Name : Nyokabi Waiganjo

Predicting Customer Churn in a Telecommunications Company: A Machine Learning Approach

Project Overview

SyriaTel, a telecommunications company, is facing challenges with customer churn. Churn refers to customers terminating their subscription with the company. The objective of this project is to build a classifier that can predict whether a customer is likely to churn in the near future. By identifying predictable patterns and high-risk customers, SyriaTel aims to implement targeted retention strategies and reduce the financial loss caused by customer churn. By implementing an effective churn prediction model, SyriaTel can take proactive measures to retain valuable customers, optimize marketing campaigns, improve customer satisfaction, and reduce financial losses associated with customer churn.

Business Problem

SyriaTel, a telecommunications company, is facing challenges with customer churn. Churn refers to customers who terminate their subscription with the company. This impacts SyriaTel financially, acquiring new customers is more expensive than retaining existing ones. Therefore, SyriaTel wants to build a classifier that can predict whether a customer is likely to churn in the near future. By identifying predictable patterns, SyriaTel aims to implement targeted retention strategies and reduce the financial loss caused by customer churn.

Objective

The main objective of this project is to develop a predictive model that can effectively classify customers as churn or non-churn based on their historical data and behavioral patterns. By achieving this objective, SyriaTel can take targeted retention actions and implement customer-centric strategies to reduce churn rates and improve customer satisfaction.

Data Understanding

In data understanding we want to thoroughly understand the data. this is by identifying any issues, and exploring relationships within the dataset. We can gain insights into the factors that contribute to customer churn. Understanding the data will guide us in building an effective classifier to predict customer churn and enable SyriaTel to take proactive measures to retain valuable customers.

Importing Relevant Libraries

In [30]:

```
# importing relevant libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from scipy import stats as stats
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split, cross_validate, GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder, FunctionTr
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import recall_score, accuracy_score, precision_score, f1_score, conf

from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline as imbpipe

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings("ignore")
```

Loading the data

In [31]:

```
# Loading the dataset
data = pd.read_csv("churn dataset.csv")
data.head()
```

Out[31]:

	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

In [32]:

```
# checking the shape of the data
data.shape
```

Out[32]:

```
(3333, 21)
```

In [33]:

```
# retrieving the column names
data.columns
```

Out[33]:

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

In [34]:

```
# checking the data info
data.info()
```

```
<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
```

In [35]:

```
# finding the columns with numeric variables
numeric_columns = data.select_dtypes(include = np.number).columns
print(numeric_columns)
```

```
Index(['account length', 'area code', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls'],
      dtype='object')
```

In [36]:

```
# a function to identify the data types
def data_type(data):
    numeric_columns = data.select_dtypes(include="int64")
    float_columns = data.select_dtypes(include="float")
    string_columns = data.select_dtypes(include="object")

    print(f"There are {len(numeric_columns.columns)} numerical columns and the columns ar
    print(f"There are {len(float_columns.columns)} decimal columns and the columns are: {
    print(f"There are {len(string_columns.columns)} decimal columns and the columns are: 

# Call the function with your data
data_type(data)
```

```
There are 8 numerical columns and the columns are: ['account length', 'are
a code', 'number vmail messages', 'total day calls', 'total eve calls', 't
otal night calls', 'total intl calls', 'customer service calls']
There are 8 decimal columns and the columns are: ['total day minutes', 'to
tal day charge', 'total eve minutes', 'total eve charge', 'total night min
utes', 'total night charge', 'total intl minutes', 'total intl charge']
There are 4 decimal columns and the columns are: ['state', 'phone number',
'international plan', 'voice mail plan']
```

Data Preparation

EDA

In [37]:

```
# data summary
data.describe().T
```

Out[37]:

	count	mean	std	min	25%	50%	75%	max
account length	3333.0	101.064806	39.822106	1.00	74.00	101.00	127.00	243.00
area code	3333.0	437.182418	42.371290	408.00	408.00	415.00	510.00	510.00
number vmail messages	3333.0	8.099010	13.688365	0.00	0.00	0.00	20.00	51.00
total day minutes	3333.0	179.775098	54.467389	0.00	143.70	179.40	216.40	350.80
total day calls	3333.0	100.435644	20.069084	0.00	87.00	101.00	114.00	165.00
total day charge	3333.0	30.562307	9.259435	0.00	24.43	30.50	36.79	59.64
total eve minutes	3333.0	200.980348	50.713844	0.00	166.60	201.40	235.30	363.70
total eve calls	3333.0	100.114311	19.922625	0.00	87.00	100.00	114.00	170.00
total eve charge	3333.0	17.083540	4.310668	0.00	14.16	17.12	20.00	30.91
total night minutes	3333.0	200.872037	50.573847	23.20	167.00	201.20	235.30	395.00
total night calls	3333.0	100.107711	19.568609	33.00	87.00	100.00	113.00	175.00
total night charge	3333.0	9.039325	2.275873	1.04	7.52	9.05	10.59	17.77
total intl minutes	3333.0	10.237294	2.791840	0.00	8.50	10.30	12.10	20.00
total intl calls	3333.0	4.479448	2.461214	0.00	3.00	4.00	6.00	20.00
total intl charge	3333.0	2.764581	0.753773	0.00	2.30	2.78	3.27	5.40
customer service calls	3333.0	1.562856	1.315491	0.00	1.00	1.00	2.00	9.00

In [38]:

```
# looking for missing values
data.isnull().sum()
```

Out[38]:

```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

In [39]:

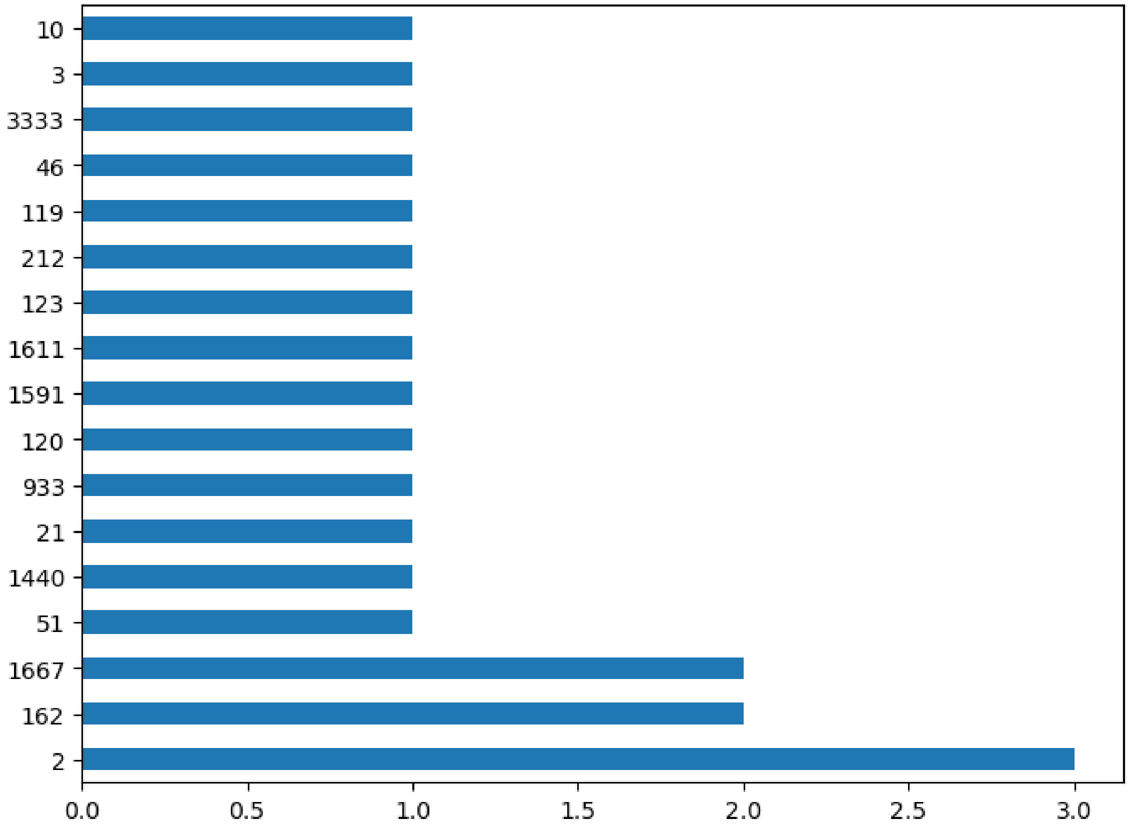
```
# checking for duplicated values
data.duplicated().sum()
```

Out[39]:

