**DIRECTING CUSTOMERS TO SUBSCRIPTION THROUGH FINANCIAL APP BEHAVIOR ANALYSIS**

# 1.OBJECTIVE

In today's market many companies have a mobile presence. Often, these companies provide free products/services in their mobile apps in an attempt to transition their customers to a paid membership. Some examples of paid products, which originate from free ones, are YouTube Red, Pandora Premium, Audible Subscription, YouTube Premium, and You Need a Budget. Since marketing efforts are never free, these companies need to know exactly who to target with offers and promotions. The objective of this code is to predict which users will not subscribe to the paid membership, so that greater marketing efforts can go into trying to convert them to paid users.

**2.SOFTWARE DETAIL**

Jupyter notebook,Anaconda Navigator

**3.ABSTRACT /INTRODUCTION**

A distinctive strategy is being offered in response to the challenges mentioned in the existing methodology. The fundamental concept is to utilize machine learning algorithms to predict user behaviour and give an organization the ability to choose precise marketing strategies. Unlike the conventional approaches, this methodology's recommended approach makes use of the latest technologies to accomplish precise targeting and improved conversion rates. The incorporation of several machine learning techniques, such as Logistic Regression and Support Vector Machines (SVM) is one of the methodology's significant innovations. These algorithms provide a strong framework for predictive modelling, making it possible to recognize customers who are less likely to subscribe with more accuracy. For comparison analysis, conventional methods like K nearest neighbors classifier also investigated. In addition, the suggested methodology includes developing a website page that dynamically shows forecasts regarding whether customers are likely to sign up for the premium membership. The procedure is made more interactive and transparent thanks to this user-friendly interface, which improves the user experience. Additionally, the system emphasizes hyperparameter tuning. The machine learning models are optimized for hyperparameters to ensure the best performance. As a result, forecasts get more precise, and marketing tactics become more successful overall. In conclusion, the suggested methodology incorporates distinctive features such numerous sophisticated algorithms, interactive web displays, and hyperparameter adjustment to improve the accuracy and efficiency of the prediction model in addition to addressing the issues raised by the current approach.

# 4.DATASET DESCRIPTION

The app usage data is only from the user's first day in the app. This limitation exists because users can enjoy a 24-hour free trial of the premium features, and the company wants to target them with new offers shortly after the trail is over.

The Data for this project is from manufacturing fields based on trends found in real world case studies. The fields describe what companies usually track from their users.

User : this is Unique id of each particular user of app

first open : this is the date/month/year, time the user frist time open the app

dayofweek : this shows the day out of 7 days a week an user join the app where 0:Sunday & 6:Saturday

hour : This is outoff 24 hour of day the user 1st open the app

age : This is simply the age of the user

screen\_list : This describe the every single screen name the user visited in that 1st 24-hour (screen name seperated by comma)

numscreens : The Number of screen the user visited in 1st 24 hour

minigame : The app has minigame feature, this shows whether the player played any minigame of Not (1:Played, 0: Not Played)

liked : There are like button for each feature in the app, shows whether the user cliked any like button of any feature in app or NOT (1: click like button, 0: Not clicked)

used\_premium\_feature : This shows whether the user used any premium feature (that is for free in 1st 24 hour) or not in 1st 24 hour (1: used, 0: not used)

enrolled : This is target that shows whether the user enrolled to premium after the free trial (1: enrolled, 0: not enrolled)

enrolled\_date : date & time of enrollment to premium product if they enrolled to

# CLASSIFIER EXPLANATION

The classifier used in this code is a K-Nearest Neighbor classifier.

**K-Nearest Neighbor classifier:**

K-Nearest Neighbor classifier can be used for regression as well as for classification but mostly it is used for classification problems. K-Nearest Neighbor classifier is a non-parametric , which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

ACCURACY: 0.7314

**SVM CLASSIFIER (SUPPORT VECTOR MACHINE) MODEL:**

The goal of the svm algorithm is to create the best line or decision boundary that can segregate n dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. Svm chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as support vector machine.

ACCURACY: 0.7680

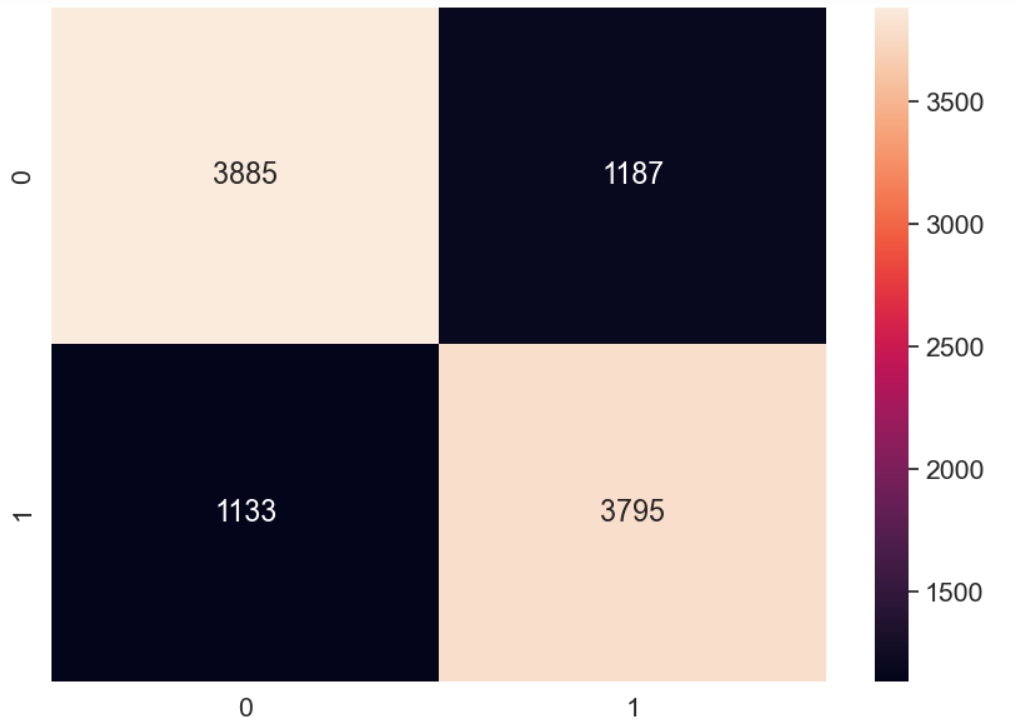
**Logistic regression:**

Logistic regression is much like the linear regression except that how they are used. Linear regression is used for solving regression problems, whereas logistic regression is used for solving the classification problems.

ACCURACY:0.767

# CONFUSION MATRIX

LOGISTIC REGRESSION:

****

There are 3795 instances where the model correctly predicted a positive label when the actual label was positive (True Positive) and 3885 instances where the model correctly predicted a negative label when the actual label was negative (True Negative), Then there are 1187 instances of false positives and there are 1133 instances of false negatives. Finally, we we will plot the confusion matrix, which is a table that gives us the number of predicted values and the number of real values given.

K-NEAREST NEIGHBOR CLASSIFIER:

A screenshot of a computer

Description automatically generated

There are 3616 instances where the model correctly predicted a positive label when the actual label was positive (True Positive) and 3698 instances where the model correctly predicted a negative label when the actual label was negative (True Negative), Then there are 1374 instances of false positives and there are 1312 instances of false negatives. Finally, we we will plot the confusion matrix, which is a table that gives us the number of predicted values and the number of real values given.

# CORRELATION MATRIX

A graph of a number of numbers

Description automatically generated with medium confidence

Now let's build a correlation matrix, which gives us the correlation between each individual variable instead of with the response variable.This will inform us which variables may be linearly dependant on each and will help us in the model building process.

* Hour of the day is negatively correlated, meaning users are more likely to subscribe earlier in the day
* Age is negatively correlated meaning younger people are more likely to subscribe
* The number of screens a user views is strongly correlated with subscribing
* If a user plays the mini-game they're more likely to enroll
* Surprisingly, using the premium features is negatively correlated with enrolment.

# CODE

Importing Required Libraries :

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sn

dataset = pd.read\_csv('appdata10.csv')

Exploratory Data Analysis

dataset.head(10) # Viewing the Data

dataset.describe() # Distribution of Numerical Variables

# First set of Feature cleaning

dataset["hour"] = dataset.hour.str.slice(1, 3).astype(int)

#Plotting

dataset2 = dataset.copy().drop(columns = ['user', 'screen\_list', 'enrolled\_date',

'first\_open', 'enrolled'])

dataset2.head()

## Histograms

plt.subtitle('Histograms of Numerical Columns', fontsize=20)

for i in range(1, dataset2.shape[1] + 1):

plt.subplot(3, 3, i)

f = plt.gca()

