Logistic Regression Project (Predict Ad click)

Logisitc Regression is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability that an instance is greater than 50%, then the model predicts that the instance belongs to that class 1, or else it predicts that it does not. This makes it a binary classifier. In this notebook we will look at the theory behind `Logistic Regression` and use it to 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

```
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")
```

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

```
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: 78.2+ KB
```

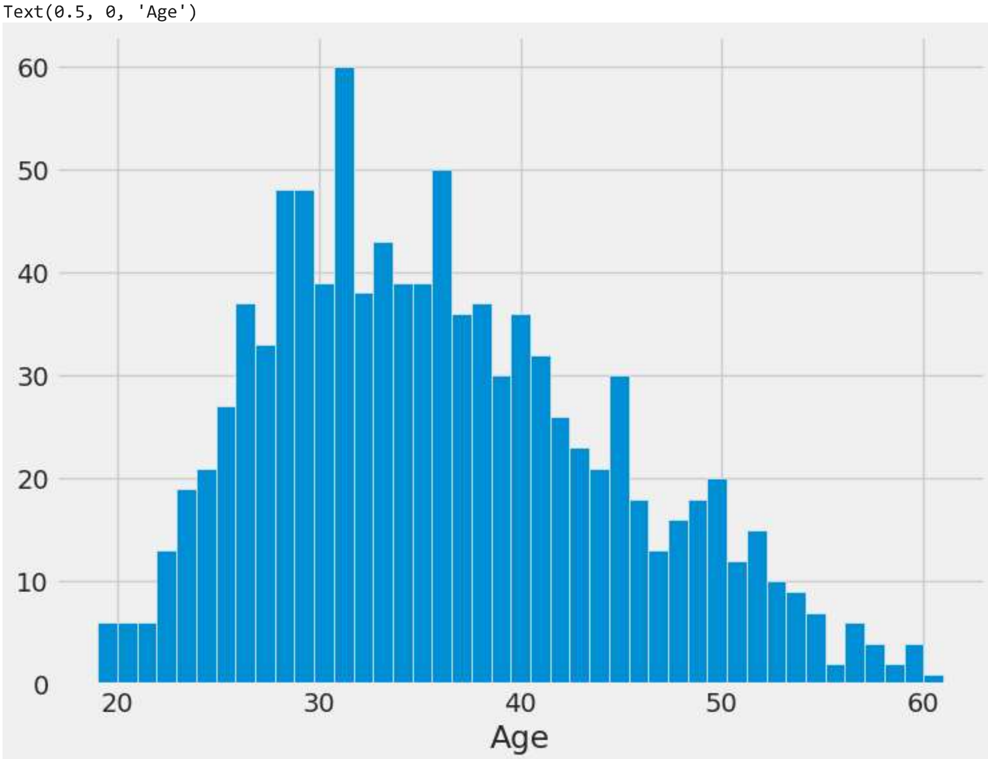
```
data.describe()
```

	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

1. Exploratory Data Analysis

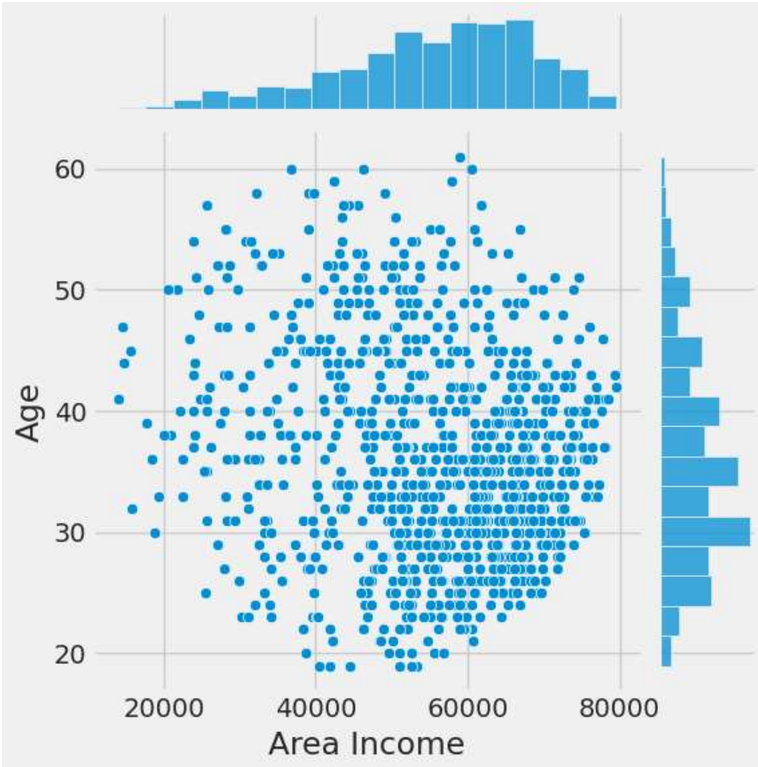
```
plt.figure(figsize=(8, 6))
data.Age.hist(bins=data.Age.nunique())
plt.xlabel('Age')
```

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



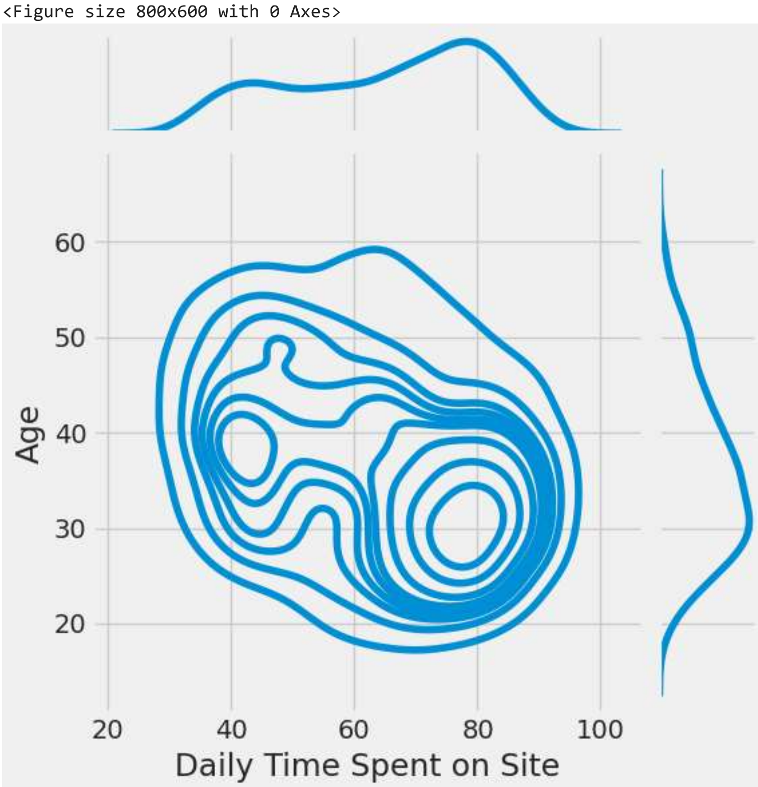

```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Area Income"], y=data.Age)
```

```
<seaborn.axisgrid.JointGrid at 0x7a0d53c5d7e0>
<Figure size 800x600 with 0 Axes>
```



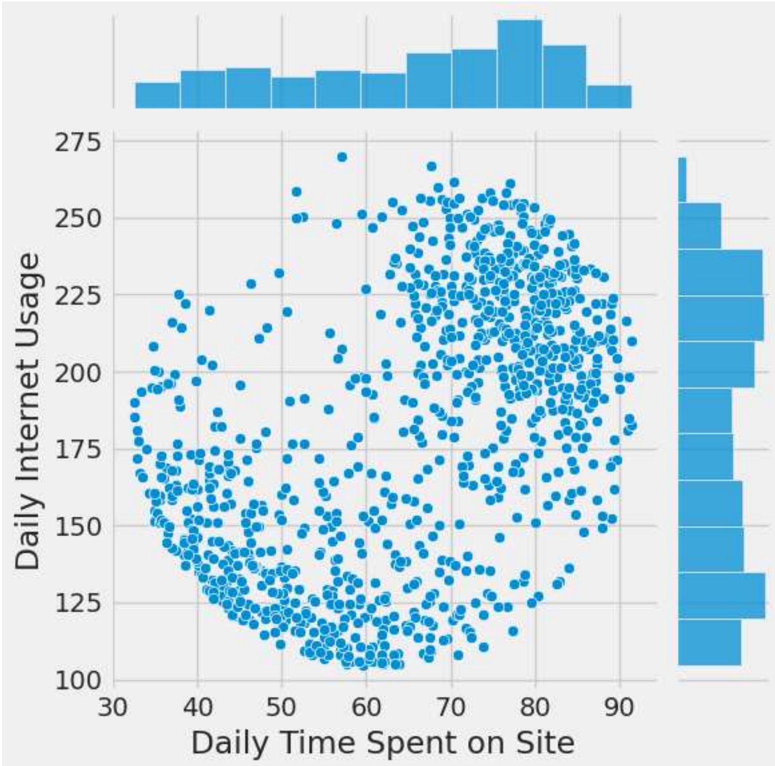

```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data.Age, kind='kde')
```

```
<seaborn.axisgrid.JointGrid at 0x7a0d98b47520>
<Figure size 800x600 with 0 Axes>
```



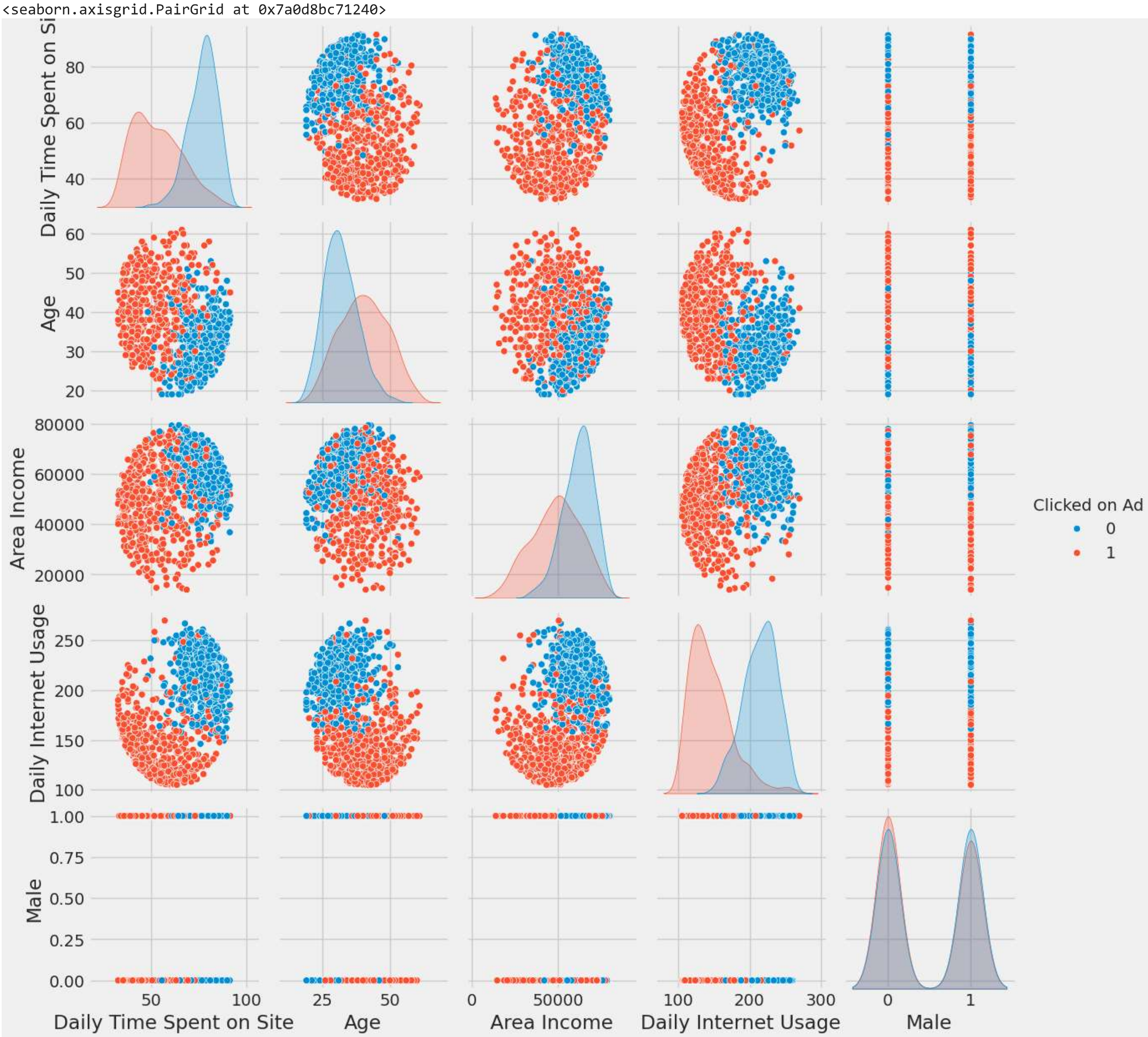

```
plt.figure(figsize=(8, 6))
sns.jointplot(x=data["Daily Time Spent on Site"], y=data["Daily Internet Usage"])
```

```
<seaborn.axisgrid.JointGrid at 0x7a0d50391450>
<Figure size 800x600 with 0 Axes>
```




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

```
<seaborn.axisgrid.PairGrid at 0x7a0d8bc71240>
```

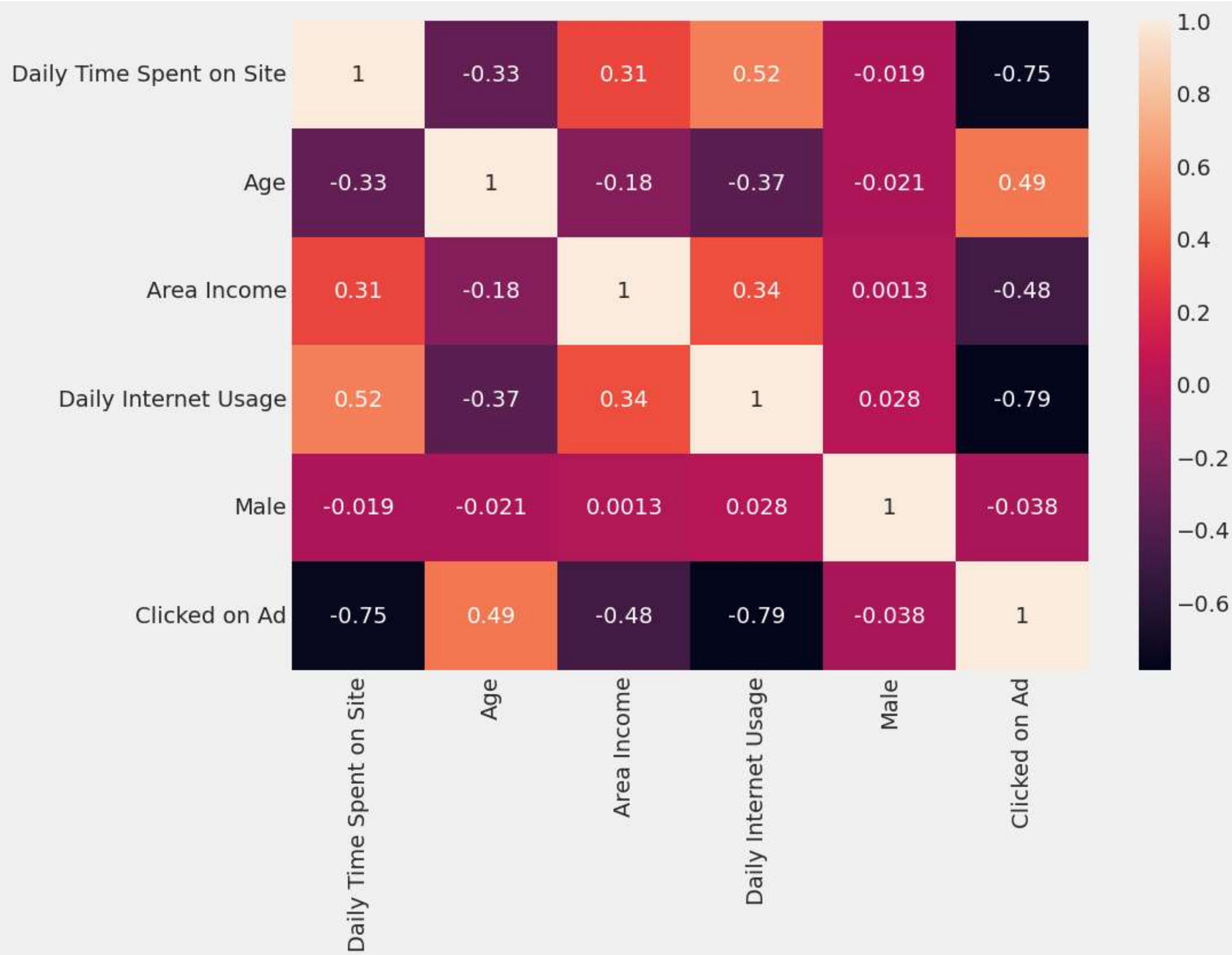



```
data['Clicked on Ad'].value_counts()
```

```
0    500
1    500
Name: Clicked on Ad, dtype: int64
```

```
plt.figure(figsize=(10, 7))
sns.heatmap(data.corr(), annot=True)
```

```
<ipython-input-10-f704f9deedb3>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versio
  sns.heatmap(data.corr(), annot=True)
<Axes: >
```



