# Goal : Create a model to predict whether or not a customer will Churn .

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

In [2]:
```
# Loading data
df = pd.read_csv(r'Telco-Customer-Churn.csv')
```

# Data Exploration

In [3]:
```
df.shape
```

Out[3]: (7032, 21)

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

Out[4]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleL |
|---|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No pl se |
| 1 | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | |
| 2 | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | |
| 3 | 7795-CFOCW | Male | 0 | No | No | 45 | No | No pl se |
| 4 | 9237-HQITU | Female | 0 | No | No | 2 | Yes | |

5 rows × 21 columns

In [5]:
```
df.tail()
```

In [5]:

```
df.tail()
```

Out[5]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | Multi |
|---|---|---|---|---|---|---|---|---|
| **7027** | 6840-RESVB | Male | 0 | Yes | Yes | 24 | Yes | |
| **7028** | 2234-XADUH | Female | 0 | Yes | Yes | 72 | Yes | |
| **7029** | 4801-JZAZL | Female | 0 | Yes | Yes | 11 | No | N |
| **7030** | 8361-LTMKD | Male | 1 | Yes | No | 4 | Yes | |
| **7031** | 3186-AJIEK | Male | 0 | No | No | 66 | Yes | |

5 rows × 21 columns

In [6]:

```
df.sample(5)
```

Out[6]:

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | Multi |
|---|---|---|---|---|---|---|---|---|
| **811** | 3551-GAEGL | Male | 0 | Yes | Yes | 34 | No | N |
| **4923** | 4298-OYIFC | Male | 0 | Yes | No | 15 | Yes | |
| **6947** | 3078-ZKNTS | Female | 0 | Yes | Yes | 13 | Yes | |
| **2664** | 4659-NZRUF | Female | 0 | No | No | 19 | Yes | |
| **5183** | 5982-PSMKW | Female | 0 | Yes | Yes | 23 | Yes | |

5 rows × 21 columns

## Quick Data Check

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7032 entries, 0 to 7031
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7032 non-null   object 
 1   gender            7032 non-null   object 
 2   SeniorCitizen     7032 non-null   int64  
 3   Partner           7032 non-null   object 
 4   Dependents        7032 non-null   object 
 5   tenure            7032 non-null   int64  
 6   PhoneService      7032 non-null   object 
 7   MultipleLines     7032 non-null   object 
 8   InternetService   7032 non-null   object 
 9   OnlineSecurity    7032 non-null   object 
 10  OnlineBackup      7032 non-null   object 
 11  DeviceProtection  7032 non-null   object 
 12  TechSupport       7032 non-null   object 
 13  StreamingTV       7032 non-null   object 
 14  StreamingMovies   7032 non-null   object 
 15  Contract          7032 non-null   object 
 16  PaperlessBilling  7032 non-null   object 
 17  PaymentMethod     7032 non-null   object 
 18  MonthlyCharges    7032 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7032 non-null   object 
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
```

## Statistical summary

In [8]:
```
df.describe()
```

Out[8]:

| | SeniorCitizen | tenure | MonthlyCharges | TotalCharges |
|---|---|---|---|---|
| **count** | 7032.000000 | 7032.000000 | 7032.000000 | 7032.000000 |
| **mean** | 0.162400 | 32.421786 | 64.798208 | 2283.300441 |
| **std** | 0.368844 | 24.545260 | 30.085974 | 2266.771362 |
| **min** | 0.000000 | 1.000000 | 18.250000 | 18.800000 |
| **25%** | 0.000000 | 9.000000 | 35.587500 | 401.450000 |
| **50%** | 0.000000 | 29.000000 | 70.350000 | 1397.475000 |
| **75%** | 0.000000 | 55.000000 | 89.862500 | 3794.737500 |
| **max** | 1.000000 | 72.000000 | 118.750000 | 8684.800000 |

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

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

Out[9]:
```
customerID          7032
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                72
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1584
TotalCharges        6530
Churn                  2
dtype: int64
```

# Part 2: Exploratory Data Analysis

## General Feature Exploration

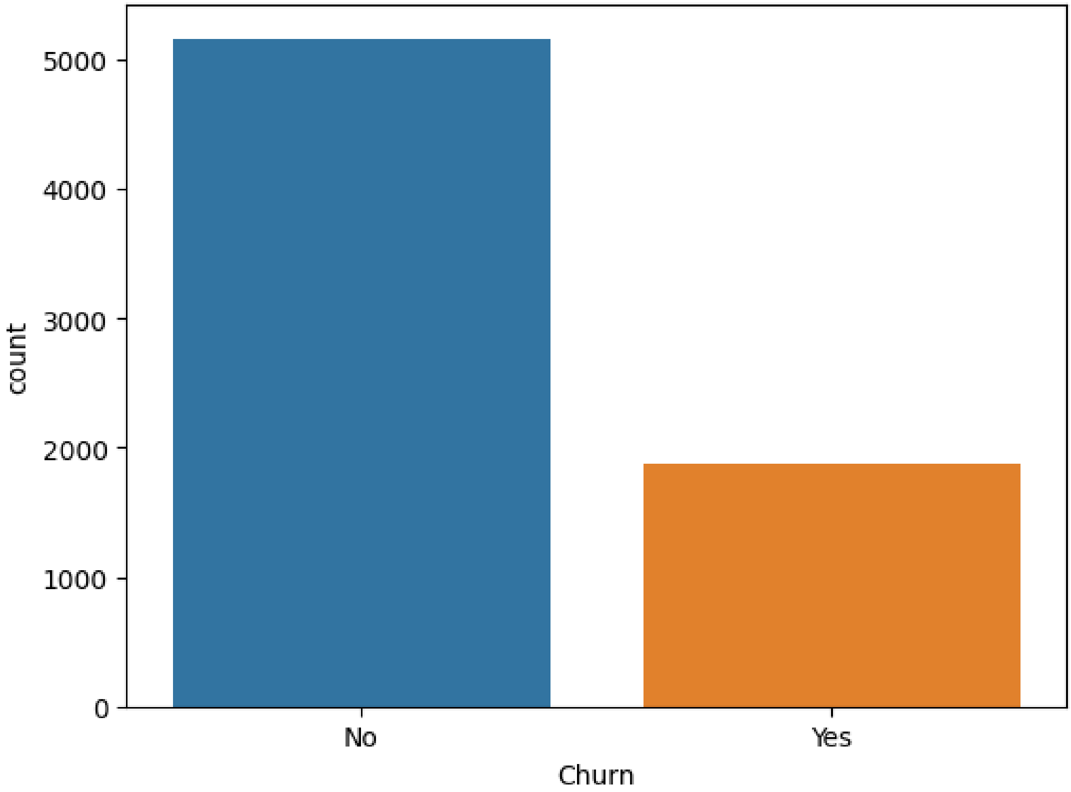
Confirming that there are no NaN cells by displaying NaN values per feature column.

In [10]:
```
df.isna().sum()
```

Out[10]:
```
customerID          0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

### Checking the balance of the class labels (Churn) with a Count Plot.

In [11]:
```
sns.countplot(data=df, x='Churn',hue='Churn');
```




- The churn rate is 28%. This is a significant percentage of customers, and it is important to understand why these customers are churning.

In [12]:
```
df.value_counts('Churn')
```

Out[12]:
```
Churn
No     5163
Yes    1869
Name: count, dtype: int64
```

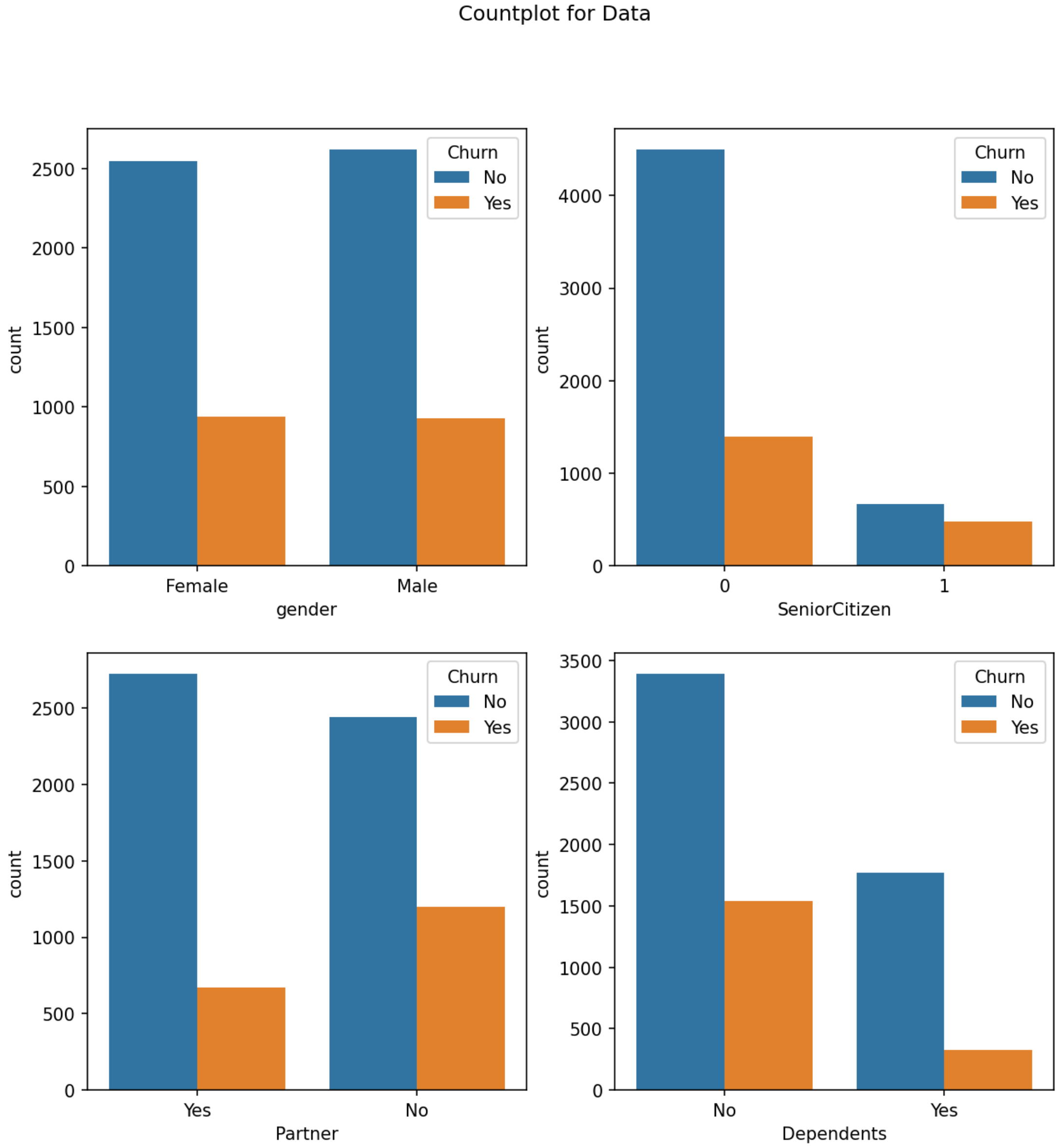
In [13]:
```
df.value_counts('SeniorCitizen')
```

Out[13]:
```
SeniorCitizen
0    5890
1    1142
Name: count, dtype: int64
```

In [14]:
```
fig, axes = plt.subplots(2, 2, figsize=(10, 10),dpi=150)
```

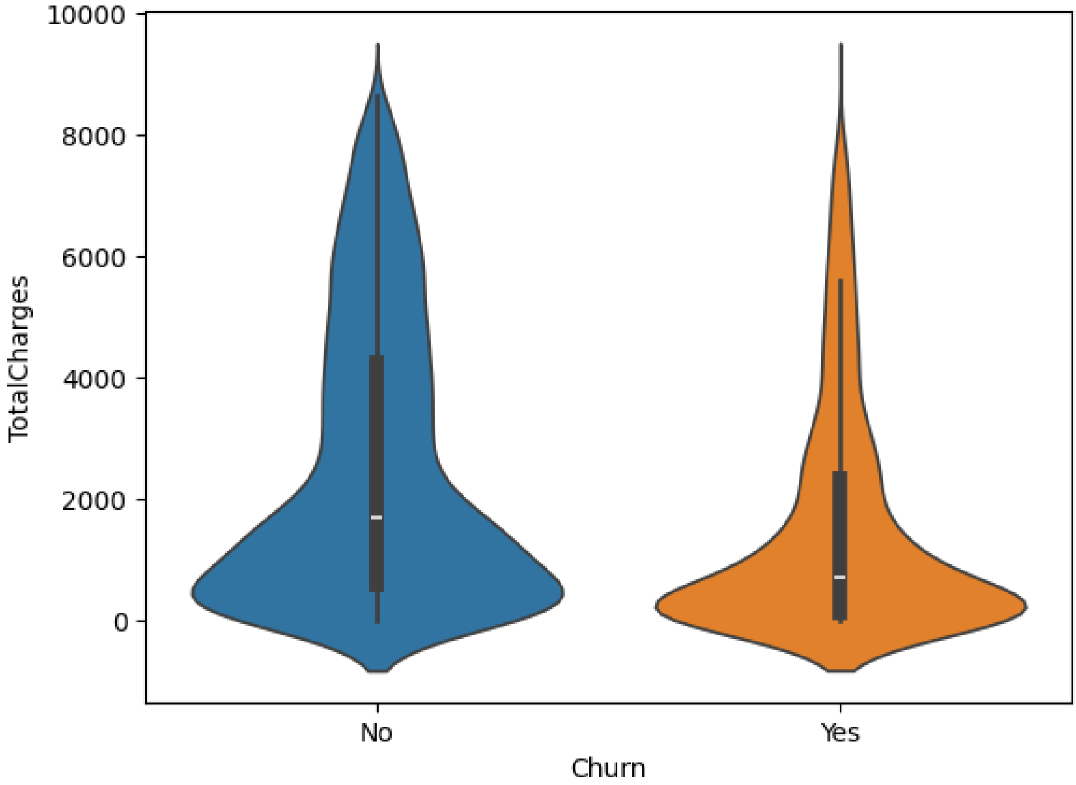
In [14]:

```
fig, axes = plt.subplots(2, 2, figsize=(10, 10),dpi=150)

fig.suptitle('Countplot for Data')

sns.countplot(ax=axes[0, 0], data=df, x='gender',hue='Churn')
sns.countplot(ax=axes[0, 1], data=df, x='SeniorCitizen',hue='Churn')
sns.countplot(ax=axes[1, 0], data=df, x='Partner',hue='Churn')
sns.countplot(ax=axes[1, 1], data=df, x='Dependents',hue='Churn');
```



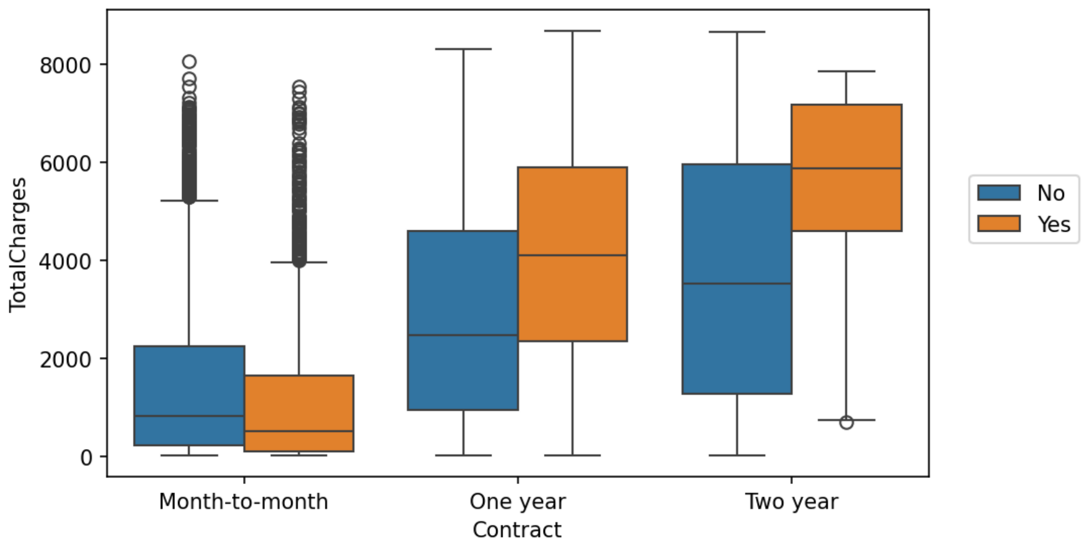
### Exploring the distrbution of TotalCharges between Churn categories with a Box Plot or Violin Plot.**

In [15]:
```python
sns.violinplot(data=df,x='Churn',y='TotalCharges',hue='Churn');
```




### Creating a boxplot to display the distribution of TotalCharges per Contract type,

In [16]:
```python
plt.figure(figsize=(7,4),dpi=150)
sns.boxplot(data=df, x='Contract',y='TotalCharges',hue='Churn')
plt.legend(loc=(1.05,0.5));
```



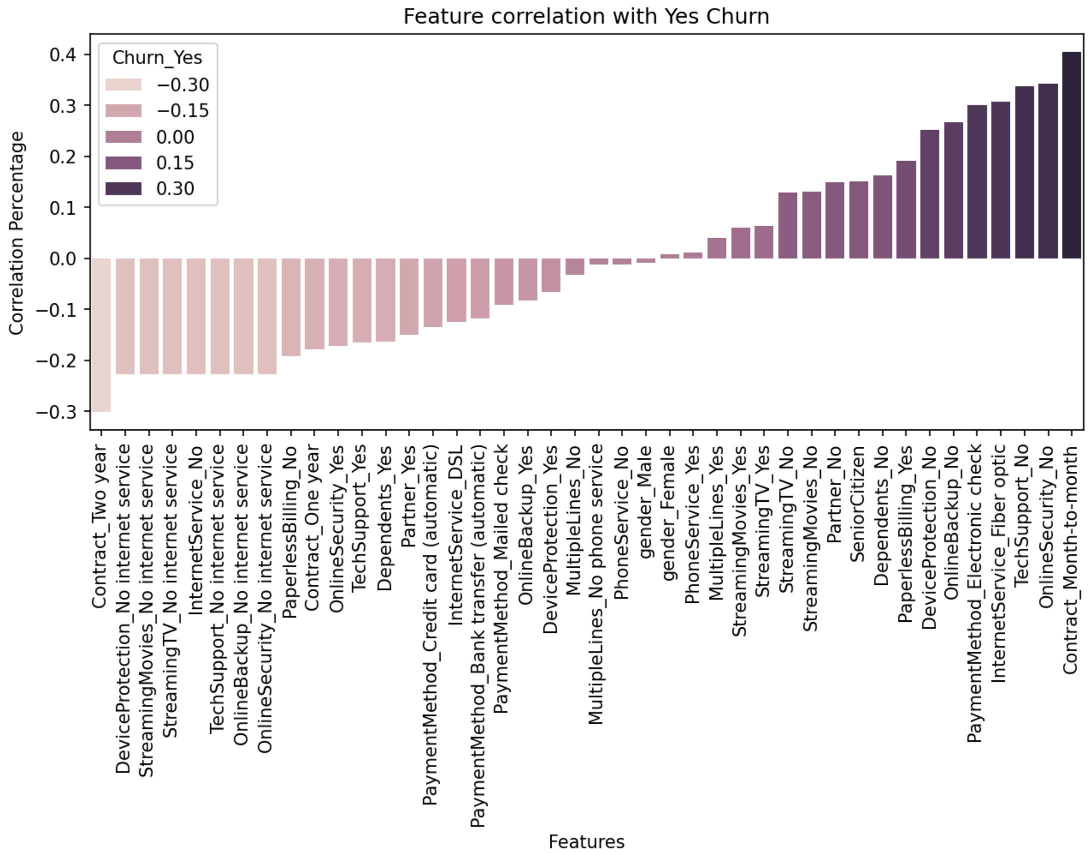

- On Month to Month Plan, we can say those customers are trying services but at the same time. We need to analyze One-year and Two-year contract customers. We can clearly see that customers who churn are paying slightly higher charges than customers who do not time. We need to analyze One-year and Two-year contract customers. We can clearly see that customers who churn are paying slightly higher charges than customers who do not churn. After one-year or two-year contracts, people are more likely to churn if they are having more total charge.

