In [92]:

**from** **IPython.display** **import** HTML

In [1]:

**from** **IPython.display** **import** HTML  
  
HTML('''<script>  
code\_show=true;   
function code\_toggle() {  
 if (code\_show){  
 $('div.input').hide();  
 } else {  
 $('div.input').show();  
 }  
 code\_show = !code\_show  
}   
$( document ).ready(code\_toggle);  
</script>  
The raw code for this IPython notebook is by default hidden for easier reading.  
To toggle on/off the raw code, click <a href="javascript:code\_toggle()">here</a>.''')

Out[1]:

The raw code for this IPython notebook is by default hidden for easier reading. To toggle on/off the raw code, click here.

**Simple Logistic Regression in Python**[**¶**](#gjdgxs)

**A project by Vivek Chattopadhyay**[**¶**](#30j0zll)

In [5]:

**import** **numpy** **as** **np**  
**import** **pandas** **as** **pd**  
**import** **matplotlib.pyplot** **as** **plt**  
**import** **seaborn** **as** **sns**  
%**matplotlib** inline

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 on a company website. 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

In [6]:

data = pd.read\_csv('advertising.csv')

**The Data**[**¶**](#1fob9te)

In [7]:

data.head(10)

Out[7]:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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 |
| **5** | 59.99 | 23 | 59761.56 | 226.74 | Sharable client-driven software | Jamieberg | 1 | Norway | 2016-05-19 14:30:17 | 0 |
| **6** | 88.91 | 33 | 53852.85 | 208.36 | Enhanced dedicated support | Brandonstad | 0 | Myanmar | 2016-01-28 20:59:32 | 0 |
| **7** | 66.00 | 48 | 24593.33 | 131.76 | Reactive local challenge | Port Jefferybury | 1 | Australia | 2016-03-07 01:40:15 | 1 |
| **8** | 74.53 | 30 | 68862.00 | 221.51 | Configurable coherent function | West Colin | 1 | Grenada | 2016-04-18 09:33:42 | 0 |
| **9** | 69.88 | 20 | 55642.32 | 183.82 | Mandatory homogeneous architecture | Ramirezton | 1 | Ghana | 2016-07-11 01:42:51 | 0 |

**Descriptive Statistics**[**¶**](#3znysh7)

In [9]:

data.describe()

Out[9]:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **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 |

**Some Technical Information about the data**[**¶**](#2et92p0)

In [11]:

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1000 entries, 0 to 999  
Data columns (total 10 columns):  
Daily Time Spent on Site 1000 non-null float64  
Age 1000 non-null int64  
Area Income 1000 non-null float64  
Daily Internet Usage 1000 non-null float64  
Ad Topic Line 1000 non-null object  
City 1000 non-null object  
Male 1000 non-null int64  
Country 1000 non-null object  
Timestamp 1000 non-null datetime64[ns]  
Clicked on Ad 1000 non-null int64  
dtypes: datetime64[ns](1), float64(3), int64(3), object(3)  
memory usage: 78.2+ KB

From the above information we ca say that there are 3 categorical variables and 6 continuous variables. We can also see that there is a timestamp object which means we can create new variables based on this timestamp variable.

In [10]:

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

In [12]:

data['year'] = data['Timestamp'].apply(**lambda** x: x.year)

In [13]:

data['month'] = data['Timestamp'].apply(**lambda** x: x.month)

In [14]:

data['day'] = data['Timestamp'].apply(**lambda** x: x.day)

In [15]:

data['DOW'] = data['Timestamp'].apply(**lambda** x: x.dayofweek)

In [16]:

data.drop('Timestamp',axis=**True**,inplace=**True**)

In [17]:

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1000 entries, 0 to 999  
Data columns (total 13 columns):  
Daily Time Spent on Site 1000 non-null float64  
Age 1000 non-null int64  
Area Income 1000 non-null float64  
Daily Internet Usage 1000 non-null float64  
Ad Topic Line 1000 non-null object  
City 1000 non-null object  
Male 1000 non-null int64  
Country 1000 non-null object  
Clicked on Ad 1000 non-null int64  
year 1000 non-null int64  
month 1000 non-null int64  
day 1000 non-null int64  
DOW 1000 non-null int64  
dtypes: float64(3), int64(7), object(3)  
memory usage: 101.6+ KB

