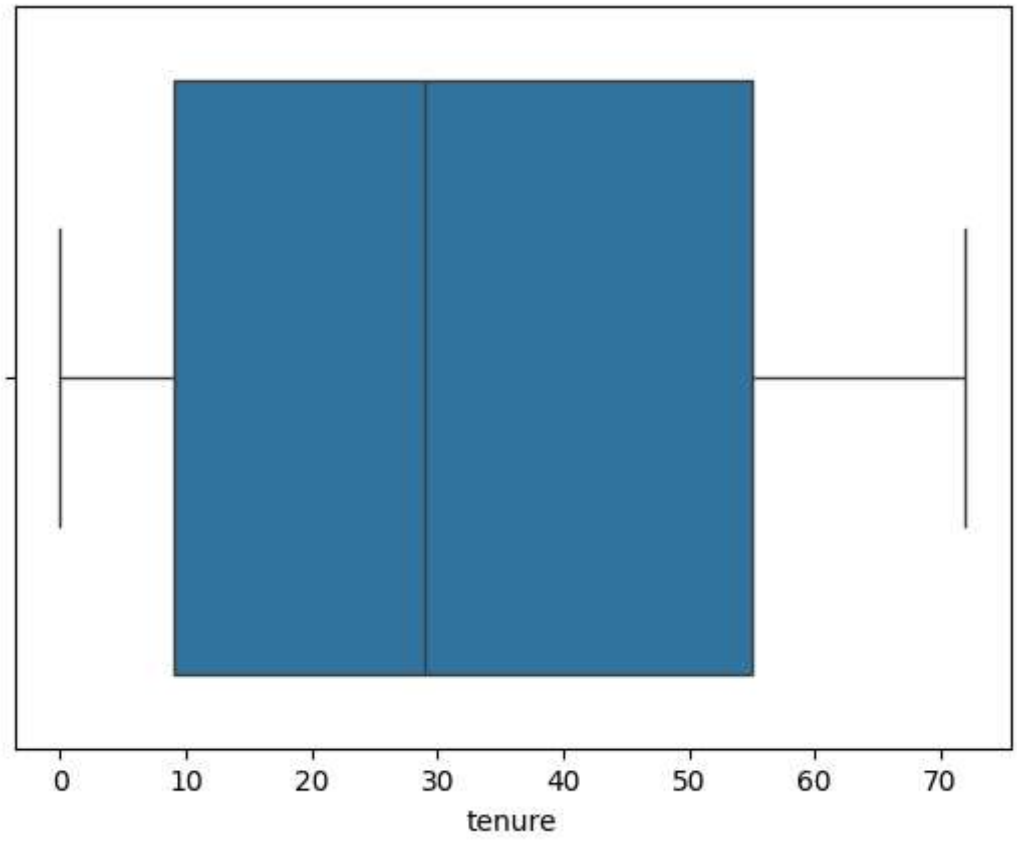
tenure	count
1	613
72	362
2	238
3	200
4	176
...	...
28	57
39	56
44	51
36	50
0	11

