Telephone-Internet Customer Churn Prediction Using Tree-Based and Boosting Models

Author: Kian Farahani

The Main Part of This Project:

- **Part 1: Comprehensive Exploratory Data Analysis**
- **Part 2: Churn Cohort Analysis**
- **Part 3: Tree-based and Boosting ML Predictive Models**

About the dataset:

The data set used for this article's classification problem is taken from IBM sample data set collection. The data set includes information about:

- Customers who left within the last month – the column is called Churn.
- Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies.
- Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges.
- Demographic info about customers – gender, age range, and if they have partners and dependents

In [2]:
```
# Import necessary libraries!
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
df = pd.read_csv('../DATA/Telco-Customer-Churn.csv')
```

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

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	M
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

Part 1: Quick Data Check

Confirming quickly with .info() methods the datatypes and non-null values in your dataframe.

In [5]:
```
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
```

In [6]:
```
# Get a quick statistical summary of the numeric columns with .describe()
df.describe()
```

Out[6]:

	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

Notice that many columns are categorical, meaning we will eventually need to convert them to dummy variables.

Part 2: Exploratory Data Analysis

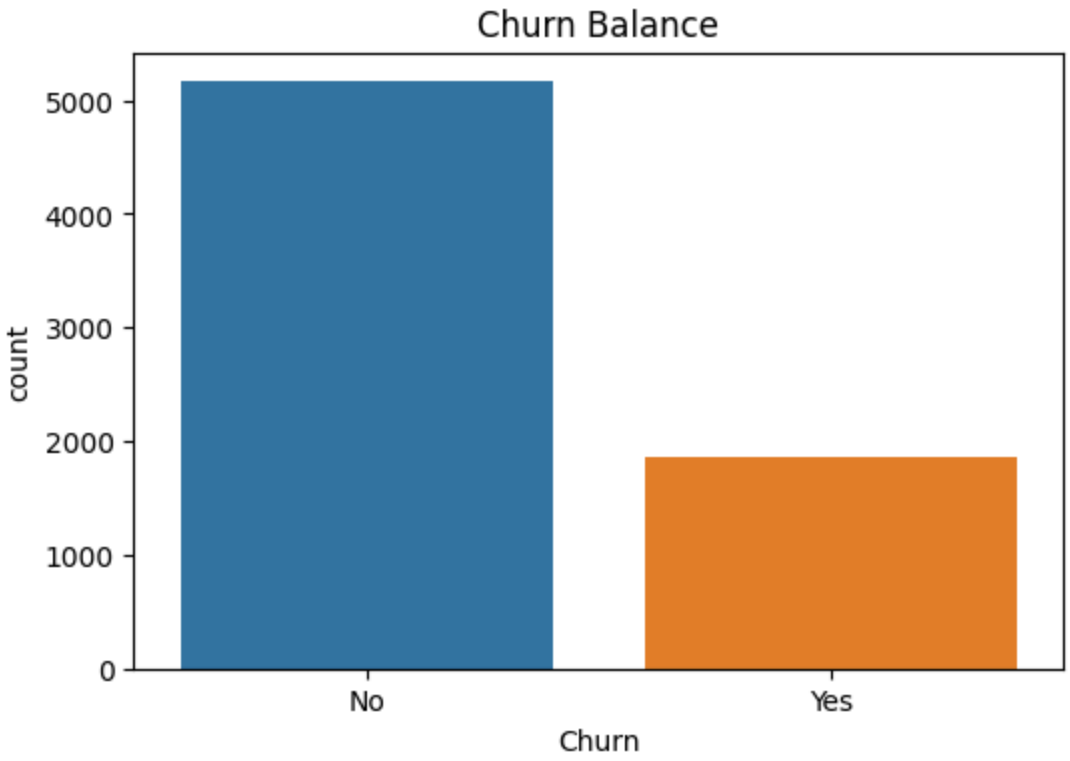
General Feature Exploration

First we need to confirm that there are no NaN cells by displaying NaN values per feature column.

In [7]:
```
df.isna().sum()
```

Out[7]:
```
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
```

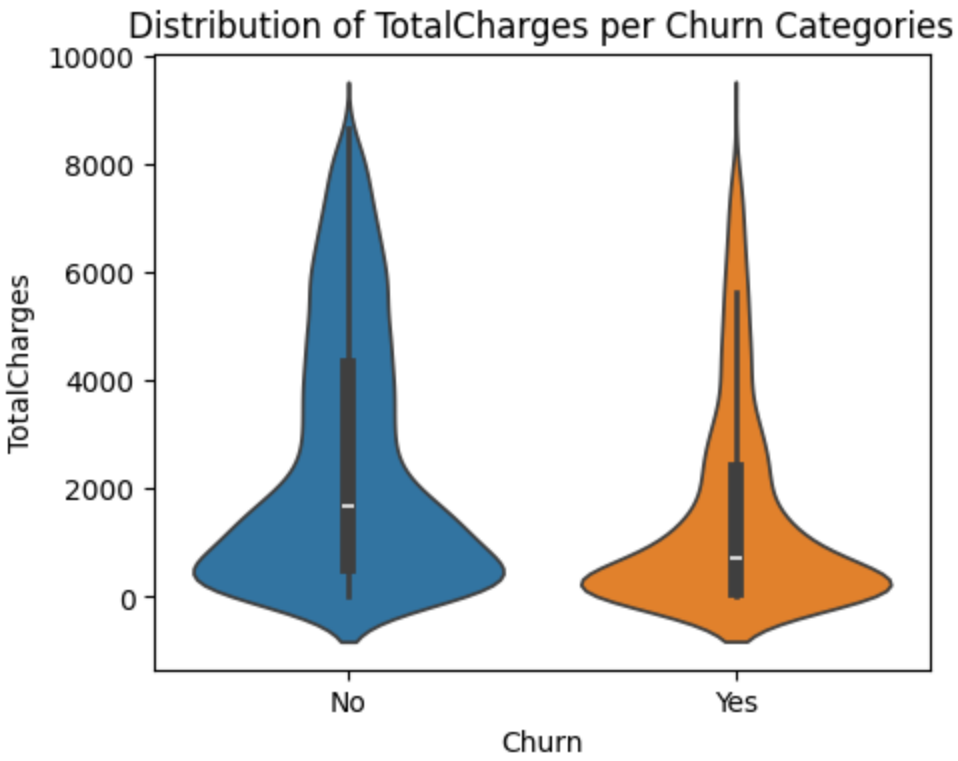
In [8]:
```
# Display the balance of the class labels (Churn)
plt.figure(figsize=(6, 4))

sns.countplot(data=df, x='Churn', hue='Churn')

plt.title('Churn Balance');
```



Although the label class is not balanced, we are not dealing with a extream unbalanced label class either. It seems that we have `~55%–60%` of the data for not churning and

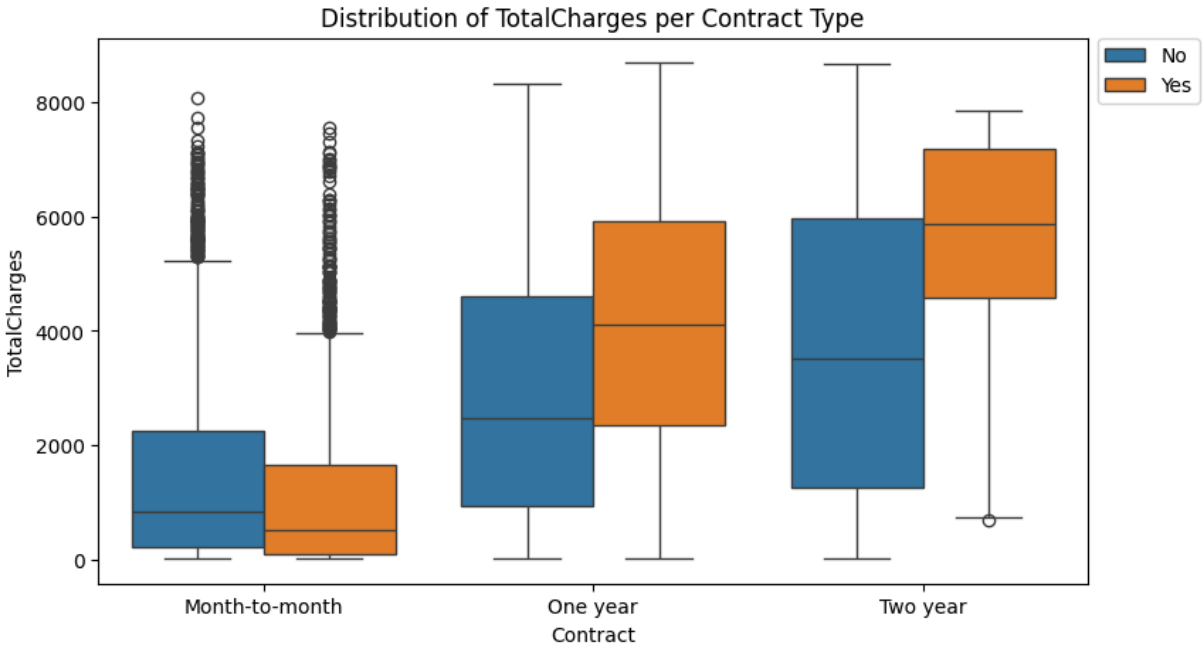
`~35%–40%` for churning. Usually a **proportion of 9 to 1 or 10 to 1** is considered to be an unbalanced class.

In [9]:
```
# Display the distribution of 'TotalCharges' between Churn categories
plt.figure(figsize=(5, 4))

sns.violinplot(data=df, x='Churn', y='TotalCharges', hue='Churn')

plt.title('Distribution of TotalCharges per Churn Categories');
```



In [ ]:
```
```

In [10]:
```
# Display the distribution of 'TotalCharges' per 'Contract type'
plt.figure(figsize=(9, 5))

sns.boxplot(data=df, x='Contract', y='TotalCharges', hue='Churn')

plt.title('Distribution of TotalCharges per Contract Type')
plt.legend(loc=(1.01, 0.88));
```



The interpretation of the plot:

- For `month-to-month` users the distribution of data for people who have churned and not churned are very similar, and it is expected to be like this! Because people who sign up for a service with a `month-to-month` subscription do not expect to remain on the service for a long time.
- However, the story is different for users who have `one year contract` and `two year contract`. These are actually users that we might want to investigate more to see how we can prevent their churn percentage. It is clear from the plot above that for both groups **the total charges increase the higher rate of churn happens**.

Therefore, from a bussiness prespective, it would be a wise step to offer a sort of **bonus program** to users with more than a year contract with the company in order to incentivise them to keep their subscription and bring down their total charges.

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 [11]:
```
# Create a list of features and the label we want to investigate
features_list = ['gender', 'SeniorCitizen', 'Partner', 'Dependents','PhoneSe
 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'Inter
   'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'Paymer

# Get the dummy variables of them and call the corr() method off of it
corr_df = pd.get_dummies(df[features_list], drop_first=False).corr()
corr_df
```

Out[11]:

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

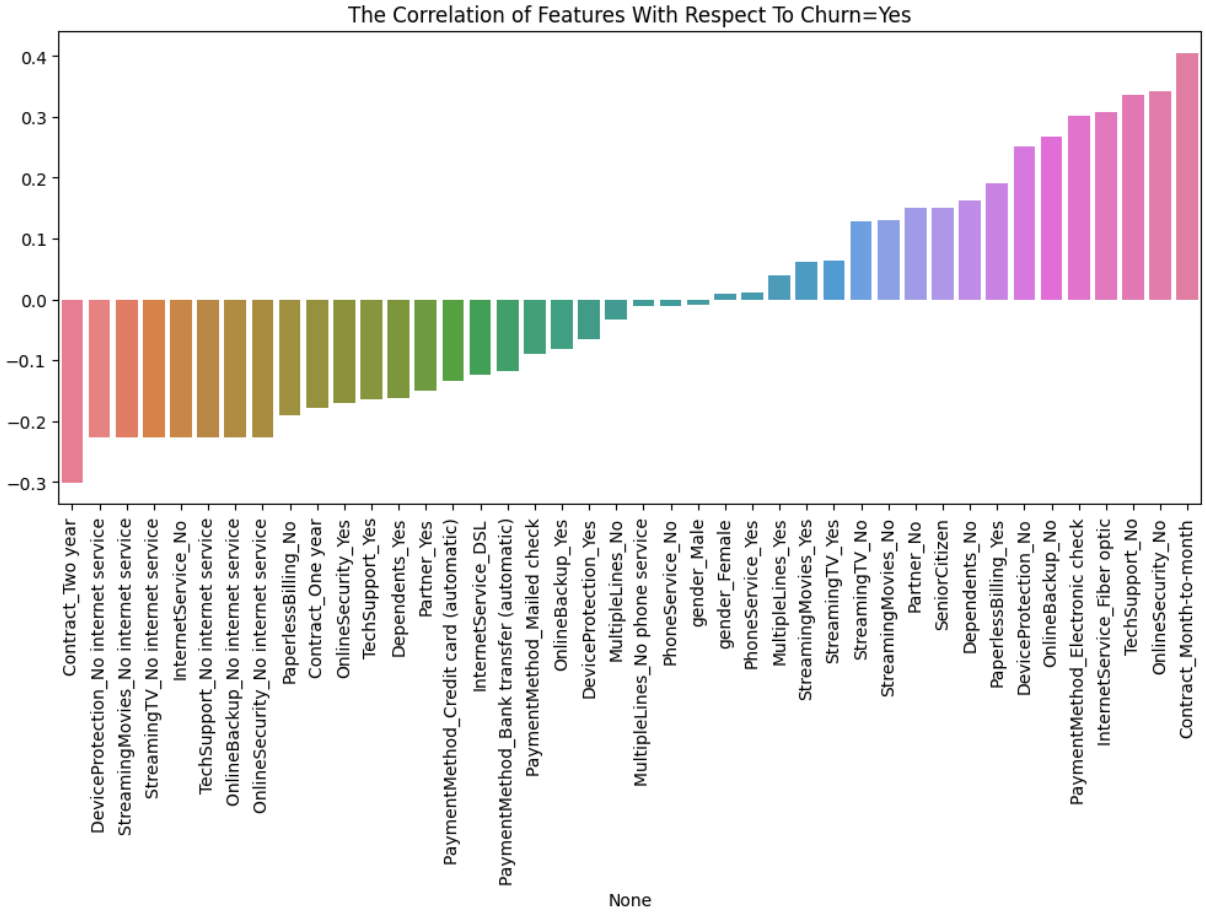
	SeniorCitizen	gender_Female	gender_Male	Partner_No	I
StreamingTV_No internet service	-0.182519	-0.004745	0.004745	0.000286	
StreamingTV_Yes	0.105445	0.007124	-0.007124	-0.124483	
StreamingMovies_No	0.034196	-0.006078	0.006078	0.117488	
StreamingMovies_No internet service	-0.182519	-0.004745	0.004745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

44 rows × 44 columns

In [12]:
```
# Get the correlations with respect to 'Churn_yes' and sort the values
churn_yes_corr = corr_df['Churn_Yes'].sort_values()[1:-1]
churn_yes_corr
```

Out[12]:
```
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 [13]:
```
# Display the result of the series above on a barplot
plt.figure(figsize=(12, 5))

sns.barplot(x=churn_yes_corr.index, y=churn_yes_corr.values, hue=churn_yes_c

plt.title('The Correlation of Features With Respect To Churn=Yes')
plt.xticks(rotation=90);
```



So the highly-correlated feature with churning=Yes is the `Contract_Month-to-month` feature which makes sence. Usually people who sign up for a monthly service are more likely to get out of their plans as opposed to people who are locked up into `two-year contracts`.

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.

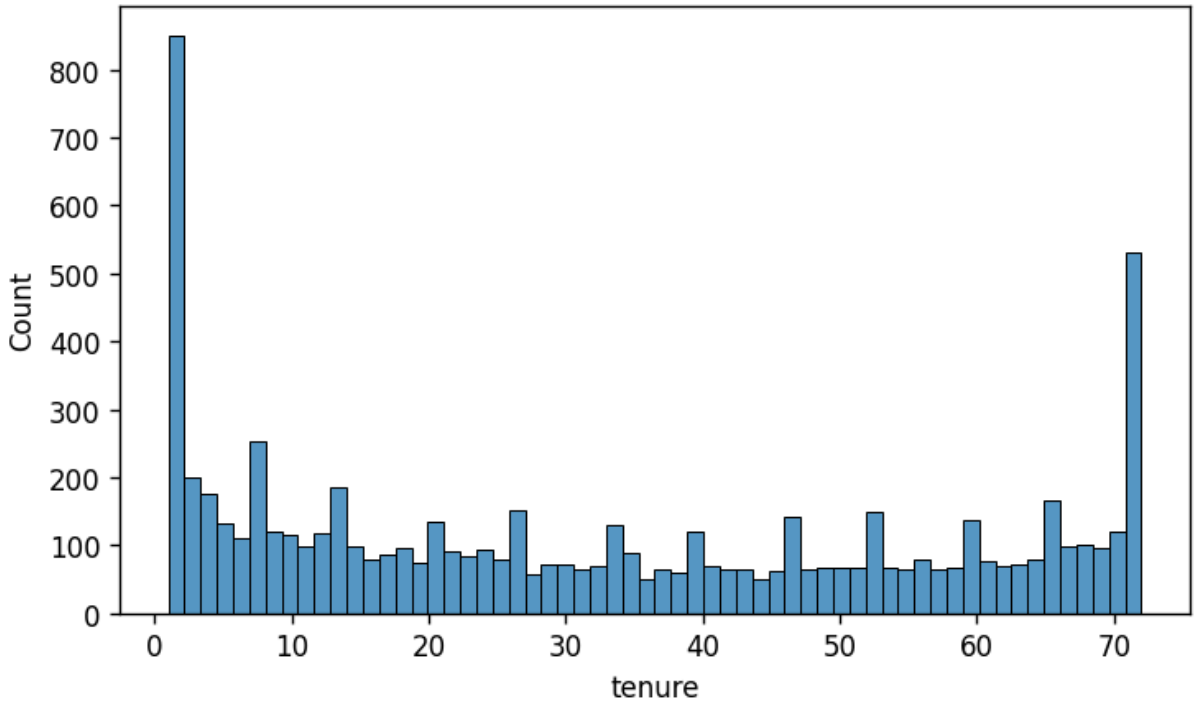
In [14]:
```
# Check the unique contract types
df['Contract'].unique()
```

Out[14]:
```
array(['Month-to-month', 'One year', 'Two year'], dtype=object)
```

Now let's create a histogram of `tenure` column which is the amount of months a customer was or has been on a customer.

In [15]:
```
# Create a histogram displaying the distribution of 'tenure' column
plt.figure(figsize=(7, 4), dpi=120)

sns.histplot(df['tenure'], bins=60)
```

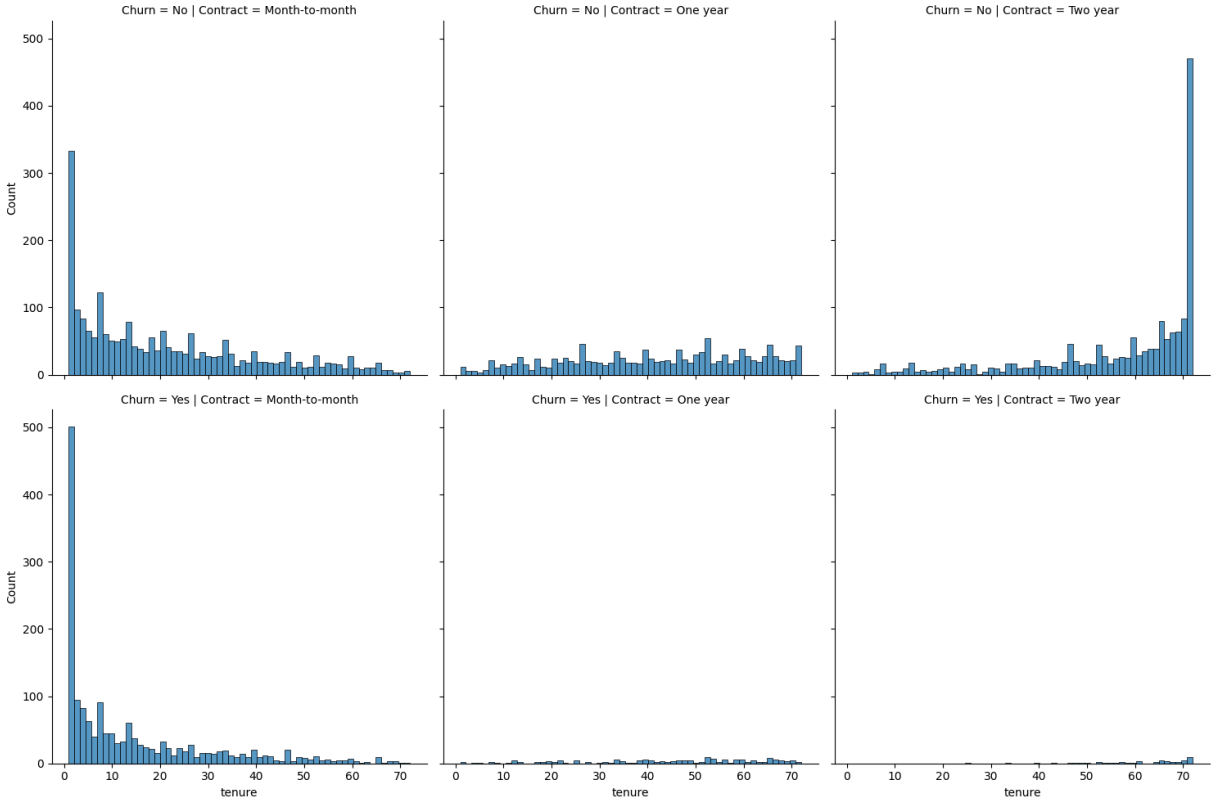
Out[15]: <Axes: xlabel='tenure', ylabel='Count'>



There are many more people in our data that are/have been service users for only 1 or 2 months! Also, there are many people who have been the users of this company for more than two years.

Now I am going to create histograms separated by two additional features, `Churn` and `Contract`.

In [16]:
```
sns.displot(df, x='tenure', row="Churn", col="Contract", bins=60);
```



Interpretation:

It is clear from the plots above that most of the churned users are on `Month-to-month` contracts and with `less than a year tenure`. Therefore, we really not need to be worry about people who have 1-year or 2-year contracts since they are very unlikely to churn.

The interesting point is that we are dealing with almost same distribution of people who are in `Month-to-month` contracts with **different churn status**! And why are there some people who have had a tenure of 10-40 months but have not signed up for 1-year or 2-year contracts?

Maybe we can approach those people with special offers and bonuses and tell them they can convert to **1-year or 2-year contracts** and lower their monthly charges! In this way they might be willing to stay and not churn!

Now let's investigate a few more variables in more details:

In [17]:
```
# Display a scatter plot of 'Total Charges' versus 'Monthly Charges'
plt.figure(figsize=(10, 5), dpi=100)

sns.scatterplot(data=df, x=df['MonthlyCharges'], y=df['TotalCharges'], hue='
```

Out[17]:
```
<Axes: xlabel='MonthlyCharges', ylabel='TotalCharges'>
```

Creating Cohorts based on Tenure

Now I will begin by treating each **unique tenure length**, 1 month, 2 month, 3 month...N months as its own cohort.

Essentially I'll be 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 X%`. We are going to have cohorts of 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 people are less likely to stop service the longer you've had it.

In [18]:
```
# Make a new dataframe and group it by `tenure` and sum up the churn values
new_df = pd.get_dummies(df[['Churn', 'tenure']])
cohort_tenure = new_df.groupby('tenure').sum('Churn_Yes')
cohort_tenure
```

Out[18]:

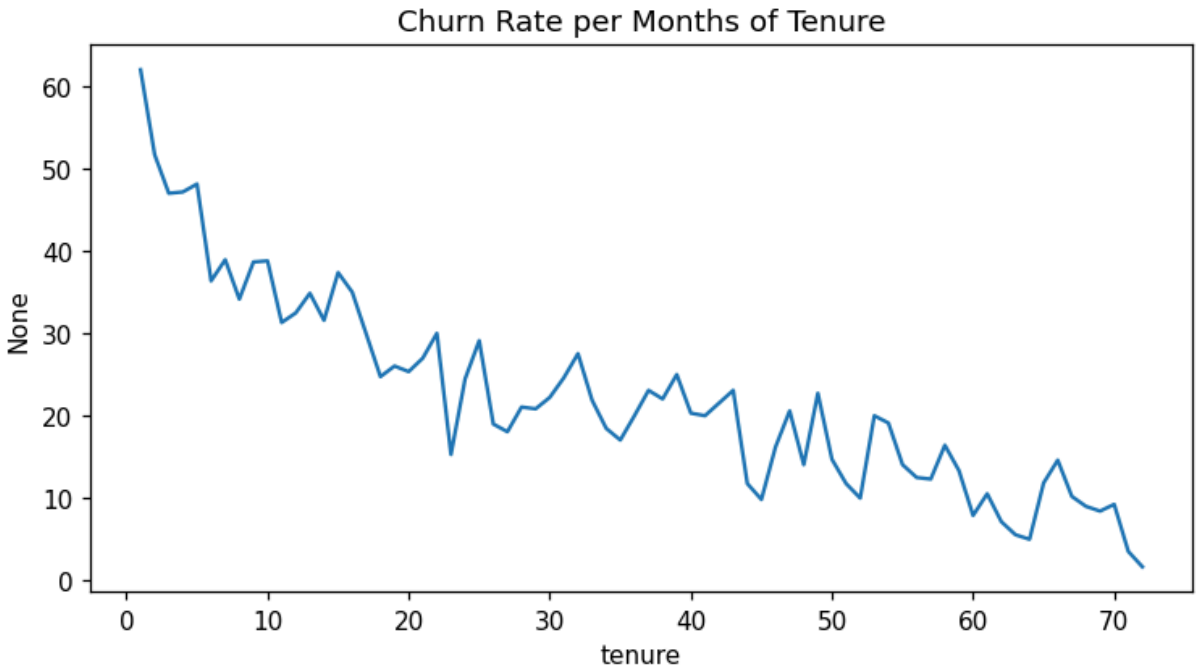
tenure	Churn_No	Churn_Yes
1	233	380
2	115	123
3	106	94
4	93	83
5	69	64
...	...	...
68	91	9
69	87	8
70	108	11
71	164	6
72	356	6

72 rows × 2 columns

In [19]:
```
# Calculate the churn rate percentage
churn_percent = cohort_tenure['Churn_Yes'] / (cohort_tenure['Churn_Yes'] + c
churn_percent
```

Out[19]:
```
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
Length: 72, dtype: float64
```

In [20]:
```
plt.figure(figsize=(8, 4), dpi=110)

sns.lineplot(data=cohort_tenure, x=cohort_tenure.index, y=churn_percent)

plt.title('Churn Rate per Months of Tenure');
```



Again, the lineplot above approves the fact that the `churn rate` **decreses** for users with `more tenure`.

Broader Cohort Groups

Now let's 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 [21]:
```
# Define a function that takes in a tenure column and returns a string
def make_tenure_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 [22]:
```
# Apply the function on the dataframe
df['Tenure Cohort'] = df['tenure'].apply(make_tenure_cohort)
df[['tenure', 'Tenure Cohort']]
```

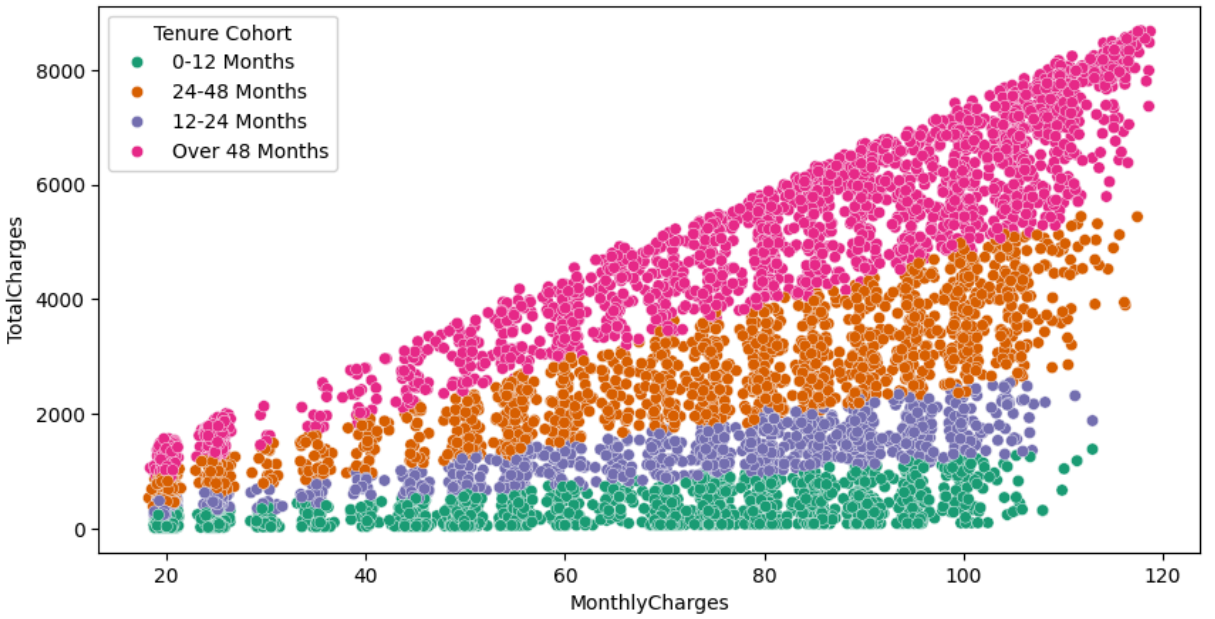
Out[22]:

	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

Now I'll create a scatterplot of `Total Charges` versus `Monthly Charts`, this time colored by `Tenure Cohort` defined in the previous task.

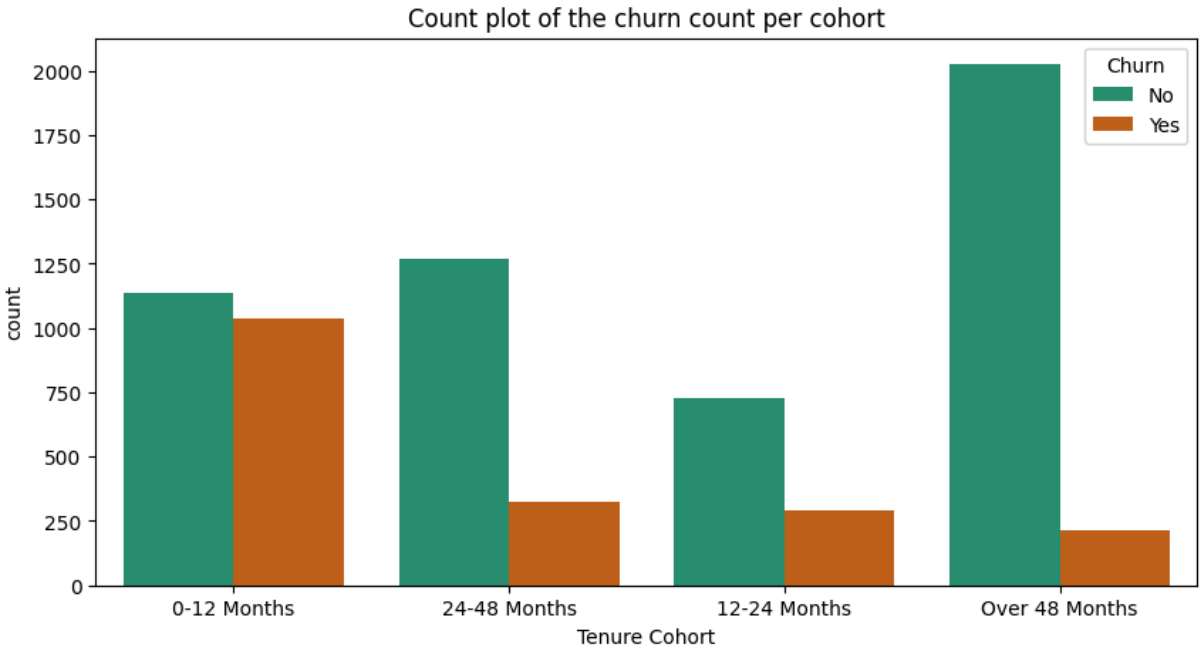
In [23]:
```
# Display a scatter plot of 'Total Charges' versus 'Monthly Charges'
plt.figure(figsize=(10, 5), dpi=100)

sns.scatterplot(data=df, x=df['MonthlyCharges'], y=df['TotalCharges'], hue='
```



It was expected to see that people who have been using the service for a longer period of time have more `Total Charges` compared to relatively new people, and the plot above validates this fact.

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

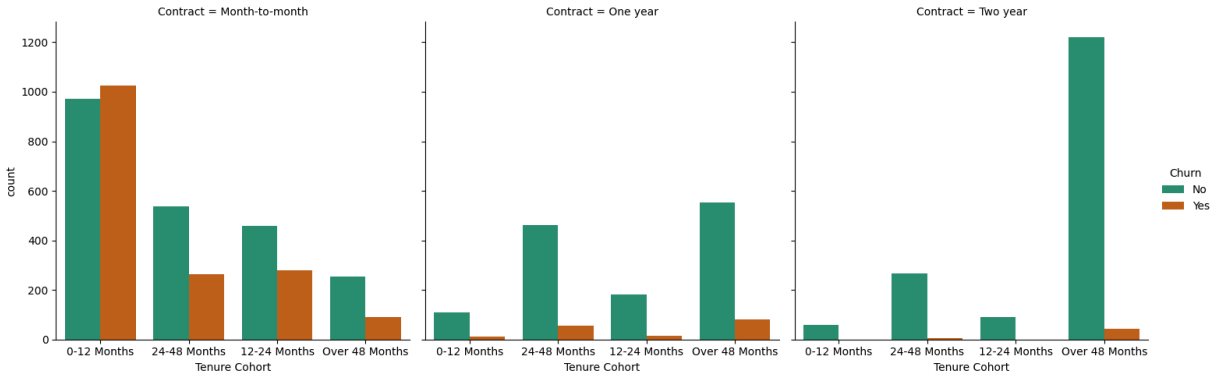
In [24]:
```
# Display a count plot showing the churn count per cohort
plt.figure(figsize=(10, 5), dpi=100)

sns.countplot(data=df, x=df['Tenure Cohort'], hue='Churn', palette='Dark2')

plt.title('Count plot of the churn count per cohort');
```



Again, we can clearly see that people in `over 48 months` cohort tend to stay and not churn compared to those who are in `0-12 months`.

In [25]:
```
# Display the counts per Tenure Cohort, separated out by contract type and c
plt.figure(figsize=(10, 5), dpi=100)

sns.catplot(data=df, x=df['Tenure Cohort'], col='Contract', hue='Churn', pal
```

```
<Figure size 1000x500 with 0 Axes>
```



Part 4: Predictive Modeling

For the final part of this project, I am going to explore 4 different tree based methods: A `Single Decision Tree`, `Random Forest`, `AdaBoost`, and `Gradient Boosting`.

Single Decision Tree

First, I am going to separate out the data into X features and Y label. Note that the initial dataframe's most features are categorical, so we need to get them to numerical. Also, the `customerID` column is not an indicator so I'll drop it. I also drop the `gender` column for ethical reasons since I don't want the model to predict based off the gender of a person.

In [29]:
```
# Seprate the features and the label
X = df.drop(['customerID', 'gender', 'Churn'], axis=1)
y = df['Churn']

# Convert categorical features to numerics
X = pd.get_dummies(X, drop_first=True)
X.head()
```

Out[29]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	Partner_Yes	Dependents_Yes
0	0	1	29.85	29.85	True	False
1	0	34	56.95	1889.50	False	False
2	0	2	53.85	108.15	False	False
3	0	45	42.30	1840.75	False	False
4	0	2	70.70	151.65	False	False

5 rows × 32 columns

Now let's perform a train test split, holding out 10% of the data for testing.

In [31]:
```
# Train-test split
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, ra
```

In [34]:
```
print('X_train size: ', X_train.shape)
print('y_train size: ', y_train.shape)
print('X_test size: ', X_test.shape)
print('y_test size: ', y_test.shape)
```

```
X_train size:  (6328, 32)
y_train size:  (6328,)
X_test size:  (704, 32)
y_test size:  (704,)
```

Single Decision Tree Model:

The steps I am going to perform are as follows:

1. Train a single decision tree model with a 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 and plot them on a barplot.

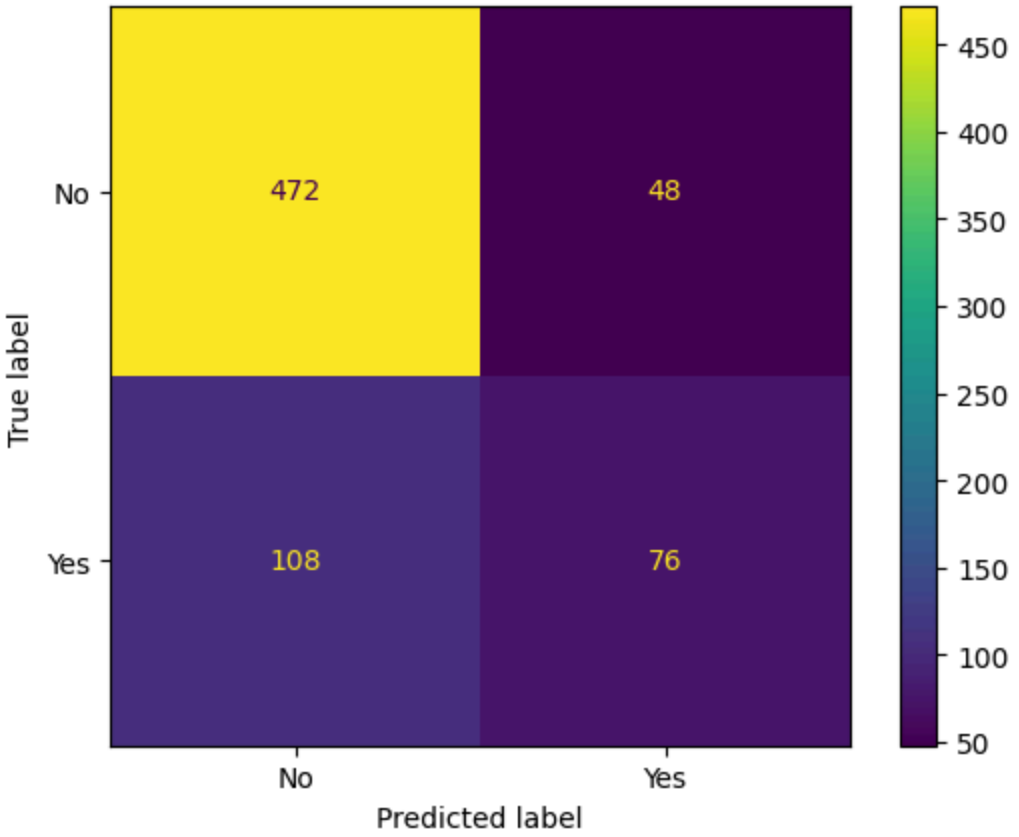
In [35]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, Confusi
from sklearn.model_selection import GridSearchCV

# Instanciate a based decision tree object
tree_model = DecisionTreeClassifier()

# Create the param_grid dictionary for performing the grid search on hyper-p
param_grid = {
    'max_depth': [None, 10, 25, 35, 45],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'max_features': ['auto', 'log2']
}

# Instanciate the grid search object
tree_grid_model = GridSearchCV(estimator=tree_model, param_grid=param_grid)

# Fit the model into training data
tree_grid_model.fit(X_train, y_train)
```

In [56]:
```
# Print the best parameters and the best score
print("Best parameters found: ", tree_grid_model.best_params_)

# Get the predictions
pred_tree = tree_grid_model.predict(X_test)

print('The classification Report: ')
print()
print(classification_report(y_test, pred_tree))
```

```
Best parameters found:  {'max_depth': 45, 'max_features': 'log2', 'min_sampl
es_leaf': 10, 'min_samples_split': 10}
The classification Report:

              precision    recall  f1-score   support

          No       0.81      0.91      0.86       520
         Yes       0.61      0.41      0.49       184

    accuracy                           0.78       704
   macro avg       0.71      0.66      0.68       704
weighted avg       0.76      0.78      0.76       704
```

In [117…
```
print('The Confusion Matrix: ')
print()
cm = confusion_matrix(y_test, pred_tree)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=tree_grid_model.c
```

```
The Confusion Matrix:
```



So for my first model which was a **grid searched single decision tree**, I got an overal accuracy of **0.78** on the hold-out set. I am going to create a dataframe of test results and append each models' accuracy to it, so we would be able to see the results next to each other and compare them easily.

In [70]:
```
# Create the test result Pandas dataframe
from sklearn.metrics import precision_score, f1_score, accuracy_score

# Initialize the DataFrame
results_df = pd.DataFrame(columns=['Model', 'Precision', 'F1 Score', 'Accura

def evaluate_and_append_results(model_name, fitted_model, X_test, y_test, re
    """
    Evaluates the given model on the test data and appends the results to th

    Parameters:
    - model_name: str, name of the model
    - fitted_model: the fitted model object
    - X_test: array-like, test features
    - y_test: array-like, test labels
    - results_df: DataFrame, the DataFrame to append the results to
```

```
    Returns:
    - results_df: DataFrame, the updated DataFrame with the new results
    """
    # Make predictions
    y_pred = fitted_model.predict(X_test)

    # Compute metrics
    precision = precision_score(y_test, y_pred, pos_label='Yes')
    f1 = f1_score(y_test, y_pred, pos_label='Yes')
    accuracy = accuracy_score(y_test, y_pred)

    # Create a DataFrame with the new results
    new_results = pd.DataFrame({
        'Model': [model_name],
        'Precision': [precision],
        'F1 Score': [f1],
        'Accuracy': [accuracy]
    })

    # Concatenate the new results to the existing DataFrame
    results_df = pd.concat([results_df, new_results], ignore_index=True)

    return results_df
```

In [81]:
```
# Check the result dataframe
results_df = evaluate_and_append_results('Decision Tree', tree_grid_model, X
results_df
```

Out[81]:

	Model	Precision	F1 Score	Accuracy
0	Decision Tree	0.612903	0.493506	0.778409

In [84]:
```
# Check the feature importances
feature_importance = tree_grid_model.best_estimator_.feature_importances_
feature_importance
```

Out[84]:
```
array([6.52483971e-03, 3.97003571e-01, 5.60442731e-02, 4.73684146e-02,
       2.70738179e-03, 6.07931701e-03, 1.16840114e-03, 2.82711538e-03,
       1.55488646e-02, 2.40545104e-01, 0.00000000e+00, 2.00575059e-02,
       5.69484882e-03, 0.00000000e+00, 6.73807552e-03, 0.00000000e+00,
       2.88913349e-03, 1.19212911e-02, 8.94520523e-03, 0.00000000e+00,
       1.44809403e-02, 4.14402157e-02, 1.34709594e-02, 8.07961750e-03,
       4.19881529e-02, 6.24123387e-03, 2.68730652e-03, 2.65938029e-02,
       6.52654204e-03, 6.90116175e-06, 3.01524447e-03, 3.40574056e-03])
```

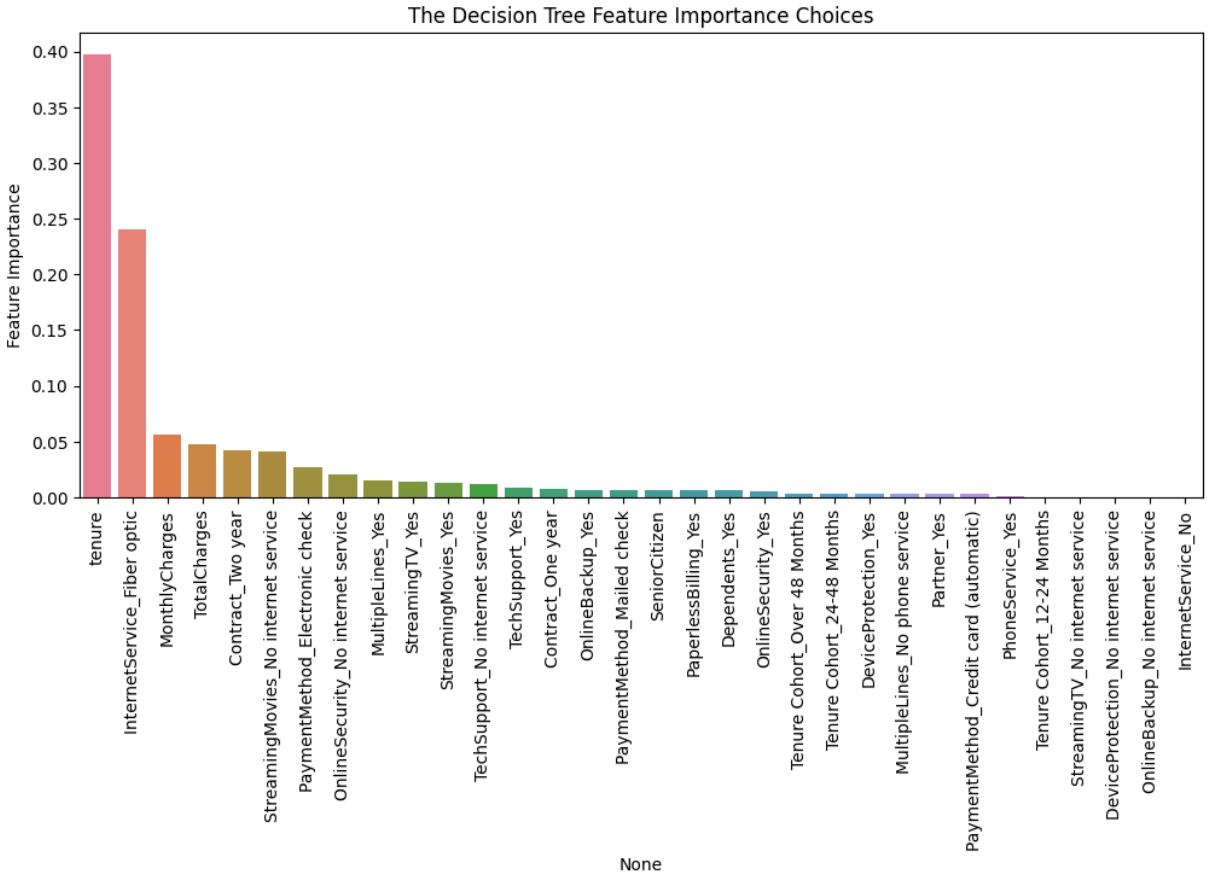
In [101…
```
feature_importance_df = pd.DataFrame(data=feature_importance,
                                     index=X_train.columns,
                                     columns=['Feature Importance'])

sorted_feature_importance = feature_importance_df.sort_values('Feature Impor
sorted_feature_importance.head()
```

Out [101…

	Feature Importance
tenure	0.397004
InternetService_Fiber optic	0.240545
MonthlyCharges	0.056044
TotalCharges	0.047368
Contract_Two year	0.041988

In [100…

```
# Display the feature importance of the decision tree on a barplot
plt.figure(figsize=(12, 5))

sns.barplot(x=sorted_feature_importance.index, y=sorted_feature_importance['

plt.title('The Decision Tree Feature Importance Choices')
plt.xticks(rotation=90);
```



So based on our single decision tree model, the `tenure` and `InternetService_Fiber optic` are the main features for spliting the data to make this predictive model. Certainly, we can improve out model metrics by making **ensemble models** such as a random forest, and even possibly improve them by **boosting methodologies** such as `Adaboosting` and `Gradient boosting`.

Let's investigate them then:

Random Forest Model:

The steps I am going to perform are as follows:

1. Train a random model with a grid search for optimal hyperparameters.
2. Evaluate performance metrics from the random forest model, including classification report and plotting a confusion matrix.
3. Calculate feature importances from the decision tree and plot them on a barplot.

In [147…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.exceptions import FitFailedWarning
import warnings

rfc = RandomForestClassifier()

# Define the param_grid dictionary
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 25, 35, 45],
    'min_samples_split': [2, 10, 20],
    'min_samples_leaf': [1, 5, 10],
    'max_features': ['auto', 'log2']
}

# Instanciate the grid search object
rfc_grid_model = GridSearchCV(estimator=rfc, param_grid=param_grid)

# Suppress FitFailedWarning
with warnings.catch_warnings():
    warnings.simplefilter('ignore', FitFailedWarning)

    # Fit the model into training data
    rfc_grid_model.fit(X_train, y_train)
```

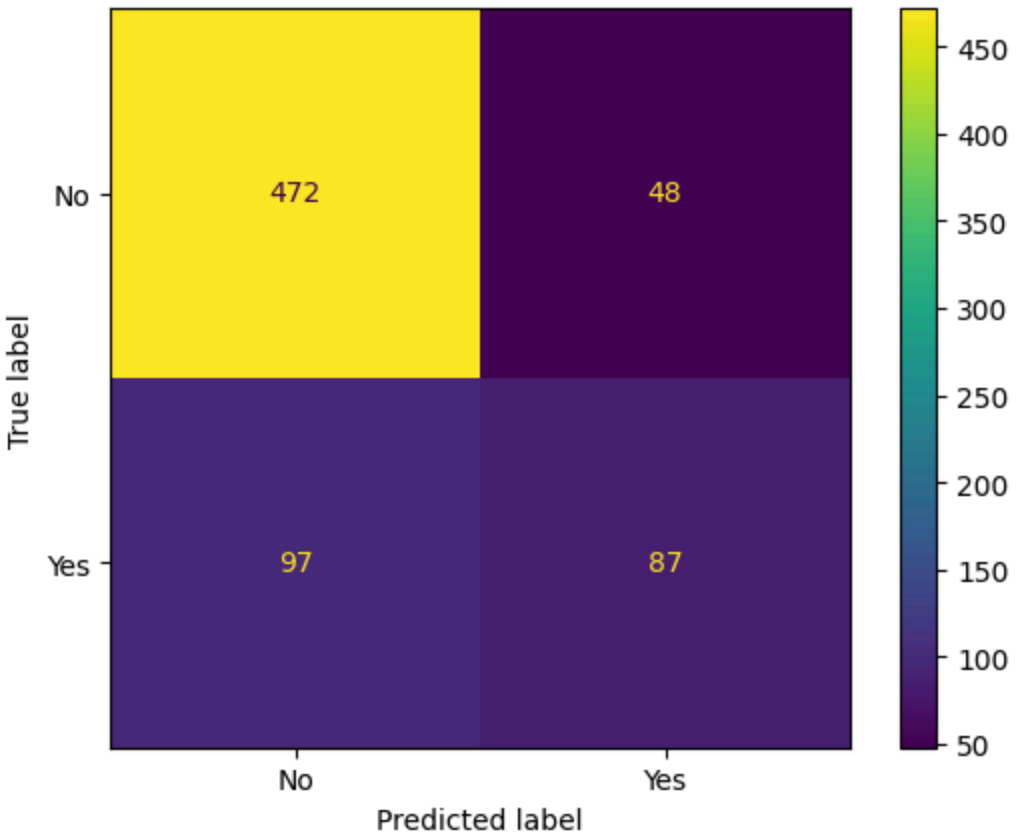
In [115…
```
# Print the best parameters and the best score
print("Best parameters found: ", rfc_grid_model.best_params_)

# Get the predictions
pred_rfc = rfc_grid_model.predict(X_test)

print('The classification Report: ')
print()
print(classification_report(y_test, pred_rfc))
```

```
Best parameters found:  {'max_depth': 25, 'max_features': 'log2', 'min_sampl
es_leaf': 5, 'min_samples_split': 10, 'n_estimators': 300}
The classification Report:
```

	precision	recall	f1-score	support
No	0.83	0.91	0.87	520
Yes	0.64	0.47	0.55	184
accuracy			0.79	704
macro avg	0.74	0.69	0.71	704
weighted avg	0.78	0.79	0.78	704

In [118…
```
print('The Confusion Matrix: ')
print()
cm = confusion_matrix(y_test, pred_rfc)
ConfusionMatrixDisplay(confusion_matrix=cm,
                       display_labels=rfc_grid_model.classes_).plot();
```

```
The Confusion Matrix:
```



In [119…
```
# Add the random forest model metrics to the result dataframe
results_df = evaluate_and_append_results('Random Forest', rfc_grid_model, X_
results_df
```

Out[119…

	Model	Precision	F1 Score	Accuracy
0	Decision Tree	0.612903	0.493506	0.778409
1	Random Forest	0.644444	0.545455	0.794034

So the improvement of all evaluation metrics are clear. Now let's see if using `Adaboosting` and `Gradient Boosting` methodologies can further improve the random forest results.

Boosted Methods Based on Tree Models:

Finally, I am going to build two popular boosting methods, `Adaboost Classifier` and `GradientBoosting Classifier`. Note that the estimator that I pass to this meta learners will be the default sklearn model which is a **DecisionTreeClassifier estimator**. However, there is this option to choose any other models as well but boosting methods usually tend to work better with tree-based models.

I am also going to perform grid search for finding the best hyper-parameter choices:

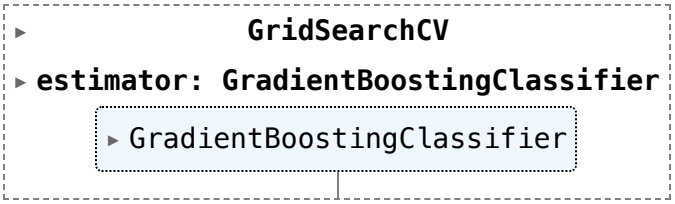
In [162…

```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier

# Instanciate the base models
ada_model = AdaBoostClassifier()
gb_model = GradientBoostingClassifier()

# Define the param_grid dictionary for Adabooster and Gradient Boddting
gb_param_grid = {'learning_rate': [0.1, 0.05, 0.2],
             'max_depth': [3, 4, 5],
             'n_estimators':[100, 150, 200]}

ada_param_grid = {'learning_rate': [0.1, 0.05, 0.2],
             'n_estimators':[100, 150, 200]}
```

In [163…

```
# Instanciate the grid search model with the defined gradient boosting based
gb_grid_model = GridSearchCV(gb_model, param_grid=gb_param_grid)

# Fit the model with the training data
gb_grid_model.fit(X_train, y_train)
```

Out[163…

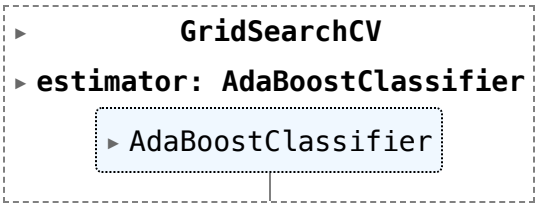


In [164…

```
# Instanciate the grid search model with the defined Adabooster based learne
ada_grid_model = GridSearchCV(ada_model, param_grid=ada_param_grid)
```

```
# Fit the model with the training data
ada_grid_model.fit(X_train, y_train)
```

Out[164…



In [167…

```
ada_pred = ada_grid_model.predict(X_test)
gb_pred = gb_grid_model.predict(X_test)
```

In [168…

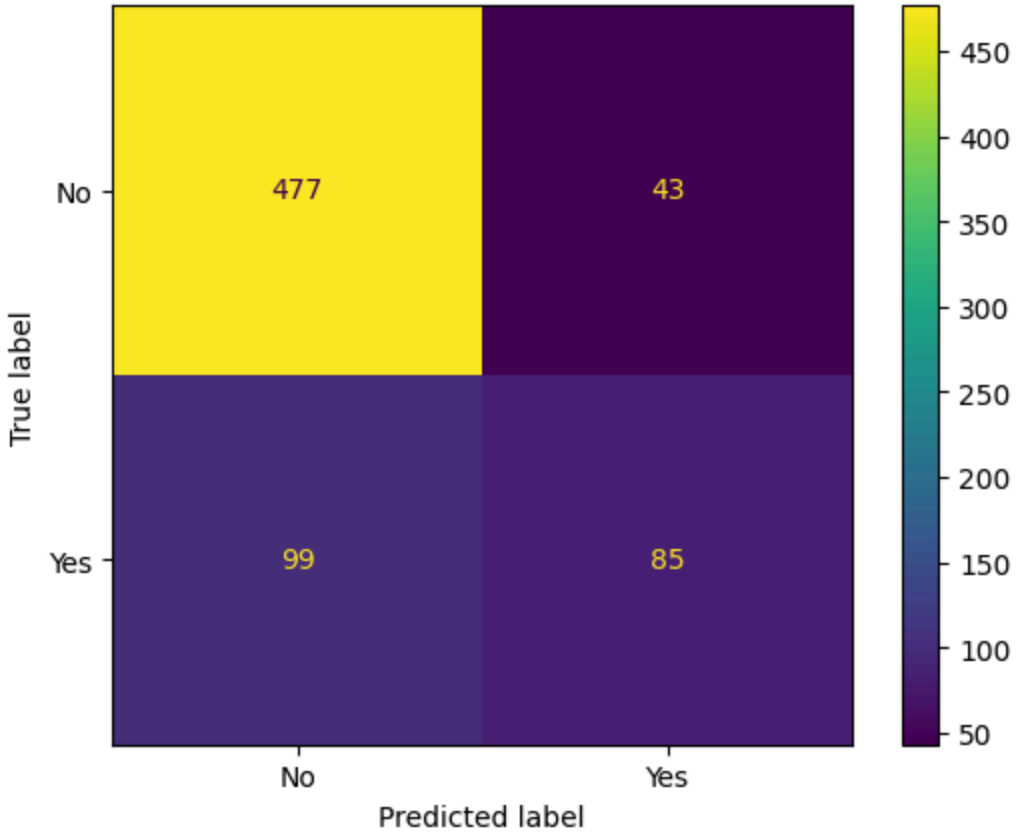
```
print('The classification Report For Adabooster Model: ')
print()
print(classification_report(y_test, ada_pred))
```

```
The classification Report For Adabooster Model:

              precision    recall  f1-score   support

          No       0.83      0.92      0.87       520
         Yes       0.66      0.46      0.54       184

    accuracy                           0.80       704
   macro avg       0.75      0.69      0.71       704
weighted avg       0.79      0.80      0.79       704
```

In [169…

```
print('The Confusion Matrix For Adabooster Model: ')
print()
cm = confusion_matrix(y_test, ada_pred)
ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=gb_grid_model.cla
```

```
The Confusion Matrix For Adabooster Model:
```

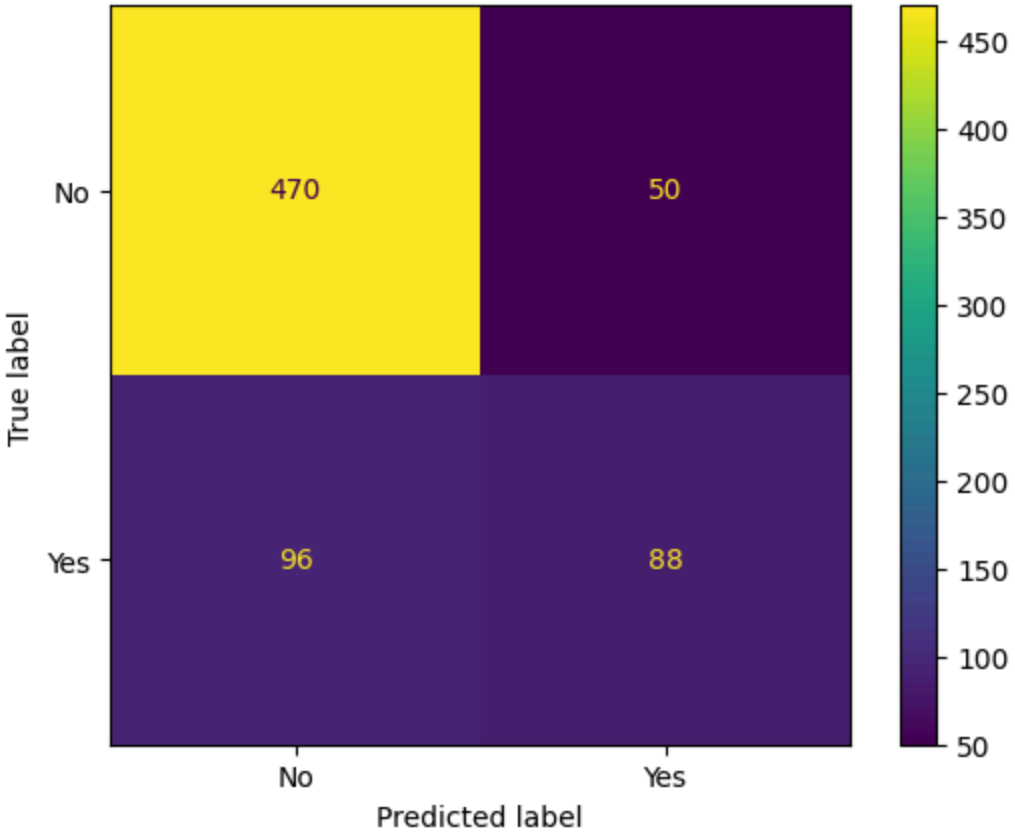


In [170…
```
print('The classification Report For Gradient Boosting Model: ')
print()
print(classification_report(y_test, gb_pred))
```

```
The classification Report For Gradient Boosting Model:

              precision    recall  f1-score   support

          No       0.83      0.90      0.87       520
         Yes       0.64      0.48      0.55       184

    accuracy                           0.79       704
   macro avg       0.73      0.69      0.71       704
weighted avg       0.78      0.79      0.78       704
```

In [171…
```
print('The Confusion Matrix For Gradient Boosting Model: ')
print()
cm = confusion_matrix(y_test, gb_pred)
ConfusionMatrixDisplay(confusion_matrix=cm,
                       display_labels=gb_grid_model.classes_).plot();
```

```
The Confusion Matrix For Gradient Boosting Model:
```



In [174…
```
# Add the Adabooster and Gradient Boosting models metrics to the result data
results_df = evaluate_and_append_results('Adabooster', ada_grid_model, X_tes
results_df = evaluate_and_append_results('Gradient Boosting', gb_grid_model,
results_df
```

Out[174…

	Model	Precision	F1 Score	Accuracy
0	Decision Tree	0.612903	0.493506	0.778409
1	Random Forest	0.644444	0.545455	0.794034
2	Adabooster	0.664062	0.544872	0.798295
3	Gradient Boosting	0.637681	0.546584	0.792614

Conclusion of the Project

The most interesting aspect of this project is that the **grid searched random forest**, **Adabooster**, and **gradien boosting methods** all performed very close to each other although the precision of `Adabooster` model was higher than those two.

If it was a real project for a company that I work for, my next steps for improving the models to choose the best one would be as follows:

- Allocate more time on hyper-parameter searching via grid search.

- Probably performing re-sampling of the **label=No_churn** in order to make the target variable more balanced.

- I would also try to use **non-tree models** and pass them to the boosting methods. It would be interesting to see how other algorithms such as `SVM` or `logestic regression` would work with boosting methods.

Note:

In this project, the main evaluation metric that we need to keep track of (and in fact, try to decrease) is the **`false negetives`**. Meaning that we need to **reduce the number of `false predictions` for people not churning, while they are actually about to churn!**

The **`false positives`** are not really that big of a deal compared to `false negatives` because in worst case scenario, we will send them some bonus offers and it will even make them more willing to stay although it might end up make the company lose some profits by offering unnecassary offers to users who did not want to churn.