In [18]:

categoricals = data[['Ad Topic Line','City','Country','year','month','day','DOW']]

In [19]:

categoricals.head()

Out[19]:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Ad Topic Line** | **City** | **Country** | **year** | **month** | **day** | **DOW** |
| **0** | Cloned 5thgeneration orchestration | Wrightburgh | Tunisia | 2016 | 3 | 27 | 6 |
| **1** | Monitored national standardization | West Jodi | Nauru | 2016 | 4 | 4 | 0 |
| **2** | Organic bottom-line service-desk | Davidton | San Marino | 2016 | 3 | 13 | 6 |
| **3** | Triple-buffered reciprocal time-frame | West Terrifurt | Italy | 2016 | 1 | 10 | 6 |
| **4** | Robust logistical utilization | South Manuel | Iceland | 2016 | 6 | 3 | 4 |

As seen above, 4 new variables are created: Year,Month,Day and DOW.

**Categorical EDA**[**¶**](#tyjcwt)

**Uniques**[**¶**](#3dy6vkm)

In [20]:

**for** i **in** categoricals.columns:  
 print(i)  
 print(len(data[i].unique()))  
 print('----------------------------------------------------------------------------------------------------------------')

Ad Topic Line  
1000  
----------------------------------------------------------------------------------------------------------------  
City  
969  
----------------------------------------------------------------------------------------------------------------  
Country  
237  
----------------------------------------------------------------------------------------------------------------  
year  
1  
----------------------------------------------------------------------------------------------------------------  
month  
7  
----------------------------------------------------------------------------------------------------------------  
day  
31  
----------------------------------------------------------------------------------------------------------------  
DOW  
7  
----------------------------------------------------------------------------------------------------------------

**The above topic shows that Ad Topic Line variable has all unique values with no repeataions. Meanwhile, second to that is City followed by Country. Such variables as of now cannot be utilised for specific analysis.**

In [21]:

print('Percentages of Clicked on Site or not!')  
print('Percentage of Clicks on Ad: ',len(data[data['Clicked on Ad']==1])/len(data)\*100,'%')  
print('Percentage of No Clicks on Ad: ',len(data[data['Clicked on Ad']==0])/len(data)\*100,'%')

Percentages of Clicked on Site or not!  
Percentage of Clicks on Ad: 50.0 %  
Percentage of No Clicks on Ad: 50.0 %

**Above shows the percentage of classes of the dependent variable 'Clicked on Ad'. The results shows that there are no class imbalances**

**Top 10 companies country-wise who clicked on most of the Ads:**[**¶**](#1t3h5sf)

In [22]:

data.groupby('Country')['Clicked on Ad'].count().head(10)

Out[22]:

Country  
Afghanistan 8  
Albania 7  
Algeria 6  
American Samoa 5  
Andorra 2  
Angola 4  
Anguilla 6  
Antarctica (the territory South of 60 deg S) 3  
Antigua and Barbuda 5  
Argentina 2  
Name: Clicked on Ad, dtype: int64

**How many Ads were clicked on monthly basis**[**¶**](#4d34og8)

In [23]:

data.groupby('month')['Clicked on Ad'].count().head(10)

Out[23]:

month  
1 147  
2 160  
3 156  
4 147  
5 147  
6 142  
7 101  
Name: Clicked on Ad, dtype: int64

**How many Ads were clicked on weekly basis**[**¶**](#2s8eyo1)

In [25]:

data.groupby('DOW')['Clicked on Ad'].count().head(10)

Out[25]:

DOW  
0 140  
1 122  
2 156  
3 142  
4 155  
5 126  
6 159  
Name: Clicked on Ad, dtype: int64

**Continuous EDA**[**¶**](#17dp8vu)

In [26]:

continuous = data[['Daily Time Spent on Site','Age','Area Income','Daily Internet Usage','Male','Clicked on Ad','year','month','day','DOW']]

In [27]:

plt.figure(figsize=(12,7))  
sns.heatmap(continuous.corr(),annot=**True**)

Out[27]:

<matplotlib.axes.\_subplots.AxesSubplot at 0xeacc8106d8>

**Clicked on Ad:**

1) Has a POSITIVE RELATION with Age.

2) Has a NEGATIVE RELATION with Daily Time Spent on Site, Area Income and Daily Internet Usage.

**Age:**

1) Has a POSITIVE RELATION with Clicked on Ad.

2) Has a NEGATIVE RELATION with Daily Time Spent on Site, Area Income and Daily Internet Usage.

**Area Income:**

1) Has a POSITIVE RELATION with Daily Time Spent on Site and Daily Internet Usage.

2) Has a NEGATIVE RELATION with Clicked on Ad and Age.

**Daily Time Spent on Site:**

1) Has a POSITIVE RELATION with Area Income and Daily internet Usage.

2) Has a NEGATIVE RELATION with Age and Clicked on Ad.

**Daily Internet Usage:**

1) Has a POSITIVE RELATION with Daily Time Spent on Site and Area Income.

2) Has a NEGATIVE RELATION with Age and Clicked on Ad.

In [28]:

sns.set\_style('whitegrid')  
data['Age'].hist(bins=30)  
plt.xlabel('Age')

Out[28]:

Text(0.5,0,'Age')

In [30]:

sns.jointplot(x='Age',y='Area Income',data=data)

Out[30]:

<seaborn.axisgrid.JointGrid at 0xeacfd99860>

In [31]:

sns.jointplot(x='Age',y='Daily Time Spent on Site',data=data,color='red',kind='kde')

Out[31]:

<seaborn.axisgrid.JointGrid at 0xeacfe7fa20>

In [33]:

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

Out[33]:

<seaborn.axisgrid.JointGrid at 0xeacfee1080>

In [34]:

sns.pairplot(data,hue='Clicked on Ad',palette='bwr')

Out[34]:

<seaborn.axisgrid.PairGrid at 0xead001e908>

**All of the visualizations above shows that there exists a non-linear relationship between variables and hence paves way for classification modelling**[**¶**](#3rdcrjn)

In [80]:

**from** **sklearn.model\_selection** **import** train\_test\_split  
  
X = data[['Daily Time Spent on Site', 'Age', 'Area Income','Daily Internet Usage', 'Male']]  
y = data['Clicked on Ad']  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=42)

**Logistic Regression Model**[**¶**](#26in1rg)

**logistic regression model on the training set.**

In [81]:

**from** **sklearn.linear\_model** **import** LogisticRegression  
  
logmodel = LogisticRegression()  
logmodel.fit(X\_train,y\_train)

Out[81]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,  
 intercept\_scaling=1, max\_iter=100, multi\_class='ovr', n\_jobs=1,  
 penalty='l2', random\_state=None, solver='liblinear', tol=0.0001,  
 verbose=0, warm\_start=False)

**Predictions and Evaluations**[**¶**](#lnxbz9)

In [82]:

predictions = logmodel.predict(X\_test)

**classification report**

In [83]:

**from** **sklearn.metrics** **import** classification\_report  
  
print(classification\_report(y\_test,predictions))

precision recall f1-score support  
  
 0 0.87 0.96 0.91 162  
 1 0.96 0.86 0.91 168  
  
avg / total 0.91 0.91 0.91 330

**From the above Classification report, we see that our model has an accuracy of 91% which means our model has performed well in predicting the dependent variable 'Clicked on Ad'.¶**

**Note- No Model Tuning is shown here because this is a showcase of simple Logistic Regression and as this is the SKLEARN's Model, Parameter Regularization(L2 - Regularization by Default) is an automatic procedure for this model and thus can be said the the model is auto-tuned.**[**¶**](#35nkun2)

**THANK YOU!!**[**¶**](#1ksv4uv)