# Logistic Regression

**Instructions:**

Please share your answers filled in-line in the word document. Submit code separately wherever applicable.

Please ensure you update all the details:

**Name: Vishvash C Batch ID:** 23012024

**Topic: Logistic Regression**

**Guidelines:**

**1. An assignment submission is considered complete only when the correct and executable code(s) and documentation explaining the method and results are submitted. Failing to submit either of those will be considered an invalid submission and not a correct submission.**

**2. Ensure that you submit your assignments correctly and in full. Resubmission is not allowed.**

**3. Post the submission you can evaluate your work by referring to the keys provided. (will be available only post the submission).**

**Hints:**

1. **Business Problem**
   1. **What is the business objective?**
   2. **Are there any constraints?**
2. **Work on each feature of the dataset to create a data dictionary as displayed in the below image:**

**Make a table as shown above and provide information about the features such as its data type and its relevance to the model building. And if not relevant, provide reasons and a description of the feature.**

**Using Python codes perform:**

1. **Data Pre-processing**

**3.1 Data Cleaning, Feature Engineering, etc.**

**3.2 Outlier Treatment.**

1. **Exploratory Data Analysis (EDA):**
   1. **Summary.**
   2. **Univariate analysis.**
   3. **Bivariate analysis.**
2. **Model Building**
   1. **Build the model on the scaled data (try multiple options).**
   2. **Build a Logistic Regression model.**
   3. **Train and test the model and compare accuracies by building a confusion matrix, and plotting ROC and AUC curves.**
   4. **Briefly explain the model output in the documentation.**
3. **Write about the benefits/impact of the solution - in what way does the business (client) benefit from the solution provided?**

Problem Statement: -

1. In this time and age of widespread internet usage, effective and targeted marketing plays a vital role. A marketing company would like to develop a strategy by analyzing its customer data. For this, data like age, location, time of activity, etc. have been collected to determine whether a user will click on an ad or not. Perform Logistic Regression on the given data to predict whether a user will click on an ad or not.

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| Name of Feature | Description | Type | Relevance |
| Daily\_Time\_Spent\_on\_Site | Daily time spent on the website (in min) | Quantitative | Relevant |
| Age | Age of the user | Quantitative | Relevant |
| Area\_Income | Income of the area where the user resides | Quantitative | Relevant |
| Daily\_Internet\_Usage | Daily internet usage (in min) | Quantitative | Relevant |
| Ad\_Topic\_Line | Topic line of the ad | Nominal | Irrelevant |
| City | City where the user resides | Nominal | Relevant |
| Male | Gender of the user (male: 1, female: 0) | Nominal | Relevant |
| Country | Country where the user resides | Nominal | Relevant |
| Timestamp | Timestamp of the interaction | Nominal | Relevant |
| Clicked\_on\_Ad | Whether the user clicked on the ad | Nominal | Relevant |

**Code:**

## Problem Statement

'''

In this time and age of widespread internet usage, effective and targeted marketing plays a vital role. A marketing company would like to develop a strategy by analyzing its customer data. For this, data like age, location, time of activity, etc. have been collected to determine whether a user will click on an ad or not. Perform Logistic Regression on the given data to predict whether a user will click on an ad or not.

\*\*Objective(s):\*\* Maximize the clickrate

\*\*Constraints:\*\* Maximize the ad relevancy

\*\*Success Criteria\*\*

- \*\*Business Success Criteria\*\*: Improve the clickrates anywhere between 10% to 20%.

- \*\*ML Success Criteria\*\*: Accuracy should be around 70% - 75%

- \*\*Economic Success Criteria\*\*: Increase the ad revenues by atleast 20%.

'''

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from feature\_engine.outliers import Winsorizer

from sklearn.compose import ColumnTransformer

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler #, MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from sklearn.pipeline import Pipeline

import pickle, joblib

# import statsmodels.formula.api as smf

import statsmodels.api as sm

from sklearn.model\_selection import train\_test\_split # train and test

# import pylab as pl

from sklearn import metrics

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.metrics import roc\_curve

from sklearn.metrics import classification\_report

# SQL Integration

from sqlalchemy import create\_engine, text

engine = create\_engine("mysql+pymysql://{user}:{pw}@localhost/{db}"

.format(user = "root", # user

pw = "1234", # passwrd

db = "ctr\_db")) # database

# Load the offline data into Database to simulate client conditions

clicks = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Logistic Regresssion/Assignments/Logistic Regression Assignment/advertising.csv").convert\_dtypes()

clicks.info()

clicks.to\_sql('clicks', con = engine, if\_exists = 'replace', chunksize = 1000, index= False)

#### Read the Table (data) from MySQL database

sql = 'SELECT \* FROM clicks'

# Convert columns to best possible dtypes using

adclicks = pd.read\_sql\_query(text(sql), engine.connect()).convert\_dtypes()

adclicks.head()

adclicks.Ad\_Topic\_Line.duplicated().sum()

adclicks.City.duplicated().sum()

adclicks.Country.duplicated().sum()

c1 = adclicks.drop('Ad\_Topic\_Line', axis = 1)

c1 = c1.drop('City', axis = 1)

c1 = c1.drop('Country', axis = 1)

# Convert 'Timestamp' column to datetime format

c1['Timestamp'] = pd.to\_datetime(c1['Timestamp'])

# Convert the datetime values to Unix time (seconds since epoch)

c1['Unix\_Time'] = (c1['Timestamp'] - pd.Timestamp("1970-01-01")) // pd.Timedelta(seconds=1)

# Now, 'df' contains a single numerical column 'Unix\_Time' representing the timestamp

c1 = c1.drop('Timestamp', axis = 1)

c1.info()

c1.describe()

c1.isna().sum()

c1.head()

# Predictors

X = c1.iloc[:, [\*range(0, 5), 6]]

X

# Target

# y = c1[['Clicked\_on\_Ad']]

y = c1.iloc[:, [5]]

y

# # Convert columns to 'object' data type

# X['City'] = X['City'].astype('object')

# X['Country'] = X['Country'].astype('object')

X.info()

# Segregating data based on their data types

numeric\_features = X.select\_dtypes(exclude = ['object', 'string']).columns

# numeric\_features = X.select\_dtypes(exclude = ['object']).columns

numeric\_features

# Seperating Integer and Float data

numeric\_features1 = X.select\_dtypes(include = ['int64']).columns

numeric\_features1

numeric\_features2 = X.select\_dtypes(include = ['float64']).columns

numeric\_features2

# categ\_features = X.select\_dtypes(include = ['object', 'string']).columns

# categ\_features

# Imputation techniques to handle missing data

# Mode imputation for Integer (categorical) data

num\_pipeline1 = Pipeline(steps=[('impute1', SimpleImputer(strategy = 'most\_frequent'))])

# Mean imputation for Continuous (Float) data

num\_pipeline2 = Pipeline(steps=[('impute2', SimpleImputer(strategy = 'mean'))])

# One-hot encoding for categorical features

# cat\_pipeline = Pipeline(steps=[('onehot', OneHotEncoder(drop = 'first'))])

# 1st Imputation Transformer

preprocessor = ColumnTransformer([

('mode', num\_pipeline1, numeric\_features1),

('mean', num\_pipeline2, numeric\_features2)])

# ('onehot', cat\_pipeline, categ\_features)])

print(preprocessor)

# Fit the data to train imputation pipeline model

impute\_data = preprocessor.fit(X)

# Save the pipeline

joblib.dump(impute\_data, 'impute')

# Transform the original data

X1 = pd.DataFrame(impute\_data.transform(X), columns = X.columns).convert\_dtypes()

# traff = pd.DataFrame(processed3.transform(df1).toarray(), columns = list(processed3.get\_feature\_names\_out()))

X1.isna().sum()

X1.info()

# Multiple boxplots in a single visualization.

# Columns with larger scales affect other columns.

# Below code ensures each column gets its own y-axis.

# pandas plot() function with parameters kind = 'box' and subplots = True

X1.iloc[:,0:6].plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))

'''sharey True or 'all': x- or y-axis will be shared among all subplots.

False or 'none': each subplot x- or y-axis will be independent.'''

# increase spacing between subplots

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

# CLMAGE and Loss features are continuous data with outliers

# Ignore other categorical features

winsor = Winsorizer(capping\_method = 'iqr', # choose IQR rule boundaries or gaussian for mean and std

tail = 'both', # cap left, right or both tails

fold = 1.5,

variables = ['Area\_Income'])

outlier\_pipeline = Pipeline(steps = [('winsor', winsor)])

outlier\_pipeline

preprocessor1 = ColumnTransformer(transformers = [('wins',

outlier\_pipeline,

['Area\_Income'])],

remainder = 'passthrough')

print(preprocessor1)

# print(X1.iloc[:,0:10].columns)

# Fit the data

winz\_data = preprocessor1.fit(X1)

# Save the pipeline

joblib.dump(winz\_data, 'winzor')

X2 = pd.DataFrame(winz\_data.transform(X1), columns = X1.columns).convert\_dtypes()

X2.info()

# Boxplot

X2.iloc[:,0:6].plot(kind = 'box', subplots = True, sharey = False, figsize = (15, 8))

plt.subplots\_adjust(wspace = 0.75) # ws is the width of the padding between subplots, as a fraction of the average Axes width.

plt.show()

# Address the scaling issue

scale\_pipeline = Pipeline(steps=[('scale', StandardScaler())])

# scale\_pipeline = Pipeline(steps=[('scale', MinMaxScaler())])

# print(X1.iloc[:,0:10].columns)

scalable\_features = X2.iloc[:,0:6].columns

preprocessor2 = ColumnTransformer(transformers = [('num',

scale\_pipeline, scalable\_features)],

remainder = 'passthrough')

print(preprocessor2)

scale = preprocessor2.fit(X2)

joblib.dump(scale, 'scale')

X3 = pd.DataFrame(scale.transform(X2), columns = X2.columns)

X3.columns

X3.info()

# Convert X3 to float dtype

X3 = X3.astype(float)

X3.info()

#######################

# Target variable

y.info()

# What is the difference between "Int64" and "int64"?

# One is a nullable integer dtype. The other is a numpy dtype.

y = y.astype('int')

y.info()

### Statsmodel

# Building the model and fitting the data

logit\_model = sm.Logit(y, X3).fit()

# Save the model

pickle.dump(logit\_model, open('logistic.pkl', 'wb'))

# Summary

logit\_model.summary()

logit\_model.summary2() # for AIC

# Prediction

pred = logit\_model.predict(X3)

pred # Probabilities

# ROC Curve to identify the appropriate cutoff value

fpr, tpr, thresholds = roc\_curve(y.Clicked\_on\_Ad, pred)

optimal\_idx = np.argmax(tpr - fpr)

optimal\_threshold = thresholds[optimal\_idx]

optimal\_threshold

auc = metrics.auc(fpr, tpr)

print("Area under the ROC curve : %f" % auc)

# Filling all the cells with zeroes

X3["pred"] = np.zeros(1000)

# taking threshold value and above the prob value will be treated as correct value

X3.loc[pred > optimal\_threshold, "pred"] = 1

# Confusion Matrix

confusion\_matrix(X3.pred, y.Clicked\_on\_Ad)

# Accuracy score of the model

print('Test accuracy = ', accuracy\_score(X3.pred, y.Clicked\_on\_Ad))

# Classification report

classification = classification\_report(X3["pred"], y)

print(classification)

### PLOT FOR ROC

plt.plot(fpr, tpr, label = "AUC="+str(auc))

plt.ylabel('True Positive Rate')

plt.xlabel('False Positive Rate')

plt.legend(loc = 4)

plt.show()

################################################################

# Model evaluation - Data Split

x\_train, x\_test, y\_train, y\_test = train\_test\_split (X3.iloc[:, :6], y,

test\_size = 0.2,

random\_state = 0,

stratify = y)

# Fitting Logistic Regression to the training set

logisticmodel = sm.Logit(y\_train, x\_train).fit()

# Evaluate on train data

y\_pred\_train = logisticmodel.predict(x\_train)

y\_pred\_train

# Metrics

# Filling all the cells with zeroes

y\_train["pred"] = np.zeros(800)

# taking threshold value and above the prob value will be treated as correct value

y\_train.loc[pred > optimal\_threshold, "pred"] = 1

auc = metrics.roc\_auc\_score(y\_train["Clicked\_on\_Ad"], y\_pred\_train)

print("Area under the ROC curve for train data : %f" % auc)

classification\_train = classification\_report(y\_train["pred"], y\_train["Clicked\_on\_Ad"])

print(classification\_train)

# confusion matrix

confusion\_matrix(y\_train["pred"], y\_train["Clicked\_on\_Ad"])

# Accuracy score of the model

print('Train accuracy = ', accuracy\_score(y\_train["pred"], y\_train["Clicked\_on\_Ad"]))

# Validate on Test data

y\_pred\_test = logisticmodel.predict(x\_test)

y\_pred\_test

# Filling all the cells with zeroes

y\_test["y\_pred\_test"] = np.zeros(200)

# Capturing the prediction binary values

y\_test.loc[y\_pred\_test > optimal\_threshold, "y\_pred\_test"] = 1

# classification report

classification1 = classification\_report(y\_test["y\_pred\_test"], y\_test["Clicked\_on\_Ad"])

print(classification1)

# confusion matrix

confusion\_matrix(y\_test["y\_pred\_test"], y\_test["Clicked\_on\_Ad"])

# Accuracy score of the model

print('Test accuracy = ', accuracy\_score(y\_test["y\_pred\_test"], y\_test["Clicked\_on\_Ad"]))

#############################################

# Test the best model on new data

model1 = pickle.load(open('logistic.pkl', 'rb'))

impute = joblib.load('impute')

winzor = joblib.load('winzor')

minmax = joblib.load('scale')

# Load the new data

data = pd.read\_csv(r"C:/Users/Lenovo/Downloads/Study material/Data Science/Logistic Regresssion/Assignments/Logistic Regression Assignment/advertising\_test.csv").convert\_dtypes()

data = data.drop(['Ad\_Topic\_Line', 'City', 'Country'], axis = 1)

# Convert 'Timestamp' column to datetime format

data['Timestamp'] = pd.to\_datetime(data['Timestamp'])

# Convert the datetime values to Unix time (seconds since epoch)

data['Unix\_Time'] = (data['Timestamp'] - pd.Timestamp("1970-01-01")) // pd.Timedelta(seconds=1)

# Now, 'df' contains a single numerical column 'Unix\_Time' representing the timestamp

data = data.drop('Timestamp', axis = 1)

data.head()

# Engine = create\_engine(f"mysql+pymysql://{user}:{pw}@localhost/{db}") #database

clean = pd.DataFrame(impute.transform(data), columns = data.columns).convert\_dtypes()

clean1 = pd.DataFrame(winzor.transform(clean), columns = data.columns).convert\_dtypes()

clean3 = pd.DataFrame(minmax.transform(clean1), columns = clean1.columns)

prediction = model1.predict(clean3)

prediction

# optimal\_threshold=0.60

data["Clicked\_on\_Ad"] = np.zeros(len(prediction))

# taking threshold value and above the prob value will be treated as correct value

data.loc[prediction > optimal\_threshold, "Clicked\_on\_Ad"] = 1

data[['Clicked\_on\_Ad']] = data[['Clicked\_on\_Ad']].astype('int64')

data[['Clicked\_on\_Ad']]

**Output:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Daily\_Time\_ Spent \_on\_Site 1000 non-null Float64

1 Age 1000 non-null Int64

2 Area\_Income 1000 non-null Float64

3 Daily\_Internet\_Usage 1000 non-null Float64

4 Ad\_Topic\_Line 1000 non-null string

5 City 1000 non-null string

6 Male 1000 non-null Int64

7 Country 1000 non-null string

8 Timestamp 1000 non-null string

9 Clicked\_on\_Ad 1000 non-null Int64

dtypes: Float64(3), Int64(3), string(4)

memory usage: 84.1 KB

adclicks.head()

Out[146]:

Daily\_Time\_ Spent \_on\_Site Age ... Timestamp Clicked\_on\_Ad

0 68.95 35 ... 27-03-2016 00:53 0

1 80.23 31 ... 04-04-2016 01:39 0

2 69.47 26 ... 13-03-2016 20:35 0

3 74.15 29 ... 10-01-2016 02:31 0

4 68.37 35 ... 03-06-2016 03:36 0

[5 rows x 10 columns]

c1.head()

Out[187]:

Daily\_Time\_ Spent \_on\_Site Age ... Clicked\_on\_Ad Unix\_Time

0 68.95 35 ... 0 1459039980

1 80.23 31 ... 0 1459733940

2 69.47 26 ... 0 1457901300

3 74.15 29 ... 0 1452393060

4 68.37 35 ... 0 1464924960

[5 rows x 9 columns]

X1.info()

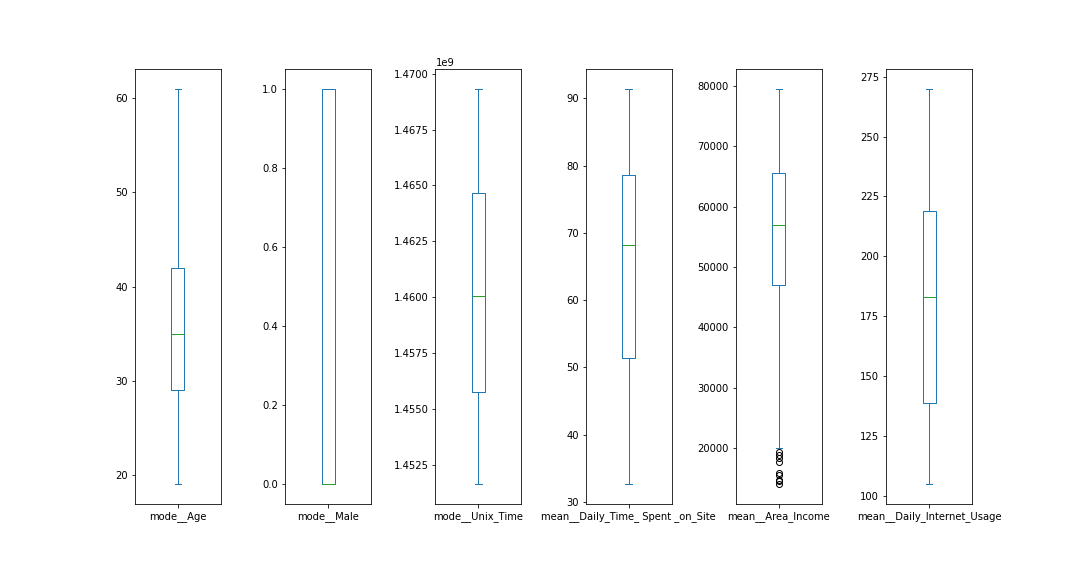
<class 'pandas.core.frame.DataFrame'>

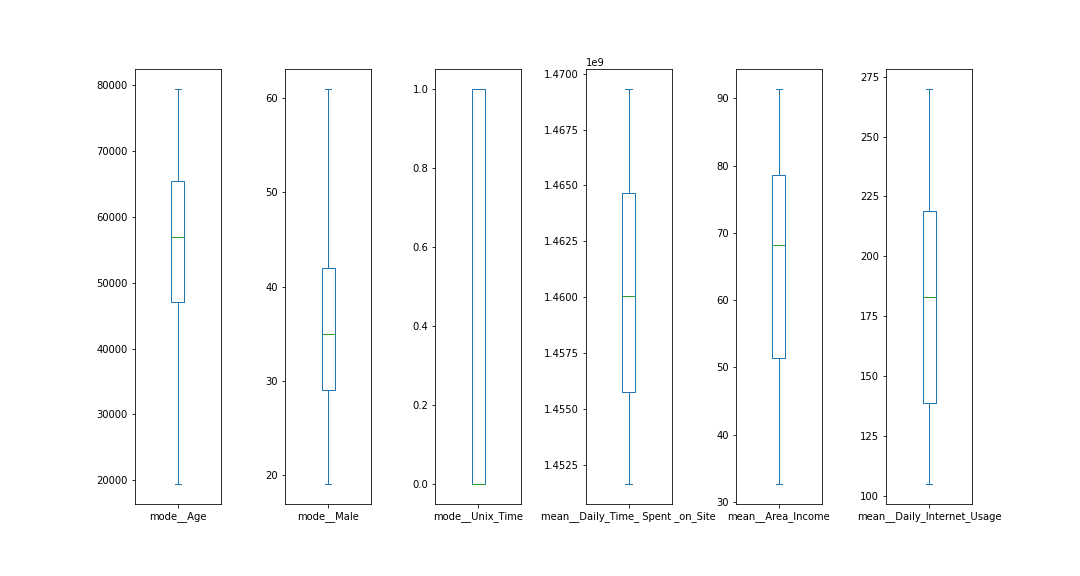
RangeIndex: 1000 entries, 0 to 999

Columns: 1210 entries, mode\_\_Age to onehot\_\_Country\_Zimbabwe

dtypes: Float64(3), Int64(1207)

memory usage: 10.4 MB





logit\_model = sm.Logit(y, X3).fit(maxiter=5000)

Warning: Maximum number of iterations has been exceeded.

Current function value: inf

Iterations: 5000

Traceback (most recent call last):

Cell In[509], line 1

logit\_model = sm.Logit(y, X3).fit(maxiter=5000)

File ~\anaconda3\envs\python\_10\lib\site-packages\statsmodels\discrete\discrete\_model.py:2601 in fit

bnryfit = super().fit(start\_params=start\_params,

File ~\anaconda3\envs\python\_10\lib\site-packages\statsmodels\discrete\discrete\_model.py:243 in fit

mlefit = super().fit(start\_params=start\_params,

File ~\anaconda3\envs\python\_10\lib\site-packages\statsmodels\base\model.py:582 in fit

Hinv = np.linalg.inv(-retvals['Hessian']) / nobs

File ~\anaconda3\envs\python\_10\lib\site-packages\numpy\linalg\linalg.py:561 in inv

ainv = \_umath\_linalg.inv(a, signature=signature, extobj=extobj)

File ~\anaconda3\envs\python\_10\lib\site-packages\numpy\linalg\linalg.py:112 in \_raise\_linalgerror\_singular

raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix

<class 'statsmodels.iolib.summary.Summary'>

"""

Logit Regression Results

==============================================================================

Dep. Variable: Clicked\_on\_Ad No. Observations: 1000

Model: Logit Df Residuals: 994

Method: MLE Df Model: 5

Date: Sat, 06 Apr 2024 Pseudo R-squ.: 0.8312

Time: 17:47:17 Log-Likelihood: -117.00

converged: True LL-Null: -693.15

Covariance Type: nonrobust LLR p-value: 6.331e-247

==============================================================================================

coef std err z P>|z| [0.025 0.975]

----------------------------------------------------------------------------------------------

Daily\_Time\_ Spent \_on\_Site -0.0411 0.180 -0.229 0.819 -0.393 0.311

Age 0.8969 0.163 5.499 0.000 0.577 1.217

Area\_Income -0.2072 0.181 -1.143 0.253 -0.563 0.148

Daily\_Internet\_Usage -2.1547 0.220 -9.777 0.000 -2.587 -1.723

Male -1.0953 0.171 -6.392 0.000 -1.431 -0.759

Unix\_Time -2.3387 0.230 -10.187 0.000 -2.789 -1.889

==============================================================================================

"""

logit\_model.summary2() # for AIC

Out[645]:

<class 'statsmodels.iolib.summary2.Summary'>

"""

Results: Logit

===========================================================================

Model: Logit Method: MLE

Dependent Variable: Clicked\_on\_Ad Pseudo R-squared: 0.831

Date: 2024-04-06 17:47 AIC: 246.0033

No. Observations: 1000 BIC: 275.4498

Df Model: 5 Log-Likelihood: -117.00

Df Residuals: 994 LL-Null: -693.15

Converged: 1.0000 LLR p-value: 6.3309e-247

No. Iterations: 9.0000 Scale: 1.0000

---------------------------------------------------------------------------

Coef. Std.Err. z P>|z| [0.025 0.975]

---------------------------------------------------------------------------

Daily\_Time\_ Spent \_on\_Site -0.0411 0.1796 -0.2289 0.8190 -0.3932 0.3110

Age 0.8969 0.1631 5.4992 0.0000 0.5773 1.2166

Area\_Income -0.2072 0.1813 -1.1429 0.2531 -0.5625 0.1481

Daily\_Internet\_Usage -2.1547 0.2204 -9.7775 0.0000 -2.5866 -1.7228

Male -1.0953 0.1714 -6.3917 0.0000 -1.4312 -0.7594

Unix\_Time -2.3387 0.2296 -10.1871 0.0000 -2.7886 -1.8887

===========================================================================

"""

pred = logit\_model.predict(X3)

pred # Probabilities

Out[647]:

0 0.006405

1 0.009705

2 0.008053

3 0.003657

4 0.012472

995 0.008718

996 0.980980

997 0.999382

998 0.599656

999 0.981974

Length: 1000, dtype: float64

optimal\_threshold

Out[651]: 0.2670696006555806

auc = metrics.auc(fpr, tpr)

print("Area under the ROC curve : %f" % auc)

Area under the ROC curve : 0.990644

confusion\_matrix(X3.pred, y.Clicked\_on\_Ad)

Out[658]:

array([[492, 21],

[ 8, 479]], dtype=int64)

print('Test accuracy = ', accuracy\_score(X3.pred, y.Clicked\_on\_Ad))

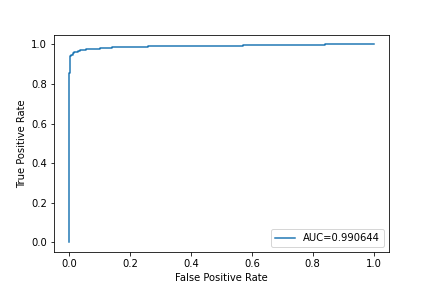
Test accuracy = 0.971

print(classification)

precision recall f1-score support

0.0 0.98 0.96 0.97 513

1.0 0.96 0.98 0.97 487



print("Area under the ROC curve for train data : %f" % auc)

Area under the ROC curve for train data : 0.992869

print(classification\_train)

precision recall f1-score support

0.0 0.99 0.96 0.98 410

1.0 0.96 0.99 0.97 390

accuracy 0.97 800

macro avg 0.98 0.98 0.97 800

weighted avg 0.98 0.97 0.98 800

array([[395, 15],

[ 5, 385]], dtype=int64)

Train accuracy = 0.975

print(classification1)

precision recall f1-score support

0.0 0.97 0.92 0.94 106

1.0 0.91 0.97 0.94 94

accuracy 0.94 200

macro avg 0.94 0.94 0.94 200

weighted avg 0.94 0.94 0.94 200

confusion\_matrix(y\_test["y\_pred\_test"], y\_test["Clicked\_on\_Ad"])

Out[695]:

array([[97, 9],

[ 3, 91]], dtype=int64)

Test accuracy = 0.94

data[['Clicked\_on\_Ad']]

Out[751]:

Clicked\_on\_Ad

0 0

1 0

2 0

3 0

4 0

.. ...

995 0

996 1

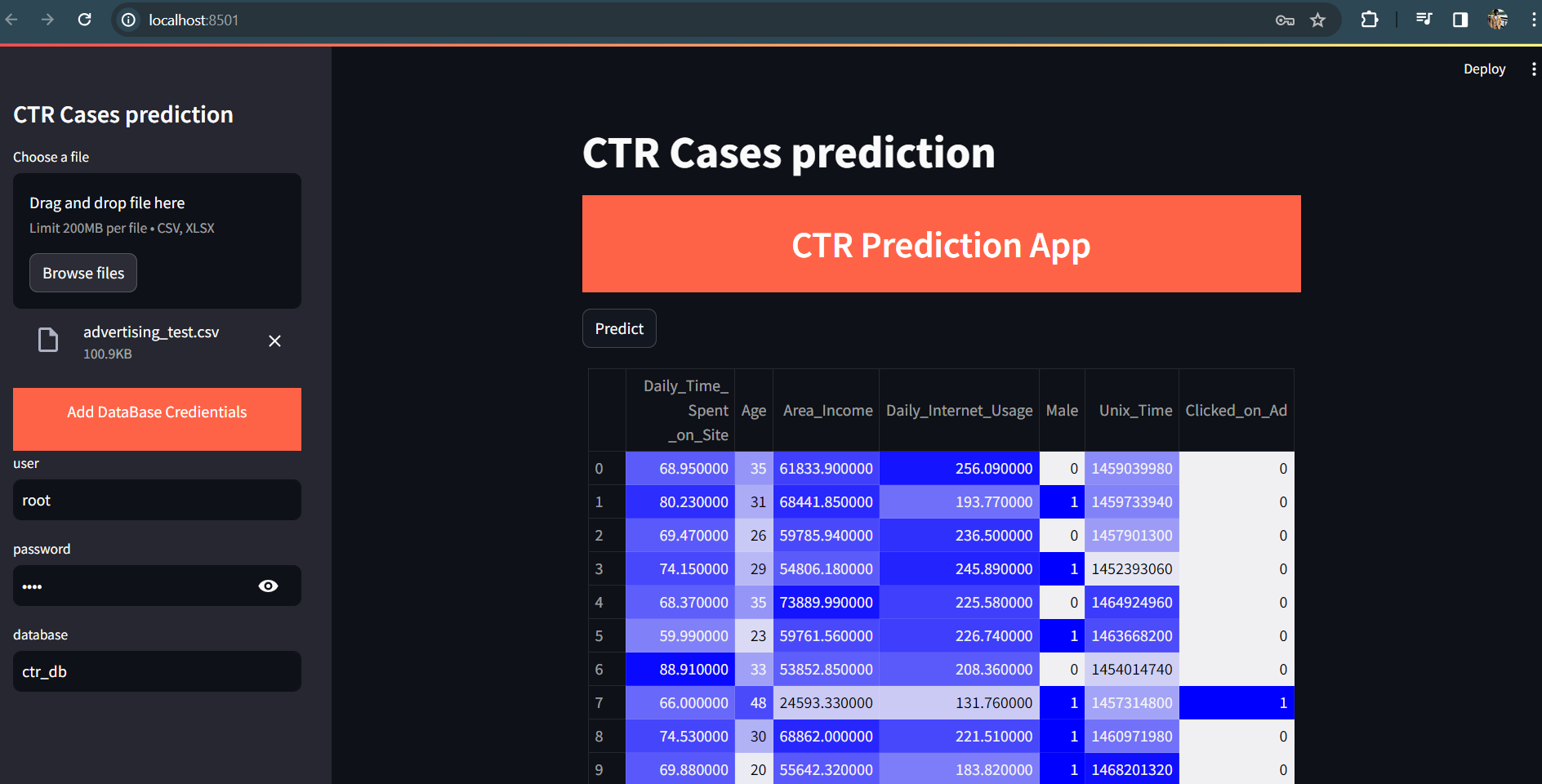
997 1

998 1

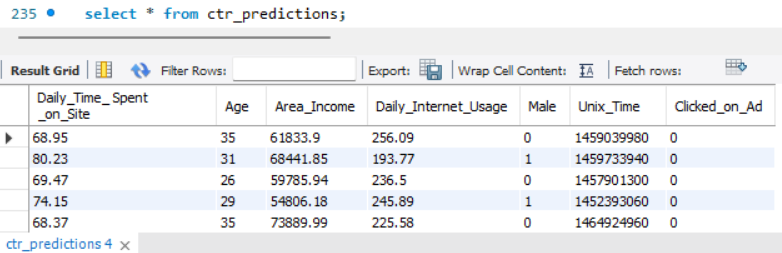
999 1

[1000 rows x 1 columns]

**Deployment of CTR cases prediction using logistic regression model through Streamlit**

****

**Saving the predicted value in MySQL database for future monitoring**

****