73 rows × 1 columns

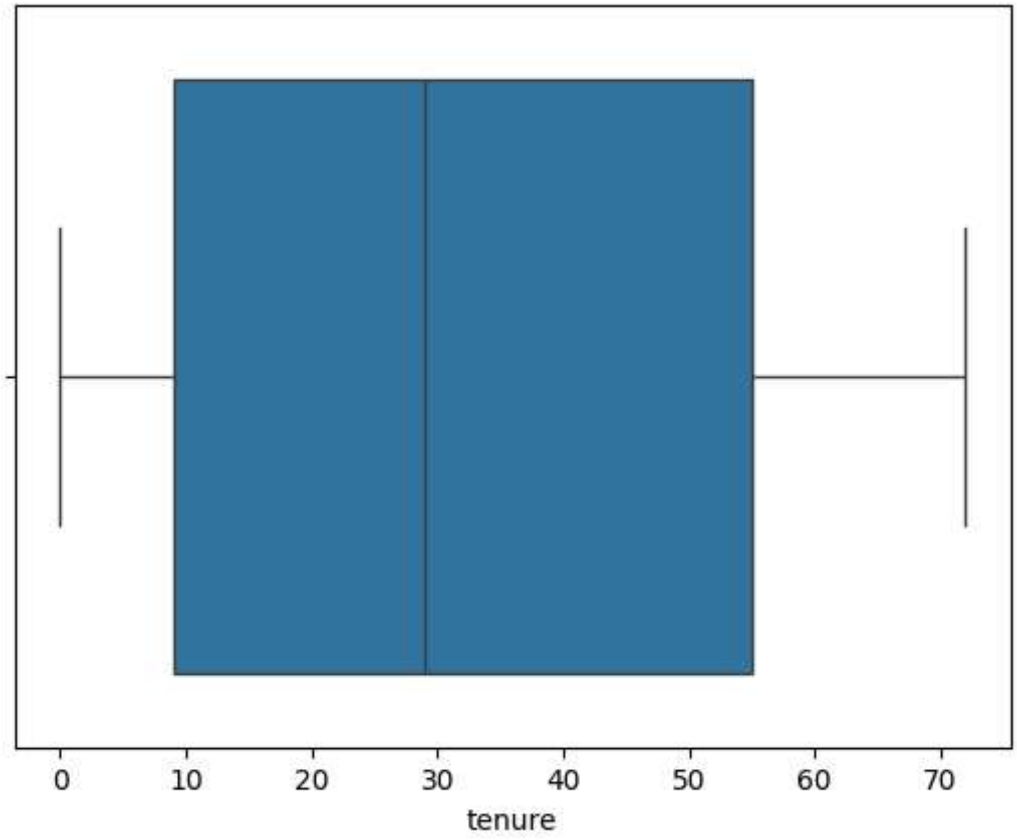
dtype: int64

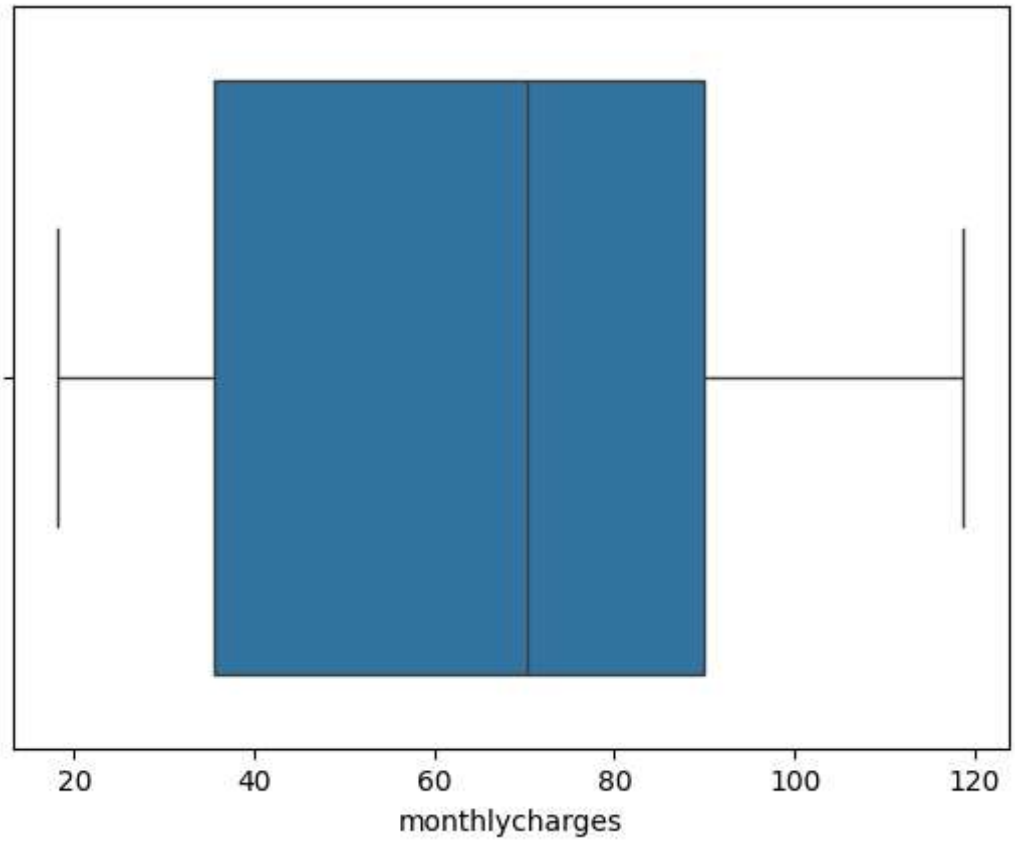
In [18]:
```
sns.boxplot(x=df["tenure"])
```

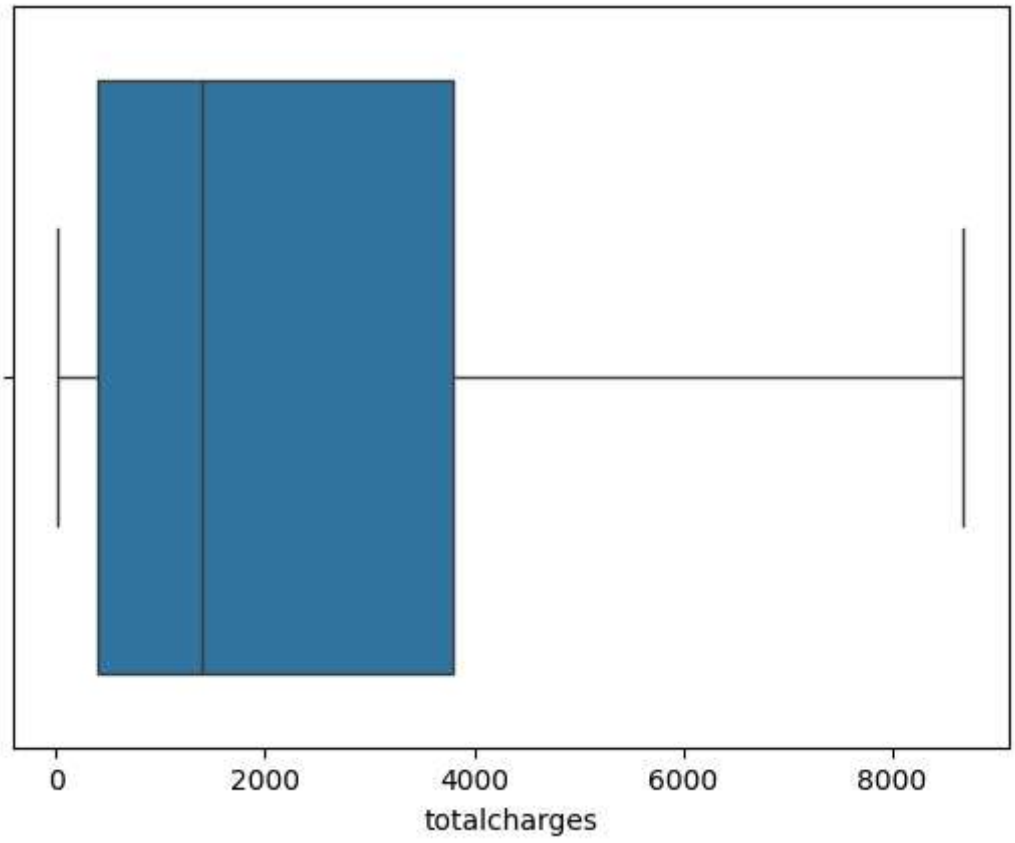
Out[18]:
```
<Axes: xlabel='tenure'>
```



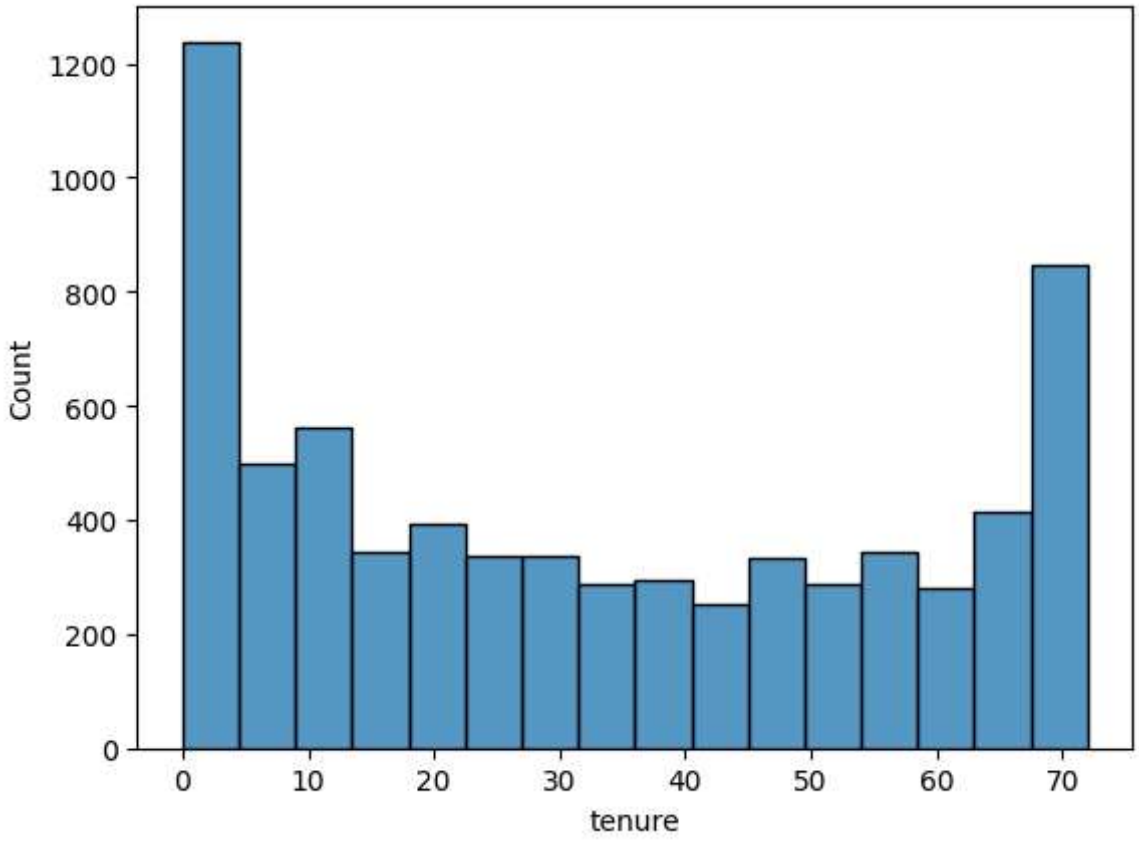
In [19]:
```
for col in df.columns:
  if df[col].dtype == "object":
    continue
  sns.boxplot(x=df[col])
  plt.show()
```

