Advertisement Click Prediction Analysis using Machine Learning Classification Techniques

Group 25

Kalyan Krishna Karumuri
Department of Computer Science
11605269
KalyanKrishnaKarumuri@my.unt.edu

Sai Kiran Reddy Kancharla
Department of Computer Science
11605339
SaiKiranReddyKancharla@my.unt.edu

Vibha Patel Erram
Department of Computer Science
11602530
VibhaPatelErram@my.unt.edu

Vaishali Konda
Department of Computer Science
11614540
VaishaliKonda@my.unt.edu

Hemanth Kumar Kakani
Department of Computer Science
11655430
KakaniHemanthKumar@my.unt.edu

Instructor

Sayed Khushal Shah

CSCE 5310 Section 002 – Methods in Empirical Analysis
Department of Computer Science

Sayed.shah@unt.edu

University of North Texas

I. Goals and Objectives

A. Motivation:

In the Digital world, internet advertisements are one of the most important ways for people to learn about products. We have all clicked on a lot of ads that made us interested in the idea. However, we are interested in how these ads affect people in different parts of the world, with different income levels, and those that use the internet in different ways and at different ages. This project aims to predict whether a given internet user will click on an advertisement based on their features classification models like Support Vector Machine, K-Nearest Neighbors and Logistic Regression models. This problem is crucial in online advertising as an effective prediction model for ad clicks enables the targeted and personalized ad delivery, optimizing ad spend (money) and enhancing user experience.

B. Significance:

It makes way for more specialized and individualized advertising tactics by predicting a user's probability to click on an advertisement based on a variety of characteristics. The potential significance includes: Optimizing Ad Spend, Enhancing User Experience, Global Reach, Adapting to Age and Internet Usage Patterns, Industry Impact. It aligns with the evolving landscape of digital marketing, where personalization and precision are increasingly crucial for success.

C. Objectives:

The objectives for this project are very straightforward.

1. Performing Linear Regression on dataset using SPSS
2. Exploratory Data Analysis
3. Support Vector Machine (SVM) modeling training
4. K-nearest Neighbors modeling training
5. SVM and KNN model evaluation
6. Hyperparameter Tuning on SVM and KNN

II. Features

A. Related Work (Background)

The dataset is taken from Kaggle which is used for various projects like Display Advertising Challenge, Football advertising banners detection etc,. This project is inspired from real world scenarios and we took references from online sources rather previous papers.

B. Dataset

The data is a .csv file taken from Kaggle. This contains 10 columns and 1001 rows with a size of 107.42 kB. As we can see below the dataset contains the variables like Daily Time Spent on Site (numerical values), Age (numerical values), Area Income (numerical values), Daily Internet Usage (Numerical Values), Ad Topic Line (String), City (String), Male (0 - Female, 1 - Male), Country (String), Timestamp (Date), Clicked on Ad (0 – No, 1 – Yes).

Daily Time	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
68.95	35	61833.9	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	27-03-2016 00:53	0
80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	04-04-2016 01:39	0
69.47	26	59785.94	236.5	Organic bottom-line service-desk	Davidton	0	San Marin	13-03-2016 20:35	0
74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	10-01-2016 02:31	0
68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	03-06-2016 03:36	0
59.99	23	59761.56	226.74	Sharable client-driven software	Jamieberg	1	Norway	19-05-2016 14:30	0
88.91	33	53852.85	208.36	Enhanced dedicated support	Brandonstad	0	Myanmar	28-01-2016 20:59	0
66	48	24593.33	131.76	Reactive local challenge	Port Jefferybury	1	Australia	07-03-2016 01:40	1
74.53	30	68862	221.51	Configurable coherent function	West Colin	1	Grenada	18-04-2016 09:33	0
69.88	20	55642.32	183.82	Mandatory homogeneous architecture	Ramirezton	1	Ghana	11-07-2016 01:42	0
47.64	49	45632.51	122.02	Centralized neutral neural-net	West Brandonton	0	Qatar	16-03-2016 20:19	1
83.07	37	62491.01	230.87	Team-oriented grid-enabled Local Area Ne	East Theresashire	1	Burundi	08-05-2016 08:10	0
69.57	48	51636.92	113.12	Centralized content-based focus group	West Katiefurt	1	Egypt	03-06-2016 01:14	1
79.52	24	51739.63	214.23	Synergistic fresh-thinking array	North Tara	0	Bosnia an	20-04-2016 21:49	0
42.95	33	30976	143.56	Grass-roots coherent extranet	West William	0	Barbados	24-03-2016 09:31	1
63.45	23	52182.23	140.64	Persistent demand-driven interface	New Travistown	1	Spain	09-03-2016 03:41	1
55.39	37	23936.86	129.41	Customizable multi-tasking website	West Dylanberg	0	Palestinia	30-01-2016 19:20	1
82.03	41	71511.08	187.53	Intuitive dynamic attitude	Pruittmouth	0	Afghanista	02-05-2016 07:00	0
54.7	36	31087.54	118.39	Grass-roots solution-oriented conglomera	Jessicastad	1	British Ind	13-02-2016 07:53	1
74.58	40	23821.72	135.51	Advanced 24/7 productivity	Millertown	1	Russian Fe	27-02-2016 04:43	1
77.22	30	64802.33	224.44	Object-based reciprocal knowledgebase	Port Jacqueline	1	Cameroon	05-01-2016 07:52	0
84.59	35	60015.57	226.54	Streamlined non-volatile analyzer	Lake Nicole	1	Cameroon	18-03-2016 13:22	0
41.49	52	32635.7	164.83	Mandatory disintermediate utilization	South John	0	Burundi	20-05-2016 08:49	1
87.29	36	61628.72	209.93	Future-proofed methodical protocol	Pamelamouth	1	Korea	23-03-2016 09:43	0
41.39	41	68962.32	167.22	Exclusive neutral parallelism	Harperborough	0	Tokelau	13-06-2016 17:27	1
78.74	28	64828	204.79	Public-key foreground groupware	Port Danielleberg	1	Monaco	27-05-2016 15:25	0
48.53	28	38067.08	134.14	Ameliorated client-driven forecast	West Jeremyside	1	Tuvalu	08-02-2016 10:46	1

C. Detail Design of Features

The project first draft is designed in such a way that it begins by:

1. Collecting the data and preprocessing it
2. Using Linear Regression, knowing the relationship between the variables in SPSS.
3. Training the Models
4. Creating a Function for the difference between performance of the Test data and Trained data
5. Creating SVM model
6. Creating KNN model
7. Plotting the Performance Metrics
8. ROC Curve for SVM and KNN
9. Hyperparameter Tuning for SVM
10. Hyperparameter Tuning for KNN

III. Analysis

The Analysis for this draft follows a sequence for better understanding.

A. Linear Regression in SPSS

The linear regression is performed in a way to know the variables are related to the other variables in the dataset.

The Dependent variable is 'Click on Ad' and the remaining are the independent variables.

ANOVA

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	205.805	4	51.451	1158.368	<.001[b]
	Residual	44.195	995	.044		
	Total	250.000	999			

a. Dependent Variable: Clicked_on_Ad

b. Predictors: (Constant), Daily_Internet_Usage, Area_Income, Age, Daily_Time_Spent_on_Site

fig.a. ANOVA Table – SPSS

Here we got Sum of Squares value as 205.805, df value as 4, Mean Square as 51.451 and F-values as 1158.368. Here the p-value is less than 0.001 which is in turn less than 0.05. That means, there is no strong evidence to reject the null hypothesis. But, at least one of the predictor values that we took is related to the "Clicking on an Ad".

From the results we can conclude that at least on variable is related to the other variable

B. Data Exploration

Kaggle is the best source for the datasets. Our dataset advertising.csv is taken from kaggle which is a csv file. First the file is loaded in the shared folder of google colab using "/content/advertising.csv". Its structure is examined using the info(), head() and describe() functions. The info() function is used to check information about the dataset, head() function is used to display the first few rows of the dataset and the function describe() is used to provide the summary statistics of the dataset.

C. Missing Values Analysis

The missing values in the data set are summed and the values are zero for every variable which means there are no null values in the dataset.

D. Outlier Detection

The outliers are plotted the variable 'Daily Internet Usage' using the boxplot. And no outliers are detected.

E. Univariate, Bivariate and Multivariate Analysis

The univariate analysis is done in such a way that histograms are created for 'Age', Distribution for 'Age', 'Gender' and 'Area income', joint plots for 'Daily Internet Usage' vs 'Age' and 'Daily time spent on Site' vs 'Age'. Plotting a pair plot displays the relationships between different numerical variables based on the 'Clicked on Ad' target variable shows bivariate analysis. For Multivariate Analysis generated a heat map to visualize the correlation matrix between numerical variables.

F. Data Preprocessing

Unnecessary columns ('Timestamp', 'Clicked on Ad', 'Ad Topic Line', 'Country', 'City') are removed from the dataset.

G. Data Splitting and Transformation

The dataset is split into training and testing sets using 'train_test_split()' function. Column transformers 'make_column_transformer()' are employed to apply MinMaxScalar and StandardScalar to selected numerical columns. The transformed data is then split into training and testing sets.

H. Model Building & Training – Support Vector Machine

The exploration transitions into model instantiation, with a Support Vector Machine model emerging as a predicting algorithm. This model is trained on the preprocessed training data, and a comprehensive evaluation ensues. Metrics such as accuracy, precision, recall and a detailed classification report offer a nuanced understanding of the model's performance on both the training and testing sets. Precision-recall and ROC curves are instrumental in providing a visual representation of the model's strengths and limitations. The analysis goes beyond accuracy, delving into the model's ability to correctly classify positive and negative instances, crucial in an ad click prediction context.

I. Model Building & Training – KNN

The exploratory journey extends to a K-Nearest Neighbors classifier, initialized with k=5. This model is created and trained on preprocessed dataset, mirroring the approach taken with the SVM model. The performance evaluation encompasses accuracy, precision, recall and a comprehensive classification report for both training and testing sets. Precision-recall and ROC curves serve as the indispensable tools for visually assessing the model's behavior across different thresholds. The thorough evaluation ensures a profound comprehension of KNN model's strengths and potential limitations in predicting ad clicks.

J. Model Building & Training – Logistic Regression

The supplied training data (X_train and y_train) are used to train a logistic regression classifier using the 'liblinear' solver. To evaluate and provide the precision and comprehensive classification report for the training and testing sets. By calculating precision, recall, and thresholds for the logisctic regression model using the test data, the function offers an understanding of the trade-off between accuracy and recall. It also displays the precision-recall curve. This exhaustive assessment and visualization procedure helps with the interpretation and appraisal of the model's predictive power by providing a complete grasp of the model's performance on the training and the test datasets.

K. SPSS for Logistic Regression

Logistic Regression is a very useful for analyzing binary outcomes in a different of domains and it is also included in SPSS as it may be used with the variety of predictor variables, SPSS's adaptable logistic regression which helps with predictive modeling. Additionally, risk assessment benefits from its ability to estimate or identify the variables that influence particular results. This on Advertising dataset facilitates click-on add rates. We can gain the valuable insights to improve and optimize the targeting strategies and can maximize the results utilizing SPSS for the logistic regression on the advertising information.

L. Hyperparameter Tuning

Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Logistic Regression are three distinct machine learning models that need hyperparameter adjustment. The best possible combination of 'C' (regularization parameter), 'gamma' (kernel coefficient), and 'kernel' (kernel type) is found for SVM by doing a grid search across a predetermined parameter grid. Then, using the specified training and testing datasets, a new SVM model is created with the optimal parameters and trained and assessed. Grid search is used in a similar manner to determine the optimal values for the number of neighbors, weight type, and distance metric in KNN. After creating a new KNN classifier with the ideal parameters, it is trained and assessed. Finally, Logistic Regression follows a similar procedure, wherein the optimal combination of penalty, regularization strength, class weights, and solver is found using a grid search. A new Logistic Regression model is trained and evaluated using the optimal parameters that are obtained. The optimal hyperparameters for every model may be chosen thanks to this thorough process, which enhances prediction performance overall.

The culminating analysis integrates the insights gleaned from exploratory data analysis, model evaluation, and hyperparameters tuning. From the meticulous exploration of the dataset's intricacies to the strategic preparation of data for model training and comprehensive evaluation of SVM, KNN, and Logistic Regression models, each step contributes to a holistic understanding. The inclusion of hyperparameter tuning ensures the models are optimized for predictive accuracy, aligning them with the dynamic demands of predicting ad clicks in the online advertising landscape. This detailed analysis empowers decision makers with actionable insights, facilitating informed choices in the deployment of models for effective online advertising strategies.

IV. Implementation

A. *Code and Results*

The Project first draft is implemented with the following code. The its obtained good results.

1. Importing libraries

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OrdinalEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve,roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

fig.1. importing libraries

These are libraries that are used in the code.

2. Loading the data

```
[ ]  data = pd.read_csv("/content/advertising.csv")
     data.info()
```

fig.2. load dataset and get info

The data is loaded read function and also the information of this dataset is known now.

```
<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   object
 5   City                      1000 non-null   object
 6   Male                      1000 non-null   int64
 7   Country                   1000 non-null   object
 8   Timestamp                 1000 non-null   object
 9   Clicked on Ad             1000 non-null   int64
dtypes: float64(3), int64(3), object(4)
memory usage: 78.2+ KB
```

fig.3. output for get info

3. head(), describe()

By using these functions, we come to know the following outputs.

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0

Fig.4. output for head function

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	19.000000	13996.500000	104.780000	0.000000	0.00000
25%	51.360000	29.000000	47031.802500	138.830000	0.000000	0.00000
50%	68.215000	35.000000	57012.300000	183.130000	0.000000	0.50000
75%	78.547500	42.000000	65470.635000	218.792500	1.000000	1.00000
max	91.430000	61.000000	79484.800000	269.960000	1.000000	1.00000

fig.5. output for describe function

4. Missing values Analysis.

```
# Missing Values Analysis
print(data.isnull().sum())

Daily Time Spent on Site    0
Age                         0
Area Income                 0
Daily Internet Usage        0
Ad Topic Line               0
City                        0
Male                        0
Country                     0
Timestamp                   0
Clicked on Ad               0
dtype: int64
```

fig.6. summing null values

This tells us there are no null or missing values in the dataset.

