Advertising Data Project

In this project we will be working with a fake advertising data set, indicating whether or not a particular internet user clicked on an Advertisement. We will try to create a model that will predict whether or not they will click on an ad based off the features of that user.

This data set contains the following features:

- 'Daily Time Spent on Site': consumer time on site in minutes
- 'Age': cutomer age in years
- 'Area Income': Avg. Income of geographical area of consumer
- 'Daily Internet Usage': Avg. minutes a day consumer is on the internet
- 'Ad Topic Line': Headline of the advertisement
- 'City': City of consumer
- 'Male': Whether or not consumer was male
- 'Country': Country of consumer
- 'Timestamp': Time at which consumer clicked on Ad or closed window
- 'Clicked on Ad': 0 or 1 indicated clicking on Ad

Let's Get Started

Import necessary Libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Read "advertising.csv" and set it to dataframe variable

In [2]:
```
df=pd.read_csv("advertising.csv" )
df
```

Out[2]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18
...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21

1000 rows × 10 columns

View the top 5 rows

In [3]:
```
df.head()
```

Out[3]:

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

View info of the data

In [4]:
```
df.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: 78.2+ KB
```

View the basic statistical information about the data

In [5]:
```
df.describe()
```

Out[5]:

	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

Check for null values

In [6]:
```
df.isna().sum()
```

Out[6]:
```
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
```

View all the countries in our data

In [7]:
```
df.columns
df[['Country']]
```

Out[7]:

	Country
0	Tunisia
1	Nauru
2	San Marino
3	Italy
4	Iceland
...	...
995	Lebanon
996	Bosnia and Herzegovina
997	Mongolia
998	Guatemala
999	Brazil

1000 rows × 1 columns

View all the unique values in 'Ad Topic Line'

In [8]:
```
df["Ad Topic Line"].unique()
```

Out[8]:
```
array(['Cloned 5thgeneration orchestration',
       'Monitored national standardization',
       'Organic bottom-line service-desk',
       'Triple-buffered reciprocal time-frame',
       'Robust logistical utilization', 'Sharable client-driven software',
       'Enhanced dedicated support', 'Reactive local challenge',
       'Configurable coherent function',
       'Mandatory homogeneous architecture',
       'Centralized neutral neural-net',
       'Team-oriented grid-enabled Local Area Network',
       'Centralized content-based focus group',
       'Synergistic fresh-thinking array',
       'Grass-roots coherent extranet',
       'Persistent demand-driven interface',
       'Customizable multi-tasking website', 'Intuitive dynamic attitude',
       'Grass-roots solution-oriented conglomeration',
       'Advanced 24/7 productivity',
       'Object-based reciprocal knowledgebase',
       'Streamlined non-volatile analyzer',
```

View all the cities

In [9]:
```
df[['City']]
```

Out[9]:

	City
0	Wrightburgh
1	West Jodi
2	Davidton
3	West Terrifurt
4	South Manuel
...	...
995	Duffystad
996	New Darlene
997	South Jessica
998	West Steven
999	Ronniemouth

1000 rows × 1 columns

Change datatype of 'Timestamp' column to datetime format

In [10]:
```
df["Timestamp"]=pd.to_datetime(df["Timestamp"])
```

In [11]:
```
df.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   datetime64[ns]
 9   Clicked on Ad             1000 non-null   int64
dtypes: datetime64[ns](1), float64(3), int64(3), object(3)
memory usage: 78.2+ KB
```

Create a new column called months by fetching month data from Timestamp

In [12]:
```
df["month"]=pd.DatetimeIndex(df["Timestamp"]).month
df
```

Out[12]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18
...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21

1000 rows × 11 columns

Create a new column called Year by fetching year from Timestamp

In [13]:
```
df["year"]=pd.DatetimeIndex(df["Timestamp"]).year
df
```

Out[13]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18
...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21

1000 rows × 12 columns

Create an new column called Date by fetching day from Timestamp

In [14]:
```
df["date"]=pd.DatetimeIndex(df["Timestamp"]).day
df
```

Out[14]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18
...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21

1000 rows × 13 columns

Create a new column called Hour by fetching hour from Timestamp

In [15]:
```
df["hour"]=pd.DatetimeIndex(df["Timestamp"]).hour
df
```

Out[15]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18
...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21

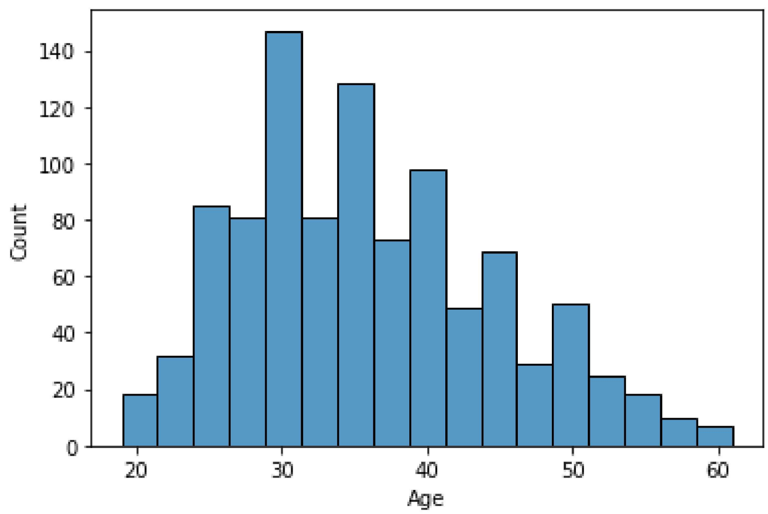
1000 rows × 14 columns

Visualization

Create a histplotof age

In [16]:
```
sns.histplot(x=df['Age'])
```

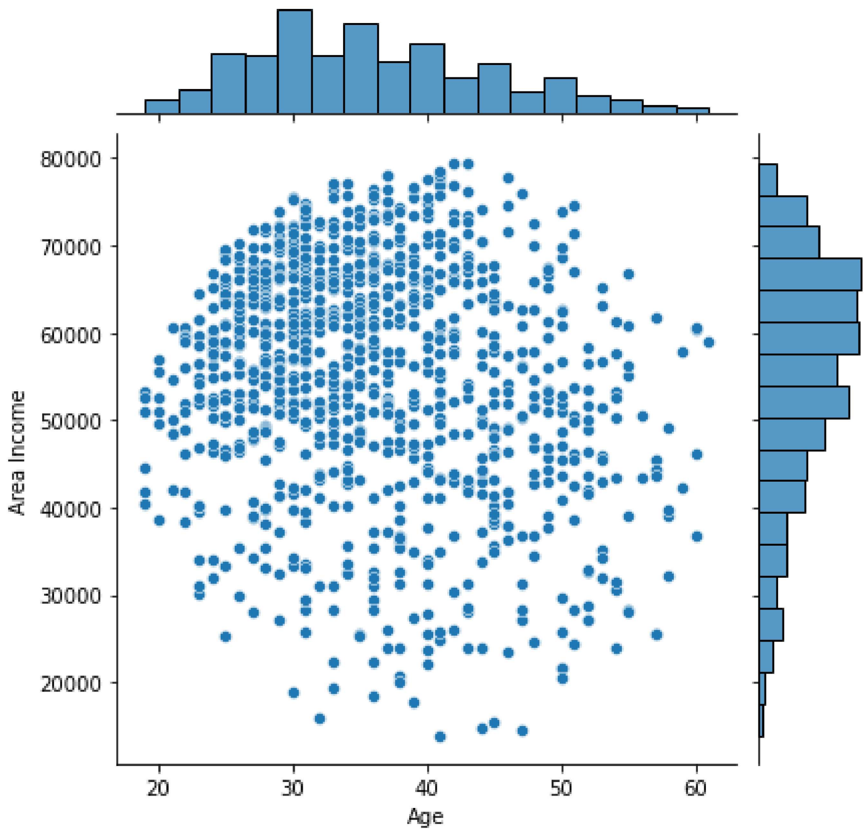
Out[16]: <AxesSubplot:xlabel='Age', ylabel='Count'>



Create a jointplot of Area Income vs Age

In [17]:
```
sns.jointplot(x=df['Age'],y=df['Area Income'])
```

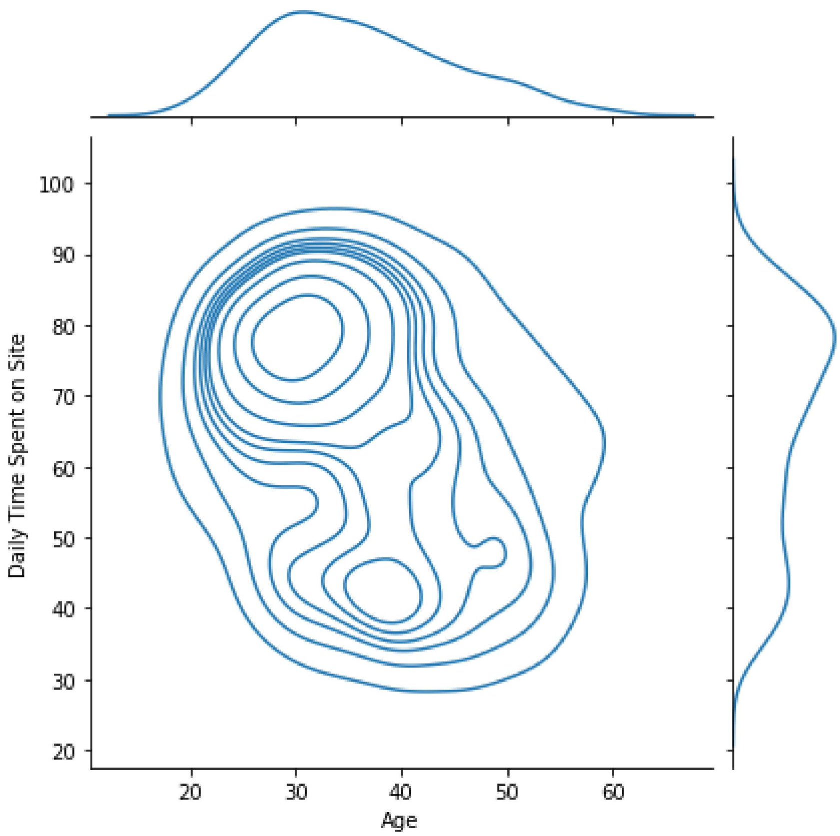
Out[17]: <seaborn.axisgrid.JointGrid at 0x1b2f1371850>



Create a jointplot showing the kde distributions of Daily Time spend on site vs Age

In [18]:
```
sns.jointplot(x=df['Age'],y=df['Daily Time Spent on Site'],kind='kde')
```

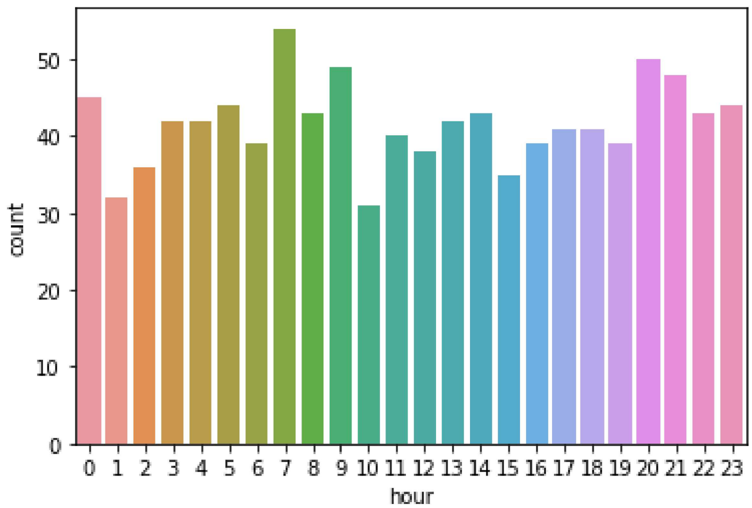
Out[18]:
```
<seaborn.axisgrid.JointGrid at 0x1b2f6706d90>
```



Create a countplot to show Hour (See which hour the users are most active)

In [19]:
```
sns.countplot(x=df["hour"])
```

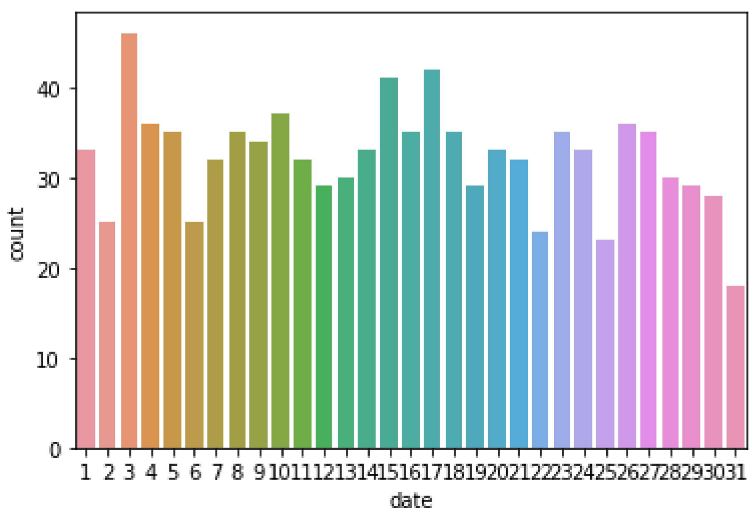
Out[19]: <AxesSubplot:xlabel='hour', ylabel='count'>



Create a countplot to show which day user are the most active

In [20]:
```
sns.countplot(x=df["date"])
```

Out[20]: <AxesSubplot:xlabel='date', ylabel='count'>



Create a heatmap to visualize the correlation between columns

In [21]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(),annot=True,fmt=".3f")
plt.show()
```

Split the data into features and target variables (X and y)

In [22]:
```
df=pd.get_dummies(df,drop_first=True)
```

In [23]:
```
# Choose the columns you see fit
X=df.drop(columns=['Clicked on Ad'])
y=df["Clicked on Ad"]
```

In [24]:
```
X.shape
```

Out[24]:
```
(1000, 2213)
```

In [25]:
```
y.shape
```

Out[25]: (1000,)

Standardize the data

In [26]:
```
from sklearn.preprocessing import StandardScaler
df=pd.get_dummies(df,drop_first=True)
df.head()
```

Out[26]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Timestamp	Clicked on Ad	month	year	date	...	Country_U
0	68.95	35	61833.90	256.09	0	2016-03-27 00:53:11	0	3	2016	27	...	
1	80.23	31	68441.85	193.77	1	2016-04-04 01:39:02	0	4	2016	4	...	
2	69.47	26	59785.94	236.50	0	2016-03-13 20:35:42	0	3	2016	13	...	
3	74.15	29	54806.18	245.89	1	2016-01-10 02:31:19	0	1	2016	10	...	
4	68.37	35	73889.99	225.58	0	2016-06-03 03:36:18	0	6	2016	3	...	

5 rows × 2214 columns

In [ ]:
```
st=StandardScaler()
xcolumns=X.columns
st.fit_transforms(X)
c=pd.DataFrame(df,columns=xcolumns)
```

In [191]:
```
c.head()
```

Out[191]:

Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	month	year	date	hour	Ad Topic Line_Adaptive asynchronous attitude	...	Timestamp_2 07-21 23:1

0 rows × 3211 columns

Split the data into training and testing set

In [143]:
```
from sklearn.model_selection import train_test_split
```

In [144]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=
```

In [145]:
```
X_train.shape
```

Out[145]:
```
(700, 3211)
```

In [146]:
```
y_train.shape
```

Out[146]:
```
(700,)
```

Create a Logistic Regression model and train it

In [147]:
```
from sklearn.linear_model import LogisticRegression
```

In [148]:
```
lr=LogisticRegression()
```

In [149]:
```
#train the model
lr.fit(X_train,y_train)
```

Out[149]:
```
LogisticRegression()
```

Check the accuracy of our model

In [150]:
```
lr.score(X_train,y_train)
```

Out[150]:
```
1.0
```

Make prediction using the X_test

In [151]:
```
y_pred= lr.predict(X_test)
y_pred
```

Out[151]:
```
array([1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1], dtype=int64)
```

Check how accurate the prediction is:

In [152]:
```
from sklearn import metrics
```

In [153]:
```
metrics.accuracy_score(y_test,y_pred)
```

Out[153]:
```
0.9233333333333333
```

In [154]:
```
metrics.mean_squared_error(y_test,y_pred)
```

Out[154]:
```
0.07666666666666666
```

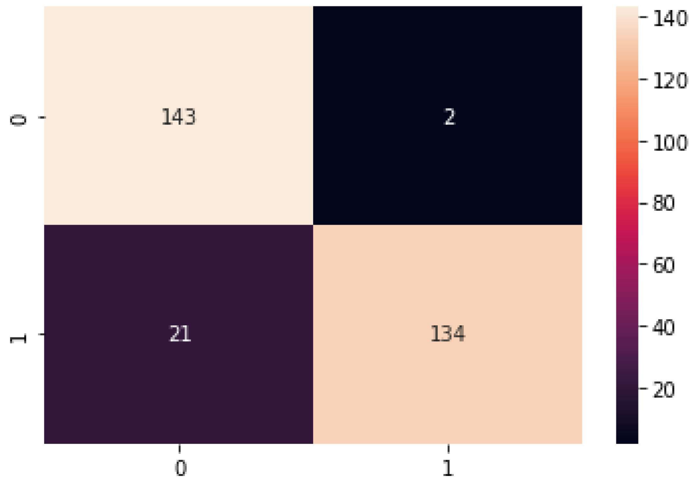
Check the confusion matrix

In [155]:
```
metrics.confusion_matrix(y_test,y_pred)
```

Out[155]:
```
array([[143,   2],
       [ 21, 134]], dtype=int64)
```

Plot the confusion matrix on a heatmap

In [156]:
```
sns.heatmap(metrics.confusion_matrix(y_test,y_pred),annot=True,fmt='d')
plt.show()
```



Create a classification report for the model

In [157]:
```
print(metrics.classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.99      0.93       145
           1       0.99      0.86      0.92       155

    accuracy                           0.92       300
   macro avg       0.93      0.93      0.92       300
weighted avg       0.93      0.92      0.92       300
```

Great Job!