# f.axes.get\_yaxis().set\_visible(False)

f.set\_title(dataset2.columns.values[i - 1])

vals = np.size(dataset2.iloc[:, i - 1].unique())

plt.hist(dataset2.iloc[:, i - 1], bins=vals, color='#3F5D7D')

plt.tight\_layout(rect=[0, 0.03, 1, 0.95])

## Correlation with Response Variable

dataset2.corrwith(dataset.enrolled).plot.bar(figsize=(20,10),

title = 'Correlation with Reposnse variable',

fontsize = 15, rot = 45,

grid = True)

Correlation Matrix

sn.set(style="white", font\_scale=1)

# Compute the correlation matrix

corr = dataset2.corr()

# Generate a mask for the upper triangle

mask = np.zeros\_like(corr, dtype=bool)

mask[np.triu\_indices\_from(mask)] = True

# Set up the matplotlib figure

f, ax = plt.subplots(figsize=(18, 15))

f.suptitle("Correlation Matrix", fontsize = 20)

# Generate a custom diverging colormap

cmap = sn.diverging\_palette(220, 10, as\_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio

sn.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,

square=True, linewidths=.5, cbar\_kws={"shrink": .5})

## Feature Engineering

# Formatting Date Columns

dataset.dtypes

dataset["first\_open"] = [parser.parse(row\_date) for row\_date in dataset["first\_open"]]

dataset["enrolled\_date"] = [parser.parse(row\_date) if isinstance(row\_date, str) else row\_date for row\_date in dataset["enrolled\_date"]]

dataset.dtypes

# Selecting Time For Response

dataset["difference"] = (dataset.enrolled\_date-dataset.first\_open).astype('timedelta64[h]')

response\_hist = plt.hist(dataset["difference"].dropna(), color='#3F5D7D')

plt.title('Distribution of Time-Since-Screen-Reached')

plt.show()

plt.hist(dataset["difference"].dropna(), color='#3F5D7D', range = [0, 100])

plt.title('Distribution of Time-Since-Screen-Reached')

plt.show()

dataset.loc[dataset.difference > 48, 'enrolled'] = 0

dataset = dataset.drop(columns=['enrolled\_date', 'difference', 'first\_open'])

## Formatting the screen\_list Field

# Load Top Screens

top\_screens = pd.read\_csv('top\_screens.csv').top\_screens.values

top\_screens

# Mapping Screens to Fields

dataset["screen\_list"] = dataset.screen\_list.astype(str) + ','

for sc in top\_screens:

dataset[sc] = dataset.screen\_list.str.contains(sc).astype(int)

dataset['screen\_list'] = dataset.screen\_list.str.replace(sc+",", "")

dataset['Other'] = dataset.screen\_list.str.count(",")

dataset = dataset.drop(columns=['screen\_list'])

# Funnels

savings\_screens = ["Saving1",

"Saving2",

"Saving2Amount",

"Saving4",

"Saving5",

"Saving6",

"Saving7",

"Saving8",

"Saving9",

"Saving10"]

dataset["SavingCount"] = dataset[savings\_screens].sum(axis=1)

dataset = dataset.drop(columns=savings\_screens)

cm\_screens = ["Credit1",

"Credit2",

"Credit3",

"Credit3Container",

"Credit3Dashboard"]

dataset["CMCount"] = dataset[cm\_screens].sum(axis=1)

dataset = dataset.drop(columns=cm\_screens)

cc\_screens = ["CC1",

"CC1Category",

"CC3"]

dataset["CCCount"] = dataset[cc\_screens].sum(axis=1)

dataset = dataset.drop(columns=cc\_screens)

loan\_screens = ["Loan",

"Loan2",

"Loan3",

"Loan4"]

dataset["LoansCount"] = dataset[loan\_screens].sum(axis=1)

dataset = dataset.drop(columns=loan\_screens)

##Saving Results

dataset.head()

dataset.describe()

dataset.columns

dataset.to\_csv('new\_appdata10.csv', index = False)

import pandas as pd

import numpy as np

import seaborn as sn

import matplotlib.pyplot as plt

import time

dataset = pd.read\_csv('new\_appdata10.csv')

#### Data Pre-Processing ####

# Splitting Independent and Response Variables

response = dataset["enrolled"]

dataset = dataset.drop(columns="enrolled")

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(dataset, response,

test\_size = 0.2,

random\_state = 0)

# Removing Identifiers

train\_identity = X\_train['user']

X\_train = X\_train.drop(columns = ['user'])

test\_identity = X\_test['user']

X\_test = X\_test.drop(columns = ['user'])

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train2 = pd.DataFrame(sc\_X.fit\_transform(X\_train))

X\_test2 = pd.DataFrame(sc\_X.transform(X\_test))

X\_train2.columns = X\_train.columns.values

X\_test2.columns = X\_test.columns.values

X\_train2.index = X\_train.index.values

X\_test2.index = X\_test.index.values

X\_train = X\_train2

X\_test = X\_test2

#### MODEL BUILDING

1ST MODEL: Logistic Regression

# Fitting Model to the Training Set

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0, penalty = 'l2')

classifier.fit(X\_train, y\_train)

# Predicting Test Set

y\_pred = classifier.predict(X\_test)

# Evaluating Results

from sklearn.metrics import confusion\_matrix, accuracy\_score, f1\_score, precision\_score, recall\_score

cm = confusion\_matrix(y\_test, y\_pred)

accuracy\_score(y\_test, y\_pred)

precision\_score(y\_test, y\_pred) # tp / (tp + fp)

recall\_score(y\_test, y\_pred) # tp / (tp + fn)

f1\_score(y\_test, y\_pred)

df\_cm = pd.DataFrame(cm, index = (0, 1), columns = (0, 1))

plt.figure(figsize = (10,7))

sn.set(font\_scale=1.4)

sn.heatmap(df\_cm, annot=True, fmt='g')

print("Test Data Accuracy: %0.4f" % accuracy\_score(y\_test, y\_pred))

2ND MODEL: Support Vector Machine

# Applying k-Fold Cross Validation

from sklearn.model\_selection import cross\_val\_score

accuracies = cross\_val\_score(estimator = classifier, X = X\_train, y = y\_train, cv = 10)

print("SVM Accuracy: %0.3f (+/- %0.3f)" % (accuracies.mean(), accuracies.std() \* 2))

# Analyzing Coefficients

pd.concat([pd.DataFrame(dataset.drop(columns = 'user').columns, columns = ["features"]),

pd.DataFrame(np.transpose(classifier.coef\_), columns = ["coef"])

],axis = 1)

#### Model Tuning ####

## Grid Search (Round 1)

from sklearn.model\_selection import GridSearchCV

# Select Regularization Method

penalty = ['l1', 'l2']

# Create regularization hyperparameter space

C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Combine Parameters

parameters = dict(C=C, penalty=penalty)

grid\_search = GridSearchCV(estimator = classifier,

param\_grid = parameters,

scoring = "accuracy",

cv = 10,

n\_jobs = -1)

t0 = time.time()

grid\_search = grid\_search.fit(X\_train, y\_train)

t1 = time.time()

print("Took %0.2f seconds" % (t1 - t0))

rf\_best\_accuracy = grid\_search.best\_score\_

rf\_best\_parameters = grid\_search.best\_params\_

rf\_best\_accuracy, rf\_best\_parameters

## Grid Search (Round 2)

# Select Regularization Method

penalty = ['l1', 'l2']

# Create regularization hyperparameter space

C = [0.1, 0.5, 0.9, 1, 2, 5]

# Combine Parameters

parameters = dict(C=C, penalty=penalty)

grid\_search = GridSearchCV(estimator = classifier, param\_grid = parameters,

scoring = "accuracy",

cv = 10,

n\_jobs = -1)

t0 = time.time()

grid\_search = grid\_search.fit(X\_train, y\_train)

t1 = time.time()

print("Took %0.2f seconds" % (t1 - t0))

rf\_best\_accuracy = grid\_search.best\_score\_

rf\_best\_parameters = grid\_search.best\_params\_

rf\_best\_accuracy, rf\_best\_parameters

grid\_search.best\_score\_

# Formatting Final Results

final\_results = pd.concat([y\_test, test\_identity], axis = 1).dropna()

final\_results['predicted\_reach'] = y\_pred

final\_results = final\_results[['user', 'enrolled', 'predicted\_reach']].reset\_index(drop=True)

print(final\_results)

3RD MODEL: K-Neighbors

from sklearn.neighbors import KNeighborsClassifier

knn\_model = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2,)

knn\_model.fit(X\_train, y\_train)

y\_pred\_knn = knn\_model.predict(X\_test)

accuracy\_score(y\_test,y\_pred\_knn)

# train with Standert Scaling dataset

knn\_model2 = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2,)

knn\_model2.fit(X\_train, y\_train)

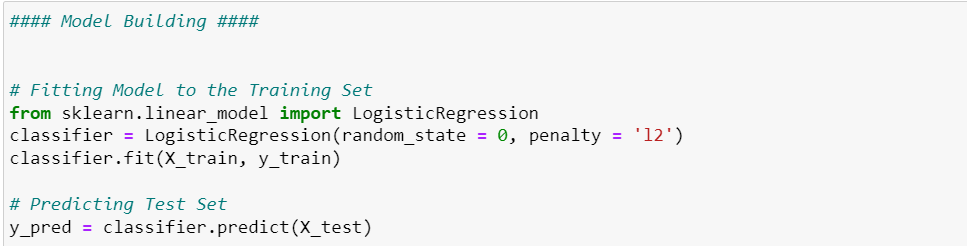
y\_pred\_knn\_sc = knn\_model2.predict(X\_test)

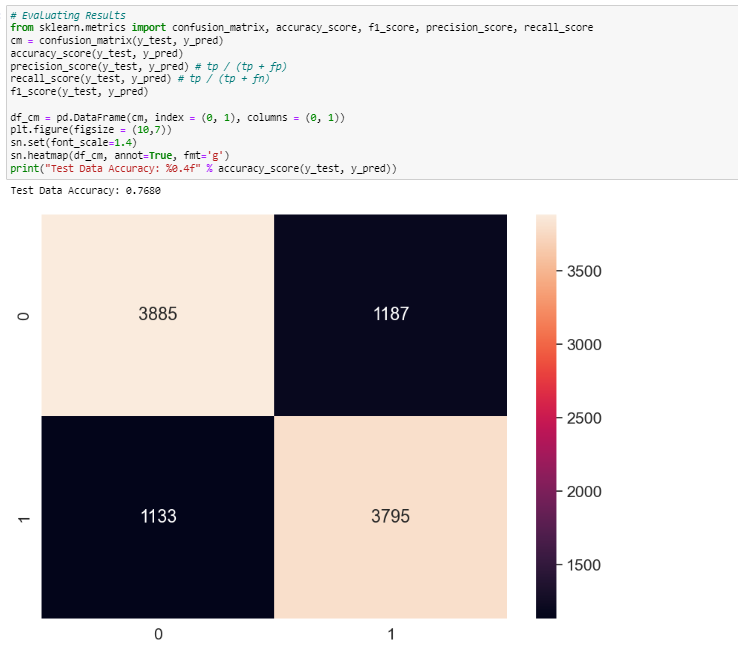
ac=accuracy\_score(y\_test,y\_pred\_knn\_sc)

print("Accuracy Score",ac)

