Supervised Learning Capstone Project - Tree Methods Focus

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

In [ ]:
```
# Import the belwo to START THE PROJECT!
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns
```

In [ ]:
```
df = pd.read_csv('C:\\Users\\Ravi Nadageri\\Desktop\\Py_Practice _file\\ML_models\\Tel
```

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

Out[ ]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

Part 1: Quick Data Check

In [ ]:
```
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
```

TASK: Get a quick statistical summary of the numeric columns with .describe() , you should notice that many columns are categorical, meaning you will eventually need to convert them to dummy variables.

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

Out[ ]:

	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

Part 2: Exploratory Data Analysis

General Feature Exploration

TASK: Confirm that there are no NaN cells by displaying NaN values per feature column.

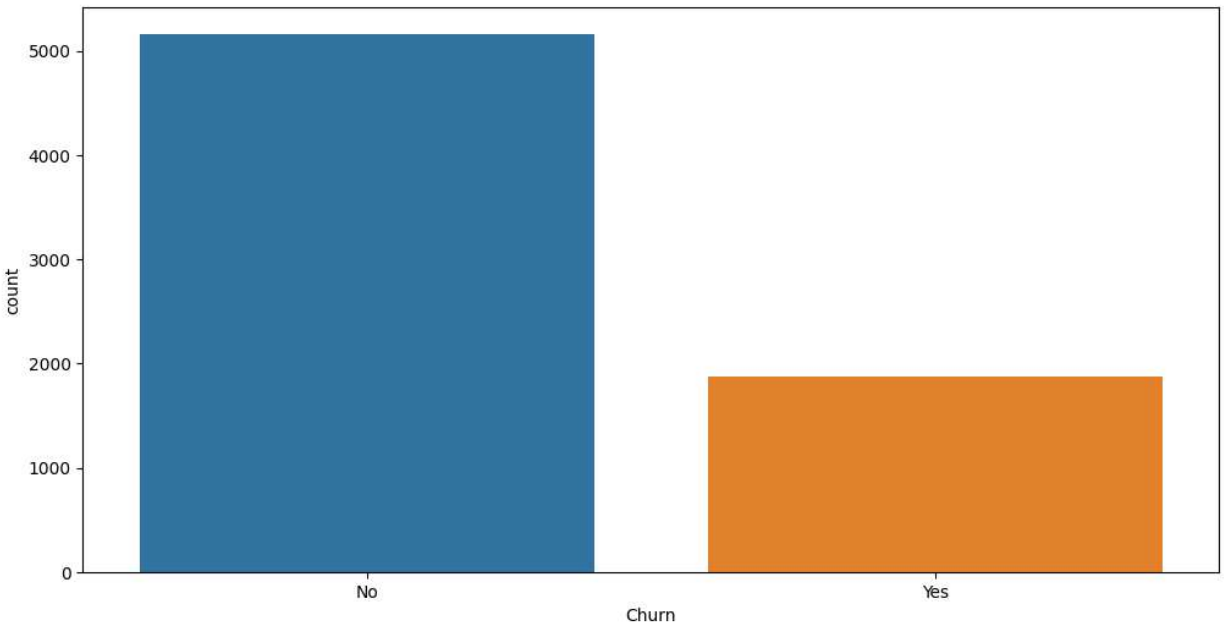
In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
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
```

TASK:Display the balance of the class labels (Churn) with a Count Plot.

In [ ]:
```
plt.rcParams['figure.figsize'] = (12,6)
sns.countplot(data=df,x='Churn')
```

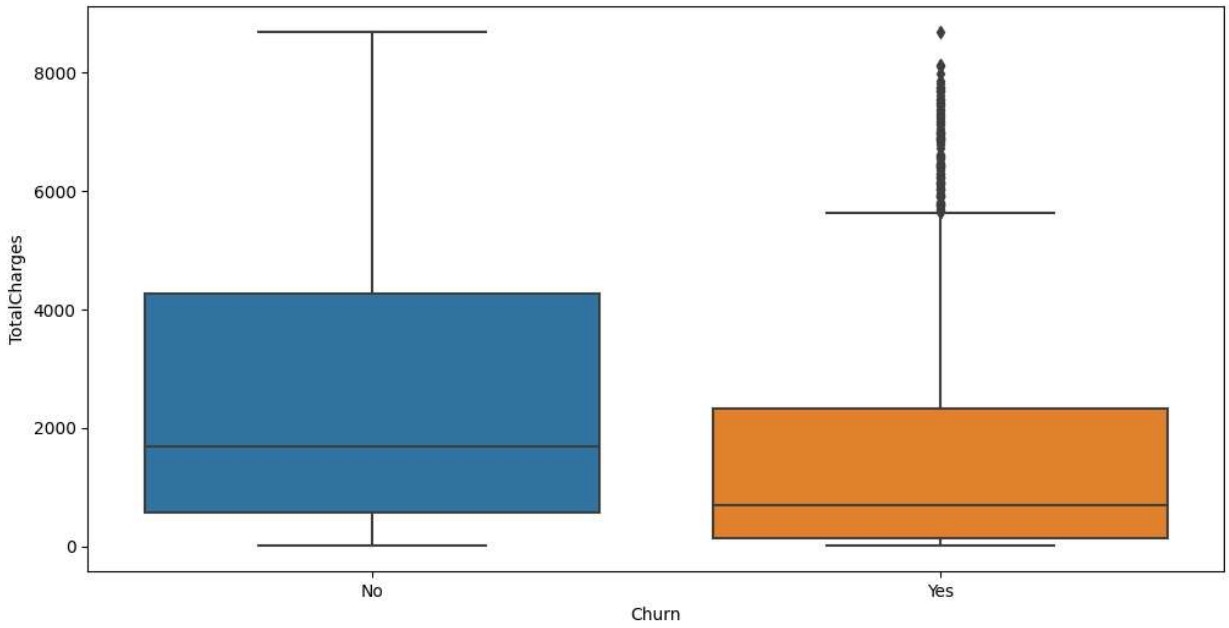
Out[ ]:
```
<AxesSubplot:xlabel='Churn', ylabel='count'>
```



TASK: Explore the distrbution of TotalCharges between Churn categories with a Box Plot or Violin Plot.

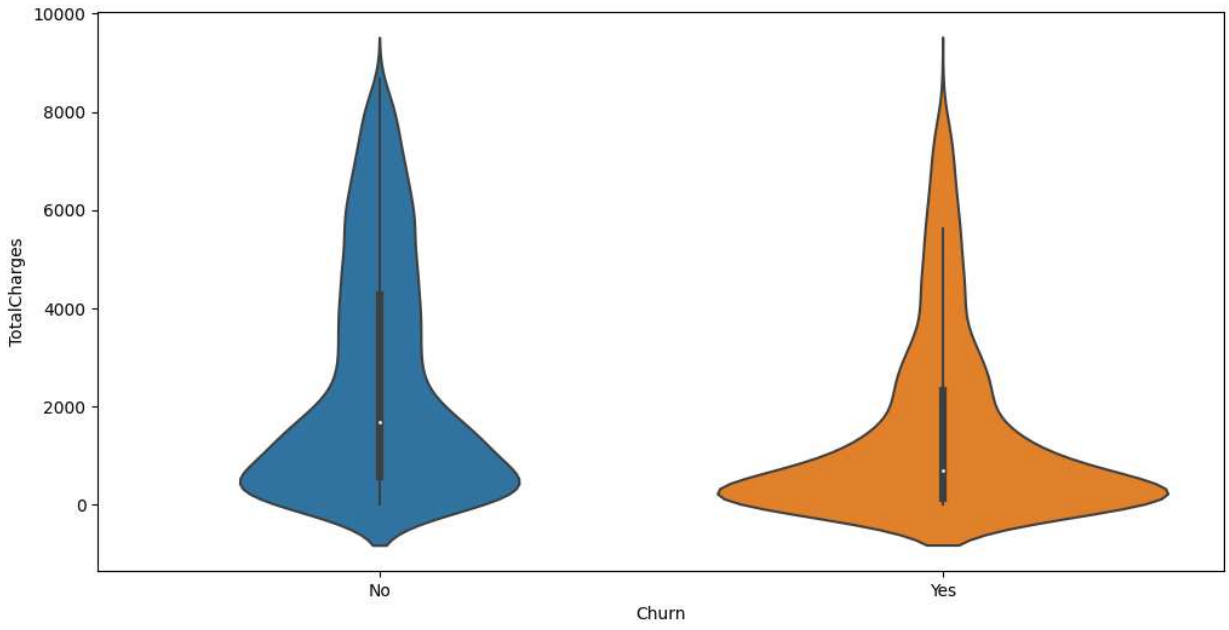
In [ ]:
```
sns.boxplot(data=df,x='Churn',y='TotalCharges')
```

Out[ ]:
```
<AxesSubplot:xlabel='Churn', ylabel='TotalCharges'>
```



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

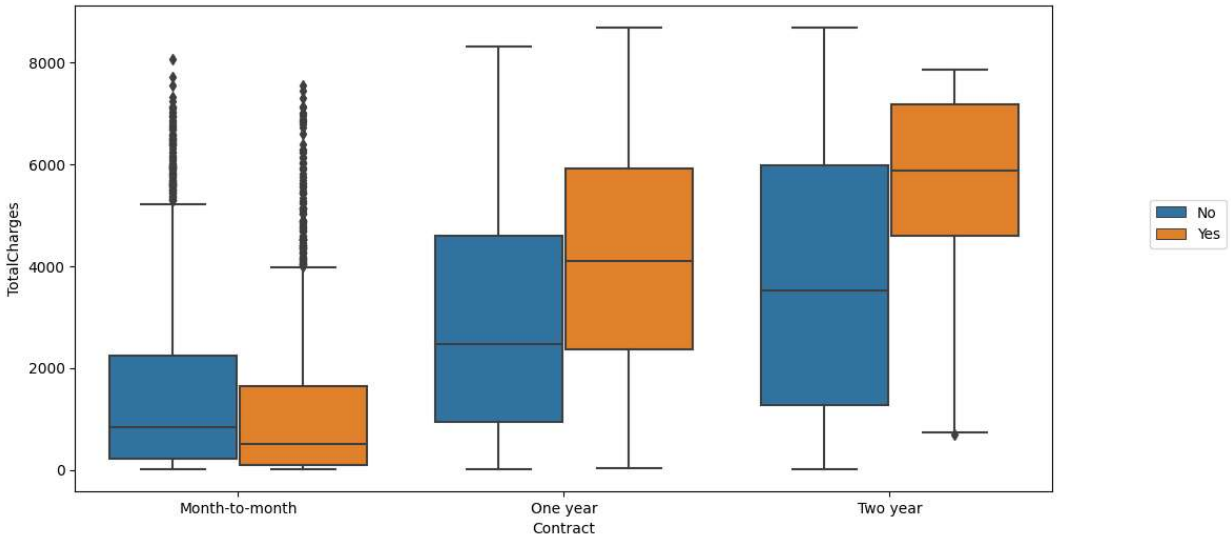
Out[ ]:
```
<AxesSubplot:xlabel='Churn', ylabel='TotalCharges'>
```



TASK: Create a boxplot showing the distribution of TotalCharges per Contract type, also add in a hue coloring based on the Churn class.

In [ ]:
```
sns.boxplot(x='Contract',y='TotalCharges',data=df,hue='Churn')
plt.legend(loc=(1.1,0.5))
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x205fbf633d0>
```



TASK: Create a bar plot showing the correlation of the following features to the class label. Keep in mind, for the categorical features, you will need to convert them into dummy variables first, as you can only calculate correlation for numeric features.

