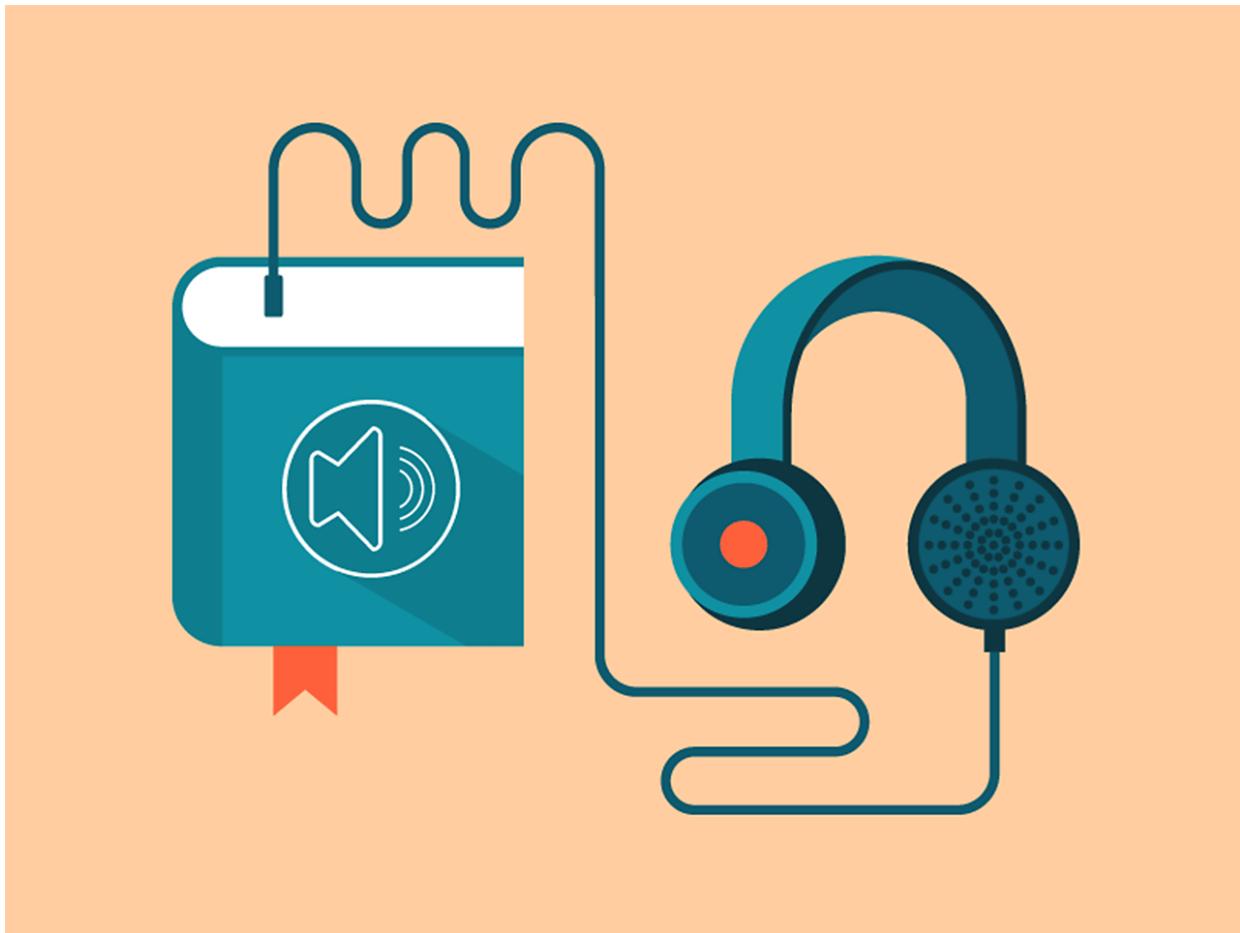


Business Case Study: Audiobook app



You are given data from an Audiobook app. Logically, it relates only to the audio versions of books. We want to create a machine learning model based on our available data that can predict if a customer will buy again from the Audiobook company.

- The data is from an audiobook app, each customer in the database has made a purchase at least once.
- The main idea is that the company shouldn't spend their money targeting individuals who are unlikely to come back.
- If we focus on clients who are more likely to convert again we'll get increase the sales and profitability figures.

The model must show us which are the most important metrics for a client to come back.

In [82]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
import tensorflow as tf
```

```
%matplotlib inline
sns.set_style("whitegrid")

pd.set_option("display.max_columns", 80)
pd.set_option("display.max_rows", 80)
pd.set_option("display.float_format", "{:.2f}".format)
```

In [2]:

```
data = pd.read_csv("audiobook_data_2.csv", index_col=0)
data.head()
```

Out[2]:

	Book_length(mins)_overall	Book_length(mins)_avg	Price_overall	Price_avg	Review	Review10/10
994	1620.00	1620	19.73	19.73	1	10.00
1143	2160.00	2160	5.33	5.33	0	8.91
2059	2160.00	2160	5.33	5.33	0	8.91
2882	1620.00	1620	5.96	5.96	0	8.91
3342	2160.00	2160	5.33	5.33	0	8.91

In [3]:

```
#count = (data['Review']).value_counts()[0]
#print("Number of users who didnt give reviews are : ", count, " out of 14084")
```

1. Getting aquainted with the dataset

- `ID` : is like a name.
- `Book_length(mins)_overall` : is the sum of the lengths of purchases.
- `Book_length(mins)_avg` : is the sum of the lengths of purchases divided by the number of purchases. Notice we don't need the number of purchases column because we can get it from `Book_length(mins)_overall / Book_length(mins)_avg` .
- `Price_overall & Price_avg` : Same as Book length, the price variable is almost always a good predictor.
- `Review` : is boolean. It shows if the customer left a review. If so, `Review10/10` saves the review left by the user. While most users don't left a review we fill the missing reviews by average review column.
- `Minutes_listened` : is a measure of engagement, the total of minutes the user listen to audiobooks.
- `Completion` : is the `Minutes_listened / Book_length(mins)_overall` .
- `Support_Request` : Shows the total number of support request (forgotten password to assistance).

- `Last_Visited_mins_Purchase_date` : the bigger the difference, the bigger sooner the engagement. If the value is 0, we are sure the customer has never accessed what he/she has bought.

The data was gathered from the audiobook app, the input data represents 2 years worth of engagement. We are doing supervised learning so we need `target`. We took extra 6 month to check if the user converted or not. 1 if the customer buys in the next 6 months, 0 if the customer didn't.

- `target` : 1 if the customer bought again in the last 6 months of data. 0 if the customer did not buy again.

2. Exploratory Data Analysis

In [4]: `data.describe()`

	<code>Book_length(mins)_overall</code>	<code>Book_length(mins)_avg</code>	<code>Price_overall</code>	<code>Price_avg</code>	<code>Review</code>	<code>Review10/10</code>
<code>count</code>	14084.00	14084.00	14084.00	14084.00	14084.00	14084.00
<code>mean</code>	1591.28	1678.61	7.10	7.54	0.16	8.9
<code>std</code>	504.34	654.84	4.93	5.56	0.37	0.6
<code>min</code>	216.00	216.00	3.86	3.86	0.00	1.0
<code>25%</code>	1188.00	1188.00	5.33	5.33	0.00	8.9
<code>50%</code>	1620.00	1620.00	5.95	6.07	0.00	8.9
<code>75%</code>	2160.00	2160.00	8.00	8.00	0.00	8.9
<code>max</code>	2160.00	7020.00	130.94	130.94	1.00	10.0

In [5]: `data.isnull().sum()`