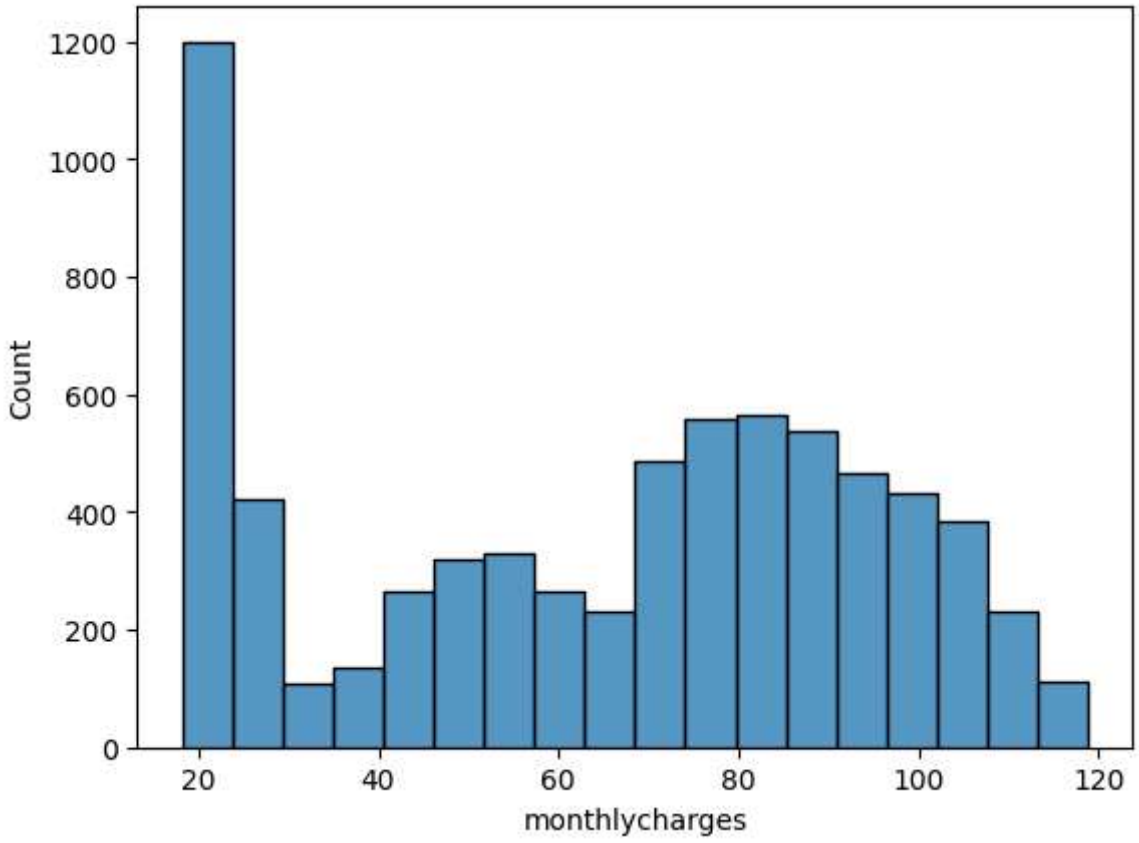


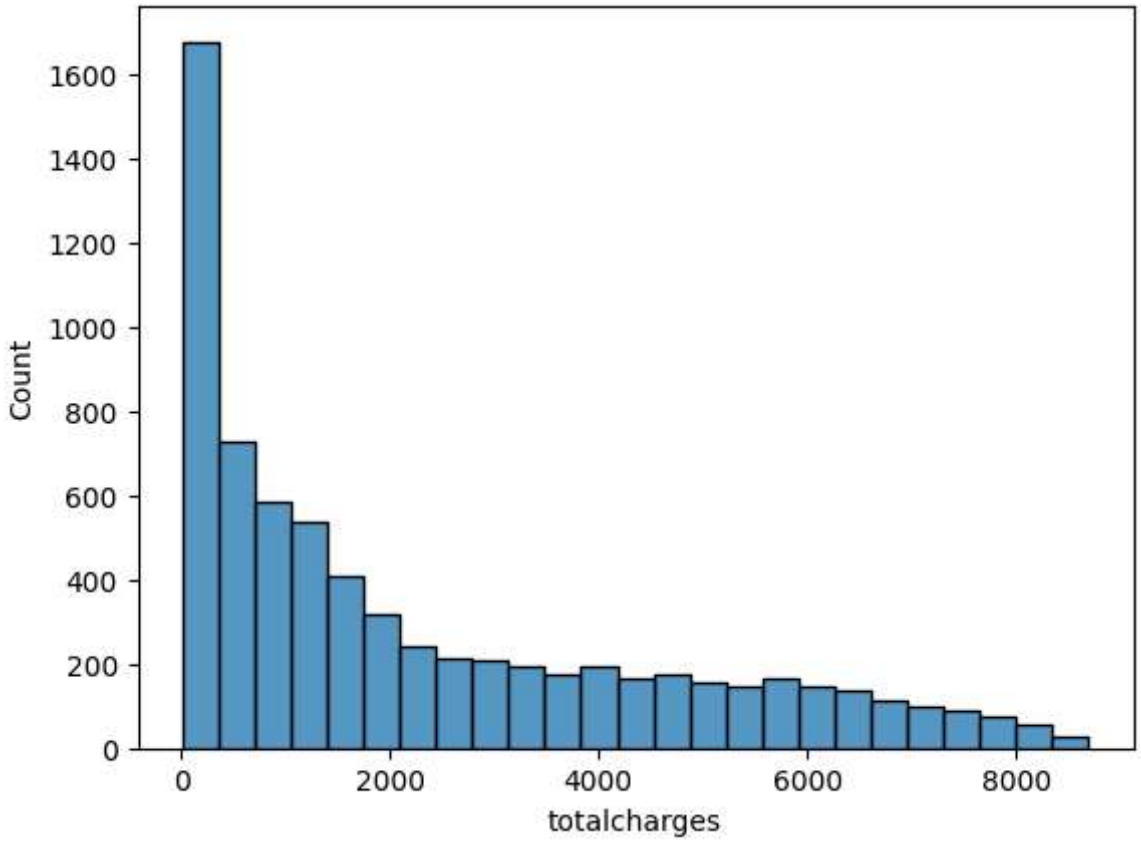




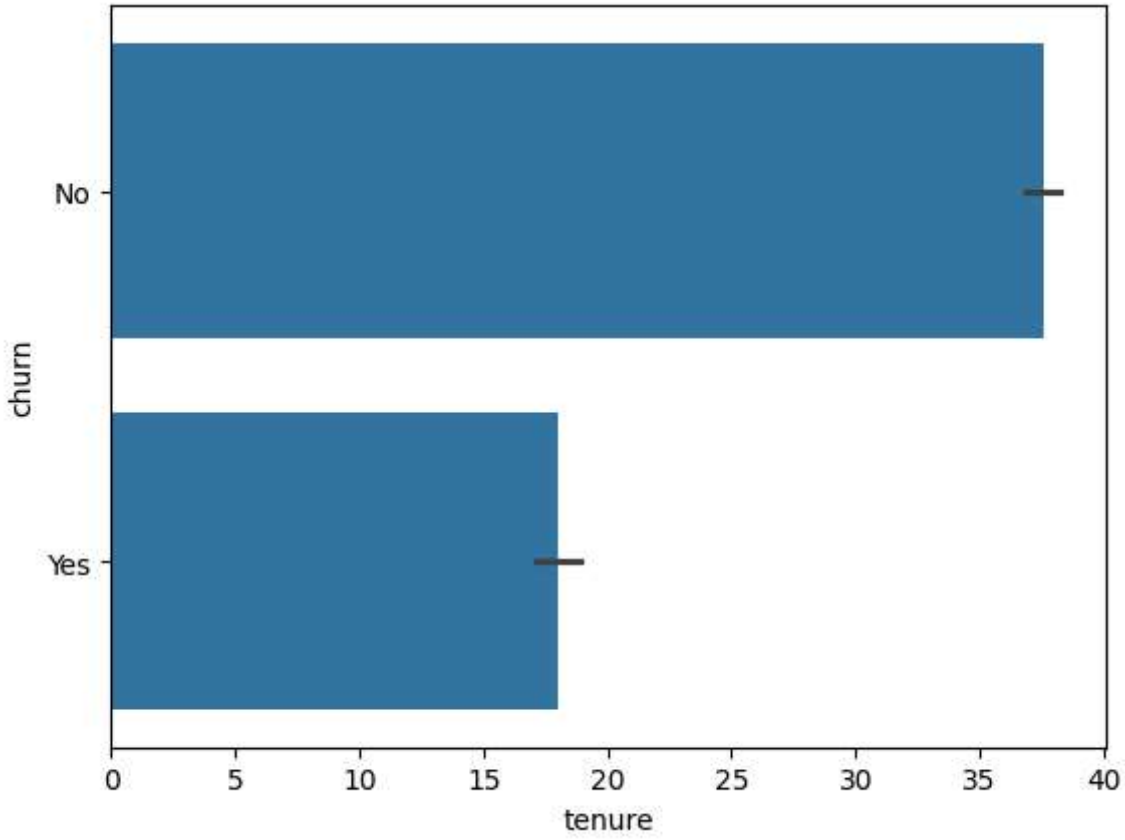
In [ ]:
```
for col in df.columns:
  if df[col].dtype == "object":
    continue
  sns.histplot(x=df[col])
  plt.show()
```