```
['gender', 'SeniorCitizen', 'Partner', 'Dependents','PhoneService',
'MultipleLines',
 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
'InternetService',
   'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
'PaymentMethod']
```

Note, we specifically listed only the features above, you should not check the correlation for every feature, as some features have too many unique instances for such an analysis, such as customerID

In [ ]:
```
df_dummy = pd.get_dummies(df[['gender', 'SeniorCitizen', 'Partner', 'Dependents','Phor
     'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'InternetSer
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMetho
```

In [ ]:
```
df_dummy.head()
```

Out[ ]:

	SeniorCitizen	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Dependents_
0	0	1	0	0	1	1	
1	0	0	1	1	0	1	
2	0	0	1	1	0	1	
3	0	0	1	1	0	1	
4	0	1	0	1	0	1	

5 rows × 44 columns

In [ ]:
```
df_dummy.corr()
```

Out[ ]:

	SeniorCitizen	gender_Female	gender_Male	Partner_No	Partner_Yes	Depe
SeniorCitizen	1.000000	0.001819	-0.001819	-0.016957	0.016957	
gender_Female	0.001819	1.000000	-1.000000	-0.001379	0.001379	
gender_Male	-0.001819	-1.000000	1.000000	0.001379	-0.001379	
Partner_No	-0.016957	-0.001379	0.001379	1.000000	-1.000000	
Partner_Yes	0.016957	0.001379	-0.001379	-1.000000	1.000000	
Dependents_No	0.210550	0.010349	-0.010349	0.452269	-0.452269	
Dependents_Yes	-0.210550	-0.010349	0.010349	-0.452269	0.452269	
PhoneService_No	-0.008392	-0.007515	0.007515	0.018397	-0.018397	
PhoneService_Yes	0.008392	0.007515	-0.007515	-0.018397	0.018397	
MultipleLines_No	-0.136377	-0.004335	0.004335	0.130028	-0.130028	
MultipleLines_No phone service	-0.008392	-0.007515	0.007515	0.018397	-0.018397	
MultipleLines_Yes	0.142996	0.008883	-0.008883	-0.142561	0.142561	
OnlineSecurity_No	0.185145	-0.010859	0.010859	0.129394	-0.129394	
OnlineSecurity_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
OnlineSecurity_Yes	-0.038576	0.016328	-0.016328	-0.143346	0.143346	
OnlineBackup_No	0.087539	-0.008605	0.008605	0.135626	-0.135626	
OnlineBackup_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
OnlineBackup_Yes	0.066663	0.013093	-0.013093	-0.141849	0.141849	
DeviceProtection_No	0.094403	0.003163	-0.003163	0.146702	-0.146702	
DeviceProtection_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
DeviceProtection_Yes	0.059514	0.000807	-0.000807	-0.153556	0.153556	
TechSupport_No	0.205254	-0.003815	0.003815	0.108875	-0.108875	
TechSupport_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
TechSupport_Yes	-0.060577	0.008507	-0.008507	-0.120206	0.120206	
InternetService_DSL	-0.108276	-0.007584	0.007584	0.001043	-0.001043	
InternetService_Fiber optic	0.254923	0.011189	-0.011189	-0.001235	0.001235	
InternetService_No	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
StreamingTV_No	0.048664	-0.003088	0.003088	0.123394	-0.123394	
StreamingTV_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	

	SeniorCitizen	gender_Female	gender_Male	Partner_No	Partner_Yes	Depe
StreamingTV_Yes	0.105445	0.007124	-0.007124	-0.124483	0.124483	
StreamingMovies_No	0.034196	-0.006078	0.006078	0.117488	-0.117488	
StreamingMovies_No internet service	-0.182519	-0.004745	0.004745	0.000286	-0.000286	
StreamingMovies_Yes	0.119842	0.010105	-0.010105	-0.118108	0.118108	
Contract_Month-to-month	0.137752	0.003251	-0.003251	0.280202	-0.280202	
Contract_One year	-0.046491	-0.007755	0.007755	-0.083067	0.083067	
Contract_Two year	-0.116205	0.003603	-0.003603	-0.247334	0.247334	
PaperlessBilling_No	-0.156258	-0.011902	0.011902	-0.013957	0.013957	
PaperlessBilling_Yes	0.156258	0.011902	-0.011902	0.013957	-0.013957	
PaymentMethod_Bank transfer (automatic)	-0.016235	0.015973	-0.015973	-0.111406	0.111406	
PaymentMethod_Credit card (automatic)	-0.024359	-0.001632	0.001632	-0.082327	0.082327	
PaymentMethod_Electronic check	0.171322	-0.000844	0.000844	0.083207	-0.083207	
PaymentMethod_Mailed check	-0.152987	-0.013199	0.013199	0.096948	-0.096948	
Churn_No	-0.150541	-0.008545	0.008545	-0.149982	0.149982	
Churn_Yes	0.150541	0.008545	-0.008545	0.149982	-0.149982	

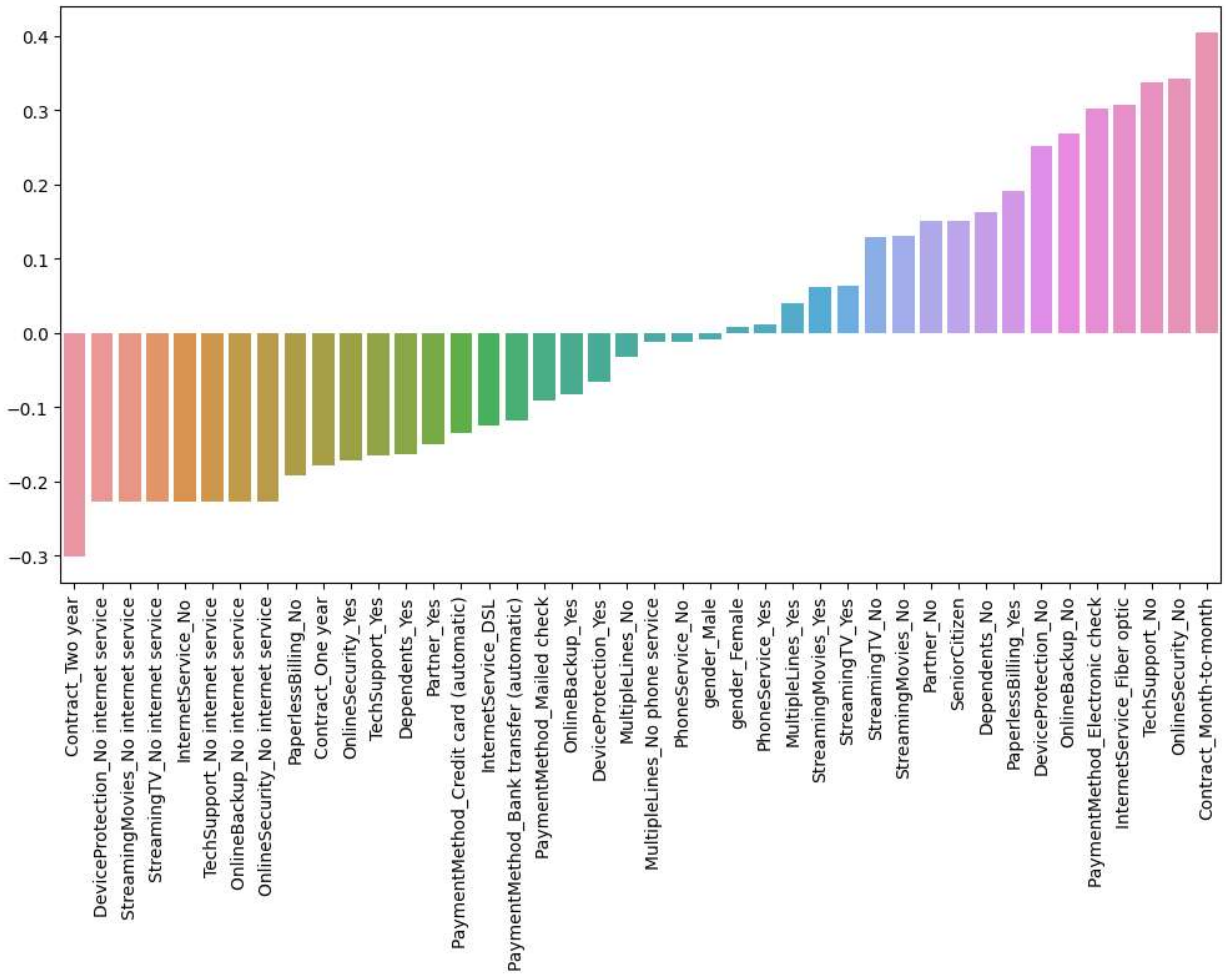
In [ ]:
```
df_corr = df_dummy.corr()['Churn_Yes'].sort_values().iloc[1:-1] #it will ommit the fir
```

In [ ]:
```
df_corr
```

Out[ ]:
```
Contract_Two year                          -0.301552
DeviceProtection_No internet service       -0.227578
StreamingMovies_No internet service        -0.227578
StreamingTV_No internet service            -0.227578
InternetService_No                         -0.227578
TechSupport_No internet service            -0.227578
OnlineBackup_No internet service           -0.227578
OnlineSecurity_No internet service         -0.227578
PaperlessBilling_No                        -0.191454
Contract_One year                          -0.178225
OnlineSecurity_Yes                         -0.171270
TechSupport_Yes                            -0.164716
Dependents_Yes                             -0.163128
Partner_Yes                                -0.149982
PaymentMethod_Credit card (automatic)      -0.134687
InternetService_DSL                        -0.124141
PaymentMethod_Bank transfer (automatic)    -0.118136
PaymentMethod_Mailed check                 -0.090773
OnlineBackup_Yes                           -0.082307
DeviceProtection_Yes                       -0.066193
MultipleLines_No                           -0.032654
MultipleLines_No phone service             -0.011691
PhoneService_No                            -0.011691
gender_Male                                -0.008545
gender_Female                               0.008545
PhoneService_Yes                            0.011691
MultipleLines_Yes                           0.040033
StreamingMovies_Yes                         0.060860
StreamingTV_Yes                             0.063254
StreamingTV_No                              0.128435
StreamingMovies_No                          0.130920
Partner_No                                  0.149982
SeniorCitizen                               0.150541
Dependents_No                               0.163128
PaperlessBilling_Yes                        0.191454
DeviceProtection_No                         0.252056
OnlineBackup_No                             0.267595
PaymentMethod_Electronic check              0.301455
InternetService_Fiber optic                 0.307463
TechSupport_No                              0.336877
OnlineSecurity_No                           0.342235
Contract_Month-to-month                     0.404565
Name: Churn_Yes, dtype: float64
```

In [ ]:
```
sns.barplot(x=df_corr.index,y=df_corr.values)
plt.xticks(rotation = 90)
```

Out[ ]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41]),
 [Text(0, 0, 'Contract_Two year'),
  Text(1, 0, 'DeviceProtection_No internet service'),
  Text(2, 0, 'StreamingMovies_No internet service'),
  Text(3, 0, 'StreamingTV_No internet service'),
  Text(4, 0, 'InternetService_No'),
  Text(5, 0, 'TechSupport_No internet service'),
  Text(6, 0, 'OnlineBackup_No internet service'),
  Text(7, 0, 'OnlineSecurity_No internet service'),
  Text(8, 0, 'PaperlessBilling_No'),
  Text(9, 0, 'Contract_One year'),
  Text(10, 0, 'OnlineSecurity_Yes'),
  Text(11, 0, 'TechSupport_Yes'),
  Text(12, 0, 'Dependents_Yes'),
  Text(13, 0, 'Partner_Yes'),
  Text(14, 0, 'PaymentMethod_Credit card (automatic)'),
  Text(15, 0, 'InternetService_DSL'),
  Text(16, 0, 'PaymentMethod_Bank transfer (automatic)'),
  Text(17, 0, 'PaymentMethod_Mailed check'),
  Text(18, 0, 'OnlineBackup_Yes'),
  Text(19, 0, 'DeviceProtection_Yes'),
  Text(20, 0, 'MultipleLines_No'),
  Text(21, 0, 'MultipleLines_No phone service'),
  Text(22, 0, 'PhoneService_No'),
  Text(23, 0, 'gender_Male'),
  Text(24, 0, 'gender_Female'),
  Text(25, 0, 'PhoneService_Yes'),
  Text(26, 0, 'MultipleLines_Yes'),
  Text(27, 0, 'StreamingMovies_Yes'),
  Text(28, 0, 'StreamingTV_Yes'),
  Text(29, 0, 'StreamingTV_No'),
  Text(30, 0, 'StreamingMovies_No'),
  Text(31, 0, 'Partner_No'),
  Text(32, 0, 'SeniorCitizen'),
  Text(33, 0, 'Dependents_No'),
  Text(34, 0, 'PaperlessBilling_Yes'),
  Text(35, 0, 'DeviceProtection_No'),
  Text(36, 0, 'OnlineBackup_No'),
  Text(37, 0, 'PaymentMethod_Electronic check'),
  Text(38, 0, 'InternetService_Fiber optic'),
  Text(39, 0, 'TechSupport_No'),
  Text(40, 0, 'OnlineSecurity_No'),
  Text(41, 0, 'Contract_Month-to-month')])
```



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.

