LOGISTIC / CLASSIFICATION :

PAGE 96

In [1]:
```
# The Classification is depends upon the 'Output Data'.
# The 'Output Data' Comes like --> yes/no, ham/spam, 1/0, True/False.
# These Particular Categories, Where We can Expect the 'Output Data' -
# Means, Based on 'Output Data' We can 'Select' the 'Classification'(or)
# We can opt. for 'Logistic/Classification'
```

In [3]:
```
# NOTE :
```

1. Calling Libraries :

page 99

In [14]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [15]:
```
train = pd.read_csv("DataS/30.titanic-train.csv.csv")
train.head()
```

Out[15]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

In [16]:
```
train.isnull()
```

Out[16]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 12 columns

In [17]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

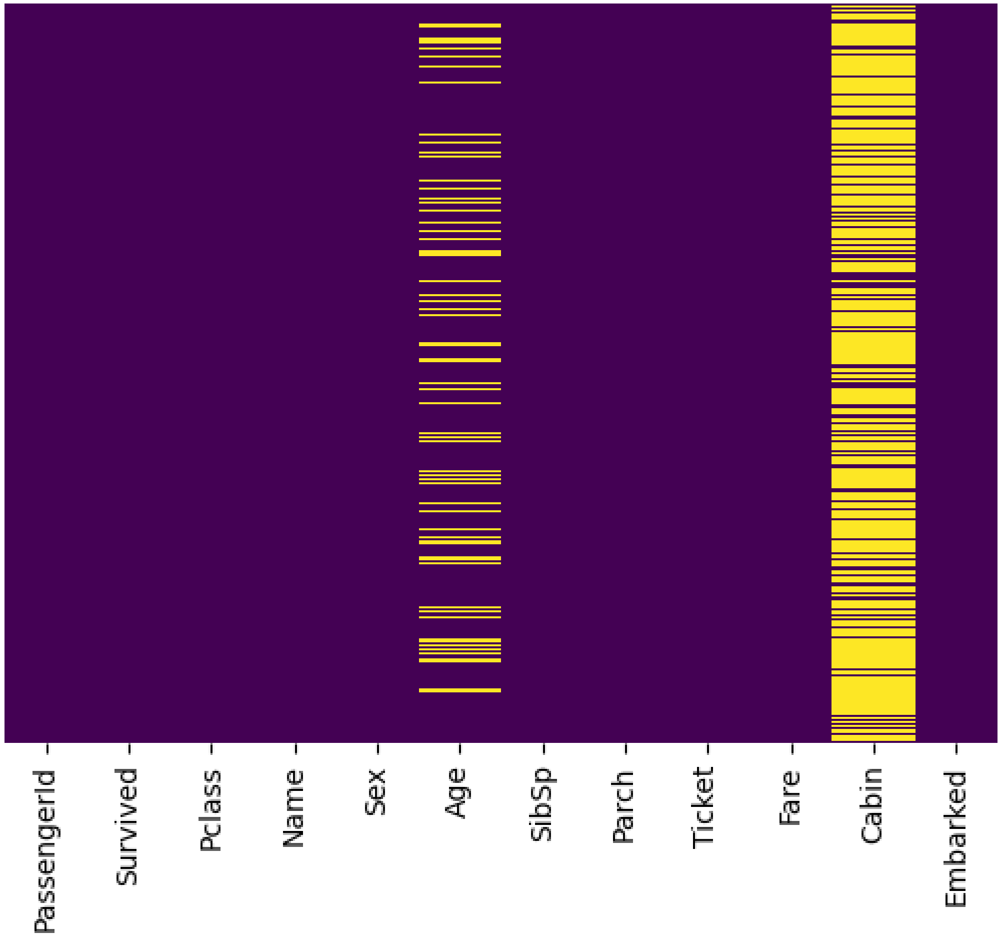
2. INDENTIFYING THE 'NULLS' :

PAGE 101

In [18]:

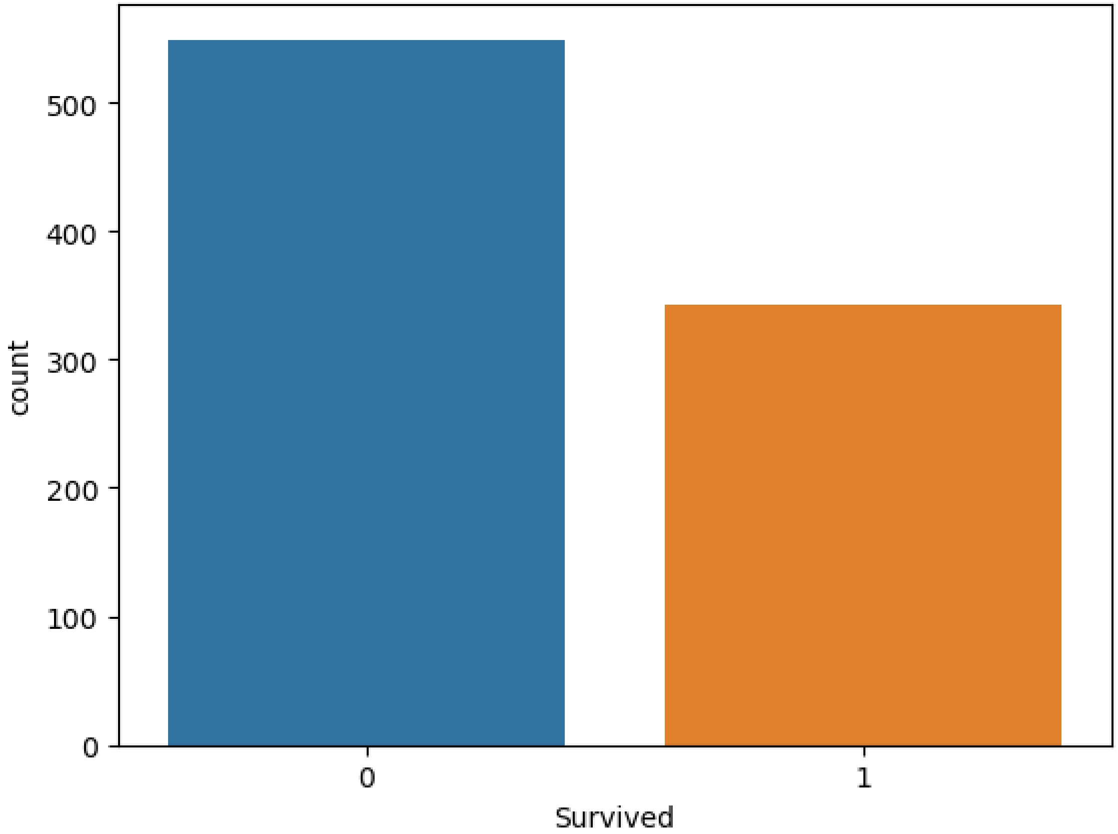
```
import seaborn as sns
sns.heatmap(train.isnull(), yticklabels = False, cbar = False, cmap = 'viridis')
```

Out[18]: <AxesSubplot:>



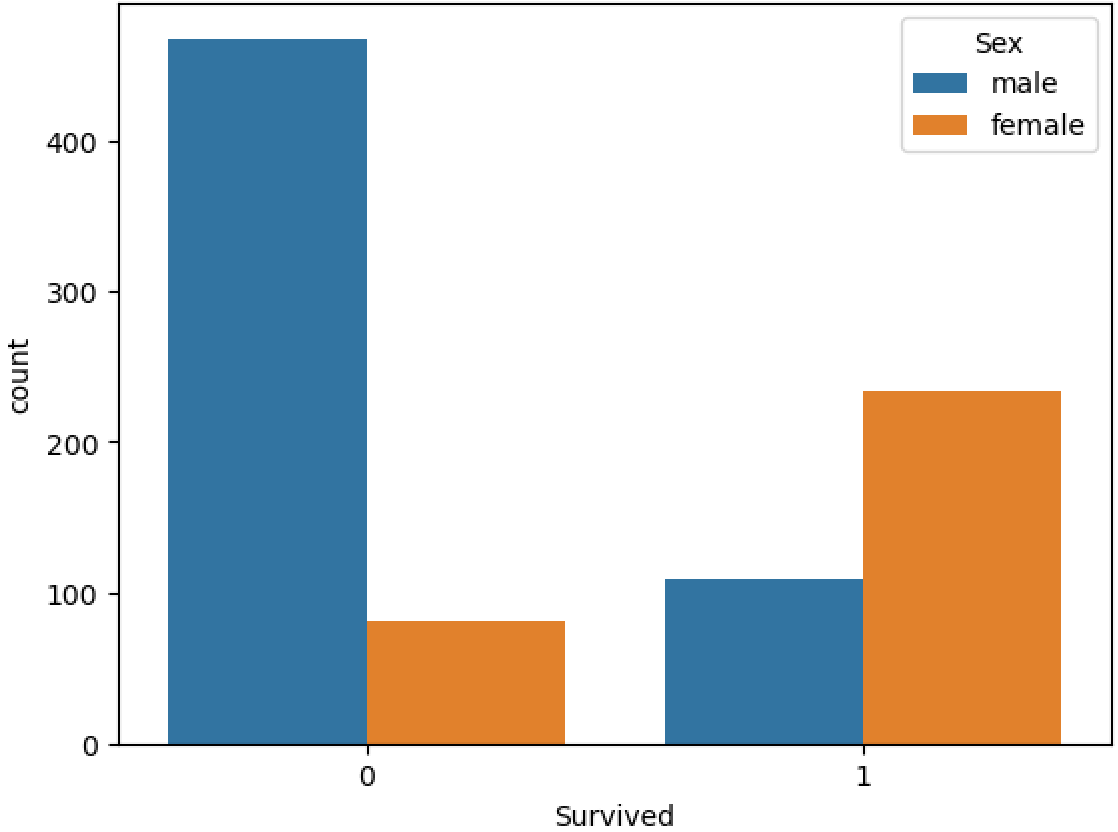
In [19]:
```
# Here, sns.countplot between 'x' = Survived data.

sns.countplot(x = 'Survived', data = train)
```

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



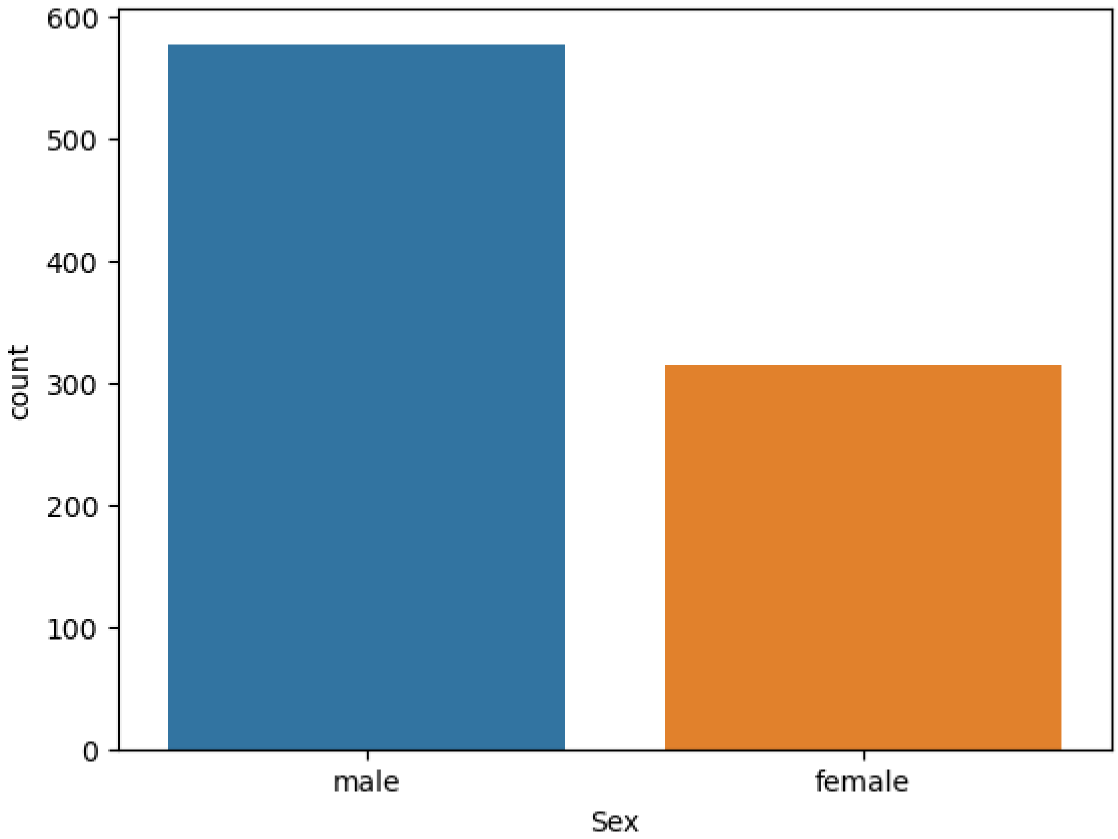
In [20]:
```
sns.countplot(x = 'Survived', hue = 'Sex', data = train)
```

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



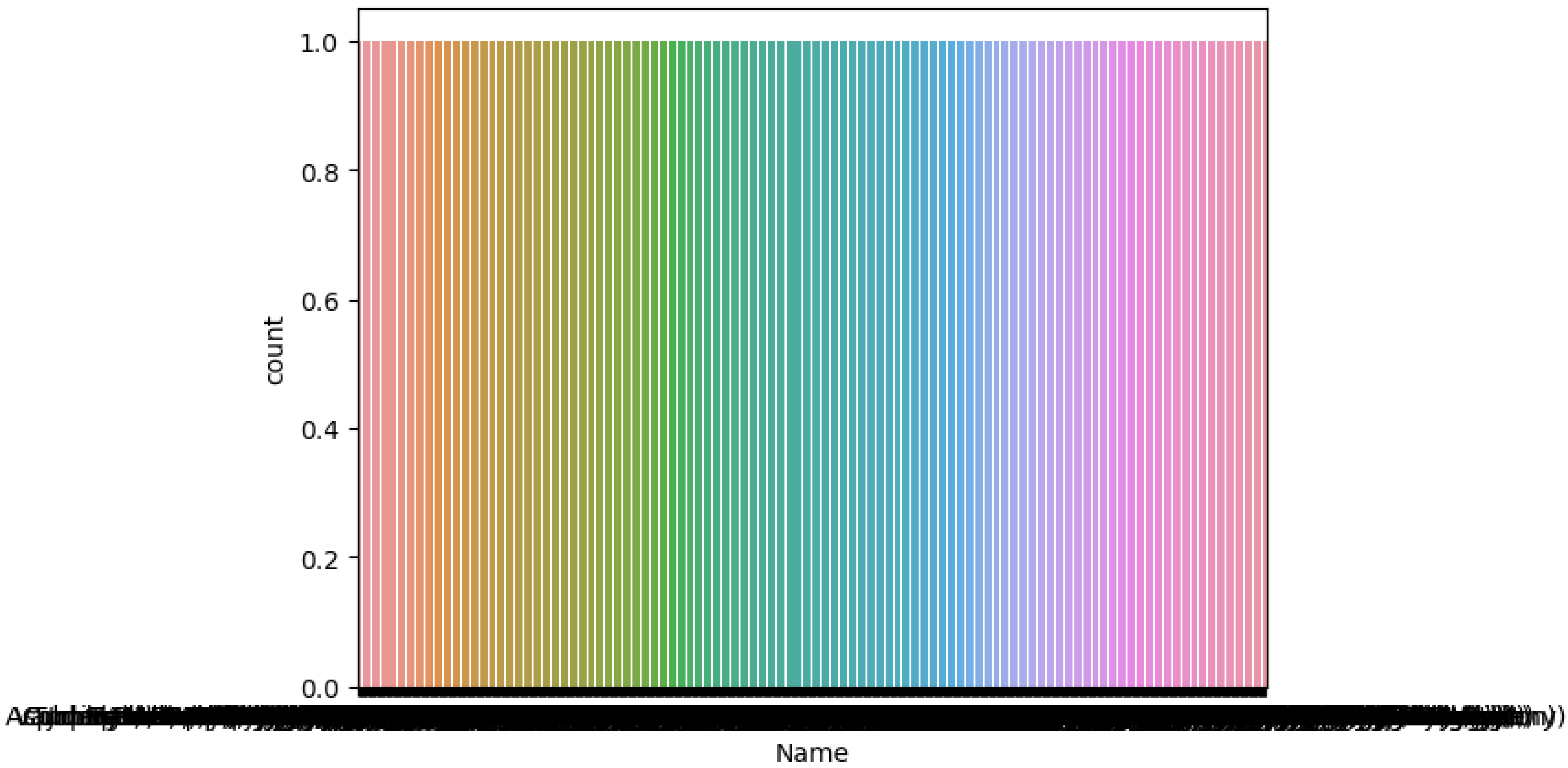
In [21]:
```
sns.countplot(x = 'Sex', data = train)
```

Out[21]: <AxesSubplot:xlabel='Sex', ylabel='count'>



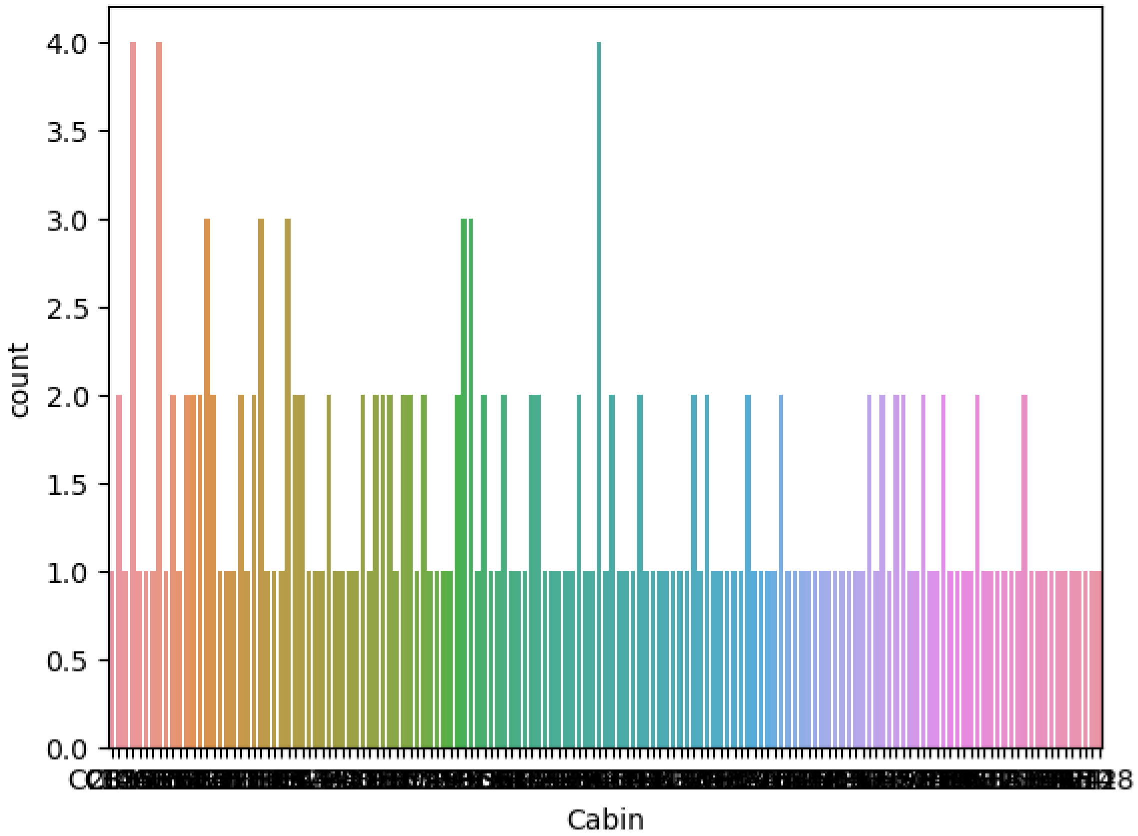
In [22]:
```
sns.countplot(x = 'Name', data = train)
```

Out[22]: <AxesSubplot:xlabel='Name', ylabel='count'>



In [23]:
```
sns.countplot(x = 'Cabin', data = train)
```

Out[23]: <AxesSubplot:xlabel='Cabin', ylabel='count'>

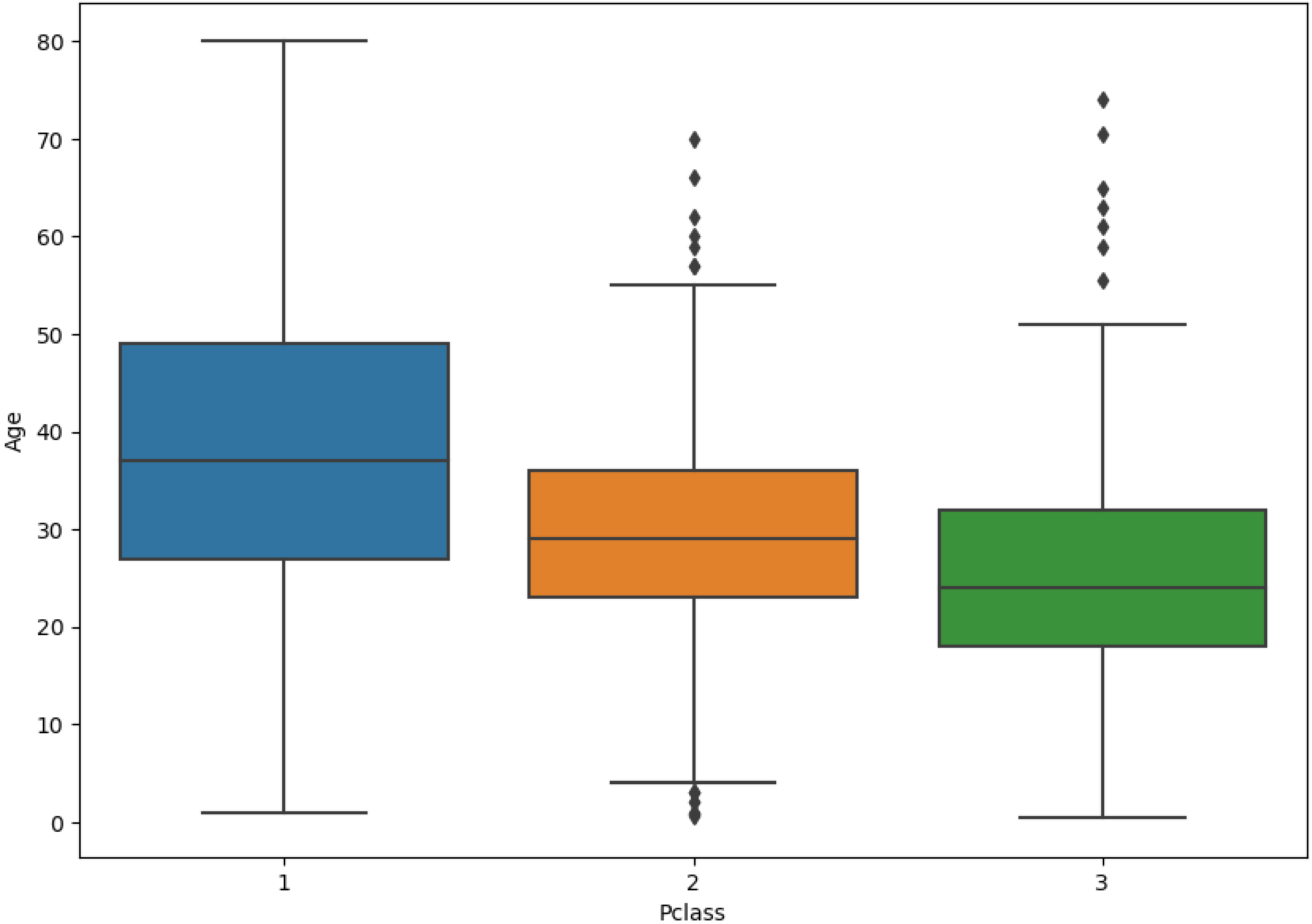


In [24]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

In [30]:
```
plt.figure(figsize = (10,7))
sns.boxplot(x = 'Pclass', y = 'Age', data = train)
```

Out[30]: <AxesSubplot:xlabel='Pclass', ylabel='Age'>



Now we need to build the LOGIC :

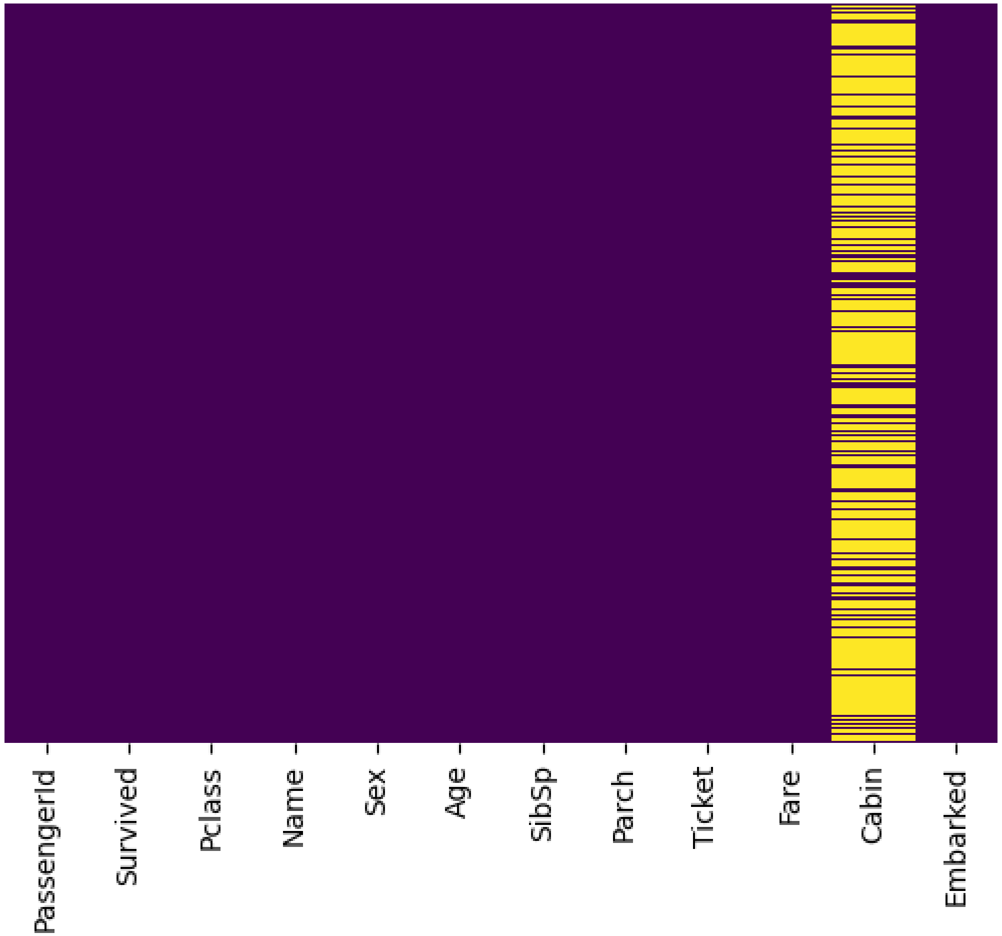
PAGE 106

In [31]:
```
def impute_age(cols):
    Age = cols[0]
    Pclass = cols[1]
    if pd.isnull(Age):
        if Pclass == 1:
            return 37
        elif Pclass == 2:
            return 29
        else:
            return 25
    else:
        return Age
```

In [33]:
```
train ['Age'] = train[['Age', 'Pclass']].apply(impute_age, axis = 1)
```

In [34]:
```
import seaborn as sns
sns.heatmap(train.isnull(), yticklabels = False, cbar = False, cmap = 'viridis')
```

Out[34]: <AxesSubplot:>



In [35]:
```
# Now, train.info(), Now it's fullfill
```

In [36]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

Removing the Column(Cabin) from the Table :

page 108

In [38]:
```
train.drop('Cabin', axis = 1, inplace = True)
```

In [39]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

In [42]:
```
# Here, We have an issue in 'Embarked' Column 889 out of 891 --> means, it has only 889 records.
# Only, '2' records why to messup with two records,
# Now let's delete permanently particular '2' records or nulls by using 'inplace = True'.
# Here, all the records became '889'.
# Because, Embarked only '2' records, When we have to process, just 0.02%.
# We just removed that particular 'Null'
# Now all the data is perfect.

train.dropna(inplace = True)
```

In [43]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Name         889 non-null    object 
 4   Sex          889 non-null    object 
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64  
 7   Parch        889 non-null    int64  
 8   Ticket       889 non-null    object 
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [45]:
```
# Now, Whenever we come to particular "Gender" :

train.head()
```

Out[45]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

In [46]:
```
# Here, Whenever we come to 'Gender', we have two categories -> Male and Female
# Now, i want to change this particular Data to an a '0' and '1'
# Male as '0' and Female as '1'
# Whenever Male is '1' then Female will be '0'

# Now Creating an a '2' Variables as an a Male and Female, By applying 'Dummies Method()'
```

Dummies Method() :

page 111

In [49]:
```
#

pd.get_dummies(train['Sex'])
```

Out[49]:

	female	male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1
...	...	...
886	0	1
887	1	0
888	1	0
889	0	1
890	0	1

889 rows × 2 columns

DROP METHOD( ) : USING 'DUMMIES' :

PAGE 111

In [52]:
```
sex = pd.get_dummies(train['Sex'], drop_first = True)
```

In [54]:
```
# Here, Only Male Data, Wherever we are getting '0'.
# Directly, here We Can Consider as an a 'Female'.

sex.head()
```

Out[54]:

	male
0	1
1	0
2	0
3	0
4	1

Now let's declare '4' records :

page 112

In [55]:
```
train.head(4)
```

Out[55]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S

# #

In [57]:
```
# Here, Whenever we have '2' zeros'0' means 'C' will be '1'

embark = pd.get_dummies(train['Embarked'], drop_first = True)
embark.head()
```

Out[57]:

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

CONCATINATION :

PAGE 113

In [58]:
```
# Now how many Columns we have in train Data lets see :
```

In [59]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  889 non-null    int64  
 1   Survived     889 non-null    int64  
 2   Pclass       889 non-null    int64  
 3   Name         889 non-null    object 
 4   Sex          889 non-null    object 
 5   Age          889 non-null    float64
 6   SibSp        889 non-null    int64  
 7   Parch        889 non-null    int64  
 8   Ticket       889 non-null    object 
 9   Fare         889 non-null    float64
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 83.3+ KB
```

In [61]:
```
train.drop(['Sex', 'Embarked', 'Name', 'Ticket', 'PassengerId'], axis = 1, inplace = True)
```

In [62]:
```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 889 entries, 0 to 890
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  889 non-null    int64  
 1   Pclass    889 non-null    int64  
 2   Age       889 non-null    float64
 3   SibSp     889 non-null    int64  
 4   Parch     889 non-null    int64  
 5   Fare      889 non-null    float64
dtypes: float64(2), int64(4)
memory usage: 48.6 KB
```

Now

page 115

In [63]:
```
X = train.drop('Survived', axis = 1)
y = train['Survived']
```

In [64]:
```
from sklearn.model_selection import train_test_split
```

In [66]:
```
# Here, i increased 'random_state' from 42 to 101:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=101)
```

In [72]:
```
from sklearn.linear_model import LogisticRegression
```

In [73]:
```
logmodel = LogisticRegression()
```

In [74]:
```
logmodel.fit(X_train, y_train)
```

Out[74]:
```
LogisticRegression()
```

In [76]:
```
LogisticRegression()
```

Out[76]:
```
LogisticRegression()
```

In [77]:
```
# Here, we are 'predicting' on 'X_test' data

predictions = logmodel.predict(X_test)
```

CONFUSION MATRIX :

PAGE 116

In [80]:
```
# Now We are applying Confusion Matrix from Sklearn.matrics -> means, we need to call from 'Matrix':
```

In [82]:
```
# These two things we need to get it :

from sklearn.metrics import classification_report, confusion_matrix
```

In [84]:
```
# Here, We print classification_report between y_test, predictions and same as to confusion :
# Here, Accuracy of 72% and 74%.
# if we have any issue, we can Caliculate it.
# We Call this one as -> Manual Caliculation, Manual Testing.
# Again, We Can 'Extend' also.
# We need to do, Something like -> 'Granphical Representation' - 'Actual VS Prediction'
# These 'Classification_report and Confusion_matrix' will give us - 'Manual Working Environment'
# We Can 'test' - What accuracy we are getting.


print(classification_report(y_test, predictions))
print('\n')
print(confusion_matrix(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.72      0.88      0.79       174
           1       0.74      0.51      0.60       120

    accuracy                           0.73       294
   macro avg       0.73      0.69      0.70       294
weighted avg       0.73      0.73      0.72       294



