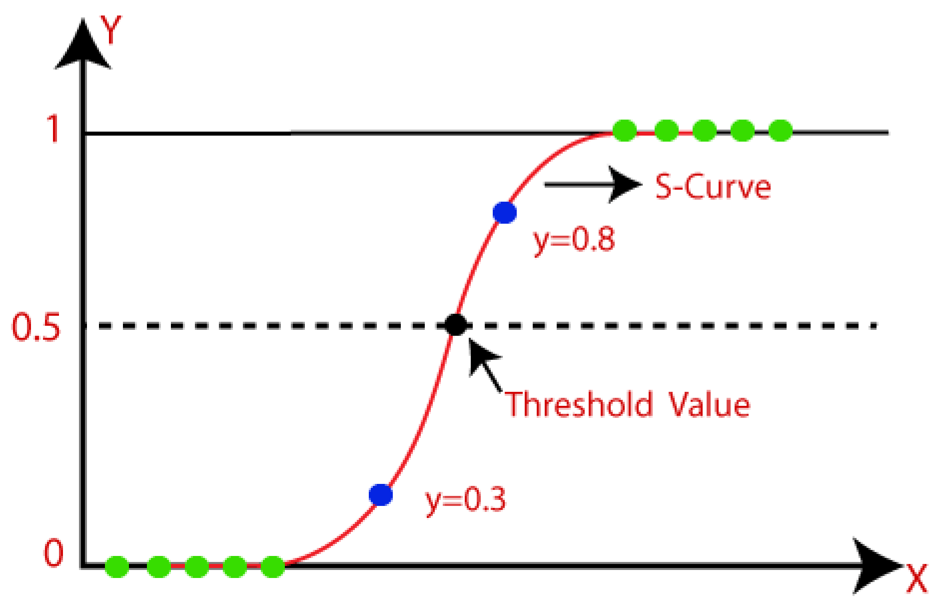
Day 17 - 100 of Data Science

Logistic regression

- Logistic regression is a supervised machine learning algorithm that accomplishes binary classification tasks by predicting the probability of an outcome, event, or observation.
- The model delivers a binary or dichotomous outcome limited to two possible outcomes: yes/no, 0/1, or true/false.

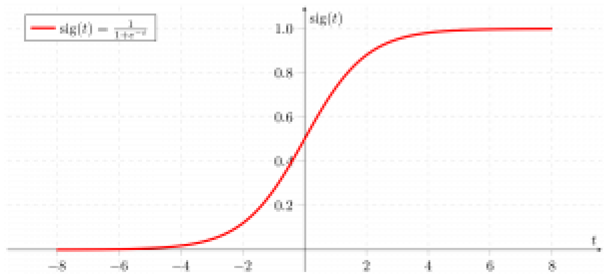


Logistic Function (Sigmoid Function)

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the Sigmoid function or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value

below the threshold values tends to 0.

$$\sigma(z) = \frac{1}{1-e^{-z}}$$



Sigmoid function

Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

- Binomial: In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- Multinomial: In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- Ordinal: In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

implementation

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_style('whitegrid')
plt.style.use("fivethirtyeight")
```

In [4]:

```
# dataset load
data = pd.read_csv("advertising.csv")
data.head()
```

Out[4]:

	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

In [5]:
```
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                       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: 78.3+ KB
```

In [6]:
```
data.describe()
```

Out[6]:

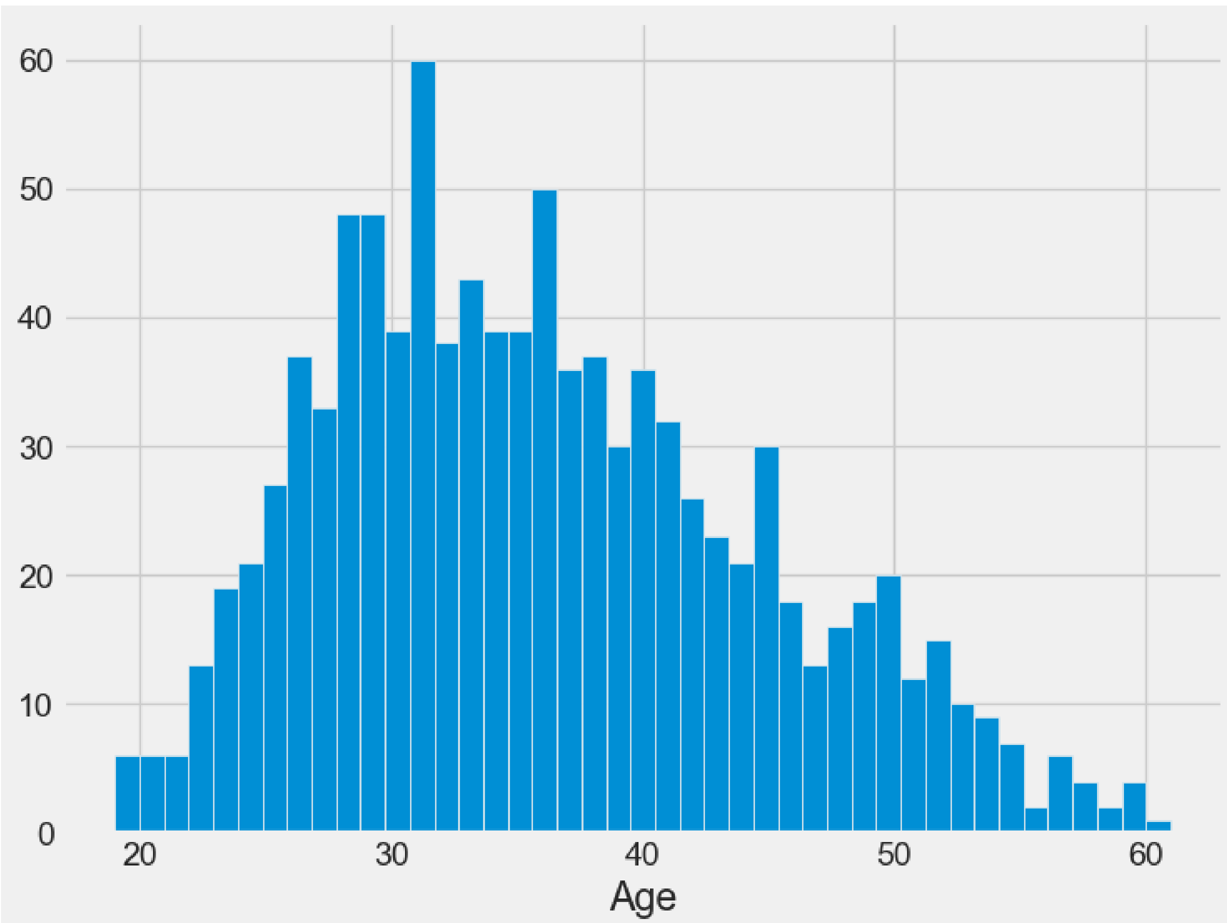
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000	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

In [7]:

```
plt.figure(figsize=(8, 6))
data.Age.hist(bins=data.Age.nunique())
plt.xlabel('Age')
```

Out[7]:

```
Text(0.5, 0, 'Age')
```



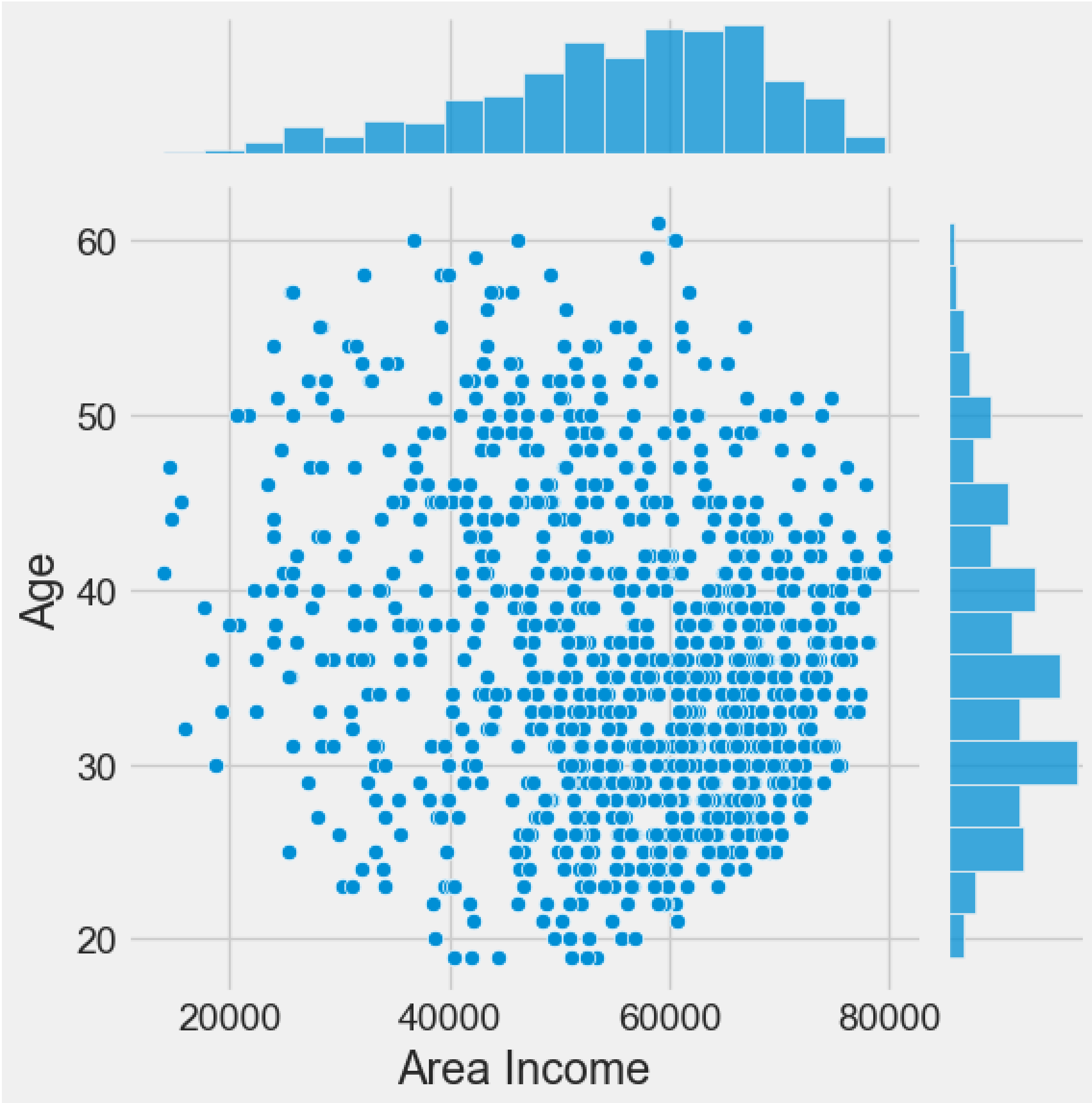
In [8]:

```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Area Income"], y=data.Age)
```

Out[8]:

```
<seaborn.axisgrid.JointGrid at 0x223f314c8d0>
```

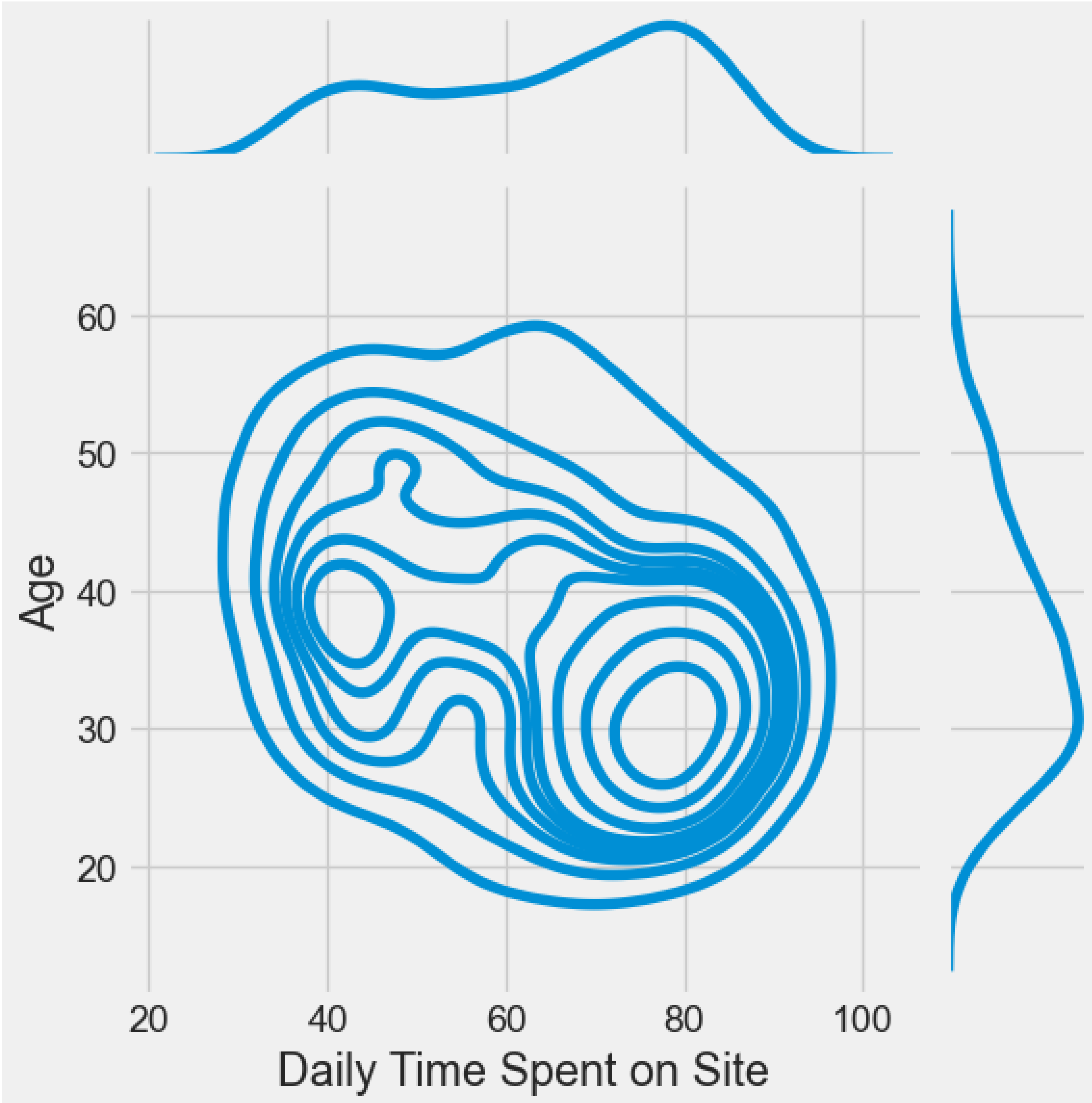
```
<Figure size 800x600 with 0 Axes>
```



In [9]:
```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data.Age, kind='kde')
```

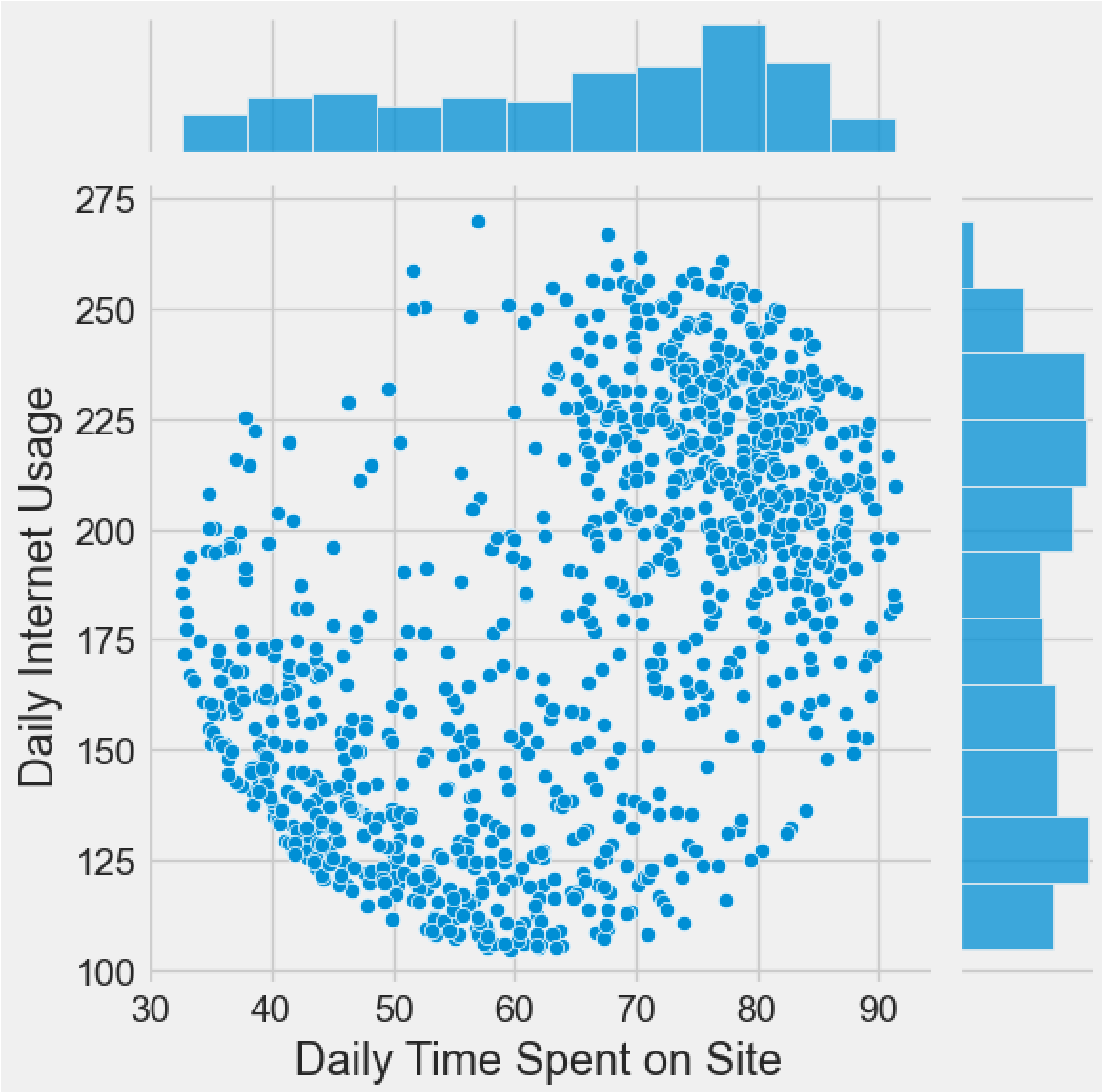
Out[9]:
```
<seaborn.axisgrid.JointGrid at 0x223f339c910>

<Figure size 800x600 with 0 Axes>
```



In [10]:
```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data["Daily Internet Usage"])
```

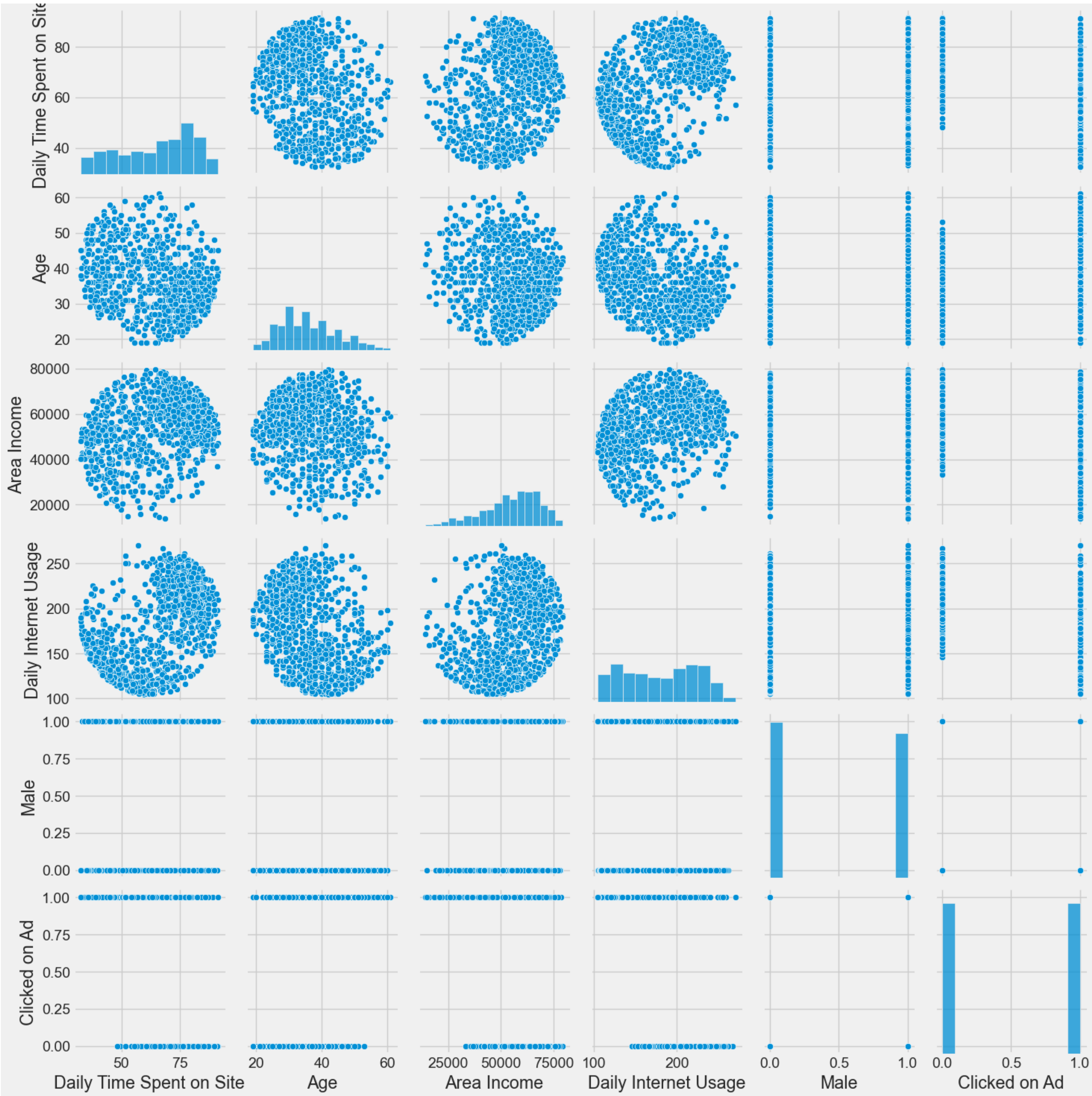
Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x223f3d06e90>
<Figure size 800x600 with 0 Axes>
```



In [11]:
```
sns.pairplot(data)
```

```
C:\Users\TEKS108\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: Th
e figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x223f35dcbd0>
```



In [12]:
```
data['Clicked on Ad'].value_counts()
```

Out[12]:
```
Clicked on Ad
0    500
1    500
Name: count, dtype: int64
```

In [19]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


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=Tru
        print("Train Result:\n= ")
        print(f"Accuracy Score: {accuracy_score(y_train, pred) * 100:.2f}%")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print(f"Confusion Matrix: \n {confusion_matrix(y_train, pred)}\n")

    elif train==False:
        pred = clf.predict(X_test)
```

```
        clf_report = pd.DataFrame(classification_report(y_test, pred, output_dict=True
        print("Test Result:\n=")
        print(f"Accuracy Score: {accuracy_score(y_test, pred) * 100:.2f}%")
        print(f"CLASSIFICATION REPORT:\n{clf_report}")
        print(f"Confusion Matrix: \n {confusion_matrix(y_test, pred)}\n")
```

In [20]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OrdinalEncoder
from sklearn.compose import make_column_transformer
from sklearn.model_selection import train_test_split


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

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

# cat_columns = []
num_columns = ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage


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)
```

In [21]:
```
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)
```

```
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
Confusion Matrix:
 [[349   5]
 [ 13 333]]

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
Confusion Matrix:
 [[143   3]
 [  6 148]]
```

In [ ]: