Data Science Project - 12/6/2023

Project title: "AdClick Predictor: Optimizing Advertising Efficiency with Predictive Modeling"

Problem description: The business problem we are addressing is the need for effective online advertising targeting. Businesses invest significant resources in online advertising, and they want to ensure that their advertisements reach users who are more likely to engage by clicking on them. This is crucial for maximizing the return on investment (ROI) in advertising expenditures.

Objective: The objective of the project is to develop a machine learning model capable of predicting whether an internet user will click on a given advertisement. By achieving this, the project aims to provide businesses with a tool that enhances the precision of their ad targeting strategies. Specifically, the model should assist in identifying users who are more likely to click on the ad, allowing businesses to optimize their advertising campaigns, allocate resources more efficiently, and ultimately improve the overall effectiveness of online advertising efforts.

Data Set source: Kaggle
https://www.kaggle.com/datasets/gabrielsantello/advertisement-click-on-ad

Step 1: Data exploration

- Imported the libraries necessary including pandas, numpy, matplotlib and seaborn as seen in Figure 1 below

```
In [1]:  #import Librariries
         import pandas as pd
         import numpy as np
```

```
In [2]:  import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
```

Figure 1

- I loaded the data set using the pandas library and read the csv file
- I checked the head of the read ad_data using .head() as seen in Figure 2 below

```
In [3]:  #Reading the data file
         ad_data = pd.read_csv('advertising_data.csv')
```

```
In [4]:  # Check the head of the ad_data
         ad_data.head()
```

Out[4]:

	Daily Time Spent on Site	Age	Area Income	Gender	Daily Internet Usage	Ad Topic Line	City	Country	Timestamp	Clicked on Ad
0	68.95	35.0	61833.90	Female	256.09	Cloned 5thgeneration orchestration	Wrightburgh	Tunisia	3/27/2016 0:53	0
1	80.23	31.0	68441.85	Male	193.77	Monitored national standardization	West Jodi	Nauru	4/4/2016 1:39	0
2	69.47	26.0	59785.94	Female	236.50	Organic bottom-line service-desk	Davidton	San Marino	3/13/2016 20:35	0
3	74.15	29.0	54806.18	Male	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	Italy	1/10/2016 2:31	0
4	68.37	35.0	73889.99	Female	225.58	Robust logistical utilization	South Manuel	Iceland	6/3/2016 3:36	0

Figure 2

- I described the data using info()
- The data set has a categorical column Gender
- The columns have 1000 entries and the Age column has 992 which means it has missing values as seen in the Figure 3 below

```
In [5]:  #use Info and describe() on ad_data
         ad_data.info()

<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                       992 non-null    float64
 2   Area Income               1000 non-null   float64
 3   Gender                    1000 non-null   object 
 4   Daily Internet Usage      1000 non-null   float64
 5   Ad Topic Line             1000 non-null   object 
 6   City                      1000 non-null   object 
 7   Country                   1000 non-null   object 
 8   Timestamp                 1000 non-null   object 
 9   Clicked on Ad             1000 non-null   int64  
dtypes: float64(4), int64(1), object(5)
memory usage: 78.2+ KB
```

Figure 3

Step 2: Data Preprocessing

- I got the sum of all the missing values using isnull() and sum()
- I established the total of missing values in 'Age ' is 8 entries as seen in the Figure 4 below

```
In [6]:  # isnull() and sum() to display the total of missing data
         ad_data.isnull().sum()

Out[6]: Daily Time Spent on Site    0
        Age                         8
        Area Income                 0
        Gender                      0
        Daily Internet Usage        0
        Ad Topic Line               0
        City                        0
        Country                     0
        Timestamp                   0
        Clicked on Ad               0
        dtype: int64
```

Figure 4

- I calculated the mean of 'Age' using mean()
- I chose to round it off to 1 decimal place to look like the rest of the age entries
- Then I filled the missing values in the 'Age' column with the rounded mean as seen in Figure 5 below

```
In [7]:  #Calculate the  mean of the 'Age' column
         mean_age=ad_data['Age'].mean()

In [8]:  mean_age

Out[8]:  35.96975806451613

In [9]:  #Round the mean value to 2 decimal places
         meanAge= round(mean_age,1)

In [10]: meanAge

Out[10]: 36.0

In [11]: # Fill the missing values  in the 'Age' column wih the rounded mean.
         ad_data['Age'].fillna(meanAge, inplace=True)
```

Figure 5

- After I double checked to confirm that all missing values are filled using isnull() and sum() and the data frame returned no missing values in any columns as seen in Figure 6 below

In [13]:
```
#Double check if there are no missing values anymore
ad_data.isnull().sum()
```

Out[13]:
```
Daily Time Spent on Site    0
Age                         0
Area Income                 0
Gender                      0
Daily Internet Usage        0
Ad Topic Line               0
City                        0
Country                     0
Timestamp                   0
Clicked on Ad               0
dtype: int64
```

Figure 6

- Dealing with categorical feature 'Gender'
- I used encoding to convert Gender it to dummy variable
- This helps the machine learning algorithm to understand it and be able to use it i the prediction as seen in Figure 7 below

```
In [15]:  pd.get_dummies(ad_data['Gender'])
```

Out[15]:

	Female	Male
0	1	0
1	0	1
2	1	0
3	0	1
4	1	0
...	...	...
995	0	1
996	0	1
997	0	1
998	1	0
999	1	0

1000 rows × 2 columns

Figure 7

- I assigned a new variable sex to our encoded column meaning is Male=1 is Male and Male =0 is Female
- Since the machine learning algorithm can run into multicollinearity I dropped the Female column and prevented it from being a perfect predictor as seen in Figure 8 below

In [18]:
```
sex=pd.get_dummies(ad_data['Gender'], drop_first=True)
```

In [19]:
```
sex
```

Out[19]:

	Male
0	0
1	1
2	0
3	1
4	0
...	...
995	1
996	1
997	1
998	0
999	0

1000 rows × 1 columns

In [20]:
```
#add the new column 'sex' to ad_data using concat()
ad_data = pd.concat([ad_data,sex], axis=1)
```

Figure 8

- I added the new column 'Male' to ad_data using concat()
- Then I displayed a new data frame bearing it as seen in figure 9 below

```
In [21]: ad_data
```

Out[21]:

	Daily Time Spent on Site	Age	Area Income	Gender	Daily Internet Usage	Ad Topic Line	City	Country	Timestamp	Clicked on Ad	Male
0	68.95	35.0	61833.90	Female	256.09	Cloned 5thgeneration orchestration	Wrightburgh	Tunisia	3/27/2016 0:53	0	0
1	80.23	31.0	68441.85	Male	193.77	Monitored national standardization	West Jodi	Nauru	4/4/2016 1:39	0	1
2	69.47	26.0	59785.94	Female	236.50	Organic bottom-line service-desk	Davidton	San Marino	3/13/2016 20:35	0	0
3	74.15	29.0	54806.18	Male	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	Italy	1/10/2016 2:31	0	1
4	68.37	35.0	73889.99	Female	225.58	Robust logistical utilization	South Manuel	Iceland	6/3/2016 3:36	0	0
...	...	...	...	...	...	...	...	...	...	...	...
995	72.97	30.0	71384.57	Male	208.58	Fundamental modular algorithm	Duffystad	Lebanon	2/11/2016 21:49	1	1
996	51.30	45.0	67782.17	Male	134.42	Grass-roots cohesive monitoring	New Darlene	Bosnia and Herzegovina	4/22/2016 2:07	1	1
997	51.63	51.0	42415.72	Male	120.37	Expanded intangible solution	South Jessica	Mongolia	2/1/2016 17:24	1	1
998	55.55	19.0	41920.79	Female	187.95	Proactive bandwidth-monitored policy	West Steven	Guatemala	3/24/2016 2:35	0	0
999	45.01	26.0	29875.80	Female	178.35	Virtual 5thgeneration emulation	Ronniemouth	Brazil	6/3/2016 21:43	1	0