TASK: What are the 3 contract types available?

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

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

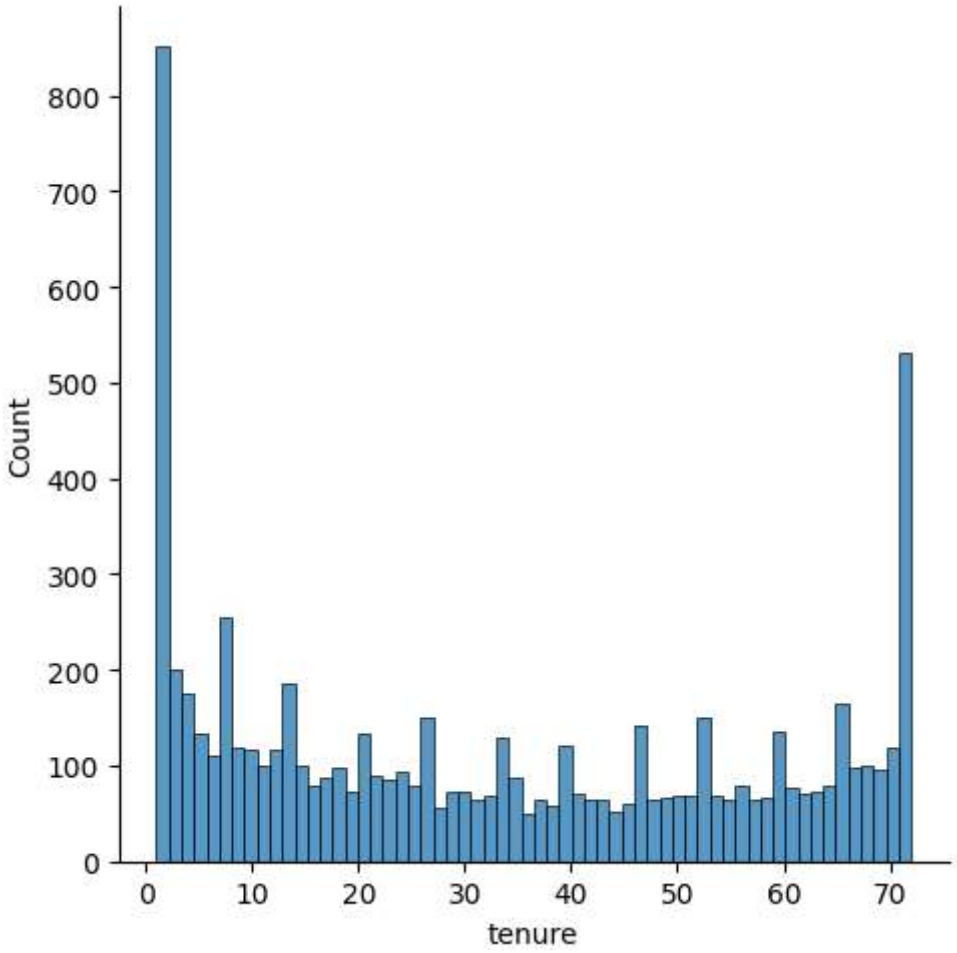
TASK: Create a histogram displaying the distribution of 'tenure' column, which is the amount of months a customer was or has been on a customer.

In [ ]:
```
plt.rcParams['figure.figsize'] = [10,6]
fig = plt.figure()
plt.Figure(figsize=(2,6))
```

```
sns.displot(data=df,x='tenure',bins=60)
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x205fc0b6cb0>

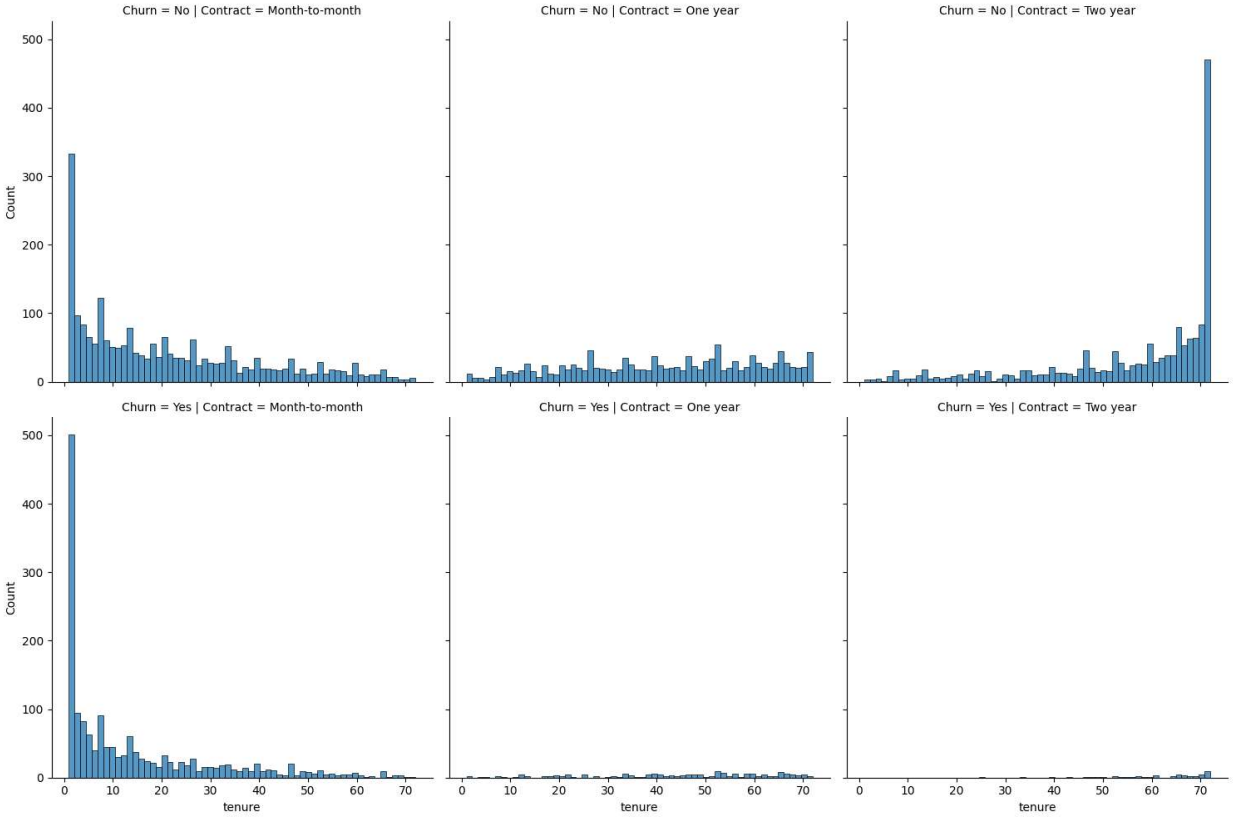
<Figure size 1000x600 with 0 Axes>



TASK: Now use the seaborn documentation as a guide to create histograms separated by two additional features, Churn and Contract.

In [ ]:
```
sns.displot(data=df,x='tenure',bins=60,col='Contract',row="Churn")
```

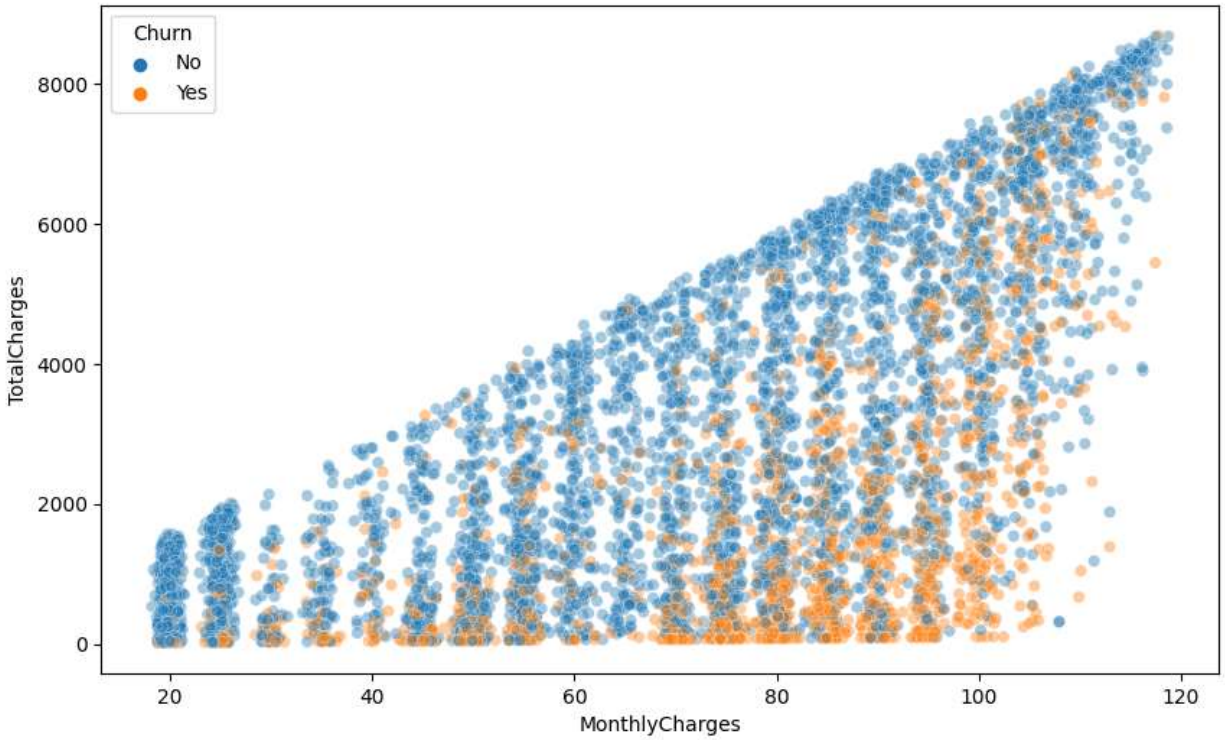
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x205fc118370>



TASK: Display a scatter plot of Total Charges versus Monthly Charges, and color hue by Churn.

In [ ]:
```
sns.scatterplot(data=df,
                x='MonthlyCharges',
                y='TotalCharges',
                hue='Churn',alpha=0.4)
```

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



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.

TASK: Treating each unique tenure group as a cohort, calculate the Churn rate (percentage that had Yes Churn) per cohort. For example, the cohort that has had a tenure of 1 month should have a Churn rate of 61.99%. You should have cohorts 1-72 months with a general trend of the longer the tenure of the cohort, the less of a churn rate. This makes sense as you are less likely to stop service the longer you've had it.

In [ ]:
```
df.groupby(['Churn','tenure']).count()
```

Out[ ]:

Churn	tenure	customerID	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines
No	**1**	233	233	233	233	233	233	233
	2	115	115	115	115	115	115	115
	3	106	106	106	106	106	106	106
	4	93	93	93	93	93	93	93
	5	69	69	69	69	69	69	69
...	**...**	...	...	...	...	...	...	...
Yes	**68**	9	9	9	9	9	9	9
	69	8	8	8	8	8	8	8
	70	11	11	11	11	11	11	11
	71	6	6	6	6	6	6	6
	72	6	6	6	6	6	6	6

144 rows × 19 columns

In [ ]:
```
yes_churn = df.groupby(['Churn','tenure']).count().T['Yes']
no_churn = df.groupby(['Churn','tenure']).count().T['No']
yes_churn
```

Out[ ]:

tenure	1	2	3	4	5	6	7	8	9	10	...	63	64	65	66	67	68	69	70	71
customerID	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
gender	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
SeniorCitizen	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
Partner	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
Dependents	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
PhoneService	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
MultipleLines	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
InternetService	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
OnlineSecurity	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
OnlineBackup	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
DeviceProtection	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
TechSupport	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
StreamingTV	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
StreamingMovies	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
Contract	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
PaperlessBilling	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
PaymentMethod	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
MonthlyCharges	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6
TotalCharges	380	123	94	83	64	40	51	42	46	45	...	4	4	9	13	10	9	8	11	6

19 rows × 72 columns

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

In [ ]:
```
churn_rate.T
```

Out[ ]:

tenure	customerID	gender	SeniorCitizen	Partner	Dependents	PhoneService	MultipleLines	In
1	61.990212	61.990212	61.990212	61.990212	61.990212	61.990212	61.990212	
2	51.680672	51.680672	51.680672	51.680672	51.680672	51.680672	51.680672	
3	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	47.000000	
4	47.159091	47.159091	47.159091	47.159091	47.159091	47.159091	47.159091	
5	48.120301	48.120301	48.120301	48.120301	48.120301	48.120301	48.120301	
...	...	...	...	...	...	...	...	
68	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	9.000000	
69	8.421053	8.421053	8.421053	8.421053	8.421053	8.421053	8.421053	
70	9.243697	9.243697	9.243697	9.243697	9.243697	9.243697	9.243697	
71	3.529412	3.529412	3.529412	3.529412	3.529412	3.529412	3.529412	
72	1.657459	1.657459	1.657459	1.657459	1.657459	1.657459	1.657459	

72 rows × 19 columns

In [ ]:
```
churn_rate.T['customerID']
```

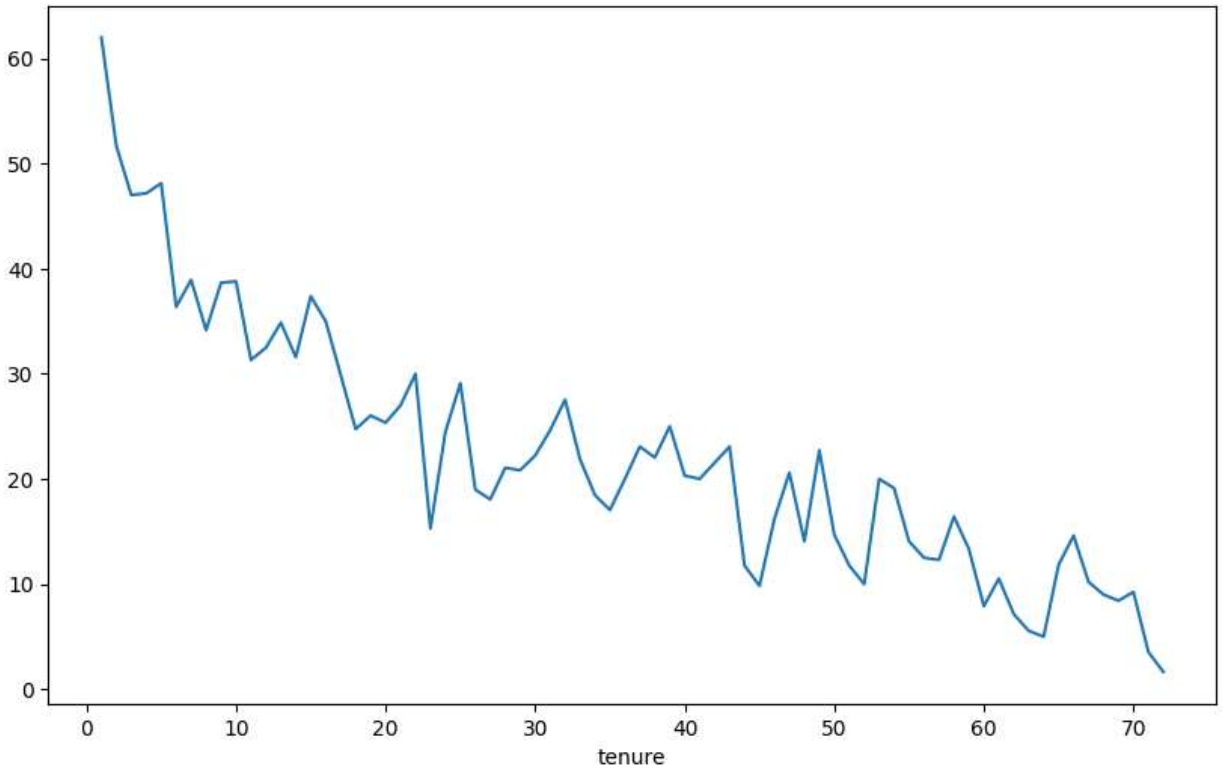
Out[ ]:
```
tenure
1     61.990212
2     51.680672
3     47.000000
4     47.159091
5     48.120301
        ...
68     9.000000
69     8.421053
70     9.243697
71     3.529412
72     1.657459
Name: customerID, Length: 72, dtype: float64
```

In [ ]:
```

```

TASK: Now that you have Churn Rate per tenure group 1-72 months, create a plot showing churn rate per months of tenure.

In [ ]:
```
churn_rate.T['customerID'].plot()
```

Out[ ]:
```
<AxesSubplot:xlabel='tenure'>
```



Broader Cohort Groups

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

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

In [ ]:
```
df['tenure'] <=12
```

Out[ ]:
```
0        True
1       False
2        True
3       False
4        True
        ...
7027    False
7028    False
7029     True
7030     True
7031    False
Name: tenure, Length: 7032, dtype: bool
```

In [ ]:
```
def cohort(tenure):
    if tenure<12:
        return '0-12 Months'
    elif tenure<25:
        return '12-24 Months'
```

```
    elif tenure <49:
        return '24-48 Months'
    else:
        return 'Over 48 Months'
```

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

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

Out[ ]:

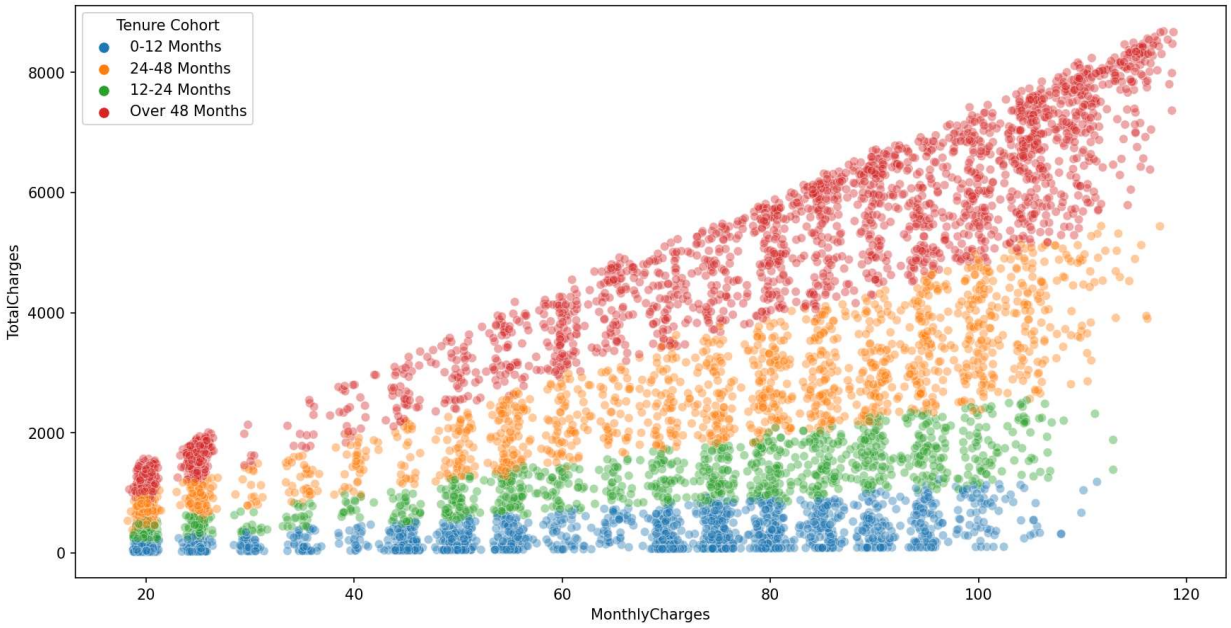
	Tenure Cohort	tenure
0	0-12 Months	1
1	24-48 Months	34
2	0-12 Months	2
3	24-48 Months	45
4	0-12 Months	2
...	...	...
7027	12-24 Months	24
7028	Over 48 Months	72
7029	0-12 Months	11
7030	0-12 Months	4
7031	Over 48 Months	66

7032 rows × 2 columns

TASK: Create a scatterplot of Total Charges versus Monthly Charts,colored by Tenure Cohort defined in the previous task.

In [ ]:
```
plt.figure(figsize=(14,7),dpi=150)
sns.scatterplot(data=df,x="MonthlyCharges",y='TotalCharges',hue='Tenure Cohort',alpha=
```

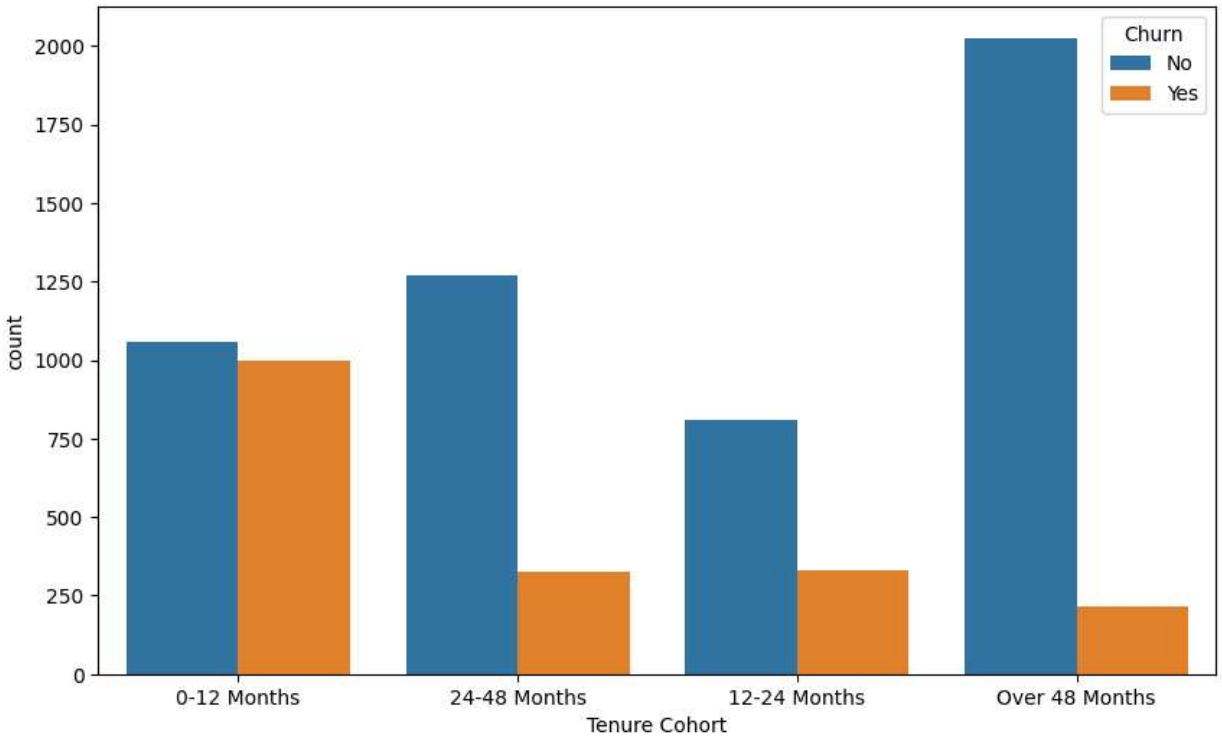
Out[ ]:
```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='TotalCharges'>
```



TASK: Create a count plot showing the churn count per cohort.

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

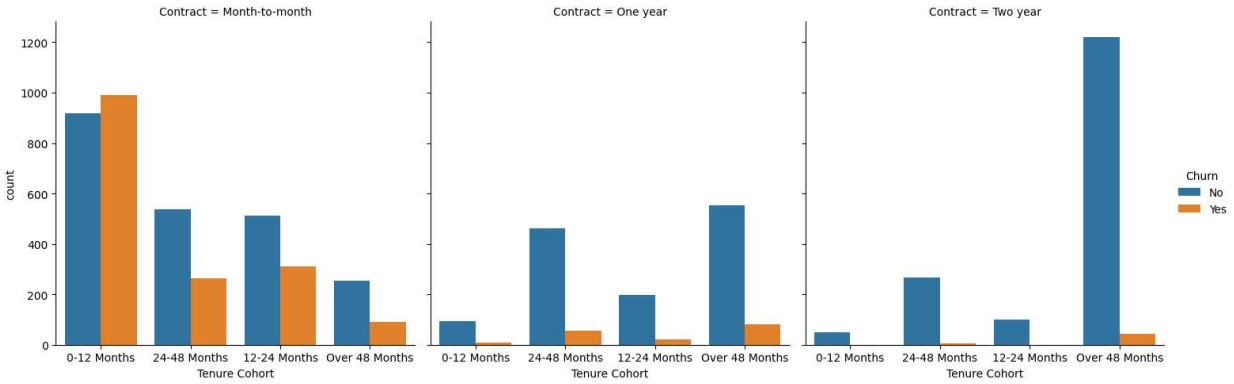
Out[ ]:
```
<AxesSubplot:xlabel='Tenure Cohort', ylabel='count'>
```



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

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

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x20580244d30>
```



Part 4: Predictive Modeling

Let's explore 4 different tree based methods: A Single Decision Tree, Random Forest, AdaBoost, Gradient Boosting. Feel free to add any other supervised learning models to your comparisons!

Single Decision Tree

TASK : Separate out the data into X features and Y label. Create dummy variables where necessary and note which features are not useful and should be dropped.

In [ ]:
```
X = df.drop(['Churn','customerID'],axis=1)
```

In [ ]:
```
X = pd.get_dummies(X,drop_first=True)
```

In [ ]:
```
y= df['Churn']
```

TASK: Perform a train test split, holding out 10% of the data for testing. We'll use a random_state of 101 in the solutions notebook/video.

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

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

TASK: Decision Tree Perfomance. Complete the following tasks:

1. Train a single decision tree model (feel free to grid search for optimal hyperparameters).
2. Evaluate performance metrics from decision tree, including classification report and plotting a confusion matrix.
3. Calculate feature importances from the decision tree.
4. OPTIONAL: Plot your tree, note, the tree could be huge depending on your pruning, so it may crash your notebook if you display it with plot_tree.

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

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

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

Out[ ]:
```
DecisionTreeClassifier(max_depth=6)
```

In [ ]:
```
from sklearn.metrics import classification_report, plot_confusion_matrix,accuracy_scor
```

In [ ]:
```
prediction = dt.predict(X_test)
```

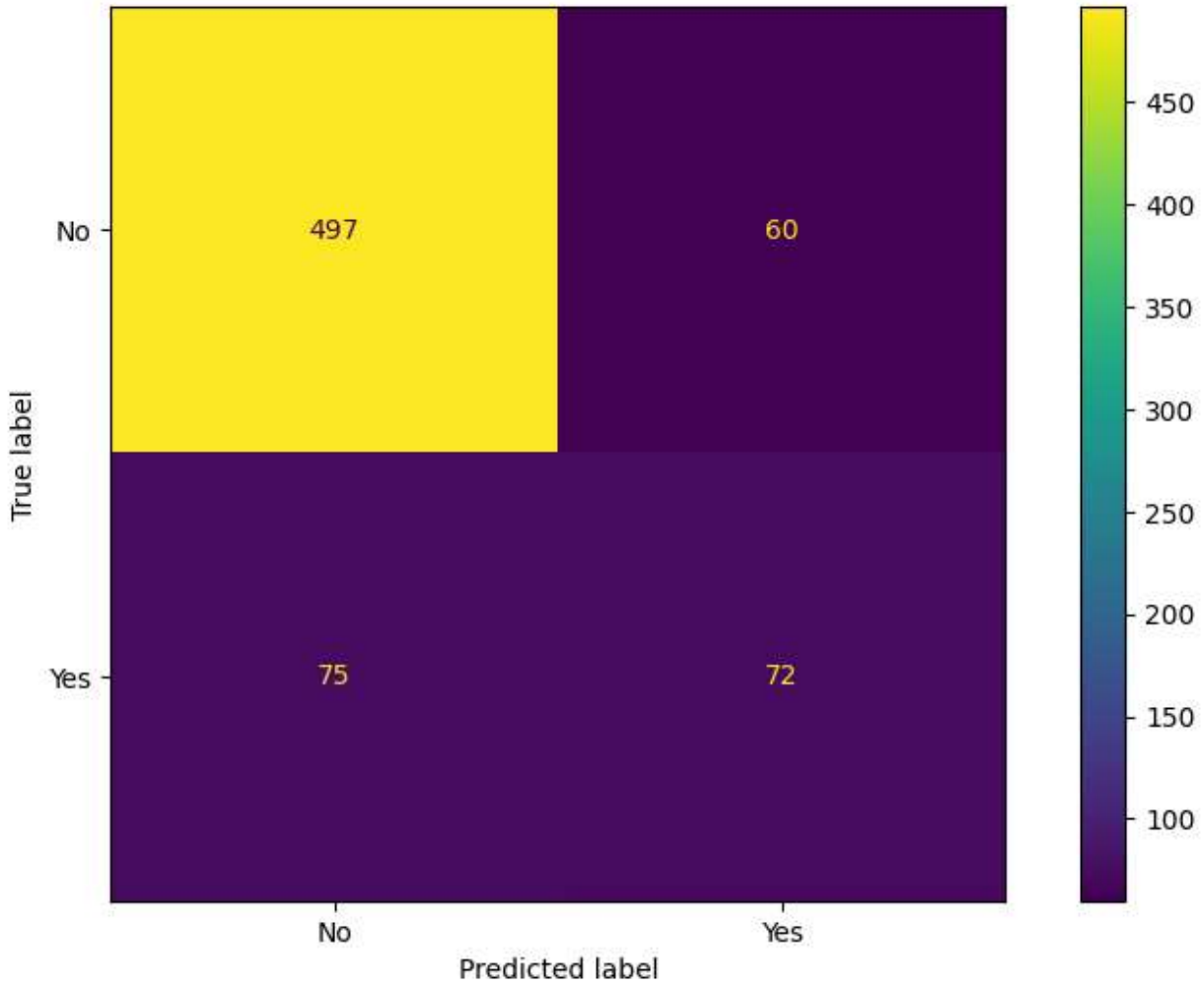
In [ ]:
```
print(classification_report(y_test,prediction))
```

```
              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
```

In [ ]:
```
plot_confusion_matrix(dt,X_test,y_test)
```

```
c:\Users\Ravi Nadageri\Python_3_10\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion
_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class method
s: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x205805f05b0>
```



In [ ]:
```
imp_feature = pd.DataFrame(data=dt.feature_importances_,index=X.columns,columns=['feat
```

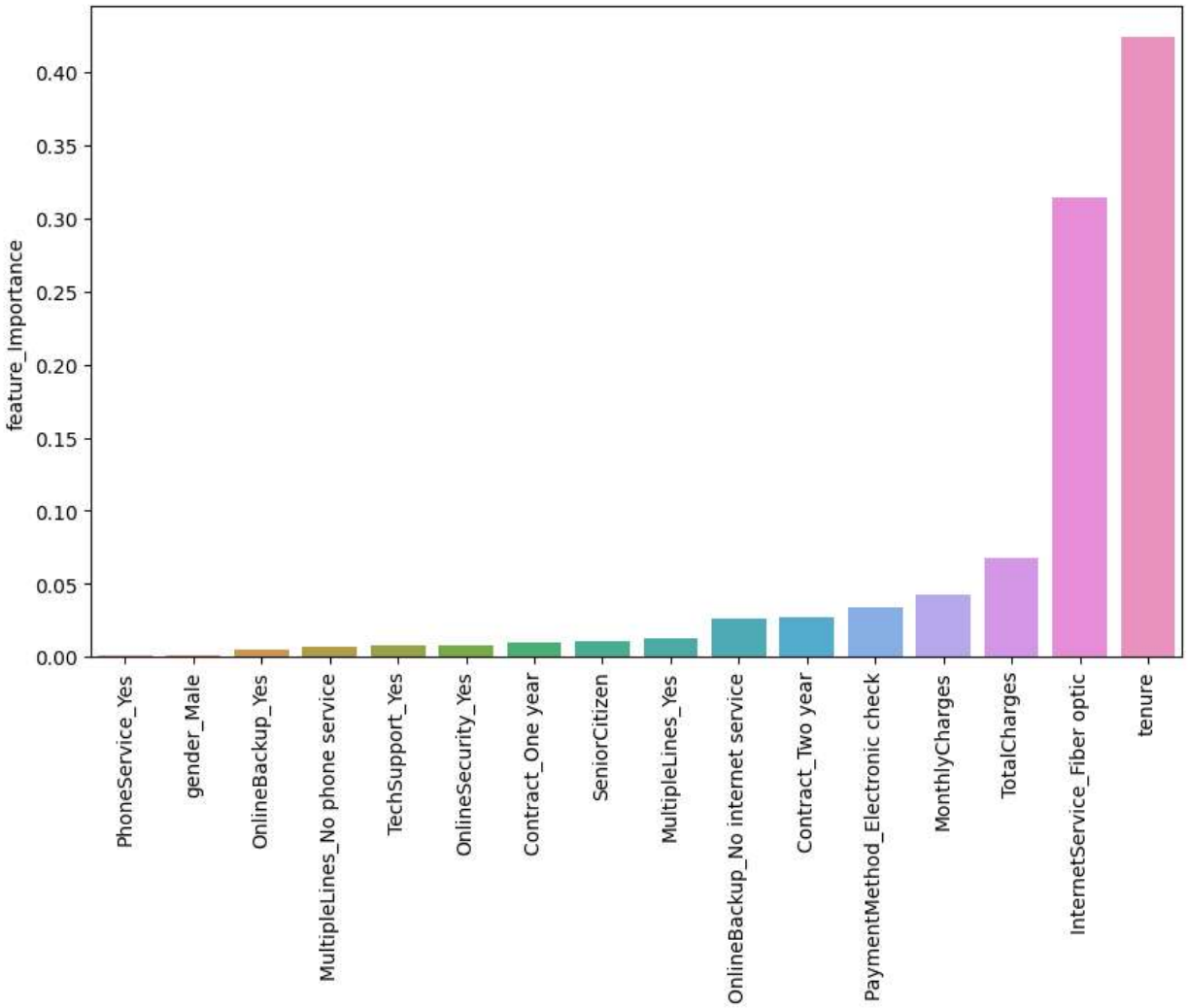
In [ ]:
```
imp_feature = imp_feature[imp_feature['feature_Importance']>0]
imp_feature
```