### Creating a bar plot to showcase the correlation of the following features

In [17]:
```
corr_df = pd.get_dummies(df[['gender', 'SeniorCitizen', 'Partner', 'Depend
 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'Int
   'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'Paym
```

In [18]:
```
corr_yes_churn = corr_df['Churn_Yes'].sort_values().iloc[1:-1]
```

In [19]:
```
plt.figure(figsize=(10,4),dpi=150)
plt.title('Feature correlation with Yes Churn')
plt.xlabel('Features')
plt.ylabel('Correlation Percentage')
sns.barplot(x=corr_yes_churn.index,y=corr_yes_churn.values,hue=corr_yes_ch
plt.xticks(rotation=90);
```




# Part 3: Churn Analysis

**This section focuses on segementing customers based on their tenure, creating "cohorts", allowing us to examine differences between customer cohort segments.**

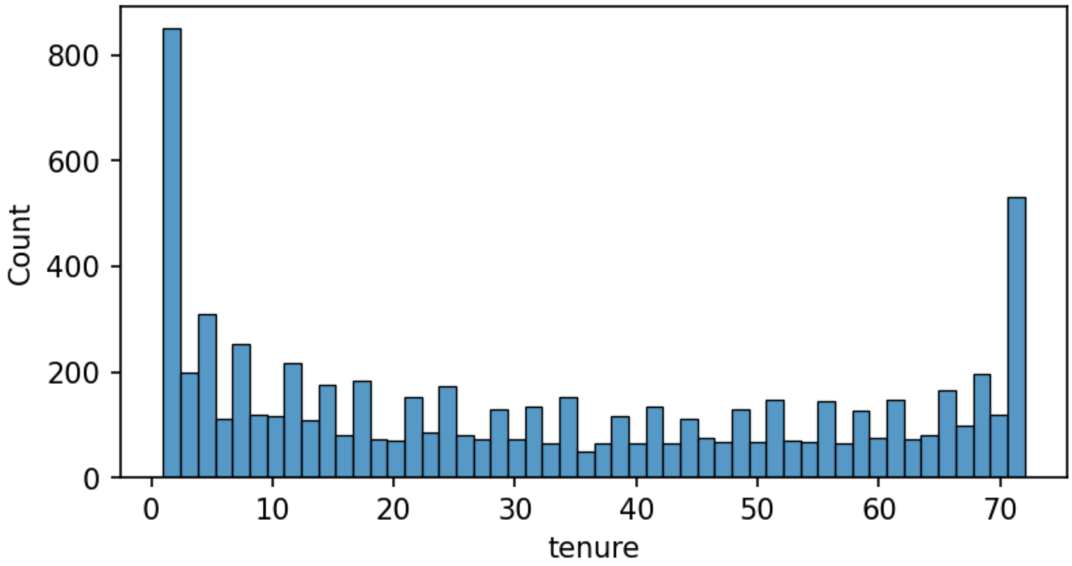
### Checking available contract types

In [20]:
```
df['Contract'].unique()
```

Out[20]:
```
array(['Month-to-month', 'One year', 'Two year'], dtype=object)
```

### Creating a histogram for displaying the distribution of 'tenure' column, which is the amount of months a customer was or has been on a customer.

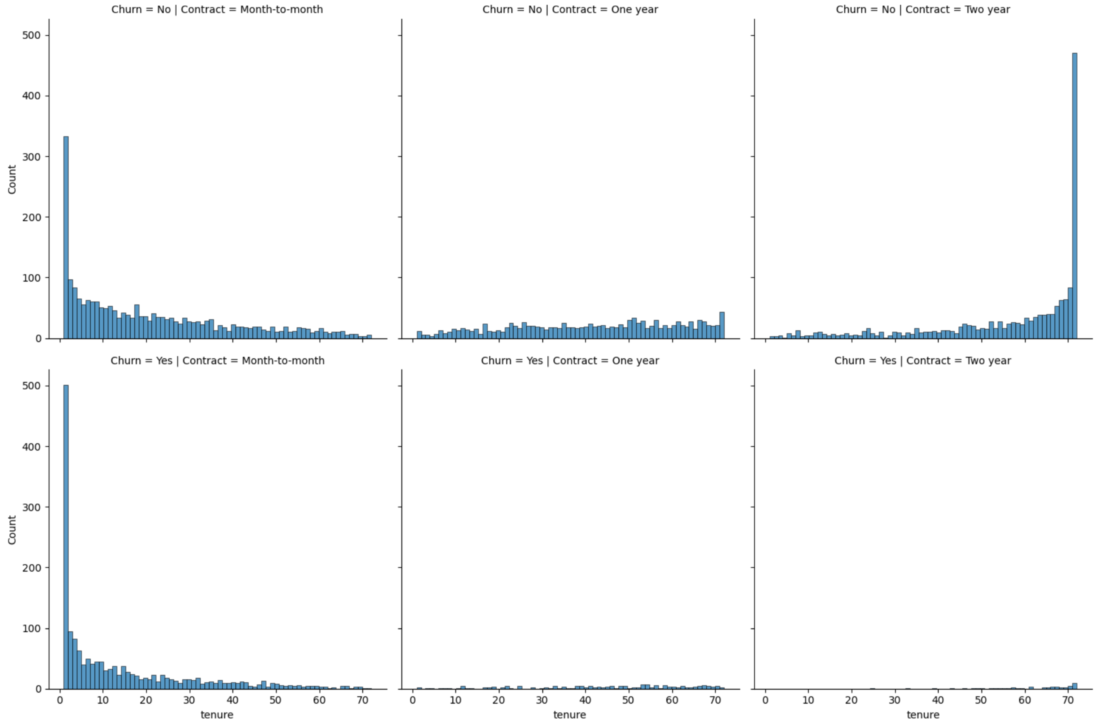
In [21]:
```
plt.figure(figsize=(6,3),dpi=150)
sns.histplot(data=df,x='tenure',bins=50);
```




- Many customers have short tenures, such as one or two months.
- Many customers are short-term customers.
- There are spikes in customer tenure at regular intervals, such as 12, 24, 36, and 48 months, which suggests that these are annual plans.

In [22]:
```
sns.displot(data=df,x='tenure',bins=70,col='Contract',row='Churn');
```

In [22]:
```
sns.displot(data=df,x='tenure',bins=70,col='Contract',row='Churn');
```

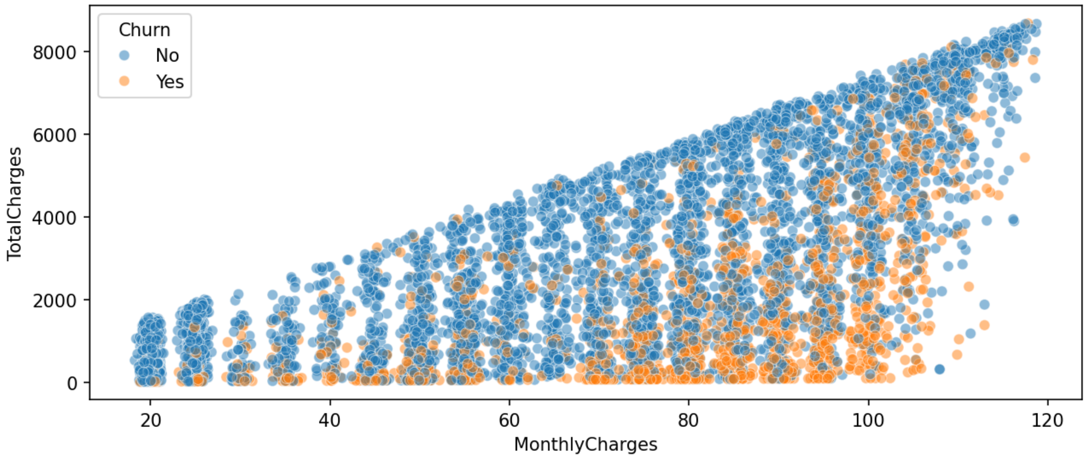



- Customers with one-year and two-year contracts are not churning.
- Customers with one-month contracts are churning at a high rate.

### Creating a scatter plot of Total Charges versus Monthly Charges, and color hue by Churn.

In [23]:
```
plt.figure(figsize=(10,4),dpi=150)
sns.scatterplot(data=df,x='MonthlyCharges', y='TotalCharges', hue='Churn',
```

Out[23]:
```
<Axes: xlabel='MonthlyCharges', ylabel='TotalCharges'>
```




- Many customers are more likely to cancel their subscriptions when their monthly charges are higher.

### Creating Cohorts based on Tenure

Let's begin by treating each unique tenure length, 1 month, 2 month, 3 month...N months as its own cohort.

## Calculating the Churn rate (percentage that had Yes Churn) per cohort.

In [24]:
```
yes_churn = df.groupby(['Churn','tenure']).count().transpose()['Yes']
```

In [25]:
```
yes_churn
```

Out[25]:

| tenure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 63 | 64 | 65 | 66 | 67 | 68 | 69 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **customerID** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **gender** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **SeniorCitizen** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **Partner** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **Dependents** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **PhoneService** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **MultipleLines** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **InternetService** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **OnlineSecurity** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **OnlineBackup** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **DeviceProtection** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **TechSupport** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **StreamingTV** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **StreamingMovies** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **Contract** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **PaperlessBilling** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **PaymentMethod** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **MonthlyCharges** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |
| **TotalCharges** | 380 | 123 | 94 | 83 | 64 | 40 | 51 | 42 | 46 | 45 | ... | 4 | 4 | 9 | 13 | 10 | 9 | 8 |

19 rows × 72 columns

In [26]:
```
no_churn = df.groupby(['Churn','tenure']).count().transpose()['No']
```

In [27]:
```
no_churn
```

In [27]:
```
no_churn
```

Out[27]:

| tenure | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 63 | 64 | 65 | 66 | 67 | 68 | 6 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **customerID** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **gender** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **SeniorCitizen** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **Partner** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **Dependents** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **PhoneService** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **MultipleLines** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **InternetService** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **OnlineSecurity** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **OnlineBackup** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **DeviceProtection** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **TechSupport** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **StreamingTV** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **StreamingMovies** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **Contract** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **PaperlessBilling** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **PaymentMethod** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **MonthlyCharges** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |
| **TotalCharges** | 233 | 115 | 106 | 93 | 69 | 70 | 80 | 81 | 73 | 71 | ... | 68 | 76 | 67 | 76 | 88 | 91 | 8 |

19 rows × 72 columns

In [28]:
```
churn_rate = 100 * yes_churn / (no_churn + yes_churn)
```

In [29]:
```
churn_rate.transpose()
```

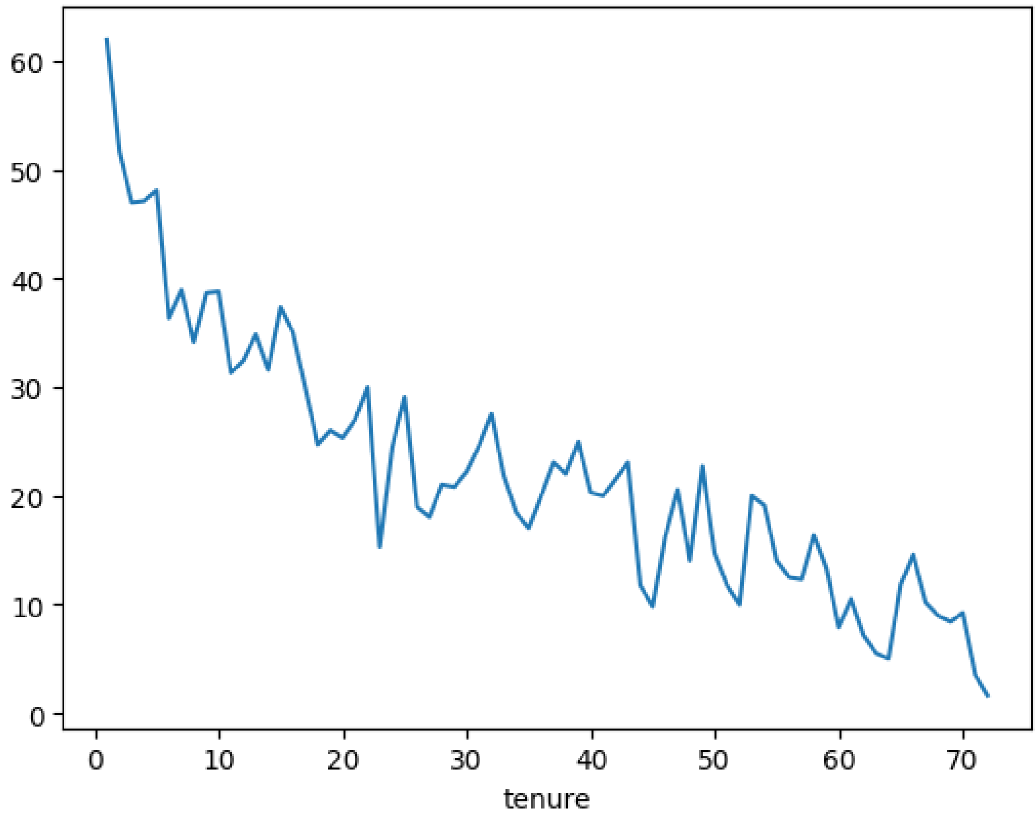
In [29]:
```
churn_rate.transpose()
```

Out[29]:

| tenure | customerID | gender | SeniorCitizen | Partner | Dependents | PhoneService | Multipl |
|---|---|---|---|---|---|---|---|
| 1 | 61.990212 | 61.990212 | 61.990212 | 61.990212 | 61.990212 | 61.990212 | 61.! |
| 2 | 51.680672 | 51.680672 | 51.680672 | 51.680672 | 51.680672 | 51.680672 | 51.( |
| 3 | 47.000000 | 47.000000 | 47.000000 | 47.000000 | 47.000000 | 47.000000 | 47.( |
| 4 | 47.159091 | 47.159091 | 47.159091 | 47.159091 | 47.159091 | 47.159091 | 47.' |
| 5 | 48.120301 | 48.120301 | 48.120301 | 48.120301 | 48.120301 | 48.120301 | 48.' |
| ... | ... | ... | ... | ... | ... | ... | |
| 68 | 9.000000 | 9.000000 | 9.000000 | 9.000000 | 9.000000 | 9.000000 | 9.( |
| 69 | 8.421053 | 8.421053 | 8.421053 | 8.421053 | 8.421053 | 8.421053 | 8.4 |
| 70 | 9.243697 | 9.243697 | 9.243697 | 9.243697 | 9.243697 | 9.243697 | 9.2 |
| 71 | 3.529412 | 3.529412 | 3.529412 | 3.529412 | 3.529412 | 3.529412 | 3.! |
| 72 | 1.657459 | 1.657459 | 1.657459 | 1.657459 | 1.657459 | 1.657459 | 1.6 |

72 rows × 19 columns

In [30]:
```
churn_rate.transpose()['customerID'].plot();
```




- Customer churn rate decreases as tenure increases.

### Broader Cohort Groups

**Based on the tenure column values, creating a new column called Tenure Cohort that creates 4 separate categories:**

- '0-12 Months'
- '12-24 Months'
- '24-48 Months'
- 'Over 48 Months'

In [31]:
```
def cohort(tenure):
    if tenure<13:
        return '0-12 Months'
    elif tenure < 25:
        return '12-24 Months'
    elif tenure<49:
        return '24-48 Months'
    else:
        return 'Over 48 Months'
```

In [32]:
```
df['Tenure Cohort'] = df['tenure'].apply(cohort)
```

In [33]:
```
df[['tenure','Tenure Cohort']]
```

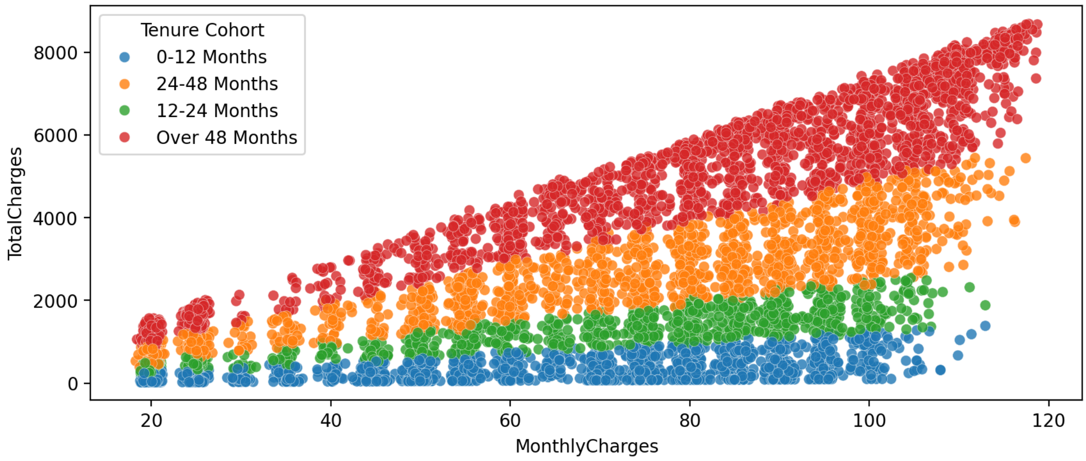
Out[33]:

| | tenure | Tenure Cohort |
|---|---|---|
| **0** | 1 | 0-12 Months |
| **1** | 34 | 24-48 Months |
| **2** | 2 | 0-12 Months |
| **3** | 45 | 24-48 Months |
| **4** | 2 | 0-12 Months |
| **...** | ... | ... |
| **7027** | 24 | 12-24 Months |
| **7028** | 72 | Over 48 Months |
| **7029** | 11 | 0-12 Months |
| **7030** | 4 | 0-12 Months |
| **7031** | 66 | Over 48 Months |

7032 rows × 2 columns

## Scatterplot of Total Charges versus Monthly Charts,

In [34]:
```
plt.figure(figsize=(10,4),dpi=200)
sns.scatterplot(data=df,x='MonthlyCharges',y='TotalCharges',hue='Tenure Co
```

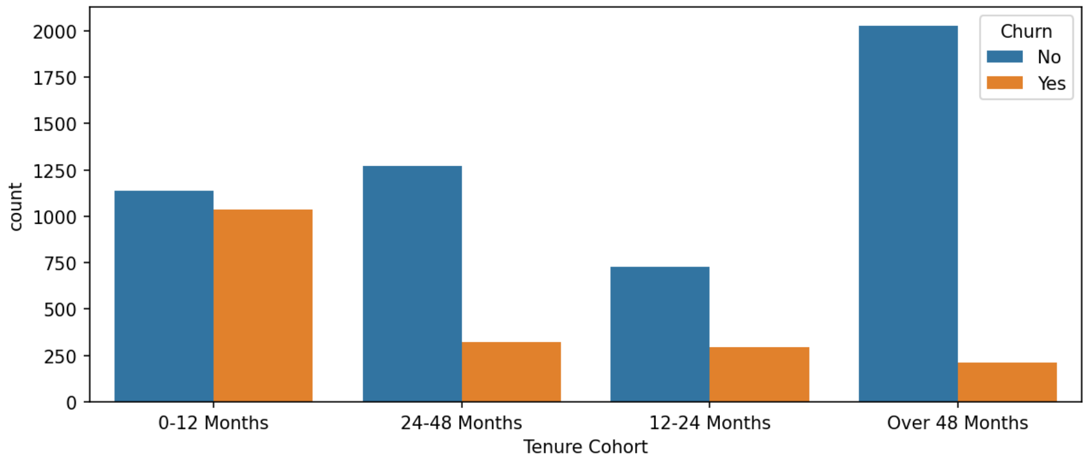



- Cohorts with longer tenures have the highest total charges.
- We can also see that there is a drop in total charges for some cohorts immediately after their monthly charges increase.

## Count plot showing the churn count per cohort.

In [35]:
```
plt.figure(figsize=(10,4),dpi=150)
sns.countplot(data=df,x='Tenure Cohort',hue='Churn')
```

Out[35]: <Axes: xlabel='Tenure Cohort', ylabel='count'>

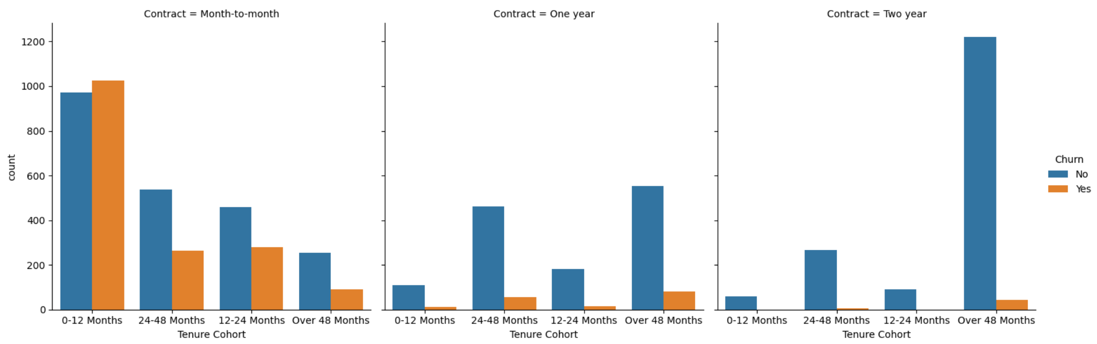



- Customers who stay for more than 48 months are less likely to churn.
- Customers with only 0-48 months of service are most likely to churn.

Create a grid of Count Plots showing counts per Tenure Cohort, separated out by contract type and colored by the Churn hue.

In [36]:
```
sns.catplot(data=df, x='Tenure Cohort', col='Contract', hue='Churn',kind='
```

In [36]:
```
sns.catplot(data=df, x='Tenure Cohort', col='Contract', hue='Churn',kind='
```




# Part 4: Predictive Modeling

**Let's explore 4 different tree based methods: A Single Decision Tree, Random Forest, AdaBoost, Gradient Boosting.

## Data Splitting

In [37]:
```
X = df.drop(['Churn','customerID'],axis=1)
X = pd.get_dummies(X,dtype=int,drop_first=True)

y = df['Churn']
```

In [38]:
```
from sklearn.model_selection import train_test_split
```

In [39]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, r
```

In [40]:
```
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test:",y_test.shape)
```

```
Shape of X_train:  (6328, 33)
Shape of X_test:  (704, 33)
Shape of y_train:  (6328,)
Shape of y_test: (704,)
```

## Model Selection : Single Decision Tree

In [41]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [42]:
```
dt = DecisionTreeClassifier(max_depth=6)
```

In [43]:
```
dt.fit(X_train, y_train)
```

In [43]:
```python
dt.fit(X_train, y_train)
```
Out[43]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=6)
```

## Model Evaluation

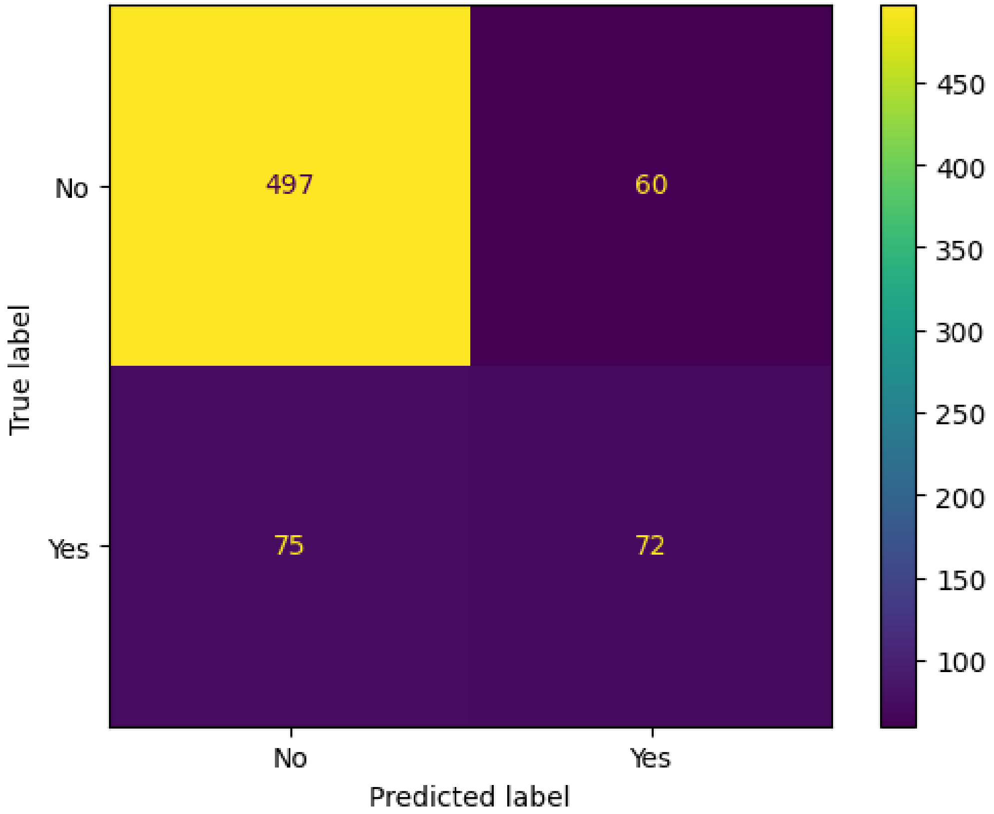
In [44]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMat
```

In [45]:
```python
preds = dt.predict(X_test)
```

In [46]:
```python
accuracy_score(y_test,preds)
```
Out[46]:
```
0.8082386363636364
```

In [47]:
```python
confusion_matrix(y_test,preds)
```
Out[47]:
```
array([[497,  60],
       [ 75,  72]], dtype=int64)
```

In [48]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test,preds), display_labels=dt.c
```




In [49]:
```python
print(classification_report(y_test,preds))
```

In [49]:
```
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

          No       0.87      0.89      0.88       557
         Yes       0.55      0.49      0.52       147

    accuracy                           0.81       704
   macro avg       0.71      0.69      0.70       704
weighted avg       0.80      0.81      0.80       704
```

- The overall accuracy of the tree model is 81%, which is good.
- The precision for the "No" class is 87%, which means that 87% of the customers that the model predicted would not churn actually did not churn.
- The recall for the "No" class is 89%, which means that the model identified 89% of all customers who did not churn.
- The precision for the "Yes" class is 55%, which means that 55% of the customers that the model predicted would churn actually did churn.
- The recall for the "Yes" class is 49%, which means that the model identified 49% of all customers who did churn.

In [50]:
```
imp_feats = pd.DataFrame(data=dt.feature_importances_,index=X.columns,colu
```

In [51]:
```
imp_feats
```

In [51]:
```
imp_feats
```

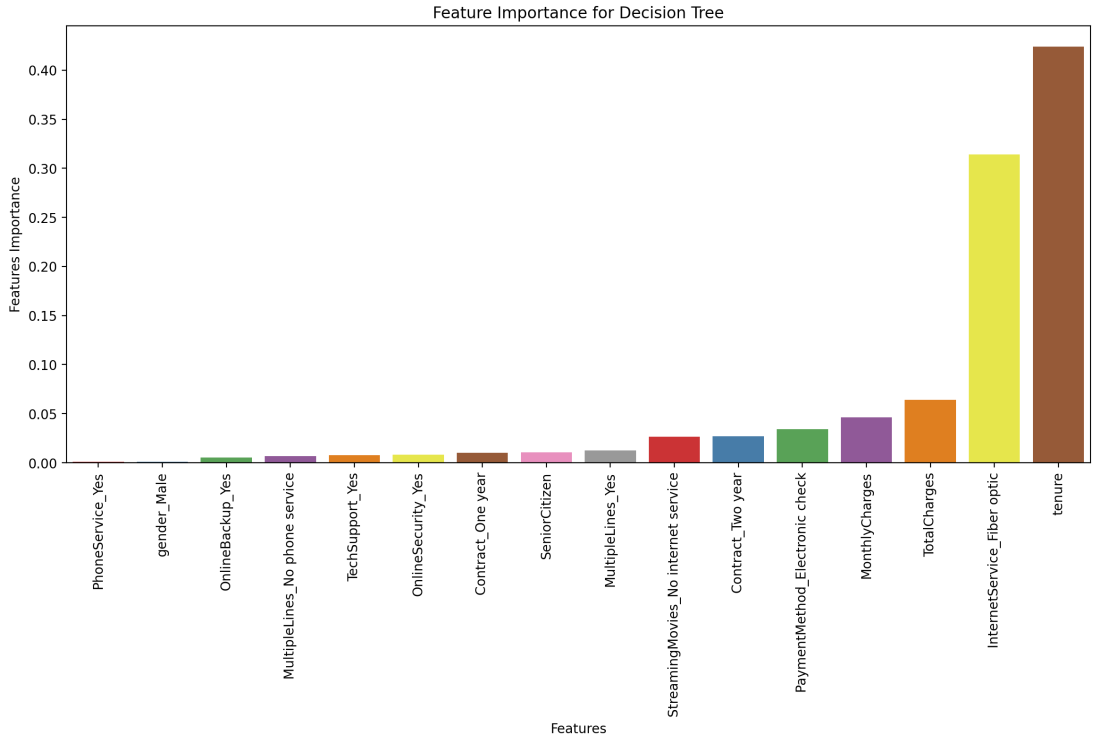
Out[51]:

| | Feature Importance |
|---|---|
| **DeviceProtection_No internet service** | 0.000000 |
| **Tenure Cohort_12-24 Months** | 0.000000 |
| **PaymentMethod_Mailed check** | 0.000000 |
| **PaymentMethod_Credit card (automatic)** | 0.000000 |
| **PaperlessBilling_Yes** | 0.000000 |
| **StreamingMovies_Yes** | 0.000000 |
| **StreamingTV_Yes** | 0.000000 |
| **StreamingTV_No internet service** | 0.000000 |
| **TechSupport_No internet service** | 0.000000 |
| **DeviceProtection_Yes** | 0.000000 |
| **Tenure Cohort_24-48 Months** | 0.000000 |
| **OnlineBackup_No internet service** | 0.000000 |
| **Tenure Cohort_Over 48 Months** | 0.000000 |
| **InternetService_No** | 0.000000 |
| **Dependents_Yes** | 0.000000 |
| **OnlineSecurity_No internet service** | 0.000000 |
| **Partner_Yes** | 0.000000 |
| **PhoneService_Yes** | 0.000890 |
| **gender_Male** | 0.001237 |
| **OnlineBackup_Yes** | 0.005341 |
| **MultipleLines_No phone service** | 0.006962 |
| **TechSupport_Yes** | 0.007868 |
| **OnlineSecurity_Yes** | 0.008376 |
| **Contract_One year** | 0.010021 |
| **SeniorCitizen** | 0.010825 |
| **MultipleLines_Yes** | 0.012432 |
| **StreamingMovies_No internet service** | 0.026290 |
| **Contract_Two year** | 0.027065 |
| **PaymentMethod_Electronic check** | 0.034436 |
| **MonthlyCharges** | 0.046115 |
| **TotalCharges** | 0.064168 |
| **InternetService_Fiber optic** | 0.314060 |
| **tenure** | 0.423914 |

In [52]:
```
imp_feats = imp_feats[imp_feats['Feature Importance'] > 0]
```

In [53]:
```
plt.figure(figsize=(14,6),dpi=200)
```

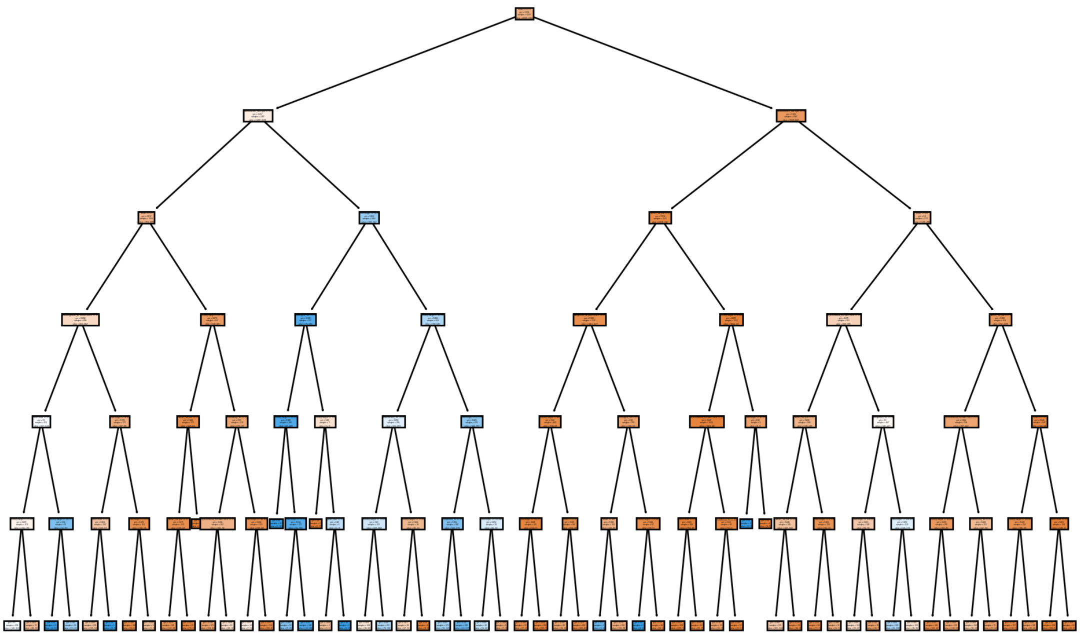
In [53]:
```
plt.figure(figsize=(14,6),dpi=200)
sns.barplot(data=imp_feats,x=imp_feats.index,y='Feature Importance',hue='F
plt.title('Feature Importance for Decision Tree')
plt.legend().remove()
plt.xlabel('Features')
plt.ylabel('Features Importance')
plt.xticks(rotation=90);
```




- Tenure is the most important feature for predicting customer churn. Customers who have been with the company for a longer period of time are less likely to churn.
- Customers with fiber optic internet service are less likely to churn than customers with other types of internet service. This suggests that fiber optic internet service is a valuable service that customers are willing to pay for.
- Customers with higher total and monthly charges are more likely to churn. This suggests that customers are price-sensitive and are looking for the best possible value.
- Customers who pay their bills with electronic checks are more likely to churn than customers who pay with other methods. This suggests that electronic check payments may be associated with financial problems, which can lead to churn.
- Customers who need tech support and customers who have contracts that are two years long are more likely to churn. This suggests that these customers may be having problems with the company's products or services.

In [54]:
```
from sklearn.tree import plot_tree
```

In [54]:
```
from sklearn.tree import plot_tree

plt.figure(figsize=(12,8),dpi=200)
plot_tree(dt,filled=True,feature_names=X.columns);
```



## Model Selection : Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [56]:
```
rf = RandomForestClassifier()
```

In [57]:
```
rf.fit(X_train,y_train)
```

Out[57]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

## Model Evaluation

In [58]:
```
preds = rf.predict(X_test)
```

In [59]:
```
accuracy_score(y_test,preds)
```

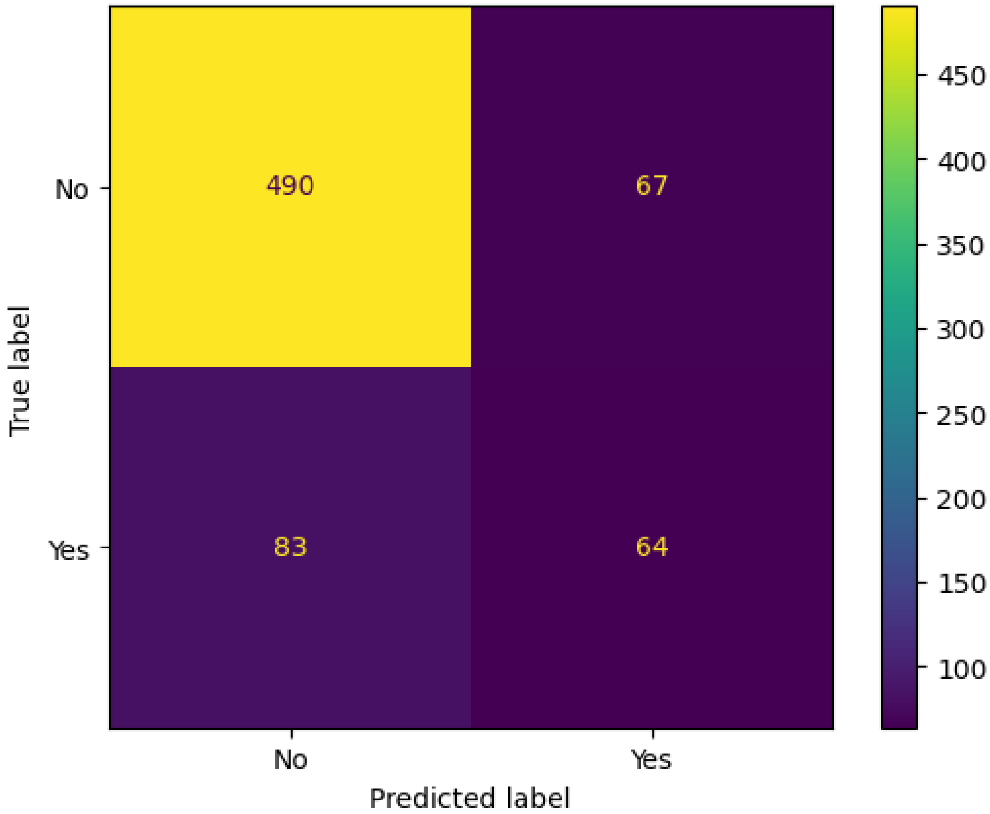
Out[59]:
```
0.7869318181818182
```

In [60]:
```
confusion_matrix(y_test,preds)
```

Out[60]:
```
array([[490,  67],
       [ 83,  64]], dtype=int64)
```

In [61]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,preds), display_labels=rf.c
```

In [61]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,preds), display_labels=rf.c
```




In [62]:
```
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

          No       0.86      0.88      0.87       557
         Yes       0.49      0.44      0.46       147

    accuracy                           0.79       704
   macro avg       0.67      0.66      0.66       704
weighted avg       0.78      0.79      0.78       704
```

- The model is performing well overall, with an accuracy of 81%.
- The model is performing better on the "No" class than on the "Yes" class.
- This could be due to a number of factors, such as imbalanced data, insufficient features, or overfitting.
- To improve the performance of the model on the "Yes" class, you can try balancing the dataset, adding more features, using a different machine learning algorithm, or understanding which features are most important for predicting customer churn.

**Tune the max depth parameter of the model to see if it improves accuracy.**

In [63]:
```
rf = RandomForestClassifier(max_depth=6)
```

In [64]:
```
rf.fit(X_train,y_train)
```

In [64]:
```
rf.fit(X_train,y_train)
```
Out[64]:
```
RandomForestClassifier
RandomForestClassifier(max_depth=6)
```

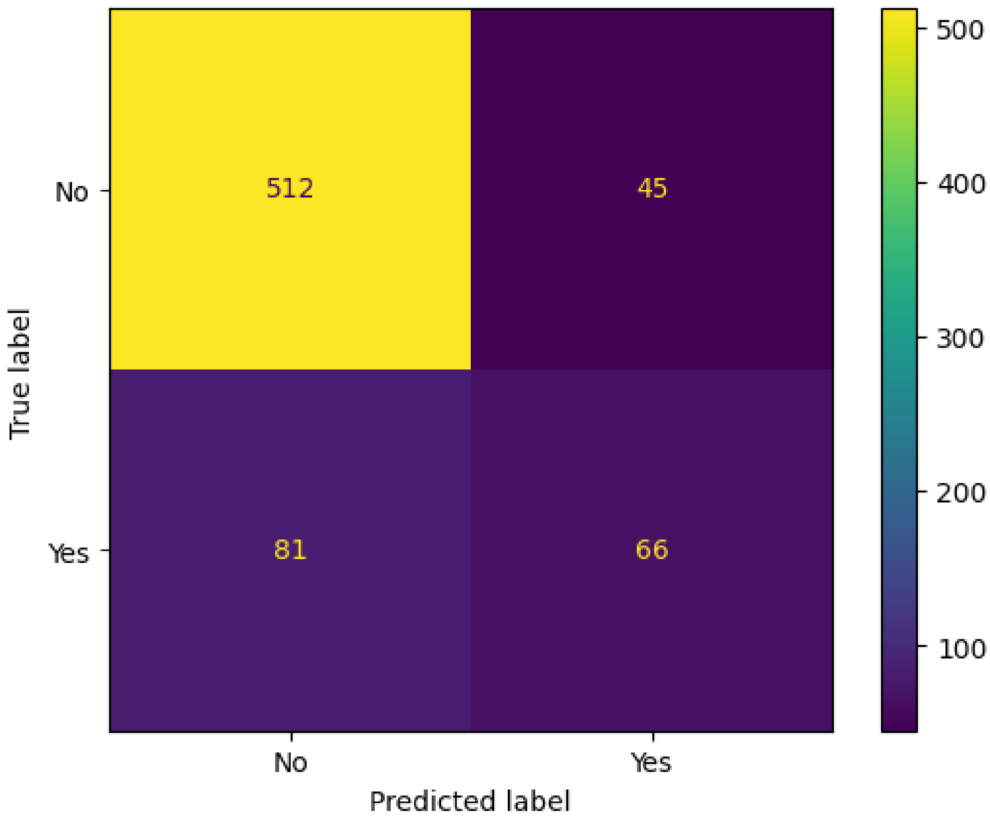
## Model Evaluation

In [65]:
```
preds = rf.predict(X_test)
```

In [66]:
```
accuracy_score(y_test,preds)
```
Out[66]:
```
0.8210227272727273
```

In [67]:
```
confusion_matrix(y_test,preds)
```
Out[67]:
```
array([[512,  45],
       [ 81,  66]], dtype=int64)
```

In [68]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,preds), display_labels=rf.c
```




In [69]:
```
print(classification_report(y_test,preds))
```

In [69]:
```python
print(classification_report(y_test,preds))
```

```
              precision    recall  f1-score   support

          No       0.86      0.92      0.89       557
         Yes       0.59      0.45      0.51       147

    accuracy                           0.82       704
   macro avg       0.73      0.68      0.70       704
weighted avg       0.81      0.82      0.81       704
```

- Overall, the RandomForestClassifier(max_depth=6) model is performing well, with an accuracy of 83%. It is also performing better on the "Yes" class than the previous model with a max_depth of 3. This suggests that increasing the max_depth parameter has helped to improve the model's ability to identify customers who are at risk of churning.
- However, it is important to note that the model is still making more false positives than false negatives. This means that the model is more likely to incorrectly predict that a customer will churn than to incorrectly predict that a customer will not churn.

## Model Selection : Gradient Boosting

In [70]:
```python
from sklearn.ensemble import GradientBoostingClassifier
```

In [71]:
```python
gb_model = GradientBoostingClassifier()
```

In [72]:
```python
gb_model.fit(X_train,y_train)
```

Out[72]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

## Model Evaluation

In [73]:
```python
gb_preds = gb_model.predict(X_test)
```

In [74]:
```python
accuracy_score(y_test,gb_preds)
```

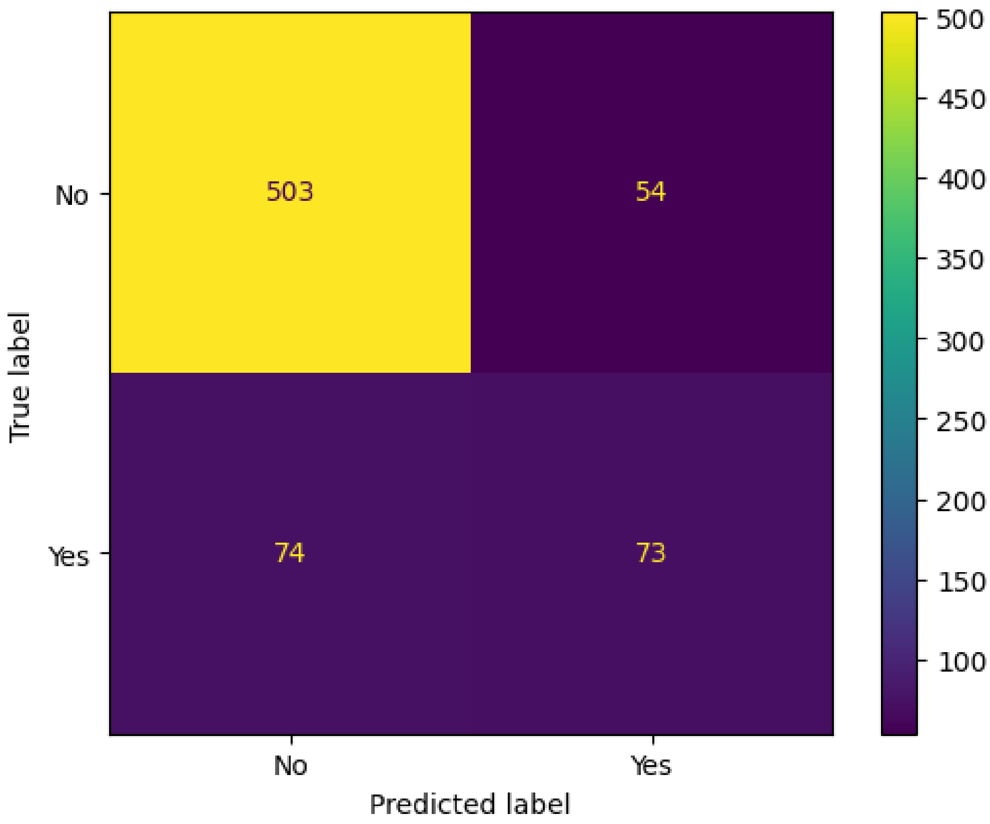
Out[74]:
```
0.8181818181818182
```

In [75]:
```python
confusion_matrix(y_test,gb_preds)
```

Out[75]:
```
array([[503,  54],
       [ 74,  73]], dtype=int64)
```

In [76]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test,gb_preds), display_labels=g
```

In [76]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,gb_preds), display_labels=g
```




In [77]:
```
print(classification_report(y_test,gb_preds))
```

```
              precision    recall  f1-score   support

          No       0.87      0.90      0.89       557
         Yes       0.57      0.50      0.53       147

    accuracy                           0.82       704
   macro avg       0.72      0.70      0.71       704
weighted avg       0.81      0.82      0.81       704
```

- Overall, the GradientBoostingClassifier model is performing well, with an accuracy of 82%. It is also performing better on the "Yes" class than the RandomForestClassifier model, with a precision of 0.57 and a recall of 0.50. This suggests that the GradientBoostingClassifier model is better able to identify customers who are at risk of churning.
- However, it is important to note that the model is still making more false positives than false negatives. This means that the model is more likely to incorrectly predict that a customer will churn than to incorrectly predict that a customer will not churn.

## Model Selection : Adaptive Boosting

In [78]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [79]:
```
ada_model = AdaBoostClassifier()
```

In [79]:
```
ada_model = AdaBoostClassifier()
```

In [80]:
```
ada_model.fit(X_train,y_train)
```

Out[80]:
```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

## Model Evaluation

In [81]:
```
ada_preds = ada_model.predict(X_test)
```

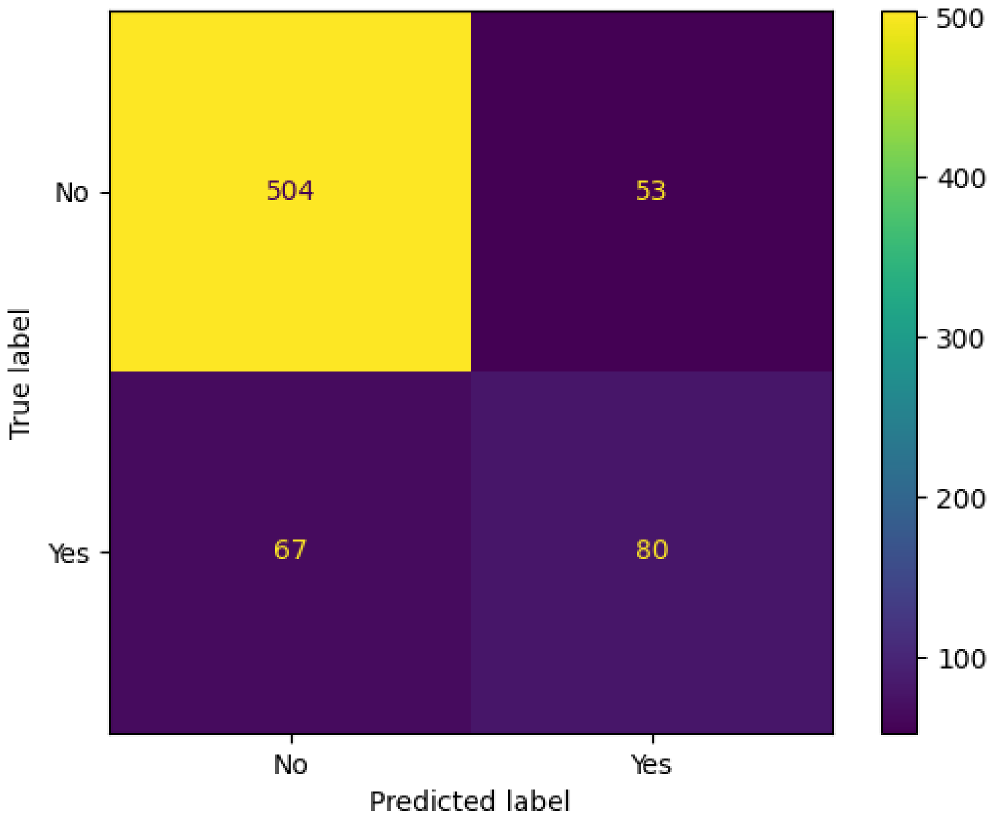
In [82]:
```
accuracy_score(y_test,ada_preds)
```

Out[82]:
```
0.8295454545454546
```

In [83]:
```
confusion_matrix(y_test,ada_preds)
```

Out[83]:
```
array([[504,  53],
       [ 67,  80]], dtype=int64)
```

In [84]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,ada_preds), display_labels=
```




In [85]:
```
print(classification_report(y_test,ada_preds))
```

In [85]:
```
print(classification_report(y_test,ada_preds))
```

```
              precision    recall  f1-score   support

          No       0.88      0.90      0.89       557
         Yes       0.60      0.54      0.57       147

    accuracy                           0.83       704
   macro avg       0.74      0.72      0.73       704
weighted avg       0.82      0.83      0.83       704
```

- Overall, the AdaBoostClassifier model is performing well, with an accuracy of 83%. It is also performing better on the "Yes" class than the previous models, with a precision of 0.60 and a recall of 0.54. This suggests that the AdaBoostClassifier model is better able to identify customers who are at risk of churning.
- However, it is important to note that the model is still making more false positives than false negatives. This means that the model is more likely to incorrectly predict that a customer will churn than to incorrectly predict that a customer will not churn.

**Tune the parameter of the adaboost to see if it improves accuracy.**

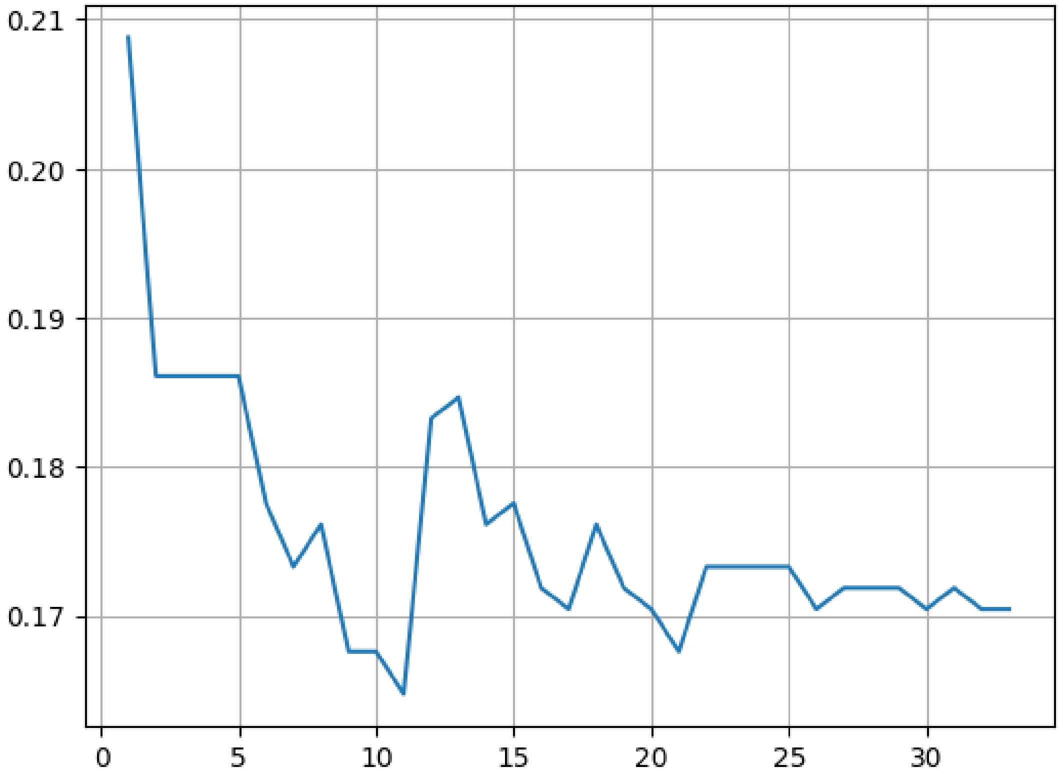
In [86]:
```
len(X.columns)
```

Out[86]:
```
33
```

In [87]:
```
error_rates = []

for n in range(1,34):
    model = AdaBoostClassifier(n_estimators=n)
    model.fit(X_train,y_train)
    preds = model.predict(X_test)

    err = 1 - accuracy_score(y_test, preds)

    error_rates.append(err)
```

In [88]:
```
plt.plot(range(1,34),error_rates)
```

In [88]:
```python
plt.plot(range(1,34),error_rates)
plt.grid()
```




In [89]:
```python
ada_model = AdaBoostClassifier(n_estimators=11)
```

In [90]:
```python
ada_model.fit(X_train,y_train)
```

Out[90]:
```
AdaBoostClassifier
AdaBoostClassifier(n_estimators=11)
```

## Model Evaluation

In [91]:
```python
ada_preds = ada_model.predict(X_test)
```

In [92]:
```python
accuracy_score(y_test,ada_preds)
```

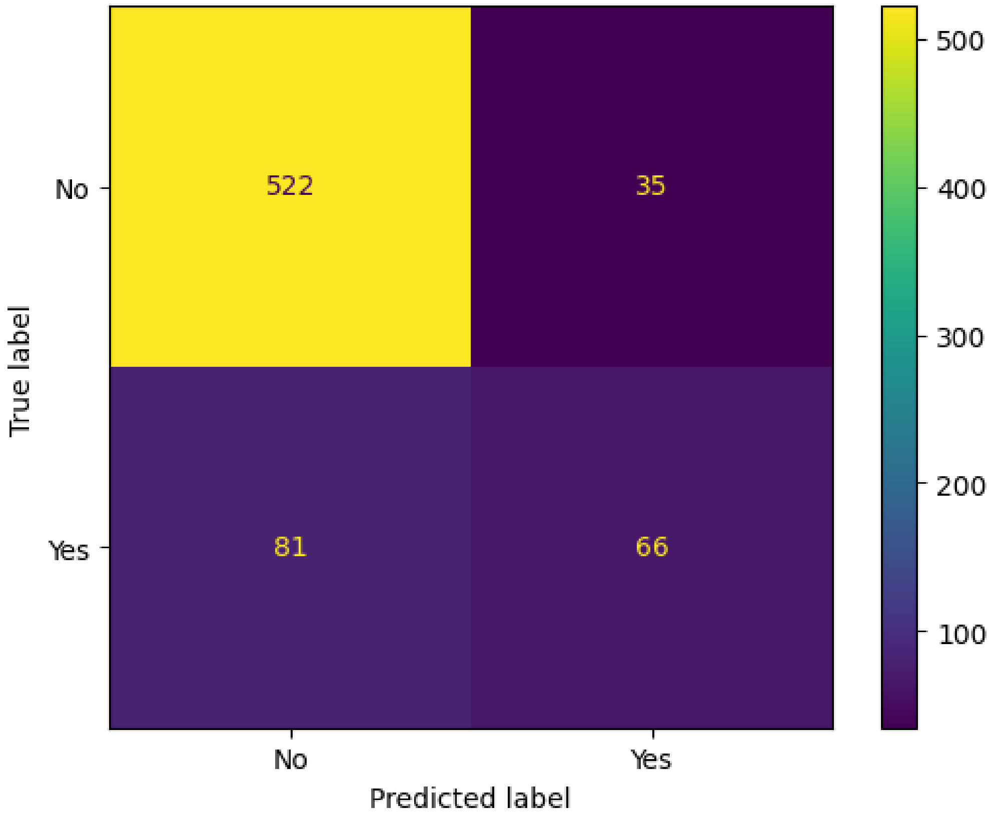
Out[92]:
```
0.8352272727272727
```

In [93]:
```python
confusion_matrix(y_test,ada_preds)
```

Out[93]:
```
array([[522,  35],
       [ 81,  66]], dtype=int64)
```

In [94]:
```python
ConfusionMatrixDisplay(confusion_matrix(y_test,ada_preds), display_labels=
```

In [94]:
```
ConfusionMatrixDisplay(confusion_matrix(y_test,ada_preds), display_labels=
```




In [95]:
```
print(classification_report(y_test,ada_preds))
```

```
              precision    recall  f1-score   support

          No       0.87      0.94      0.90       557
         Yes       0.65      0.45      0.53       147

    accuracy                           0.84       704
   macro avg       0.76      0.69      0.72       704
weighted avg       0.82      0.84      0.82       704
```

- Overall, the AdaBoostClassifier(n_estimators=11) model is performing well, with an accuracy of 84%. It is also performing better on the "Yes" class than the previous models, with a precision of 0.65 and a recall of 0.45. This suggests that increasing the number of base estimators in the AdaBoost model has helped to improve the model's ability to identify customers who are at risk of churning.
- However, it is important to note that the model is still making more false positives than false negatives. This means that the model is more likely to incorrectly predict that a customer will churn than to incorrectly predict that a customer will not churn.

# Model Deployment

In [96]:
```
# from joblib import dump,load
```

In [97]:
```
# dump(final_model, 'ada_model.joblib')
```

In [97]:

```
# dump(final_model, 'ada_model.joblib')
```