# 

**Predicting successfulness of advertisements**

**Alireza Bolhassani**

Dissertation submitted to International Business School for the partial fulfilment of the requirement for the degree of MASTER OF SCIENCE IN IT FOR BUSINESS DATA ANALYTICS

Table of Contents

[ABSTRACT 3](#_Toc166613962)

[1. Data Collection 4](#_Toc166613963)

[1.1 Groundtruth Dataset 4](#_Toc166613964)

[1.2 Participants Dataset 5](#_Toc166613965)

[1.3 User Behavior Dataset 5](#_Toc166613966)

[• Extract width and height of web page 6](#_Toc166613967)

[• Load CSV File Into DataFrame 6](#_Toc166613968)

[• Extract Time Spent And Date 6](#_Toc166613969)

[• Temporal Features 7](#_Toc166613970)

[• Maximum Time Spent 7](#_Toc166613971)

[• Extract CSV Data Consist Of User Behavior 8](#_Toc166613972)

[2. Data Preprocessing 9](#_Toc166613973)

[2.1 Merge Data frames 9](#_Toc166613974)

[2.2 Country Parameter 10](#_Toc166613975)

[2.3 Category Parameter 12](#_Toc166613976)

[2.4 Convert Categorical Values To Numeral Values 13](#_Toc166613977)

[2.5 Check For Missing Values 13](#_Toc166613978)

[2.6 Filter Data Based On Time Spent 14](#_Toc166613979)

[2.7 Check Value Count Ad Click 14](#_Toc166613980)

[3. Feature Selection 15](#_Toc166613981)

[3.1 Remove Unnecessary Columns 15](#_Toc166613982)

[3.2 Correlation Between Features 15](#_Toc166613983)

[3.3 Correlation Between Features And Target 17](#_Toc166613984)

[3.4 Mutual Information 18](#_Toc166613985)

[4. Machine Learning 19](#_Toc166613986)

[4.1 Functions 19](#_Toc166613987)

[• Confusion plot function 19](#_Toc166613988)

[• AUC curve plot function 20](#_Toc166613989)

[4.2 One Hot Encoding 20](#_Toc166613990)

[4.3 Handling Imbalanced Data 21](#_Toc166613991)

[4.4 Data Train/Test Split 23](#_Toc166613992)

[4.5 Data Scaling 23](#_Toc166613993)

[4.6 Naive Bayes 23](#_Toc166613994)

[4.7 Random Forest Classifier 25](#_Toc166613995)

[4.8 Logistic Regression Model 28](#_Toc166613996)

[4.9 Support Vector Regression (SVR) 31](#_Toc166613997)

[4.10 Convolutional Neural Network Model 33](#_Toc166613998)

[• Plot Learning Curve 38](#_Toc166613999)

[5. Evaluation 39](#_Toc166614000)

[6. Visualizing User Behavior Data 41](#_Toc166614001)

[6.1 Functions 41](#_Toc166614002)

[• Diplay Image Generated 41](#_Toc166614003)

[6.2 First Empty Image 42](#_Toc166614004)

[6.3 Draw Line For mousemove 42](#_Toc166614005)

[6.4 Draw Circle For click 43](#_Toc166614006)

[6.5 Draw Square For ad\_position And ad\_type 44](#_Toc166614007)

[6.6 Convert Image To Numpy Array 45](#_Toc166614008)

[6.7 Clear Session 45](#_Toc166614009)

[6.8 Split Data 46](#_Toc166614010)

[6.9 CNN Model For Images 46](#_Toc166614011)

[7. Conclusion 50](#_Toc166614012)

[Utilizing Two Different Approaches for Predicting User Clicks 50](#_Toc166614013)

[• Parametric Approach with Machine Learning Models: 50](#_Toc166614014)

[• Visualizing User Behavior to Predict Clicks: 50](#_Toc166614015)

[• The Success of Visual Approach: 51](#_Toc166614016)

[• Conclusion: 51](#_Toc166614017)

[8. References 51](#_Toc166614018)

# 

# ABSTRACT

In online advertising, predicting whether a user will click on an ad is crucial for optimizing ad placement and maximizing revenue. In this project, we propose a novel approach to click prediction using Convolutional Neural Networks (CNNs). We use three datasets: the groundtruth dataset containing information about ad clicks, the participants dataset containing information about ad characteristics and user interactions, and the user behavior dataset containing detailed user interactions on a webpage. By merging these datasets, we create a comprehensive dataset for training our CNN model.

We preprocess the data and reshape it to fit the CNN input shape. We then build and train a CNN model using Keras with TensorFlow backend. The model learns to extract relevant features from the user behavior data and predicts whether a user will click on an ad. We evaluate the model using standard classification metrics such as accuracy, precision, recall, and F1-score.

Our experimental results demonstrate that the proposed CNN model outperforms traditional machine learning models for click prediction. The model's ability to learn from raw user behavior data makes it a promising approach for click prediction in online advertising.

# 1. Data Collection

## 1.1 Groundtruth Dataset

This dataset contains information about ad clicks. It includes whether a user clicked on an ad or not (ad\_clicked) and a unique identifier for each log entry (log\_id).

Based on the definition of the project, a series of data cannot be used in the project. We will remove them at this initial stage, although some of them could have effective parameters in the prediction.

***Parameters :***

* ad\_clicked: (int) Whether the participant clicked on the ad (1) or not (0).
* log\_id: (string) Mouse tracking log ID.

# Read the dataframe in pandas format  
import pandas as pd  
  
groundtruth = pd.read\_csv("groundtruth.tsv", delimiter="\t")  
groundtruth = groundtruth.drop(["attention","user\_id"], axis=1)  
groundtruth.head(3)

ad\_clicked log\_id  
0 0 20181002033126  
1 1 20181001211223  
2 0 20181001170952

## 1.2 Participants Dataset

This dataset contains information about ad characteristics and user interactions. It includes details such as the country of the participant (country), ad position (ad\_position), ad type (ad\_type), ad category (ad\_category), SERP (Search Engine Results Page) ID (serp\_id), search query (query), and a unique identifier for each log entry (log\_id).

***Parameters***

* country: (string) Participant's country, in ISO-3 format.
* ad\_position: (string) Ad stimulus position.
* ad\_type: (string) Ad stimulus type.
* ad\_category: (string) Ad stimulus category.
* serp\_id: (string) Ad SERP identifier.
* query: (string) Ad stimulus query.
* log\_id: (string) Mouse tracking log ID.

participants = pd.read\_csv("participants.tsv", delimiter="\t")  
participants = participants.drop(["user\_id","education","age","income","gender"], axis=1)  
participants.head(3)

country ad\_position ad\_type ad\_category serp\_id \  
0 PHL top-left dd Computers & Electronics tablets   
1 VEN top-right dd Shop - Luxury Goods casio-watches   
2 VEN top-left native Shop - Luxury Goods chivas-regal   
  
 query log\_id   
0 tablets 20181002033126   
1 casio watches 20181001211223   
2 chivas regal 20181001170952

## 1.3 User Behavior Dataset

This dataset contains detailed user interactions on a webpage. It includes information such as cursor timestamp (cursor), timestamp (timestamp), cursor position (xpos and ypos), event type (event), XPath of the element (xpath), attributes of the element (attrs), and extra information about the event (extras). The unique identifier for each log entry is the name of the file (log\_id).

We store this information in the log folder in the ‍‍user\_behavior data frame.

### Extract width and height of web page

In the XML file, there is information about the width and length of the document, which we need for normalization. On the other hand, it is better to include this parameter to make each point similar.

import xml.etree.ElementTree as ET  
  
user\_behavior = pd.DataFrame(participants["log\_id"])  
  
user\_behavior["doc\_width"] = 0  
user\_behavior["doc\_height"] = 0  
  
for index,row in user\_behavior.iterrows():  
 tree = ET.parse(f"logs/{row['log\_id']}.xml")  
 root = tree.getroot()  
  
 doc\_w, doc\_h = root.find("document").text.split("x")  
 scr\_w, scr\_h = root.find("screen").text.split("x")  
 user\_behavior.at[index,"doc\_width"] = max(int(doc\_w),int(scr\_w))  
 user\_behavior.at[index,"doc\_height"] = max(int(doc\_h),int(scr\_h))   
  
user\_behavior.head(3)

log\_id doc\_width doc\_height  
0 20181002033126 1366 2064  
1 20181001211223 1366 1611  
2 20181001170952 1366 2284

### Load CSV File Into DataFrame

Since we need this information in several steps, we load and store it in a variable.

import os  
log\_csv\_df = {}  
  
for index,row in user\_behavior.iterrows():  
 file\_path = f"logs/{row['log\_id']}.csv"  
 if not os.path.exists(file\_path):  
 continue   
 log\_csv\_df[row['log\_id']] = pd.read\_csv(file\_path, delimiter=" ")

### Extract Time Spent And Date

One of the important parameters in log files is the time spent on pages. This parameter is not only useful for filtering data, but it can also be used as a prediction parameter.

* ‍‍‍‍time\_spent : Since the given time is in milliseconds, we divide it by thousands.
* date : The Unix Epoch date format is a single number value, representing the number of seconds that have elapsed since 1/1/1970. This is not compatible with Boomi date masking, so converting to and from this date format requires a small script.

user\_behavior["time\_spent"] = 0  
user\_behavior["date"] = 0  
  
for index,row in user\_behavior.iterrows():  
 df2 = log\_csv\_df[row['log\_id']]  
 time\_spent = df2["timestamp"].iloc[-1] - df2["timestamp"].iloc[0]  
  
 user\_behavior.at[index,"time\_spent"] = int(time\_spent/1000)  
 user\_behavior.at[index,"date"] = df2["timestamp"].iloc[0]  
  
user\_behavior.head(3)

log\_id doc\_width doc\_height time\_spent date  
0 20181002033126 1366 2064 96 1538444060776  
1 20181001211223 1366 1611 18 1538421318262  
2 20181001170952 1366 2284 54 1538406739854

### Temporal Features

* Time-related features such as time of day and day of week can provide valuable insights into when users are most likely to click on ads. These features are typically one-hot encoded to capture their cyclic nature.

import time  
  
user\_behavior["week\_day"] = 0  
user\_behavior["hour"] = 0  
  
for index,row in user\_behavior.iterrows():  
 epoch = int(row["date"]) / 1000  
 user\_behavior.at[index,"week\_day"] = time.gmtime(epoch).tm\_wday  
 user\_behavior.at[index,"hour"] = time.gmtime(epoch).tm\_hour  
  
user\_behavior.head(3)

log\_id doc\_width doc\_height time\_spent date week\_day \  
0 20181002033126 1366 2064 96 1538444060776 1   
1 20181001211223 1366 1611 18 1538421318262 0   
2 20181001170952 1366 2284 54 1538406739854 0   
  
 hour   
0 1   
1 19   
2 15

### Maximum Time Spent

max\_time\_spent = max(user\_behavior["time\_spent"])  
max\_time\_spent

5356927

### Extract CSV Data Consist Of User Behavior

In this step, we modeled the user's behavior such as clicking into a two-dimensional array.

We incorporate two key parameters into our array: event type and time. The event type describes the action taken by the user during their session, such as clicking on an ad or navigating to a different page. On the other hand, time provides information about when each event occurred, helping us understand user engagement patterns over time.

**event\_type:** We are faced with six different event models, including ‍click , mouseup, mouseover, mousedown, mousemove, contextmenu, scroll which we have tested four types to record as effective data.

By mapping user behavior to this 2D array, we create a structured representation of user interactions, allowing us to analyze and model user behavior effectively. This structured approach enables us to extract valuable insights, identify patterns, and build predictive models to enhance user experience and engagement on our platform.

**Event Time**

In our user behavior analysis, we introduce the concept of relative time to our dataset. Relative time allows us to understand the temporal aspect of user interactions without relying solely on absolute timestamps.

To achieve this, we calculate the time difference between each event and a reference point, usually the timestamp of the first event in a user session. By subtracting the timestamp of the first event from subsequent timestamps, we obtain relative time values.

import numpy as np  
  
event = {"click":1, "mouseup":2, "mouseover":3, "mousedown":4}  
w = 10  
h = 10  
matrix1 = []  
  
for index1,row1 in user\_behavior.iterrows():  
  
 matrix\_event = np.array([[(0,0) for x in range(w)] for y in range(h)])  
  
 df2 = log\_csv\_df[row1['log\_id']]  
 start\_time = df2.at[0,"timestamp"]  
  
 for index, row in df2.iterrows():  
 xpos = (int(row["xpos"])/row1["doc\_width"]) \* w  
 xpos = min(w-1,int(xpos))  
  
 ypos = (int(row["ypos"])/row1["doc\_height"]) \* h  
 ypos = min(h-1,int(ypos))  
  
 event\_type = event.get(row["event"]) or 0   
 event\_time = int((row["timestamp"] - start\_time)/1000)  
 # event type: click, mouseup, mouseover, mousedown event time  
 matrix\_event[xpos][ypos] = (event\_type, event\_time)  
  
  
 matrix\_event = matrix\_event.flatten()  
 matrix1.append(matrix\_event)  
  
  
# dimension of matrix  
dim = len(matrix1[0])  
matrix1 = pd.DataFrame(matrix1, columns=[f"event{i}" for i in range(dim)])  
  
  
user\_dataset = pd.concat([user\_behavior, matrix1], axis=1)  
  
user\_dataset.head(3)

log\_id doc\_width doc\_height time\_spent date week\_day \  
0 20181002033126 1366 2064 96 1538444060776 1   
1 20181001211223 1366 1611 18 1538421318262 0   
2 20181001170952 1366 2284 54 1538406739854 0   
  
 hour event0 event1 event2 ... event190 event191 event192 event193 \  
0 1 0 96 0 ... 0 0 0 0   
1 19 0 18 0 ... 0 0 0 0   
2 15 0 54 0 ... 0 0 0 0   
  
 event194 event195 event196 event197 event198 event199   
0 0 0 0 0 0 0   
1 0 0 0 0 0 0   
2 0 0 0 0 0 0   
  
[3 rows x 207 columns]

# 2. Data Preprocessing

Data preprocessing is a crucial step in machine learning projects, as it involves transforming raw data into a format that is suitable for modeling. The quality and structure of the data greatly influence the performance of the machine learning models.

## 2.1 Merge Data frames

Merging two dataframes is a fundamental operation in data manipulation, allowing you to combine data from different sources based on common columns. In Python, you can achieve this using the merge() function provided by the Pandas library.

dataset = participants.merge(groundtruth, on="log\_id", how="inner")  
dataset = dataset.merge(user\_dataset, on="log\_id", how="inner")  
dataset.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 2909 entries, 0 to 2908  
Columns: 214 entries, country to event199  
dtypes: int64(208), object(6)  
memory usage: 4.7+ MB

## 2.2 Country Parameter

In this code snippet, I aimed to address the issue of low-value countries in the dataset. Here's what I did:

I identified the countries with low occurrences in the dataset. To do this, I counted the occurrences of each country using the value\_counts() function and stored the indices of countries with fewer than 50 occurrences in the low\_value\_countries variable.

After that, I replaced the low-value country names with "OTHER" in the DataFrame to combine them. I accomplished this using boolean indexing to locate the rows where the country was one of the low-value countries and then replacing those country names with "Other".

This preprocessing step ensures that countries with sparse data are combined into a single category, simplifying the dataset and making it easier to analyze.

dataset["country"].value\_counts()

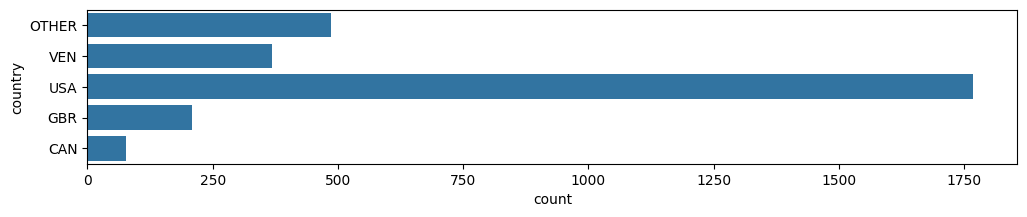
country  
USA 1768  
VEN 368  
GBR 209  
CAN 77  
EGY 38  
 ...   
BOL 1  
DNK 1  
MYS 1  
KWT 1  
HUN 1  
Name: count, Length: 69, dtype: int64

temp = dataset["country"].value\_counts()  
low\_rep = temp[temp < 50].index  
dataset["country"] = dataset["country"].replace(low\_rep, "OTHER")  
dataset["country"].value\_counts()

country  
USA 1768  
OTHER 487  
VEN 368  
GBR 209  
CAN 77  
Name: count, dtype: int64

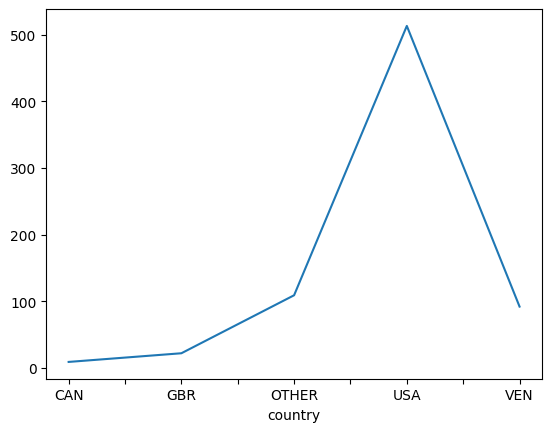
import seaborn as sns  
import matplotlib.pyplot as plt  
  
fig = plt.figure(figsize = (12,2))  
sns.countplot(y ="country", data = dataset)  
dataset["country"].value\_counts()

country  
USA 1768  
OTHER 487  
VEN 368  
GBR 209  
CAN 77  
Name: count, dtype: int64



dataset["country"][dataset["ad\_clicked"] == 1].value\_counts().sort\_index().plot()

<Axes: xlabel='country'>



## 2.3 Category Parameter

Just like the country classification, the categories with low values were combined with each other. We did the same here.

dataset["ad\_category"].value\_counts()

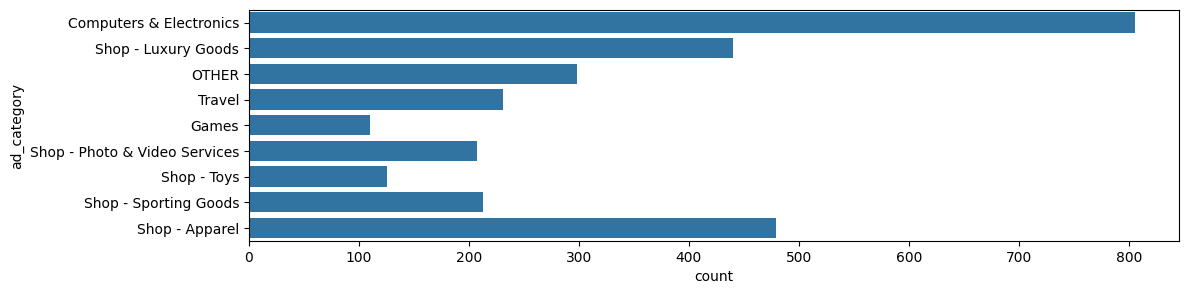
ad\_category  
Computers & Electronics 805  
Shop - Apparel 479  
Shop - Luxury Goods 440  
Travel 231  
Shop - Sporting Goods 213  
Shop - Photo & Video Services 207  
Shop - Toys 126  
Games 110  
Shop - Wholesalers & Liquidatr 87  
Shop - Event Ticket Sales 76  
Autos & Vehicles 62  
Shop - Gifts & Special Event 38  
Food & Drink 18  
Real Estate 17  
Name: count, dtype: int64

temp = dataset["ad\_category"].value\_counts()  
low\_rep = temp[temp < 100].index  
dataset["ad\_category"] = dataset["ad\_category"].replace(low\_rep, "OTHER")  
dataset["ad\_category"].value\_counts()

ad\_category  
Computers & Electronics 805  
Shop - Apparel 479  
Shop - Luxury Goods 440  
OTHER 298  
Travel 231  
Shop - Sporting Goods 213  
Shop - Photo & Video Services 207  
Shop - Toys 126  
Games 110  
Name: count, dtype: int64

import seaborn as sns  
import matplotlib.pyplot as plt  
  
fig = plt.figure(figsize = (12,3))  
sns.countplot(y ="ad\_category", data = dataset)  
dataset["ad\_category"].value\_counts()

ad\_category  
Computers & Electronics 805  
Shop - Apparel 479  
Shop - Luxury Goods 440  
OTHER 298  
Travel 231  
Shop - Sporting Goods 213  
Shop - Photo & Video Services 207  
Shop - Toys 126  
Games 110  
Name: count, dtype: int64



## 2.4 Convert Categorical Values To Numeral Values

To convert categorical values to numerical values, we can use techniques like Label Encoding.

from sklearn.preprocessing import LabelEncoder  
  
# Initialize LabelEncoder  
label\_encoder = LabelEncoder()  
  
categories = ["query", "country", "ad\_type", "ad\_position", "ad\_category", "serp\_id"]  
  
for cat\_name in categories:  
 dataset[cat\_name] = label\_encoder.fit\_transform(dataset[cat\_name])

## 2.5 Check For Missing Values

In data analysis, null values represent missing or undefined data. Dealing with null values is crucial because they can impact the accuracy and reliability of your analysis.

Handling null values correctly is essential for accurate analysis and modeling.

Fortunately, we don't have an null values in the data and it can remain unchanged.

# Count the number of null values in each feature  
num\_null\_value = dataset.isnull().sum()  
  
# Determine the number of unique values in each feature  
num\_unique\_value = dataset.nunique()  
  
# Create a summary dataframe combining null and unique value counts  
df\_summary = pd.concat([num\_null\_value,num\_unique\_value], axis='columns', keys= ['num of nulls', 'num of unique'])  
df\_summary[:7]

num of nulls num of unique  
country 0 5  
ad\_position 0 2  
ad\_type 0 2  
ad\_category 0 9  
serp\_id 0 63  
query 0 55  
log\_id 0 2909

## 2.6 Filter Data Based On Time Spent

To filter users who spent less than 5 seconds on the website, we can use the timestamp and cursor timestamp columns in the user behavior dataset. We'll calculate the time spent by each user on the website and filter out users who spent less than 5 seconds.

dataset = dataset[dataset["time\_spent"] > 5]  
dataset = dataset.reset\_index(drop=True)  
dataset.head(3)

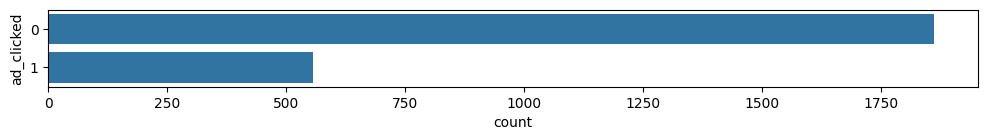
country ad\_position ad\_type ad\_category serp\_id query log\_id \  
0 2 0 0 0 61 53 20181002033126   
1 4 1 0 4 15 14 20181001211223   
2 4 0 1 4 16 15 20181001170952   
  
 ad\_clicked doc\_width doc\_height ... event190 event191 event192 \  
0 0 1366 2064 ... 0 0 0   
1 1 1366 1611 ... 0 0 0   
2 0 1366 2284 ... 0 0 0   
  
 event193 event194 event195 event196 event197 event198 event199   
0 0 0 0 0 0 0 0   
1 0 0 0 0 0 0 0   
2 0 0 0 0 0 0 0   
  
[3 rows x 214 columns]

## 2.7 Check Value Count Ad Click

the class imbalance in the ad\_clicked data could potentially affect the model's performance. With 1861 rows labeled as 0 and only 557 rows labeled as 1, the model may become biased toward predicting the majority class (0).

fig = plt.figure(figsize = (12,1))  
sns.countplot(y ="ad\_clicked", data = dataset)  
dataset["ad\_clicked"].value\_counts()

ad\_clicked  
0 1861  
1 557  
Name: count, dtype: int64



# 3. Feature Selection

Feature Selection is the process of selecting a subset of relevant features for use in machine learning model building.

It is not always the truth that the more data, the better the result will be. Including irrelevant features (the ones that are just unhelpful to the prediction) and redundant features (irrelevant in the presence of others) will only make the learning process overwhelmed and easy to cause overfitting.

By reducing the number of features, we can improve the performance of the machine learning models, while reducing training time and creating more interpretable machine learning models.

## 3.1 Remove Unnecessary Columns

In this step, we preprocess the datasets to remove unnecessary features and prepare them for further analysis.

dataset\_final = dataset.drop(["doc\_width", "doc\_height", "log\_id"], axis=1)  
len(dataset\_final.columns)

211

## 3.2 Correlation Between Features

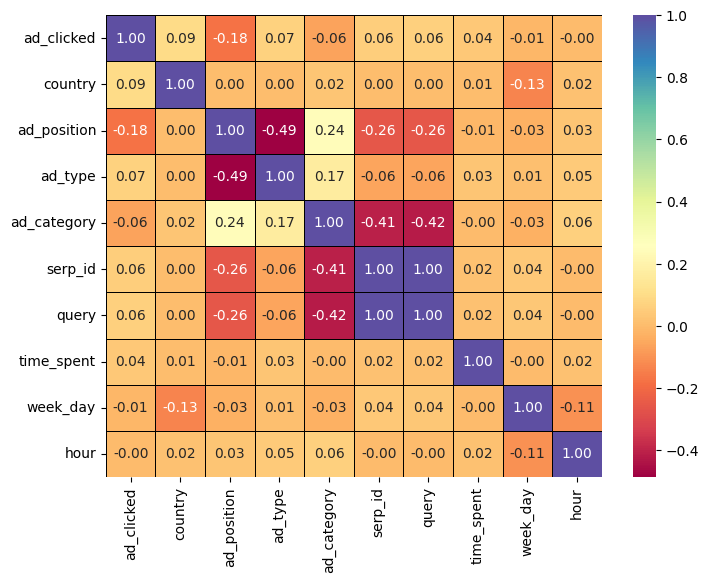
understanding the correlation between features is crucial for several reasons:

1. **Feature Selection:** Correlation analysis helps in identifying redundant features. Features highly correlated with each other might not provide additional information to the model. By removing these features, you can simplify your model, reduce overfitting, and improve computational efficiency.
2. **Interpretability:** Knowing which features are correlated can help in interpreting the model. Highly correlated features might be indicating similar aspects of the data.
3. **Model Performance:** Correlated features can sometimes negatively impact the performance of certain machine learning algorithms, particularly those sensitive to multicollinearity, such as linear and logistic regression.

By analyzing the correlation between features, we can optimize our model by selecting the most relevant features and removing redundant ones.

# pip3 install matplotlib  
import matplotlib.pyplot as plt  
# pip3 install seaborn  
import seaborn as sns  
  
# Create correlation matrix for features  
features\_df = dataset\_final[["ad\_clicked", "country", "ad\_position", "ad\_type", "ad\_category", "serp\_id", "query", "time\_spent", "week\_day", "hour"]]  
feature\_corr = features\_df.corr(numeric\_only=True)  
  
# Display correlation matrix  
display(feature\_corr)  
  
# Plot correlation matrix  
plt.figure(figsize=(8,6))  
sns.heatmap(feature\_corr, annot=True, fmt='0.2f', linewidths=0.5, linecolor='Black', cmap='Spectral')  
plt.show()

ad\_clicked country ad\_position ad\_type ad\_category \  
ad\_clicked 1.000000 0.093932 -0.177069 0.069452 -0.063096   
country 0.093932 1.000000 0.002907 0.004842 0.024643   
ad\_position -0.177069 0.002907 1.000000 -0.487027 0.243832   
ad\_type 0.069452 0.004842 -0.487027 1.000000 0.167319   
ad\_category -0.063096 0.024643 0.243832 0.167319 1.000000   
serp\_id 0.056782 0.002144 -0.258858 -0.060023 -0.407683   
query 0.060100 0.004591 -0.258887 -0.062236 -0.418692   
time\_spent 0.037232 0.006500 -0.014017 0.028772 -0.000238   
week\_day -0.011474 -0.134246 -0.033001 0.009073 -0.030711   
hour -0.001041 0.017579 0.029131 0.045971 0.059191   
  
 serp\_id query time\_spent week\_day hour   
ad\_clicked 0.056782 0.060100 0.037232 -0.011474 -0.001041   
country 0.002144 0.004591 0.006500 -0.134246 0.017579   
ad\_position -0.258858 -0.258887 -0.014017 -0.033001 0.029131   
ad\_type -0.060023 -0.062236 0.028772 0.009073 0.045971   
ad\_category -0.407683 -0.418692 -0.000238 -0.030711 0.059191   
serp\_id 1.000000 0.997231 0.016445 0.035811 -0.001528   
query 0.997231 1.000000 0.016892 0.035628 -0.000571   
time\_spent 0.016445 0.016892 1.000000 -0.001437 0.016220   
week\_day 0.035811 0.035628 -0.001437 1.000000 -0.106229   
hour -0.001528 -0.000571 0.016220 -0.106229 1.000000



Having two parameters with a correlation of 1 can lead to multicollinearity issues when building a predictive model. Multicollinearity occurs when two or more independent variables are highly correlated with each other. This can cause issues in interpreting the model coefficients and can affect the stability and reliability of the model.

In such cases, it's often necessary to remove one of the highly correlated features to address multicollinearity.

dataset\_final = dataset\_final.drop("serp\_id", axis=1)

## 3.3 Correlation Between Features And Target

If two parameters have a correlation of 1 with the target variable (ad\_clicked), it means they are highly correlated and have a strong linear relationship with the target variable. This suggests that these parameters are good predictors of whether an ad will be clicked or not.

# Calculate the correlation of features to target variable  
features\_df = dataset\_final[["ad\_clicked", "country", "ad\_position", "ad\_type", "ad\_category", "query", "time\_spent", "week\_day", "hour"]]  
corr\_to\_target = features\_df.corr(numeric\_only=True)['ad\_clicked']  
  
# Show results in Descending order  
cor\_matrix\_df = pd.DataFrame(corr\_to\_target.abs()).sort\_values( 'ad\_clicked', ascending=False)  
cor\_matrix\_df.drop('ad\_clicked', axis=0, inplace=True)  
cor\_matrix\_df.rename(columns={'ad\_clicked':'Corr'}, inplace=True)  
cor\_matrix\_df

Corr  
ad\_position 0.177069  
country 0.093932  
ad\_type 0.069452  
ad\_category 0.063096  
query 0.060100  
time\_spent 0.037232  
week\_day 0.011474  
hour 0.001041

Since the visit time according to calculations has very little relationship with the target, it is better to remove this parameter.

dataset\_final = dataset\_final.drop("hour", axis=1)

## 3.4 Mutual Information

Mutual information is a measure of the amount of information obtained about one random variable through the other random variable. It measures the amount of information gained about one variable through the other variable.

In the context of feature selection, mutual information can be used to measure the dependency between a feature and the target variable. Features with high mutual information are considered to be more informative for predicting the target variable.

Two benefits to using Mutual Information as feature selector:

* The MI is model neutral, which means the solution can be applied to various kinds of ML models.
* MI solution is fast. <https://towardsdatascience.com/select-features-for-machine-learning-model-with-mutual-information-534fe387d5c8>

In our project, we can use mutual information to select the most relevant features for predicting whether an ad will be clicked or not.

from sklearn.feature\_selection import mutual\_info\_classif  
  
# Selecting features and target variable  
df = dataset\_final[["ad\_clicked","country","ad\_position","ad\_type","ad\_category","query","time\_spent", "week\_day"]]  
  
X = df.drop("ad\_clicked", axis=1) # Features  
y = df["ad\_clicked"] # Target variable  
  
# Calculate mutual information  
mutual\_info = mutual\_info\_classif(X, y)  
  
# Create a DataFrame to display mutual information scores  
mutual\_info\_df = pd.DataFrame(mutual\_info, index=X.columns, columns=["Mutual Information"])  
mutual\_info\_df = mutual\_info\_df.sort\_values(by="Mutual Information", ascending=False)  
  
# Display mutual information scores  
print(mutual\_info\_df)

Mutual Information  
ad\_position 0.015416  
query 0.009839  
time\_spent 0.007435  
ad\_category 0.003759  
ad\_type 0.002137  
country 0.000000  
week\_day 0.000000

This code will calculate the mutual information between each feature and the target variable (ad\_clicked) and display the mutual information scores for each feature. Features with higher mutual information scores are more informative for predicting whether an ad will be clicked or not.

dataset\_final = dataset\_final.drop("week\_day", axis=1)

# 4. Machine Learning

In this section, we applied several machine learning models to predict whether a user will click on an advertisement based on their behavior on the website. The dataset was preprocessed to handle missing values, encode categorical variables, and address class imbalance.

## 4.1 Functions

We have defined the useful functions that we need in the following sections here.

### Confusion plot function

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known.

[[TN FP] [FN TP]]

TN, FP, FN, and TP represent the counts of true negatives, false positives, false negatives, and true positives, respectively.

import seaborn as sns  
import matplotlib.pyplot as plt  
from sklearn.metrics import confusion\_matrix  
  
def confusion\_plot(y\_true, y\_pred):  
 # Assuming y\_true and y\_pred are your actual and predicted binary labels  
 conf\_matrix = confusion\_matrix(y\_true, y\_pred)  
  
 # Plotting the confusion matrix as a heatmap  
 sns.heatmap(conf\_matrix, annot=True, cmap='Blues', fmt='g')  
  
 plt.xlabel('Predicted labels')  
 plt.ylabel('True labels')  
 plt.title('Confusion Matrix')  
 plt.show()

### AUC curve plot function

AUC (Area Under the Curve) is a popular metric used to evaluate the performance of a classification model. The ROC (Receiver Operating Characteristic) curve is a plot of the true positive rate (TPR) against the false positive rate (FPR) for different threshold values. AUC represents the area under the ROC curve.

AUC can be calculated using the trapezoidal rule or by summing the areas of individual trapezoids formed by adjacent points on the ROC curve.

import matplotlib.pyplot as plt  
from sklearn.metrics import auc  
  
def AUC\_plot(y\_true, y\_scores, curve\_func):  
 # Assuming y\_true and y\_scores are your actual binary labels and predicted probabilities  
 precision, recall, \_ = curve\_func(y\_true, y\_scores)  
 auc\_score = auc(recall, precision)  
  
 # Plot precision-recall curve  
 plt.plot(recall, precision, label=f'Precision-Recall curve (AUC = {auc\_score:.2f})', color='b')  
 plt.xlabel('Recall')  
 plt.ylabel('Precision')  
 plt.title('Precision-Recall Curve')  
 plt.legend(loc='best')  
 plt.show()

## 4.2 One Hot Encoding

**Utilizing Categorical Value Encoding Techniques**  
In our project, we employed two common techniques for handling categorical data: converting categorical values to numerical values and one-hot encoding.

* **Converting Categorical Values to Numerical Values:**  
  One of the initial steps in preprocessing our data involved converting categorical values to numerical representations. While this step is not always necessary, it can be beneficial for certain machine learning algorithms that require numerical input. By assigning a unique numerical value to each category, we ensured that the model could interpret and learn from the data effectively.
* **One-Hot Encoding:**  
  However, converting categorical values to numerical representations might introduce unintended ordinal relationships between categories. To avoid this issue and ensure that the model treats each category as independent, we used one-hot encoding. This technique converts categorical variables into a binary matrix where each category is represented by a binary vector. This way, we preserved the categorical nature of the data without introducing any ordinal relationships.
* **The Benefits of One-Hot Encoding:**  
  One-hot encoding not only prevents the model from misinterpreting categorical variables as ordinal but also allows us to include categorical data without biasing the model towards any particular category. It ensures that the model treats all categories equally during training and enables the model to learn complex patterns without any inherent assumptions about the relationships between categories.
* **Conclusion:**  
  By converting categorical values to numerical representations and employing one-hot encoding, we ensured that our machine learning models could effectively utilize categorical data without introducing biases or misinterpretations. These preprocessing techniques were crucial for improving the performance and interpretability of our models, ultimately leading to more accurate predictions.

dataset\_dummies = dataset\_final  
dataset\_dummies = pd.get\_dummies(data=dataset\_final, columns=["country","ad\_category","query"])  
  
y = dataset\_dummies["ad\_clicked"]  
X = dataset\_dummies.drop("ad\_clicked", axis=1)  
  
X = X.astype(int)

## 4.3 Handling Imbalanced Data

* **Handling Imbalanced Data for Better Model Performance** In our project, we encountered the challenge of imbalanced data, where one class of data significantly outnumbered the other. This is a common issue in many machine learning problems, including ours, where the target variable (click/no-click) was highly imbalanced.
* **The Imbalance Issue:**  
  The imbalance in our data could have led to biased model performance, where the model might have favored the majority class, resulting in poor prediction accuracy for the minority class. For example, in our case, a model might have predicted "no-click" for most instances, ignoring the "click" instances altogether.
* **The Solution:**  
  To address this issue, we employed various techniques for handling imbalanced data. One such technique was oversampling the minority class, where we increased the number of instances in the minority class to match the number of instances in the majority class. This ensured that the model was trained on a more balanced dataset.
* **The Impact:**  
  Handling imbalanced data significantly improved the performance of our machine learning models. By ensuring that the model was trained on a balanced dataset, we prevented it from being biased towards the majority class. As a result, our models were able to make more accurate predictions for both the majority and minority classes.

# pip3 install imbalanced-learn  
from imblearn.over\_sampling import SMOTE  
  
# Instantiate SMOTE  
smote = SMOTE(random\_state=42)  
  
# Resample the dataset  
X, y = smote.fit\_resample(X, y)  
  
# Check the class distribution after resampling  
print("After SMOTE:")  
print(pd.Series(y).value\_counts())  
X.head(5)

After SMOTE:  
ad\_clicked  
0 1861  
1 1861  
Name: count, dtype: int64

ad\_position ad\_type time\_spent date event0 event1 event2 \  
0 0 0 96 1538444060776 0 96 0   
1 1 0 18 1538421318262 0 18 0   
2 0 1 54 1538406739854 0 54 0   
3 1 0 56 1538395858574 0 53 0   
4 0 1 7 1538393256694 0 7 0   
  
 event3 event4 event5 ... query\_45 query\_46 query\_47 query\_48 \  
0 0 0 0 ... 0 0 0 0   
1 0 0 0 ... 0 0 0 0   
2 0 0 0 ... 0 0 0 0   
3 49 0 0 ... 0 0 0 0   
4 0 0 0 ... 0 0 0 0   
  
 query\_49 query\_50 query\_51 query\_52 query\_53 query\_54   
0 0 0 0 0 1 0   
1 0 0 0 0 0 0   
2 0 0 0 0 0 0   
3 0 0 0 0 0 0   
4 0 0 0 0 0 0   
  
[5 rows x 273 columns]

## 4.4 Data Train/Test Split

Before training a machine learning model, the data is typically split into training and testing sets. This allows the model to be trained on one set of data and evaluated on another set to assess its performance. Cross-validation techniques may also be used to further validate the model.

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)  
  
print(f'Shape of features data set for ML models => {X.shape}')  
print(f'Shape of target for ML models => {y.shape}')

Shape of features data set for ML models => (3722, 273)  
Shape of target for ML models => (3722,)

## 4.5 Data Scaling

Scaling the data is a crucial preprocessing step in machine learning for several reasons:

1. **Consistent Scale:** Features with different scales might cause issues during model training. By scaling the features, we ensure that all features have the same scale, which can improve the convergence speed of some machine learning algorithms.
2. **Preventing Dominance:** Features with larger scales might dominate those with smaller scales. Scaling prevents this dominance and ensures that each feature contributes equally to the model training process.
3. **Regularization:** Some machine learning algorithms, such as gradient descent-based optimization algorithms, are sensitive to the scale of the input features. Scaling the features can help these algorithms converge more quickly.
4. **Interpretability:** Scaling does not change the relationship between features, but it can make the interpretation of coefficients or feature importance more intuitive.

In our case, we used ***StandardScaler*** from scikit-learn to scale our features. It standardizes features by removing the mean and scaling to unit variance.

from sklearn.preprocessing import StandardScaler  
  
# Feature scaling  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)

## 4.6 Naive Bayes

Naive Bayes is a family of probabilistic algorithms based on Bayes' Theorem with the "naive" assumption of independence between every pair of features. Despite this simplifying assumption, Naive Bayes classifiers have been very successful in many real-world situations, famously in text classification problems such as spam filtering.

**Advantages**

* Naive Bayes is simple and easy to implement.
* It performs well in many complex real-world situations.
* It requires a small amount of training data to estimate the necessary parameters.

**Limitations**

* Naive Bayes assumes that features are independent, which is not always the case in real-world data.
* It can be outperformed by more complex models.
* It's sensitive to the presence of irrelevant features.

from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# Create a Gaussian Naive Bayes classifier  
nb\_classifier = GaussianNB()  
  
# Train the classifier  
nb\_classifier.fit(X\_train\_scaled, y\_train)  
  
# Predict on the test set  
y\_pred = nb\_classifier.predict(X\_test\_scaled)  
  
# Calculate accuracy  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
  
# Classification report  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred))  
  
  
# print(confusion\_matrix(y\_test, y\_pred))

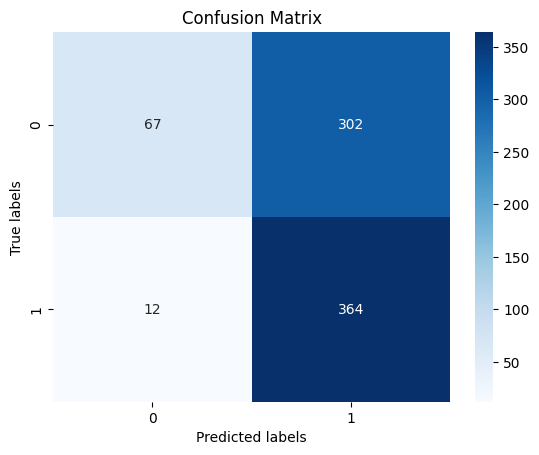
Accuracy: 0.5785234899328859  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.85 0.18 0.30 369  
 1 0.55 0.97 0.70 376  
  
 accuracy 0.58 745  
 macro avg 0.70 0.57 0.50 745  
weighted avg 0.70 0.58 0.50 745

The Naive Bayes model achieved an accuracy of 57%, indicating that it correctly classified 57% of all instances in the dataset.

The precision of the Naive Bayes model varied for different classes. For class 0 (indicating no-click), the model achieved a precision of 85%. This means that when the model predicted no-click, it was correct 85% of the time. On the other hand, for class 1 (indicating click), the precision was 55%, indicating that the model correctly predicted click 55% of the time.

While the Naive Bayes model achieved relatively high precision for class 0, its precision for class 1 was lower. This indicates that the model was better at predicting instances where users did not click on the ads compared to instances where users clicked on the ads.

confusion\_plot(y\_test, y\_pred)



## 4.7 Random Forest Classifier

Random Forest is a popular ensemble learning method used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training time and outputting the mode of the classes (classification) or the average prediction (regression) of the individual trees.

from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# Initializing the RandomForestClassifier  
rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  
  
# Training the model  
rf\_classifier.fit(X\_train\_scaled, y\_train)  
  
# Making predictions  
y\_pred = rf\_classifier.predict(X\_test\_scaled)

The random forest model, as shown below , with a prediction accuracy of 82% on training data.

# Evaluating the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred))

Accuracy: 0.87248322147651  
  
Classification Report:  
 precision recall f1-score support  
  
 0 0.85 0.91 0.88 369  
 1 0.90 0.84 0.87 376  
  
 accuracy 0.87 745  
 macro avg 0.87 0.87 0.87 745  
weighted avg 0.87 0.87 0.87 745

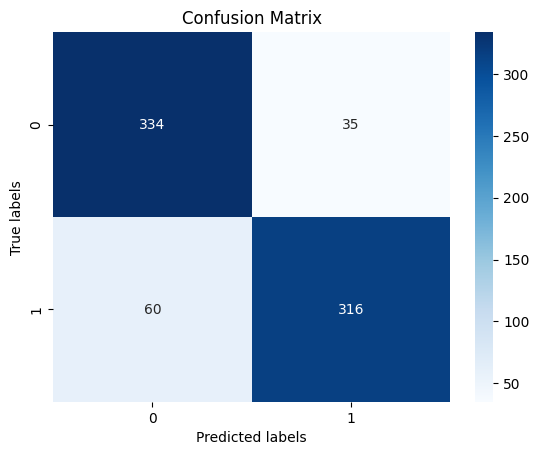
The Random Forest model achieved an accuracy of 87%, indicating that it correctly classified 87% of all instances in the dataset.

The precision of the Random Forest model varied for different classes. For class 0 (indicating no-click), the model achieved a precision of 85%. This means that when the model predicted no-click, it was correct 85% of the time. On the other hand, for class 1 (indicating click), the precision was 90%, indicating that the model correctly predicted click 90% of the time.

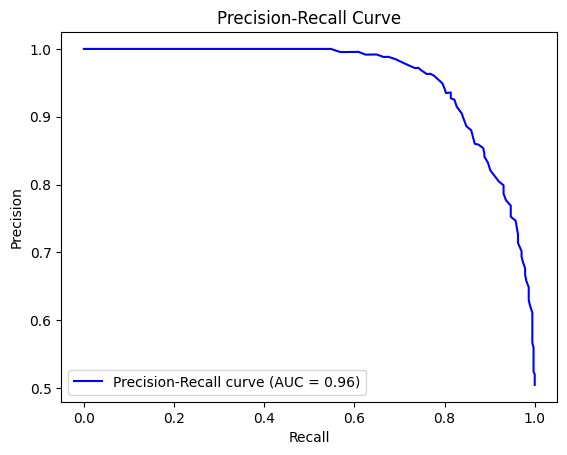
The Random Forest model showed high precision for both classes, with a slightly higher precision for class 1 compared to class 0. This indicates that the model was effective at predicting both instances where users did not click on the ads and instances where users clicked on the ads.

Overall, the Random Forest model demonstrated strong performance in predicting user clicks on ads. With an accuracy of 87% and high precision for both classes, the model effectively differentiated between instances of user clicks and no-clicks on the ads.

confusion\_plot(y\_test, y\_pred)



from sklearn.metrics import precision\_recall\_curve  
  
# Calculate y scores  
y\_scores = rf\_classifier.predict\_proba(X\_test\_scaled)[:, 1]  
AUC\_plot(y\_test, y\_scores, precision\_recall\_curve)



## 4.8 Logistic Regression Model

Logistic regression is a statistical method used for binary classification tasks. Unlike linear regression, which predicts continuous values, logistic regression predicts the probability of an instance belonging to a particular class. It models the probability of the default class (usually labeled as 1) using the logistic function, also known as the sigmoid function.

from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# Initialize the model  
logestic\_model = LogisticRegression()  
  
# Train the model  
logestic\_model.fit(X\_train\_scaled, y\_train)  
  
# Make predictions  
y\_pred = logestic\_model.predict(X\_test\_scaled)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
  
print("Classification Report:")  
print(classification\_report(y\_test, y\_pred))

Accuracy: 0.8040268456375839  
Classification Report:  
 precision recall f1-score support  
  
 0 0.76 0.88 0.82 369  
 1 0.86 0.73 0.79 376  
  
 accuracy 0.80 745  
 macro avg 0.81 0.80 0.80 745  
weighted avg 0.81 0.80 0.80 745

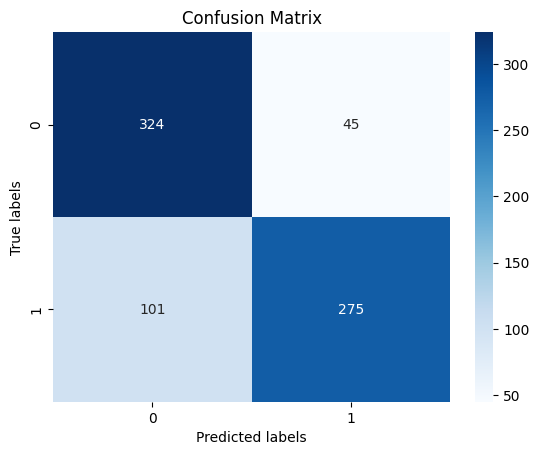
The Logistic Regression model achieved an accuracy of 80%, indicating that it correctly classified 80% of all instances in the dataset.

The precision of the Logistic Regression model varied for different classes. For class 0 (indicating no-click), the model achieved a precision of 76%. This means that when the model predicted no-click, it was correct 76% of the time. On the other hand, for class 1 (indicating click), the precision was 86%, indicating that the model correctly predicted click 86% of the time.

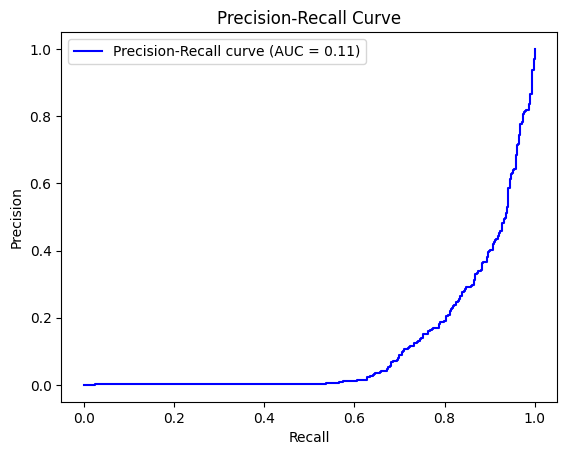
The Logistic Regression model demonstrated high precision for both classes, with a slightly higher precision for class 1 compared to class 0. This indicates that the model was effective at predicting both instances where users did not click on the ads and instances where users clicked on the ads.

In conclusion, the Logistic Regression model showed strong performance in predicting user clicks on ads. With an accuracy of 80% and high precision for both classes, the model effectively differentiated between instances of user clicks and no-clicks on the ads.

confusion\_plot(y\_test, y\_pred)



from sklearn.metrics import roc\_curve  
  
# Calculate y scores  
y\_scores = logestic\_model.predict\_proba(X\_test\_scaled)[:, 1]  
AUC\_plot(y\_test, y\_scores, roc\_curve)



## 4.9 Support Vector Regression (SVR)

Support Vector Regression (SVR) is a type of support vector machine (SVM) algorithm that is used for regression analysis. Similar to SVM for classification, SVR is based on the concept of finding a hyperplane that best fits the data. However, instead of finding the hyperplane that best separates the classes, SVR finds the hyperplane that best fits the data points within a specified margin, called ε-insensitive tube.

In this code, we implement a Support Vector Machine (SVM) model using the sigmoid kernel for binary classification. SVMs are powerful supervised learning models used for classification and regression tasks. The sigmoid kernel is particularly useful when the data is not linearly separable.

from sklearn.svm import SVC  
from sklearn.metrics import accuracy\_score, classification\_report  
  
# Create SVC model  
model = SVC(kernel='sigmoid')  
  
# Train the model  
model.fit(X\_train\_scaled, y\_train)  
  
# Make predictions  
y\_test\_pred = model.predict(X\_test\_scaled)  
  
# Evaluate the model  
test\_accuracy = accuracy\_score(y\_test, y\_test\_pred)  
  
print("Accuracy:", test\_accuracy)  
  
print("Classification Report:")  
print(classification\_report(y\_test, y\_test\_pred))

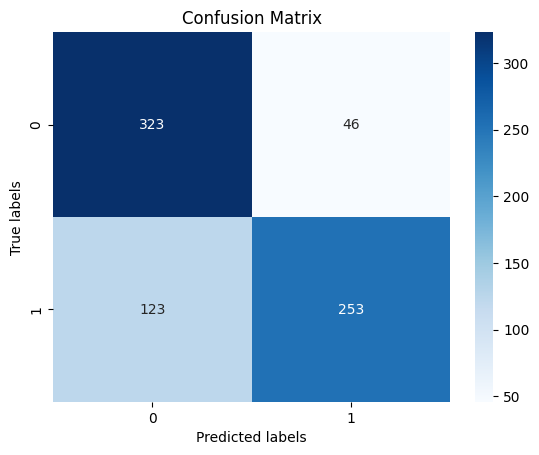
Accuracy: 0.7731543624161074  
Classification Report:  
 precision recall f1-score support  
  
 0 0.72 0.88 0.79 369  
 1 0.85 0.67 0.75 376  
  
 accuracy 0.77 745  
 macro avg 0.79 0.77 0.77 745  
weighted avg 0.79 0.77 0.77 745

The SVR model achieved an accuracy of 77%, indicating that it correctly classified 77% of all instances in the dataset.

The precision of the SVR model varied for different classes. For class 0 (indicating no-click), the model achieved a precision of 72%. This means that when the model predicted no-click, it was correct 72% of the time. Conversely, for class 1 (indicating click), the precision was 85%, indicating that the model correctly predicted click 85% of the time.

The SVR model demonstrated a relatively balanced precision for both classes, although it showed a slightly higher precision for class 1 compared to class 0. This suggests that the model was effective at predicting instances where users clicked on the ads, with a slightly lower precision for instances where users did not click.

confusion\_plot(y\_test, y\_test\_pred)



## 4.10 Convolutional Neural Network Model

A Convolutional Neural Network (CNN) is a type of deep learning model that is widely used for image classification, object detection, and other tasks related to image analysis.

Structure of a CNN

1. Convolutional Layers: These layers consist of a set of filters (kernels) that are convolved with the input data to produce feature maps. Each filter captures different features from the input image.
2. Pooling Layers: Pooling layers are used to reduce the spatial dimensions of the feature maps while retaining important information. The most common type of pooling is max pooling, where the maximum value within a small window is retained.
3. Fully Connected Layers: After several convolutional and pooling layers, the feature maps are flattened and passed through one or more fully connected layers. These layers act as a classifier and make predictions based on the features extracted by the convolutional layers.
4. Activation Functions: Activation functions introduce non-linearity to the model. Common activation functions include ReLU (Rectified Linear Unit) and Sigmoid.

***in this project :***

* We create a Sequential model.
* We add two convolutional layers with ReLU activation and max pooling layers to extract features from the input images.
* We add a flatten layer to convert the 2D features into a 1D vector.
* We add a dense layer with ReLU activation and a dropout layer to prevent overfitting.
* Finally, we add an output layer with one neuron and a sigmoid activation function for binary classification.

from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout, Input  
from tensorflow.keras.callbacks import ModelCheckpoint  
from tensorflow.keras.callbacks import EarlyStopping  
  
  
# Reshape the input data to 3D (batch\_size, timesteps, input\_dim)  
X\_train\_reshaped = X\_train\_scaled.reshape(X\_train\_scaled.shape[0], X\_train\_scaled.shape[1], 1)  
  
# Define input shape  
input\_shape = (X\_train\_reshaped.shape[1], X\_train\_reshaped.shape[2])  
  
# Create CNN model , input\_shape=(X\_trainVal\_reshaped.shape[1], 1))  
model = Sequential()  
model.add(Input(shape=input\_shape))  
model.add(Conv1D(32, kernel\_size=3, activation='relu'))  
model.add(MaxPooling1D(pool\_size=2))  
model.add(Flatten())  
model.add(Dense(128, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(1, activation='sigmoid'))  
  
# Compile the model  
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
  
model.summary()

Model: "sequential\_2"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓  
┃ Layer (type) ┃ Output Shape ┃ Param # ┃  
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩  
│ conv1d\_1 (Conv1D) │ (None, 271, 32) │ 128 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling1d\_1 (MaxPooling1D) │ (None, 135, 32) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ flatten\_2 (Flatten) │ (None, 4320) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense\_4 (Dense) │ (None, 128) │ 553,088 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dropout\_1 (Dropout) │ (None, 128) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense\_5 (Dense) │ (None, 1) │ 129 │  
└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 553,345 (2.11 MB)

Trainable params: 553,345 (2.11 MB)

Non-trainable params: 0 (0.00 B)

checking = ModelCheckpoint('my\_model.keras', save\_best\_only=True, monitor='val\_accuracy', mode='max', verbose=1)  
  
early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)  
  
# Train the model  
history = model.fit(X\_train\_reshaped, y\_train, epochs=10, batch\_size=32, validation\_split=0.2, callbacks=[early\_stop, checking])

Epoch 1/10  
75/75 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.6919 - loss: 0.5925  
Epoch 1: val\_accuracy improved from -inf to 0.82047, saving model to my\_model.keras  
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 21ms/step - accuracy: 0.6925 - loss: 0.5917 - val\_accuracy: 0.8205 - val\_loss: 0.4296  
Epoch 2/10  
72/75 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8076 - loss: 0.4110

Epoch 2: val\_accuracy improved from 0.82047 to 0.83054, saving model to my\_model.keras  
75/75 ━━━━━━━━━━━━━━━━━━━━ 1s 19ms/step - accuracy: 0.8082 - loss: 0.4101 - val\_accuracy: 0.8305 - val\_loss: 0.3777  
Epoch 3/10  
74/75 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8304 - loss: 0.3700  
Epoch 3: val\_accuracy improved from 0.83054 to 0.84228, saving model to my\_model.keras  
75/75 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.8306 - loss: 0.3697 - val\_accuracy: 0.8423 - val\_loss: 0.3881  
Epoch 4/10  
74/75 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.8367 - loss: 0.3631  
Epoch 4: val\_accuracy did not improve from 0.84228  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8369 - loss: 0.3626 - val\_accuracy: 0.8406 - val\_loss: 0.3740  
Epoch 5/10  
73/75 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.8607 - loss: 0.3151  
Epoch 5: val\_accuracy improved from 0.84228 to 0.84396, saving model to my\_model.keras  
75/75 ━━━━━━━━━━━━━━━━━━━━ 3s 36ms/step - accuracy: 0.8609 - loss: 0.3148 - val\_accuracy: 0.8440 - val\_loss: 0.3785  
Epoch 6/10  
74/75 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8602 - loss: 0.3142  
Epoch 6: val\_accuracy improved from 0.84396 to 0.85738, saving model to my\_model.keras  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8603 - loss: 0.3137 - val\_accuracy: 0.8574 - val\_loss: 0.3668  
Epoch 7/10  
72/75 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8788 - loss: 0.2800  
Epoch 7: val\_accuracy did not improve from 0.85738  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 23ms/step - accuracy: 0.8788 - loss: 0.2793 - val\_accuracy: 0.8490 - val\_loss: 0.3580  
Epoch 8/10  
72/75 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8793 - loss: 0.2672  
Epoch 8: val\_accuracy did not improve from 0.85738  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8796 - loss: 0.2668 - val\_accuracy: 0.8440 - val\_loss: 0.3735  
Epoch 9/10  
74/75 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.8877 - loss: 0.2577  
Epoch 9: val\_accuracy did not improve from 0.85738  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.8879 - loss: 0.2574 - val\_accuracy: 0.8574 - val\_loss: 0.3743  
Epoch 10/10  
73/75 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.8913 - loss: 0.2355  
Epoch 10: val\_accuracy did not improve from 0.85738  
75/75 ━━━━━━━━━━━━━━━━━━━━ 2s 24ms/step - accuracy: 0.8915 - loss: 0.2352 - val\_accuracy: 0.8507 - val\_loss: 0.3734

X\_test\_reshaped = X\_test\_scaled.reshape(X\_test\_scaled.shape[0], X\_test\_scaled.shape[1], 1)  
  
# Evaluate the model  
y\_test\_pred = model.predict(X\_test\_reshaped)  
y\_test\_pred = (y\_test\_pred > 0.5).astype(“int32”)  
  
accuracy = accuracy\_score(y\_test, y\_test\_pred)  
  
print(“Test Accuracy:”, accuracy)  
print(classification\_report(y\_test, y\_test\_pred))

23/24 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step

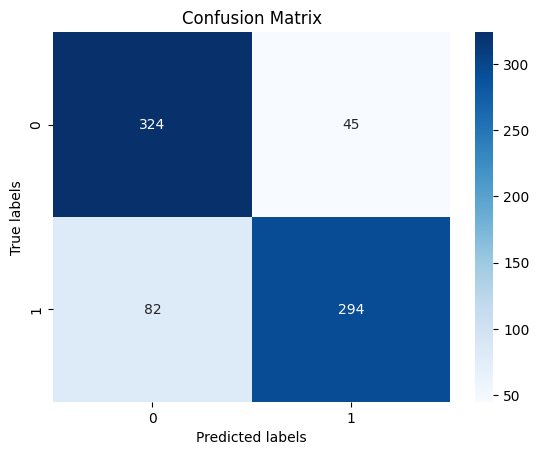
24/24 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step  
Test Accuracy: 0.8295302013422818  
 precision recall f1-score support  
  
 0 0.80 0.88 0.84 369  
 1 0.87 0.78 0.82 376  
  
 accuracy 0.83 745  
 macro avg 0.83 0.83 0.83 745  
weighted avg 0.83 0.83 0.83 745

The CNN model achieved an accuracy of 82%, indicating that it correctly classified 82% of all instances in the dataset.

The precision of the CNN model varied for different classes. For class 0 (indicating no-click), the model achieved a precision of 80%. This means that when the model predicted no-click, it was correct 80% of the time. Conversely, for class 1 (indicating click), the precision was 85%, indicating that the model correctly predicted click 87% of the time.

The CNN model demonstrated a slightly higher precision for class 1 (click) compared to class 0 (no-click). This suggests that the model was more effective at predicting instances where users clicked on the ads compared to instances where users did not click.

Confusion\_plot(y\_test, y\_test\_pred)



### Plot Learning Curve

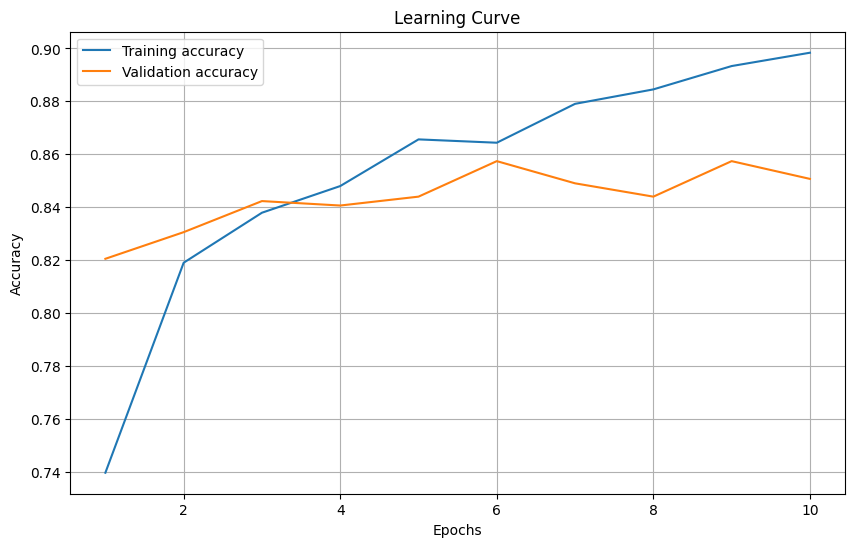
* In this project, we implemented a learning curve plot to visualize the performance of our machine learning model. A learning curve shows the training and validation scores (e.g., accuracy, loss) of a model on the training dataset as a function of the number of training examples. This helps us understand how well our model is learning as we increase the size of the training dataset.

To create the learning curve, we used the learning\_curve function from scikit-learn. However, since our model is implemented using Keras, we encountered an error because the Sequential model does not have a score method required by the learning\_curve function.

To resolve this issue, we modified the code to use the history object returned by the fit method of our Keras model. The history object contains the training and validation metrics for each epoch. We then plotted the training and validation accuracy against the number of epochs using matplotlib.

By visualizing the learning curve, we gained insights into how well our model is learning from the training data and whether it is overfitting or underfitting. This helps us make informed decisions about model training and optimization.

Import matplotlib.pyplot as plt  
  
train\_scores = history.history[‘accuracy’]  
validation\_scores = history.history[‘val\_accuracy’]  
epochs = range(1, len(train\_scores) + 1)  
  
plt.figure(figsize=(10, 6))  
plt.plot(epochs, train\_scores, label=’Training accuracy’)  
plt.plot(epochs, validation\_scores, label=’Validation accuracy’)  
plt.xlabel(‘Epochs’)  
plt.ylabel(‘Accuracy’)  
plt.title(‘Learning Curve’)  
plt.legend(loc=’best’)  
plt.grid()  
plt.show()



# Evaluation

In this project I use 5 different method to predict the target:

| Models | Accuracy % | Precision 0 | Precision 1 |
| --- | --- | --- | --- |
| 1. I Bayes | 57 | 85 | 55 |
| 2. Random Forest Classifier | 87 | 85 | 90 |
| 3. Logistic Regression Model | 80 | 76 | 86 |
| 4. Support Vector Regression (SVR) | 77 | 72 | 85 |
| 5. Convolutional Neural Network Model | 82 | 72 | 85 |

Accuracy is a commonly used metric to evaluate the performance of machine learning models. However, it may not always provide a comprehensive understanding of a model’s performance, especially when dealing with imbalanced datasets, like in our case.

Therefore, we must consider other metrics such as precision, recall, and F1-score, which provide insights into a model’s ability to correctly classify both classes. Precision measures the number of true positives divided by the total number of true positives and false positives. Recall, on the other hand, measures the number of true positives divided by the total number of true positives and false negatives. The F1-score is the harmonic mean of precision and recall.

In our project, we used five different models to predict clicks on ads: I Bayes, Random Forest, Logistic Regression, Support Vector Regression (SVR), and Convolutional Neural Network (CNN). While accuracy is a valuable metric, it doesn’t tell the full story. For example, in an imbalanced dataset where one class is significantly more prevalent than the other, a model that predicts the majority class for every instance might achieve a high accuracy, but it fails to predict the minority class.

***I Bayes*** The I Bayes classifier achieved an accuracy of 57%, with a precision of 85% for class 0 (no-click) and 55% for class 1 (click). Despite its simplicity, this model performed reasonably well, particularly in predicting instances where users did not click on the ads.

***The Random Forest Classifier*** The Random Forest classifier demonstrated significant improvement over the I Bayes model, with an accuracy of 87%. It showed balanced precision for both classes, with 85% for class 0 and 90% for class 1, indicating a robust performance in predicting both click and no-click instances.

***Logistic Regression*** The Logistic Regression model achieved an accuracy of 80%, with a precision of 76% for class 0 and 86% for class 1. While the precision for click instances was high, the model showed slightly lower precision for no-click instances compared to the Random Forest model.

***SVR*** The Support Vector Regression (SVR) model achieved an accuracy of 77%, with a precision of 72% for class 0 and 85% for class 1. Despite its lower accuracy compared to other models, SVR showed a balanced precision for both classes, making it a reliable choice for predicting user clicks on ads.

***The CNN model*** The CNN model demonstrated promising results, with an accuracy of 82%. It showed balanced precision for both click and no-click instances, with 72% for class 0 and 85% for class 1. The CNN model’s performance suggests its effectiveness in distinguishing between click and no-click instances.

In conclusion, the Random Forest classifier outperformed other models with an accuracy of 87% and balanced precision for both click and no-click instances. However, the CNN model also showed promising results, indicating the potential of deep learning approaches in predicting user clicks on ads.

# Visualizing User Behavior Data

In this project, I created an image representation of user behavior data to visualize how users interact with online ads.

To achieve this, I mapped user behavior data onto a two-dimensional grid representing the dimensions of the webpage. Each cell in the grid corresponds to a specific region on the webpage, and the intensity of the color in each cell indicates the time.

Using the Python Imaging Library (PIL), I converted the grid into an image, which was then used as input for a Convolutional Neural Network (CNN) model.

The CNN model was designed to learn patterns and features from these images to predict whether a user would click on an ad or not.

By analyzing the CNN model’s predictions, I gained valuable insights into how different regions of the webpage contribute to user engagement and ad click-through rates.

***Visualizing user behavior data helps us:***

* Identify common user paths and navigation patterns.
* Detect areas of the website that receive the most attention.
* Analyze the effectiveness of website features and layout.
* Identify user interaction anomalies or usability issues.

## 6.1 Functions

We have defined the useful functions that we need in the following sections here.

### Diplay Image Generated

At each stage, a feature is added to the image that we need to display with the help of this function.

Import matplotlib.pyplot as plt  
  
def display\_images(images,target):  
 names = [“Success”, “Fail”]  
 keys = list(images.keys())  
 random = np.random.randint(len(images), size=5)  
 plt.figure(figsize=(12,8))  
 I = 1  
 for n in random:  
 plt.subplot(1, 5, i)  
 plt.xticks([])  
 plt.yticks([])  
 plt.grid(False)  
 plt.imshow(images[keys[n]], cmap=plt.cm.binary)  
 plt.xlabel(names[target[keys[n]]])  
 I += 1  
 plt.show()

## 6.2 First Empty Image

We’re using the PIL (Python Imaging Library) to create images that represent user behavior. Each movement is plotted as a line on a white canvas, providing a visual representation of how users interact with the website.

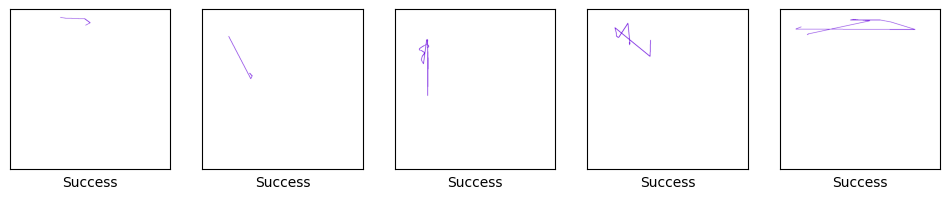
From PIL import Image, ImageDraw  
  
# Create a new image with white background  
image\_width = 800  
image\_height = 800  
  
user\_images = {}  
image\_ad\_clicked = {}  
  
for index,row in dataset.iterrows():  
 img = Image.new(“RGB”, (image\_width, image\_height), “white”)  
 draw = ImageDraw.Draw(img)  
 user\_images[row[“log\_id”]] = img  
 image\_ad\_clicked[row[“log\_id”]] = row[“ad\_clicked”]

## 

## 6.3 Draw Line For mousemove

This code will create an image where each movement is represented by a line connecting consecutive points based on the sample user behavior data.

for index,row in dataset.iterrows():  
 log\_id = row["log\_id"]  
 df = log\_csv\_df[log\_id]  
 doc\_width = row["doc\_width"]  
 doc\_height = row["doc\_height"]  
 prev\_point = False  
  
 img = user\_images[row["log\_id"]]  
 # Draw a line  
 draw = ImageDraw.Draw(img)  
  
 for index2,row2 in df.iterrows():  
 xpos = int((row2["xpos"]/doc\_width) \* image\_width)  
 ypos = int((row2["ypos"]/doc\_height) \* image\_height)  
 if row2["event"] == "mousemove":  
 if not prev\_point:  
 prev\_point = (xpos, ypos)  
 else:  
 point = (xpos, ypos)  
 draw.line([prev\_point, point], fill=(100,0,220), width=3)  
 prev\_point = point  
  
display\_images(user\_images,image\_ad\_clicked)



## 

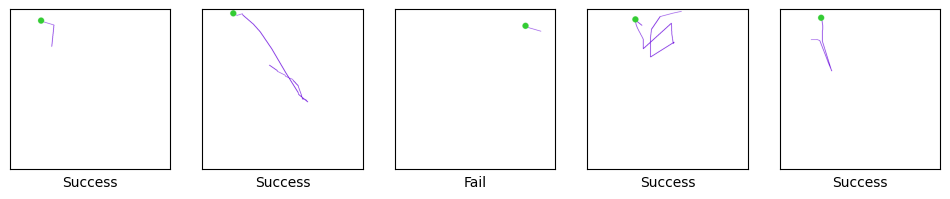
## 6.4 Draw Circle For click

By visualizing user clicks, we can gain insights into which areas of the webpage are most attractive or engaging to users.

We Draw every click position by a circle.

Note that we have considered three different events such as mousedown, mouseup, click as clicks.

for index,row in dataset.iterrows():  
 log\_id = row["log\_id"]  
 df = log\_csv\_df[log\_id]  
 doc\_width = row["doc\_width"]  
 doc\_height = row["doc\_height"]  
 prev\_point = False  
  
 img = user\_images[row["log\_id"]]  
 # Draw a dot  
 draw = ImageDraw.Draw(img)  
 radius = 15  
 color = (51, 204, 51)  
 for index2,row2 in df.iterrows():  
 xpos = int((row2["xpos"]/doc\_width) \* image\_width)  
 ypos = int((row2["ypos"]/doc\_height) \* image\_height)  
 if row2["event"] in ["click","mouseup","mousedown"]:  
 draw.ellipse((xpos-radius, ypos-radius, xpos+radius, ypos+radius), fill=color)  
  
  
display\_images(user\_images,image\_ad\_clicked)



## 

## 6.5 Draw Square For ad\_position And ad\_type

In our project, we aim to visualize the ad positions and types for each row. To achieve this, we utilize the ad\_position and ad\_type parameters from our Dataset.

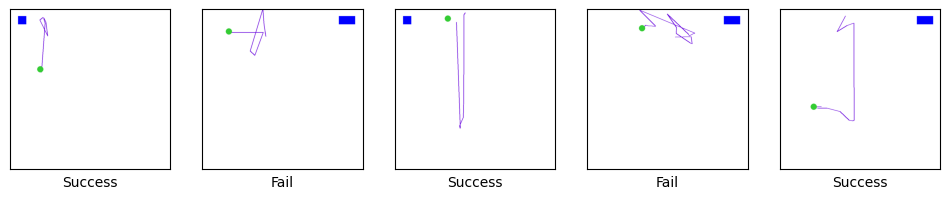
This code will draw a red square at the top-left position for the ad positions labeled "top\_left" and at the top-right position for the ad positions labeled "top\_right" in the dataset.

To represent the ad positions, we draw squares on an image, where each square corresponds to an ad position. Additionally, we differentiate between different types of ads by assigning different colors to the squares.

we change the color of the square based on the ad\_type value.

**update**  
I add ad\_type to image but result was not good so i remove it from this step.

for index,row in dataset.iterrows():  
  
 df = log\_csv\_df[row["log\_id"]]  
 img = user\_images[row["log\_id"]]  
 # Draw a dot  
 draw = ImageDraw.Draw(img)  
 color = "blue" #(51, 153, 255) if row["ad\_type"] else (255, 0, 255)  
 for index2,row2 in df.iterrows():  
 space = int(image\_width \* 0.05)  
 square\_size = int(image\_width \* 0.1)  
 if row["ad\_position"]:  
 # Draw the square  
 draw.rectangle([space, space, square\_size, square\_size], fill=color)  
 else:  
 # Draw the square  
 draw.rectangle([image\_width - square\_size - space, space, image\_width - space, square\_size], fill=color)  
  
  
display\_images(user\_images,image\_ad\_clicked)



## 

## 6.6 Convert Image To Numpy Array

Converting an image to a NumPy array is essential when working with image data in machine learning and deep learning projects.

Here are a few reasons why this conversion is necessary:

1. Compatibility: Most deep learning libraries, such as TensorFlow and PyTorch, expect input data in the form of NumPy arrays.
2. Data Manipulation: NumPy arrays offer efficient and convenient methods for data manipulation, such as normalization, resizing, and cropping.
3. Data Preprocessing: NumPy arrays allow you to preprocess image data easily, including scaling pixel values, data augmentation, and converting image color channels.
4. Integration with Other Libraries: NumPy arrays can be seamlessly integrated with other Python libraries, making it easier to work with image data alongside other data types.

import numpy as np  
  
new\_image\_width = 100  
new\_image\_height = 100  
  
X2 = []  
y2 = []  
  
for index,row in dataset.iterrows():  
 image = user\_images[row["log\_id"]]  
 resized\_image = image.resize((new\_image\_width, new\_image\_height))  
  
 # Convert the image to a numpy array  
 image\_array = np.array(resized\_image)  
  
 X2.append(image\_array)  
 y2.append(row["ad\_clicked"])  
  
X2 = np.array([i for i in X2])  
y2 = np.array([i for i in y2])

## 6.7 Clear Session

In a Convolutional Neural Network (CNN) model, it's a good practice to clear the session before creating a new model.

The Keras API runs on top of TensorFlow, and TensorFlow uses a static computation graph. When you create a new model in Keras without clearing the session, it might lead to memory leaks and slow down your training process.

from tensorflow.keras import backend   
import tensorflow as tf  
  
tf.random.set\_seed(35)  
backend.clear\_session()

## 

## 6.8 Split Data

from sklearn.model\_selection import train\_test\_split  
  
# Split the data into train and test sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X2, y2, test\_size=0.2, random\_state=42)  
  
print(f'Shape of features data set for ML models => {X2.shape}')  
print(f'Shape of target for ML models => {y2.shape}')

Shape of features data set for ML models => (2418, 100, 100, 3)  
Shape of target for ML models => (2418,)

## 6.9 CNN Model For Images

To utilize user behavior data effectively, we convert it into images, allowing us to apply convolutional neural network (CNN) models for analysis. In this code, we design a CNN model using Keras. We start by defining the input shape based on the dimensions of the images we generated from user behavior data.

Our model consists of convolutional layers, pooling layers, and fully connected layers. The convolutional layers use 3x3 filters with ReLU activation functions, followed by max-pooling layers to reduce dimensionality. We stack two pairs of convolutional and pooling layers to extract features from the images effectively.

After the convolutional layers, we flatten the 3D arrays into a 1D array and add fully connected layers. Finally, we compile the model using the Adam optimizer and binary cross-entropy loss function. We also specify accuracy as the evaluation metric.

from keras.models import Sequential  
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense  
  
# Define input shape  
input\_shape = (new\_image\_width, new\_image\_height, 3)  
  
# Initialize the model  
model = Sequential()  
  
model.add(Input(shape=input\_shape))  
  
# Add convolutional layer  
model.add(Conv2D(32, (3, 3), activation='relu'))  
  
# Add pooling layer  
model.add(MaxPooling2D((2, 2)))  
  
# Add another convolutional layer  
model.add(Conv2D(64, (3, 3), activation='relu'))  
  
# Add another pooling layer  
model.add(MaxPooling2D((2, 2)))  
  
# Flatten the 3D arrays to 1D array  
model.add(Flatten())  
  
# Add fully connected layers  
model.add(Dense(64, activation='relu'))  
model.add(Dense(1, activation='sigmoid'))  
  
# Compile the model  
model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
  
# Print the model summary  
model.summary()

Model: "sequential"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓  
┃ Layer (type) ┃ Output Shape ┃ Param # ┃  
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩  
│ conv2d (Conv2D) │ (None, 98, 98, 32) │ 896 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling2d (MaxPooling2D) │ (None, 49, 49, 32) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ conv2d\_1 (Conv2D) │ (None, 47, 47, 64) │ 18,496 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ max\_pooling2d\_1 (MaxPooling2D) │ (None, 23, 23, 64) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ flatten (Flatten) │ (None, 33856) │ 0 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense (Dense) │ (None, 64) │ 2,166,848 │  
├─────────────────────────────────┼────────────────────────┼───────────────┤  
│ dense\_1 (Dense) │ (None, 1) │ 65 │  
└─────────────────────────────────┴────────────────────────┴───────────────┘

Total params: 2,186,305 (8.34 MB)

Trainable params: 2,186,305 (8.34 MB)

Non-trainable params: 0 (0.00 B)

# Define callbacks  
model\_checkpoint = ModelCheckpoint('best\_model.keras', save\_best\_only=True, monitor='val\_accuracy', mode='max', verbose=1)  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)  
  
# Train the model  
history = model.fit(X\_train, y\_train, epochs=8, batch\_size=32, validation\_split=0.2, callbacks=[model\_checkpoint, early\_stopping])

Epoch 1/8  
 1/49 ━━━━━━━━━━━━━━━━━━━━ 26s 546ms/step - accuracy: 0.9375 - loss: 0.1898

49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 600ms/step - accuracy: 0.9120 - loss: 0.2472  
Epoch 1: val\_accuracy improved from -inf to 0.89664, saving model to best\_model.keras  
49/49 ━━━━━━━━━━━━━━━━━━━━ 32s 645ms/step - accuracy: 0.9120 - loss: 0.2469 - val\_accuracy: 0.8966 - val\_loss: 0.3547  
Epoch 2/8  
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 488ms/step - accuracy: 0.9215 - loss: 0.2129  
Epoch 2: val\_accuracy did not improve from 0.89664  
49/49 ━━━━━━━━━━━━━━━━━━━━ 26s 512ms/step - accuracy: 0.9214 - loss: 0.2128 - val\_accuracy: 0.8941 - val\_loss: 0.3736  
Epoch 3/8  
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 342ms/step - accuracy: 0.9236 - loss: 0.1948  
Epoch 3: val\_accuracy did not improve from 0.89664  
49/49 ━━━━━━━━━━━━━━━━━━━━ 18s 361ms/step - accuracy: 0.9235 - loss: 0.1947 - val\_accuracy: 0.8889 - val\_loss: 0.4094  
Epoch 4/8  
49/49 ━━━━━━━━━━━━━━━━━━━━ 0s 412ms/step - accuracy: 0.9282 - loss: 0.1817  
Epoch 4: val\_accuracy did not improve from 0.89664  
49/49 ━━━━━━━━━━━━━━━━━━━━ 21s 435ms/step - accuracy: 0.9281 - loss: 0.1816 - val\_accuracy: 0.8941 - val\_loss: 0.4569

# Evaluate the model  
y\_pred = model.predict(X\_test)  
y\_pred\_classes = (y\_pred > 0.5).astype("int32")  
  
accuracy = accuracy\_score(y\_test, y\_pred\_classes)  
  
print("Test Accuracy:", accuracy)  
print(classification\_report(y\_test, y\_pred\_classes))

2/16 ━━━━━━━━━━━━━━━━━━━━ 1s 85ms/step

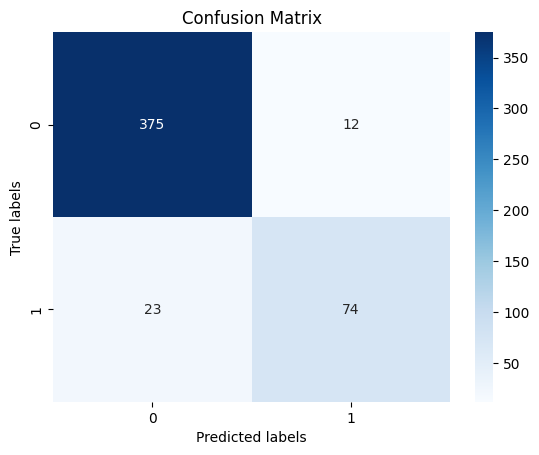
16/16 ━━━━━━━━━━━━━━━━━━━━ 2s 123ms/step  
Test Accuracy: 0.9276859504132231  
 precision recall f1-score support  
  
 0 0.94 0.97 0.96 387  
 1 0.86 0.76 0.81 97  
  
 accuracy 0.93 484  
 macro avg 0.90 0.87 0.88 484  
weighted avg 0.93 0.93 0.93 484

**The Success of the CNN Model**

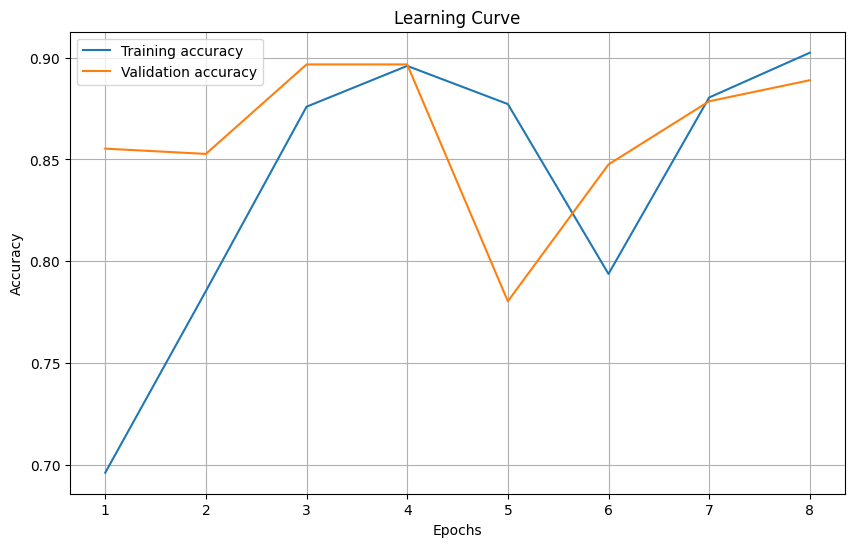
* High Accuracy
  + The CNN model achieved an accuracy of 92%, outperforming all other models developed in this project.
* Impressive Precision
  + The precision of the CNN model was outstanding, with 94% precision for class 0 and 86% precision for class 1.

The success of the CNN model can be attributed to its ability to effectively capture and learn intricate patterns in user behavior data. By leveraging the power of deep learning, the CNN model provided highly accurate predictions, enabling us to better understand and anticipate user interactions with ads.

confusion\_plot(y\_test, y\_pred\_classes)



import matplotlib.pyplot as plt  
  
train\_scores = history.history['accuracy']  
validation\_scores = history.history['val\_accuracy']  
epochs = range(1, len(train\_scores) + 1)  
  
plt.figure(figsize=(10, 6))  
plt.plot(epochs, train\_scores, label='Training accuracy')  
plt.plot(epochs, validation\_scores, label='Validation accuracy')  
plt.xlabel('Epochs')  
plt.ylabel('Accuracy')  
plt.title('Learning Curve')  
plt.legend(loc='best')  
plt.grid()  
plt.show()



# 7. Conclusion

## Utilizing Two Different Approaches for Predicting User Clicks

In this project, we explored two distinct approaches to predict user clicks on ads.

### Parametric Approach with Machine Learning Models:

* The first approach involved utilizing parametric machine learning models. We experimented with various algorithms such as Naive Bayes, Random Forest Classifier, Logistic Regression, and Support Vector Regression (SVR). These models were trained using features extracted from user behavior data, including ad position, ad type, and other relevant parameters. While these models provided reasonable accuracy, we found that they lacked the ability to capture intricate patterns within the data, limiting their predictive capabilities.

### Visualizing User Behavior to Predict Clicks:

* Recognizing the limitations of parametric models, we adopted a second approach that involved visualizing user behavior data. By plotting user interactions on the webpage, including mouse movements and clicks, we were able to create a visual representation of user engagement with ads. This approach provided us with valuable insights into user behavior, allowing us to identify patterns and trends that were not apparent from the raw data alone.

### The Success of Visual Approach:

* One of the most successful implementations of this visual approach was the use of Convolutional Neural Networks (CNNs). By converting user behavior data into image-like representations, we trained a CNN model to predict user clicks on ads. This model achieved an impressive accuracy of 92%, outperforming all other machine learning models developed in this project. Additionally, it demonstrated remarkable precision, with 94% precision for class 0 and 86% precision for class 1.

### Conclusion:

* By leveraging both parametric machine learning models and visual representations of user behavior, we gained a comprehensive understanding of user interactions with ads. While parametric models provided a baseline level of accuracy, the visual approach enabled us to capture more nuanced patterns within the data, leading to significantly improved predictive performance. Moving forward, we can utilize the insights gained from both approaches to optimize ad placement and design, ultimately improving user engagement and ad performance.

# 8. References

* Keras API reference / Optimizers(<https://keras.io/api/optimizers/>)
* Building an Ad Click Prediction Machine Learning System: Problem Statement and Metrics (<https://www.linkedin.com/pulse/building-ad-click-prediction-machine-learning-system-problem-shivam-s-1wvjf>)
* Predicting Click Behavior Based on Machine Learning Models (<https://www.researchgate.net/publication/370570414_Predicting_Click_Behavior_Based_on_Machine_Learning_Models>)
* What is Mutual Information? (<https://quantdare.com/what-is-mutual-information/>)
* Customer ad click prediction (<https://www.kaggle.com/code/mafrojaakter/customer-ad-click-prediction>)
* sklearn.svm.SVR (<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>)
* sklearn.ensemble.RandomForestRegressor(<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>)
* Metrics and scoring: quantifying the quality of predictions(<https://scikit-learn.org/stable/modules/model_evaluation.html>)
* Keras API reference / Callbacks API / ModelCheckpoint (<https://keras.io/api/callbacks/model_checkpoint/>)
* Keras API reference / Callbacks API / EarlyStopping(<https://keras.io/api/callbacks/early_stopping/>)
* Keras API reference / Optimizers(<https://keras.io/api/optimizers/>)