[[153  21]
 [ 59  61]]
```

Now Working on -- 'Advertising' :

page 118

In [85]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [86]:
```
ad_data = pd.read_csv("DataS/31.advertising.csv")
```

In [87]:
```
ad_data
```

Out[87]:

	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
...	...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows × 10 columns

In [88]:
```
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                       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
```

In [89]:
```
ad_data.describe()
```

Out[89]:

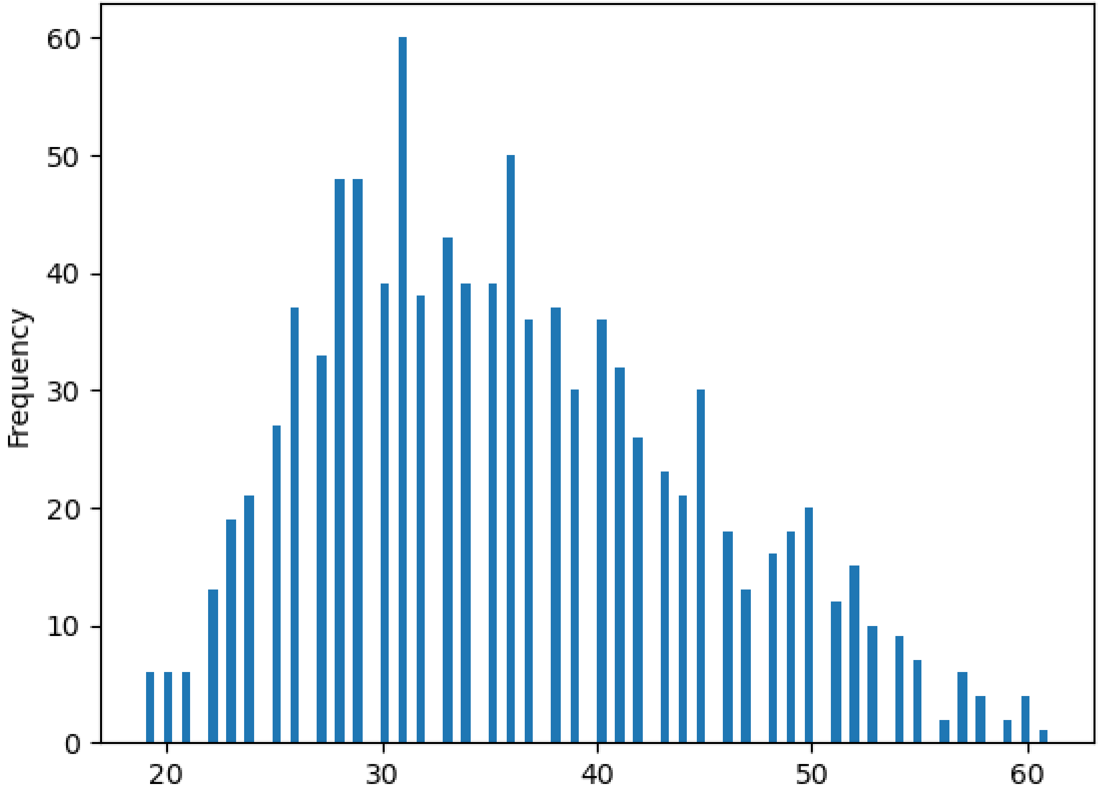
	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male	Clicked on Ad
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	65.000200	36.009000	55000.000080	180.000100	0.481000	0.50000
std	15.853615	8.785562	13414.634022	43.902339	0.499889	0.50025
min	32.600000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

Graphical Representation :

page 119

In [90]:
```
ad_data['Age'].plot.hist(bins = 100)
```

Out[90]:
```
<AxesSubplot:ylabel='Frequency'>
```

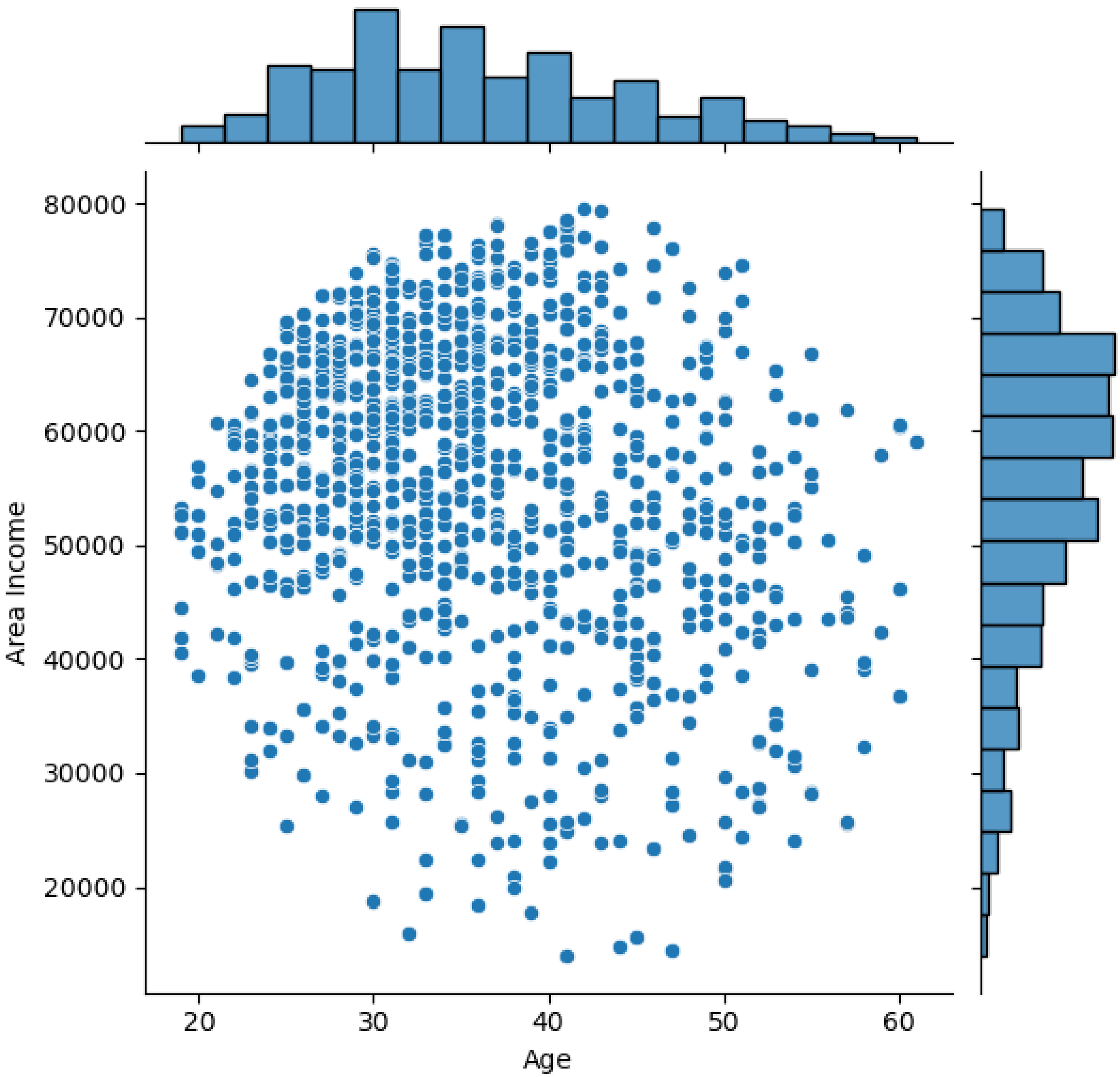


Graphical Representation : Area Income, Age :

page 120

In [91]:
```
sns.jointplot(x = 'Age', y = 'Area Income', data = ad_data)
```

Out[91]: <seaborn.axisgrid.JointGrid at 0x21dbae8ac70>

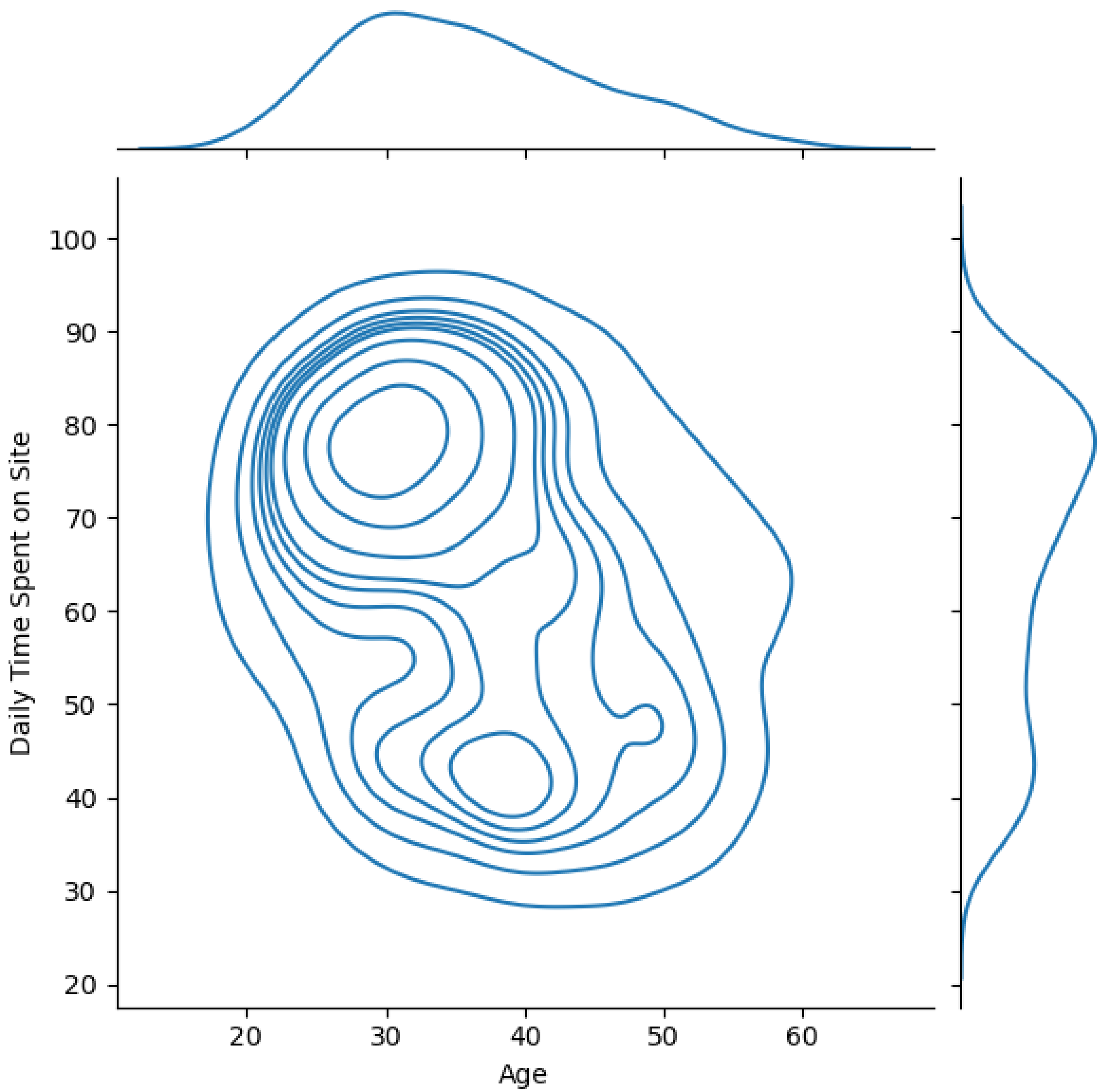


Graphical Representation : Age, Daily Time Spent on Site :

page 120

In [100]:
```
sns.jointplot(x = 'Age', y = 'Daily Time Spent on Site', data = ad_data, kind = 'kde')
```

Out[100]: <seaborn.axisgrid.JointGrid at 0x21dbc4df0d0>



In [102]:

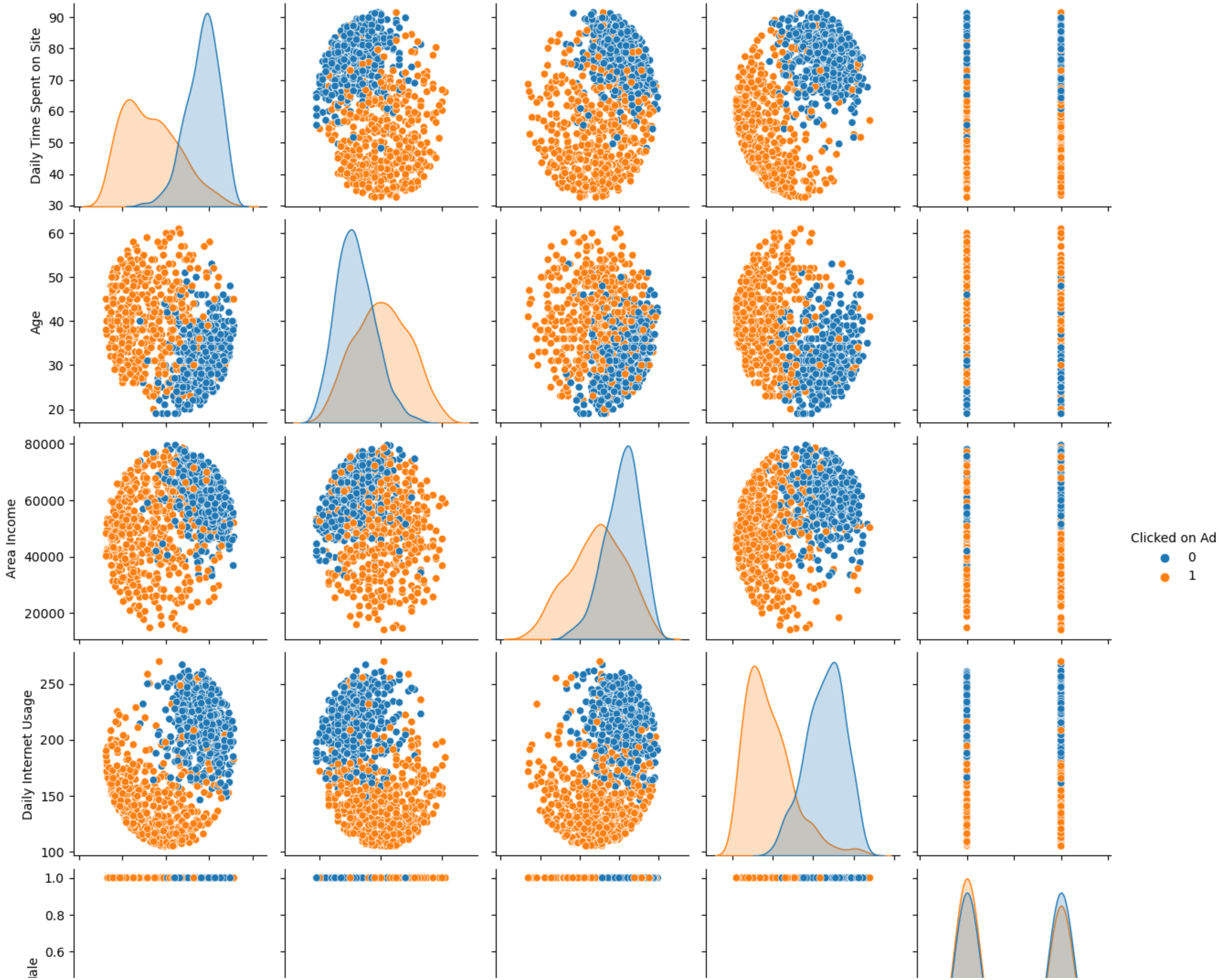
```
# Now, let's take particular Column 'Clicked on Ad' and Declaring 'pairplot'
# means, How many 'Numerics' are there, that many it will be 'Represented' :


ad_data.columns
```

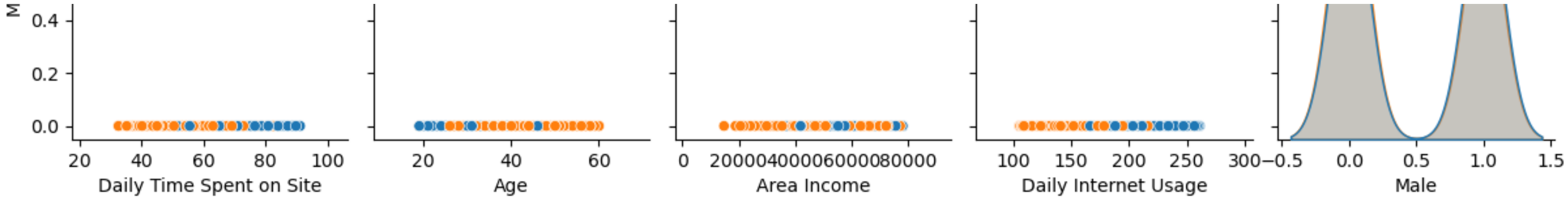
Out[102]:

```
Index(['Daily Time Spent on Site', 'Age', 'Area Income',
       'Daily Internet Usage', 'Ad Topic Line', 'City', 'Male', 'Country',
       'Timestamp', 'Clicked on Ad'],
      dtype='object')
```

In [103]:
```
sns.pairplot(ad_data, hue = 'Clicked on Ad')
```

Out[103]:
```
<seaborn.axisgrid.PairGrid at 0x21dbe359f70>
```


Daily Time Spent on Site
Age
Area Income
Daily Internet Usage
ale
Clicked on Ad
0
1



In [104]:
```
ad_data.columns
```

Out[104]:
```
Index(['Daily Time Spent on Site', 'Age', 'Area Income',
       'Daily Internet Usage', 'Ad Topic Line', 'City', 'Male', 'Country',
       'Timestamp', 'Clicked on Ad'],
      dtype='object')
```

In [105]:
```
from sklearn.model_selection import train_test_split
```

In [129]:
```
# Here, Capital 'X' :

X = ad_data[['Daily Time Spent on Site', 'Age', 'Area Income',
       'Daily Internet Usage', 'Male']]
y = ad_data['Clicked on Ad']
```

Now Copying Data from ad_data.columns-output to 'X' :

page 122

In [130]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

In [128]:
```
X.head()
```

Out[128]:

	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

In [117]:
```
X.head(2)
```

Out[117]:

	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

In [131]:
```
from sklearn.linear_model import LogisticRegression
```

In [132]:
```
# logmodel.fit between X_train, y_train

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

Out[132]:
```
LogisticRegression()
```

In [133]:
```
# logmodel.fit between X_train, y_train
# Due to ValueError: could not convert string to float: 'Configurable impactful capacity'
# Here, We removed the data from 114th Sum - "City, Country, Ad Topic Line, Time Stamp"
# Now it's Working :

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

Out[133]:
```
LogisticRegression()
```

In [136]:
```
X.head()
```

Out[136]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Male
0	68.95	35	61833.90	256.09	0
1	80.23	31	68441.85	193.77	1
2	69.47	26	59785.94	236.50	0
3	74.15	29	54806.18	245.89	1
4	68.37	35	73889.99	225.58	0

Now let's do 'PREDICTION' :

PAGE 124

In [137]:
```
predictions = logmodel.predict(X_test)
```

In [138]:
```
# Once we do prediction, What We need to do means,
```

In [139]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [140]:
```
# This is the particular thing, We are getting an a Classification_report and Confusion_matrix :

print(classification_report(y_test, predictions))
print('\n')
print(confusion_matrix(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91       162
           1       0.96      0.85      0.90       168

    accuracy                           0.91       330
   macro avg       0.91      0.91      0.91       330
weighted avg       0.91      0.91      0.91       330



[[156   6]
 [ 25 143]]
```

In [ ]: