Phase 3 project

Business Understanding

Problem Statement

The marketing team in syriatel would like to understand churn trends help them become more competitive against competition. This will help to improve their customer acquistion and retention strategy

Objectives

1. Understanding the reasons behind customer churn
2. Build a prediction model to help proof the business against churn
3. Reduce churn to improve business performance

Data Understanding

Below we will perform a series of steps to prepare the data. We will import the data, preview a few rows, then create a class to help us query the data for some basic information

Importing Data

In [94]:

```
# perform necessary imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('dark')

# load dataset
df = pd.read_csv('data/churn.csv')

# preview first 5 rows
df.head()
```

Out[94]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...

5 rows × 21 columns

The code below builds a class `quer_d` that will help us query the data

In [95]:

```
# build the class

class quer_d:

    """ Query dataframe for specific information"""

    def __init__(self):
        self.data = data

    def dshape(self, data):
        """Simple method to provide the shape of the data"""

        return print(f"The DataFrame has:\n\t* {data.shape[0]} rows\n\t* {data.shape[1]}

    def dinfo(self, data):
        """Simple method to provide the info of the data"""
        return print(data.info(), '\n')

    def dmissing(self, data):
        """ Identify missing values"""
        # identify if data has missing values(data.isnull().any())
        # empty dict to store missing values
        missing = []
        for i in data.isnull().any():
            # add the bool values to empty list
            missing.append(i)
        # covert list to set (if data has missing value, the list should have true and f
        missing_set = set(missing)
        if (len(missing_set) == 1):
            out = print("The Data has no missing values", '\n')
        else:
            out = print(f"The Data has missing values.", '\n')

        return out

    def d_duplic(self, data):
        """method to identify any duplicates"""
        # identify the duplicates (dataframename.duplicated() , can add .sum() to get to
        # empty list to store Bool results from duplicated
        duplicates = []
        for i in data.duplicated():
            duplicates.append(i)
        # identify if there is any duplicates. (If there is any we expect a True value i
        duplicates_set = set(duplicates)
        if (len(duplicates_set) == 1):
            out = print("The Data has no duplicates", '\n')
        else:
            no_true = 0
            for val in duplicates:
                if (val == True):
                    no_true += 1
            # percentage of the data represented by duplicates
            duplicates_percentage = np.round(((no_true / len(data)) * 100), 3)
            out = print(f"The Data has {no_true} duplicated rows.\nThis constitutes {dup

        return out

    def col_dup(self, data, column):
        """handling duplicates in unique column"""
```

```
        # empty list to store the duplicate bools
        duplicates = []
        for i in data[column].duplicated():
            duplicates.append(i)

        # identify if there are any duplicates
        duplicates_set = set(duplicates)
        if (len(duplicates_set) == 1):
            out = print(f"The column {column.title()} has no duplicates", '\n')
        else:
            no_true = 0
            for val in duplicates:
                if (val == True):
                    no_true += 1
            # percentage of the data represented by duplicates
            duplicates_percentage = np.round(((no_true / len(data)) * 100), 3)
            out = print(f"The column {column.title()} has {no_true} duplicated rows.\nTh

        return out

    def d_describe(self, data):

        """method to check the descriptive values of the data"""
        return print(data.describe(), '\n')
```

Below shows that taining data has 3333 cases and 21 features. There are a mixture of strings, floats and integers

- phone number is saved as string but will need to be converted to numeric
- churn column is saved as boolean, but will be converted to integer
- at first glance, data has no missing values and no duplicates

In [96]:

```
# instantiate class
inst = quer_d()

inst.dshape(df) # shape
inst.dinfo(df)  # info
inst.dmissing(df) # missing
inst.d_duplic(df) # duplicates
inst.col_dup(df, 'phone number') # unique col duplicates
inst.d_describe(df) # descriptive stats
```

```
The DataFrame has:
        * 3333 rows
        * 21 columns

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
None

The Data has no missing values

The Data has no duplicates

The column Phone Number has no duplicates

       account length    area code  number vmail messages  total day minut
es  \
count     3333.000000  3333.000000            3333.000000        3333.0000
00   
mean       101.064806   437.182418               8.099010         179.7750
98   
std         39.822106    42.371290              13.688365          54.4673
89   
min          1.000000   408.000000               0.000000           0.0000
00   
25%         74.000000   408.000000               0.000000         143.7000
00   
50%        101.000000   415.000000               0.000000         179.4000
00   
75%        127.000000   510.000000              20.000000         216.4000
00   
max        243.000000   510.000000              51.000000         350.8000
00   

       total day calls  total day charge  total eve minutes  total eve cal
ls  \
```

```
count      3333.000000       3333.000000        3333.000000      3333.0000
00
mean        100.435644         30.562307         200.980348       100.1143
11
std          20.069084          9.259435          50.713844        19.9226
25
min           0.000000          0.000000           0.000000         0.0000
00
25%          87.000000         24.430000         166.600000        87.0000
00
50%         101.000000         30.500000         201.400000       100.0000
00
75%         114.000000         36.790000         235.300000       114.0000
00
max         165.000000         59.640000         363.700000       170.0000
00

       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
75%             3.270000                2.000000
max             5.400000                9.000000
```

Data Cleaning

Below we create a class that will:

- convert phone numbers encoded as string to integers
- convert target column encoded as boolean to integer

In [97]:

```
# create class
class trans:
    """ converting columns to appropriate data type"""

    def __init__(self):
        self.data = data

    def conv(self, data, col):
        """ convert phone number to integer"""
        data[col] = data[col].str.replace('-', '').astype('int')
        return data

    def lab(self, data, col):
        """convert churn col to integer"""
        data[col] = data[col].astype('int')

        return data

# instantiate class
chg = trans()
```

Convert Phone Number to Integer

Phone number was saved as a string. The code below will convert it to an integ

In [98]:

```
# apply instantiated class on dataframe
chg.conv(df, 'phone number')
```

Out[98]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
0	KS	128	415	3824657	no	yes	25	265.1	110	45.07
1	OH	107	415	3717191	no	yes	26	161.6	123	27.47
2	NJ	137	415	3581921	no	no	0	243.4	114	41.38
3	OH	84	408	3759999	yes	no	0	299.4	71	50.90
4	OK	75	415	3306626	yes	no	0	166.7	113	28.34
...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	4144276	no	yes	36	156.2	77	26.55
3329	WV	68	415	3703271	no	no	0	231.1	57	39.29
3330	RI	28	510	3288230	no	no	0	180.8	109	30.74
3331	CT	184	510	3646381	yes	no	0	213.8	105	36.35
3332	TN	74	415	4004344	no	yes	25	234.4	113	39.85

3333 rows × 21 columns

Convert Churn Column to Integer

The `churn` col is currently encoded as boolean. The function below converts it to a binary variable

In [99]:

```
# apply instantiated class on dataframe

chg.lab(df, 'churn')
```

Out[99]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge
0	KS	128	415	3824657	no	yes	25	265.1	110	45.07
1	OH	107	415	3717191	no	yes	26	161.6	123	27.47
2	NJ	137	415	3581921	no	no	0	243.4	114	41.38
3	OH	84	408	3759999	yes	no	0	299.4	71	50.90
4	OK	75	415	3306626	yes	no	0	166.7	113	28.34
...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	4144276	no	yes	36	156.2	77	26.55
3329	WV	68	415	3703271	no	no	0	231.1	57	39.29
3330	RI	28	510	3288230	no	no	0	180.8	109	30.74
3331	CT	184	510	3646381	yes	no	0	213.8	105	36.35
3332	TN	74	415	4004344	no	yes	25	234.4	113	39.85

3333 rows × 21 columns

Exploratory Data Analysis

Mismatch: Voicemail and International

The code below shows that columns `voice mail plan` and `international plan` categories don't match with actual data. Below we analyse day minutes and see that non subscribers have values. We will drop both columns as I believe the actual transactional data will be able to relay the required patterns for the different subscribers

We will also drop the `state` column because the data contains 56 unique states, thus one hot encoding this will be cumbersome. Since the `area code` column also contains geographic information, we'll use this for our model

In [100]:

```
print(df.groupby('voice mail plan')['total day minutes'].mean(), '\n')
print(df.groupby('international plan')['total intl minutes'].mean())
```

```
voice mail plan
no     179.831813
yes    179.626790
Name: total day minutes, dtype: float64

international plan
no     10.195349
yes    10.628173
Name: total intl minutes, dtype: float64
```

In [101]:

```
# function to drop columns

def drp(data, col):
    """drop specified column"""
    data.drop(col, axis=1, inplace=True)

    return data

# apply to dataframe

drp(df, ['voice mail plan', 'international plan', 'state'])
```

Out[101]:

	account length	area code	phone number	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	m
0	128	415	3824657	25	265.1	110	45.07	197.4	99	16.78	
1	107	415	3717191	26	161.6	123	27.47	195.5	103	16.62	
2	137	415	3581921	0	243.4	114	41.38	121.2	110	10.30	
3	84	408	3759999	0	299.4	71	50.90	61.9	88	5.26	
4	75	415	3306626	0	166.7	113	28.34	148.3	122	12.61	
...	...	...	...	...	...	...	...	...	...	...	
3328	192	415	4144276	36	156.2	77	26.55	215.5	126	18.32	
3329	68	415	3703271	0	231.1	57	39.29	153.4	55	13.04	
3330	28	510	3288230	0	180.8	109	30.74	288.8	58	24.55	
3331	184	510	3646381	0	213.8	105	36.35	159.6	84	13.57	
3332	74	415	4004344	25	234.4	113	39.85	265.9	82	22.60	

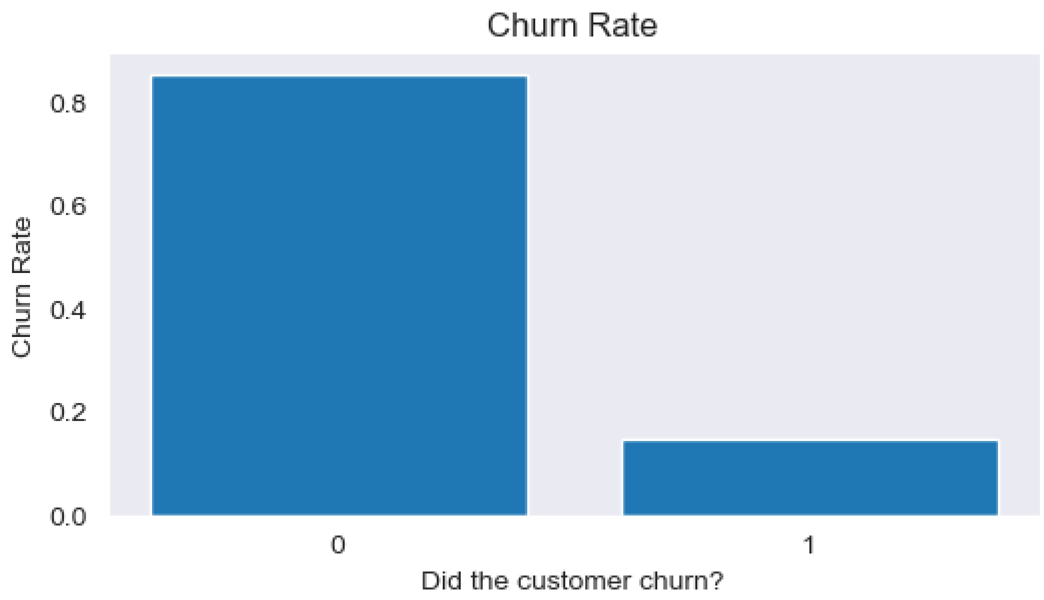
3333 rows × 18 columns

Churn rate

The data shows a churn rate of 14%, meaning that our target variable is imbalanced. We will therefore have to correct for the imbalances when modeling

In [114]:

```
# plot the churn rate
fig, ax = plt.subplots(figsize=(6, 3))

plt.bar(df.churn.value_counts(normalize=True).index, df.churn.value_counts(normalize=Tru
plt.xticks([0, 1])
plt.title('Churn Rate')
plt.xlabel('Did the customer churn?')
plt.ylabel('Churn Rate');
```



Geographic Distribution

Roughly half of the subscribers are located in area code 415. The remainder are evenly distributed between 408 and 510

In [133]:

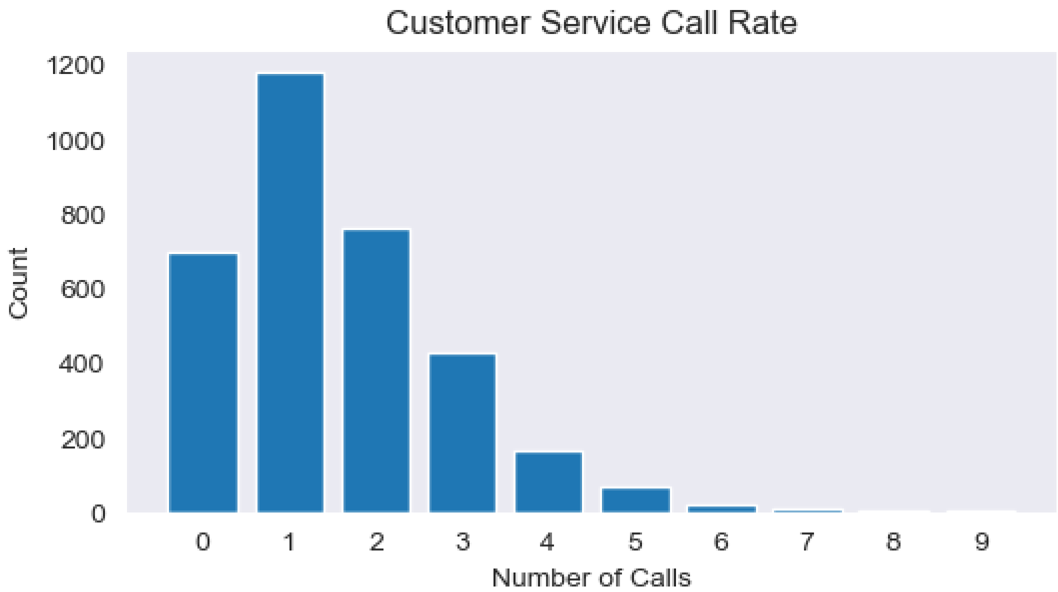
```
# regional distribution
fig, ax = plt.subplots(figsize=(6, 3))

plt.bar(['415', '408', '510'], df['area code'].value_counts(normalize=True).values)
plt.title('Area code distribution')
plt.xlabel('Area code')
plt.ylabel('Percentage');
```

Customer Service Calls

Calls to customer service is binomially distributed with most people making 1 to 3 calls

In [137]:

```
# customer service calls
fig, ax = plt.subplots(figsize=(6, 3))

plt.bar(df['customer service calls'].value_counts().index, df['customer service calls'].
plt.xticks(df['customer service calls'].value_counts().index)
plt.title('Customer Service Call Rate')
plt.xlabel('Number of Calls')
plt.ylabel('Count');
```

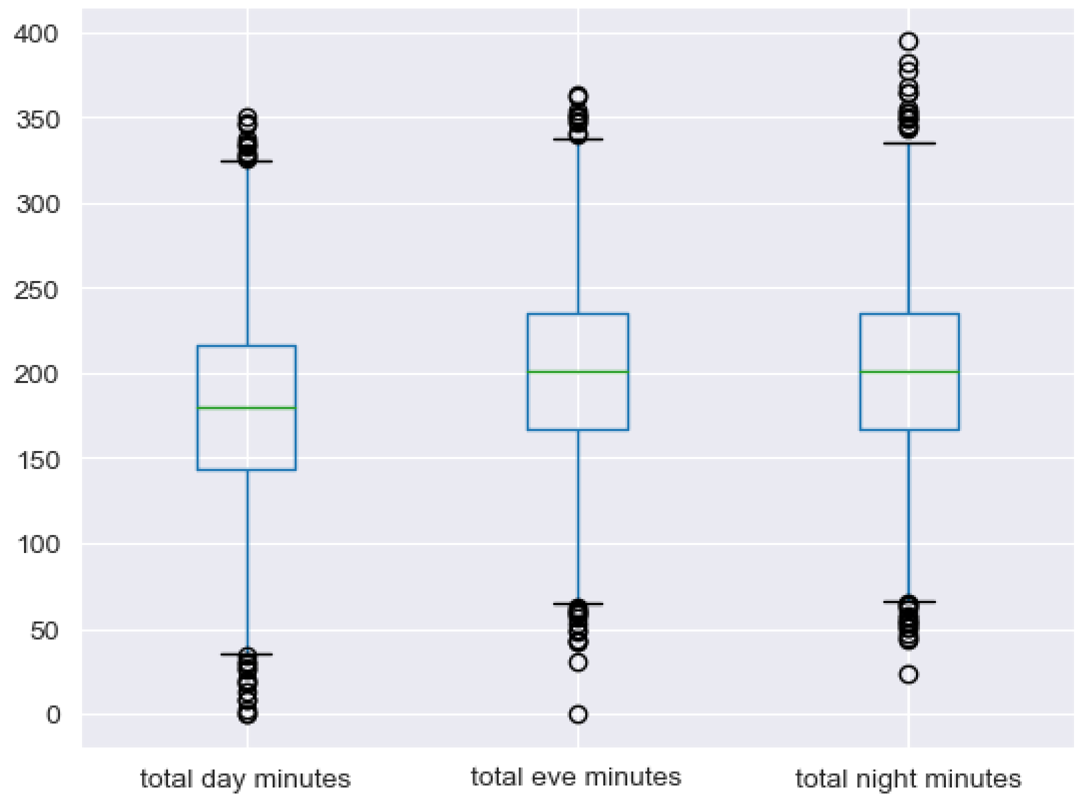


How Does Call Duration Vary By Day Part?

Call duration increases by day part, thus evening and night calls last longer than day calls

In [141]:

```
df[['total day minutes', 'total eve minutes', 'total night minutes']].boxplot();
```



Modelling

Since our target variable is not continuous, we will consider classification models:

- we will split training and test data
- standardize the data since the columns have data on varying scales
- start with logistic regression as our basline model, then move on to more sophisticated models
- account for imbalanced classes in the target variable

In [143]:

```
# imports
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import roc_curve, auc
from sklearn.linear_model import LogisticRegression

# split data into features labels
X = df.drop('churn', axis=1)
y = df.churn

# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
```

Baseline Model

We start with logistic regression as our baseline model. We will use a pipeline to streamline our work and balanced class weight to account for class imbalances

In [150]:

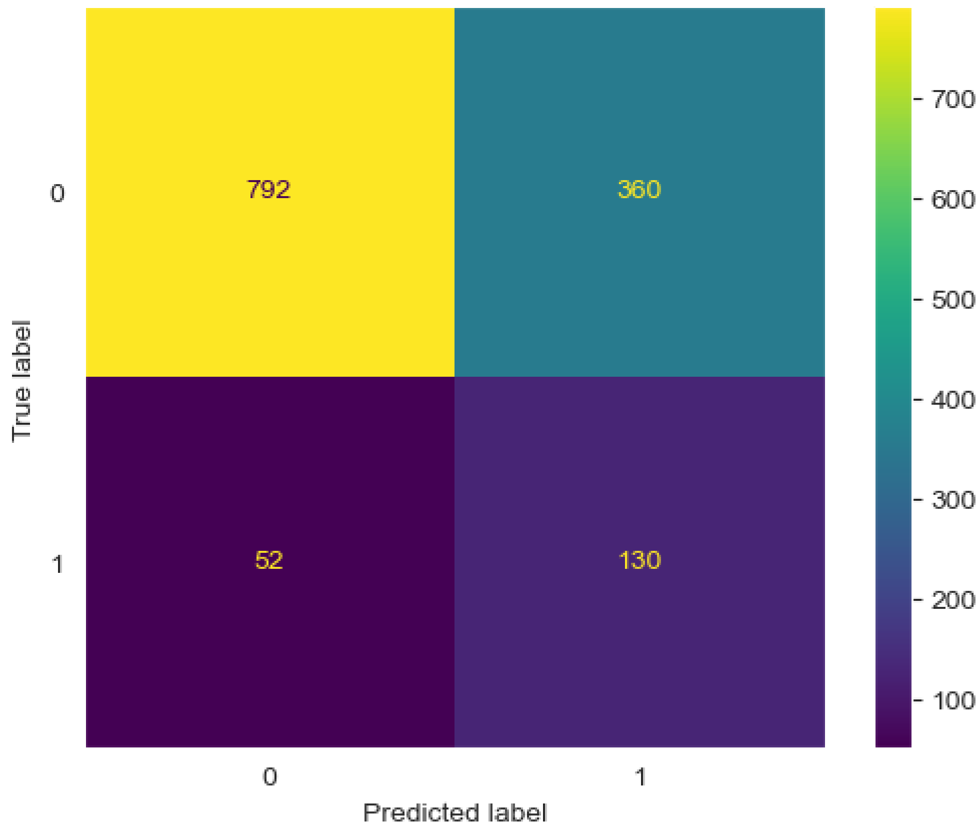
```
# create pipeline
log_p = Pipeline([('scale', StandardScaler()),
                 ('log', LogisticRegression(random_state=0, class_weight='balanced', C=1

# fit to training data
log_p.fit(X_train, y_train)

# predicted output
y_pred_train = log_p.predict(X_train)
y_pred_test = log_p.predict(X_test)
```

From the metrics below, we see that our basline model can be improved. We have a high number of false positives affecting the precision score. However, our key metrics to consider are the accuracy and auc scores

In [154]:

```
from sklearn.metrics import ConfusionMatrixDisplay, precision_score, recall_score, accur

# function to print metrics

def clas_score(y, y_pred):
    a = print(f'precision: {round(precision_score(y, y_pred), 3)}')
    b = print(f'recall: {round(recall_score(y, y_pred), 3)}')
    c = print(f'accuracy: {round(accuracy_score(y, y_pred), 3)}')
    fpr, tpr, thresholds = roc_curve(y, y_pred)
    d = print('AUC: {}'.format(round(auc(fpr, tpr), 3)))

    return a, b, c, d

clas_score(y_test, y_pred_test)

# plot confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred_test);
```

```
precision: 0.265
recall: 0.714
accuracy: 0.691
AUC: 0.701
```



Improved Model: Decision Tree

Next we will use DecisionTree with grid search to look for optimal solutions

In [161]:

```
# import
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Create the pipeline
pipe = Pipeline([('sc', StandardScaler()),
                 ('tree', DecisionTreeClassifier(random_state=123, class_weight='balance

# Create the grid parameter
grid = [{'tree__max_depth': [None, 2, 6, 10],
         'tree__min_samples_split': [5, 10]}]


# Create the grid, with "pipe" as the estimator
gridsearch = GridSearchCV(estimator=pipe,
                          param_grid=grid,
                          scoring='accuracy',
                          cv=5)

# fit to training data
gridsearch.fit(X_train, y_train)

# predicted output
y_h_train = gridsearch.predict(X_train)
y_h_test = gridsearch.predict(X_test)
```

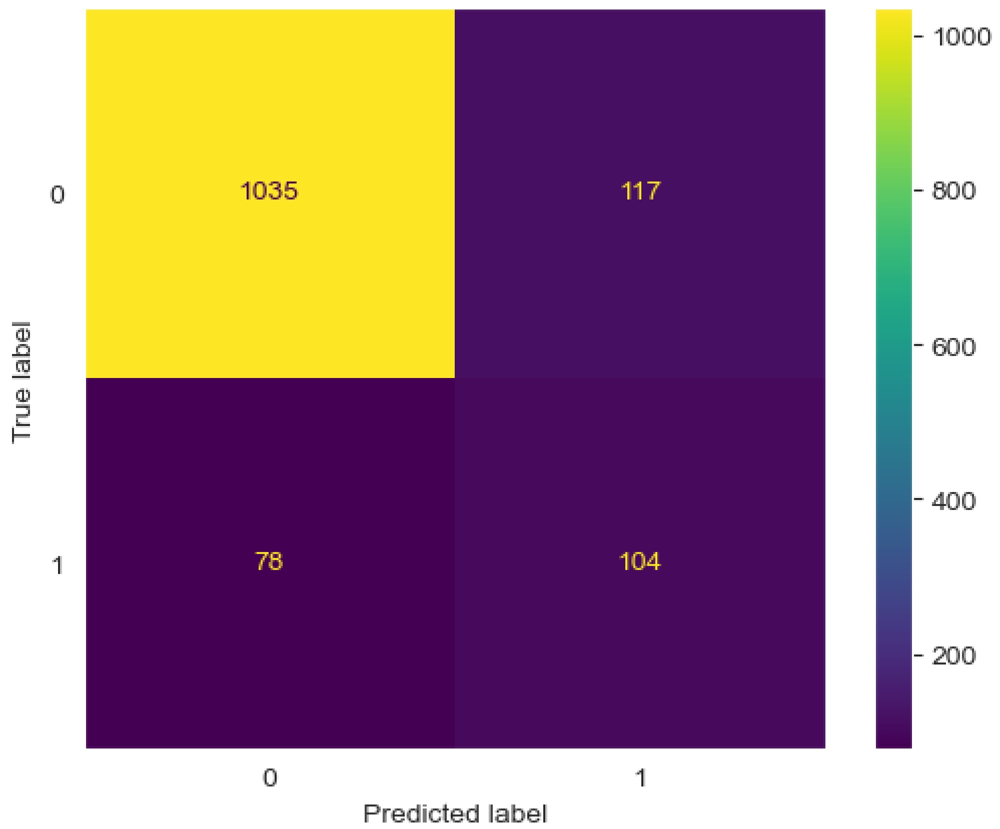
With Decision Tree classifier, we have been able to improve the metrics as shown below

- accuracy improves from 69% to 85%
- auc score improves from 70% to 73%

In [162]:

```
# print metrics
clas_score(y_test, y_h_test)

# plot confusion matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_h_test);
```

```
precision: 0.471
recall: 0.571
accuracy: 0.854
AUC: 0.735
```



In [163]:

```
gridsearch.best_params_
```

Out[163]:

```
{'tree__max_depth': None, 'tree__min_samples_split': 5}
```

Conclusion and Recommendations

We therefore conclude that a decision tree model of `max_depth` None and `min_samples_split` 5 is the better model at predicting churn

Areas of further investigation include:

- trying other models like ensemble methods
- further tuning of the model
- applying dimensionality reduction to engineer correlated features

In [ ]:

```

```