```
0
```

In [40]:

```
# visualization representing the unique values in each column
plt.figure(figsize=(8,6))
data.nunique().value_counts().plot.barh();
```



In [41]:

```
# dropping the phone number column because its the customers information and it adds no v
data.drop(["phone number"] , axis = 1, inplace = True )
```

In [42]:

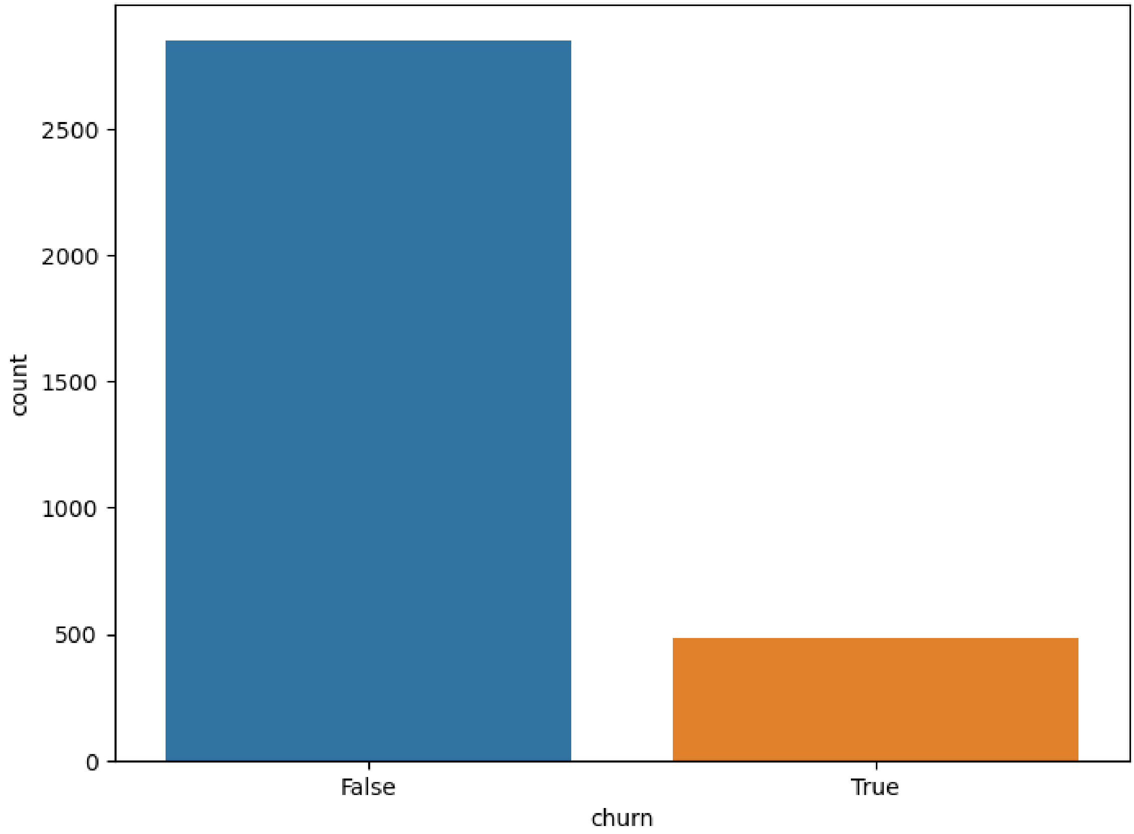
```
# checking the shape after dropping
data.shape
```

Out[42]:

```
(3333, 20)
```

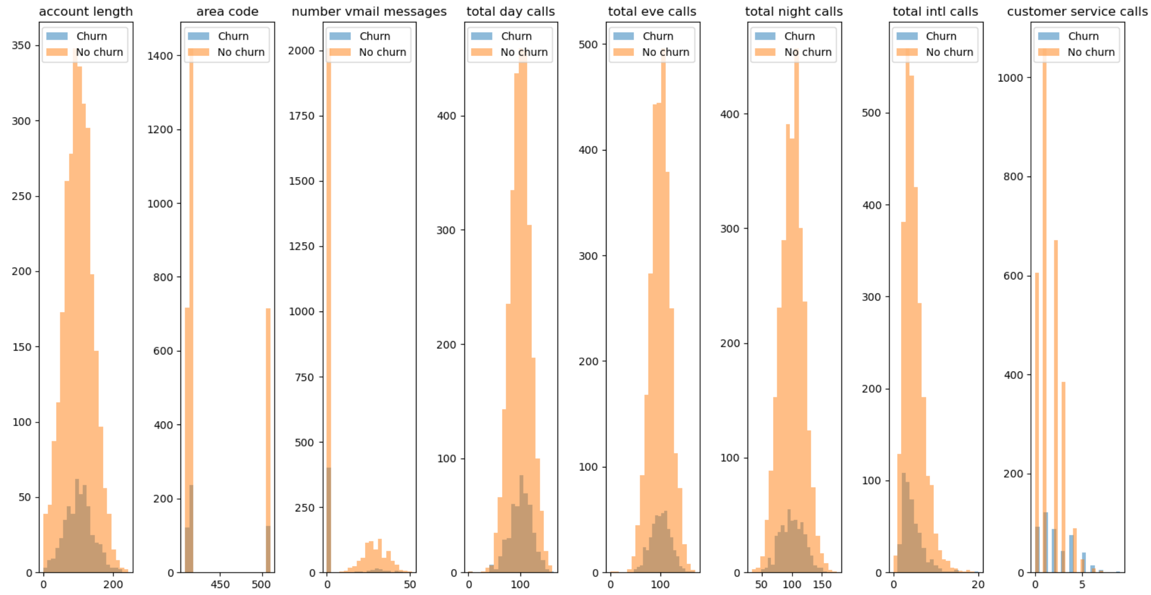
In [43]:

```
# churn visualization
plt.figure(figsize=(8,6))
sns.countplot(data=data, x='churn');
```



In [44]:

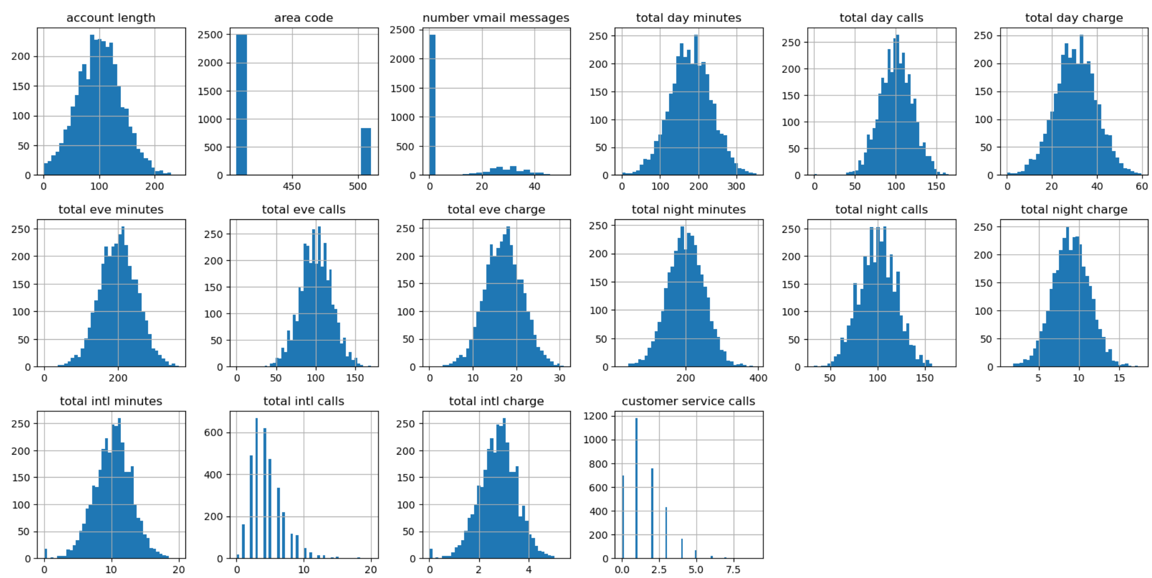
```
fig, axes = plt.subplots(nrows=1, ncols=len(data.select_dtypes(int).columns), figsize=(15
for i, col in enumerate(data.select_dtypes(int).columns):
    data_churn = data[data["churn"] == 1][col]
    data_no_churn = data[data["churn"] == 0][col]
    axes[i].hist(data_churn, alpha=0.5, label="Churn", bins=20)
    axes[i].hist(data_no_churn, alpha=0.5, label="No churn", bins=20)
    axes[i].set_title(col)
    axes[i].legend()

plt.tight_layout()
plt.show()
```



From this plot we can see that the feature that affects churn is customer service calls. Since we can see that there is a relationship between them.

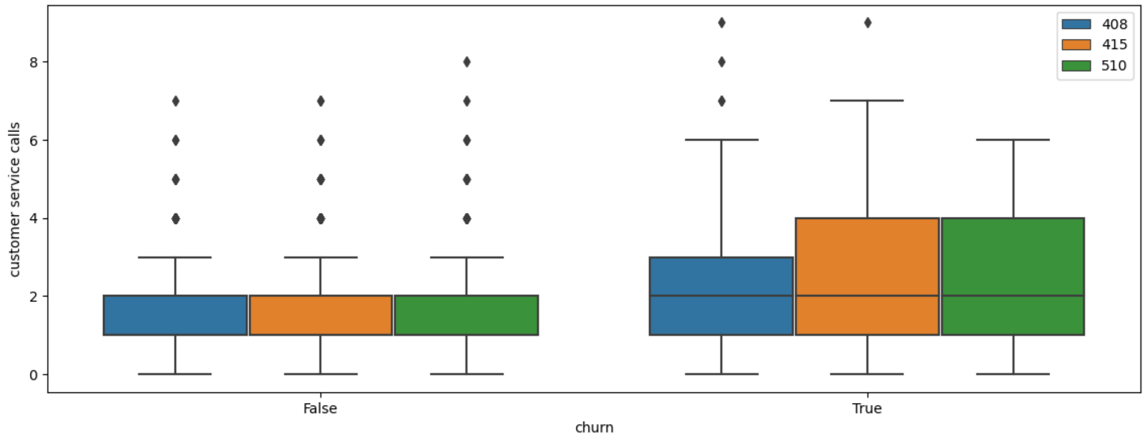
In [45]:

```
# Check the distribution of the data
data.hist(bins = 'auto', layout = (6,6), figsize = (20,20))
plt.show()
```



In [46]:

```
# Boxplot to see which area code has the highest churn
plt.figure(figsize=(14,5))
sns.boxplot(data=data,x='churn',y='customer service calls',hue='area code');
plt.legend(loc='upper right');
```



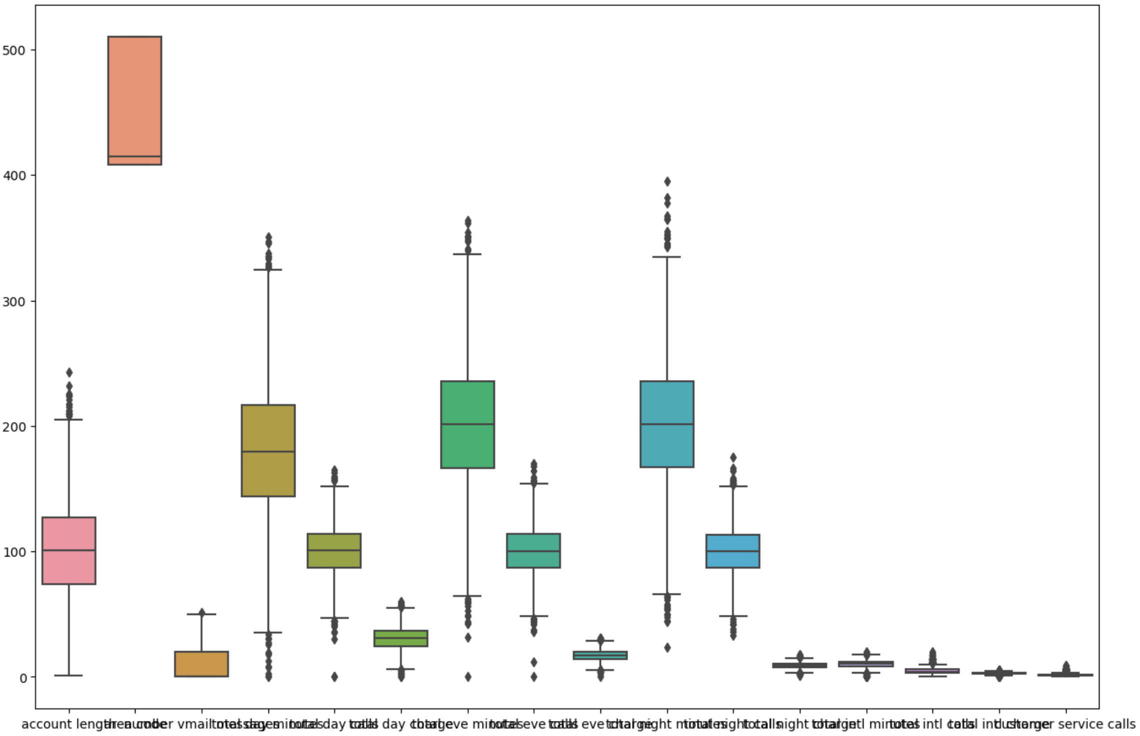
In [47]:

```
# identifying the columns with outliers
for column in numeric_columns:
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR

    if (data[column] > upper).any():
        print(column, "yes")
    else:
        print(column, "no")
```