5. Outlier Detection

The detection of outlier is done using boxplot from seaborn libraries.

```
# outlier Detection
sns.boxplot(x="Daily Internet Usage",data=data)
plt.title("Data Usage with outliers")
plt.show()
```

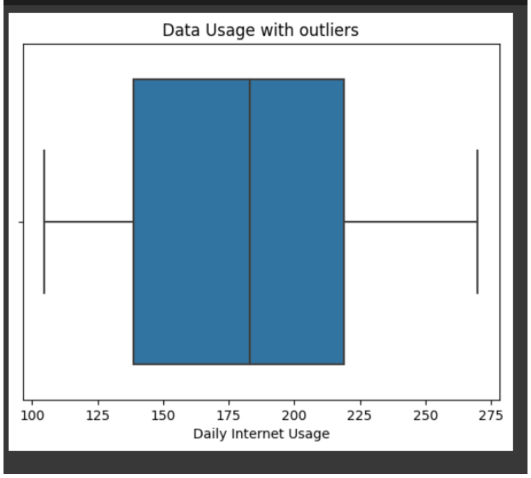

fig.7. Detecting outliers

6. Univariate Analysis

This is just basically representing one variable in a graph. And so we have done this for the variables ‘Age’ and ‘Area Income’. Both of these variables are represented in a histogram where y axis as the count and x axis as the Age and Area Income.

```
#univariate Analysis
plt.figure(figsize=(8, 5))
plt.subplot(2, 2, 1)
sns.histplot(data['Age'], bins=20, kde=True)
plt.title('Age Distribution')
plt.figure(figsize=(8, 5))
plt.subplot(2, 2, 2)
sns.histplot(data['Area Income'], kde=True)
plt.title('Income Distribution by Gender')
plt.show()
```

fig.8. Univariate Analysis

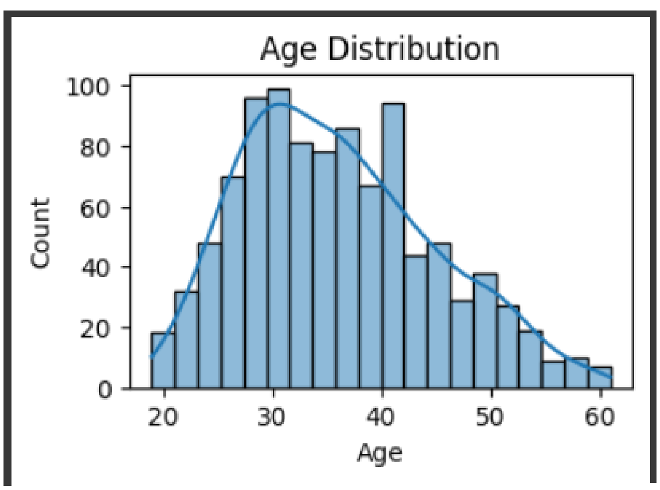

fig.9. Age distribution

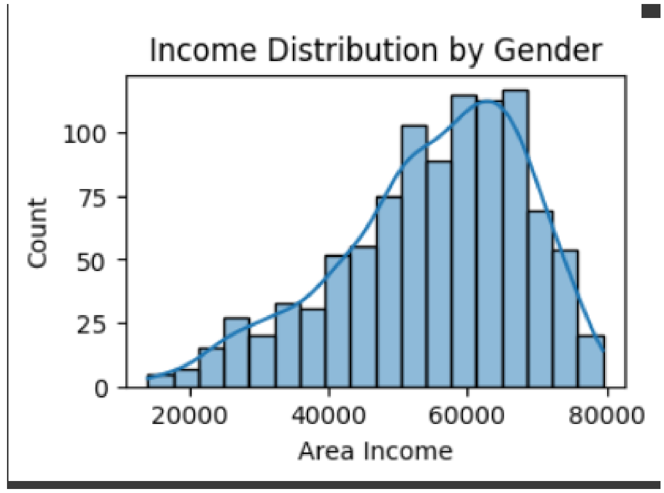

fig.10. income distribution

7. Data Distribution

The data of Age is distributed using kdeplot function from seaborn libraries.

```
# Data Distribution
sns.kdeplot(data['Age'],shade=True)
plt.title('Age Distribution')
plt.show()
```

fig.12. code for data distribution

The y-axis represents the density and the x-axis represents Age variable.

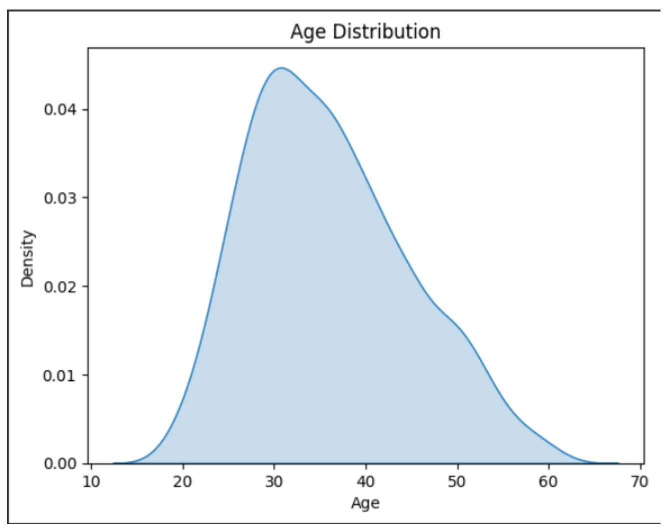

fig.13. Age Distribution using kdeplot

8. Categorical Variable Analysis

Using countplot from seaborn, the gender distribution chart is created from the variable ‘male’.

```
# Categorical Variable Analysis
plt.subplot(2,2,1)
sns.countplot(x='Male',data=data)
plt.title('Gender Distribution')
```

fig.14. Categorical Variable Analysis

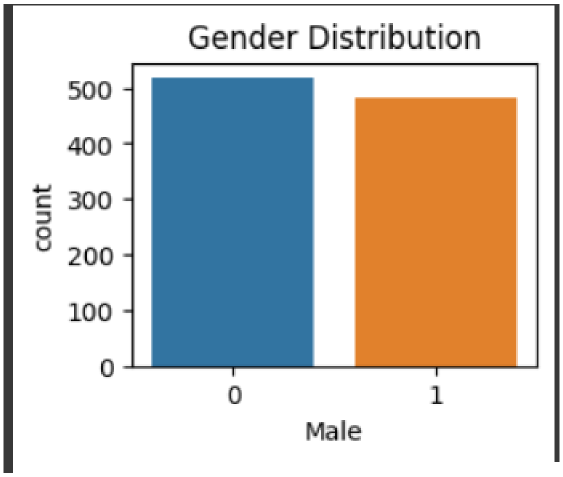

fig.15. Gender Distribution

9. Bivariate Analysis

The pair plot itself is a grid of scatterplots where each variable is plotted against every other variable in the dataset. Diagonal elements typically show the distribution of a single variable, while the off-diagonal elements show the relationships (scatter plots) between pairs of variables.

```
# Bivariate Analysis
sns.pairplot(data, hue='Clicked on Ad')
plt.show()
```

fig.16. code for Bivariate Analysis

The points are colored or marked differently based on whether the individual clicked on an ad or not, providing insights into how different variables relate to the probability of clicking on an Ad.

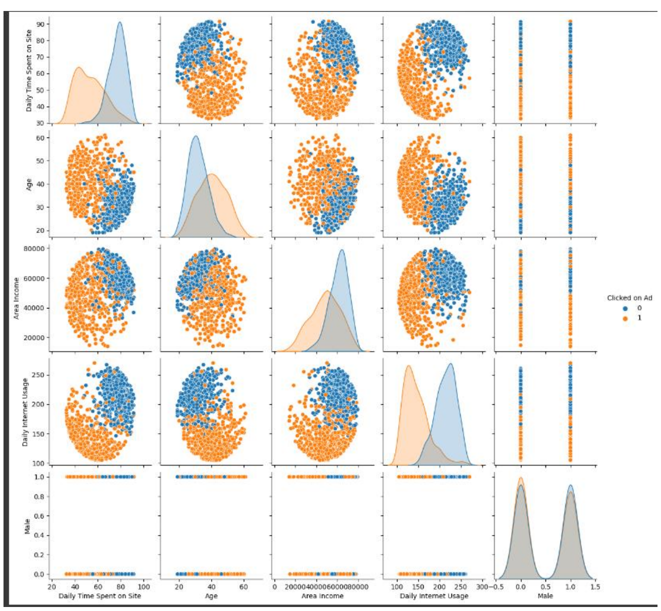

fig.17. figures for the analysis

10. Multivariate Analysis

This analysis is done with a heatmap.

```
# Multivariate Analysis
plt.figure(figsize=(7, 7))
sns.heatmap(data.corr(), annot=True)
plt.show()
```

fig.18. code for Multivariate Analysis

It is the visual representation of the correlation between different variables in the dataset. The color intensity represents the strength and direction of the correlation. Warn colors indicate the positive correlation and the cool colors represents negative correlation.

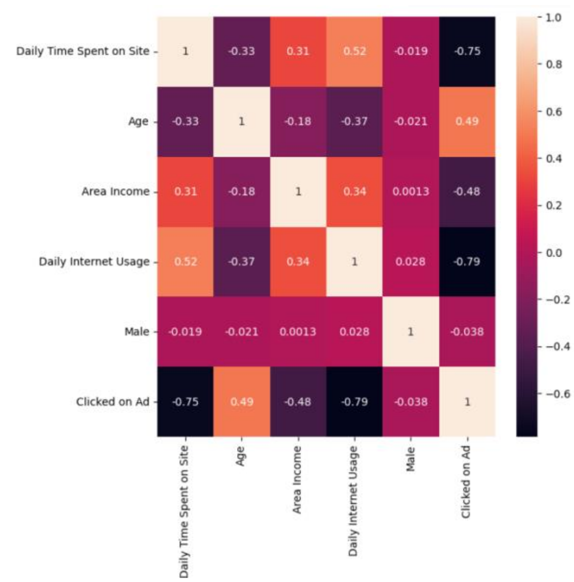

fig.19. Heatmaps

This can help in understanding the interplay between variables in the dataset and can guide further analysis or modeling decisions.

11. Splitting and Training the Data

This part prepares a machine learning dataset by splitting it into training and testing sets, selecting relevant features, and applying a column transformer to scale numeric columns using Min-Max scaling and standardization, facilitating the subsequent training and evaluation of a model predicting whether users clicked on an ad based on various features.

```
X = data.drop(['Timestamp', 'Clicked on Ad', 'Ad Topic Line', 'Country', 'City'], axis=1)
y = data['Clicked on Ad']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

num_columns = ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage', 'Male


ct = make_column_transformer(
    (MinMaxScaler(), num_columns),
    (StandardScaler(), num_columns),
    remainder='passthrough'
)

X_train = ct.fit_transform(X_train)
X_test = ct.transform(X_test)
```

fig.20. Code for Splitting and training

12. Train Data vs Test Data

This part code defines a function named `print_score` that takes a classifier (`clf`), training and testing sets (`X_train`, `y_train`, `X_test`, `y_test`), and a boolean parameter (`train`) indicating whether to evaluate the model on the training or testing set. If `train` is True, it predicts the labels on the training set using the classifier, calculates and prints the accuracy score, and generates a detailed classification report (including precision, recall, and F1-score) using the `classification_report` function from scikit-learn. The results are then displayed under the "Train Result" section. If `train` is False, it performs the same operations on the testing set and presents the results under the "Test Result" section. The accuracy score and classification report offer insights into the model's performance on either the training or testing data, aiding in the evaluation of its predictive capabilities and generalization to new, unseen data. This function provides a convenient way to assess and compare model performance on different datasets.

```
def print_score(clf, X_train, y_train, X_test, y_test, train=True):
    if train:
        pred = clf.predict(X_train)
        clf_report = pd.DataFrame(classification_report(y_train, pred, output_dict=True))
        print("\nTrain Result:")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")

    elif train==False:
        pred = clf.predict(X_test)
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
        print("\n")
        print("Test Result:")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
```

fig.20. Train vs Test

13. SVM

The `SVC} (Support Vector Classification) implementation of the Support Vector Machine (SVM) technique is used in this code to train a binary classification model. The `fit` method is used to create the `svm_model} and then fit it to the training data (`X_train} and `y_train}). The model's performance on the training and testing sets is then assessed by two calls to the `print_score` function. It produces the accuracy score for the training set along with a comprehensive classification report that includes metrics like F1-score, precision, and recall.

```
svm_model = SVC()

svm_model.fit(X_train, y_train)
print_score(svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(svm_model, X_train, y_train, X_test, y_test, train=False)
```

fig.21. SVM model building

On the testing set, it offers information on how well the model applies to fresh, untested data. By using this procedure, one may better comprehend how well the SVM model predicts outcomes on both the training dataset and an untrained dataset. The outcomes aid in evaluating the model's comprehension of training data patterns and its ability to generalize to cases that are similar but have not been seen before.

```
Train Result:
Accuracy Score: 97.57%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.966759    0.985251  0.975714    0.976005      0.975899
recall       0.985876    0.965318  0.975714    0.975597      0.975714
f1-score     0.976224    0.975182  0.975714    0.975703      0.975709
support    354.000000  346.000000  0.975714  700.000000    700.000000


Test Result:
Accuracy Score: 96.00%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.940789    0.979730      0.96    0.960260      0.960779
recall       0.979452    0.941558      0.96    0.960505      0.960000
f1-score     0.959732    0.960265      0.96    0.959998      0.960005
support    146.000000  154.000000      0.96  300.000000    300.000000
```

fig.22. SVM train and test results

With excellent accuracy and balanced precision and recall values for both classes, these findings show that the SVM model generalizes effectively to new data. When separating people based on whether they clicked on advertisements or not, the model performs admirably.

14. KNN

This code sample creates a k-Nearest Neighbors (KNN) classifier using {k=5}, which means that when it comes to making predictions, it takes into account the five closest neighbors. The instance of `knn_clf} is the `KNeighborsClassifier}. Then, the `fit` technique is used to fit the model to the training data (`X_train} and `y_train}). The training data is kept in memory during this process, which enables the model to predict things based on how close together data points are in the domain of features.

```
# Create KNN classifier with k=5 (you can choose a different value for k)
knn_clf = KNeighborsClassifier(n_neighbors=5)

# Fit the model
knn_clf.fit(X_train, y_train)

# Print scores for training and testing sets
print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
```

fig.23. KNN model Building

The performance of the KNN classifier on the training and testing sets is then assessed by making two calls to the {print_score} function. A thorough picture of the model's performance in classifying instances is provided by the function, which computes and publishes important metrics including accuracy, precision, recall, and F1-score. With the use of this study, one can determine whether the KNN classifier shows any signs of overfitting or underfitting and how effectively it generalizes to new data. By including the class labels of the five closest neighbors into the prediction process, the model's decision-making process is influenced by the selection of {k=5}. In general, this code makes it easier to comprehend the performance of the KNN model inside the given dataset.

```
Train Result:
Accuracy Score: 97.29%
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision    0.949062    1.000000  0.972857    0.974531      0.974240
recall       1.000000    0.945087  0.972857    0.972543      0.972857
f1-score     0.973865    0.971768  0.972857    0.972817      0.972829
support    354.000000  346.000000  0.972857  700.000000    700.000000

Test Result:
Accuracy Score: 94.67%
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision    0.922078    0.972603  0.946667    0.947340      0.948014
recall       0.972603    0.922078  0.946667    0.947340      0.946667
f1-score     0.946667    0.946667  0.946667    0.946667      0.946667
support    146.000000  154.000000  0.946667  300.000000    300.000000
```

fig.24. KNN train and test results

In comparison to the previous SVM model, the K-Nearest Neighbors (KNN) classifier with k=5 exhibits slightly different performance characteristics on the training and testing sets. The KNN model does somewhat worse in terms of accuracy on both training and testing sets when compared to the SVM model. KNN produces a similar F1-score for Class 1 in the training set, exhibiting better accuracy but worse recall. By balancing recall and precision fairly for both classes, KNN maintains a competitive performance in the testing set.

15. Logistic Regression

The implementation of Logistic Regression is done by training dataset which was also used for SVM, KNN models and evaluates its performance on both the training and test sets. Here we used scikit-learn's Logistic Regression class to create a logistic regression classifier. The 'liblinear' library is used for optimization of data. The logistic regression model is then trained on the provided training data('X_train' and 'y_train'- as mentioned above), using the 'fit' method. Later we printed the performance metrics of the logistic regression model on both the training and test datasets for further comparison of models and also for a comprehensive assessment of the model's performance on both the data it was trained on and new, unseen data.

```
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(solver='liblinear')
lr_clf.fit(X_train, y_train)

print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
```

fig.25. Logistic Regression model Building

Here, by evaluating the results of the logistic regression model on the training and test sets, we got an accuracy of 97.43% for the training set and 97% for the test set. Additional insights for precision, recall, F1-score for both classes (0 and 1), along with support counts, can also be seen below in the classification report, which shows the effectiveness of binary classification. The classification report for the test set mirrors the training set, which indicates that the model's ability to generalize well to new, unseen data is also evident.

```
Train Result:
Accuracy Score: 97.43%
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision    0.964088    0.985207  0.974286    0.974648      0.974527
recall       0.985876    0.962428  0.974286    0.974152      0.974286
f1-score     0.974860    0.973684  0.974286    0.974272      0.974279
support    354.000000  346.000000  0.974286  700.000000    700.000000

Test Result:
Accuracy Score: 97.00%
CLASSIFICATION REPORT:
                   0           1  accuracy   macro avg  weighted avg
precision    0.959732    0.980132      0.97    0.969932      0.970204
recall       0.979452    0.961039      0.97    0.970246      0.970000
f1-score     0.969492    0.970492      0.97    0.969992      0.970005
support    146.000000  154.000000      0.97  300.000000    300.000000
```

fig.26. Train and Test results for Logistic Regrssion

In comparison to the previous SVM model, the K-Nearest Neighbors (KNN) classifier with k=5 exhibits slightly different performance characteristics on the training and testing sets. The KNN model does somewhat worse in terms of accuracy on both training and testing sets when compared to the SVM model and Logistic Regression model. Overall, the SVM model shows better accuracy when compared to other models.

16. Plotting the Performance Metrics

This code defines a function named plot_precision_recall_vs_threshold that takes precision, recall, and threshold values as inputs and plots precision and recall against different threshold values. It's a useful function to visualize the trade-off between precision and recall at various decision thresholds. The code then applies this function to two different models, SVM (svm_model), K-Nearest Neighbors (knn_clf), and Logistic Regression (lr_clf). It uses the precision_recall_curve function from scikit-learn to calculate precision and recall at different thresholds for both models.For the SVM model, a subplot is created with two side-by-side plots.

```
# Function to plot precision and recall against different thresholds
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
  # Plot precision values against thresholds in blue dashed line
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    # Plot recall values against thresholds in green dashed line
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    # Set x-axis label as "Threshold"
    plt.xlabel("Threshold")
    # Place legend in the upper left corner
    plt.legend(loc="upper left")
    # Set the title of the plot
    plt.title("Precisions/recalls tradeoff")
# Calculate precision, recall, and thresholds for the SVM model
precisions, recalls, thresholds = precision_recall_curve(y_test, svm_model.predict(X_test))

plt.figure(figsize=(15, 10))
# Create a 2x2 subplot, and plot precision-recall tradeoff for SVM model
plt.figure(figsize=(15, 10))
plt.subplot(3, 2, 1)
plot_precision_recall_vs_threshold(precisions, recalls, thresholds)
# Create a subplot for the PR curve of the SVM model
plt.subplot(3, 2, 2)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");

# Calculate precision, recall, and thresholds for the KNN model
precisions1, recalls1, thresholds1 = precision_recall_curve(y_test, knn_clf.predict(X_test))
# Create a 4x2 subplot, and plot precision-recall tradeoff for KNN model
plt.figure(figsize=(15, 10))
plt.subplot(3, 2, 3)
plot_precision_recall_vs_threshold(precisions1, recalls1, thresholds1)
# Create a subplot for the PR curve of the KNN model
plt.subplot(3, 2, 4)
plt.plot(precisions1, recalls1)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");
```

```python
# Calculate precision, recall, and thresholds for the Logistic Regression model
precisions2, recalls2, thresholds2 = precision_recall_curve(y_test, lr_clf.predict(X_test))

plt.figure(figsize=(15, 10))
plt.subplot(3, 2, 5)
plot_precision_recall_vs_threshold(precisions2, recalls2, thresholds2)

plt.subplot(3, 2, 6)
plt.plot(precisions, recalls)
plt.xlabel("Precision")
plt.ylabel("Recall")
plt.title("PR Curve: precisions/recalls tradeoff");
```

fig.27. Performance metrics for SVM, KNN and Logistic Regression

Understanding the trade-off between the three measures is made possible by the first graphic, which illustrates how recall and accuracy change with various thresholds. The link between recall and accuracy across various threshold levels is shown in the second graphic, which is an accuracy-Recall (PR) curve. Using the same framework for all the classification models below their respective subplots can be seen below:..

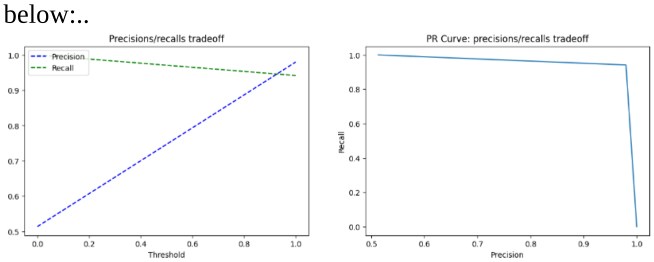

fig.28. Graphs for SVM

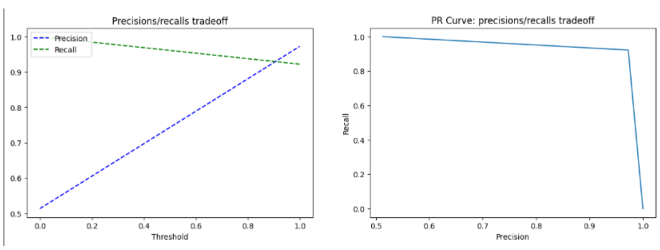

fig.29. graphs for KNN

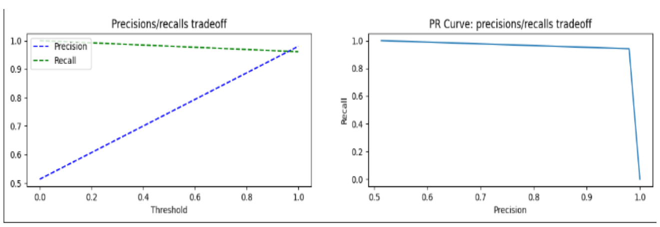

fig.30. graphs for Logistic Regression

The interpretation of how the decision threshold selection affects the models' memory and accuracy is aided by these drawings. Specifically, the PR curve shows the trade-off between recall and accuracy for various threshold values and offers a thorough overview of the overall performance. In real-world circumstances, where the focus may be on optimizing accuracy, recall, or striking a fair trade-off between the two measures, these charts are essential for making well-informed judgments regarding the model's behavior.

17. ROC-Curve

The primary objective of this code is to visually evaluate the performance of two models in binary classification tasks: SVM (svm_model) and K-Nearest Neighbors (knn_clf)—by visualizing their Receiver Operating Characteristic (ROC) curves. To generate ROC curves, the plot_roc_curve function requires three inputs: True Positive Rate (TPR), False Positive Rate (FPR), and an optional label. It uses two plots stacked vertically to define a subplot. The function creates a dashed line to represent random chance (no discrimination) and shows the ROC curve on the upper subplot to show TPR against FPR. The upper subplot now has axes labels and a title.

```python
# Function to plot ROC curve
def plot_roc_curve(fpr, tpr, label=None):
  # Create a subplot for ROC curve
    plt.subplot(3, 1, 1)
    # Plot the ROC curve with specified label
    plt.plot(fpr, tpr, linewidth=2, label=label)
     # Plot the diagonal dashed line
    plt.plot([0, 1], [0, 1], "k--")
    # Set axis limits and labels
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    # Set the title of the ROC curve
    plt.title('ROC Curve-SVM')
# Calculate ROC curve values for the SVM model
fpr, tpr, thresholds = roc_curve(y_test, svm_model.predict(X_test))
# Create a figure for the SVM ROC curve
plt.figure(figsize=(9,6));
plot_roc_curve(fpr, tpr)
plt.show();

# Calculate ROC curve values for the KNN model
def plot_roc_curve(fpr, tpr, label=None):
    plt.subplot(3, 1, 2)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve-KNN')

fpr, tpr, thresholds = roc_curve(y_test, knn_clf.predict(X_test))
# Create a figure for the KNN ROC curve
plt.figure(figsize=(9,6));
plot_roc_curve(fpr, tpr)
plt.show();


# Calculate ROC curve values for Logistic Regression Model
def plot_roc_curve(fpr, tpr, label=None):
    plt.subplot(3, 1, 3)
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], "k--")
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
# Create a figure for the Logistic Regression ROC curve
fpr, tpr, thresholds = roc_curve(y_test, lr_clf.predict(X_test))
plt.figure(figsize=(9,6));
plot_roc_curve(fpr, tpr)
plt.show();
```

fig.31. ROC plot for SVM, KNN and Logistic Regression

Based on predictions made on the testing set, the FPR, TPR, and thresholds for the SVM model are determined using the roc_curve function from scikit-learn. Using plt.figure(figsize=(9,6)), a figure with a single subplot (top subplot) is produced. To draw the ROC curve for the SVM model, the plot_roc_curve function is called. Similar to the SVM model, the KNN model uses the roc_curve function to determine thresholds, TPR, and FPR based on testing set predictions. Using plt.figure(figsize=(9,6)), another figure is produced with a single subplot (bottom subplot). To draw the ROC curve for the KNN model, the plot_roc_curve function is invoked.

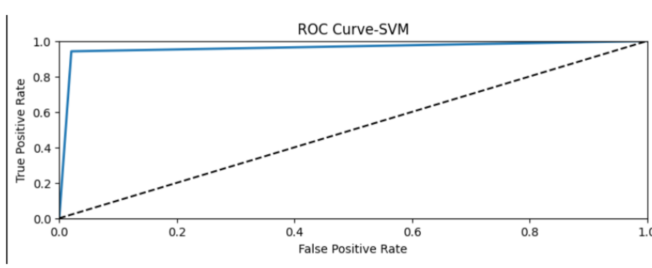

fig.32. ROC-Curve SVM

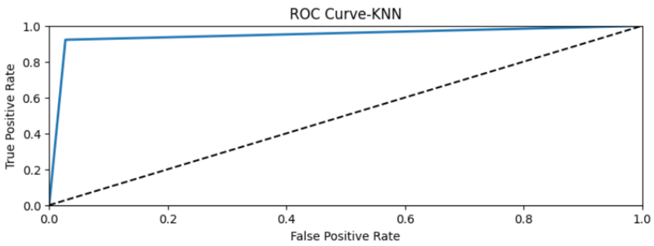

fig.33. ROC-Curve KNN

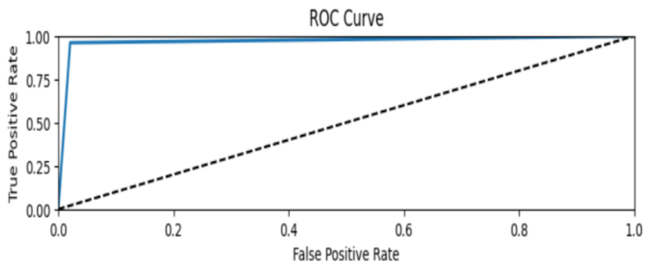

fig.34. ROC-Curve Logistic Regression

The true positive rate against false positive rate trade-off at various categorization levels is depicted by these ROC curves. A model that performs better will have a curve that is more in line with the upper-left corner, signifying lower false positive rates and greater true positive rates at different decision thresholds. The ROC curve comparison between the KNN, Logistic Regression and SVM models sheds light on how well each model can discriminate in a scenario involving binary classification.

```python
# Code for combined ROC
# ROC Curve for SVM
svm_probs = svm_model.predict(X_test)
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test, svm_probs)

# ROC Curve for KNN
knn_probs = knn_clf.predict(X_test)
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, knn_probs)

# ROC Curve for Logistic Regression
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, lr_clf.predict(X_test))

# Plot ROC Curves for comparison
plt.figure(figsize=(10, 7))
plt.plot(fpr_svm, tpr_svm, label='SVM')
plt.plot(fpr_knn, tpr_knn, label='KNN')
plt.plot(fpr_lr, tpr_lr, label='Logistic Regression')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```

fig.35. ROC Plot for the three models

Here, we compared the three models ROC curve which we produced earlier to get a better understanding, as we can see we got logistic regression model has high true positive rate compared to other models and KNN has least true positive rates.

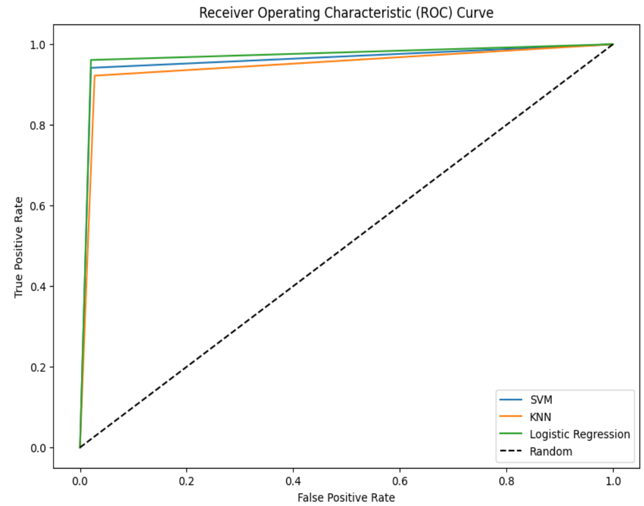

fig.36. ROC Curves Comparisons

18. Hyperparameter Tuning for SVM

By considering several combinations of hyperparameters, the approach guarantees that the SVM model is optimized for performance. The kernel type, the kernel coefficient gamma, and the regularization parameter C are all represented by various values on the parameter grid. Using 5-fold cross-validation and accuracy optimization, the GridSearchCV is used to find the ideal set of hyperparameters. Next, using the optimal hyperparameters, the model is fitted to the training set of data. After extracting the optimum grid search parameters, a new SVM model (best_svm_model) is built using these ideal parameters. The training set of data is then used to fit this model. A more reliable and efficient SVM model can be created thanks to the grid search's outcomes, which may enhance the model's ability to forecast fresh, untested data.

```python
#define Parameter Grid
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [0.1, 0.01, 0.001], 'kernel':['linear', 'rbf']}
# create svm model
svm_model = SVC()
#create GridSearchCV
grid_search = GridSearchCV(svm_model, param_grid, cv=5, scoring='accuracy', verbose=1)
# fit the model
grid_search.fit(X_train, y_train)
# Get the best parameters
best_params = grid_search.best_params_
# create new svm model with best parameters
best_svm_model = SVC(C=best_params['C'], gamma = best_params['gamma'], kernel=best_params['ker
# fitting svm model with best parameters
best_svm_model.fit(X_train, y_train)
# Print Scores for Train and Test data
print_score(best_svm_model, X_train, y_train, X_test, y_test, train=True)
print_score(best_svm_model, X_train, y_train, X_test, y_test, train=False)
```

fig.37. Hyperparameter Tuning-SVM

19. Hyperparameter Tuning for KNN

This code explores various values for the number of neighbors, weights, and distance metric in order to do hyperparameter tuning for a K-Nearest Neighbors (KNN) classifier using GridSearchCV. Then, using the F1-score metric, it builds a new KNN model with the optimal parameters and assesses how well it performs on the training and testing sets. After fitting the data to the improved KNN model, a more precise and efficient classifier is produced.

```python
# Create KNN classifier
knn_clf = KNeighborsClassifier()

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # You can adjust the range of neighbors
    'weights': ['uniform', 'distance'],
    'p': [1, 2]  # 1 for Manhattan distance, 2 for Euclidean distance
}

# Create GridSearchCV
knn_cv = GridSearchCV(
    estimator=knn_clf,
    param_grid=param_grid,
    scoring='f1',  # You can choose another scoring metric
    verbose=1,
    n_jobs=-1,
    cv=10
)

# Fit the model
knn_cv.fit(X_train, y_train)

# Get the best parameters
best_params_knn = knn_cv.best_params_
print(f"Best parameters: {best_params_knn}")

# Create a new KNN classifier with the best parameters
knn_clf = KNeighborsClassifier(**best_params_knn)

# Fit the model with the training data
knn_clf.fit(X_train, y_train)

# Print scores for training and testing sets
print_score(knn_clf, X_train, y_train, X_test, y_test, train=True)
print_score(knn_clf, X_train, y_train, X_test, y_test, train=False)
```

fig.38. Hyperparameter Tuning- KNN

20. Hyperparameter Tuning for Logistic Regression

Grid search is used to find the best hyperparameters for training and fine-tuning a Logistic Regression classifier. Options for class weights, solver type ('liblinear' or'saga'), regularization strength ('C'), penalty type ('l1' or 'l2'), and class weights are all included in the parameter grid. The model's performance is optimized based on the F1 score by thoroughly searching through the parameter combinations using the GridSearchCV. A new Logistic Regression classifier is then instantiated and fitted to the training set of data using the optimal hyperparameters found by the grid search. The next assessment entails printing detailed results for the training and testing sets, including accuracy, precision, recall, and F1 score.

```python
# Create Logistic Regression classifier
lr_clf = LogisticRegression()

# Define the parameter grid
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.5, 0.6, 0.7, 0.8],
    'class_weight': [{1: 0.5, 0: 0.5}, {1: 0.4, 0: 0.6}, {1: 0.6, 0: 0.4}, {1: 0.7, 0: 0.3}],
    'solver': ['liblinear', 'saga']
}

# Create GridSearchCV
lr_cv = GridSearchCV(
    estimator=lr_clf,
    param_grid=param_grid,
    scoring='f1',
    verbose=1,
    n_jobs=-1,
    cv=10
)

# Fit the model
lr_cv.fit(X_train, y_train)

# Get the best parameters
best_params_lr = lr_cv.best_params_
print(f"Best parameters: {best_params_lr}")

# Create a new Logistic Regression classifier with the best parameters
lr_clf = LogisticRegression(**best_params_lr)

# Fit the model with the training data
lr_clf.fit(X_train, y_train)

# Print scores for training and testing sets
print_score(lr_clf, X_train, y_train, X_test, y_test, train=True)
print_score(lr_clf, X_train, y_train, X_test, y_test, train=False)
```

fig.39. Hyperparameter Tuning - Logistic Regression

By using a careful approach, it is ensured that the Logistic Regression model is optimized to produce the best results on the provided datasets, balancing precision and recall.

21. Hyperparameter tuning results – SVM, KNN and Logistic Regression

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits

Train Result:
Accuracy Score: 97.43%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.964088    0.985207  0.974286    0.974648      0.974527
recall       0.985876    0.962428  0.974286    0.974152      0.974286
f1-score     0.974860    0.973684  0.974286    0.974272      0.974279
support    354.000000  346.000000  0.974286  700.000000    700.000000


Test Result:
Accuracy Score: 97.00%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.959732    0.980132      0.97    0.969932      0.970204
recall       0.979452    0.961039      0.97    0.970246      0.970000
f1-score     0.969492    0.970492      0.97    0.969992      0.970005
support    146.000000  154.000000      0.97  300.000000    300.000000
```

fig.40. Train and Test Results using Best Params – SVM

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits
Best parameters: {'n_neighbors': 9, 'p': 2, 'weights': 'uniform'}

Train Result:
Accuracy Score: 97.00%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.948787    0.993921      0.97    0.971354      0.971096
recall       0.994350    0.945087      0.97    0.969718      0.970000
f1-score     0.971034    0.968889      0.97    0.969962      0.969974
support    354.000000  346.000000      0.97  700.000000    700.000000


Test Result:
Accuracy Score: 94.67%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.916667    0.979167  0.946667    0.947917      0.948750
recall       0.979452    0.915584  0.946667    0.947518      0.946667
f1-score     0.947020    0.946309  0.946667    0.946664      0.946655
support    146.000000  154.000000  0.946667  300.000000    300.000000
```

fig.41. Train and Test Results using Best Params – KNN

```
Fitting 10 folds for each of 64 candidates, totalling 640 fits
Best parameters: {'C': 0.6, 'class_weight': {1: 0.5, 0: 0.5}, 'penalty': 'l2', 'solver': 'saga'}

Train Result:
Accuracy Score: 97.29%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.961433    0.985163  0.972857    0.973298      0.973162
recall       0.985876    0.959538  0.972857    0.972707      0.972857
f1-score     0.973501    0.972182  0.972857    0.972841      0.972849
support    354.000000  346.000000  0.972857  700.000000    700.000000


Test Result:
Accuracy Score: 97.00%
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.953642    0.986577      0.97    0.970110      0.970549
recall       0.986301    0.954545      0.97    0.970423      0.970000
f1-score     0.969697    0.970297      0.97    0.969997      0.970005
support    146.000000  154.000000      0.97  300.000000    300.000000
```

fig.42. Train and Test Results using Best Params – Logistic Regression

Comparing the results of the two models:

Performance on both training and testing sets is exceptionally good for all three machine learning models: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Logistic Regression. With F1-scores of 0.974 and 0.970, respectively, SVM demonstrated balanced precision and recall, achieving an accuracy of 97.43% on the training set and 97.00% on the test set. KNN demonstrated strong performance, but with a little lower accuracy on the test set, with accuracy of 97.00% on the training set and 94.67% on the test set. With an accuracy of 97.29% on the training set and 97.00% on the test set, logistic regression demonstrated reliable and effective results on both datasets.

While SVM and Logistic Regression have somewhat greater accuracies and F1-scores than KNN, all three models show good prediction ability overall. Specific use-case needs and interpretability factors may influence which of these models is selected.

The accuracy of the improved SVM model is marginally higher on the training and testing sets. Higher F1-scores are obtained by the SVM model because it shows better balanced accuracy and recall for both classes. The KNN model and Logistic Regression exhibits a trade-off between precision and recall, particularly for Class 1, even if it achieves a fair accuracy.

The particular needs of the application determine which of the two models is best. The SVM model could be chosen if a balanced performance between accuracy and recall is important.

Nonetheless, the KNN model might be taken into consideration if interpretability or computing economy are of utmost importance.

22. SPSS results on Logistic Regression

Here we got Chi-Square value as 623.703 which is large, that means there is a significant relation between the predictive variables and the outcome variables. Degree of Freedom value is 131 which are the number of parameters that are estimated in the model. We got the significant value as <0.001 which is less than 0.05 that concludes that the entire model which also includes predictor variables is significant for predicting the click-on Ad variable in the advertising data. From classification table, we can observe that there are 434 cases which were correctly classified as zero while 66 cases were wrongly classified as 1, for cases when the true outcome(observed) is 0 (Did not click). There are 500 actual occurrences in total with a 0 result. 96 cases were mistakenly labeled as 0 and 404 cases were accurately classified as 1 for cases where the true outcome is 1(Clicked on Ad). There are actual in total with a result of 1 is 500. The predictions made by the model using the logistic regression analysis are displayed in this section. 96 cases were wrongly predicted as 0 (False Negatives) while 434 cases correctly predicted as 0 (True Positives) and 404 cases were correctly predicted as 1(True Positives) and 66 cases were incorrectly predicted as 1(False Positives). The Overall percentage of correct predictions that are made by this model is 83.8%.

Block 1: Method = Enter

Omnibus Tests of Model Coefficients

		Chi-square	df	Sig.
Step 1	Step	623.703	131	<.001
	Block	623.703	131	<.001
	Model	623.703	131	<.001

Model Summary

Step	-2 Log likelihood	Cox & Snell R Square	Nagelkerke R Square
1	762.591[a]	.464	.619

a. Estimation terminated at iteration number 20 because maximum iterations has been reached. Final solution cannot be found.

Classification Table[a]

	Observed		Predicted ClickedonAd 0	1	Percentage Correct
Step 1	ClickedonAd	0	434	66	86.8
		1	96	404	80.8
	Overall Percentage				83.8

a. The cut value is .500

fig.43. SPSS results – Logistic Regression

V. CONCLUSIONS

In conclusion, Strong predictive skills are shown by the Support Vector Machine (SVM), K-Nearest Neighbors (KNN), and Logistic Regression models when it comes to predicting clicks on online advertisements. With their balanced precision-recall tradeoffs and excellent accuracy, SVM and Logistic Regression stand out as particularly interesting options for predicting consumer involvement with online marketing. In the context of digital advertising, where precise forecasting enables the distribution of tailored and targeted advertisements, these models can be useful instruments. These models are consistent with the project's main objective of comprehending the effects of internet advertisements on users across various demographics and geographical locations by optimizing ad spend and improving user experience. The strong performance of SVM and Logistic Regression indicates that they may be applied in practical settings, which has the potential to greatly increase the efficacy and efficiency of online advertising campaigns globally.

VI. Project Management

A. *Responsibility*

	TASK	PERSON
Work Completed For the 70%	Documentation, KNN Model, Roc-Curve, Performance Metrics	Kalyan Krishna Karumuri
	Documentation, Hyperparameter Tuning, SVM Model, ROC – Curve	Sai Kiran Reddy Kancharla
	Documentation, SVM Model, Hyperparameter Tuning, Linear Regression in SPSS	Vibha Patel Erram
	Documentation, SVM Model, EDA, Data Preprocessing	Hemanth Kakani
	Documentation, Data Preprocessing, Performance Metrics	Vaishali Konda
Work Completed for remaining 30%	Documentation, model building, model evaluation,	Kalyan Krishna Karumuri
	Documentation, spss analysis, Roc curve comparison	Sai Kiran Reddy Kancharla
	Documentation, spss analysis, Roc-curve Comparison	Vibha Patel Erram
	Documentation, model evaluation, Hyperparameter tuning	Hemanth Kakani
	Documentation, model evaluation, Hyperparameter tuning	Vaishali Konda

B. *Contributions on completed work*

Kalyan Krishna Karumuri	100%
Sai Kiran Reddy Kancharla	100%
Vibha Patel Erram	100%
Vaishali Konda	100%
Hemanth Kumar Kakani	100%

REFERENCES

Advertising Dataset:
https://www.kaggle.com/datasets/gabrielsantello/advertisement-click-on-ad/

KNN model:
https://www.geeksforgeeks.org/k-nearest-neighbours/
https://www.w3schools.com/python/python_ml_knn.asp

SVM model:
https://www.geeksforgeeks.org/introduction-to-support-vector-machines-svm/
https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_classification_algorithms_support_vector_machine.htm

Logistic regression model:
https://www.tensorflow.org/guide/core/logistic_regression_core

Hyperparameters Tuning:
https://keras.io/keras_tuner/

EDA:
https://www.geeksforgeeks.org/exploratory-data-analysis-eda-types-and-tools/
https://www.javatpoint.com/workflow-of-data-analytics

GitHub:
For 70 percent
https://github.com/kk0858/Project-Group-25
For Full