Introduction

Customer Churn Prediction

Customer attrition or churn, is when customers stop doing business with a company. It can have a significant impact on a company's revenue and it's crucial for businesses to find out the reasons why customers are leaving and take steps to reduce the number of customers leaving. One way to do this is by identifying customer segments that are at risk of leaving, and implementing retention strategies to keep them. Also, by using data and machine learning techniques, companies can predict which customers are likely to leave in the future and take actions to keep them before they decide to leave.

We are going to build a basic model for predicting customer churn using Customer Churn dataset. We are using some classification algorithm to model customers who have left, using Python tools such as pandas for data manipulation and matplotlib for visualizations.

Steps Involved to Predict Customer Churn

- Importing Libraries
- Loading Dataset
- Exploratory Data Analysis
- Outliers using IQR method
- Cleaning and Transforming Data
- One-hot Encoding
- Rearranging Columns
- Feature Scaling
- Feature Selection
- Prediction using Logistic Regression
- Prediction using Support Vector Classifier
- Prediction using Decision Tree Classifier

We have 2 types of features in the dataset: categorical (two or more values and without any order) and numerical. Most of the feature names are self-explanatory, except for:

- Partner: whether the customer has a partner or not (Yes, No),
- Dependents: whether the customer has dependents or not (Yes, No),
- OnlineBackup: whether the customer has online backup or not (Yes, No, No internet service),
- tenure: number of months the customer has stayed with the company,
- MonthlyCharges: the amount charged to the customer monthly,
- TotalCharges: the total amount charged to the customer.
- There are 7043 customers in the dataset and 19 features without customerID (non-informative) and Churn column (target variable). Most of the categorical features have 4 or less unique values.

Importing Libraries

In [1]:
```
import pandas as pd
import sklearn
import numpy as np
#import graphviz
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
# import plotly.express as px
# import plotly.graph_objects as go

%matplotlib inline
```

Loading Dataset

We use pandas to read the dataset and preprocess it

In [2]:
```
df = pd.read_csv('customer data.csv')
df.shape
```

Out[2]:
```
(7043, 21)
```

Data cleaning and filing missing values

In [3]:
```
df.head()
```

Out[3]:

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

5 rows × 21 columns

In [4]:

```
df.tail()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Devic
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	...	
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	...	
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	...	
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	...	
7042	3186-AJIEK	Male	0	No	No	66	Yes	No	Fiber optic	Yes	...	

5 rows × 21 columns

In [5]:

```
df.size
```

Out[5]:

```
147903
```

In [6]:

```
df.dtypes
```

Out[6]:

```
customerID           object
gender               object
SeniorCitizen         int64
Partner              object
Dependents           object
tenure                int64
PhoneService         object
MultipleLines        object
InternetService      object
OnlineSecurity       object
OnlineBackup         object
DeviceProtection     object
TechSupport          object
StreamingTV          object
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

In [7]:

```
df.columns
```

Out[7]:

```
Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

In [9]:

```
df.isnull().sum()
```

Out[9]:

```
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 [10]:

```
df.duplicated().sum()
```

Out[10]:

```
0
```

Basic Data Cleaning:

As we have already observered in above cell that Totalcharges is given as object datatype but it is float datatype. We will fix it here.

In [11]:

```
df['TotalCharges'].dtype
```

Out[11]:

```
dtype('O')
```

In [12]:

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],errors = 'coerce')
```

In [13]:

```
df['TotalCharges'].dtype
```

Out[13]:

```
dtype('float64')
```

In [14]:

```
categorical_features = [
    "gender",
    "SeniorCitizen",
    "Partner",
    "Dependents",
    "PhoneService",
    "MultipleLines",
    "InternetService",
    "OnlineSecurity",
    "OnlineBackup",
    "DeviceProtection",
    "TechSupport",
    "StreamingTV",
    "StreamingMovies",
```

```
    "Contract",
    "PaperlessBilling",
    "PaymentMethod",
]
numerical_features = ["tenure", "MonthlyCharges", "TotalCharges"]
target = "Churn"
```

In [15]:
```
df.skew(numeric_only= True)
```

Out[15]:
```
SeniorCitizen     1.833633
tenure            0.239540
MonthlyCharges   -0.220524
TotalCharges      0.961642
dtype: float64
```

In [16]:
```
df.corr(numeric_only=True)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_24252\2803712668.py in <module>
----> 1 df.corr(numeric_only=True)

TypeError: corr() got an unexpected keyword argument 'numeric_only'
```

Feature distribution

We plot distributions for numerical and categorical features to check for outliers and compare feature distributions with target variable.

Numerical features distribution

Numeric summarizing techniques (mean, standard deviation, etc.) don't show us spikes, shapes of distributions and it is hard to observe outliers with it. That is the reason we use histograms.

In [17]:
```
df[numerical_features].describe()
```

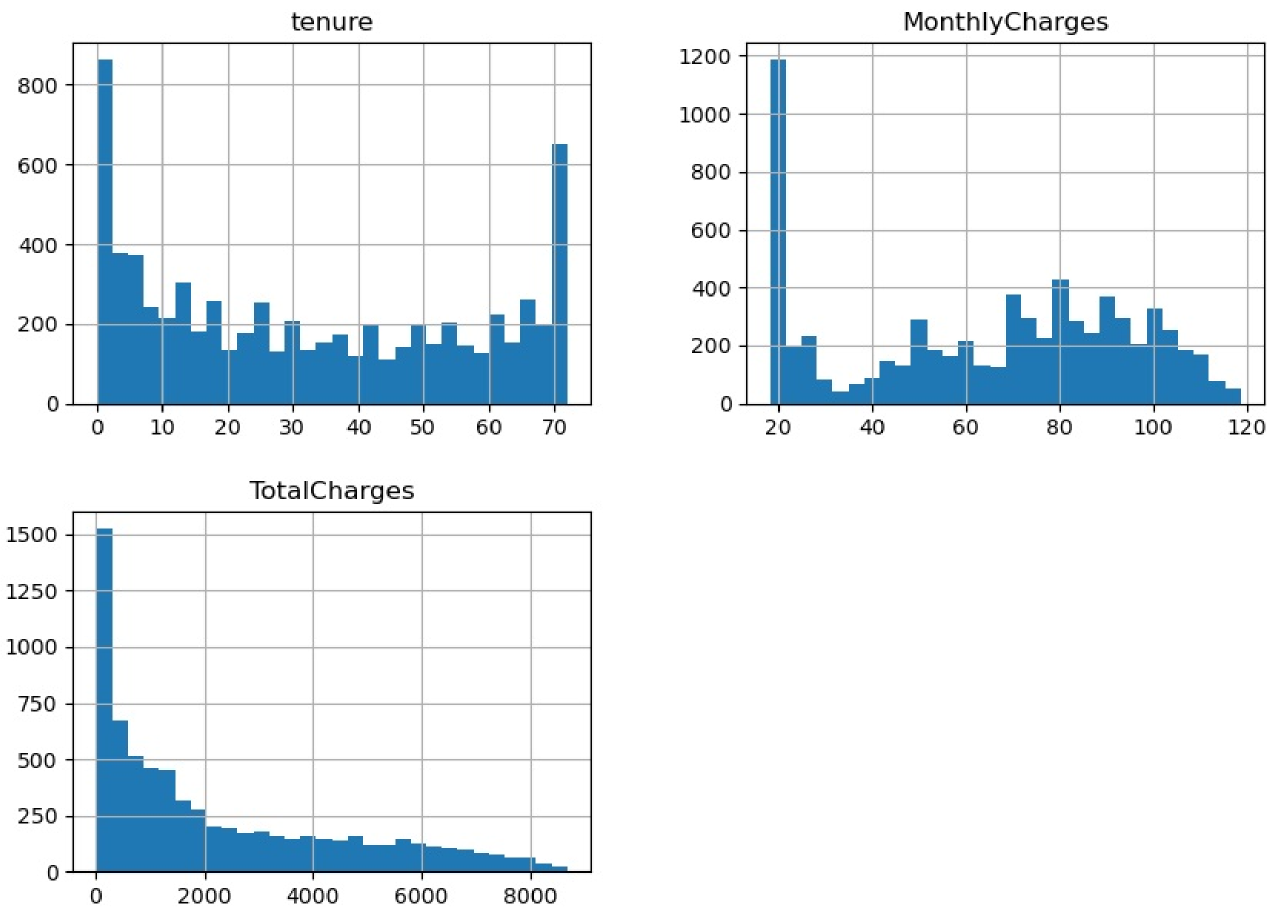
Out[17]:

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

Exploratory Data Analysis

In [18]:
```
df[numerical_features].hist(bins=30, figsize=(10, 7))
```

Out[18]:
```
array([[<AxesSubplot:title={'center':'tenure'}>,
        <AxesSubplot:title={'center':'MonthlyCharges'}>],
       [<AxesSubplot:title={'center':'TotalCharges'}>, <AxesSubplot:>]],
      dtype=object)
```



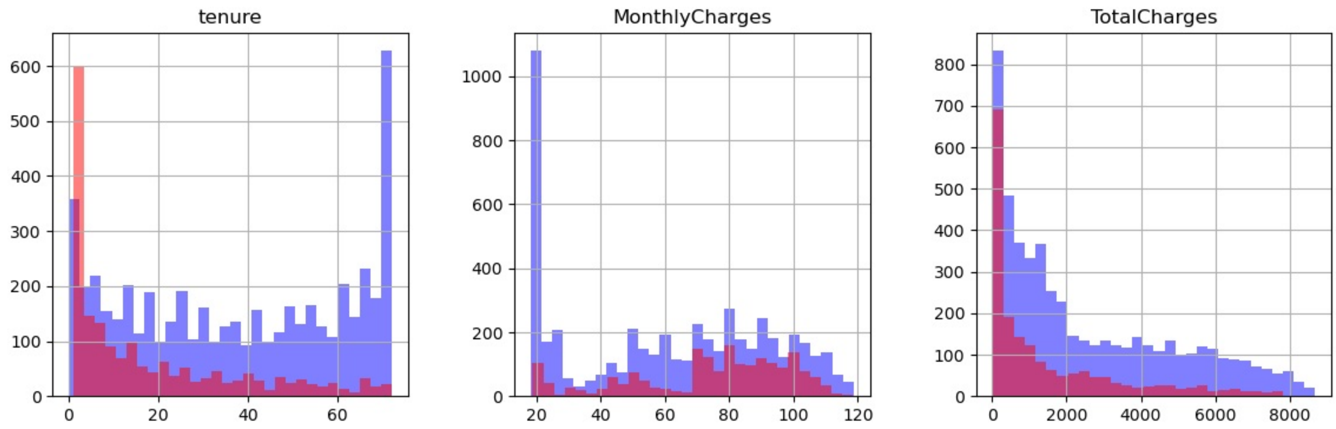
We look at distributions of numerical features in relation to the target variable. We can observe that the greater TotalCharges and tenure are the less is the probability of churn.

In [20]:

```
fig, ax = plt.subplots(1, 3, figsize=(14, 4))
df[df.Churn == "No"][numerical_features].hist(bins=30, color="blue", alpha=0.5, ax=ax)
df[df.Churn == "Yes"][numerical_features].hist(bins=30, color="red", alpha=0.5, ax=ax)
```

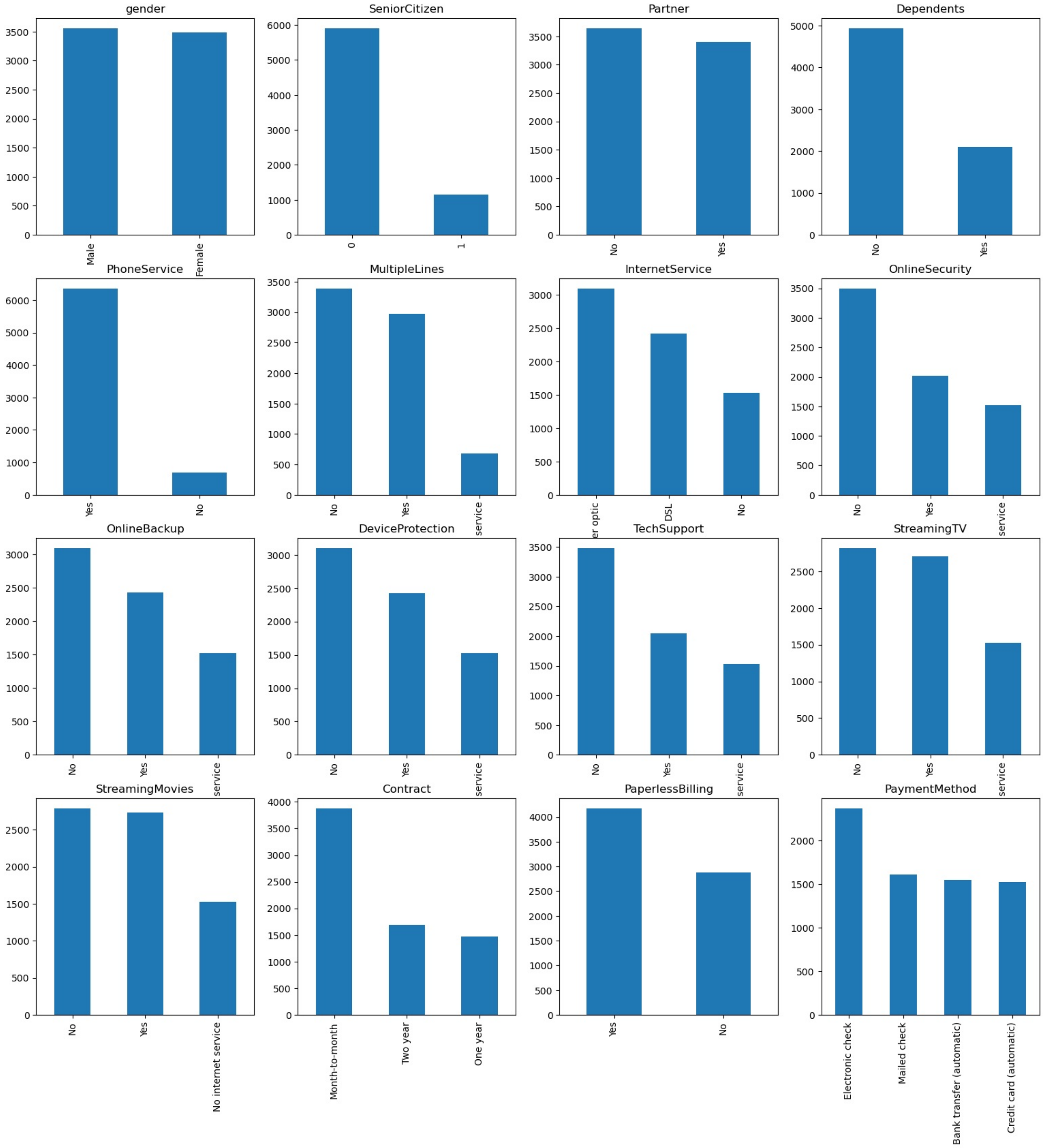
Out[20]:

```
array([<AxesSubplot:title={'center':'tenure'}>,
       <AxesSubplot:title={'center':'MonthlyCharges'}>,
       <AxesSubplot:title={'center':'TotalCharges'}>], dtype=object)
```



Categorical feature distribution To analyze categorical features, we use bar charts. We observe that Senior citizens and customers without phone service are less represented in the data.

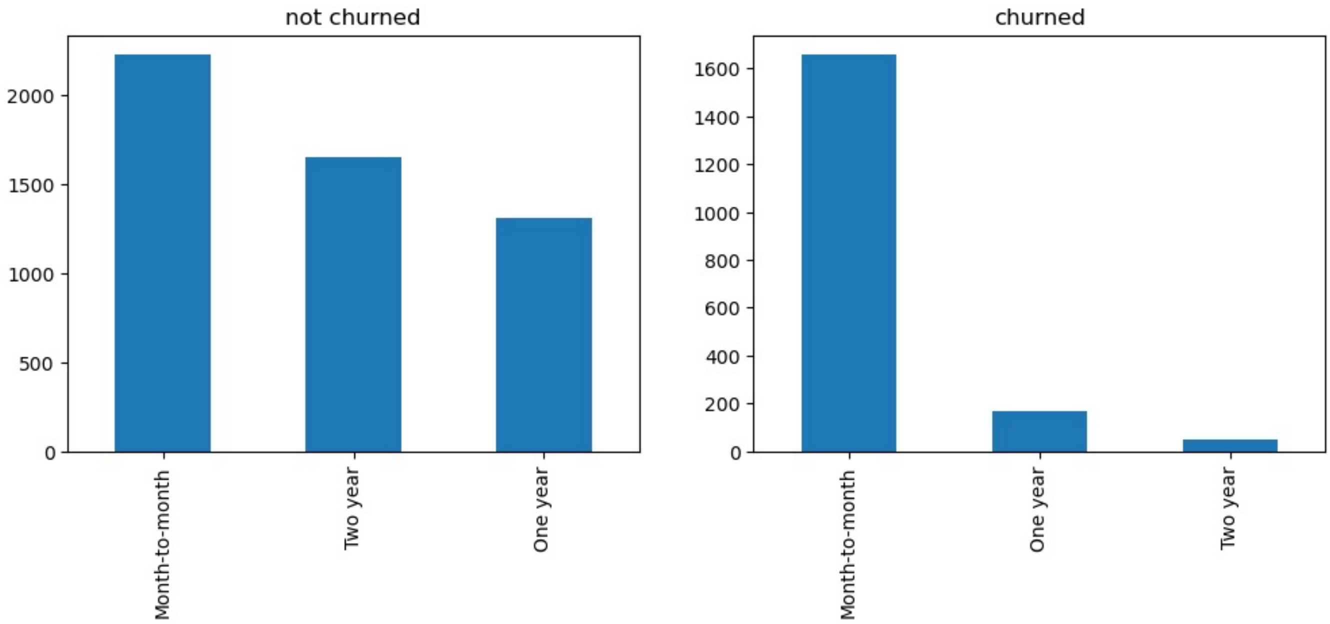
In [22]:
```
ROWS, COLS = 4, 4
fig, ax = plt.subplots(ROWS,COLS, figsize=(19,19))
row, col = 0, 0,
for i, categorical_feature in enumerate(categorical_features):
    if col == COLS - 1:
        row += 1
    col = i % COLS
    df[categorical_feature].value_counts().plot(kind='bar', ax=ax[row, col]).set_title(categorical_feature)
```



The next step is to look at categorical features in relation to the target variable. We do this only for contract feature. Users who have a month-to-month contract are more likely to churn than users with long term contracts.

In [23]:
```
feature = 'Contract'
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
df[df.Churn == "No"][feature].value_counts().plot(kind='bar', ax=ax[0]).set_title('not churned')
df[df.Churn == "Yes"][feature].value_counts().plot(kind='bar', ax=ax[1]).set_title('churned')
```

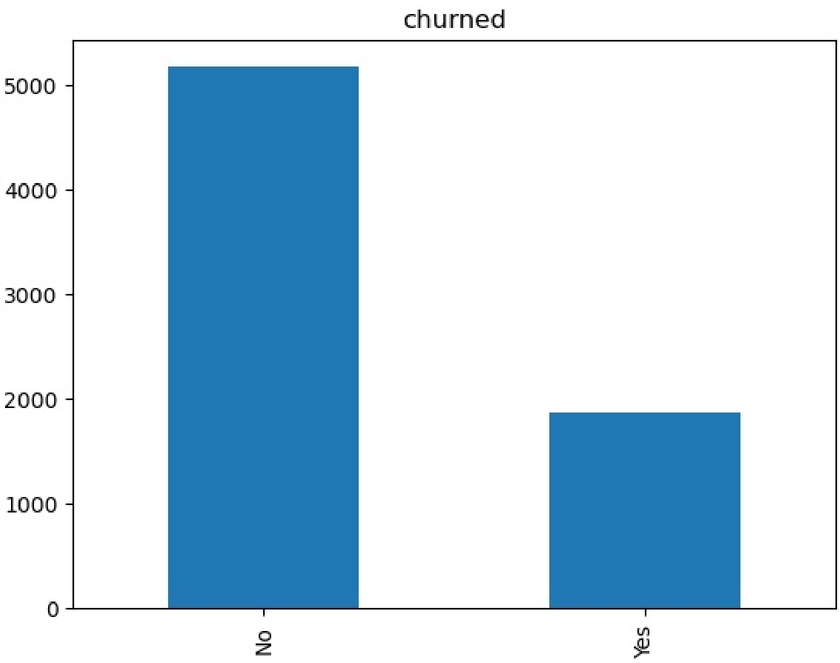
Out[23]:
```
Text(0.5, 1.0, 'churned')
```



Target variable distribution

In [24]:
```
df[target].value_counts().plot(kind='bar').set_title('churned')
```

Out[24]:
```
Text(0.5, 1.0, 'churned')
```



Target variable distribution shows that we are dealing with an imbalanced problem as there are many more non-churned as compare to churned users. The model would achieve high accuracy as it would mostly predict majority class - users who didn't churn in our example.

Few things we can do to minimize the influence of imbalanced dataset:

resample data, collect more samples, use precision and recall as accuracy metrics

Outliers Analysis with IQR Method

Outliers Analysis with IQR Method

In [25]:
```python
x = ['tenure','MonthlyCharges']
def count_outliers(data,col):
        q1 = data[col].quantile(0.25,interpolation='nearest')
        q2 = data[col].quantile(0.5,interpolation='nearest')
        q3 = data[col].quantile(0.75,interpolation='nearest')
        q4 = data[col].quantile(1,interpolation='nearest')
        IQR = q3 -q1
        global LLP
        global ULP
        LLP = q1 - 1.5*IQR
        ULP = q3 + 1.5*IQR
        if data[col].min() > LLP and data[col].max() < ULP:
            print("No outliers in",i)
        else:
            print("There are outliers in",i)
            x = data[data[col]<LLP][col].size
            y = data[data[col]>ULP][col].size
            a.append(i)
            print('Count of outliers are:',x+y)
global a
a = []
for i in x:
    count_outliers(df,i)
```

```
No outliers in tenure
No outliers in MonthlyCharges
```

Cleaning and Transforming Data

In [26]:
```python
df.drop(['customerID'],axis = 1,inplace = True)
```

In [27]:
```python
df.head()
```

Out[27]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DevicePro
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	

Dropped customerID because it is not needed

On Hot Encoding

In [28]:
```python
df1=pd.get_dummies(data=df,columns=['gender', 'Partner', 'Dependents',
       'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'Churn'], drop_first=True)
```

In [29]:
```python
df1.head()
```

Out[29]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	gender_Male	Partner_Yes	Dependents_Yes	PhoneService_Yes	MultipleLines_No phone service	Mu
0	0	1	29.85	29.85	0	1	0	0	1	
1	0	34	56.95	1889.50	1	0	0	1	0	
2	0	2	53.85	108.15	1	0	0	1	0	
3	0	45	42.30	1840.75	1	0	0	0	1	
4	0	2	70.70	151.65	0	0	0	1	0	

5 rows × 31 columns

In [30]:
```python
df1.columns
```

Out[30]:
```
Index(['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges',
       'gender_Male', 'Partner_Yes', 'Dependents_Yes', 'PhoneService_Yes',
       'MultipleLines_No phone service', 'MultipleLines_Yes',
       'InternetService_Fiber optic', 'InternetService_No',
       'OnlineSecurity_No internet service', 'OnlineSecurity_Yes',
       'OnlineBackup_No internet service', 'OnlineBackup_Yes',
       'DeviceProtection_No internet service', 'DeviceProtection_Yes',
       'TechSupport_No internet service', 'TechSupport_Yes',
       'StreamingTV_No internet service', 'StreamingTV_Yes',
       'StreamingMovies_No internet service', 'StreamingMovies_Yes',
       'Contract_One year', 'Contract_Two year', 'PaperlessBilling_Yes',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check',
       'Churn_Yes'],
      dtype='object')
```

Rearranging Columns

In [31]:
```
df1 = df1[['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges',
        'gender_Male', 'Partner_Yes', 'Dependents_Yes',
       'PhoneService_Yes', 'MultipleLines_No phone service',
       'MultipleLines_Yes', 'InternetService_Fiber optic',
       'InternetService_No', 'OnlineSecurity_No internet service',
       'OnlineSecurity_Yes', 'OnlineBackup_No internet service',
       'OnlineBackup_Yes', 'DeviceProtection_No internet service',
       'DeviceProtection_Yes', 'TechSupport_No internet service',
       'TechSupport_Yes', 'StreamingTV_No internet service', 'StreamingTV_Yes',
       'StreamingMovies_No internet service', 'StreamingMovies_Yes',
       'Contract_One year', 'Contract_Two year', 'PaperlessBilling_Yes',
       'PaymentMethod_Credit card (automatic)',
       'PaymentMethod_Electronic check', 'PaymentMethod_Mailed check','Churn_Yes']]
```

In [32]:
```
df1.head()
```

Out[32]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges	gender_Male	Partner_Yes	Dependents_Yes	PhoneService_Yes	MultipleLines_No phone service	Mu
0	0	1	29.85	29.85	0	1	0	0	1	
1	0	34	56.95	1889.50	1	0	0	1	0	
2	0	2	53.85	108.15	1	0	0	1	0	
3	0	45	42.30	1840.75	1	0	0	0	1	
4	0	2	70.70	151.65	0	0	0	1	0	

5 rows × 31 columns

In [33]:
```
df1.shape
```

Out[33]:
```
(7043, 31)
```

In [34]:
```
from sklearn.impute import SimpleImputer

# The imputer will replace missing values with the mean of the non-missing values for the respective columns
imputer = SimpleImputer(missing_values=np.nan, strategy="mean")

df1.TotalCharges = imputer.fit_transform(df1["TotalCharges"].values.reshape(-1, 1))
```

Feature Scaling

In [35]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [36]:
```
scaler.fit(df1.drop(['Churn_Yes'],axis = 1))
scaled_features = scaler.transform(df1.drop('Churn_Yes',axis = 1))
```

Feature Selection

In [37]:
```
from sklearn.model_selection import train_test_split
X = scaled_features
Y = df1['Churn_Yes']
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.3,random_state=44)
```

Prediction using Logistic Regression

In [38]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report,accuracy_score ,confusion_matrix
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,Y_train)
```

Out[38]:

```
LogisticRegression()
```

In [39]:

```
predLR = logmodel.predict(X_test)
```

In [40]:

```
predLR
```

Out[40]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=uint8)
```

In [41]:

```
Y_test
```

Out[41]:

```
5616    0
2937    0
1355    0
5441    1
3333    0
       ..
2797    1
412     0
174     0
5761    0
5895    0
Name: Churn_Yes, Length: 2113, dtype: uint8
```

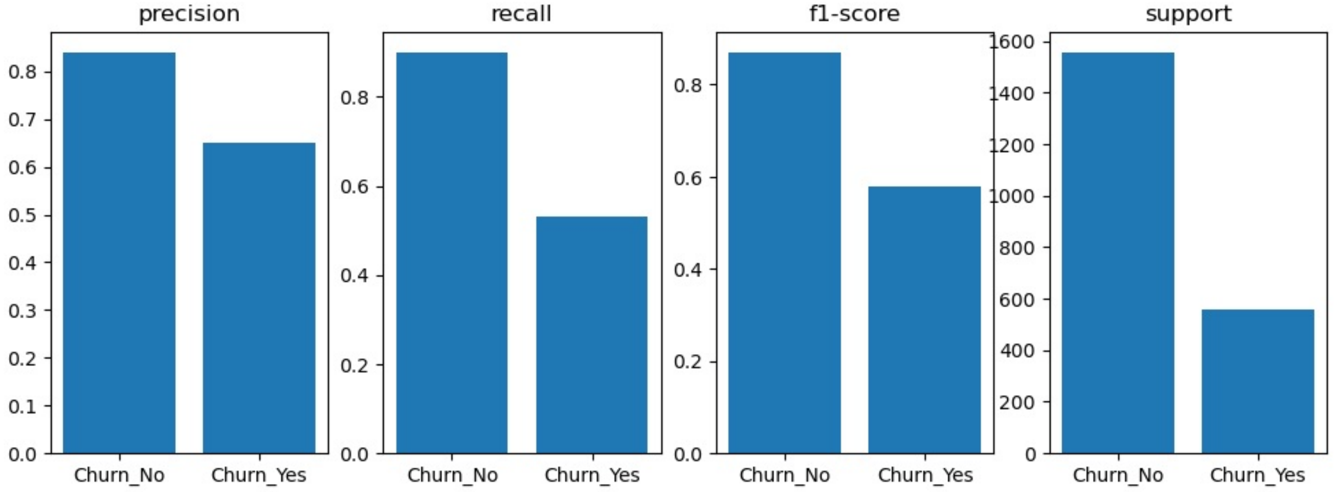
In [42]:

```
print(classification_report(Y_test, predLR))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1557
           1       0.65      0.53      0.58       556

    accuracy                           0.80      2113
   macro avg       0.74      0.71      0.73      2113
weighted avg       0.79      0.80      0.79      2113
```

In [43]:

```
# calculate the classification report
report = classification_report(Y_test, predLR, target_names=['Churn_No', 'Churn_Yes'])

# split the report into lines
lines = report.split('\n')

# split each line into parts
parts = [line.split() for line in lines[2:-5]]

# extract the metrics for each class
class_metrics = dict()
for part in parts:
    class_metrics[part[0]] = {'precision': float(part[1]), 'recall': float(part[2]), 'f1-score': float(part[3])}

# create a bar chart for each metric
fig, ax = plt.subplots(1, 4, figsize=(12, 4))
metrics = ['precision', 'recall', 'f1-score', 'support']
for i, metric in enumerate(metrics):
    ax[i].bar(class_metrics.keys(), [class_metrics[key][metric] for key in class_metrics.keys()])
    ax[i].set_title(metric)

# display the plot
plt.show()
```



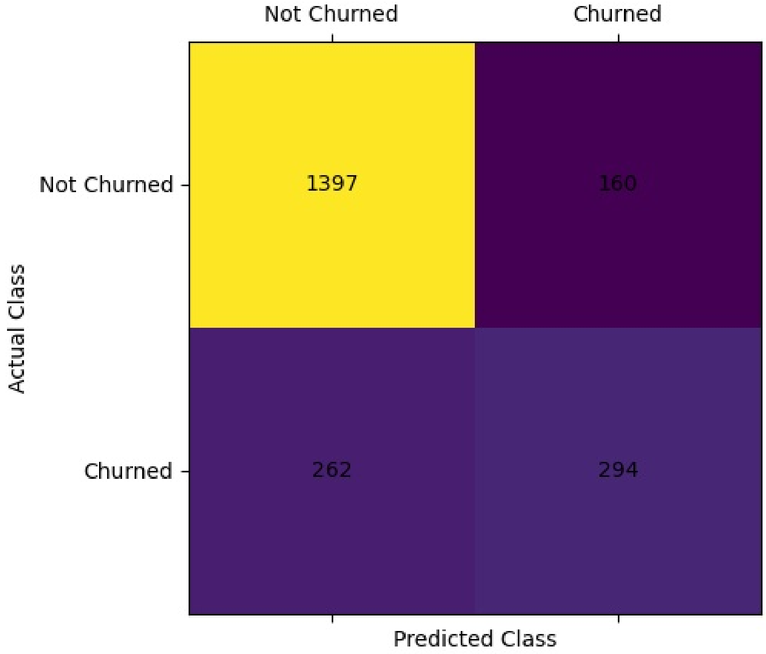
In [44]:

```
confusion_matrix_LR = confusion_matrix(Y_test, predLR)
```

In [45]:

```
# create a heatmap of the matrix using matshow()
```

```
plt.matshow(confusion_matrix(Y_test, predLR))

# add labels for the x and y axes
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

for i in range(2):
    for j in range(2):
        plt.text(j, i, confusion_matrix_LR[i, j], ha='center', va='center')


# Add custom labels for x and y ticks
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.yticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```



In [46]:
```
logmodel.score(X_train, Y_train)
```

Out[46]: 0.8062880324543611

In [47]:
```
accuracy_score(Y_test, predLR)
```

Out[47]: 0.8002839564600095

Prediction using Support Vector Classifie

In [48]:
```
from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, Y_train)
y_pred_svc = svc.predict(X_test)
```

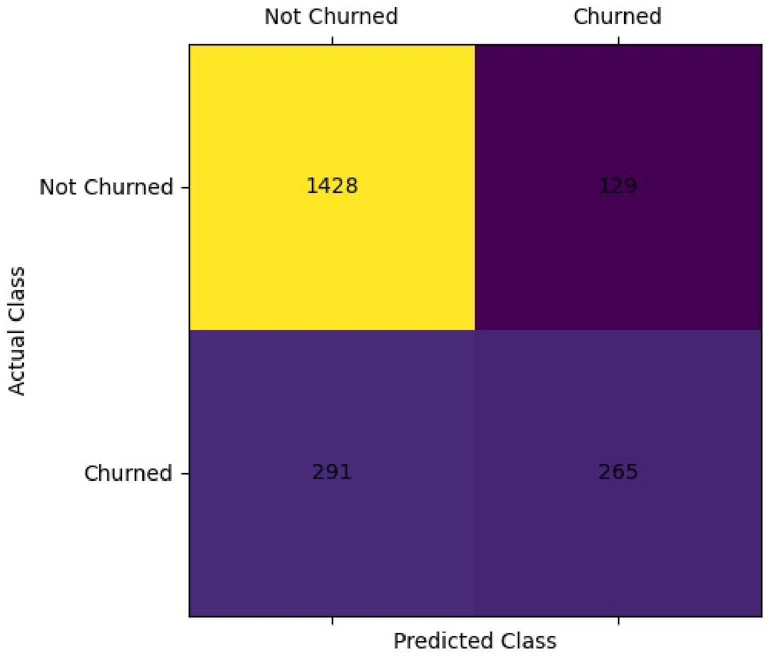
In [49]:
```
print(classification_report(Y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.83      0.92      0.87      1557
           1       0.67      0.48      0.56       556

    accuracy                           0.80      2113
   macro avg       0.75      0.70      0.71      2113
weighted avg       0.79      0.80      0.79      2113
```

In [50]:
```
confusion_matrix_svc = confusion_matrix(Y_test, y_pred_svc)
```

In [51]:
```
plt.matshow(confusion_matrix_svc)

# add labels for the x and y axes
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

for i in range(2):
    for j in range(2):
        plt.text(j, i, confusion_matrix_svc[i, j], ha='center', va='center')
```

```python
# Add custom labels for x and y ticks
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.yticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```



In [52]:
```python
svc.score(X_train,Y_train)
```

Out[52]:
```
0.81703853955537525
```

In [53]:
```python
accuracy_score(Y_test, y_pred_svc)
```

Out[53]:
```
0.8012304779933743
```

Prediction using Decision Tree Classifier

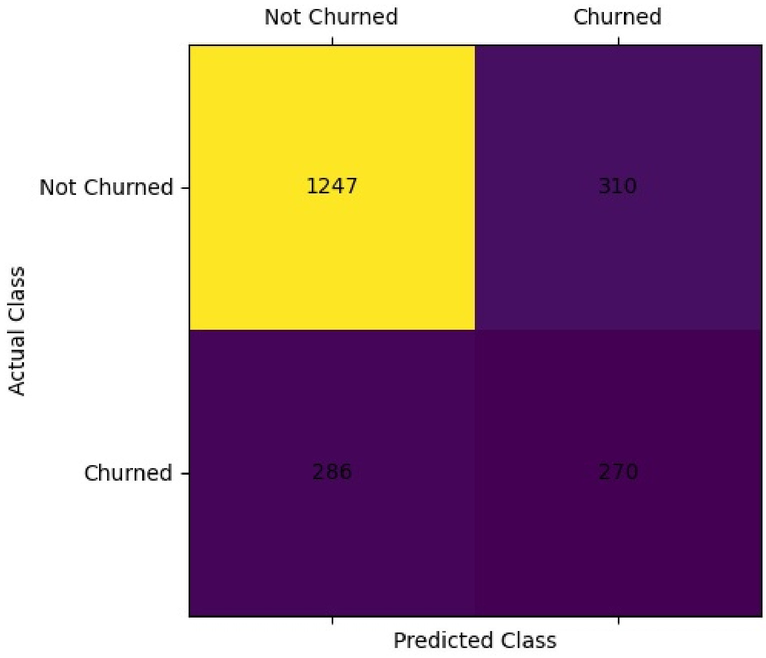
In [54]:
```python
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier()

dtc.fit(X_train, Y_train)
y_pred_dtc = dtc.predict(X_test)
```

In [55]:
```python
print(classification_report(Y_test, y_pred_dtc))
```

```
              precision    recall  f1-score   support

           0       0.81      0.80      0.81      1557
           1       0.47      0.49      0.48       556

    accuracy                           0.72      2113
   macro avg       0.64      0.64      0.64      2113
weighted avg       0.72      0.72      0.72      2113
```

In [56]:
```python
confusion_matrix_dtc = confusion_matrix(Y_test, y_pred_dtc)
```

In [57]:
```python
# create a heatmap of the matrix using matshow()

plt.matshow(confusion_matrix_dtc)

# add labels for the x and y axes
plt.xlabel('Predicted Class')
plt.ylabel('Actual Class')

for i in range(2):
    for j in range(2):
        plt.text(j, i, confusion_matrix_dtc[i, j], ha='center', va='center')


# Add custom labels for x and y ticks
plt.xticks([0, 1], ["Not Churned", "Churned"])
plt.yticks([0, 1], ["Not Churned", "Churned"])
plt.show()
```




```
In [58]: dtc.score(X_train,Y_train)
```

```
Out[58]: 0.9987829614604462
```

```
In [59]: accuracy_score(Y_test, y_pred_dtc)
```

```
Out[59]: 0.7179365830572646
```

Conclusion Learning more about this dataset, we can explore other machine learning classification models such as Ada Boost Classifier, Gradient Boosting Classifier, Stochastic Gradient Boosting (SGB) Classifier, Cat Boost Classifier and XGB Boost Classifier. Additionally, we can try tuning the model's hyperparameters using techniques like GridSearchCV.

```
In [ ]:
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js