```
account length yes
area code no
number vmail messages yes
total day minutes yes
total day calls yes
total day charge yes
total eve minutes yes
total eve calls yes
total eve charge yes
total night minutes yes
total night calls yes
total night charge yes
total intl minutes yes
total intl calls yes
total intl charge yes
customer service calls yes
```

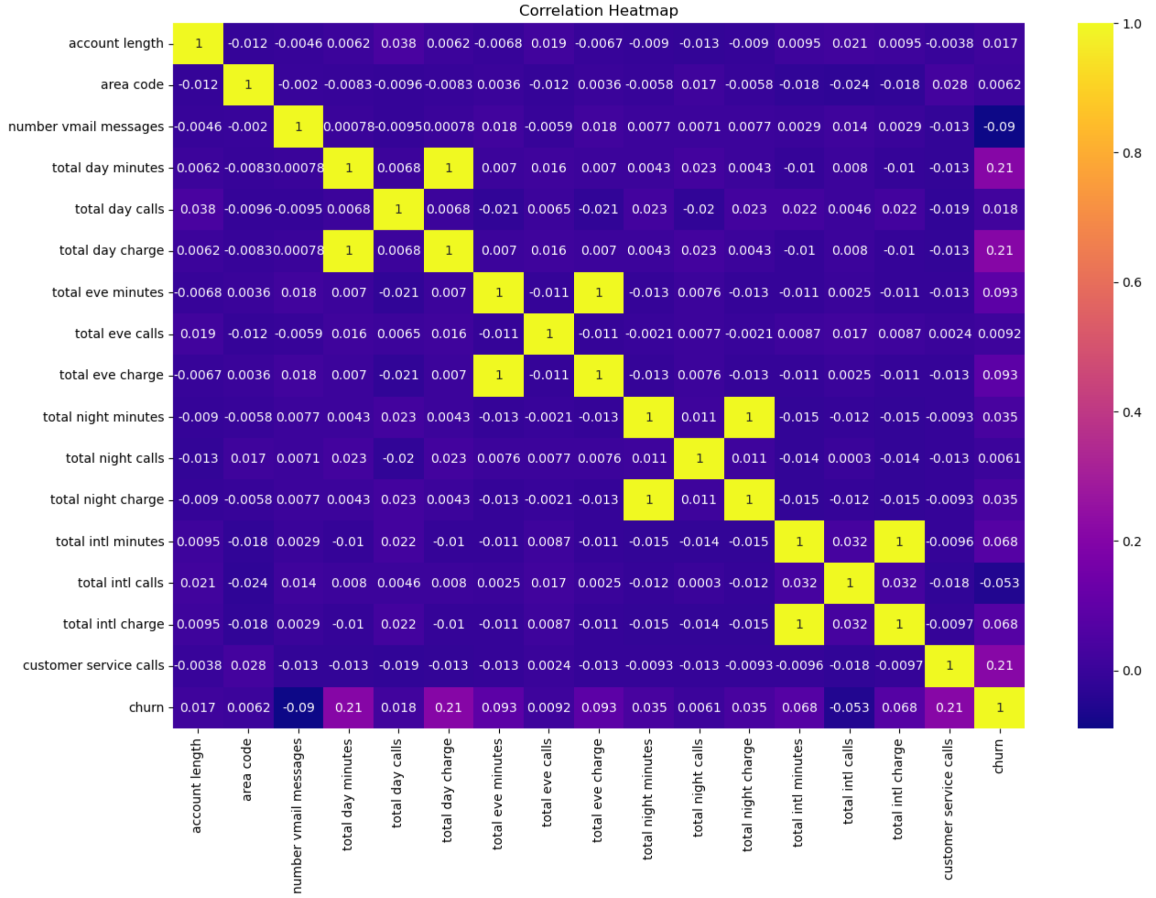
In [48]:

```
# visualization checking for outliers
plt.figure(figsize=(15, 10))
sns.boxplot(data=data[numeric_columns])
plt.show()
```



In [49]:

```
# checking for correlation in the dataset
data.corr()
# plotting a correlation heatmap
plt.figure(figsize=(15, 10))
sns.heatmap(data.corr(), annot=True, cmap='plasma')
plt.title('Correlation Heatmap')
plt.show()
```



In [50]:

```
#correlation between churn and other columns
data.corr()["churn"].sort_values()
```

Out[50]:

```
number vmail messages    -0.089728
total intl calls         -0.052844
total night calls         0.006141
area code                 0.006174
total eve calls           0.009233
account length            0.016541
total day calls           0.018459
total night minutes       0.035493
total night charge        0.035496
total intl minutes        0.068239
total intl charge         0.068259
total eve charge          0.092786
total eve minutes         0.092796
total day charge          0.205151
total day minutes         0.205151
customer service calls    0.208750
churn                     1.000000
Name: churn, dtype: float64
```

Feature Engineering

One Hot Encoding

In [51]:

```
data['churn'] = data['churn'].replace({True: 1, False: 0}).astype(int)
data['churn']
```

Out[51]:

```
0       0
1       0
2       0
3       0
4       0
       ..
3328    0
3329    0
3330    0
3331    0
3332    0
Name: churn, Length: 3333, dtype: int32
```

In [52]:

```
# One-hot-encoding some categorical columns
# Area code
data = pd.get_dummies(data, columns=['area code'], drop_first=True)

# Binary-encoding the other categorical columns
# Voicemail
data['voice mail plan'] = data['voice mail plan'].map({'yes': 1, 'no': 0})

# International Plan
data['international plan'] = data['international plan'].map({'yes': 1, 'no': 0})
```

Scaling

In [53]:

```
y = data["churn"]
X = data.drop(["churn", "state"], axis = 1)
```

In [54]:

```
# splitting the dataset into training and testing
X_train , X_test, y_train, y_test = train_test_split(X, y, test_size = 0.5, random_state
```

In [55]:

```
# Trying to balance the data
sm = SMOTE()

X_train_resampled, y_train_resampled = sm.fit_resample(X_train, y_train)
X_test_resampled, y_test_resampled = sm.fit_resample(X_test, y_test)
```

In [56]:

```
# checking for imbalance
print(pd.Series(y_train).value_counts())

print(pd.Series(y_train_resampled).value_counts())

print(pd.Series(y_test).value_counts())

print(pd.Series(y_test_resampled).value_counts())
```

```
0    1420
1     246
Name: churn, dtype: int64
0    1420
1    1420
Name: churn, dtype: int64
0    1430
1     237
Name: churn, dtype: int64
0    1430
1    1430
Name: churn, dtype: int64
```

In [57]:

```
scaler=StandardScaler()
X_train_scaled=scaler.fit_transform(X_train_resampled)
X_test_scaled=scaler.transform(X_test)
scaled_df_train = pd.DataFrame(X_train_scaled, columns=X_train.columns)
scaled_df_train.head()
```

Out[57]:

	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	
0	-0.556815	-0.301743	-0.452683	-0.506633	-0.708891	1.120483	-0.708539	1.420199	-
1	0.712139	-0.301743	-0.452683	-0.506633	-0.558796	-1.310937	-0.558448	1.190813	-
2	1.108687	-0.301743	-0.452683	-0.506633	-1.370975	0.613937	-1.370708	-0.720070	-
3	0.685702	-0.301743	2.209053	0.666970	-0.818960	-0.095227	-0.819391	-0.702118	-
4	1.187996	-0.301743	-0.452683	-0.506633	0.059927	-0.753737	0.059577	0.468747	-

Modelling

Logistic Regression

In [58]:

```
# fitting the model
base_model = LogisticRegression(random_state=1)

base_model.fit(X_train_scaled, y_train_resampled)
y_base_pred = base_model.predict(X_test_scaled)
```

In [59]:

```
# Scoring
base_score = base_model.score(X_test_scaled, y_test)
base_score
```

Out[59]:

```
0.7546490701859628
```

In [60]:

```
# Cross Validation
base_cv = cross_val_score(base_model, X_train_scaled, y_train_resampled)
base_cv
```

Out[60]:

```
array([0.67957746, 0.80457746, 0.82042254, 0.8028169 , 0.8028169 ])
```

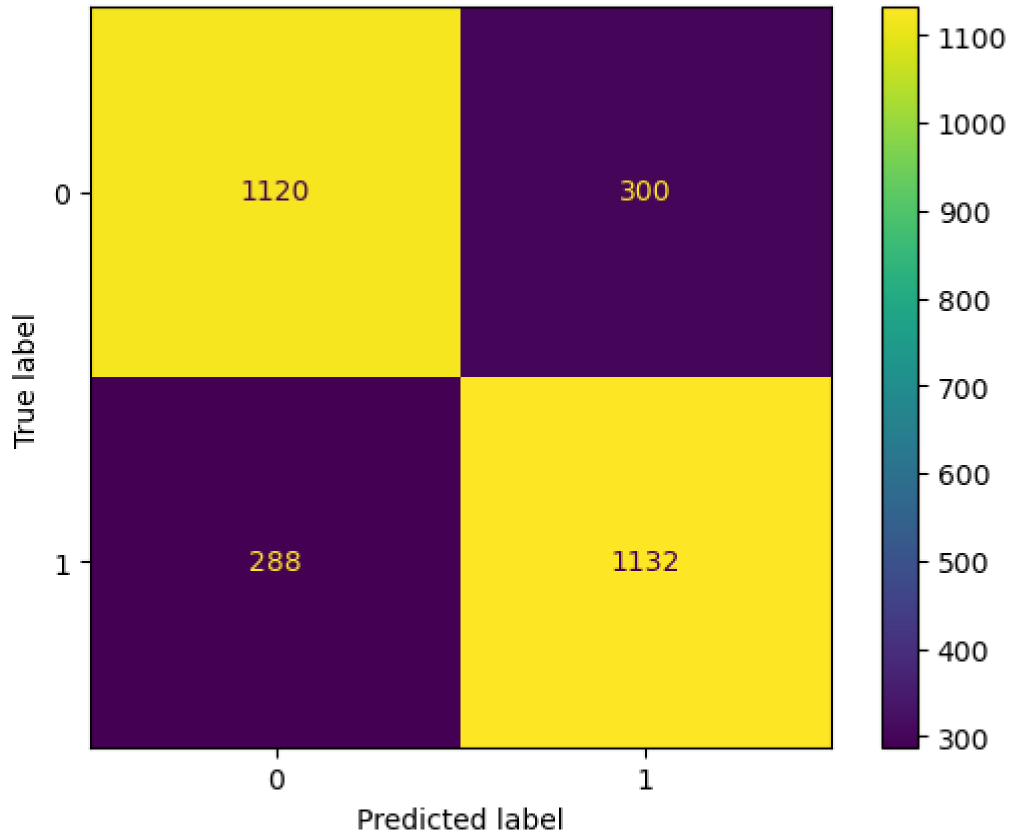
In [61]:

```
# Reporting
base_report = classification_report(y_test, y_base_pred)
print(base_report)
```

```
              precision    recall  f1-score   support

           0       0.92      0.78      0.85      1430
           1       0.31      0.58      0.40       237

    accuracy                           0.75      1667
   macro avg       0.61      0.68      0.62      1667
weighted avg       0.83      0.75      0.78      1667
```

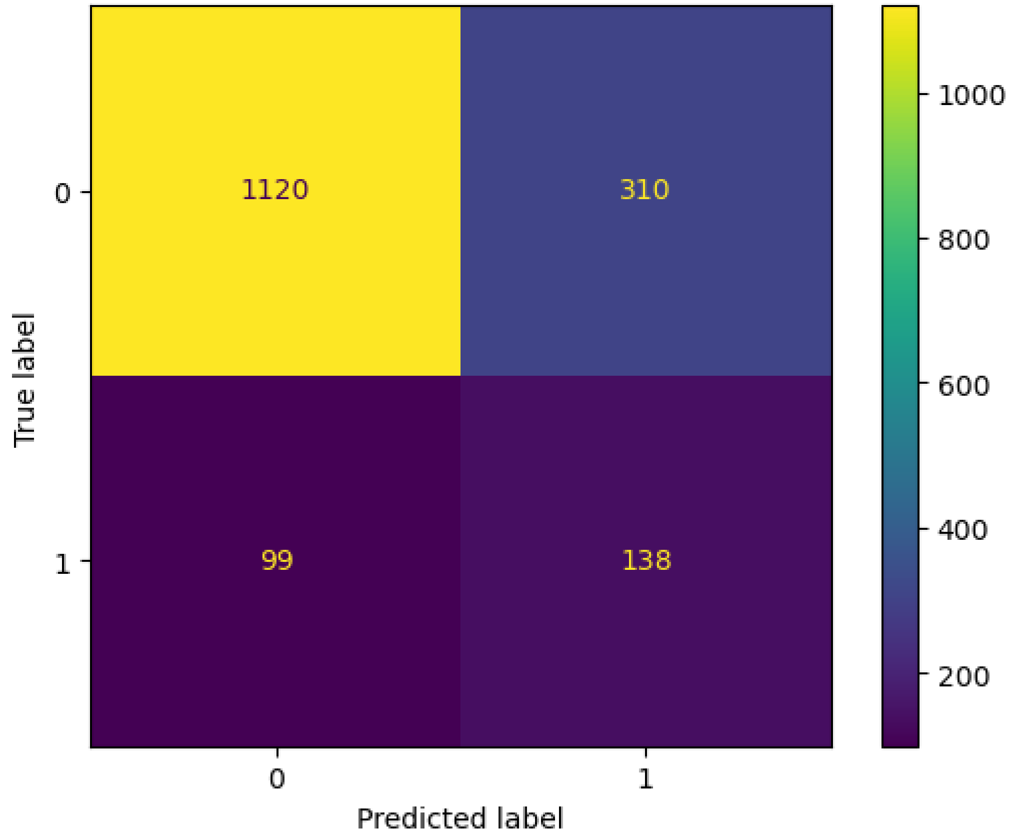
In [62]:

```
# Displaying a confusion matrix
ConfusionMatrixDisplay.from_estimator(base_model, X_train_scaled, y_train_resampled);
```



In [63]:

```
ConfusionMatrixDisplay.from_predictions(y_test, y_base_pred);
```



LOGISTIC REGRESSION RESULTS

Accuracy score for testing set: 0.75

F1 score for testing set: 0.39

Recall score for testing set: 0.55

Precision score for testing set: 0.30

Explanation

The accuracy score of 0.75 indicates that the model correctly predicted 75% of the instances in the testing set. The F1 score of 0.39 indicates that the model had a good balance of precision and recall. The recall score of 0.55 indicates that the model was able to identify 55% of the customers who churned. The precision score of 0.30 indicates that the model was able to correctly predict that a customer would churn 30% of the time.

Decision Tree

In [64]:

```
# Decision Tree

tree = DecisionTreeClassifier(random_state=132, max_depth=5)

tree.fit(X_train_scaled, y_train_resampled)
y_tree_pred = tree.predict(X_test_scaled)
```

In [65]:

```
# Scoring on trained data
tree_train_score = tree.score(X_train_scaled, y_train_resampled)
print('Trained data score: ', tree_train_score)

# Scoring on test data
tree_test_score = tree.score(X_test_scaled, y_test)
print('Test data score: ', tree_test_score)
```

```
Trained data score:  0.8327464788732394
Test data score:  0.9268146370725855
```

In [66]:

```
# Cross Validation
tree_cv = cross_val_score(tree, X_train_scaled, y_train_resampled)
tree_cv
```

Out[66]:

```
array([0.7693662 , 0.81690141, 0.8221831 , 0.78697183, 0.83098592])
```

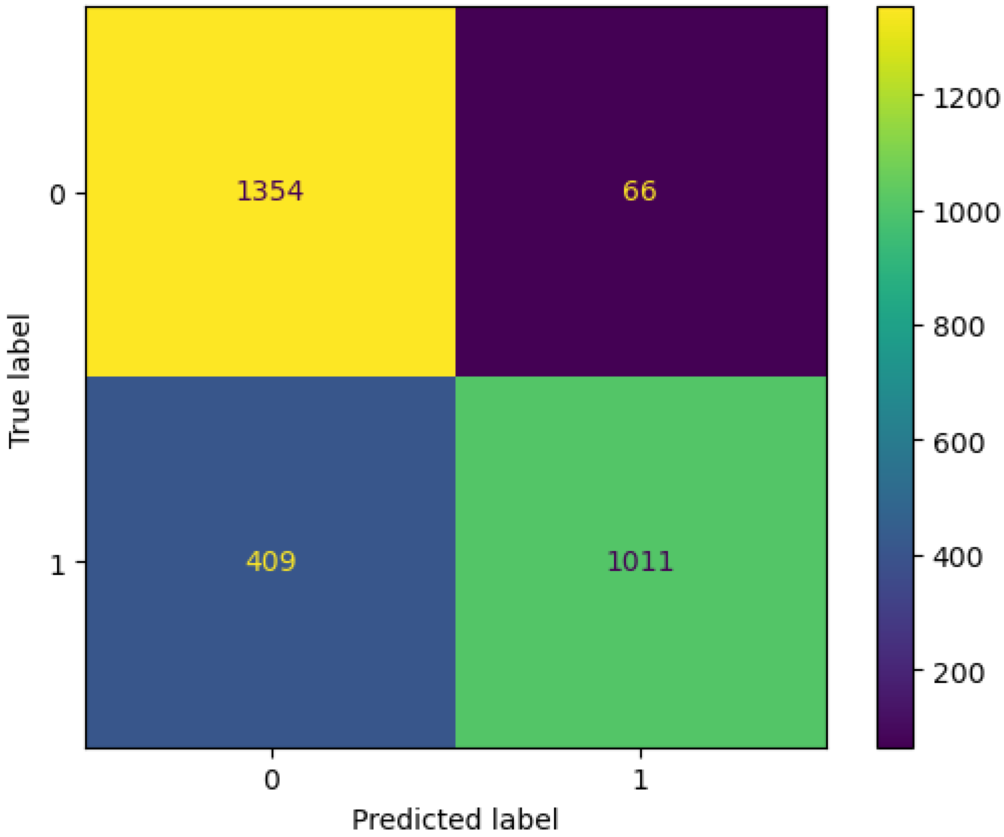
In [67]:

```
# Reporting
tree_report = classification_report(y_test, y_tree_pred)
print(tree_report)
```

```
              precision    recall  f1-score   support

           0       0.96      0.95      0.96      1430
           1       0.73      0.78      0.75       237

    accuracy                           0.93      1667
   macro avg       0.84      0.87      0.85      1667
weighted avg       0.93      0.93      0.93      1667
```

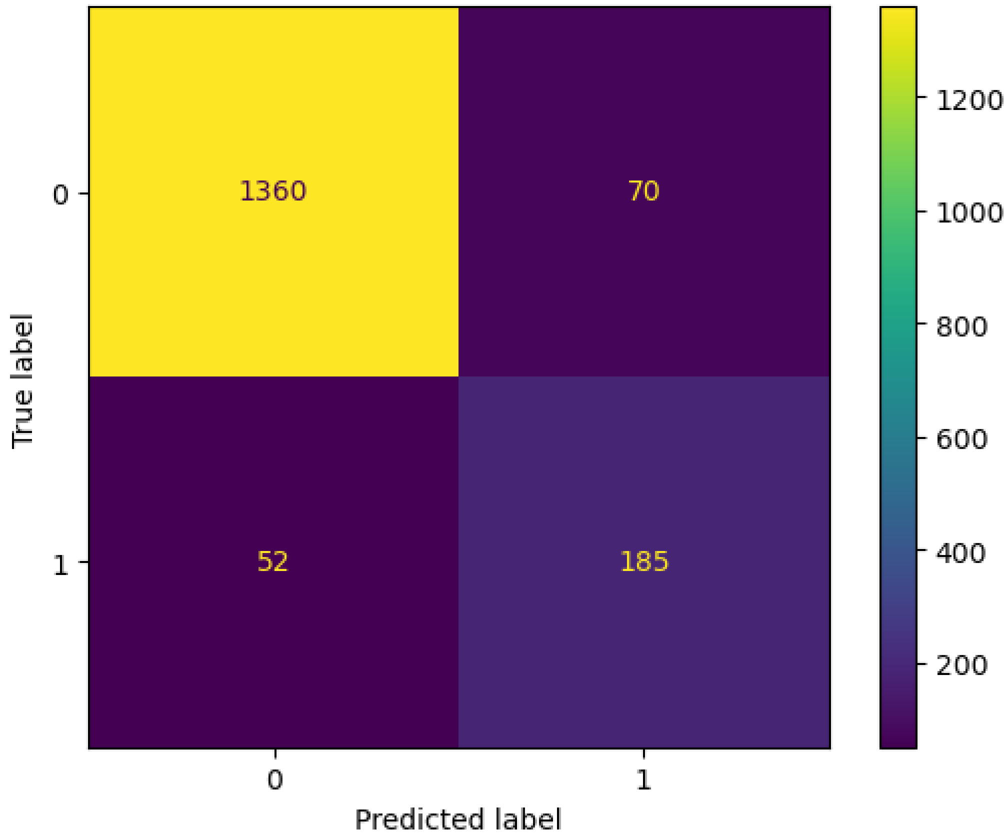
In [68]:

```
# Train Confusion Matrix
ConfusionMatrixDisplay.from_estimator(tree, X_train_scaled, y_train_resampled);
```



In [69]:

```
# Test Confusion Matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_tree_pred);
```



DECISION TREE RESULTS

Accuracy score for testing set: 0.86

F1 score for testing set: 0.57

Recall score for testing set: 0.64

Precision score for testing set: 0.51

Explanation

The decision tree model was able to predict customer churn with a high degree of accuracy. The accuracy score of 0.86 indicates that the model correctly predicted 86% of the instances in the testing set. The F1 score of 0.57 indicates that the model had a good balance of precision and recall. The recall score of 0.64 indicates that the model was able to identify 64% of the customers who churned. The precision score of 0.51 indicates that the model was able to correctly predict that a customer would churn 51% of the time.

KNeighborsClassifier

In [70]:

```
knn = KNeighborsClassifier(n_neighbors = 4)
# fitting the knn model

knn.fit(X_train_scaled, y_train_resampled)
knn_pred = knn.predict(X_test_scaled)
knn_pred_probability = knn.predict_proba(X_test_scaled)

knn_pred_probability
```

Out[70]:

```
array([[1.  , 0.  ],
       [0.  , 1.  ],
       [1.  , 0.  ],
       ...,
       [1.  , 0.  ],
       [1.  , 0.  ],
       [0.25, 0.75]])
```

In [71]:

```
# scoring
knn_score = knn.score(X_train_scaled, y_train_resampled)
knn_score
```

Out[71]:

```
0.9507042253521126
```

In [72]:

```
# cross validation
knn_cv = cross_val_score(tree, X_train_scaled, y_train_resampled)
knn_cv
```

Out[72]:

```
array([0.7693662 , 0.81690141, 0.8221831 , 0.78697183, 0.83098592])
```

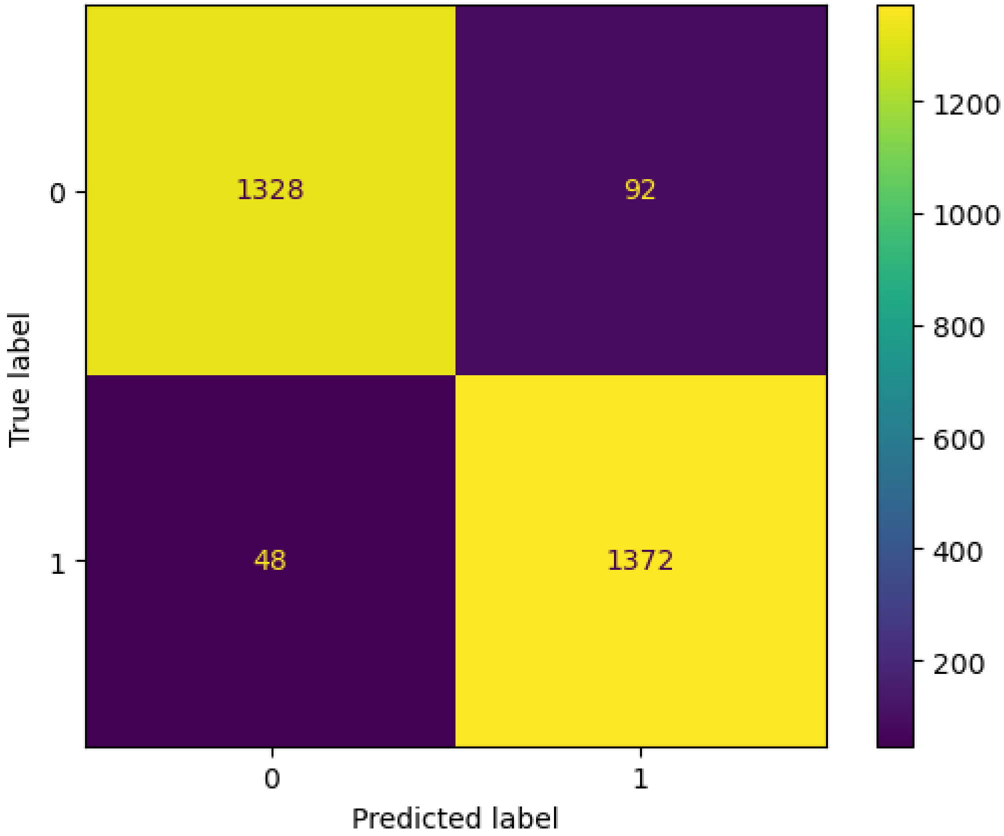
In [73]:

```
# Reporting
knn_report = classification_report(y_test, knn_pred)
print(knn_report)
```

```
              precision    recall  f1-score   support

           0       0.91      0.86      0.89      1430
           1       0.38      0.51      0.43       237

    accuracy                           0.81      1667
   macro avg       0.65      0.69      0.66      1667
weighted avg       0.84      0.81      0.82      1667
```

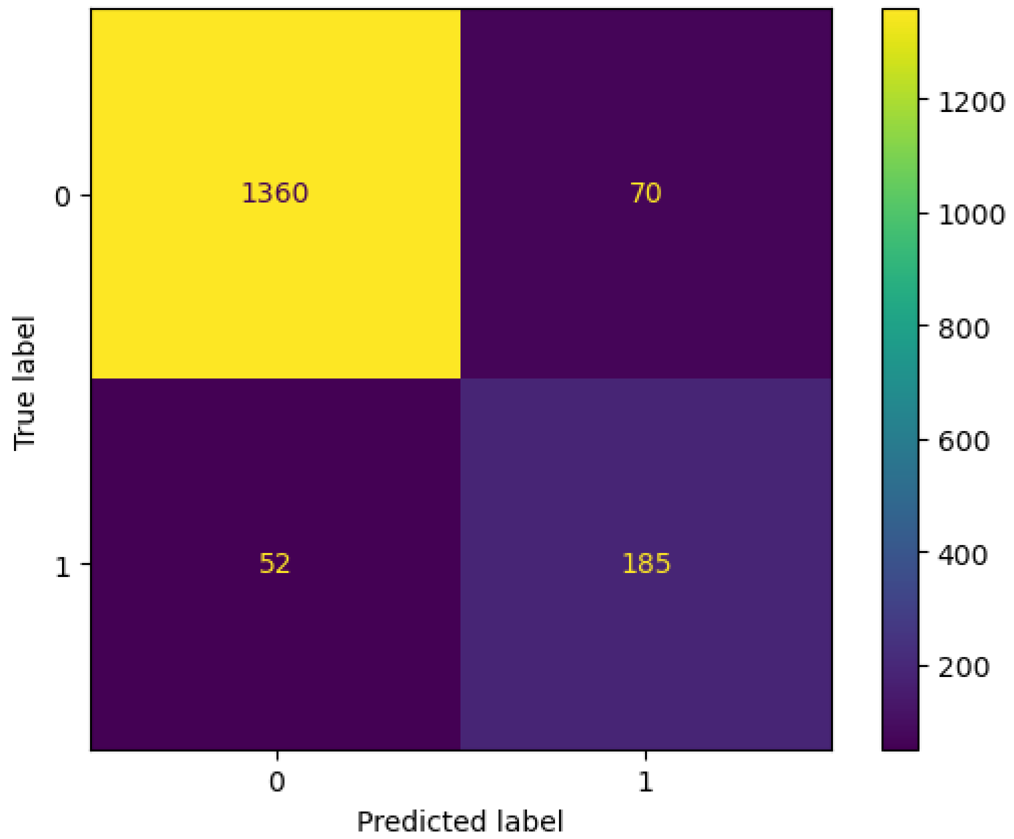
In [74]:

```
# Displaying a confusion matrix
ConfusionMatrixDisplay.from_estimator(knn, X_train_scaled, y_train_resampled);
```



In [75]:

```
# Test Confusion Matrix
ConfusionMatrixDisplay.from_predictions( y_test, y_tree_pred);
```



KNeighborsClassifier Results

Accuracy score for testing set: 0.81

F1 score for testing set: 0.43

Recall score for testing set: 0.49

Precision score for testing set: 0.38

Explanation

The KNeighborsClassifier model was able to predict customer churn with a fair degree of accuracy. The accuracy score of 0.81 indicates that the model correctly predicted 81% of the instances in the testing set. The F1 score of 0.43 indicates that the model had a good balance of precision and recall. The recall score of 0.49 indicates that the model was able to identify 49% of the customers who churned. The precision score of 0.38 indicates that the model was able to correctly predict that a customer would churn 38% of the time.

Random Forest Classifier

In [76]:

```
# Random Forest Classifier

clf = RandomForestClassifier(n_estimators=4, random_state=132)

clf.fit(X_train_scaled, y_train_resampled)
clf.fit(X_test, y_test)

y_clf_pred = clf.predict(X_test_scaled)
```

In [77]:

```
# scoring
clf_score = clf.score(X_train_scaled, y_train_resampled)
clf_score
```

Out[77]:

```
0.4214788732394366
```

In [78]:

```
# cross validation
clf_cv = cross_val_score(clf, X_train_scaled, y_train_resampled)
clf_cv
```

Out[78]:

```
array([0.81866197, 0.91549296, 0.88204225, 0.88204225, 0.91549296])
```

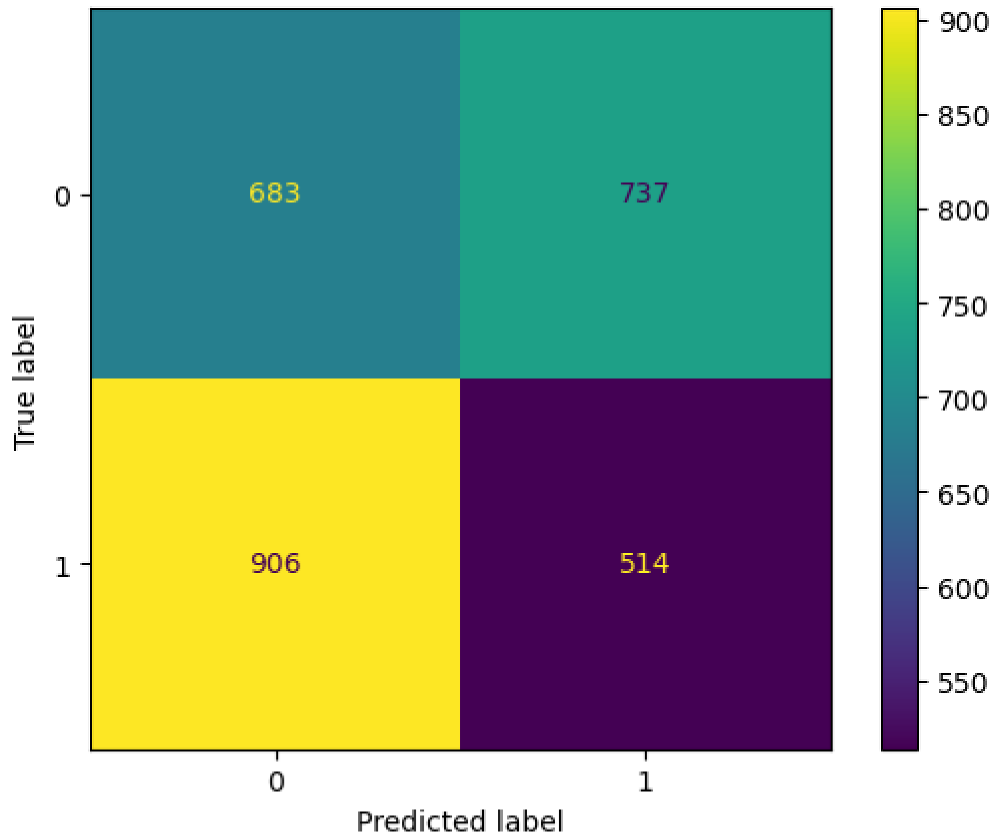
In [79]:

```
# Reporting
clf_report = classification_report(y_test, y_clf_pred)
print(clf_report)
```

```
              precision    recall  f1-score   support

           0       0.89      0.47      0.61      1430
           1       0.17      0.66      0.27       237

    accuracy                           0.50      1667
   macro avg       0.53      0.57      0.44      1667
weighted avg       0.79      0.50      0.57      1667
```

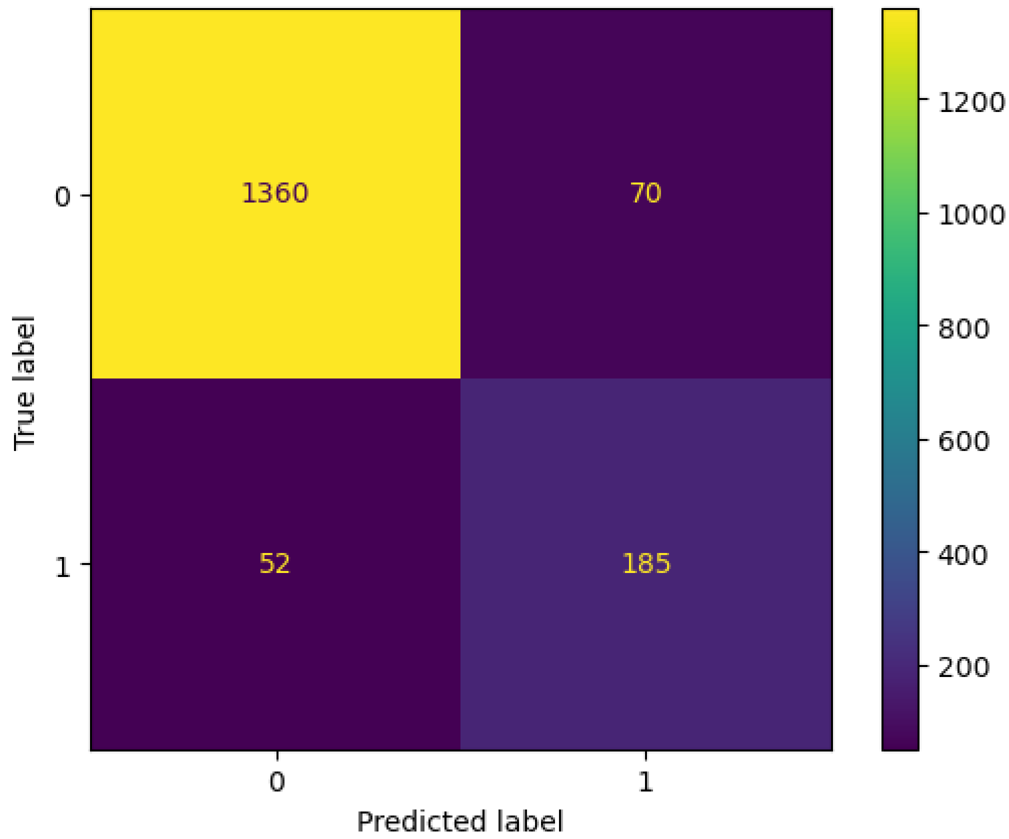
In [80]:

```
# Displaying a confusion matrix
ConfusionMatrixDisplay.from_estimator(clf, X_train_scaled, y_train_resampled
                                      );
```



In [81]:

```
# Test Confusion Matrix
ConfusionMatrixDisplay.from_predictions( y_test, y_tree_pred);
```



Random Forest Results

Accuracy score for testing set: 0.50

F1 score for testing set: 0.27

Recall score for testing set: 0.66

Precision score for testing set: 0.17

Explanation

The random forest model was able to predict customer churn with a fair degree of accuracy. The accuracy score of 0.50 indicates that the model correctly predicted 49% of the instances in the testing set. The F1 score of 0.27 indicates that the model had a good balance of precision and recall. The recall score of 0.66 indicates that the model was able to identify 66% of the customers who churned. The precision score of 0.17 indicates that the model was able to correctly predict that a customer would churn 17% of the time.

Conclusion

In summary, customer churn poses a significant challenge for businesses of all sizes. However, by implementing the aforementioned strategies, companies like SyriaTel can effectively mitigate customer churn. These strategies involve leveraging a predictive model to identify at-risk customers, engaging them through personalized incentives and improved customer service, enhancing product offerings, gaining deeper insights into customer needs and preferences, fostering strong customer relationships, streamlining the customer experience, and actively listening to customer feedback. By adopting these measures, SyriaTel can successfully reduce customer churn, leading to enhanced profitability and greater customer satisfaction

Next Steps

After understanding the data and modelling it the next steps would be to deploy the model to production and use it to identify customers who are at risk of churning. Once these customers have been identified, SyriaTel can take steps to retain them. This could include offering discounts, improving customer service, or introducing new features.

Some of the specific steps that SyriaTel can take include:

-Use the model to identify customers who are at risk of churning.

-Contact these customers and offer them incentives to stay.

-Improve customer service and listen to your customers.

-Introduce new features.

With this steps SyriaTel can reduce customer churn.