2. Theory Behind Logistic Regression

Logistic regression is the go-to linear classification algorithm for two-class problems. It is easy to implement, easy to understand and gets great results on a wide variety of problems, even when the expectations the method has for your data are violated.

Description

Logistic Regression

Logistic regression is named for the function used at the core of the method, the [logistic function](#).

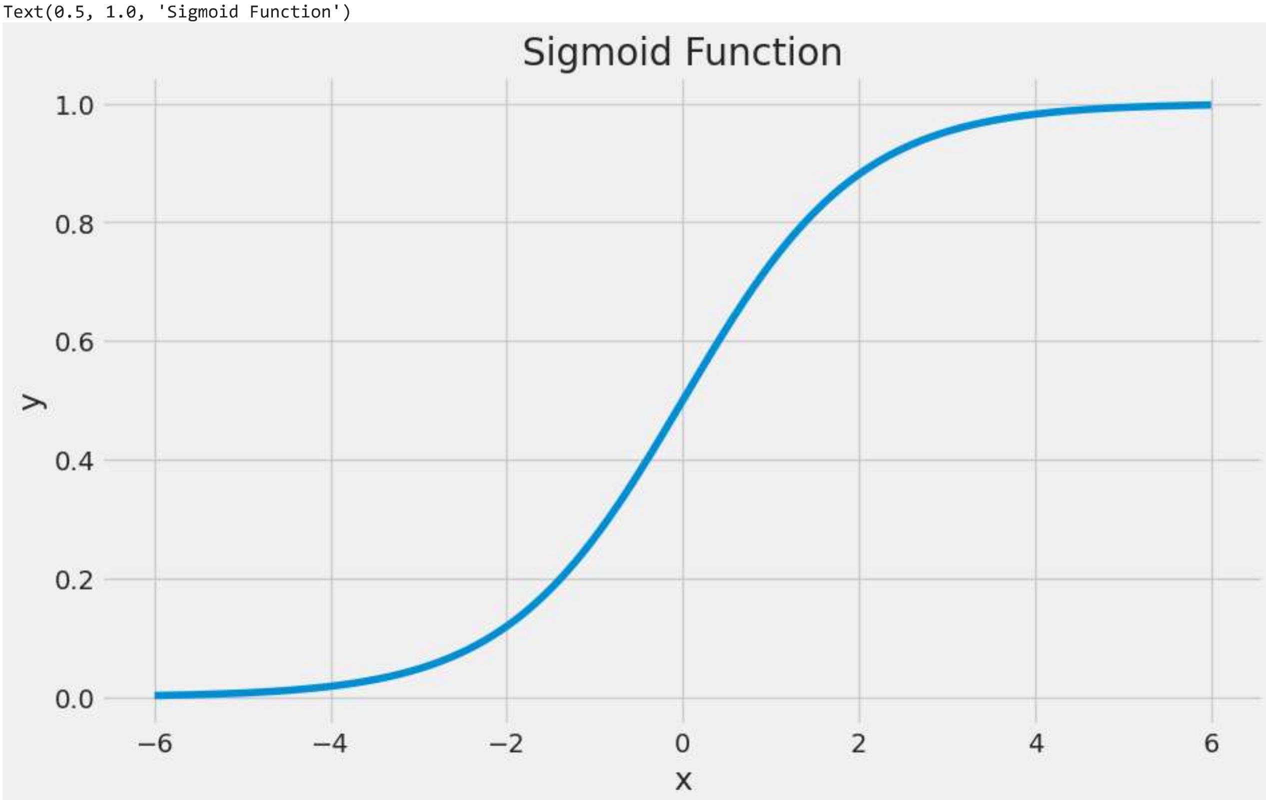
The logistic function, also called the `Sigmoid function` was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

$$\frac{1}{1+e^{-x}}$$

e is the base of the natural logarithms and x is value that you want to transform via the logistic function.

```
x = np.linspace(-6, 6, num=1000)
plt.figure(figsize=(10, 6))
plt.plot(x, (1 / (1 + np.exp(-x))))
plt.xlabel("x")
plt.ylabel("y")
plt.title("Sigmoid Function")
```

```
Text(0.5, 1.0, 'Sigmoid Function')
```



The logistic regression equation has a very similar representation like linear regression. The difference is that the output value being modelled is binary in nature.

$$\hat{y} = \frac{e^{\beta_0 + \beta_1 x_1}}{1 + \beta_0 + \beta_1 x_1}$$

or

$$\hat{y} = \frac{1.0}{1.0 + e^{-\beta_0 - \beta_1 x_1}}$$

β_0 is the intecept term

β_1 is the coefficient for x_1

$\hat{y}$ is the predicted output with real value between 0 and 1. To convert this to binary output of 0 or 1, this would either need to be rounded to an integer value or a cutoff point be provided to specify the class segregation point.

Learning the Logistic Regression Model

The coefficients (Beta values b) of the logistic regression algorithm must be estimated from your training data. This is done using [maximum-likelihood estimation]().

Maximum-likelihood estimation is a common learning algorithm used by a variety of machine learning algorithms, although it does make assumptions about the distribution of your data (more on this when we talk about preparing your data).

The best coefficients would result in a model that would predict a value very close to 1 (e.g. male) for the default class and a value very close to 0 (e.g. female) for the other class. The intuition for maximum-likelihood for logistic regression is that a search procedure seeks values for the coefficients (Beta values) that minimize the error in the probabilities predicted by the model to those in the data (e.g. probability of 1 if the data is the primary class).

We are not going to go into the math of maximum likelihood. It is enough to say that a minimization algorithm is used to optimize the best values for the coefficients for your training data. This is often implemented in practice using efficient numerical optimization algorithm (like the Quasi-newton method).

When you are learning logistic, you can implement it yourself from scratch using the much simpler gradient descent algorithm.

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

Reasons of using scikit-learn (not pandas) for ML preprocessing:

1. You can cross-validate the entire workflow.
2. You can grid search model & preprocessing hyperparameters.
3. Avoids adding new columns to the source DataFrame.
4. Pandas lacks separate fit/transform steps to prevent data leakage.

```
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=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)

# cat_columns = []
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)
```

3. Prepare Data for Logistic Regression

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

- **Binary Output Variable:** This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.

- **Remove Noise:** Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
- **Gaussian Distribution:** Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
- **Remove Correlated Inputs:** Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
- **Fail to Converge:** It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

4. Implimenting Logistic Regression in Scikit-Learn

```
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]]
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_clf = RandomForestClassifier(n_estimators=1000)
rf_clf.fit(X_train, y_train)

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

```
Train Result:
================================================
Accuracy Score: 100.00%
_______________________________________________
CLASSIFICATION REPORT:
               0      1  accuracy  macro avg  weighted avg
precision    1.0    1.0       1.0        1.0           1.0
recall       1.0    1.0       1.0        1.0           1.0
f1-score     1.0    1.0       1.0        1.0           1.0
support    354.0  346.0       1.0      700.0         700.0
```

```
_______________________________________________
Confusion Matrix:
 [[354   0]
 [  0 346]]

Test Result:
================================================
Accuracy Score: 95.67%
_______________________________________________
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.946309    0.966887  0.956667    0.956598      0.956872
recall       0.965753    0.948052  0.956667    0.956903      0.956667
f1-score     0.955932    0.957377  0.956667    0.956655      0.956674
support    146.000000  154.000000  0.956667  300.000000    300.000000
_______________________________________________
Confusion Matrix:
 [[141   5]
 [  8 146]]
```

5. Performance Measurement

1. Confusion Matrix

- Each row: actual class
- Each column: predicted class

First row: Non-clicked Ads, the negative class:

- 143 were correctly classified as Non-clicked Ads. **True negatives**.
- Remaining 6 were wrongly classified as clicked Ads. **False positive**

Second row: The clicked Ads, the positive class:

- 3 were incorrectly classified as Non-clicked Ads. **False negatives**
- 146 were correctly classified clicked Ads. **True positives**

2. Precision