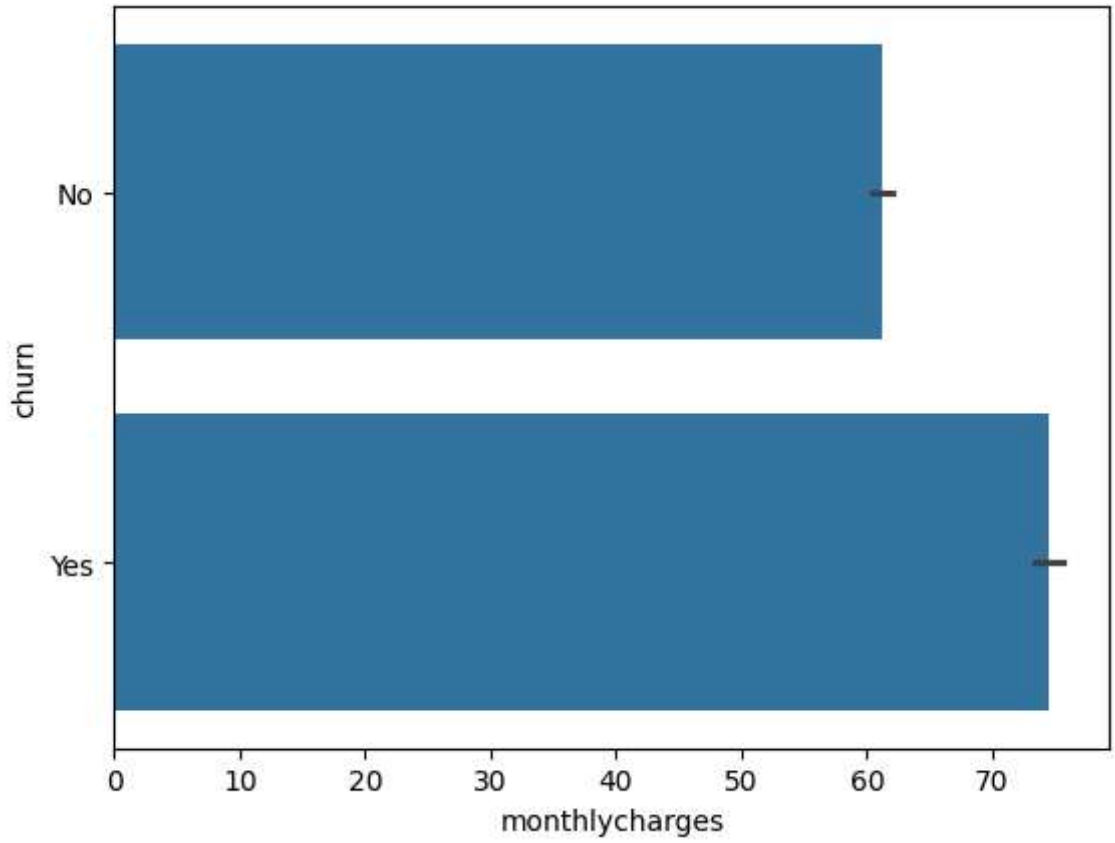

Count
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60
70
tenure

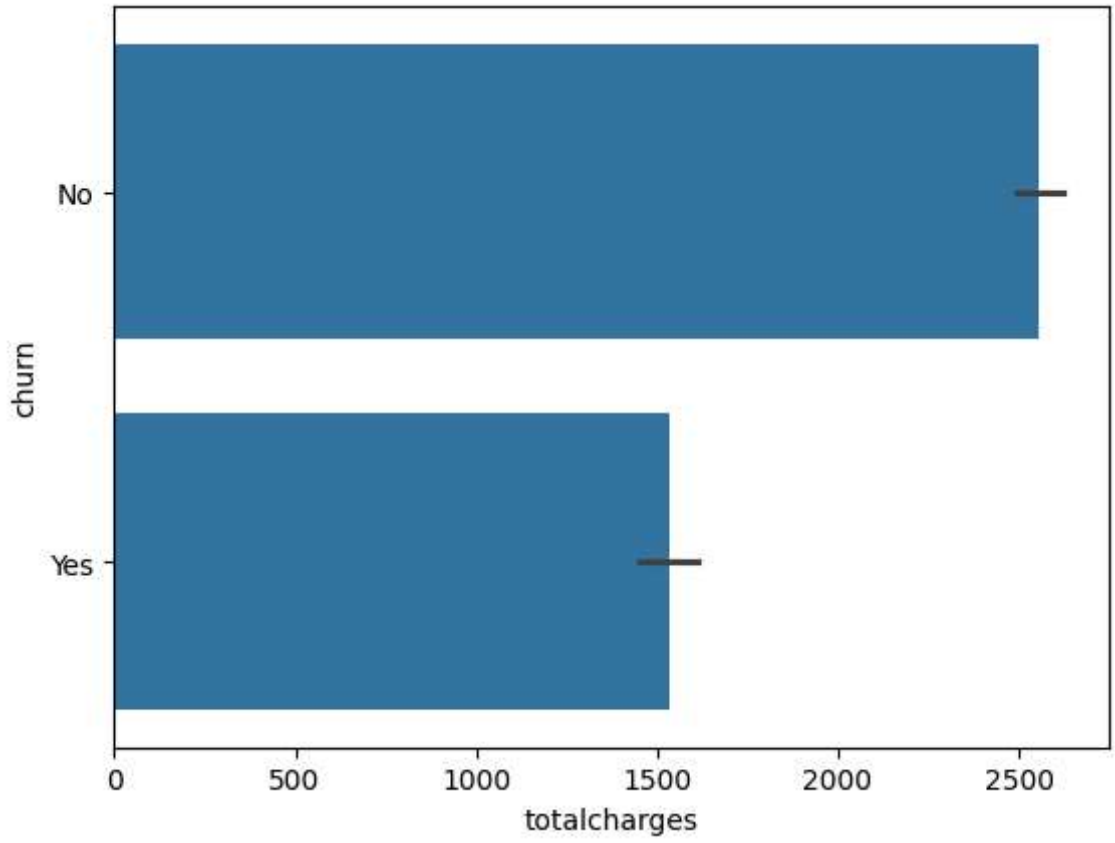

Count
1200
1000
800
600
400
200
0
20
40
60
80
100
120
monthlycharges



In [ ]:
```
for col in df.columns:
  if df[col].dtype == "object":
    continue
  sns.barplot(x=df[col], y=df["churn"])
  plt.show()
```

