In [1]:

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from matplotlib.ticker import FormatStrFormatter

'''
Goal of the project:
Creating predictive models to see whether a customer will churn(terminate contra
given a dataset of customer information
'''
```

In [2]:

```python
df = pd.read_csv("/Users/pacosun/Downloads/Telco-Customer-Churn.csv")
```

In [3]:

```python
df.head()

'''
Each customer comes with 21 different features which will help us
create predictive model later in the project

Will not be using every single one; some trash features will be dropped
'''
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
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

In [4]:

```python
# Quick overview, confirming there's no missing values (NaN) using .isna() to co

df.isna().sum()
```

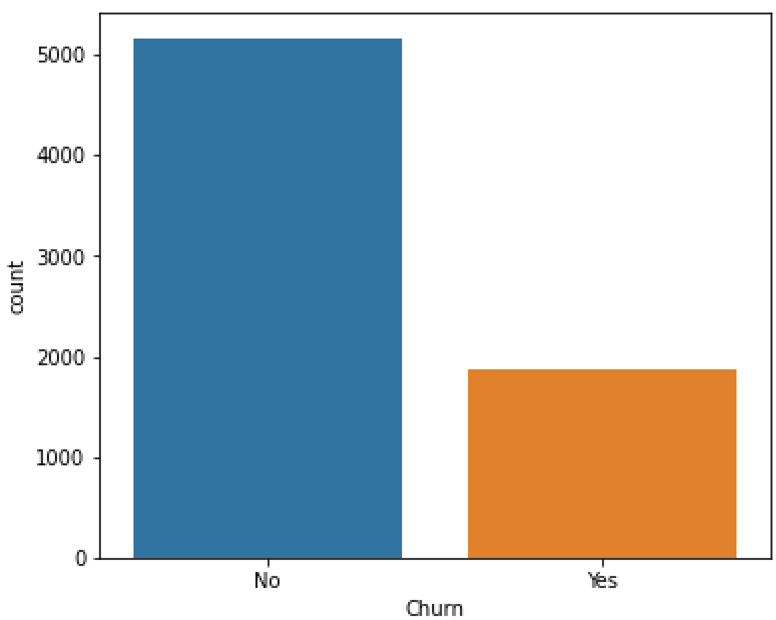
Out[4]:

```
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 [5]:

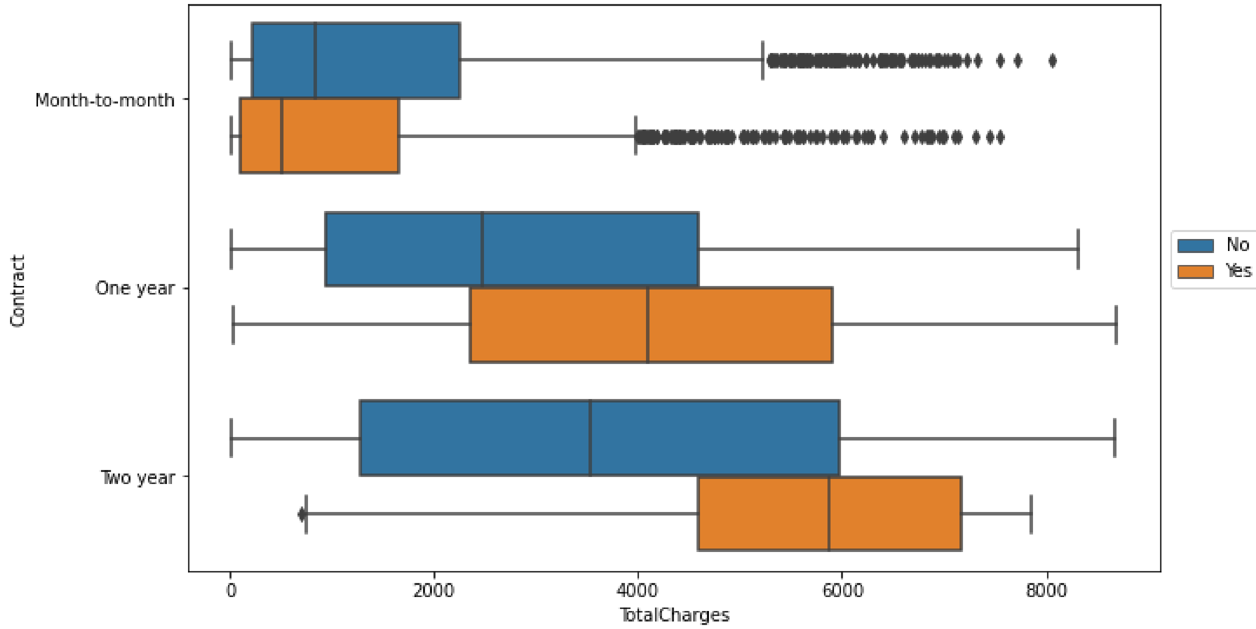
```
# Displaying the balance with CountPlot

plt.figure(figsize = (6,5))
sns.countplot(data = df, x = "Churn")
plt.show()

'''
Overall, the majority did not churn
However more specific analysis needed
'''
```



In [6]:

```
# Creating BoxPlots for distribution of TotalCharges per contract type
# Adding hue coloring based on Churn class

plt.figure(figsize = (10,6))
sns.boxplot(data = df, x = "TotalCharges", y = "Contract",
            hue = "Churn")
plt.legend(loc = (1.01,.5))
plt.show()

'''
Month-to-month is hard to predict because customers come expecting to cancel the
```

```
Two-year does the telling: those who did churn were charged higher (also higher
'''
```



In [7]:

```
# Selecting features with less unique instances
# customerID, tenure will not be selected to convert to dummy values due to no o
# -> highly unique features

corr = pd.get_dummies(df[['gender', 'SeniorCitizen', 'Partner',
                          'Dependents','PhoneService', 'MultipleLines',
                          'InternetService','OnlineSecurity', 'OnlineBackup',
                          'DeviceProtection', 'TechSupport','StreamingTV',
                          'StreamingMovies', 'Contract', 'PaperlessBilling',
                          'PaymentMethod','Churn']]).corr()
```

In [8]:

```
# Creating correlation for features selected above

corr["Churn_Yes"].sort_values().iloc[1:-1].head()

'''
Two-year has a correlation of -.301, meaning it's not likely for someone who's o
contract to churn
'''
```

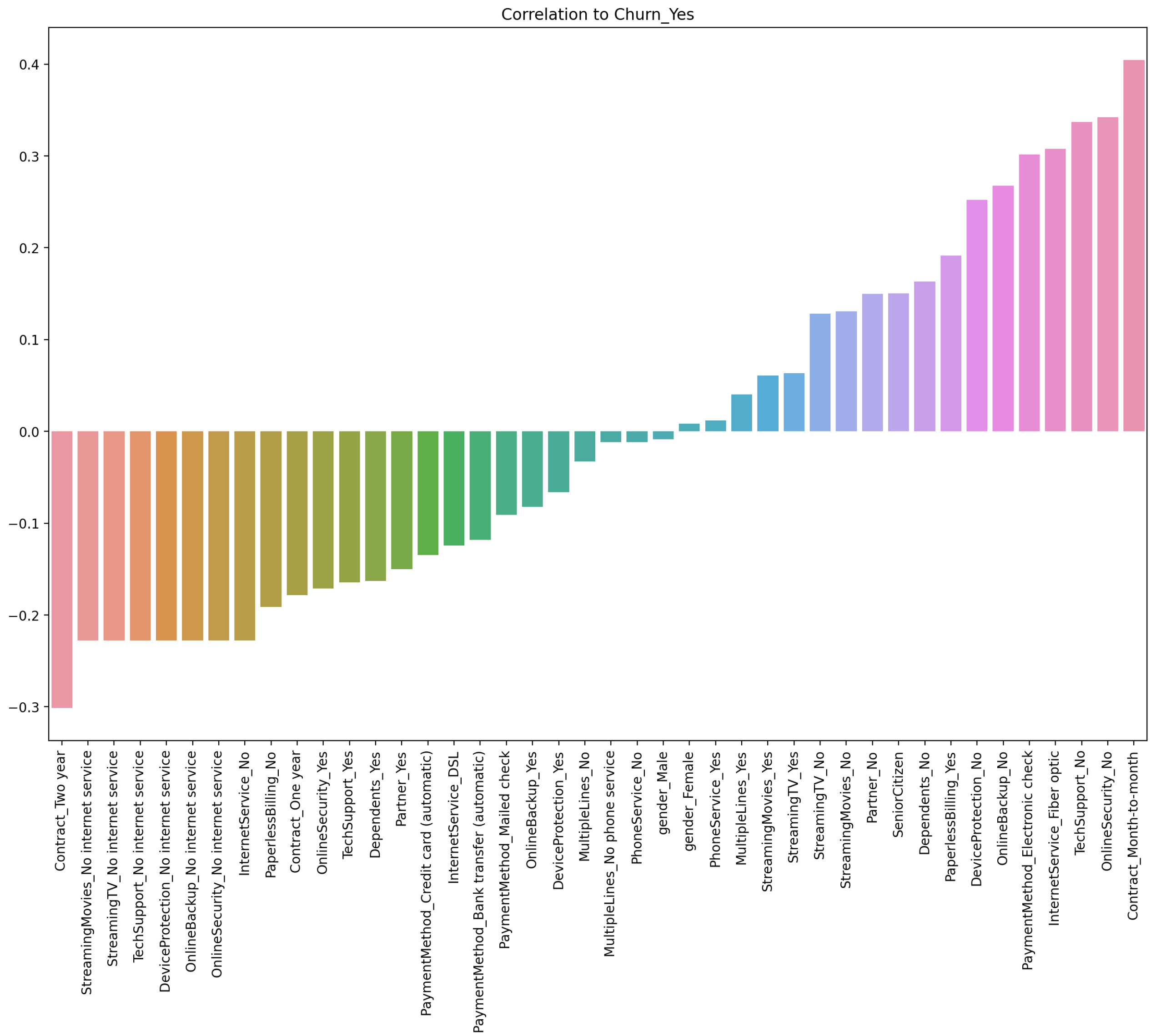
Out[8]:

```
Contract_Two year                        -0.301552
StreamingMovies_No internet service      -0.227578
StreamingTV_No internet service          -0.227578
TechSupport_No internet service          -0.227578
DeviceProtection_No internet service     -0.227578
Name: Churn_Yes, dtype: float64
```

In [9]:

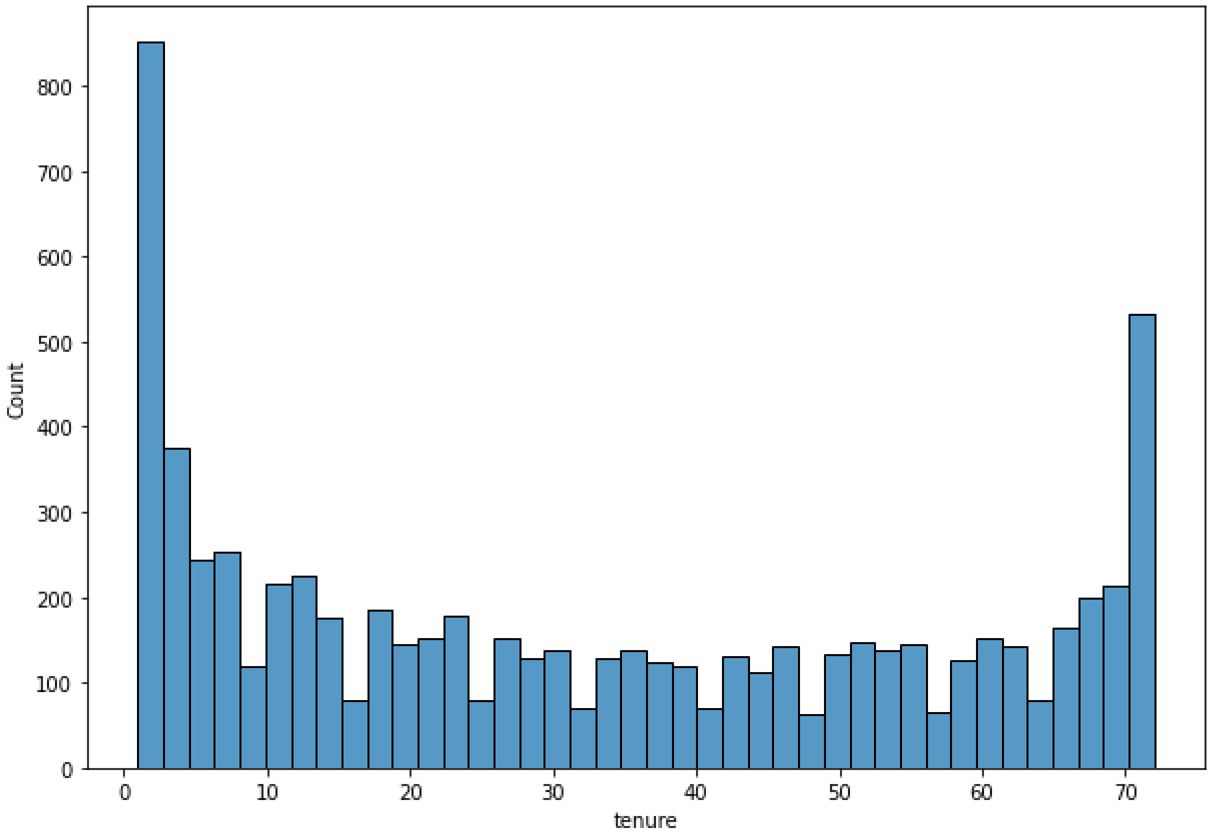
```
# Visualization of correlations

plt.figure(figsize = (15,10), dpi = 200)
sns.barplot(x = corr["Churn_Yes"].sort_values().iloc[1:-1].index,
            y = corr["Churn_Yes"].sort_values().iloc[1:-1].values)
plt.title("Correlation to Churn_Yes")
```

```
plt.xticks(rotation = 90)
plt.show()

'''
Again, it's unlikely for one to churn if has a two-year contract
-> Two-year contract on the far left with a negative correlation

On the contrary one would be a lot more likely to churn if on a monthly contract
-> Month-to-month on the far right with a positive correlation
'''
```



In [10]:

```
# Distributing based on tenure -> (Amount of time one stays as a customer)
# Visualizing the distribution of tenure using histogram

plt.figure(figsize = (10,7))
sns.histplot(data = df, x = "tenure", bins = 40)
plt.show()
```



In [11]:

```
# Grouping based on Tenure length (1 Month, 2 Months, ... n Months)
# Treaing each length as a single group/cohort
# And then calculate the "Churn Rate" according to tenure

noChurn = df.groupby(["Churn","tenure"]).count().transpose()["No"]
yesChurn = df.groupby(["Churn","tenure"]).count().transpose()["Yes"]
```

In [12]:

```
churnRate = yesChurn / (noChurn + yesChurn) * 100
```

In [13]:

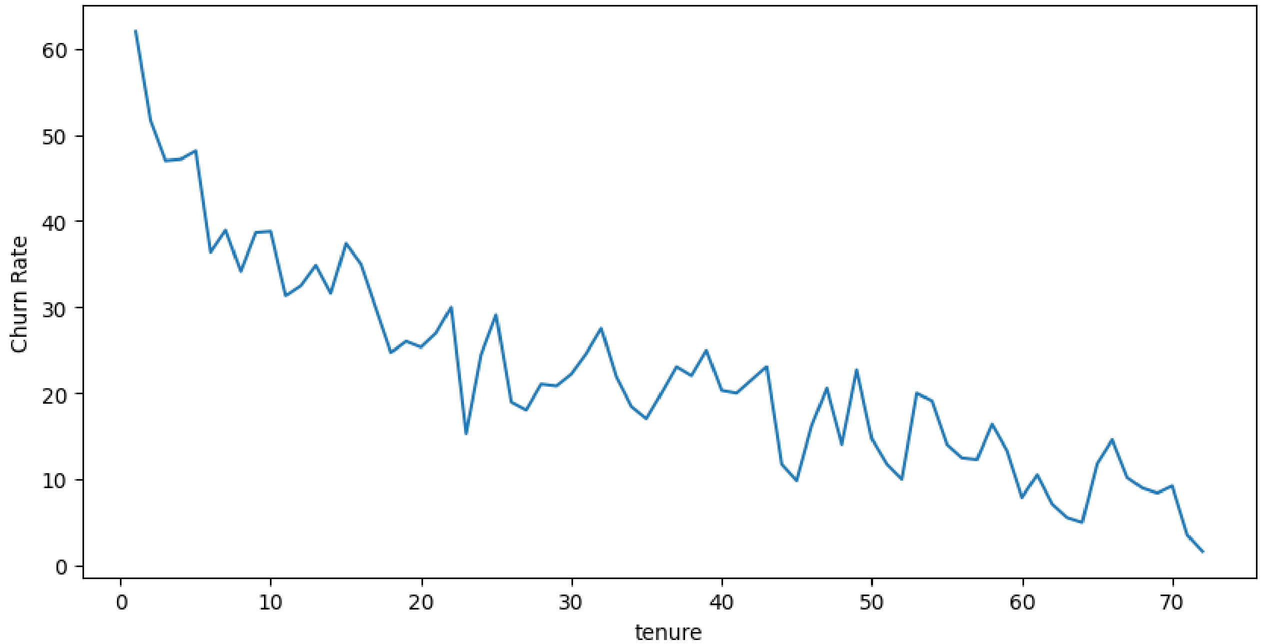
```
churnRate.transpose()["customerID"]

'''
List below shows there's a negative correlation: LONGER the tenure, LESS likely
-> e.g. One-month tenure would churn 61.99% percent of the time
'''
```

Out[13]:

```
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 [14]:

```
# Visualizing the correlation

plt.figure(figsize = (10,5),dpi = 100)
churnRate.iloc[0].plot()
plt.ylabel("Churn Rate")
plt.show()
```



In [15]:

```
# Broadening the cohorts
# Separating into 4 groups
# 0 - 12, 12 - 24, 24 - 48, Over 48

def cohorts(n):
    if n < 13:
        return "0 to 12 Months"
    elif n < 25:
        return "12 to 24 Months"
    elif n < 49:
        return "24 to 48 Months"
    else:
        return "Over 48 Months"
```

In [16]:

```
# Adding a new column -> Tenure Group

df["Tenure Group"] = df["tenure"].apply(cohorts)
```

In [17]:

```
df.head(11)[["tenure","Tenure Group"]]
```

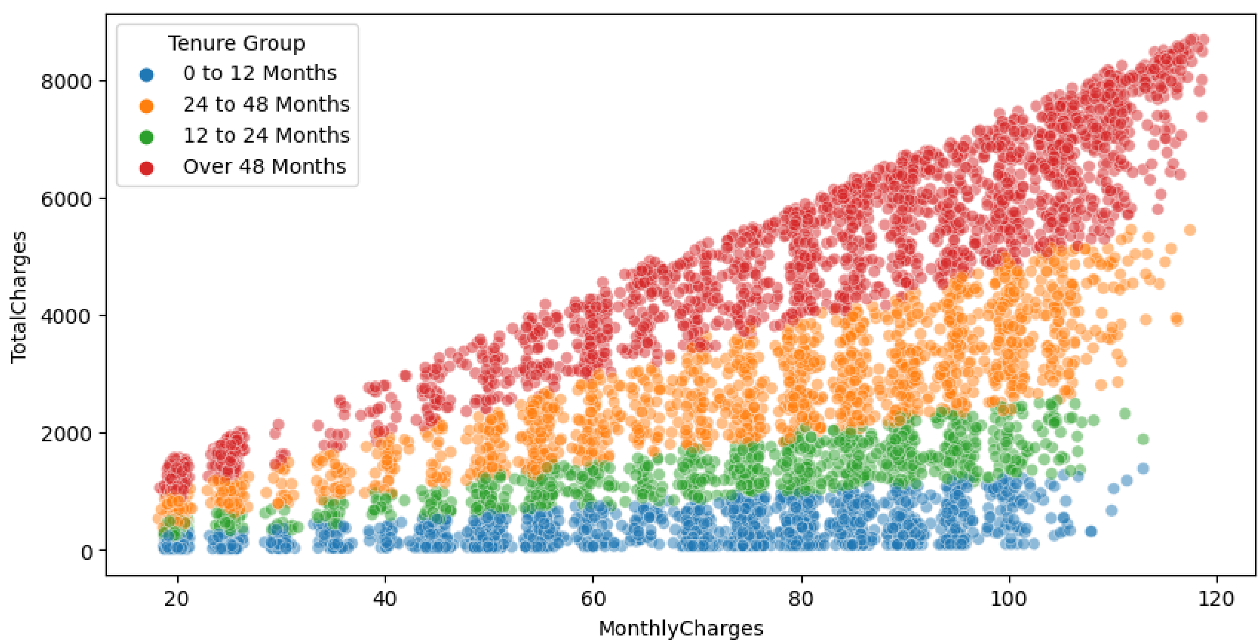
Out[17]:

	tenure	Tenure Group
0	1	0 to 12 Months
1	34	24 to 48 Months
2	2	0 to 12 Months

	tenure	Tenure Group
3	45	24 to 48 Months
4	2	0 to 12 Months
5	8	0 to 12 Months
6	22	12 to 24 Months
7	10	0 to 12 Months
8	28	24 to 48 Months
9	62	Over 48 Months
10	13	12 to 24 Months

In [18]:

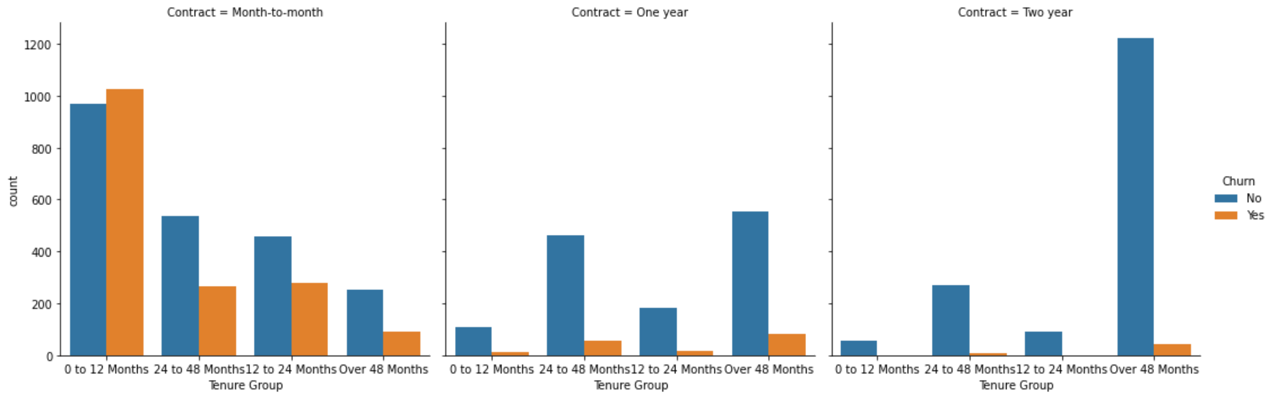
```
# Visualizing the relationship between Monthly and TotalCharges
# Colored by Tenure Group

plt.figure(figsize = (10,5), dpi = 100)
sns.scatterplot(data = df, x = "MonthlyCharges",
                y = "TotalCharges", hue = "Tenure Group", alpha = .5)
plt.show()
```



In [19]:

```
# Creating a grid of CountPlots
# Showing counts per Tenure Group, separated by contract type
# Colored by Churn

plt.figure(figsize = (10,8), dpi = 100)
sns.catplot(data = df, x = "Tenure Group",
            hue = "Churn", col = "Contract", kind = "count")
plt.show()

'''
Here we are seeing a significant drop in churn rate as the contract length(type)
'''
```

```
<Figure size 1000x800 with 0 Axes>
```



In [20]:

```
# Now deploying tree based methods for predictive modeling
# DecisionTree, RandomForest, BoostedTrees(Ada & Gradient)
```

In [21]:

```
# Separating data into X and y label
# Creating dummy values

X = df.drop(["Churn", "customerID"], axis = 1)
X = pd.get_dummies(X, drop_first = True)
y = df["Churn"]
```

In [22]:

```
# Performing a train-test split
# 10% for testing, randomState = 101

from sklearn.model_selection import train_test_split
```

In [23]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = .1,
                                                    random_state = 101)
```

In [24]:

```
# 1. Train DecisionTree model

# 2. Evaluate performance metrics
# -> Classification report, confusion matrix

# 3. Calculate feature importance
# 4. Plot the tree

from sklearn.tree import DecisionTreeClassifier
```

In [25]:

```
tree = DecisionTreeClassifier(max_depth = 6)
```

In [26]:

```
tree.fit(X_train, y_train)
```

Out[26]:

```
DecisionTreeClassifier(max_depth=6)
```

In [27]:

```python
prediction = tree.predict(X_test)
```

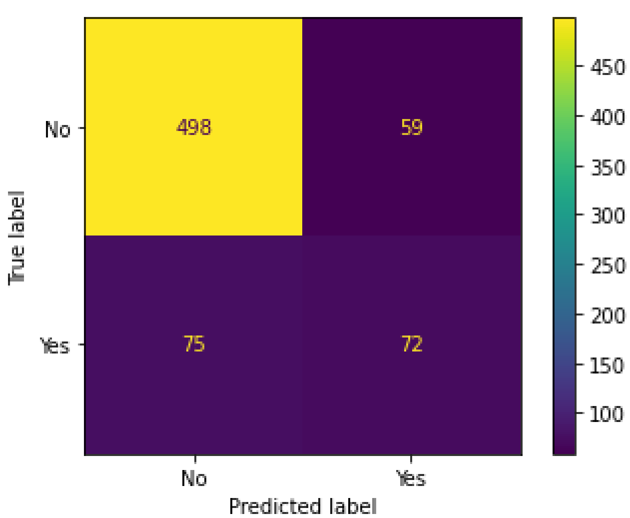
In [28]:
```python
from sklearn.metrics import accuracy_score, plot_confusion_matrix, classificatio
```

In [29]:
```python
# Comparison between true values and predictions

print(classification_report(y_test, prediction))

'''
The model performed better when it comes to predicting those who did NOT churn c
-> (.87 vs .55)
'''
```

```
              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      0.81       704
```

In [30]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [31]:
```python
plot_confusion_matrix(tree,X_test,y_test)
plt.show()

'''
Most important to note & reduce:
The model predicted that 75 people were not going to churn, whereas in reality t
'''
```

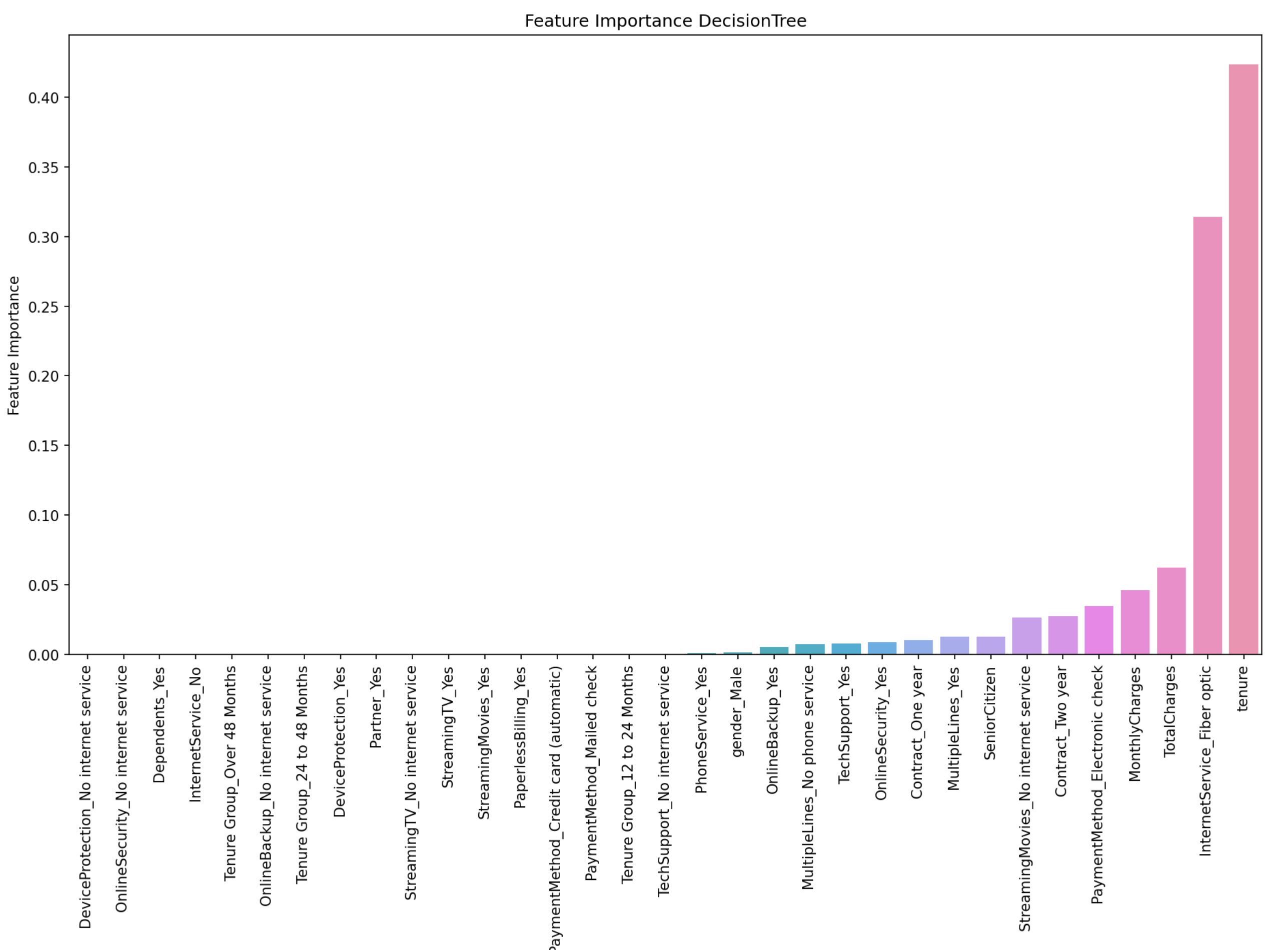


In [32]:
```python
# Calculating feature importances

features = pd.DataFrame(data = tree.feature_importances_,
```

```
                                index = X.columns,
                                columns = ["Feature Importance"]).sort_values("Feature I
```

In [33]:

```
plt.figure(figsize = (15,8),dpi=150)
sns.barplot(data = features.sort_values('Feature Importance'),
            x=features.sort_values('Feature Importance').index,
            y='Feature Importance')
plt.xticks(rotation=90)
plt.title("Feature Importance DecisionTree")
plt.show()

'''
Here we can see which features are 0, meaning that they are not at all important

*Codes to make the graph look neater:
-> features = feature[feature["Feature Importance"] > 0]

This line, if run, gets rid of features with value = 0
'''
```



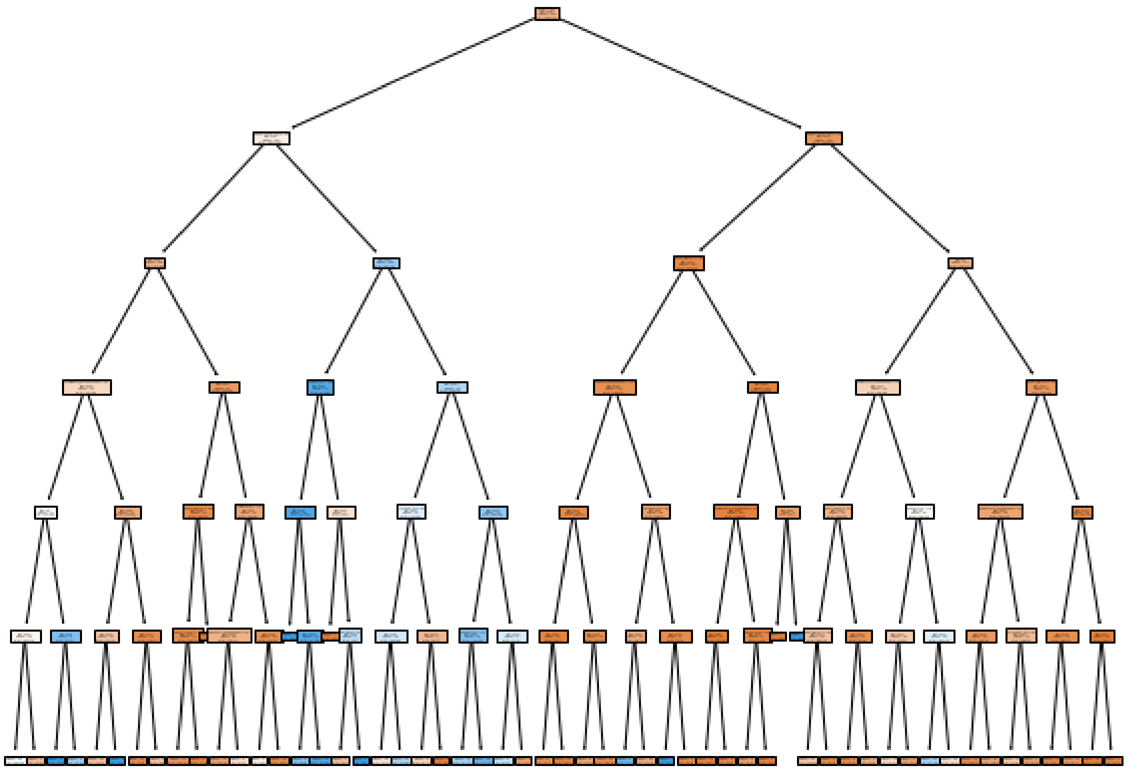
In [34]:

```
from sklearn.tree import plot_tree
```

In [36]:

```
plt.figure(figsize = (10,8))
plot_tree(tree, filled = True, feature_names = X.columns);

'''
```

```
Not necessarily a practical model since Jupyter Notebook does not allow zooming
'''
```



In [37]:

```python
# Creating a RandomForest model, and
# a classification report, and
# a confusion matrix from predicted values

from sklearn.ensemble import RandomForestClassifier
```

In [51]:

```python
random = RandomForestClassifier(n_estimators = 100,
                                max_depth = 6)
```

In [52]:

```python
random.fit(X_train, y_train)
```

Out[52]:

```
RandomForestClassifier(max_depth=6)
```

In [53]:

```python
predictions = random.predict(X_test)
```

In [58]:

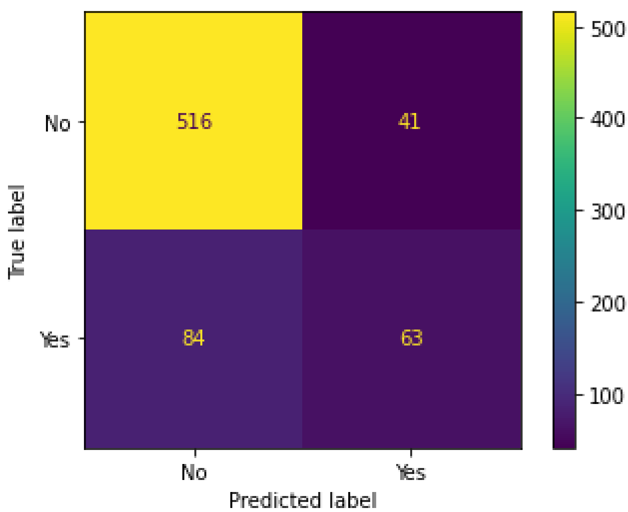
```python
print(classification_report(y_test,predictions))

'''
Slightly worse than DecisionTree when using default value -> maxDepth = None
-> accuracy = .80

Model became better when maxDepth = 6 (same as DecisionTree)
```

```
-> accuracy = .82
'''
```

```
              precision    recall  f1-score   support

          No       0.86      0.93      0.89       557
         Yes       0.61      0.43      0.50       147

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

In [59]:

```
plot_confusion_matrix(random,X_test,y_test)
plt.show()

'''
Overall better performance than DecisionTree despite predicting that 84 customer
wheras they did have churned.
'''
```



In [44]:

```
# Creating a model using AdaBoost & GradientBoosting,
# reporting back the classification report, and
# ploting a confusion matrix

from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier
```

In [45]:

```
ada = AdaBoostClassifier()
```

In [46]:

```
ada.fit(X_train, y_train)
```

Out[46]:

```
AdaBoostClassifier()
```

In [60]:

```
adaPreds = ada.predict(X_test)
```

In [61]:

```
gboost = GradientBoostingClassifier()
```

In [62]:
```
gboost.fit(X_train, y_train)
```

Out[62]:
```
GradientBoostingClassifier()
```

In [64]:
```
gboostPreds = gboost.predict(X_test)
```

In [65]:
```
print(classification_report(y_test, adaPreds))
```

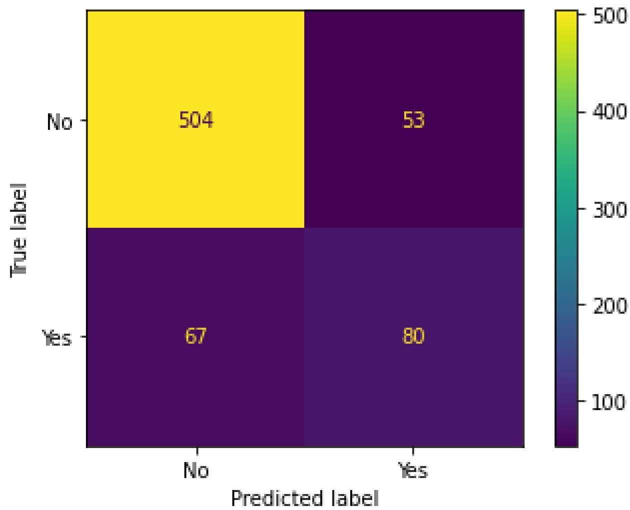
```
              precision    recall  f1-score   support

          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 [66]:
```
print(classification_report(y_test, gboostPreds))
```

```
              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
```

In [ ]:
```
'''
Despite both models being similarly accurate, AdaBoost recall rate is quite high
-> .54 vs .50

This statline indicates that AdaBoost identifies those who were going to churn m

Therefore confusion matrix will be based on AdaBoost
'''
```

In [67]:
```
plot_confusion_matrix(ada, X_test, y_test)
plt.show()
```



In [50]:

```
'''
To sum up, we got the best performance from AdaBoostClassfier

Potential future improvement for this project:

1. Perform GridSearching for optimal hyperparameters since none of the models si
those who were going to churn.

2. Focus more on adjusting n_estimators such as doubling up
'''
```