Out[ ]:

	feature_Importance
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
OnlineBackup_No internet service	0.026290
Contract_Two year	0.027065
PaymentMethod_Electronic check	0.034436
MonthlyCharges	0.042293
TotalCharges	0.067990
InternetService_Fiber optic	0.314060
tenure	0.423914

In [ ]:
```
sns.barplot(data=imp_feature,x=imp_feature.index,y=imp_feature['feature_Importance'])
plt.xticks(rotation=90)
```

Out[ ]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'PhoneService_Yes'),
  Text(1, 0, 'gender_Male'),
  Text(2, 0, 'OnlineBackup_Yes'),
  Text(3, 0, 'MultipleLines_No phone service'),
  Text(4, 0, 'TechSupport_Yes'),
  Text(5, 0, 'OnlineSecurity_Yes'),
  Text(6, 0, 'Contract_One year'),
  Text(7, 0, 'SeniorCitizen'),
  Text(8, 0, 'MultipleLines_Yes'),
  Text(9, 0, 'OnlineBackup_No internet service'),
  Text(10, 0, 'Contract_Two year'),
  Text(11, 0, 'PaymentMethod_Electronic check'),
  Text(12, 0, 'MonthlyCharges'),
  Text(13, 0, 'TotalCharges'),
  Text(14, 0, 'InternetService_Fiber optic'),
  Text(15, 0, 'tenure')])
```



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

In [ ]:
```
plot_tree(dt)
```

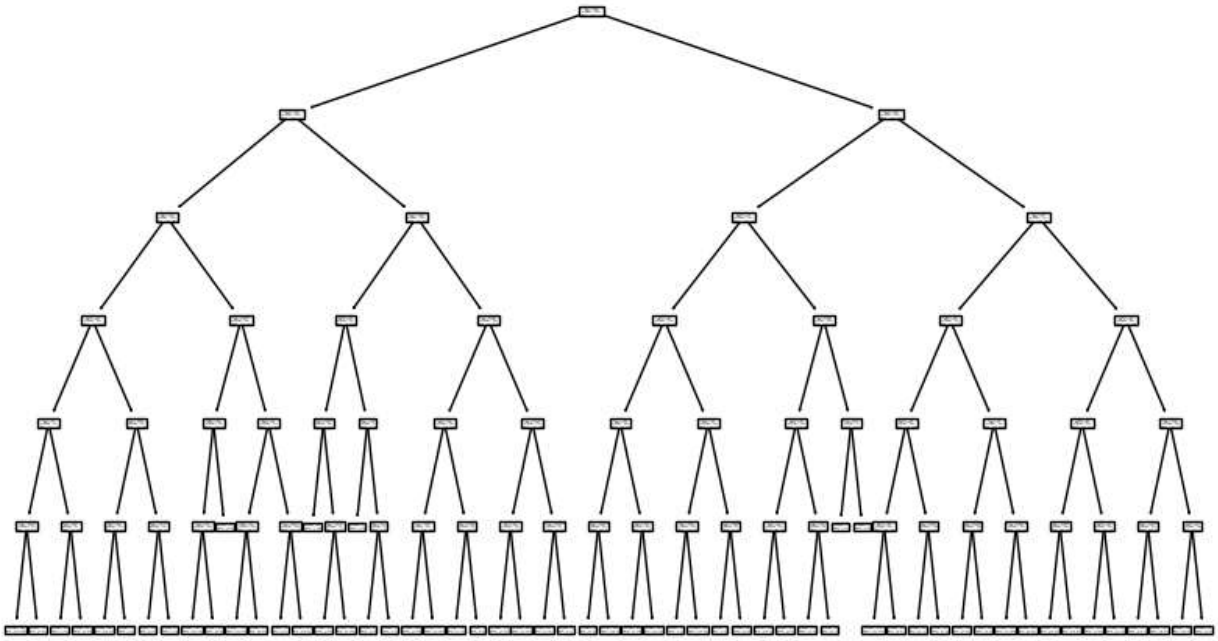
Out[ ]:
```
[Text(0.48522727272727273, 0.9285714285714286, 'X[1] <= 17.5\ngini = 0.396\nsamples =
6328\nvalue = [4606, 1722]'),
 Text(0.2375, 0.7857142857142857, 'X[10] <= 0.5\ngini = 0.497\nsamples = 2387\nvalue
= [1287, 1100]'),
 Text(0.1340909090909091, 0.6428571428571429, 'X[1] <= 5.5\ngini = 0.412\nsamples = 1
343\nvalue = [954, 389]'),
 Text(0.07272727272727272, 0.5, 'X[14] <= 0.5\ngini = 0.482\nsamples = 696\nvalue =
[414, 282]'),
 Text(0.03636363636363636, 0.35714285714285715, 'X[0] <= 0.5\ngini = 0.5\nsamples = 4
21\nvalue = [207, 214]'),
 Text(0.01818181818181818, 0.21428571428571427, 'X[2] <= 55.225\ngini = 0.499\nsample
s = 370\nvalue = [194, 176]'),
 Text(0.00909090909090909, 0.07142857142857142, 'gini = 0.5\nsamples = 298\nvalue =
[145, 153]'),
 Text(0.02727272727272727, 0.07142857142857142, 'gini = 0.435\nsamples = 72\nvalue =
[49, 23]'),
 Text(0.05454545454545454, 0.21428571428571427, 'X[3] <= 43.65\ngini = 0.38\nsamples
= 51\nvalue = [13, 38]'),
 Text(0.045454545454545456, 0.07142857142857142, 'gini = 0.0\nsamples = 13\nvalue =
[0, 13]'),
 Text(0.06363636363636363, 0.07142857142857142, 'gini = 0.45\nsamples = 38\nvalue =
[13, 25]'),
 Text(0.10909090909090909, 0.35714285714285715, 'X[3] <= 36.35\ngini = 0.372\nsamples
= 275\nvalue = [207, 68]'),
 Text(0.09090909090909091, 0.21428571428571427, 'X[0] <= 0.5\ngini = 0.456\nsamples =
162\nvalue = [105, 57]'),
 Text(0.08181818181818182, 0.07142857142857142, 'gini = 0.449\nsamples = 159\nvalue =
[105, 54]'),
 Text(0.1, 0.07142857142857142, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.12727272727272726, 0.21428571428571427, 'X[3] <= 107.55\ngini = 0.176\nsample
s = 113\nvalue = [102, 11]'),
 Text(0.11818181818181818, 0.07142857142857142, 'gini = 0.142\nsamples = 104\nvalue =
[96, 8]'),
 Text(0.13636363636363635, 0.07142857142857142, 'gini = 0.444\nsamples = 9\nvalue =
[6, 3]'),
 Text(0.19545454545454546, 0.5, 'X[2] <= 25.325\ngini = 0.276\nsamples = 647\nvalue =
[540, 107]'),
 Text(0.17272727272727273, 0.35714285714285715, 'X[25] <= 0.5\ngini = 0.149\nsamples
= 271\nvalue = [249, 22]'),
 Text(0.16363636363636364, 0.21428571428571427, 'X[24] <= 0.5\ngini = 0.179\nsamples
= 222\nvalue = [200, 22]'),
 Text(0.15454545454545454, 0.07142857142857142, 'gini = 0.233\nsamples = 141\nvalue =
[122, 19]'),
 Text(0.17272727272727273, 0.07142857142857142, 'gini = 0.071\nsamples = 81\nvalue =
[78, 3]'),
 Text(0.18181818181818182, 0.21428571428571427, 'gini = 0.0\nsamples = 49\nvalue = [4
9, 0]'),
 Text(0.21818181818181817, 0.35714285714285715, 'X[15] <= 0.5\ngini = 0.35\nsamples =
376\nvalue = [291, 85]'),
 Text(0.2, 0.21428571428571427, 'X[28] <= 0.5\ngini = 0.404\nsamples = 253\nvalue =
[182, 71]'),
 Text(0.19090909090909092, 0.07142857142857142, 'gini = 0.334\nsamples = 170\nvalue =
[134, 36]'),
 Text(0.20909090909090908, 0.07142857142857142, 'gini = 0.488\nsamples = 83\nvalue =
[48, 35]'),
 Text(0.23636363636363636, 0.21428571428571427, 'X[3] <= 348.025\ngini = 0.202\nsampl
es = 123\nvalue = [109, 14]'),
```

```
 Text(0.22727272727272727, 0.07142857142857142, 'gini = 0.497\nsamples = 13\nvalue =
[7, 6]'),
 Text(0.24545454545454545, 0.07142857142857142, 'gini = 0.135\nsamples = 110\nvalue =
[102, 8]'),
 Text(0.3409090909090909, 0.6428571428571429, 'X[3] <= 120.0\ngini = 0.434\nsamples =
1044\nvalue = [333, 711]'),
 Text(0.2818181818181818, 0.5, 'X[19] <= 0.5\ngini = 0.209\nsamples = 211\nvalue = [2
5, 186]'),
 Text(0.2636363636363636, 0.35714285714285715, 'X[2] <= 69.875\ngini = 0.185\nsamples
= 204\nvalue = [21, 183]'),
 Text(0.2545454545454545, 0.21428571428571427, 'gini = 0.0\nsamples = 32\nvalue = [0,
32]'),
 Text(0.2727272727272727, 0.21428571428571427, 'X[3] <= 70.725\ngini = 0.214\nsamples
= 172\nvalue = [21, 151]'),
 Text(0.2636363636363636, 0.07142857142857142, 'gini = 0.404\nsamples = 32\nvalue =
[9, 23]'),
 Text(0.2818181818181818, 0.07142857142857142, 'gini = 0.157\nsamples = 140\nvalue =
[12, 128]'),
 Text(0.3, 0.35714285714285715, 'X[3] <= 75.3\ngini = 0.49\nsamples = 7\nvalue = [4,
3]'),
 Text(0.2909090909090909, 0.21428571428571427, 'gini = 0.0\nsamples = 2\nvalue = [2,
0]'),
 Text(0.3090909090909091, 0.21428571428571427, 'X[4] <= 0.5\ngini = 0.48\nsamples = 5
\nvalue = [2, 3]'),
 Text(0.3, 0.07142857142857142, 'gini = 0.444\nsamples = 3\nvalue = [2, 1]'),
 Text(0.3181818181818182, 0.07142857142857142, 'gini = 0.0\nsamples = 2\nvalue = [0,
2]'),
 Text(0.4, 0.5, 'X[2] <= 80.575\ngini = 0.466\nsamples = 833\nvalue = [308, 525]'),
 Text(0.36363636363636365, 0.35714285714285715, 'X[13] <= 0.5\ngini = 0.496\nsamples
= 359\nvalue = [164, 195]'),
 Text(0.34545454545454546, 0.21428571428571427, 'X[2] <= 73.525\ngini = 0.49\nsamples
= 323\nvalue = [139, 184]'),
 Text(0.33636363636363636, 0.07142857142857142, 'gini = 0.494\nsamples = 108\nvalue =
[60, 48]'),
 Text(0.35454545454545455, 0.07142857142857142, 'gini = 0.465\nsamples = 215\nvalue =
[79, 136]'),
 Text(0.38181818181818183, 0.21428571428571427, 'X[2] <= 79.725\ngini = 0.424\nsample
s = 36\nvalue = [25, 11]'),
 Text(0.37272727272727274, 0.07142857142857142, 'gini = 0.477\nsamples = 28\nvalue =
[17, 11]'),
 Text(0.39090909090909093, 0.07142857142857142, 'gini = 0.0\nsamples = 8\nvalue = [8,
0]'),
 Text(0.43636363636363634, 0.35714285714285715, 'X[3] <= 1271.75\ngini = 0.423\nsampl
es = 474\nvalue = [144, 330]'),
 Text(0.41818181818181815, 0.21428571428571427, 'X[9] <= 0.5\ngini = 0.394\nsamples =
382\nvalue = [103, 279]'),
 Text(0.4090909090909091, 0.07142857142857142, 'gini = 0.457\nsamples = 173\nvalue =
[61, 112]'),
 Text(0.42727272727272725, 0.07142857142857142, 'gini = 0.321\nsamples = 209\nvalue =
[42, 167]'),
 Text(0.45454545454545453, 0.21428571428571427, 'X[3] <= 1632.825\ngini = 0.494\nsamp
les = 92\nvalue = [41, 51]'),
 Text(0.44545454545454544, 0.07142857142857142, 'gini = 0.484\nsamples = 83\nvalue =
[34, 49]'),
 Text(0.4636363636363636, 0.07142857142857142, 'gini = 0.346\nsamples = 9\nvalue =
[7, 2]'),
 Text(0.7329545454545454, 0.7857142857142857, 'X[10] <= 0.5\ngini = 0.266\nsamples =
```

```
3941\nvalue = [3319, 622]'),
 Text(0.6113636363636363, 0.6428571428571429, 'X[25] <= 0.5\ngini = 0.114\nsamples =
2170\nvalue = [2038, 132]'),
 Text(0.5454545454545454, 0.5, 'X[8] <= 0.5\ngini = 0.185\nsamples = 1133\nvalue = [1
016, 117]'),
 Text(0.509090909090909, 0.35714285714285715, 'X[2] <= 48.1\ngini = 0.14\nsamples = 8
83\nvalue = [816, 67]'),
 Text(0.4909090909090909, 0.21428571428571427, 'X[3] <= 391.275\ngini = 0.055\nsample
s = 321\nvalue = [312, 9]'),
 Text(0.4818181818181818, 0.07142857142857142, 'gini = 0.188\nsamples = 19\nvalue =
[17, 2]'),
 Text(0.5, 0.07142857142857142, 'gini = 0.045\nsamples = 302\nvalue = [295, 7]'),
 Text(0.5272727272727272, 0.21428571428571427, 'X[1] <= 22.5\ngini = 0.185\nsamples =
562\nvalue = [504, 58]'),
 Text(0.5181818181818182, 0.07142857142857142, 'gini = 0.343\nsamples = 82\nvalue =
[64, 18]'),
 Text(0.5363636363636364, 0.07142857142857142, 'gini = 0.153\nsamples = 480\nvalue =
[440, 40]'),
 Text(0.5818181818181818, 0.35714285714285715, 'X[19] <= 0.5\ngini = 0.32\nsamples =
250\nvalue = [200, 50]'),
 Text(0.5636363636363636, 0.21428571428571427, 'X[1] <= 18.5\ngini = 0.388\nsamples =
163\nvalue = [120, 43]'),
 Text(0.5545454545454546, 0.07142857142857142, 'gini = 0.32\nsamples = 5\nvalue = [1,
4]'),
 Text(0.5727272727272728, 0.07142857142857142, 'gini = 0.372\nsamples = 158\nvalue =
[119, 39]'),
 Text(0.6, 0.21428571428571427, 'X[2] <= 29.025\ngini = 0.148\nsamples = 87\nvalue =
[80, 7]'),
 Text(0.5909090909090909, 0.07142857142857142, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.6090909090909091, 0.07142857142857142, 'gini = 0.13\nsamples = 86\nvalue = [8
0, 6]'),
 Text(0.6772727272727272, 0.5, 'X[2] <= 92.425\ngini = 0.029\nsamples = 1037\nvalue =
[1022, 15]'),
 Text(0.6545454545454545, 0.35714285714285715, 'X[28] <= 0.5\ngini = 0.027\nsamples =
1032\nvalue = [1018, 14]'),
 Text(0.6363636363636364, 0.21428571428571427, 'X[0] <= 0.5\ngini = 0.021\nsamples =
964\nvalue = [954, 10]'),
 Text(0.6272727272727273, 0.07142857142857142, 'gini = 0.015\nsamples = 902\nvalue =
[895, 7]'),
 Text(0.6454545454545455, 0.07142857142857142, 'gini = 0.092\nsamples = 62\nvalue =
[59, 3]'),
 Text(0.6727272727272727, 0.21428571428571427, 'X[7] <= 0.5\ngini = 0.111\nsamples =
68\nvalue = [64, 4]'),
 Text(0.6636363636363637, 0.07142857142857142, 'gini = 0.32\nsamples = 15\nvalue = [1
2, 3]'),
 Text(0.6818181818181818, 0.07142857142857142, 'gini = 0.037\nsamples = 53\nvalue =
[52, 1]'),
 Text(0.7, 0.35714285714285715, 'X[3] <= 6473.2\ngini = 0.32\nsamples = 5\nvalue =
[4, 1]'),
 Text(0.6909090909090909, 0.21428571428571427, 'gini = 0.0\nsamples = 1\nvalue = [0,
1]'),
 Text(0.7090909090909091, 0.21428571428571427, 'gini = 0.0\nsamples = 4\nvalue = [4,
0]'),
 Text(0.8545454545454545, 0.6428571428571429, 'X[1] <= 50.5\ngini = 0.4\nsamples = 17
71\nvalue = [1281, 490]'),
 Text(0.7818181818181819, 0.5, 'X[28] <= 0.5\ngini = 0.475\nsamples = 915\nvalue = [5
```

```
60, 355]'),
 Text(0.7454545454545455, 0.35714285714285715, 'X[24] <= 0.5\ngini = 0.408\nsamples =
448\nvalue = [320, 128]'),
 Text(0.7272727272727273, 0.21428571428571427, 'X[25] <= 0.5\ngini = 0.446\nsamples =
345\nvalue = [229, 116]'),
 Text(0.7181818181818181, 0.07142857142857142, 'gini = 0.462\nsamples = 315\nvalue =
[201, 114]'),
 Text(0.7363636363636363, 0.07142857142857142, 'gini = 0.124\nsamples = 30\nvalue =
[28, 2]'),
 Text(0.7636363636363637, 0.21428571428571427, 'X[3] <= 4779.1\ngini = 0.206\nsamples
= 103\nvalue = [91, 12]'),
 Text(0.7545454545454545, 0.07142857142857142, 'gini = 0.153\nsamples = 84\nvalue =
[77, 7]'),
 Text(0.7727272727272727, 0.07142857142857142, 'gini = 0.388\nsamples = 19\nvalue =
[14, 5]'),
 Text(0.8181818181818182, 0.35714285714285715, 'X[9] <= 0.5\ngini = 0.5\nsamples = 46
7\nvalue = [240, 227]'),
 Text(0.8, 0.21428571428571427, 'X[3] <= 3210.65\ngini = 0.463\nsamples = 148\nvalue
= [94, 54]'),
 Text(0.7909090909090909, 0.07142857142857142, 'gini = 0.486\nsamples = 113\nvalue =
[66, 47]'),
 Text(0.8090909090909091, 0.07142857142857142, 'gini = 0.32\nsamples = 35\nvalue = [2
8, 7]'),
 Text(0.8363636363636363, 0.21428571428571427, 'X[3] <= 3188.475\ngini = 0.496\nsampl
es = 319\nvalue = [146, 173]'),
 Text(0.8272727272727273, 0.07142857142857142, 'gini = 0.461\nsamples = 169\nvalue =
[61, 108]'),
 Text(0.8454545454545455, 0.07142857142857142, 'gini = 0.491\nsamples = 150\nvalue =
[85, 65]'),
 Text(0.9272727272727272, 0.5, 'X[25] <= 0.5\ngini = 0.266\nsamples = 856\nvalue = [7
21, 135]'),
 Text(0.8909090909090909, 0.35714285714285715, 'X[28] <= 0.5\ngini = 0.342\nsamples =
508\nvalue = [397, 111]'),
 Text(0.8727272727272727, 0.21428571428571427, 'X[2] <= 99.425\ngini = 0.259\nsamples
= 295\nvalue = [250, 45]'),
 Text(0.8636363636363636, 0.07142857142857142, 'gini = 0.153\nsamples = 144\nvalue =
[132, 12]'),
 Text(0.8818181818181818, 0.07142857142857142, 'gini = 0.342\nsamples = 151\nvalue =
[118, 33]'),
 Text(0.9090909090909091, 0.21428571428571427, 'X[13] <= 0.5\ngini = 0.428\nsamples =
213\nvalue = [147, 66]'),
 Text(0.9, 0.07142857142857142, 'gini = 0.464\nsamples = 142\nvalue = [90, 52]'),
 Text(0.9181818181818182, 0.07142857142857142, 'gini = 0.317\nsamples = 71\nvalue =
[57, 14]'),
 Text(0.9636363636363636, 0.35714285714285715, 'X[1] <= 71.5\ngini = 0.128\nsamples =
348\nvalue = [324, 24]'),
 Text(0.9454545454545454, 0.21428571428571427, 'X[2] <= 105.925\ngini = 0.189\nsample
s = 218\nvalue = [195, 23]'),
 Text(0.9363636363636364, 0.07142857142857142, 'gini = 0.112\nsamples = 117\nvalue =
[110, 7]'),
 Text(0.9545454545454546, 0.07142857142857142, 'gini = 0.267\nsamples = 101\nvalue =
[85, 16]'),
 Text(0.9818181818181818, 0.21428571428571427, 'X[0] <= 0.5\ngini = 0.015\nsamples =
130\nvalue = [129, 1]'),
 Text(0.9727272727272728, 0.07142857142857142, 'gini = 0.0\nsamples = 106\nvalue = [1
06, 0]'),
```

```
 Text(0.990909090909091, 0.07142857142857142, 'gini = 0.08\nsamples = 24\nvalue = [2
3, 1]')]
```



Random Forest

TASK: Create a Random Forest model and create a classification report and confusion matrix from its predicted results on the test set.

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

In [ ]:
```
rfc = RandomForestClassifier(max_depth=6)
```

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

Out[ ]:
```
RandomForestClassifier(max_depth=6)
```

In [ ]:
```
pred = rfc.predict(X_test)
```

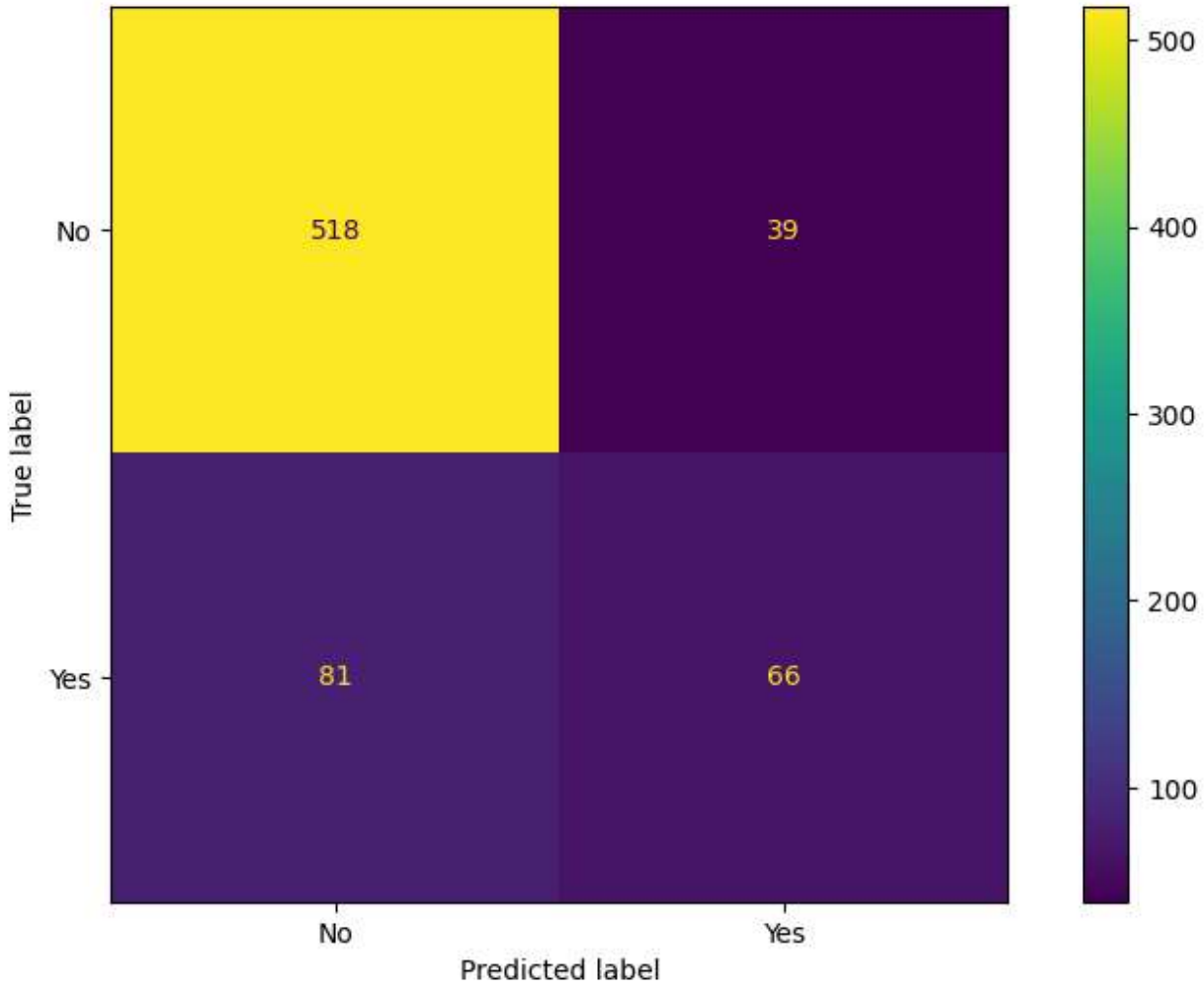
In [ ]:
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

          No       0.86      0.93      0.90       557
         Yes       0.63      0.45      0.52       147

    accuracy                           0.83       704
   macro avg       0.75      0.69      0.71       704
weighted avg       0.82      0.83      0.82       704
```

In [ ]:
```
plot_confusion_matrix(rfc,X_test,y_test)
```

```
c:\Users\Ravi Nadageri\Python_3_10\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion
_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class method
s: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2058205a5f0>



Boosted Trees

TASK: Use AdaBoost or Gradient Boosting to create a model and report back the classification report and plot a confusion matrix for its predicted results

In [ ]:
```
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier
```

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

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

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

Out[ ]:
```
AdaBoostClassifier

AdaBoostClassifier()
```

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

Out[ ]:
```
GradientBoostingClassifier

GradientBoostingClassifier()
```

In [ ]:
```
ada_pred = ada_model.predict(X_test)
gb_pred = gb_model.predict(X_test)
```

In [ ]:
```
print(classification_report(y_test,ada_pred))
```

```
              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
```

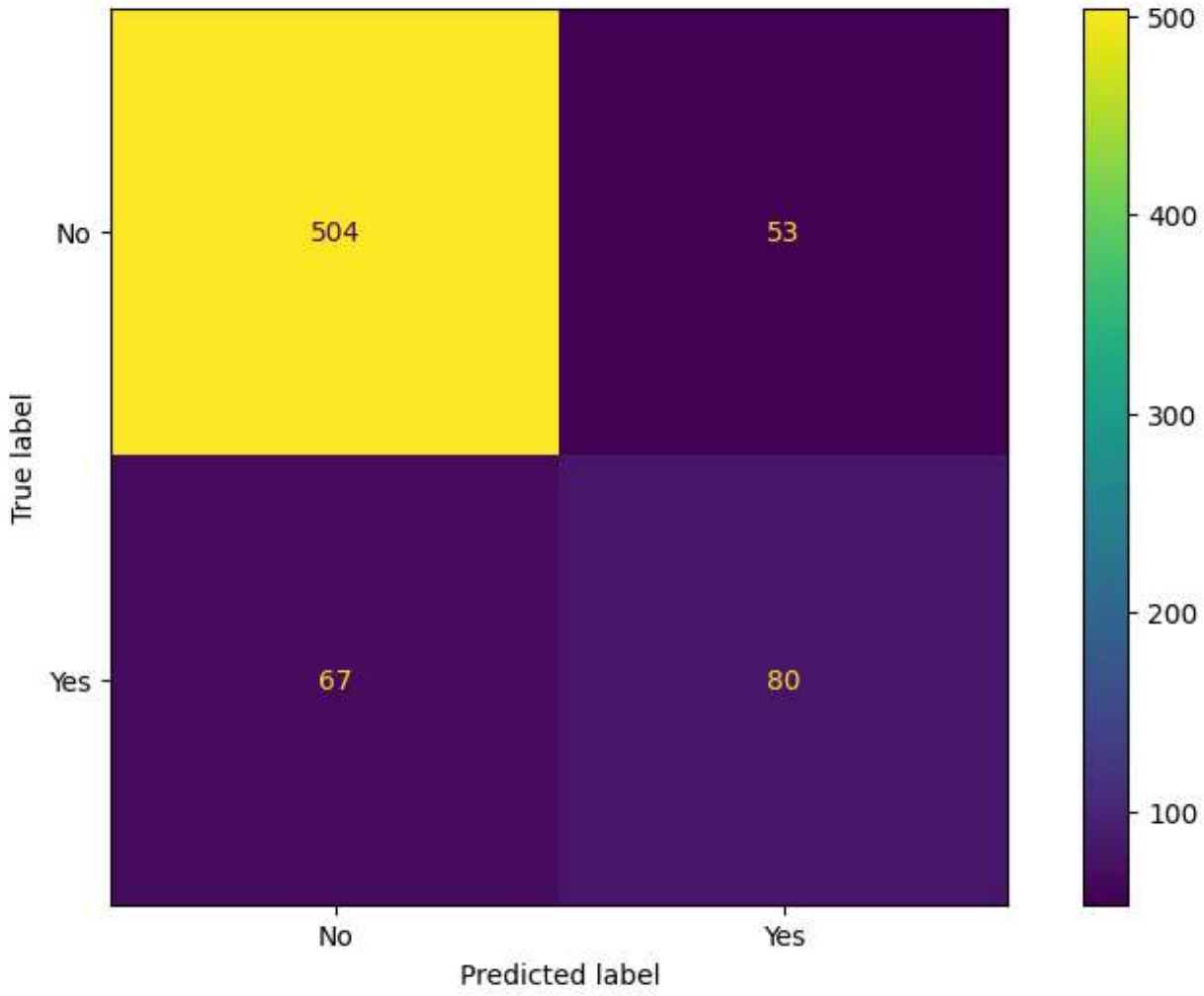
In [ ]:
```
print(classification_report(y_test,gb_pred))
```

```
              precision    recall  f1-score   support

          No       0.87      0.90      0.89       557
         Yes       0.58      0.51      0.54       147

    accuracy                           0.82       704
   macro avg       0.73      0.71      0.71       704
weighted avg       0.81      0.82      0.82       704
```

In [ ]:
```
plot_confusion_matrix(ada_model,X_test,y_test)
```

```
c:\Users\Ravi Nadageri\Python_3_10\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion
_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class method
s: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

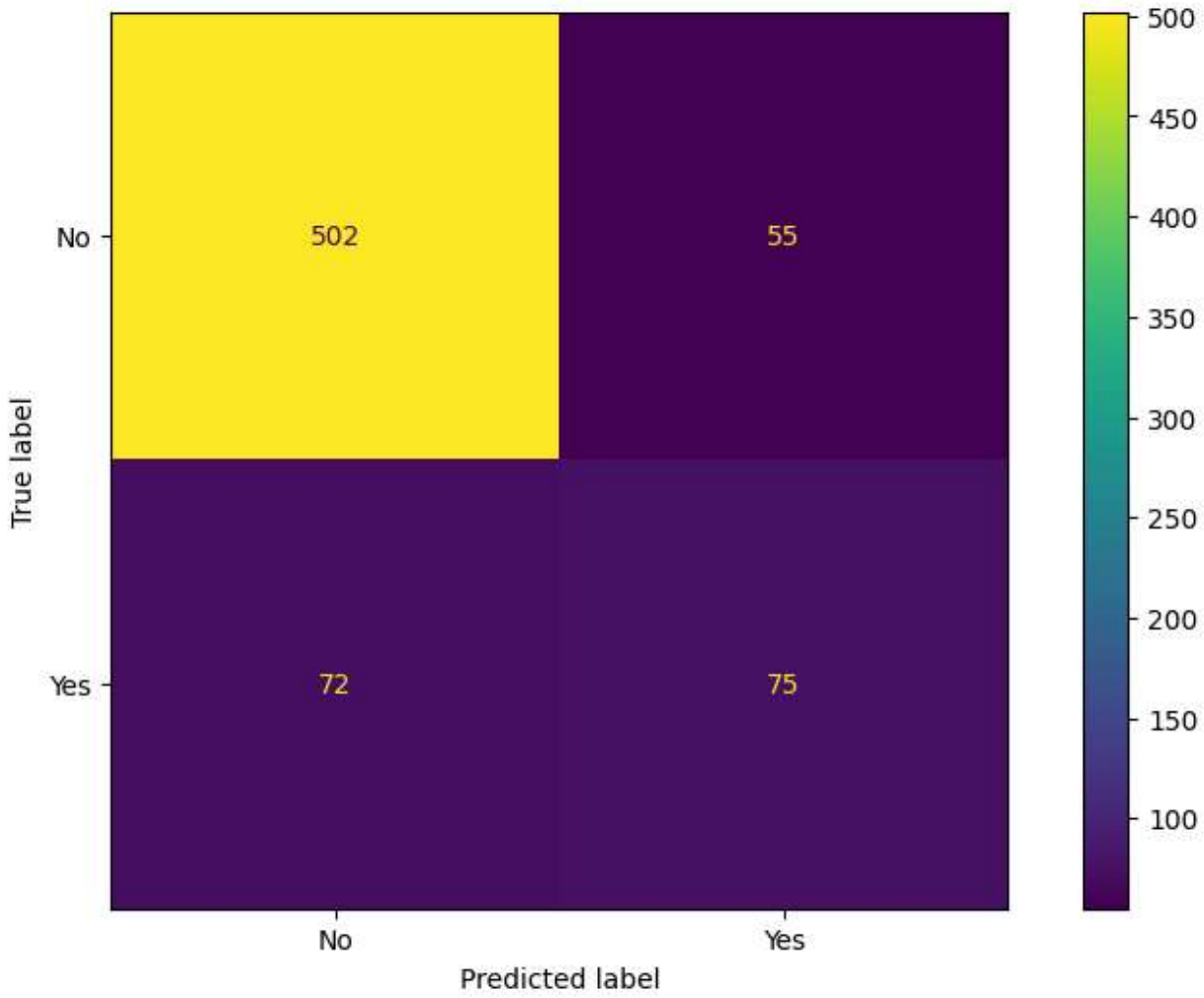
Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2058205a320>
```



In [ ]:
```
plot_confusion_matrix(gb_model,X_test,y_test)
```

```
c:\Users\Ravi Nadageri\Python_3_10\lib\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion
_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class method
s: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x205851cada0>
```



TASK: Analyze your results, which model performed best for you?

In [ ]:
```
# With base models, we got best performance from an AdaBoostClassifier, but note, we a
```

Great job!