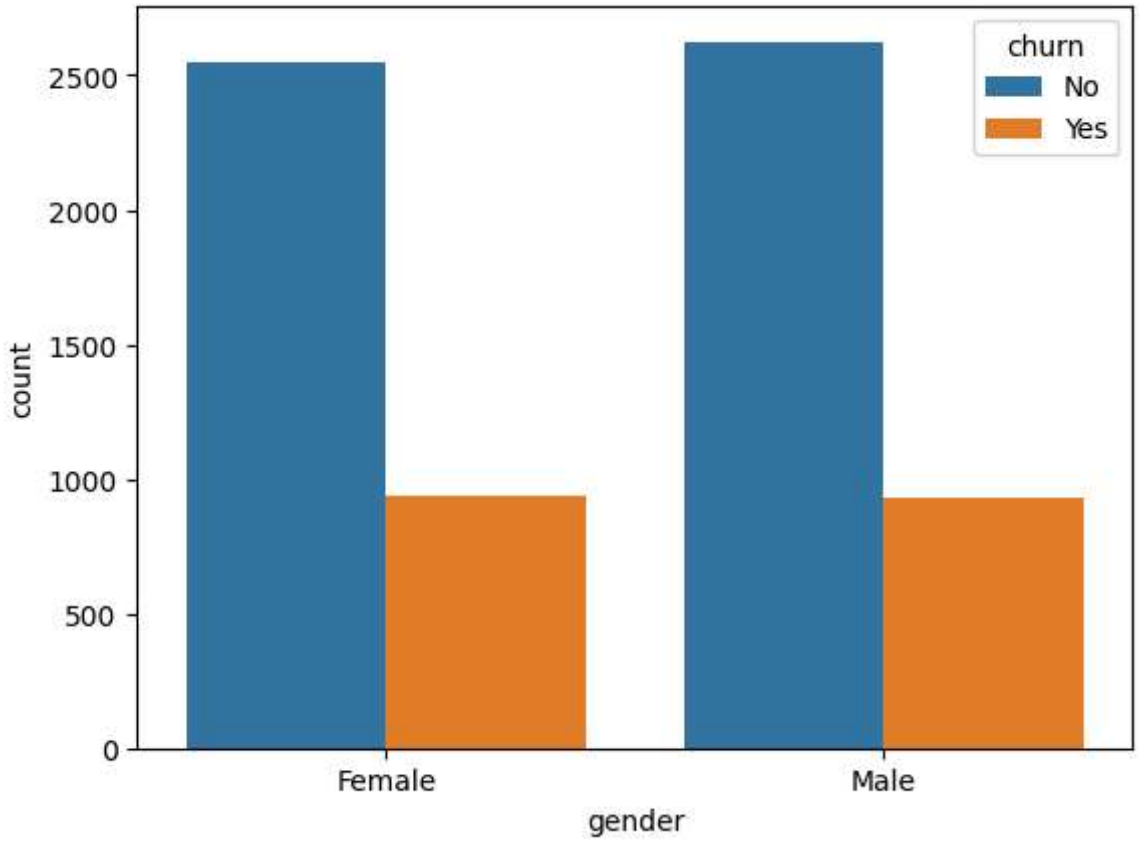






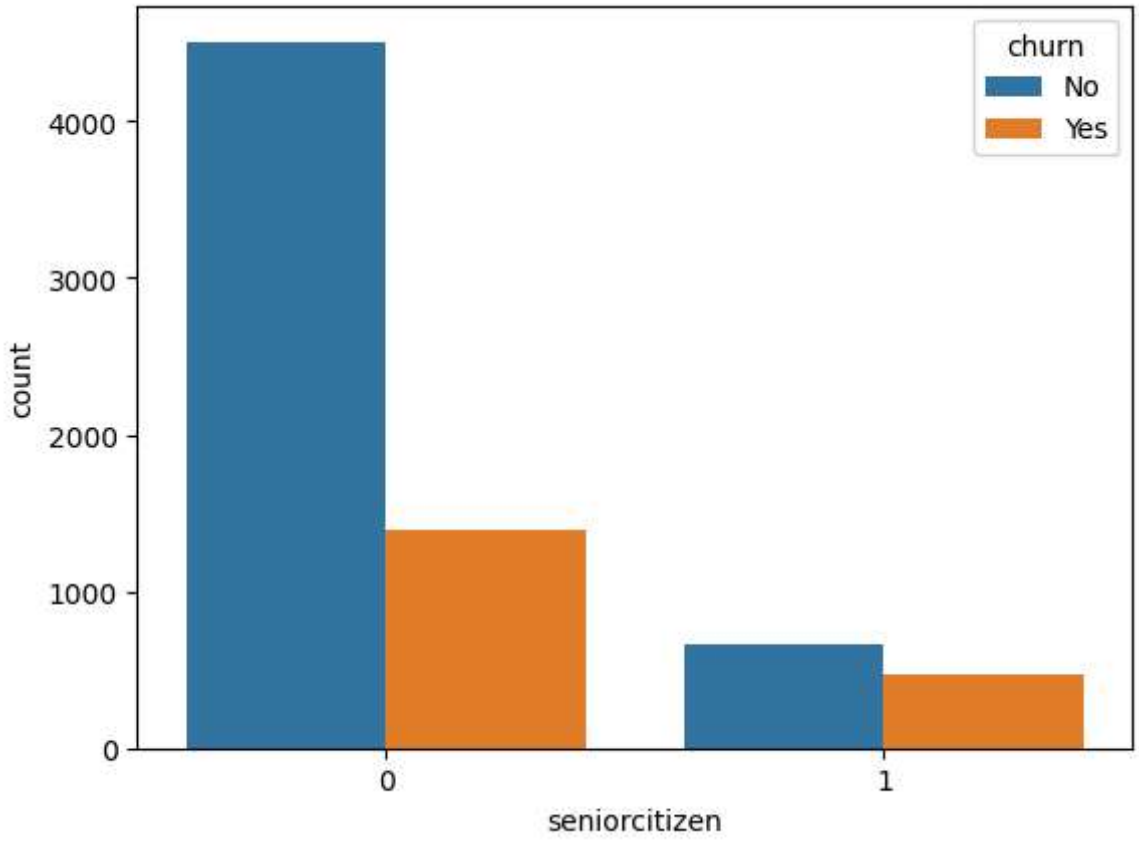
In [ ]:
```
sns.countplot(x=df["gender"], hue=df["churn"])
```

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



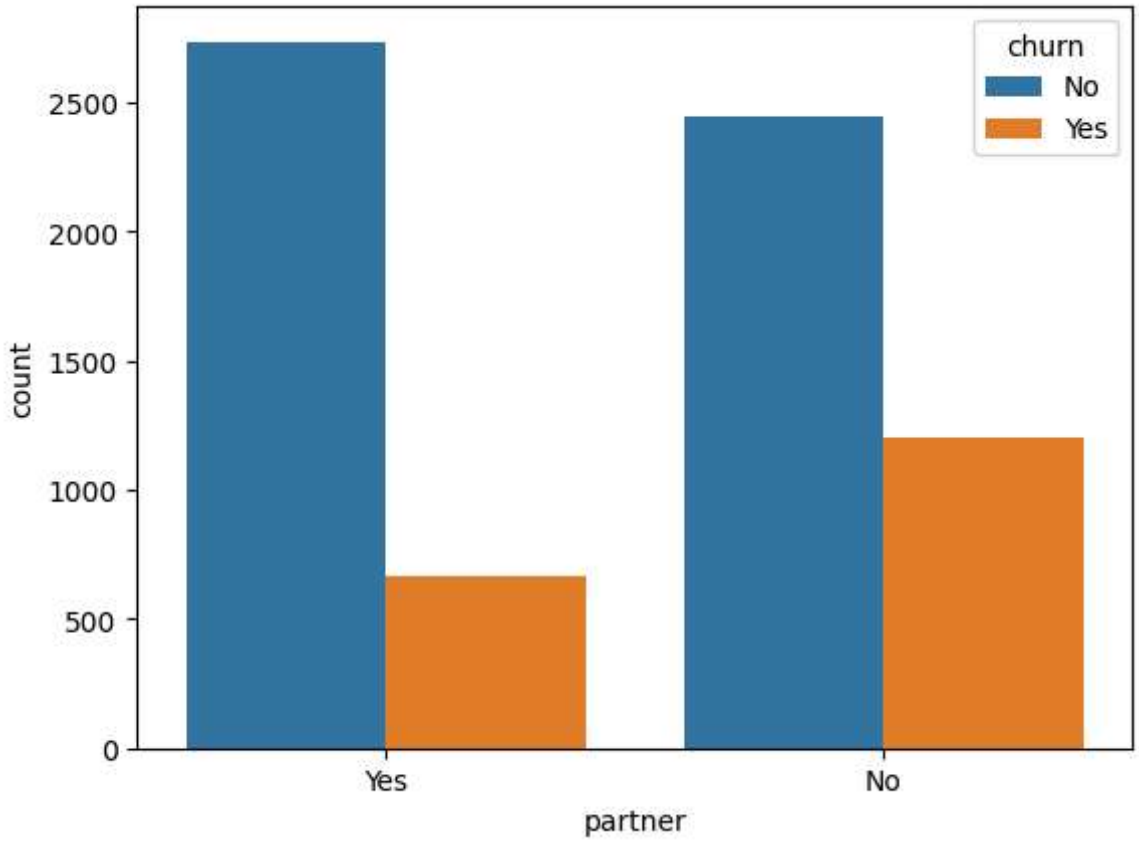
In [63]:
```
sns.countplot(x=df["seniorcitizen"], hue=df["churn"])
```

