N-kioko /
Phase-3-project

Code Issues Pull requests Actions Projects Wiki Security

Phase-3-project / index.ipynb

N-kioko Updated the Powerpont Presentation c118c4e · 1 minute ago

1235 lines (1235 loc) · 281 KB

Preview Code Blame Raw

TABLE OF CONTENTS

1. INTRODUCTION

In the telecommunications industry , alot of factors contribute to customer churn. Some of the factors include, quality of service provison, customer service effectivceness, and competition from service providers. Employing predictive models such as logistice regression and decission trees can enhance the sevice providers to easily anticipate churn and put in place measures to conteract loss of clients.

2. BUSINESS PROBLEM

Customer retention is one of the key factor in growth sustenance for any telecomunication firm. Therefore customer churn; a phenomena whereby a customer stops to utilize company services,stands to be one of greatest threat for any telecomunications firm. This study explores predictive analysis to explore various factors contributing to customer churn. As well as employing data-driven approach to predict likelihood of customers churn based on various characteristics.

3. OBJECTIVES

1. Understand the customer churn rate.

2. Estimate the churn effects on the company's revenue.

3. Develop a predictive model that can accurately predict the likelihood of customers churn based on their past behavior and characteristics.

4. DATA UNDERSTANDING

The dataset used in this study is from Kaggle, https://www.kaggle.com/datasets/mnassrib/telecom-churn-datasets. The dataset contains 3333 observations with 20 variables. The variables include:

The variables include:

State: Categorical variable indicating the customer's state.

Account length: Numeric variable indicating the length of the customer account.

Area code: Numeric variable indicating the area code of the

Phone number: Categorical variable (likely to be excluded as it won't contribute to churn prediction).

International plan: Categorical variable indicating if the customer has an international plan.

Voice mail plan: Categorical variable indicating if the customer has a voicemail plan.

Number vmail messages: Numeric variable indicating the number of voicemail messages.

Total day/eve/night/intl minutes: Numeric variables indicating usage minutes in various time segments.

Total day/eve/night/intl calls: Numeric variables indicating the number of calls in various time segments.

Total night minutes: Numeric variables indicating usage minutes in various time segments.

Total day/eve/night/intl charge: Numeric variables indicating charges in various time segments.

Customer service calls: Numeric variable indicating the number of customer service calls made by the customer.

Churn: Binary target variable indicating customer churn (True/False).

The variables in the dataset will enable us to explore connection between various customer preferences/behaviours and churn.

5. DATA PREPARATION

Importing Packages

In [21]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
import numpy as np
```

Loading Data

In [22]:

```
import pandas as pd
```

```
df= pd.read_csv("Data/bigml_59c28831336c6604c800002a.csv", index_col=0)
print(df.head())
print("--------------------")
print(df.shape)
```

```
       account length  area code phone number international plan  \
state
KS                128        415     382-4657                 no
OH                107        415     371-7191                 no
NJ                137        415     358-1921                 no
OH                 84        408     375-9999                yes
OK                 75        415     330-6626                yes

      voice mail plan  number vmail messages  total day minutes  \
state
KS                yes                     25              265.1
OH                yes                     26              161.6
NJ                 no                      0              243.4
OH                 no                      0              299.4
OK                 no                      0              166.7

       total day calls  total day charge  total eve minutes  total eve call
s  \
state
KS                 110             45.07              197.4               9
9
OH                 123             27.47              195.5              10
3
NJ                 114             41.38              121.2              11
0
OH                  71             50.90               61.9               8
8
OK                 113             28.34              148.3              12
2

       total eve charge  total night minutes  total night calls  \
state
KS                16.78                244.7                 91
OH                16.62                254.4                103
NJ                10.30                162.6                104
OH                 5.26                196.9                 89
OK                12.61                186.9                121

       total night charge  total intl minutes  total intl calls  \
state
KS                  11.01                10.0                 3
OH                  11.45                13.7                 3
NJ                   7.32                12.2                 5
OH                   8.86                 6.6                 7
OK                   8.41                10.1                 3

       total intl charge  customer service calls  churn
state
KS                  2.70                       1  False
OH                  3.70                       1  False
NJ                  3.29                       0  False
OH                  1.78                       2  False
OK                  2.73                       3  False
--------------------
(3333, 20)
```

In [23]:

```
#drop the hone number column
df.drop("phone number",axis=1,inplace=True)
```

Checking for Missing values

The data set contains no missing values

In [24]:
```
df.isna().sum()
```

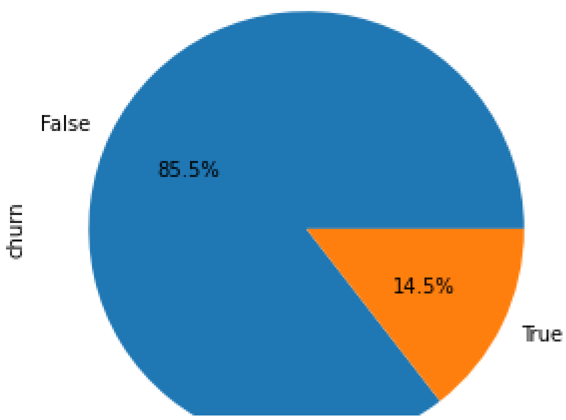
Out[24]:
```
account length            0
area code                 0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve 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
```

6. EXPLORATORY DATA ANALYSIS

Exploring Churn rates

In [25]:
```
Total_entries = df["churn"].count()
churn = df["churn"].value_counts()[1]
print(Total_entries)
print(churn)
print("--------------------------")
print('churn rate:',(churn/Total_entries)*100)
Plot = df["churn"].value_counts().plot(kind="pie",autopct="%1.1f%%",figsiz
```

```
3333
483
--------------------------
churn rate: 14.491449144914492
```



Estimating Customer Churn Impact on Revenue

In [26]:

```
#total revenue brought in by customers
df["total charge"]=df["total day charge"]+ df["total eve charge"]+ df["tot
Total_Revenue = df["total charge"].sum()
Total_revenue_per_churn_status=df["total charge"].groupby(df["churn"]).sum
print(Total_revenue_per_churn_status)
Percentage_revenue_loss = Total_revenue_per_churn_status[1]/Total_Revenue
print("----------------------------------------")
print(" With the presented churn rate of 14%, the syriatel company is bour
```

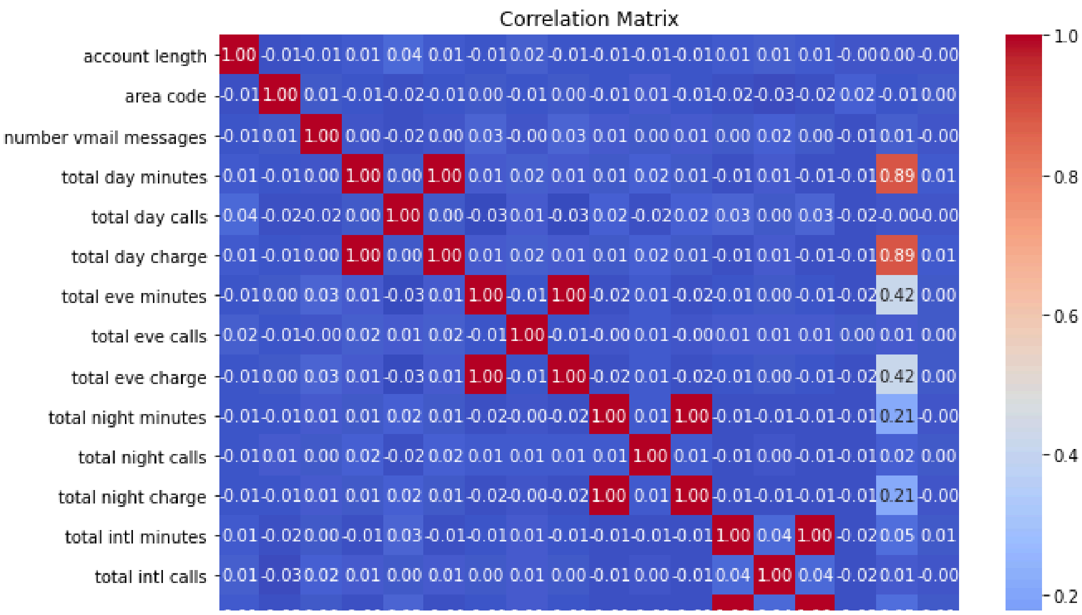
```
churn
False    166579.10
True      31566.93
Name: total charge, dtype: float64
-----------------------------------------
 With the presented churn rate of 14%, the syriatel company is bound to los
s, 15.93% of the revenue.
```

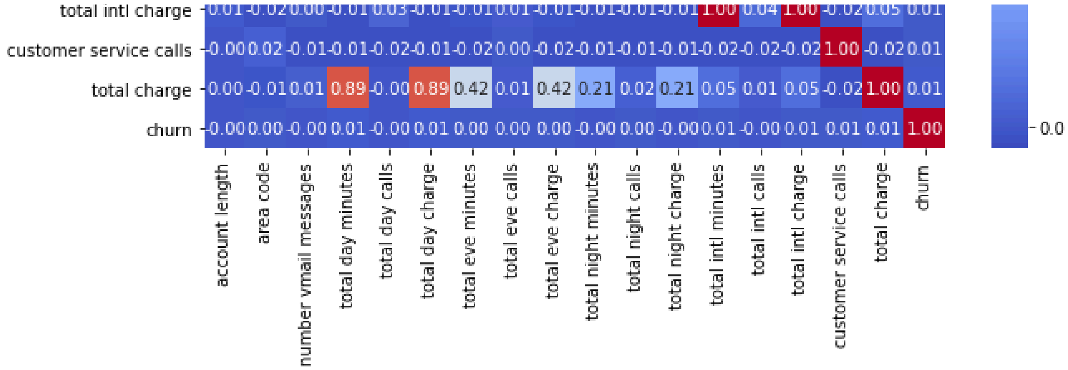
Correlation Analysis

Could there be fetures that are highly realted to the variable "Churn",a correlation Matrix in necessary just to point out features with the highest correlation.

In [27]:

```
import seaborn as sns
import matplotlib.pyplot as plt
# Compute correlation matrix
x=df.drop("churn",axis=1)
y1=df["churn"]
corr = x.join(y1).corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```





Checking Data Types

This in necessary to guide us on the best way process data for modelling.

In [28]:

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3333 entries, KS to TN
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account length          3333 non-null   int64  
 1   area code               3333 non-null   int64  
 2   international plan      3333 non-null   object 
 3   voice mail plan         3333 non-null   object 
 4   number vmail messages   3333 non-null   int64  
 5   total day minutes       3333 non-null   float64
 6   total day calls         3333 non-null   int64  
 7   total day charge        3333 non-null   float64
 8   total eve minutes       3333 non-null   float64
 9   total eve calls         3333 non-null   int64  
 10  total eve charge        3333 non-null   float64
 11  total night minutes     3333 non-null   float64
 12  total night calls       3333 non-null   int64  
 13  total night charge      3333 non-null   float64
 14  total intl minutes      3333 non-null   float64
 15  total intl calls        3333 non-null   int64  
 16  total intl charge       3333 non-null   float64
 17  customer service calls  3333 non-null   int64  
 18  churn                   3333 non-null   bool   
 19  total charge            3333 non-null   float64
dtypes: bool(1), float64(9), int64(8), object(2)
memory usage: 604.0+ KB
None
```

Defining X and Y variables

In this case Churn is the target varibales while all the other variables besides, phone_number serve as the predictors in our modelling process.

In [29]:

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

Performing test train split

This ensures that our model is trained on a representative sample of the data and evaluated on a different, unseen sample. This helps to prevent data leakage and overfitting for optimal performance of our model.

In [30]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.5,random_st
```

7. PREPROCESSING

Selecting categorical variables

In [31]:
```
X_train_cat = X_train.select_dtypes(include="object")
X_train_num = X_train.select_dtypes(exclude="object")
X_test_cat = X_test.select_dtypes(include="object")
X_test_num = X_test.select_dtypes(exclude="object")
```

One hot encoding

In [32]:
```
Ohe = OneHotEncoder(sparse_output=False, handle_unknown="ignore")
X_train_cat_ohe =Ohe.fit_transform(X_train_cat)
X_test_cat_ohe =Ohe.transform(X_test_cat)
```

Creating Dataframes

In [33]:
```
X_train_cat_ohe_df= pd.DataFrame(X_train_cat_ohe, index=X_train_cat.index,
X_test_cat_ohe_df= pd.DataFrame(X_test_cat_ohe, index=X_test_cat.index,col
X_train_cat_ohe_df.head()
X_test_cat_ohe_df.head()
```

Out[33]:

	no	yes	no	yes
state				
WY	1.0	0.0	1.0	0.0
IL	1.0	0.0	1.0	0.0
SD	1.0	0.0	1.0	0.0
KY	1.0	0.0	1.0	0.0
WY	1.0	0.0	1.0	0.0

Scaling the Numeric Variables

In [34]:
```
from sklearn.preprocessing import MinMaxScaler
Scaler= MinMaxScaler()
X_train_num_scaled= Scaler.fit_transform(X_train_num)
X_test_num_scaled = Scaler.transform(X_test_num)
```

Creating Final Datasets

- This step involves converting the both scaled and encoded data iinto dataframes to enable merging and creation of the final dataset for modelling dataframes to enable merging and creation of the final dataset for modelling purposes

In [35]:
```
X_train_cat_ohe_df= pd.DataFrame(X_train_cat_ohe, index=X_train_cat.index,
X_train_num_scaled_df = pd.DataFrame(X_train_num_scaled, index=X_train_num
X_test_cat_ohe_df= pd.DataFrame(X_test_cat_ohe, index=X_test_cat.index,col
X_test_num_scaled_df = pd.DataFrame(X_test_num_scaled, index=X_test_num.in
X_train_final_df = pd.concat([X_train_num, X_train_cat_ohe_df], axis=1)
X_test_final_df = pd.concat([X_test_num, X_test_cat_ohe_df], axis=1)
print(X_train_final_df.shape)
print(X_test_final_df.shape)

#print (X_train_cat_ohe_df.shape)
#print(X_train_num_scaled_df.shape)
#X_train_final_df = pd.concat([X_train_num, X_train_cat_ohe_df], axis=1)
#print(X_train_final_df.shape)
#drop the second row
#X_train_final_df.drop(X_train_final_df.index[1], inplace=True)
#_train_final_df.head()
```

```
(1666, 21)
(1667, 21)
```

8. MODELLING

Fitting the baseline logistic regression model

This step involves creating a model that uses the logistic regression algorithm to predict Churn among the telcom cleints. first we define/instantiate the model and then fit the training data.

In [36]:
```
#Instantiate the model
LogisticRegressionModel = LogisticRegression(fit_intercept=False, solver=
#Fit the model
Baseline_model = LogisticRegressionModel.fit(X_train_final_df, y_train)
```

Prediction

This step involves making predictions on the test data using the model that we just trained.

In [37]:
```
y_train_pred = Baseline_model.predict(X_train_final_df)
y_test_pred = Baseline_model.predict(X_test_final_df)
```

In [38]:
```
from sklearn.metrics import precision_score, recall_score, accuracy_score,

print('Train precision: %.3f' % precision_score(y_train, y_train_pred))
print('Test precision: %.3f' % precision_score(y_test, y_test_pred))


print("--------------------------------------------")
print('Train recall: %.3f' % recall_score(y_train, y_train_pred))
print('Test recall: %.3f' % recall_score(y_test, y_test_pred))
```

```
print("-------------------------------------------")
print('Train accuracy: %.3f' % accuracy_score(y_train, y_train_pred))
print('Test accuracy: %.3f' % accuracy_score(y_test, y_test_pred))

print("-------------------------------------------")
print('Train f1: %.3f' % f1_score(y_train, y_train_pred))
print('Test f1: %.3f' % f1_score(y_test, y_test_pred))
```

```
Train precision: 0.575
Test precision: 0.581
-------------------------------------------
Train recall: 0.180
Test recall: 0.172
-------------------------------------------
Train accuracy: 0.867
Test accuracy: 0.857
-------------------------------------------
Train f1: 0.275
Test f1: 0.265
```

Conducting SMOTE

The baseline Model shows high accuracy levels but has relatively low precisions. This is shows presence of class imbalance in our dataset. There is need to encode the data before conducting SMOTE as this type of resampling requires numeric variables.

In [39]:

```
#encoding the data before smote
X_train_dm = pd.get_dummies(X_train, drop_first=True)
X_test_dm = pd.get_dummies(X_test, drop_first=True)
```

In [41]:

```
#aplying SMOTE

from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train_dm, y_train)

print(y_train_res.value_counts())
print("-----------------")
print(y_train.value_counts())
```

```
True     1433
False    1433
Name: churn, dtype: int64
-----------------
False    1433
True      233
Name: churn, dtype: int64
```

In [42]:

```
#fiting the model
LogisticRegressionModel = LogisticRegression(fit_intercept=False, solver=
smote_model = LogisticRegressionModel.fit(X_train_res, y_train_res)
y_train_pred_sm = smote_model.predict(X_train_res)
y_test_pred_sm= smote_model.predict(X_test_dm)
```

In [43]:

```
from sklearn.metrics import precision_score, recall_score, accuracy_score,
print('Train precision: %.3f' % precision_score(y_train_res, y_train_pred_
print('Test precision: %.3f' % precision_score(y_test, y_test_pred_sm))

print("------------------------------------------")
print('Train recall: %.3f' % recall_score(y_train_res, y_train_pred_sm))
print('Test recall: %.3f' % recall_score(y_test, y_test_pred_sm))

print("------------------------------------------")
print('Train accuracy: %.3f' % accuracy_score(y_train_res, y_train_pred_sm
print('Test accuracy: %.3f' % accuracy_score(y_test, y_test_pred_sm))

print("------------------------------------------")
print('Train f1: %.3f' % f1_score(y_train_res, y_train_pred_sm))
print('Test f1: %.3f' % f1_score(y_test, y_test_pred_sm))
```

```
Train precision: 0.714
Test precision: 0.324
------------------------------------------
Train recall: 0.754
Test recall: 0.728
------------------------------------------
Train accuracy: 0.726
Test accuracy: 0.731
------------------------------------------
Train f1: 0.733
Test f1: 0.448
```

Ridge and Lasso Regression Models

This step involves creating 2 models: Ridge and Lasso regression. Both these models are regularized versions of the logistic regression model fited after resampling the data i.e "Smote_model". In ridge and lasso models, the loss function is modified to minimize the complexity of the model.

In [44]:

```
from sklearn.linear_model import Lasso, Ridge, LinearRegression
Ridge_reg=Ridge(alpha=1.0)
Lasso_reg=Lasso(alpha=1.0)
Ridge_mod=Ridge_reg.fit(X_train_res, y_train_res)
Lasso_mod=Lasso_reg.fit(X_train_res, y_train_res)
y_train_pred_Ridge=Ridge_mod.predict(X_train_res)
y_test_pred_Ridge=Ridge_mod.predict(X_test_dm)
y_train_pred_Lasso=Lasso_mod.predict(X_train_res)
y_test_pred_Lasso=Lasso_mod.predict(X_test_dm)
```

In [45]:

```
threshold = 0.5
y_train_pred_Ridge_binary = (y_train_pred_Ridge > threshold).astype(int)
y_test_pred_Ridge_binary = (y_test_pred_Ridge > threshold).astype(int)
```

In [46]:

```
print('Ridge Train precision: %.3f' % precision_score(y_train_res, y_trair
print('Ridge Test precision: %.3f' % precision_score(y_test, y_test_pred_F

print("------------------------------------------")
print('Ridge Train recall: %.3f' % recall_score(y_train_res, y_train_pred_
print('Ridge Test recall: %.3f' % recall_score(y_test, y_test_pred_Ridge_t

print("------------------------------------------")
```

```
print('Ridge Train accuracy: %.3f' % accuracy_score(y_train_res, y_train_p
print('Ridge Test accuracy: %.3f' % accuracy_score(y_test, y_test_pred_Ric

print("-------------------------------------------")
print('Ridge Train f1: %.3f' % f1_score(y_train_res, y_train_pred_Ridge_bi
print('Ridge Test f1: %.3f' % f1_score(y_test, y_test_pred_Ridge_binary))
```

```
Ridge Train precision: 0.728
Ridge Test precision: 0.315
-------------------------------------------
Ridge Train recall: 0.772
Ridge Test recall: 0.720
-------------------------------------------
Ridge Train accuracy: 0.742
Ridge Test accuracy: 0.723
-------------------------------------------
Ridge Train f1: 0.749
Ridge Test f1: 0.438
```

In [47]:

```
threshold = 0.5
y_train_pred_Lasso_binary = (y_train_pred_Ridge > threshold).astype(int)
y_test_pred_Lasso_binary = (y_test_pred_Ridge > threshold).astype(int)

print('Lasso Train precision: %.3f' % precision_score(y_train_res, y_trair
print('Lasso Test precision: %.3f' % precision_score(y_test, y_test_pred_L

print("-------------------------------------------")
print('Lasso Train recall: %.3f' % recall_score(y_train_res, y_train_pred_
print('Lasso Test recall: %.3f' % recall_score(y_test, y_test_pred_Lasso_b

print("-------------------------------------------")
print('Lasso Train accuracy: %.3f' % accuracy_score(y_train_res, y_train_p
print('Lasso Test accuracy: %.3f' % accuracy_score(y_test, y_test_pred_Las

print("-------------------------------------------")
print('Lasso Train f1: %.3f' % f1_score(y_train_res, y_train_pred_Lasso_bi
print('Lasso Test f1: %.3f' % f1_score(y_test, y_test_pred_Lasso_binary))
```

```
Lasso Train precision: 0.728
Lasso Test precision: 0.315
-------------------------------------------
Lasso Train recall: 0.772
Lasso Test recall: 0.720
-------------------------------------------
Lasso Train accuracy: 0.742
Lasso Test accuracy: 0.723
-------------------------------------------
Lasso Train f1: 0.749
Lasso Test f1: 0.438
```

Decision Tree Fitting

This step involves fitting a decision tree on the resampled data where the class imbalance has already been dealt with.This will enable us choose the best prediction model as well as strategy.

In [48]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

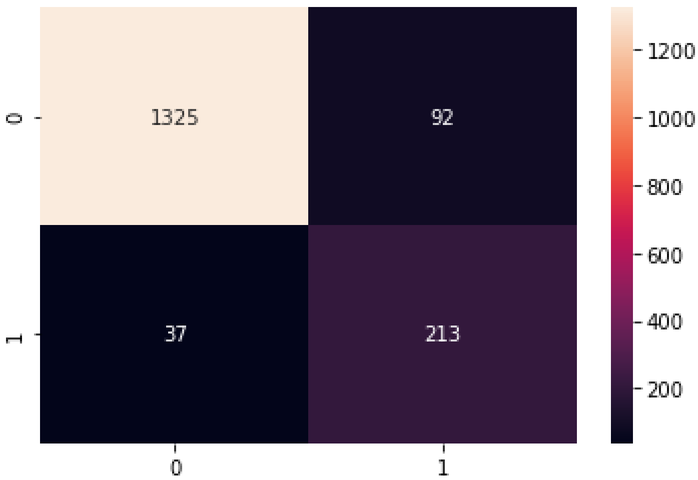
```
classifier=DecisionTreeClassifier(criterion='entropy',min_samples_split=5,
classifier.fit(X_train_res, y_train_res)
#accuracy for the training set
y_pred_train_dt=classifier.predict(X_train_res)
print("Accuracy for train is",accuracy_score(y_train_res,y_pred_train_dt)*

#Predicting the Test set results
y_pred_test_dt=classifier.predict(X_test_dm)
print("Accuracy for test is",accuracy_score(y_test,y_pred_test_dt)*100)
```

```
Accuracy for train is 81.47243545010467
Accuracy for test is 92.26154769046191
```

In [49]:

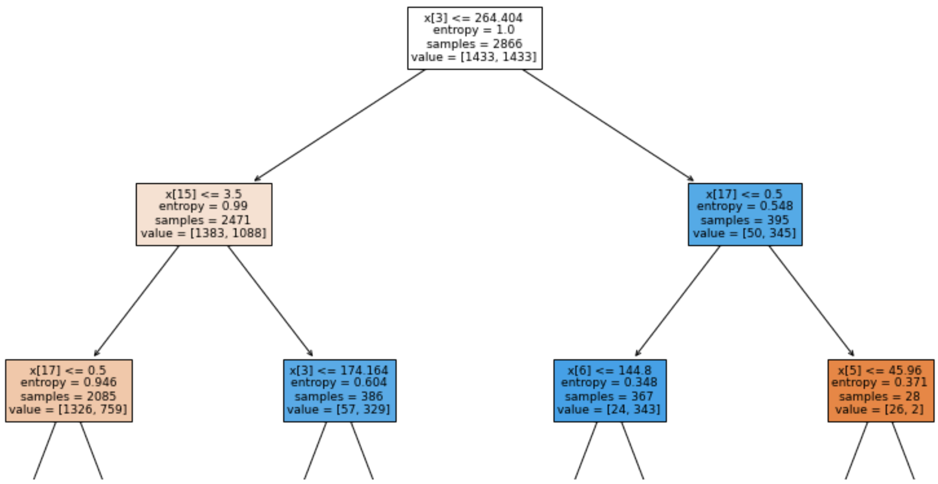
```
#Fitting a confusion Matrix for the de
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred_test_dt)
#plotting the confusion matrix

sns.heatmap(confusion_matrix(y_test,y_pred_test_dt),annot=True,fmt='d')
```

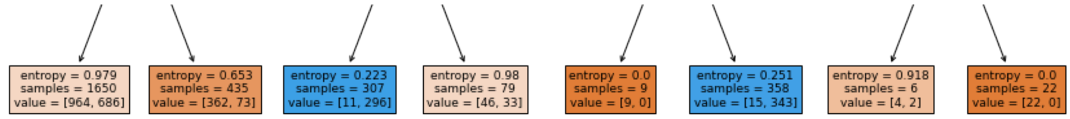
Out[49]: <AxesSubplot:>



In [ ]:

```
#plotting the decision tree
from sklearn import tree
import matplotlib.pyplot as plt
plt.figure(figsize=(15,10))
tree.plot_tree(classifier,filled=True)
plt.show()
```





9. EVALUATION

Performance Metrics of the baseline model performance

Below is a brief description of the evaluation metrics of the baseline model:

Precision: Train: 0.568, Test: 0.579

The precision values for both the train and test sets are similar, which suggests that the model has a consistent rate of correctly identifying positive instances across both sets. However, the precision is relatively low, indicating that a significant number of predicted positives are actually false positives.

Recall: Train: 0.180 Test: 0.176

The recall values are quite low for both train and test sets, which means the model is missing a large number of actual positive instances. This might suggest that the model is not very sensitive and is underfitting the positive class.

Accuracy: Train: 0.866 Test: 0.857

The accuracy is relatively high, but this might be misleading if the dataset is imbalanced (i.e., there are many more negatives than positives). High accuracy in such cases could simply reflect the model's ability to predict the majority class correctly.

F1 Score: Train: 0.274 Test: 0.270

The F1 score, which is the harmonic mean of precision and recall, is quite low for both train and test sets. This indicates a poor balance between precision and recall, reflecting overall weak performance in identifying the positive class.

Performance Metrics of Second Model After Solving Class Imbalance with SMOTE

Precision: Train Precision: 0.709, Test Precision: 0.324

Precision on the training data is high, but it drops significantly on the test data. This suggests that while the model performs well in identifying positive instances during training, it struggles to maintain this performance on unseen data. This drop in precision could be due to overfitting.

Recall: Train Recall: 0.749, Test Recall: 0.732

Recall is relatively consistent between training and testing. This means the model is fairly good at identifying positive instances across both datasets, although it might not capture all possible positive cases.

Accuracy: Train Accuracy: 0.721, Test Accuracy: 0.731

Accuracy is fairly similar for both training and testing, which is a good sign.

F1 Score: Train F1: 0.729, Test F1: 0.450

The F1 score, which balances precision and recall, is much lower on the test data compared to the training data. This indicates that the model might be performing well on precision and recall during training but is not generalizing well to new data.

In summary, the Evaluation Metrics seem to be performing well on training data but not very well on the test data. This suggests overfitting and need to regularrize the model

Performance Metrics of Ridge and Lasso Models

The model evaluation Metrics for both lasso and ridge models seem constant the Log_reg model trained asfter conducting SMOTE. We therefore condlude that the second model is the best we have for predicting Churn among the Syria Tel Customers.

Performance Metrics of the Decision Tree

The Decision Tree model performs well on both the training with an accuracy of 71% and testing data, with a high accuracy score of 91%. This suggests that the model is not overfitting the data and can generalize well to new, unseen data. This makes it the most accurate prediction model for the syriatel data.

10. CONCLUSIONS

The results of this analysis show that SyriaTel has a significant customer churn rate of approximately 14%. This is a significant issue, as it results in a loss of