1000 rows × 11 columns

Figure 9

- Since I do not need the original 'Gender' column, I dropped it using drop()
- Then displayed the new data frame with only the encoded categorical value as seen in figure 10 below:

```
In [22]: # Drop the 'Gender' column
         ad_data.drop('Gender', axis=1, inplace=True)
```

```
In [23]: #Display Data Frame after dropping 'Gender'
         print(ad_data)
```

```
996        Grass-roots cohesive monitoring     New Darlene
997           Expanded intangible solution   South Jessica
998   Proactive bandwidth-monitored policy     West Steven
999         Virtual 5thgeneration emulation     Ronniemouth

                    Country        Timestamp  Clicked on Ad  Male
0                   Tunisia   3/27/2016 0:53              0     0
1                     Nauru    4/4/2016 1:39              0     1
2                San Marino  3/13/2016 20:35              0     0
3                     Italy   1/10/2016 2:31              0     1
4                   Iceland    6/3/2016 3:36              0     0
..                      ...              ...            ...   ...
995                 Lebanon  2/11/2016 21:49              1     1
996  Bosnia and Herzegovina   4/22/2016 2:07              1     1
997                Mongolia   2/1/2016 17:24              1     1
998               Guatemala   3/24/2016 2:35              0     0
999                  Brazil   6/3/2016 21:43              1     0

[1000 rows x 10 columns]
```

Figure 10

Step 3: Data Visualization

- Create a Histogram of the Age using matplotlib
- Age is normally distributed around 30 and 35 years as seen in the Figure 11 below

```
In [25]: # Create a Histogram of the Age
         ad_data['Age'].plot.hist(bins=30)

Out[25]: <AxesSubplot:ylabel='Frequency'>
```

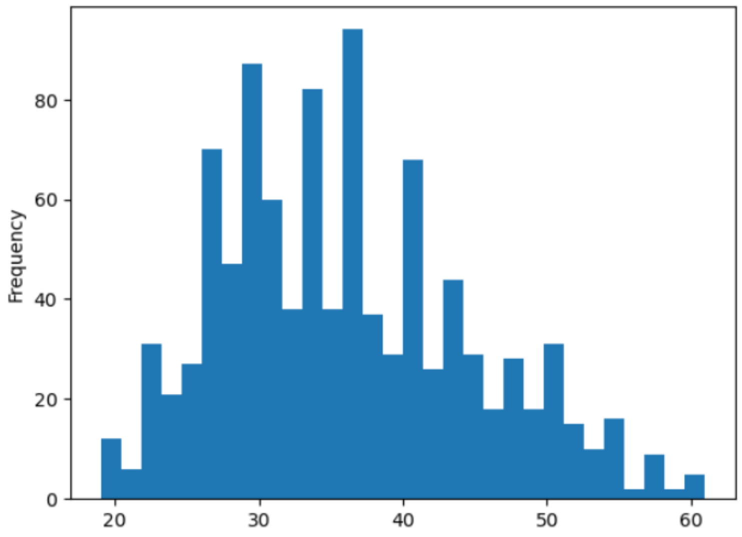

Figure 11

- I Created a joint plot showing Area Income as y-axis versus Age ax x-axis using seaborn library
- It's a little scattered but it shows that people start earning when they get to their 20s, as they get older the income starts to increase between 30 and 50 and the more they get towards retirement, the income starts to drop towards 60 as seen in the Figure 12 below.

```
sns.jointplot(x='Age', y='Area Income', data=ad_data)
plt.show()
```

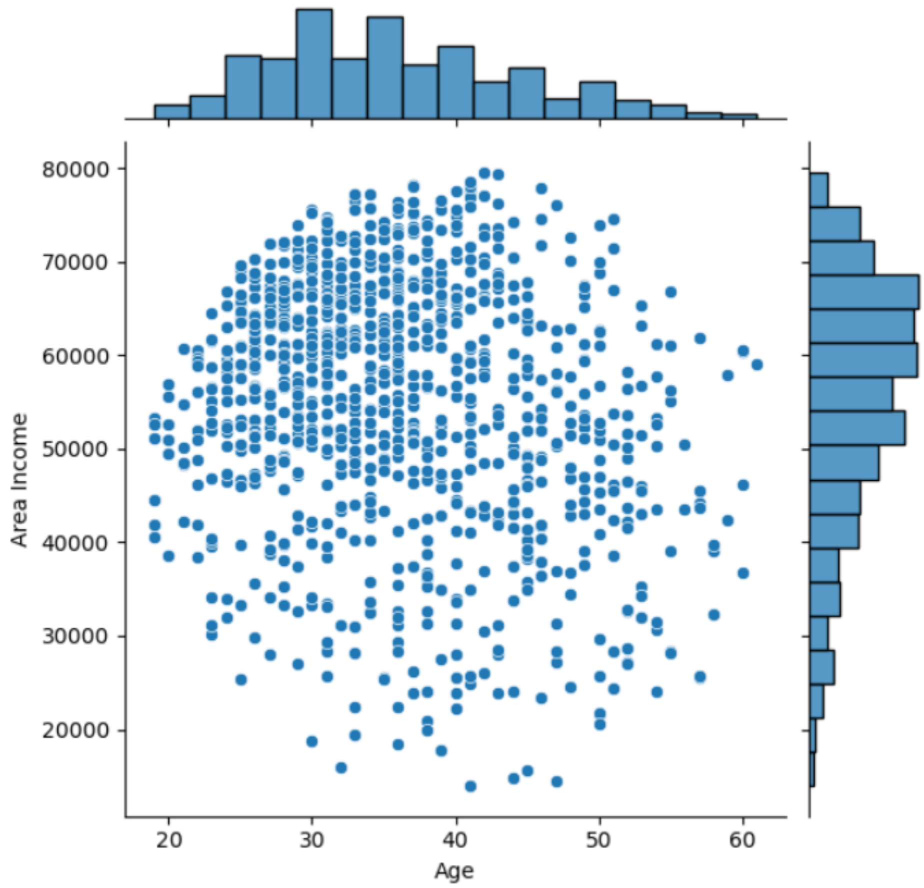

Figure 12

- Create a joinplot showing the kde distribution of Daily Time Spent on Site as y-Axis vs Age as x-axis.
- People from the age group 25 and 40 spend more time on the internet as compared to those in 50s and 60s as seen in the Figure 13 below

```
#Create a joinplot showing the kde distributionof Daily Time Spent on Site vs Age
sns.jointplot(x='Age', y='Daily Time Spent on Site', data=ad_data, kind='kde', color='red')
plt.show()
```

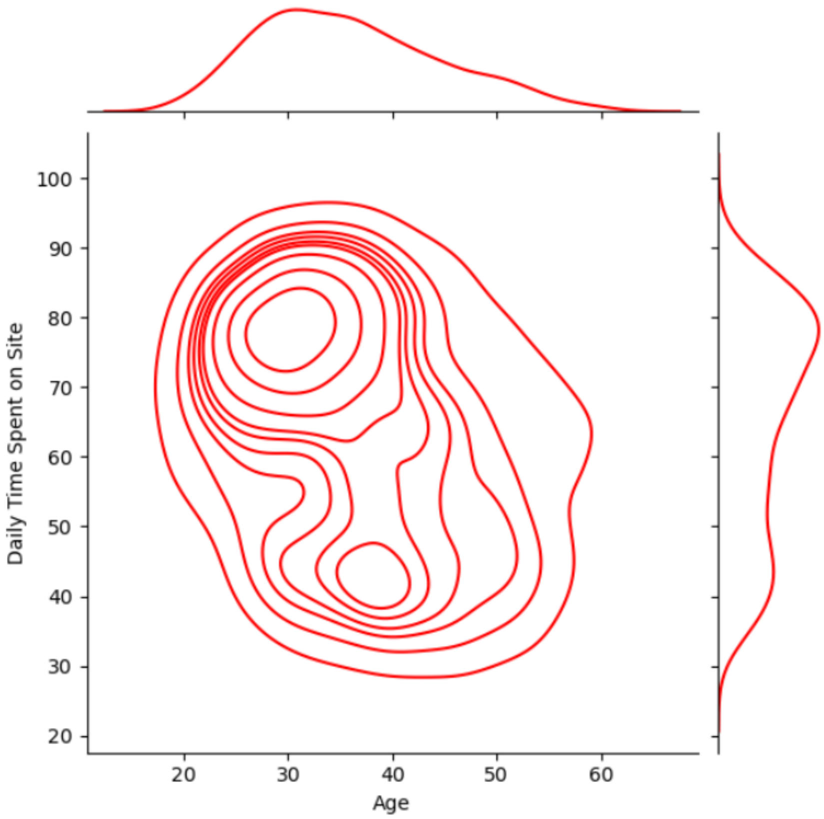

Figure 13

- Created a Jointplot of Daily Time Spent on Site vs Daily Internet Usage
- I noticed 2 clusters, one as you go higher in daily internet usage which is almost a circular pattern as seen in Figure 14 below

```
sns.jointplot(x='Daily Time Spent on Site', y='Daily Internet Usage', data=ad_data, color='green')
```

```
Out[42]: <seaborn.axisgrid.JointGrid at 0x1d4a7cd0700>
```

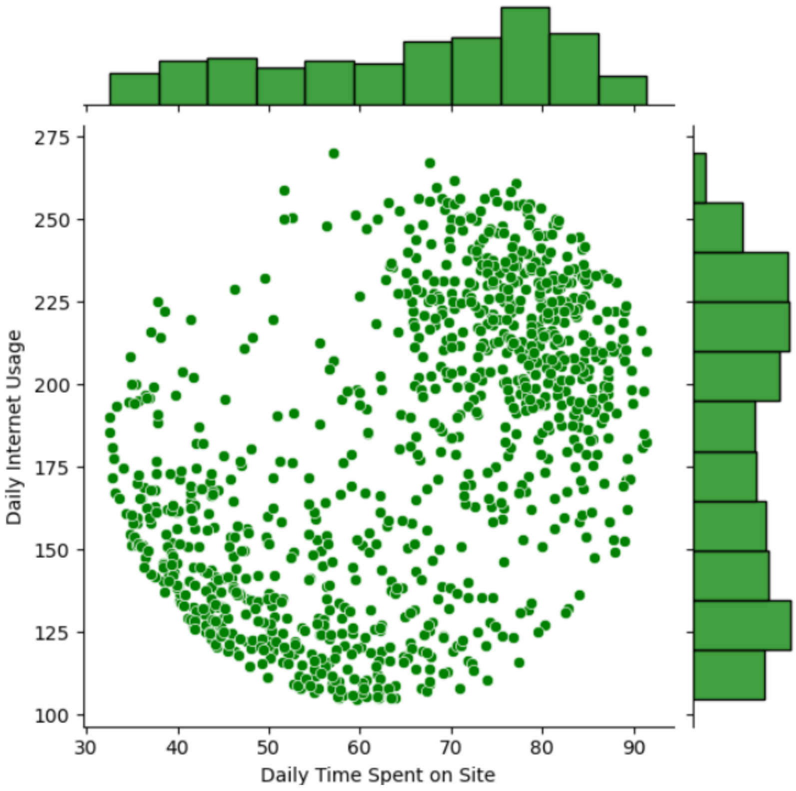

Figure 14

- Finally, I created a pairplot with the hue defined by the ‘Clicked on Ad column feature
- This finds a relationship between all the numerical columns and ‘Clicked on Ad’ as seen in the Figure 15 and Figure 16 below

```
In [43]:  #Create a  pairplot with the hue defined by the ‘Clicked on Ad column feature
          sns.pairplot(ad_data,hue='Clicked on Ad')

Out[43]: <seaborn.axisgrid.PairGrid at 0x1d4a7fc9520>
```

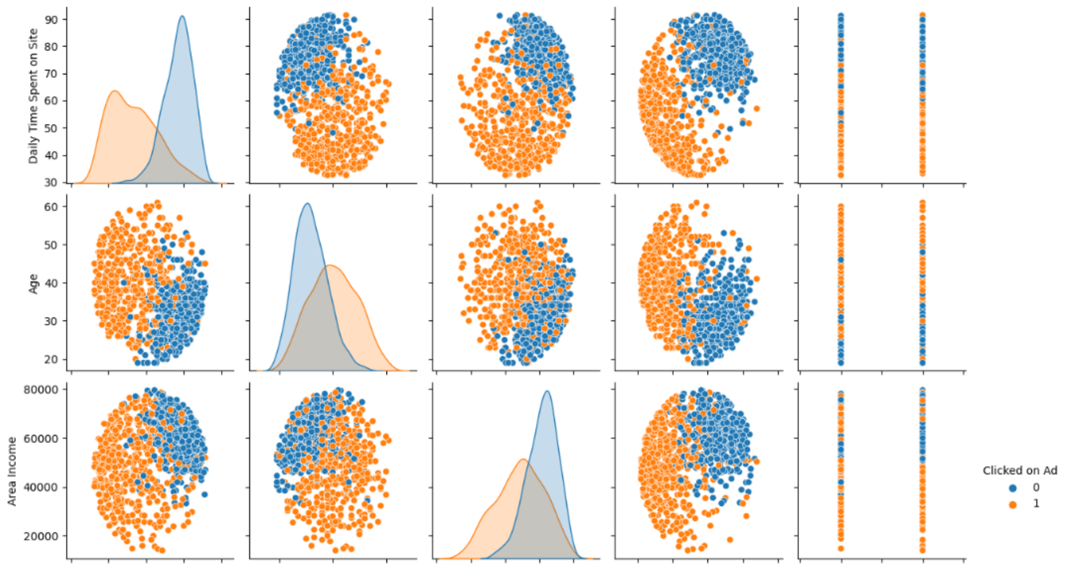

Figure 15

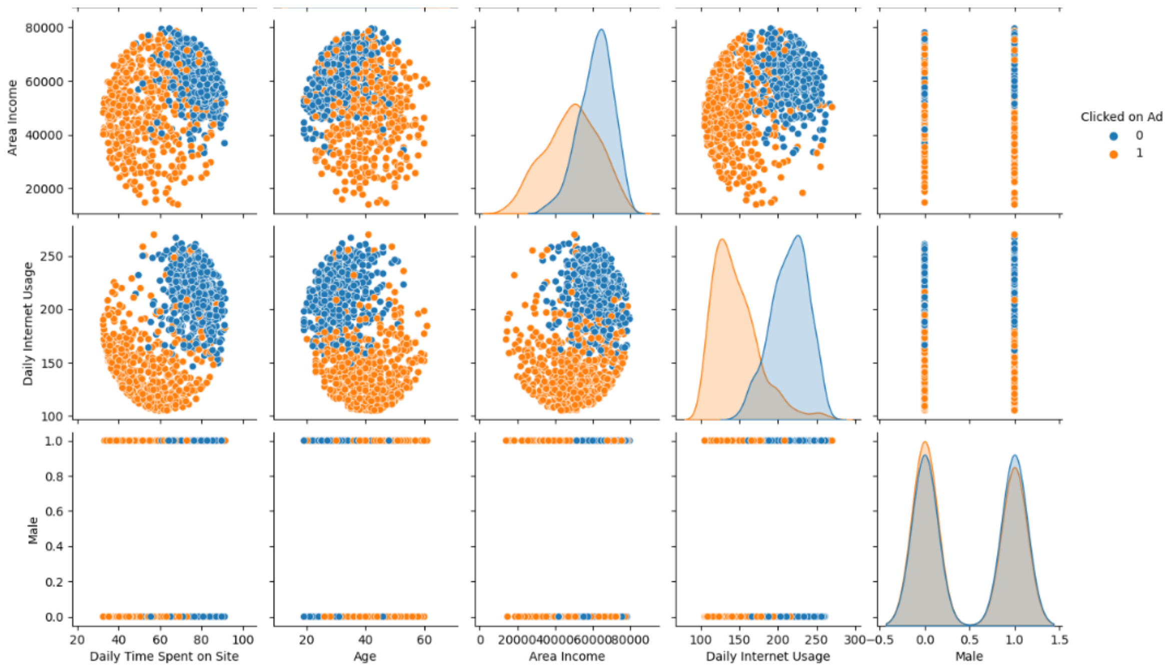

Figure 16

Step 4: Feature Selection

- I used the Feature importance method to determine features most relevant in the prediction of Ad Clicking.
- I imported RandomForestClassifier from sklearn.esemble
- Exclude non-numeric and non-binary columns ['Clicked on Ad', 'Ad Topic Line', 'City', 'Country', 'Timestamp']
- Initialized the model, fit(X,y)
- **Justification of the choice:** This method uses a random forest classifier to determine feature importances and provides robust results and is less prone to overfitting.
- The feature_importances DataFrame displays the importance scores for each feature in descending order with Daily Internet Usage being most important and Male being least important.
- The higher the importance score, the more relevant the feature is for predicting the target variable.
- Feature importance is printed as seen in Figure 17 below

```
In [44]:  from sklearn.ensemble import RandomForestClassifier

In [46]:  # Exclude non-numeric and non-binary columns
          X = ad_data.drop(['Clicked on Ad', 'Ad Topic Line', 'City', 'Country', 'Timestamp'], axis=1)
          y = ad_data['Clicked on Ad']

In [47]:  # Initialize the model
          model = RandomForestClassifier()

In [48]:  # Fit the model
          model.fit(X, y)

Out[48]:  ▾ RandomForestClassifier
          RandomForestClassifier()

In [49]:  # Get feature importances
          feature_importances = pd.DataFrame(model.feature_importances_, index=X.columns, columns=['Importance'])
          feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

In [50]:  # Display the feature importances
          print(feature_importances)

                          Importance
Daily Internet Usage        0.458574
Daily Time Spent on Site    0.340418
Area Income                 0.108678
Age                         0.087100
Male                        0.005230
```

Figure 17

Step5: Model Training- Logistic Regression

- I imported train_test_split from sklearn.model_selection
- Train and fit the logistic regression model on the training set as sen in Figure 18 below

```
#Logistic Regression
#Train test split and train the model
from sklearn.model_selection import train_test_split
```

```
X= ad_data[['Daily Time Spent on Site','Age','Area Income','Daily Internet Usage','Male']]
y= ad_data['Clicked on Ad']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

```
# Train and fit the logistic regression model on the training set
from sklearn.linear_model import LogisticRegression
```

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Predictions: When we feed in X_test in our model, we get the below results in Figure 19

```
predictions = logmodel.predict(X_test)
```

```
predictions
```

```
]:  array([1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1,
           1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
           0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
           0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1,
           0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
           0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0,
           0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
           1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
           0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1,
           1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1,
           0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
           1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1,
           1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
           0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0], dtype=int64)
```

Figure 19

Step 6:Model Evaluation

- I Created a classification report and Confusion matrix
  My model had:
  Average recall of 97%
  F1 Score of 97%
  Accuracy is 97%

Step7: Model Training- Decision Tree

- Import decision tree classifier from sklearn.tree
- Drop non-numeric and non-binary columns and columns that won't contribute much
- Split the data into training and testing sets
- Initialize the Decision Tree classifier
- Fit the model on the training data as seen in Figure 20 below

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Drop non-numeric and non-binary columns and columns that won't contribute much
X = ad_data.drop(['Clicked on Ad', 'Ad Topic Line', 'City', 'Country', 'Timestamp'], axis=1)
y = ad_data['Clicked on Ad']
```

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)
```

```
# Fit the model on the training data
model.fit(X_train, y_train)
```

Figure 20

Predictions: When we feed in X_test in our model, we get the below results in Figure 21

- Make predictions on the testing data

```
# Make predictions on the testing data
y_pred = model.predict(X_test)
```

```
y_pred
```

```
[91]: array([1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
             1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
             0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
             1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0,
             0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
             1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1,
             1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
             1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1,
             1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
             0, 1], dtype=int64)
```

Figure 22

Step8: Model Evaluation- Decision Tree

- I Created a classification report and Confusion matrix
  My model had:
  Average recall of 91%
  F1 Score of 91%
  Accuracy is 91%

Step9: Decision Tree Model

- **Data Bias**: If the training data used to build the Decision Tree is biased, the model will learn and propagate those biases. For instance, if certain groups are overrepresented or underrepresented, the model may favor those groups in its predictions.
- **Feature Bias**: Decision Trees can be sensitive to features with more levels or categories.

Logistic Regression Model

- **Data Bias**: Similar to the Decision Tree, Logistic Regression is influenced by the quality of the training data. If the data is biased, the model will reflect and potentially amplify those biases.
- **Feature Bias**: Logistic Regression assumes a linear relationship between features and the log-odds of the target variable. If this assumption is violated or if features are chosen poorly, the model may introduce biases.

Step10: Comparison between the model Performance

- The table below compares the performance of the above models

Logistic regression model	Decision tree
Average recall of 97%	Average recall of 91%
F1 Score of 97%	F1 Score of 91%
Accuracy is 97%	Accuracy is 91%

Overall Logistic regression performed better than Decision tree model