Out[63]:
```
<Axes: xlabel='seniorcitizen', ylabel='count'>
```



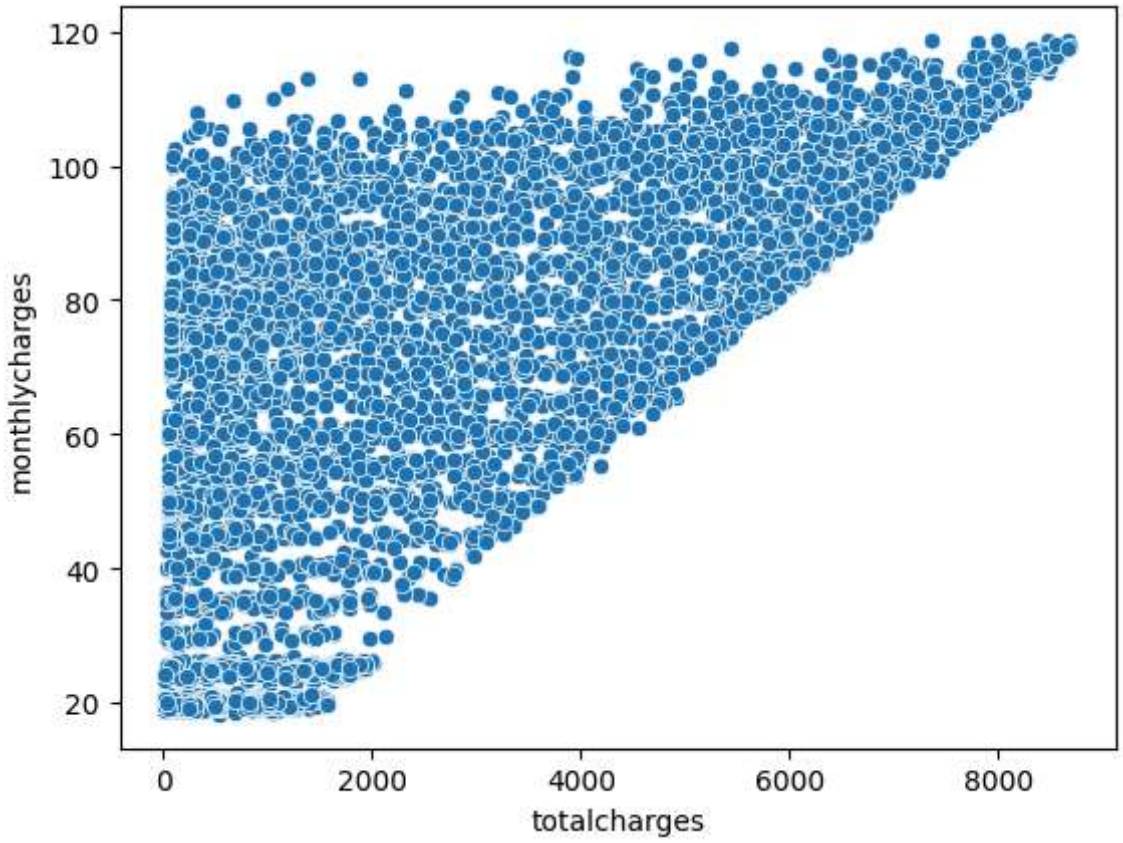
In [ ]:
```
sns.countplot(x=df["partner"], hue=df["churn"])
```

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



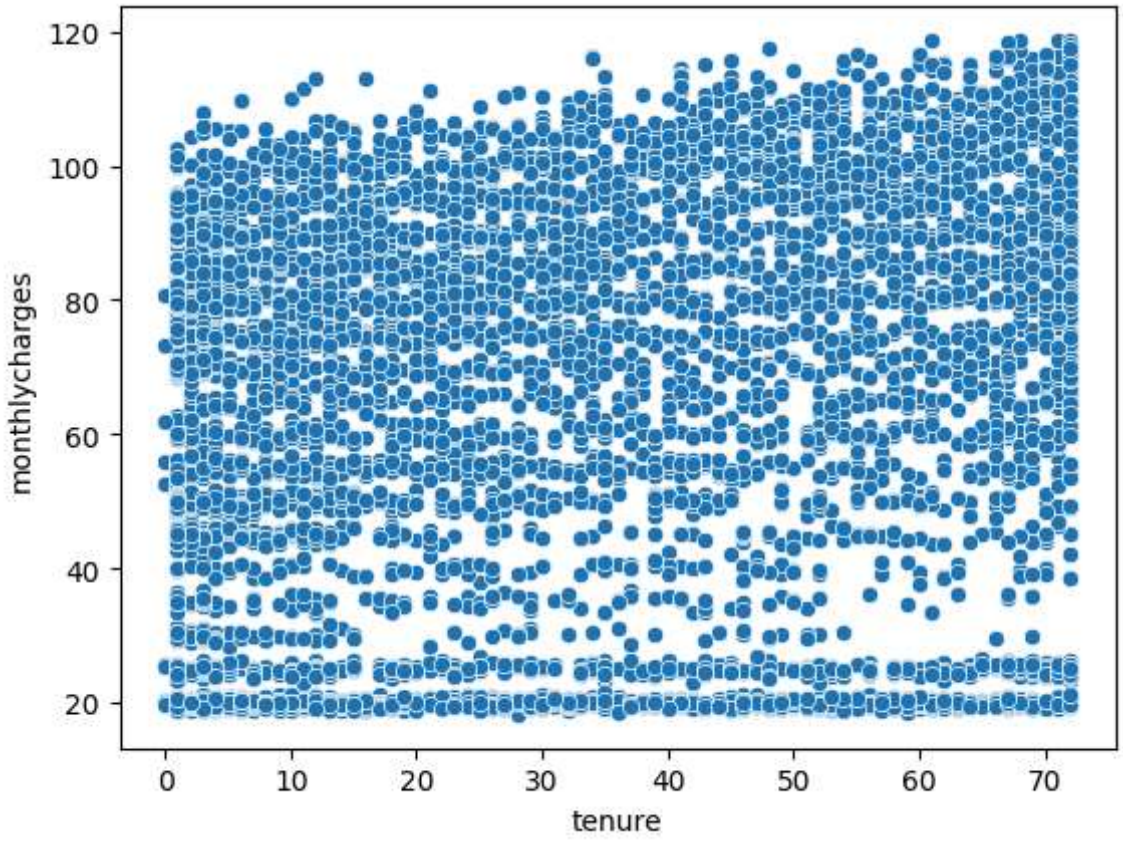
In [ ]:
```
sns.scatterplot(x=df["totalcharges"], y=df["monthlycharges"])
```