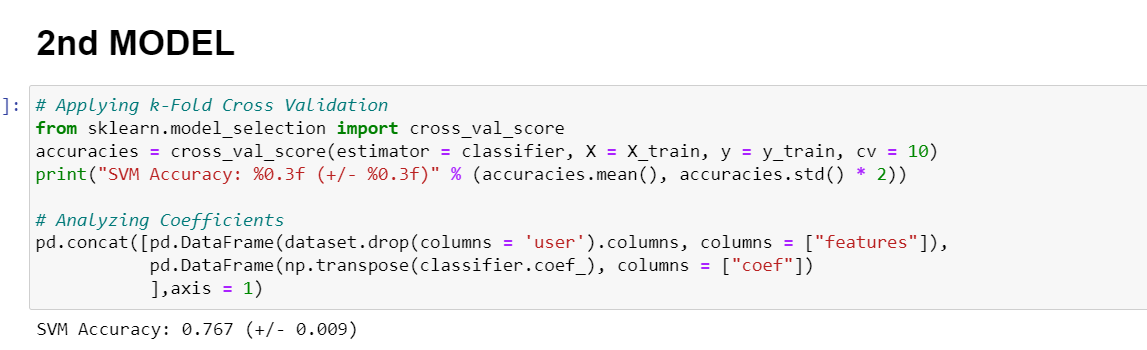
# RESULT

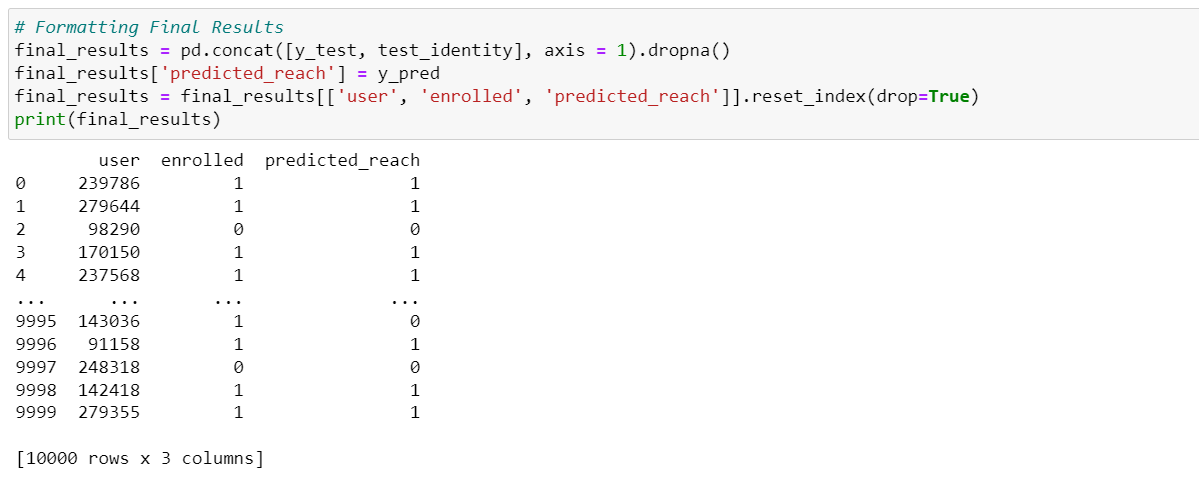
1st Model: Logistic Regression





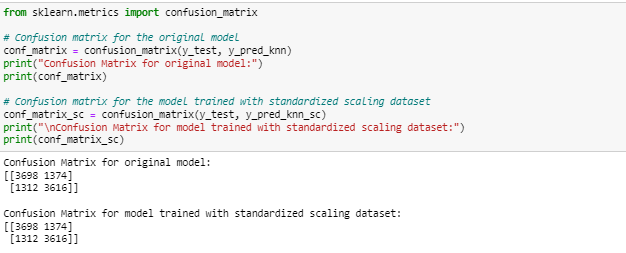
2nd Model: Support Vector Machine

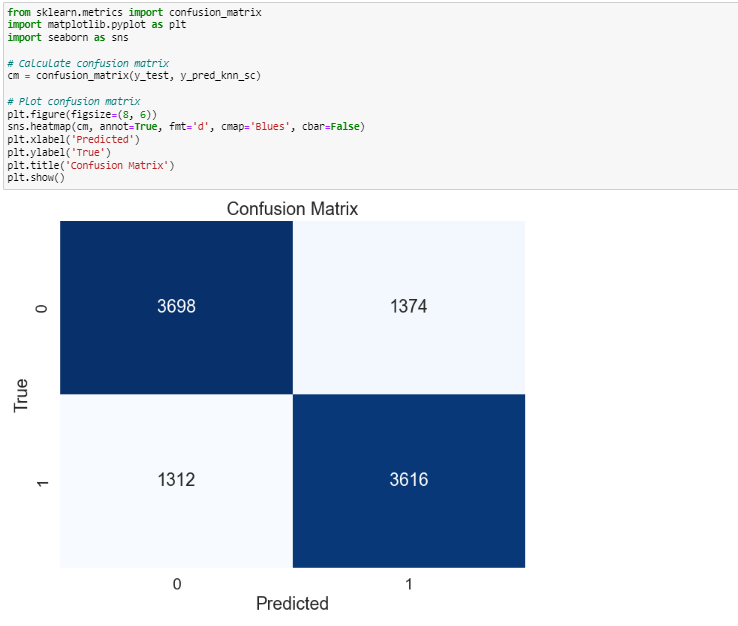




3rd Model: K-Nearest Neighbours







# CONCLUSION

Logistic Regression has the best fit model among the three with accuracy of 0.768 and other models like Support vector machine has the accuracy of 0.767 and K-nearest neighbors has the accuracy of 0.7314.

In today's competitive landscape, effectively directing customers towards subscription is essential for driving revenue growth and fostering long-term user relationships. By leveraging insights from behavioral analysis, financial apps can tailor their subscription strategies to meet evolving user needs and preferences. Adopting a data-driven approach to subscription optimization will be key to maintaining a competitive edge in the market and maximizing app monetization opportunities.

In conclusion, the behavior analysis conducted on customer interactions within the financial app provides valuable insights into directing users towards subscription. Through a comprehensive examination of user demographics, engagement metrics, usage patterns, and conversion funnel analysis, several key findings have emerged