```
Out[5]: Book_length(mins)_overall      0
Book_length(mins)_avg          0
Price_overall                  0
Price_avg                      0
Review                          0
Review10/10                     0
Completion                      0
Minutes_listened                0
Support_Request                 0
Last_Visited_mins_Purchase_date 0
Target                          0
dtype: int64
```

In [6]: `data.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14084 entries, 994 to 251
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
 ---  -- 
 0   Book_length(mins)_overall    14084 non-null   float64
 1   Book_length(mins)_avg        14084 non-null   float64
 2   Price_overall               14084 non-null   float64
 3   Price_avg                   14084 non-null   float64
 4   Review                       14084 non-null   float64
 5   Review10/10                 14084 non-null   float64
 6   Completion                   14084 non-null   float64
 7   Minutes_listened            14084 non-null   float64
 8   Support_Request              14084 non-null   float64
 9   Last_Visited_mins_Purchase_date 14084 non-null   float64
 10  Target                      14084 non-null   int64
```

```

0 Book_length(mins)_overall      14084 non-null float64
1 Book_length(mins)_avg          14084 non-null int64
2 Price_overall                 14084 non-null float64
3 Price_avg                      14084 non-null float64
4 Review                         14084 non-null int64
5 Review10/10                   14084 non-null float64
6 Completion                     14084 non-null float64
7 Minutes_listened               14084 non-null float64
8 Support_Request                14084 non-null int64
9 Last_Visited_mins_Purchase_date 14084 non-null int64
10 Target                        14084 non-null int64
dtypes: float64(6), int64(5)
memory usage: 1.3 MB

```

Book_length(mins)_overall & Book_length(mins)_avg

- Book_length(mins)_overall : is the sum of the lengths of purchases.
- Book_length(mins)_avg : is the sum of the lengths of purchases divided by the number of purchases. Notice we don't need the number of purchases column because we can get it from Book_length(mins)_overall / Book_length(mins)_avg .

```
In [7]: data['Book_length(mins)_overall'].value_counts()
```

```

Out[7]: 2160.00    4712
1620.00    4149
1188.00    2851
648.00     712
1080.00    567
324.00     300
540.00     115
1404.00    112
1890.00    110
1674.00    98
1134.00    52
216.00     51
1656.00    27
1476.00    23
1350.00    21
756.00     21
918.00     15
1332.00    14
864.00     13
1242.00    13
1296.00    10
1152.00    8
972.00     7
1116.00    7
486.00     6
1377.00    5
1224.00    5
1512.00    5
1368.00    5
594.00     4
432.00     4
1044.00    4
702.00     3
1008.00    3

```

```
684.00      3
1161.00     2
1269.00     2
504.00      2
576.00      2
945.00      1
720.00      1
1231.20     1
990.00      1
378.00      1
1170.00     1
1098.00     1
1252.80     1
828.00      1
1058.40     1
928.80      1
999.00      1
810.00      1
1431.00     1
1440.00     1
1002.86     1
1062.00     1
1339.20     1
270.00      1
612.00      1
1260.00     1
Name: Book_length(mins)_overall, dtype: int64
```

```
In [8]: def book_length(length):
    if length > 1200:
        return 1
    else:
        return 0

data['purchases_hour_>3h'] = data['Book_length(mins)_overall'].apply(book_length)
```

```
In [9]: data['Book_length(mins)_avg'].apply(book_length).value_counts()
```

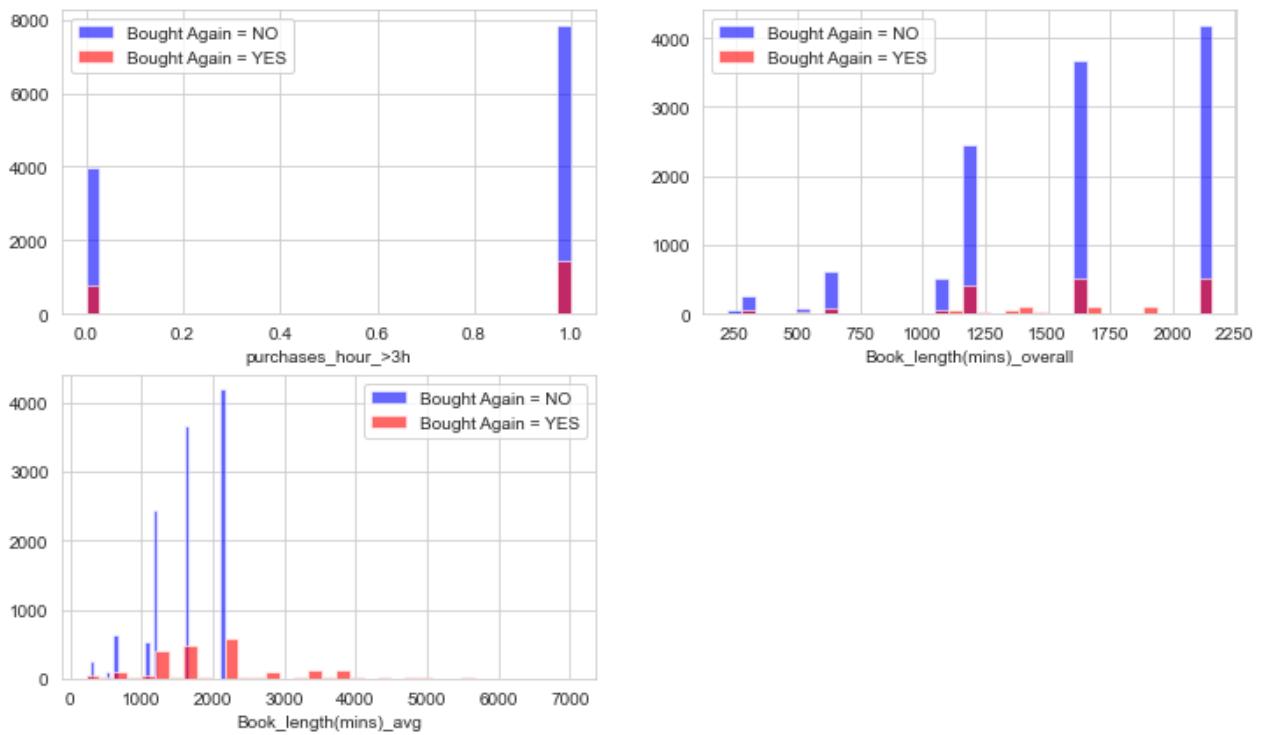
```
Out[9]: 1    9480
0    4604
Name: Book_length(mins)_avg, dtype: int64
```

```
In [10]: data['purchases_hour_>3h'].value_counts()
```

```
Out[10]: 1    9317
0    4767
Name: purchases_hour_>3h, dtype: int64
```

```
In [11]: columns = ['purchases_hour_>3h', 'Book_length(mins)_overall', 'Book_length(mins)_avg']
plt.figure(figsize=(12, 7))

for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    data[data["Target"] == 0][column].hist(bins=35, color='blue', label='Bought Again = 0')
    data[data["Target"] == 1][column].hist(bins=35, color='red', label='Bought Again = 1')
    plt.legend()
    plt.xlabel(column)
```

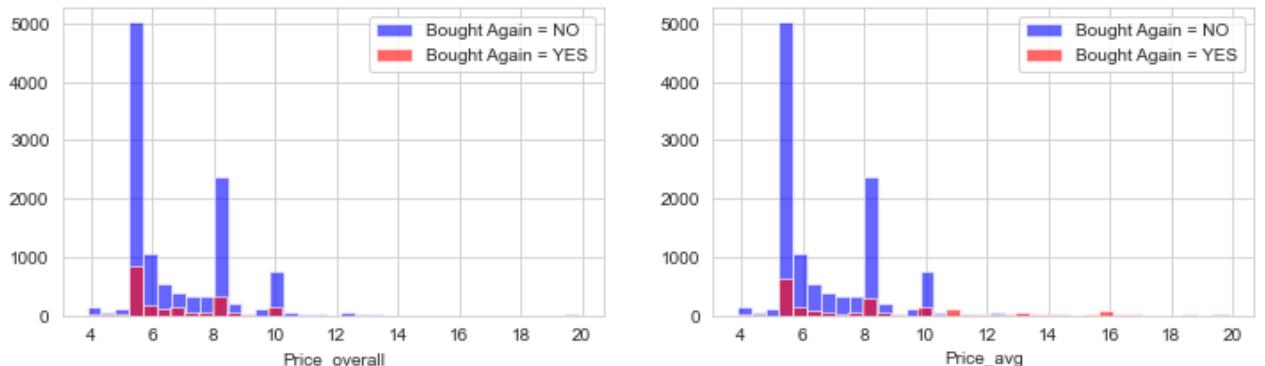


Price_overall & Price_avg

- Price_overall & Price_avg : Same as Book length, the price variable is almost always a good predictor.

```
In [12]: columns = ["Price_overall", "Price_avg"]
plt.figure(figsize=(12, 7))
df = data[(data.Price_overall < 20) & (data.Price_avg < 20)]

for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    df[df["Target"] == 0][column].hist(bins=35, color='blue', label='Bought Again = NO')
    df[df["Target"] == 1][column].hist(bins=35, color='red', label='Bought Again = YES')
    plt.legend()
    plt.xlabel(column)
```



Review & Review10/10

- Review : is boolean. It shows if the customer left a review. If so, Review10/10 saves the review left by the user. While most users don't left a review we fill the missing reviews by average review column.

```
In [13]: print(data[data['Review'] == 0].Target.value_counts(normalize=True))
print(data[data['Review'] == 1].Target.value_counts(normalize=True))
```

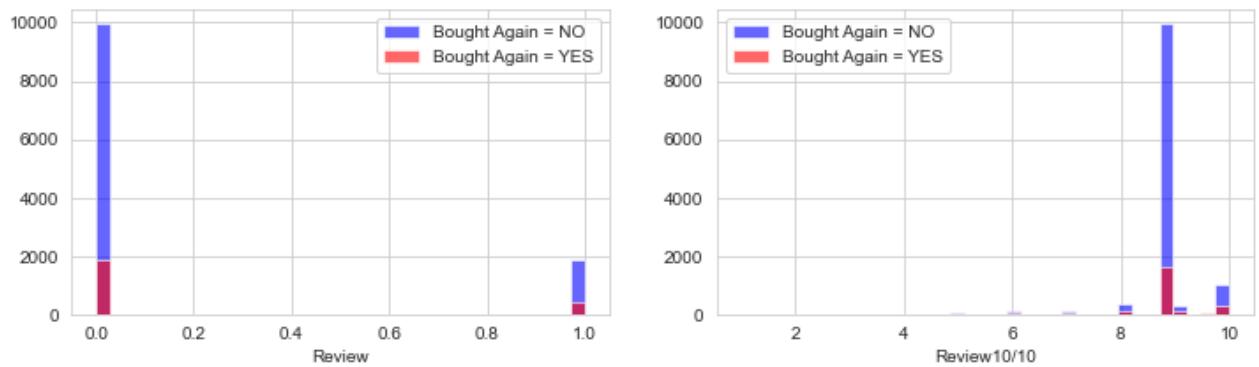
```
0    0.84
1    0.16
Name: Target, dtype: float64
0    0.83
1    0.17
Name: Target, dtype: float64
```

```
In [14]: data['Review10/10'].value_counts()
```

```
Out[14]: 8.91      11616
10.00     1284
8.00      404
9.00      381
7.00      157
6.00      104
5.00       43
9.50       21
4.00       18
8.50       11
1.00       10
3.00        9
2.00        7
6.50        5
8.33        2
5.50        2
8.67        2
7.50        2
6.67        1
1.50        1
9.40        1
7.75        1
4.50        1
9.67        1
Name: Review10/10, dtype: int64
```

```
In [15]: columns = ["Review", "Review10/10"]
plt.figure(figsize=(12, 7))
```

```
for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    data[data["Target"] == 0][column].hist(bins=35, color='blue', label='Bought Again = 0')
    data[data["Target"] == 1][column].hist(bins=35, color='red', label='Bought Again = 1')
    plt.legend()
    plt.xlabel(column)
```



Minutes_listened & Completion

- Minutes_listened : is a measure of engagement, the total of minutes the user listen to audiobooks.
- Completion : is the Minutes_listened / Book_length(mins)_overall .

```
In [16]: def listened_to_books(minutes):
    if minutes > 0.0:
        return 0
    else:
        return 1
data['listened_to_books'] = data.Minutes_listened.apply(listened_to_books)
```

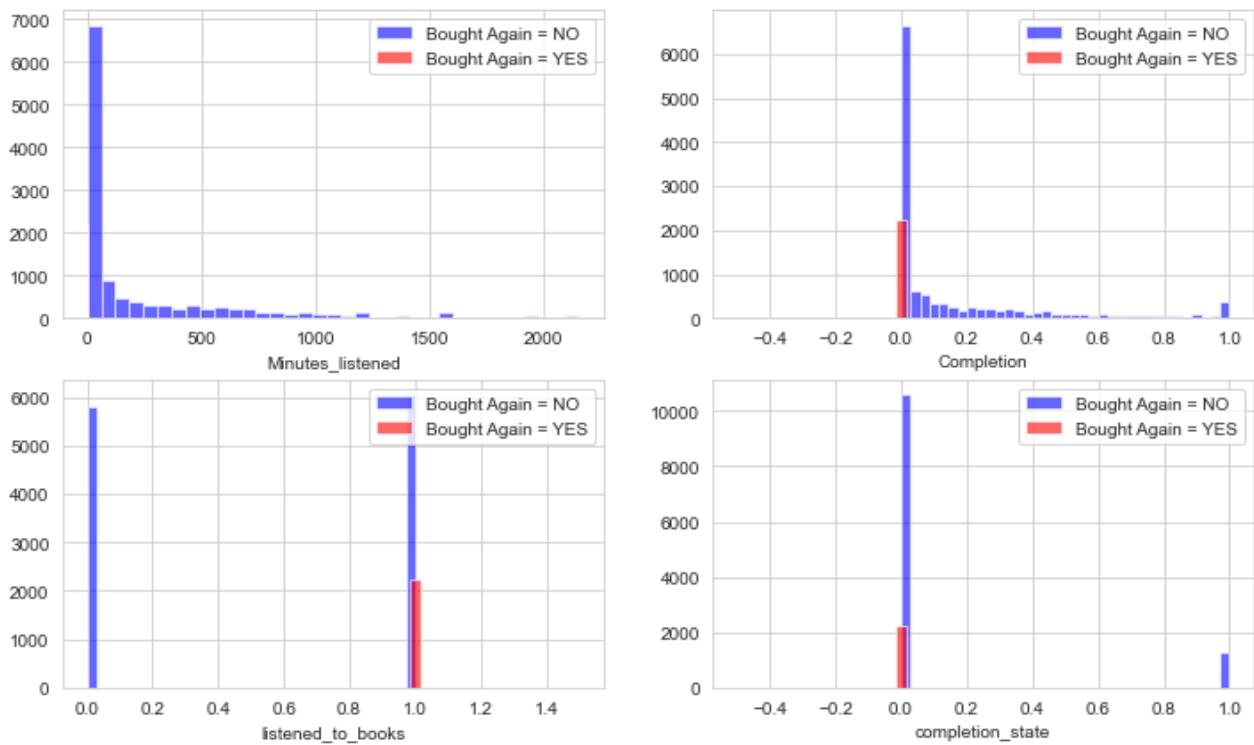


```
In [17]: def completion_state(minutes):
    if minutes > 0.5:
        return 1
    else:
        return 0
data['completion_state'] = data.Completion.apply(completion_state)
```



```
In [18]: columns = ["Minutes_listened", "Completion", "listened_to_books", "completion_state"]
plt.figure(figsize=(12, 7))

for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    data[data["Target"] == 0][column].hist(bins=35, color='blue', label='Bought Again = NO')
    data[data["Target"] == 1][column].hist(bins=35, color='red', label='Bought Again = YES')
    plt.legend()
    plt.xlabel(column)
```



```
In [19]: data.drop('Minutes_listened', axis=1, inplace=True)
```

Support_Request & Last_Visited_mins_Purchase_date

- **Support_Request** : Shows the total number of support request (forgotten password to assistance).
- **Last_Visited_mins_Purchase_date** : the bigger the difference, the bigger sooner the engagement. If the value is 0, we are sure the customer has never accessed what he/she has bought.

```
In [20]: def asked_for_request(request):
    if request == 0:
        return 0
    else:
        return 1

data["asked_for_request"] = data.Support_Request.apply(asked_for_request)
```

```
In [21]: def acc_purchases(purchase):
    if purchase == 0:
        return 0
    else:
        return 1
data['acc_purchases'] = data.Last_Visited_mins_Purchase_date.apply(acc_purchases)
```

```
In [22]: data.Last_Visited_mins_Purchase_date.value_counts()
```

```
Out[22]: 0      5493
          1      357
```

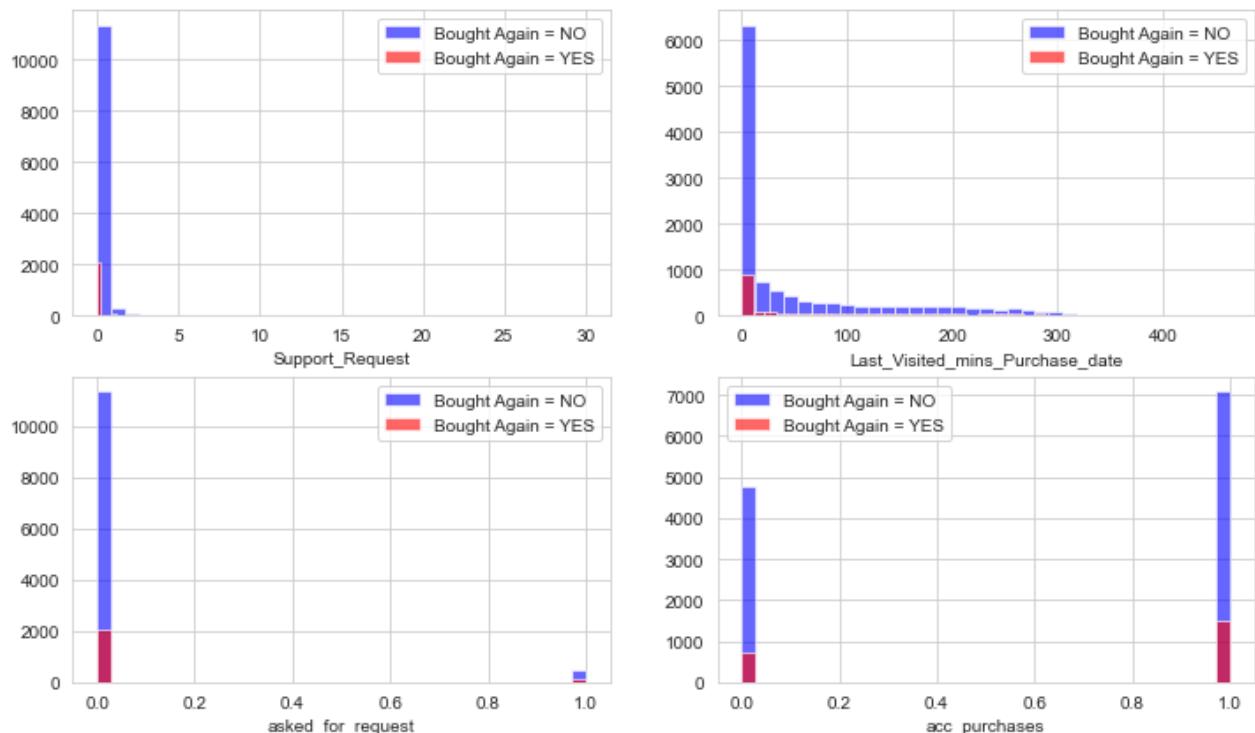
```

2      198
3      165
5      140
...
354      1
338      1
369      1
464      1
379      1
Name: Last_Visited_mins_Purchase_date, Length: 371, dtype: int64

```

```
In [23]: columns = ["Support_Request", "Last_Visited_mins_Purchase_date", "asked_for_request", "acc_purchases"]
plt.figure(figsize=(12, 7))

for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    data[data["Target"] == 0][column].hist(bins=35, color='blue', label='Bought Again = NO')
    data[data["Target"] == 1][column].hist(bins=35, color='red', label='Bought Again = YES')
    plt.legend()
    plt.xlabel(column)
```



```
In [24]: data.drop('Support_Request', axis=1, inplace=True)
```

```
In [25]: print(f"{data.Target.value_counts()}")
print(f"{data.Target.value_counts()[0] / data.Target.value_counts()[1]}")
```

```

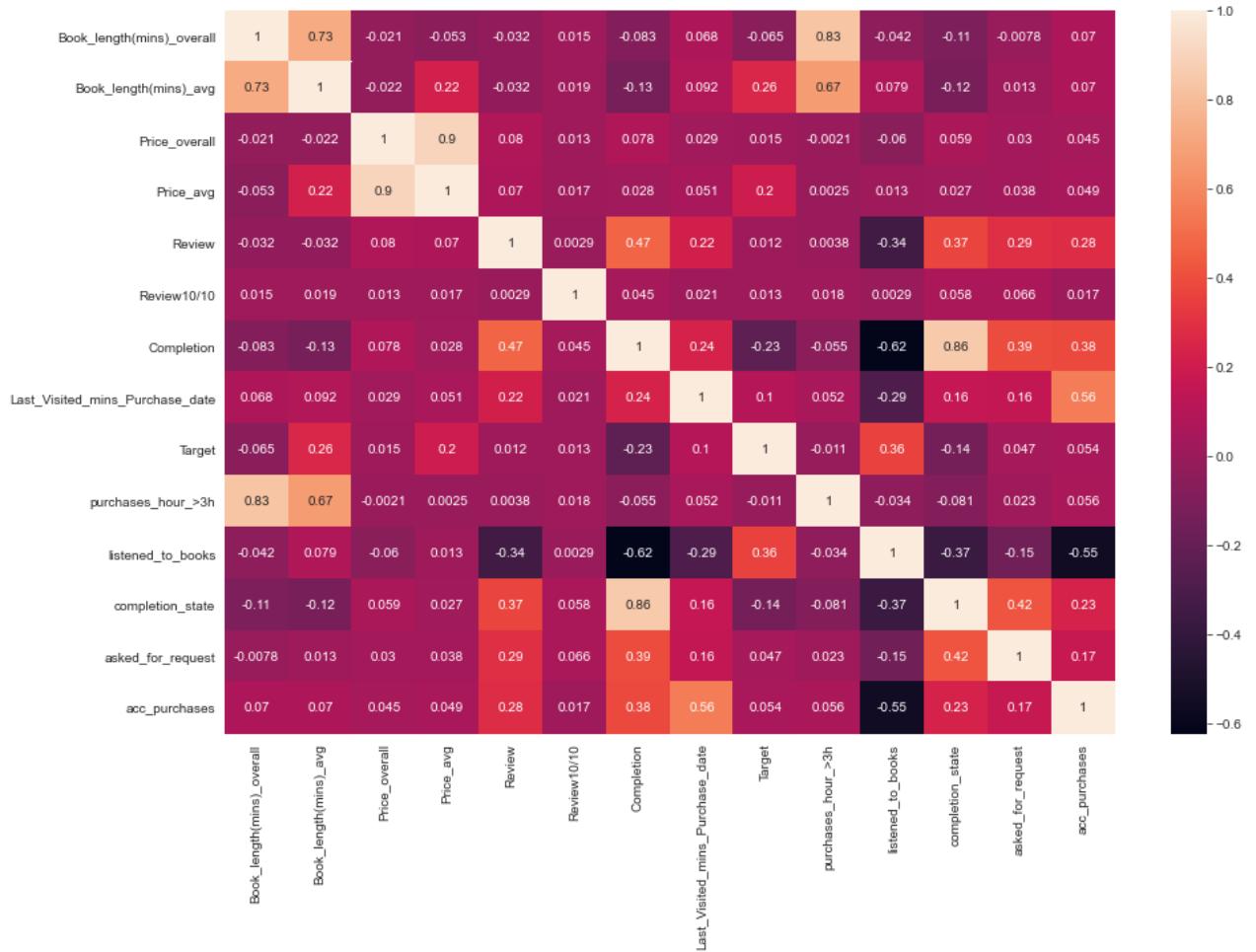
0      11847
1      2237
Name: Target, dtype: int64
5.295932051855163

```

It is important to notice that our `target` variable is imbalanced. We have only 2237 user who convert again in the 6 month period. The data need to be balanced.

```
In [26]: plt.figure(figsize=(15, 10))
sns.heatmap(data.corr(), annot=True)
```

Out[26]: <AxesSubplot:>



3. Data Pre-processing

Since we are dealing with real life data, we will need to preprocess it a bit. This is the relevant code which is not that hard but refers to data engineering more than machine learning.

- Balance the dataset.
- Standardising
- Shuffling
- Splitting

Loading data

```
In [27]: r_data = pd.read_csv("audiobook_data_2.csv", index_col=0)
```

Removing duplicates

```
In [28]: print(f"Data shape before removing duplicates: {r_data.shape}")
```

```
# Remove duplicate Features
r_data = r_data.T.drop_duplicates()
r_data = r_data.T
```

```
# Remove Duplicate Rows
r_data.drop_duplicates(inplace=True)

print(f"Data shape after removing duplicates: {r_data.shape}")
```

Data shape before removing duplicates: (14084, 11)
 Data shape after removing duplicates: (9350, 11)

```
In [29]: X = r_data.loc[:, r_data.columns != 'Target']
y = r_data.loc[:, r_data.columns == 'Target']

raw_csv_data = np.loadtxt('audiobook_data_2.csv', delimiter = ',', skiprows=1)

unscaled_inputs_all = raw_csv_data[:,1:-1]

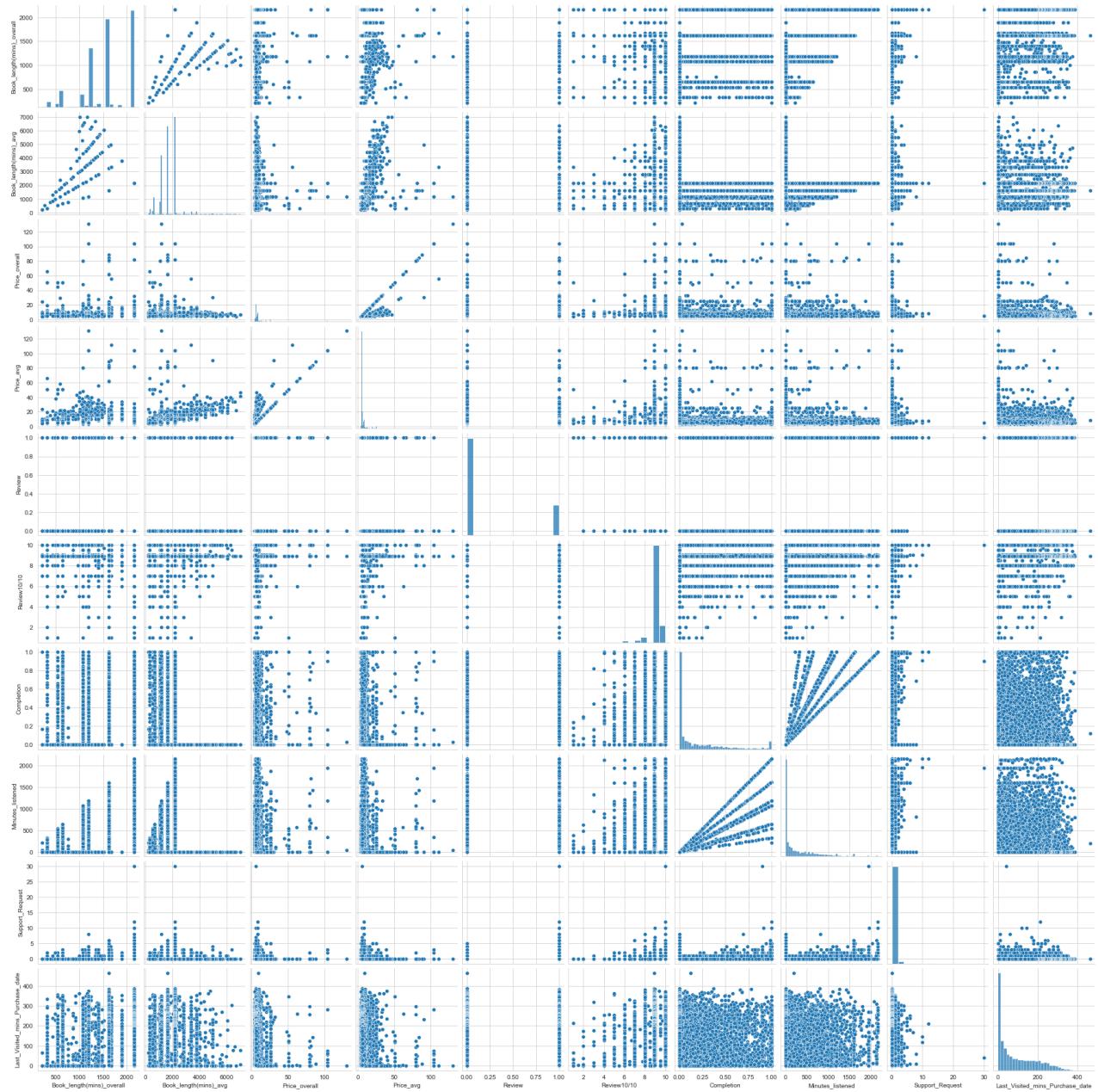
targets_all = raw_csv_data[:, -1]
raw_csv_data
```

```
Out[29]: array([[9.9400e+02, 1.6200e+03, 1.6200e+03, ..., 5.0000e+00, 9.2000e+01,
   0.0000e+00],
 [1.1430e+03, 2.1600e+03, 2.1600e+03, ..., 0.0000e+00, 0.0000e+00,
   0.0000e+00],
 [2.0590e+03, 2.1600e+03, 2.1600e+03, ..., 0.0000e+00, 3.8800e+02,
   0.0000e+00],
 ...,
 [3.1134e+04, 2.1600e+03, 2.1600e+03, ..., 0.0000e+00, 0.0000e+00,
   0.0000e+00],
 [3.2832e+04, 1.6200e+03, 1.6200e+03, ..., 0.0000e+00, 9.0000e+01,
   0.0000e+00],
 [2.5100e+02, 1.6740e+03, 3.3480e+03, ..., 0.0000e+00, 0.0000e+00,
   1.0000e+00]])
```

Pair-Wise Coorelation with each other

```
In [110... sns.pairplot(r_data.iloc[:,0:10])
```

```
Out[110... <seaborn.axisgrid.PairGrid at 0x1c3dec89df0>
```



Pair-Wise Correlation with each other aspects to our Target variable

In [114]: `sns.pairplot(r_data.iloc[:,0:11], hue="Target")`

```
C:\Users\ARYAN GULATI\anaconda3\lib\site-packages\seaborn\distributions.py:305: UserWarning: Dataset has 0 variance; skipping density estimate.
    warnings.warn(msg, UserWarning)
C:\Users\ARYAN GULATI\anaconda3\lib\site-packages\seaborn\distributions.py:305: UserWarning: Dataset has 0 variance; skipping density estimate.
    warnings.warn(msg, UserWarning)
```

Out[114]: <seaborn.axisgrid.PairGrid at 0x1c3badf9a90>

Stage 2 Churn rate duplicatesRemoved



4. Model Building

4. 1. Logistic Regression

```
In [30]: from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import seaborn as sns

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)

(6545, 10)
(2805, 10)
(6545, 1)
```

Converting 2D dataframe 'y_train' into 1D array (expected during training)

```
In [31]: print(y_train.shape)
y_arr = y_train.values
y_train = y_arr.flatten()
print(y_train.shape)

(6545, 1)
(6545,)
```

```
In [32]: print(y_train[5])
```

1.0

```
In [33]: y_train.shape
sum1 = 0
sum0 = 0
for i in range(y_train.shape[0]):
    if(y_train[i]==1):
        sum1+=1
    else:
        sum0+=1

zeros = (sum0 / y_train.shape[0])
ones = (sum1 / y_train.shape[0])
print(sum1)
print(sum0)
print(f"Doesn't purchase again users Rate: {zeros * 100:.2f}%")
print(f"Purchase again users Rate: {ones * 100 :.2f}%")
```

1264
5281
Doesn't purchase again users Rate: 80.69%
Purchase again users Rate: 19.31%

.values convert dataframe into ndarray | .flatten converts ndarray into 1D

```
In [34]: logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
Out[34]: LogisticRegression()
```

```
In [35]: y_pred = logreg.predict(X_test)
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.sc
```

Accuracy of logistic regression classifier on test set: 0.90

Accuracy of Logistic regression is 90.00%

Confusion matrix

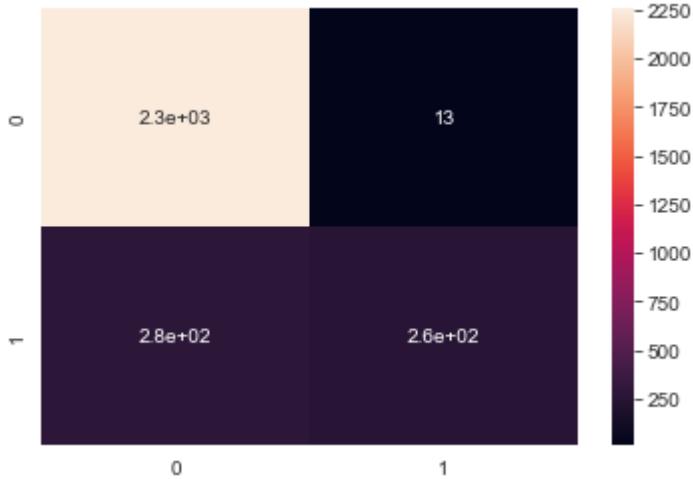
```
In [74]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_pred)
print(confusion_matrix)
```

```
[[2254  13]
 [ 280 258]]
```

The result is telling us that we have 2254+258 = '2512' correct predictions and 280+13 = '293' incorrect predictions.

In [75]: `sns.heatmap(confusion_matrix, annot=True)`

Out[75]: <AxesSubplot:>



In [37]: `from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))`

	precision	recall	f1-score	support
0.0	0.89	0.99	0.94	2267
1.0	0.95	0.48	0.64	538
accuracy			0.90	2805
macro avg	0.92	0.74	0.79	2805
weighted avg	0.90	0.90	0.88	2805

The precision is the ratio $tp / (tp + fp)$ where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier to not label a sample as positive if it is negative.

The recall is the ratio $tp / (tp + fn)$ where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights the recall more than the precision by a factor of beta. beta = 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in `y_test`.

4. 2 Support Vector Machine

In [38]: `from sklearn.svm import SVC`

```
svm_clf = SVC()
svm_clf.fit(X_train, y_train)
```

Out[38]: `SVC()`

```
In [39]: y_predSVM = svm_clf.predict(X_test)
print('Accuracy of Support Vector Machine classifier on test set: {:.2f}'.format(svm_clf))

Accuracy of Support Vector Machine classifier on test set: 0.86
```

Accuracy of SVM classification is 86.00%

Confusion matrix

```
In [62]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_predSVM)
print(confusion_matrix)

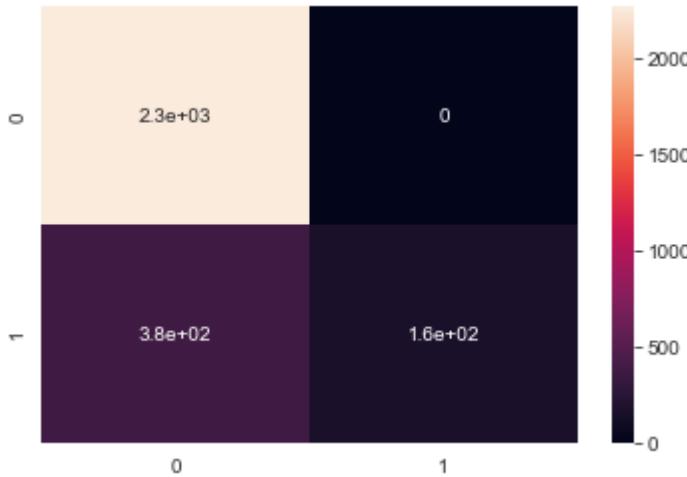
[[2267    0]
 [ 379  159]]
```

Correct predictions : $2267 + 159 = 2426$

Incorrect predictions : $379 + 0 = 379$

```
In [63]: sns.heatmap(confusion_matrix, annot=True)
```

Out[63]: <AxesSubplot:>



```
In [41]: from sklearn.metrics import classification_report
print(classification_report(y_test, y_predSVM))
```

	precision	recall	f1-score	support
0.0	0.86	1.00	0.92	2267
1.0	1.00	0.30	0.46	538
accuracy			0.86	2805
macro avg	0.93	0.65	0.69	2805
weighted avg	0.88	0.86	0.83	2805

4. 3 Decision tree

```
In [42]: from sklearn.tree import DecisionTreeClassifier
DTclf_model = DecisionTreeClassifier(criterion="gini", random_state=1, max_depth=5, min_
```

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

```
Out[42]: DecisionTreeClassifier(max_depth=5, random_state=1)
```

```
In [43]: y_predDT = DTclf_model.predict(X_test)
print('Accuracy of Decision tree classifier on test set: {:.2f}'.format(DTclf_model.score(X_test, y_test)))
```

Accuracy of Decision tree classifier on test set: 0.90

Accuracy of Decision tree classification is 90.00%

Confusion matrix

```
In [72]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_predDT)
print(confusion_matrix)
```

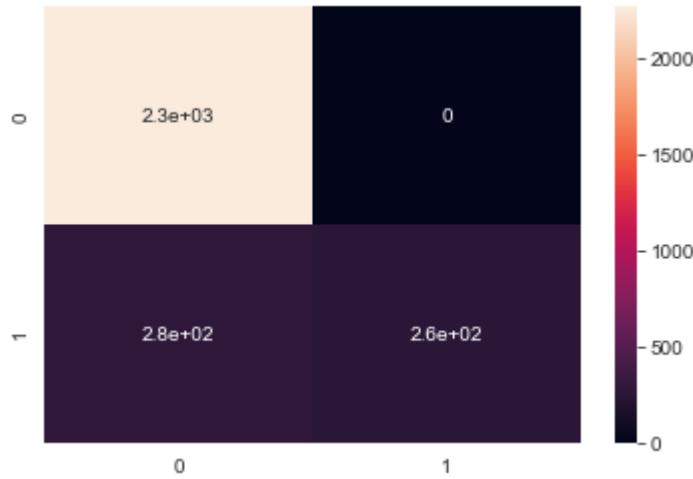
[[2267 0]
 [283 255]]

Correct predictions : $2267 + 255 = 2522$

Incorrect predictions : $283 + 0 = 283$

```
In [73]: sns.heatmap(confusion_matrix, annot=True)
```

```
Out[73]: <AxesSubplot:>
```



```
In [45]: from sklearn.metrics import classification_report
print(classification_report(y_test, y_predDT))
```

	precision	recall	f1-score	support
0.0	0.89	1.00	0.94	2267
1.0	1.00	0.47	0.64	538
accuracy			0.90	2805
macro avg	0.94	0.74	0.79	2805
weighted avg	0.91	0.90	0.88	2805

4.3.1 XG Boost

In [55]:

```
from xgboost import XGBClassifier
xgb_clf = XGBClassifier(learning_rate=0.5, n_estimators=150, base_score=0.5)
xgb_clf.fit(X_train, y_train)
y_predXG = xgb_clf.predict(X_test)
print('Accuracy of Extreme Gradient Boosting classifier on test set: {:.2f}'.format(xgb_clf.score(X_test, y_test)))
```

Accuracy of Extreme Gradient Boosting classifier on test set: 0.88

Accuracy of Decision tree classification is 88.00%

Confusion matrix

In [70]:

```
from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_predXG)
print(confusion_matrix)
```

```
[[2142 125]
 [ 212 326]]
```

In [71]:

```
sns.heatmap(confusion_matrix, annot=True)
```

Out[71]:



In [58]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predXG))
```

	precision	recall	f1-score	support
0.0	0.91	0.94	0.93	2267
1.0	0.72	0.61	0.66	538
accuracy			0.88	2805
macro avg	0.82	0.78	0.79	2805
weighted avg	0.87	0.88	0.88	2805

4.4 Random Forest

In [46]:

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier(n_estimators=1500, oob_score=True)
rf_clf.fit(X_train, y_train)
```

```
Out[46]: RandomForestClassifier(n_estimators=1500, oob_score=True)
```

```
In [47]: y_predRF = rf_clf.predict(X_test)
print('Accuracy of Random Forest classifier on test set: {:.2f}'.format(rf_clf.score(X_))

Accuracy of Random Forest classifier on test set: 0.87
```

Accuracy of Random Forest is 87.00%

```
In [68]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_predRF)
print(confusion_matrix)
```

```
[[2115 152]
 [ 213 325]]
```

Correct predictions : $2113 + 327 = 2440$

Incorrect predictions : $211 + 154 = 365$

```
In [69]: sns.heatmap(confusion_matrix, annot=True)
```

```
Out[69]: <AxesSubplot:>
```



```
In [49]: from sklearn.metrics import classification_report
print(classification_report(y_test, y_predRF))
```

	precision	recall	f1-score	support
0.0	0.91	0.93	0.92	2267
1.0	0.68	0.60	0.64	538
accuracy			0.87	2805
macro avg	0.79	0.77	0.78	2805
weighted avg	0.86	0.87	0.87	2805

4.5 K Nearest neighbour classification

```
In [50]: from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
knn_clf.fit(X_train, y_train)
```

```
Out[50]: KNeighborsClassifier()
```

```
In [51]: y_predKNN = knn_clf.predict(X_test)
print('Accuracy of KNN classifier on test set: {:.2f}'.format(knn_clf.score(X_test, y_t))

Accuracy of KNN classifier on test set: 0.86
```

Accuracy of KNN classifier is 86.00%

```
In [76]: from sklearn import metrics
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test, y_predKNN)
print(confusion_matrix)
```

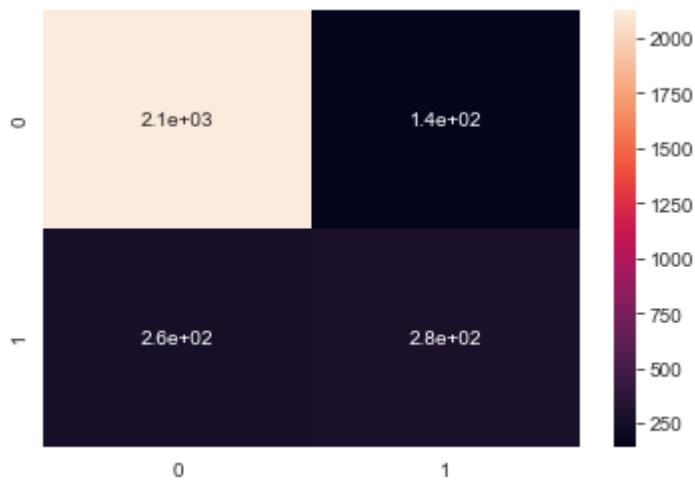
```
[[2123 144]
 [ 260 278]]
```

Correct predictions : $2120 + 277 = 2397$

Incorrect predictions : $261 + 147 = 408$

```
In [77]: sns.heatmap(confusion_matrix, annot=True)
```

```
Out[77]: <AxesSubplot:>
```



```
In [53]: from sklearn.metrics import classification_report
print(classification_report(y_test, y_predKNN))
```

	precision	recall	f1-score	support
0.0	0.89	0.94	0.91	2267
1.0	0.66	0.52	0.58	538
accuracy			0.86	2805
macro avg	0.77	0.73	0.75	2805
weighted avg	0.85	0.86	0.85	2805

Comparing Machine Learning Models

```
In [78]: from sklearn.metrics import roc_auc_score
ml_models = {
```

```
'Logistic Regression': logreg,
'Random Forest': rf_clf,
'XGboost': xgb_clf,
'Decision Tree': DTclf_model,
'KNN': knn_clf,
'SVM': svm_clf
}
```

In [138...]

```
plt.figure(figsize=(12,6))

for model in ml_models:
    print(f"{model.upper()} ACCURACY: {ml_models[model].score(X_test,y_test):.3f}")
    plt.bar(model.upper(),ml_models[model].score(X_test,y_test),label=model.upper(),align='center')
plt.title("ML Model Comparision Based On Accuracy",fontsize=20)
plt.xlabel("MODELS",fontsize=20)
plt.xticks(rotation=90,ha='right')
plt.ylabel("(ACCURACY)SCORE",fontsize=20)
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2,loc ="right")
plt.show()
```

LOGISTIC REGRESSION ACCURACY: 0.896

RANDOM FOREST ACCURACY: 0.870

XGBOOST ACCURACY: 0.880

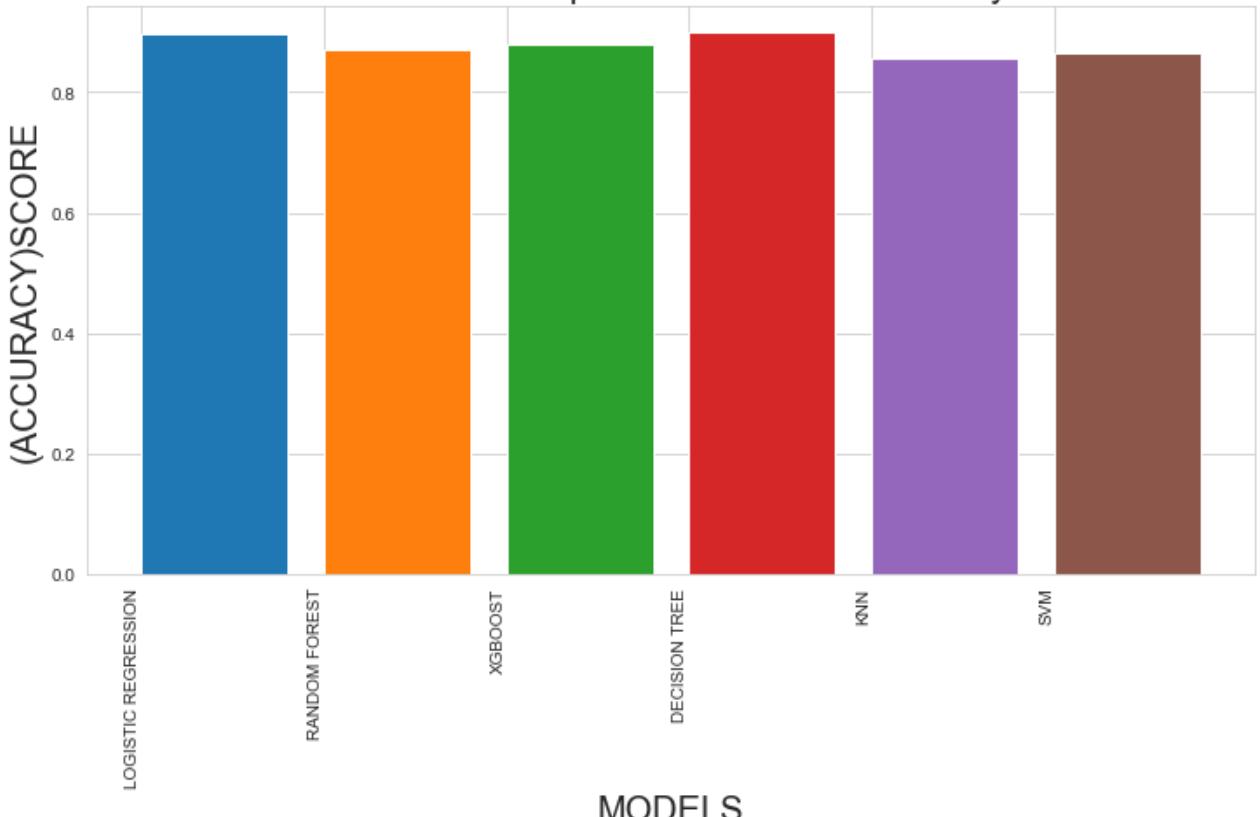
DECISION TREE ACCURACY: 0.899

KNN ACCURACY: 0.856

SVM ACCURACY: 0.865



ML Model Comparision Based On Accuracy



In [137...]

```
plt.figure(figsize=(12,6))
```

```
for model in ml_models:
```

```
print(f"{model.upper()} roc_auc_score: {roc_auc_score(y_test, ml_models[model].predict(X_test))}")
plt.bar(model.upper(),roc_auc_score(y_test, ml_models[model].predict(X_test)),label=model)
plt.title("ML Model Comparision based on AUC Score",fontsize=20)
plt.xlabel("MODELS",fontsize=20)
plt.xticks(rotation=90,ha='right')
plt.ylabel("ROC AUC SCORE(Area under the ROC Curve)",fontsize=20)
plt.legend(bbox_to_anchor =(0.75, 1.15), ncol = 2,loc ="right")
plt.show()
```

LOGISTIC REGRESSION roc_auc_score: 0.737

RANDOM FOREST roc_auc_score: 0.769

XGBOOST roc_auc_score: 0.775

DECISION TREE roc_auc_score: 0.737

KNN roc_auc_score: 0.727

SVM roc_auc_score: 0.648