Out[ ]:
```
<Axes: xlabel='totalcharges', ylabel='monthlycharges'>
```



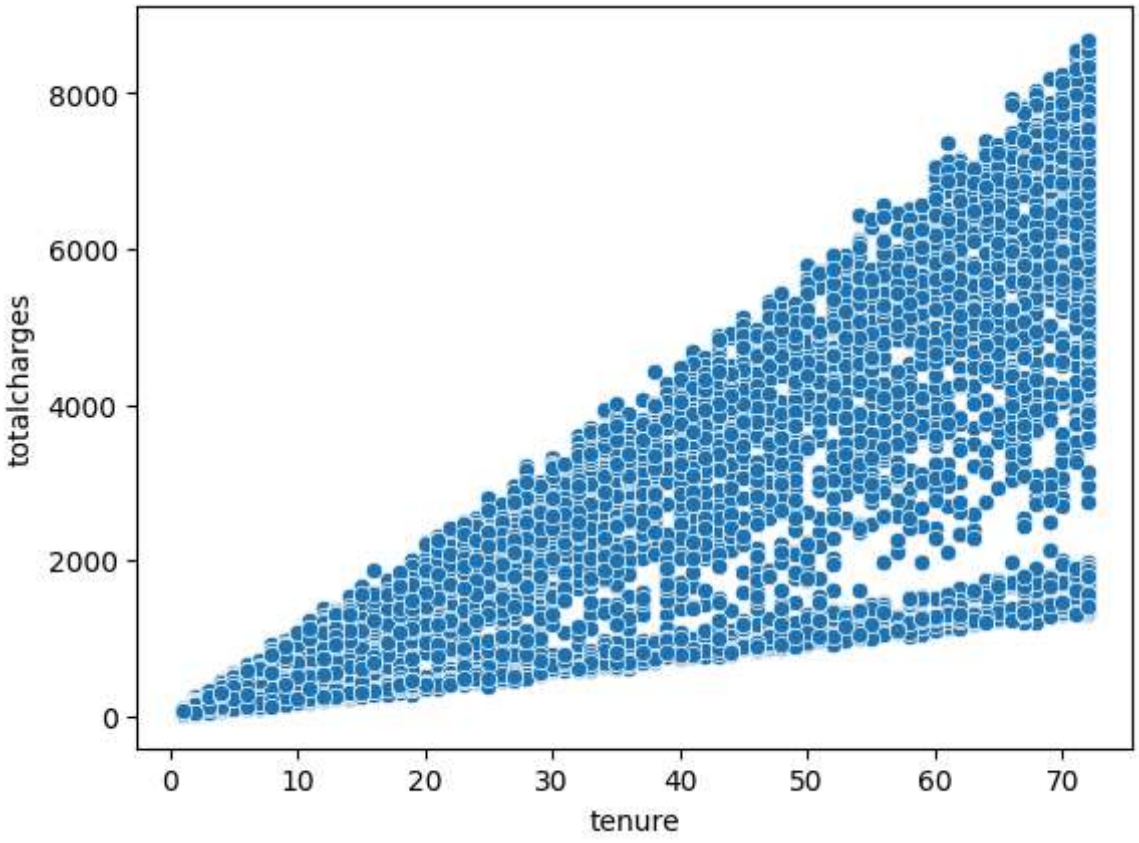
In [ ]:
```
sns.scatterplot(x=df["tenure"], y=df["monthlycharges"])
```

Out[ ]:
```
<Axes: xlabel='tenure', ylabel='monthlycharges'>
```



In [ ]:
```
sns.scatterplot(x=df["tenure"], y=df["totalcharges"])
```

Out[ ]: <Axes: xlabel='tenure', ylabel='totalcharges'>



In [20]:
```
sns.heatmap(df.corr(numeric_only=True))
```

Out[20]: <Axes: >

In [21]:
```
from sklearn.model_selection import train_test_split
```

In [22]:
```
df = df.drop("customerid",axis=1)
```

In [ ]:
```
x = df.drop("churn",axis=1)
y = df["churn"]
```

In [ ]:
```
df["paymentmethod"].value_counts()
```

Out[ ]:

paymentmethod	count
Electronic check	2365
Mailed check	1612
Bank transfer (automatic)	1544
Credit card (automatic)	1522

dtype: int64

In [27]:
```
df.drop("customerid",axis=1,inplace=True)
```

In [24]:
```
x.head()
```

Out[24]:

	customerid	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multi
0	7590-VHVEG	Female	0	Yes	No	1	No	N
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	N
4	9237-HQITU	Female	0	No	No	2	Yes	

In [43]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score,f1_score,roc_auc_scor
x = df.drop("churn",axis=1)
y = df["churn"]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=
scaler = StandardScaler()
label = LabelEncoder()
x_train[["tenure","monthlycharges","totalcharges"]] = scaler.fit_transform(x_tra
x_test[["tenure","monthlycharges","totalcharges"]] = scaler.transform(x_test[["t
for i in x_train.columns:
  if x_train[i].dtype == "object":
    x_train[i] = label.fit_transform(x_train[i])
    x_test[i] = label.transform(x_test[i])
y_train = label.fit_transform(y_train)
y_test = label.transform(y_test)
```

In [35]:

```
x_train.head()
```

Out[35]:

	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multipleline
4922	0	0	1	1	1.173615	1	
2512	0	0	1	0	-0.377248	1	
5816	0	0	0	0	-1.193493	1	
6418	1	0	1	0	-0.050751	1	
131	0	0	1	1	-0.295624	0	

In [46]:
```
x_train["totalcharges"] =x_train["totalcharges"].fillna(x_train["totalcharges"].
x_test["totalcharges"] =x_test["totalcharges"].fillna(x_test["totalcharges"].med
```

In [42]:
```
log = LogisticRegression()
log.fit(x_train,y_train)
y_pred = log.predict(x_test)
print("log_precision",precision_score(y_test,y_pred))
print("log_f1",f1_score(y_test,y_pred))
print("log_accuracy",accuracy_score(y_test,y_pred))
```

```
log_precision 0.6470588235294118
log_f1 0.6009104704097117
log_accuracy 0.8133427963094393
```

In [44]:
```
rnd = RandomForestClassifier()
rnd.fit(x_train,y_train)
y_pred = rnd.predict(x_test)
print("rnd_precision",precision_score(y_test,y_pred))
print("rnd_f1",f1_score(y_test,y_pred))
print("rnd_accuracy",accuracy_score(y_test,y_pred))
```

```
rnd_precision 0.6254416961130742
rnd_f1 0.5566037735849056
rnd_accuracy 0.7998580553584103
```

In [47]:
```
knn = KNeighborsClassifier()
knn.fit(x_train,y_train)
y_pred = knn.predict(x_test)
print("knn_precision",precision_score(y_test,y_pred))
print("knn_f1",f1_score(y_test,y_pred))
print("knn_accuracy",accuracy_score(y_test,y_pred))
```

```
knn_precision 0.5626911314984709
knn_f1 0.5411764705882353
knn_accuracy 0.7785663591199432
```

In [48]:
```
svc = SVC()
svc.fit(x_train,y_train)
y_pred = svc.predict(x_test)
print("svc_precision",precision_score(y_test,y_pred))
print("svc_f1",f1_score(y_test,y_pred))
print("svc_accuracy",accuracy_score(y_test,y_pred))
```

```
svc_precision 0.6829268292682927
svc_f1 0.5609348914858097
svc_accuracy 0.8133427963094393
```

In [49]:
```
from xgboost import XGBClassifier
xgb  = XGBClassifier()
xgb.fit(x_train,y_train)
y_pred = xgb.predict(x_test)
print("xgb_precision",precision_score(y_test,y_pred))
print("xgb_f1",f1_score(y_test,y_pred))
print("xgb_accuracy",accuracy_score(y_test,y_pred))
```

```
xgb_precision 0.5876923076923077
xgb_f1 0.5634218289085545
xgb_accuracy 0.7899219304471257
```

In [50]:
```
dct = DecisionTreeClassifier()
dct.fit(x_train,y_train)
```

```
y_pred = dct.predict(x_test)
print("dct_precision",precision_score(y_test,y_pred))
print("dct_f1",f1_score(y_test,y_pred))
print("dct_accuracy",accuracy_score(y_test,y_pred))
```

```
dct_precision 0.5091383812010444
dct_f1 0.529891304347826
dct_accuracy 0.7544357700496807
```

Conclusion :

- best algorithm is `logistic regression` and `support vector classifier` based on accuracy score

  which is having both `81.3 %`

- but `logistic regression` makes more reliable choice based on other evalution score

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: