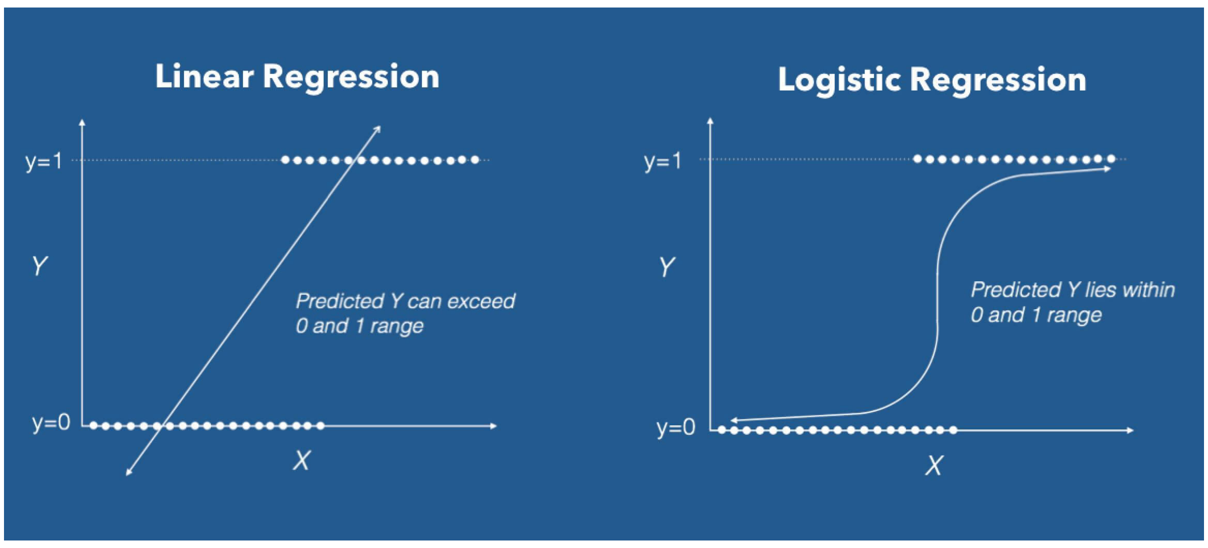
Logistics Regression.

In [9]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/xx.jpg',height=600,width=600)
```

Out[9]:



Logistic Regression is one of the basic and popular algorithm to solve a classification problem. It is named as 'Logistic Regression', because it's underlying technique is quite the same as Linear Regression. The term "Logistic" is taken from the **Logit** function that is used in this method of classification.

Why Logistics Regression ?

We identify problem as classification problem when independent variables are continuous in nature and dependent variable is in categorical form.

i.e. in classes like positive class and negative class. The real life example of classification example would be, to categorize the mail as spam or not spam, to categorize the tumor as malignant or benign and to categorize the transaction as fraudulent or genuine. All these problem's answers are in categorical form i.e. Yes or No. and that is why they are two class classification problems.

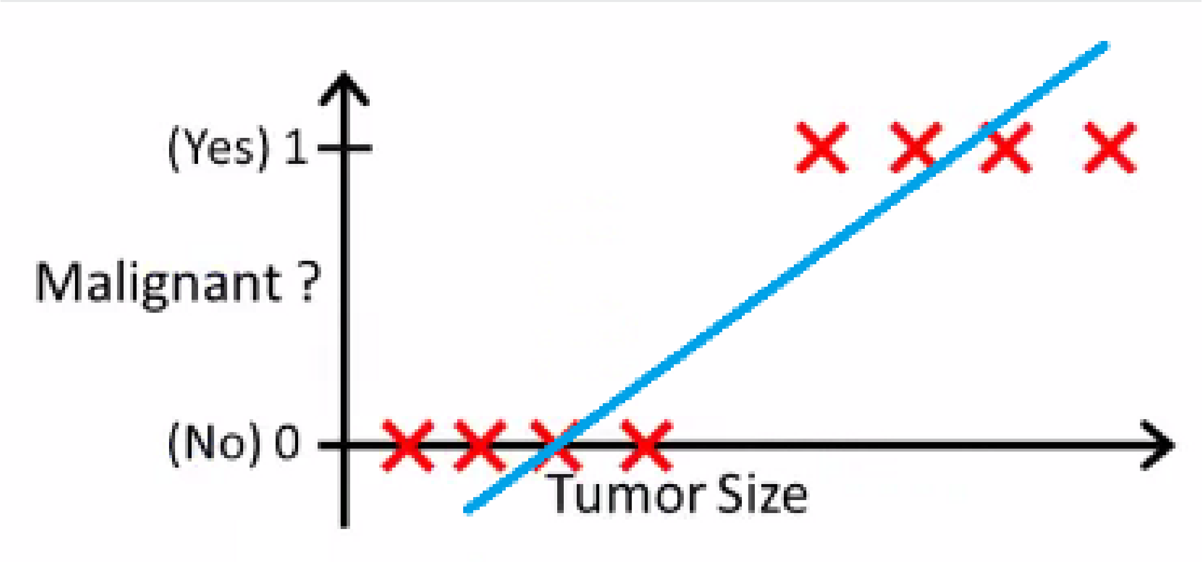
Although, sometime we come across **more than 2 classes and still it is a classification problem.** These types of problems are known as **multi class classification problems.**

Why not use Linear Regression?

Suppose we have a data of tumor size vs its malignancy. As it is a classification problem, if we plot, we can see, all the values will lie on 0 and 1. And if we fit best found regression line, by assuming the threshold at 0.5, we can do line pretty reasonable job.

In [10]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/ye.png',height=600,width=600)
```

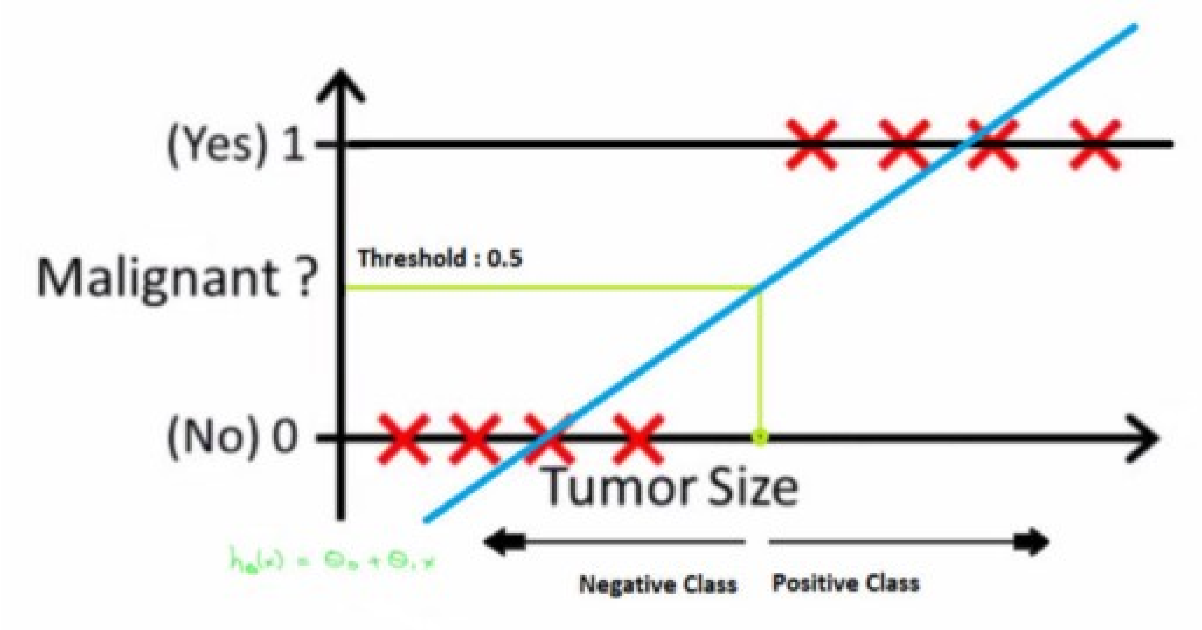
Out[10]:



We can decide the point on the x axis from where all the values lie to its left side are considered as negative class and all the values lie to its right side are positive class.

In [14]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/dp.jpeg',height=600,width=600)
```

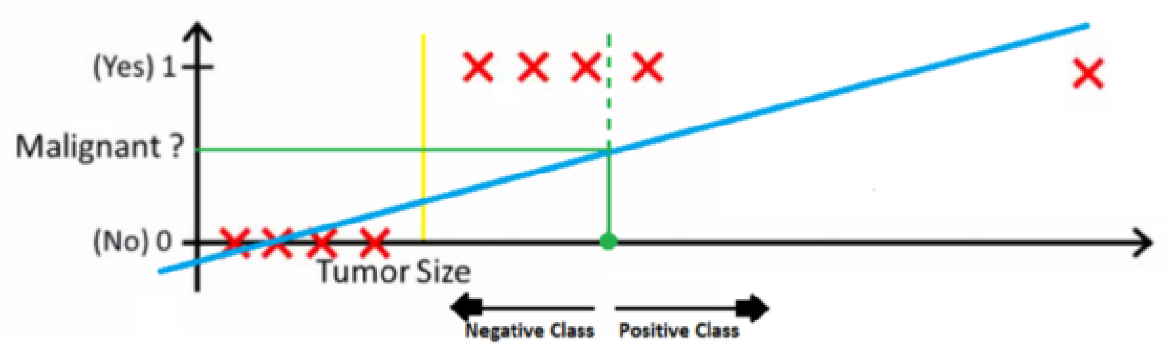
Out[14]:



But what if there is an outlier in the data. Things would get pretty messy. For example, for 0.5 threshold,

In [15]:
```
from IPython.display import Image
Image(filename='C:/Users/Microsoft/Desktop/pandas/os.png',height=600,width=600)
```

Out[15]:



If we fit best found regression line, it still won't be enough to decide any point by which we can differentiate classes. It will put some positive class examples into negative class. The green dotted line (Decision Boundary) is dividing malignant tumors from benign tumors but the line should have been at a yellow line which is clearly dividing the positive and negative examples. **So just a single outlier is disturbing the whole linear regression predictions. And that is where logistic regression comes into a picture.**

Logistics Regression With Python In Machine Learning.

Logistics regression is a classification algorithm used to assign observations to decreate set of classes unlike to the linear regression.

- the linear regression we get the continuous observations but in the logistics regression we get the decreate observations.

True or False.

Some of the eaxample of classification problem are :-

- Mail spam orf not spam
- Online Trnsactions froud or not froud
- Tumor Malignant or Benign.

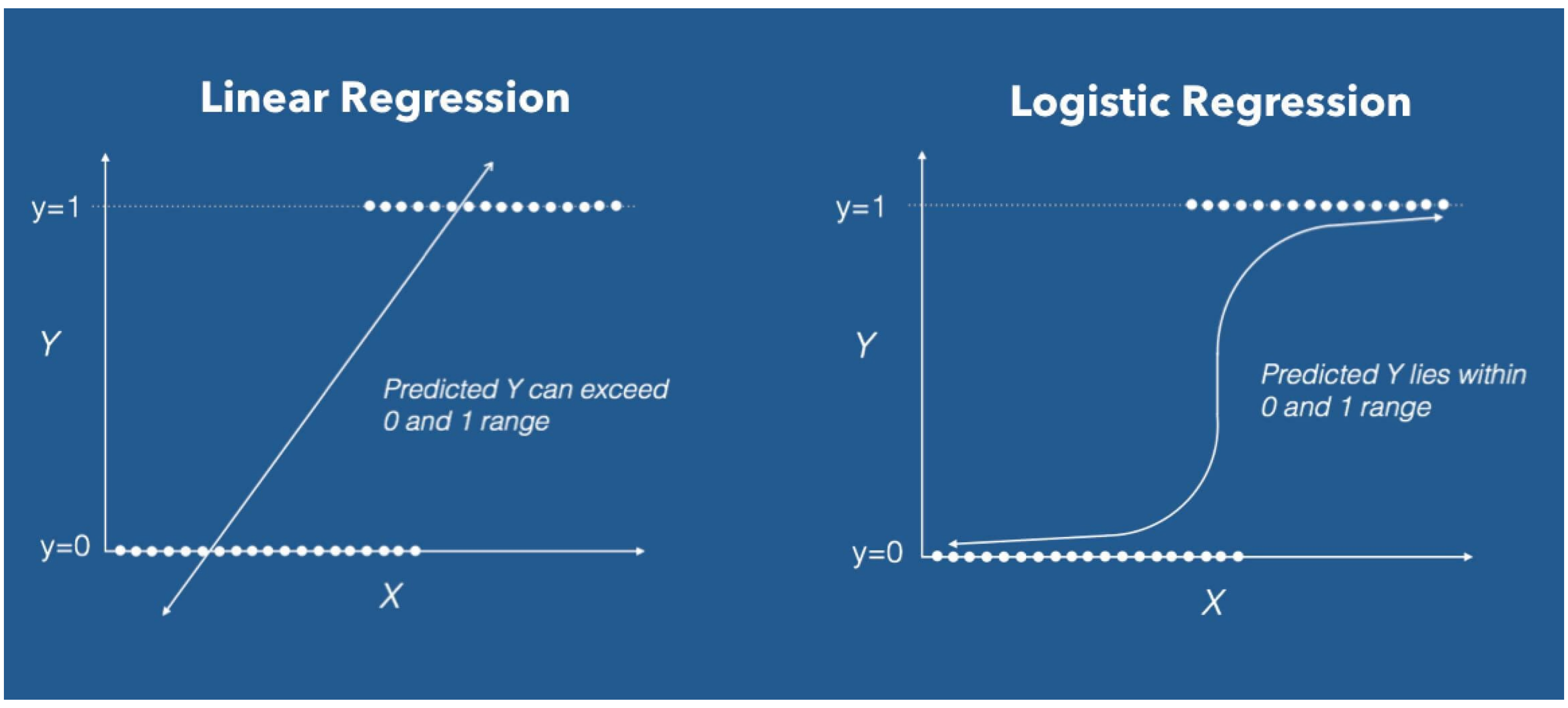
Logistics regression transforms it's output using the logistics **signomoid function** to return a **probablity** value.

- In between the probablity 0 and 1 there is signomoid function in that it returns the only true(1) or false(0) if we consider baseline as 0.5 that is threshold.
- If we get the probablity more than the 0 but less than the 0.5 then it will return the 0.
- Similarly,if we got the value of probablity is more than the 0.5 but the less than 1 that it will return the 1 as an output.

Logistics Regression.

In [4]:
```
from IPython.display import Image
Image(filename = 'C:/Users/Microsoft/Desktop/pandas/HG.jpg',height=500,width=800)
```

Out[4]:



Logistics regression is machine learning algorithm which is use for the classification problem.It is a predictive analysis algorithm and based on the concept of probablity.

You can also apply for the rain forecasting but if we can to measure amount of the rain then we have to apply Linear Regression.

We can call logistic Regression as Linear Regression model but the Logistics Regression uses more complex cost function,the cost function can be defined as the 'Sigmoid Funtion' or known as 'Logistics Function' instaed of linear function.

The hypothesis of logistics regression tends it to limit the cost function between the 0 and 1.

Threrefore the linear function fails to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of the logistics regression.

Sigmoid Funtion

In [7]:
```
from IPython.display import Image
Image(filename = 'C:/Users/Microsoft/Desktop/pandas/file.png',height=400,width=300)
```

Out[7]:

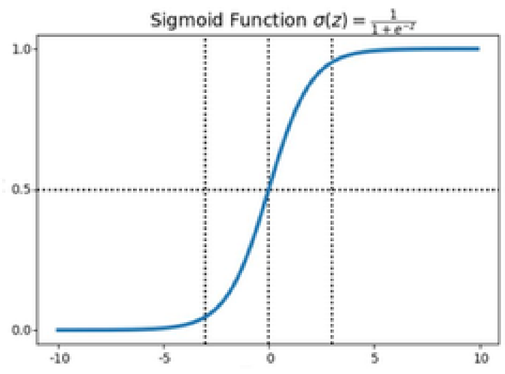


Fig. 1

It's value always in between 0 and 1.

In order to predict the values of probablities.We use the sigmoid function.This function maps any real value into another real value between 0 and 1.

In machine learning,we use sigmoid to map predictions to probablities.

Decision Boundary.

Let's consider that we having the scatter plot and observed data points are the classified in two categories by the straight line which is going through the observed points.

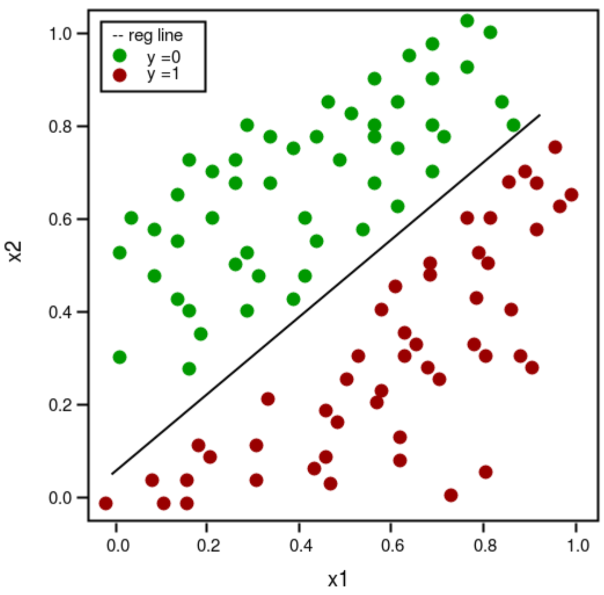
and above the line is categoised with probablity 1 and below the having the probablity 0.

The line is known as Regression line.

We expect our classifier to give us a set of outputs or classes based on probablity when we pass the inputs through a prediction function and returns a probablity score between 0 and 1.

In [8]:
```
from IPython.display import Image
Image(filename = 'C:/Users/Microsoft/Desktop/pandas/Logistic-Regression.png',height=400,width=300)
```

Out[8]:



In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```python
data =pd.read_csv('advertising.csv')
```

In [4]:
```python
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]:
```python
data.tail()
```

Out[5]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
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

In [6]:
```python
data.info()
```

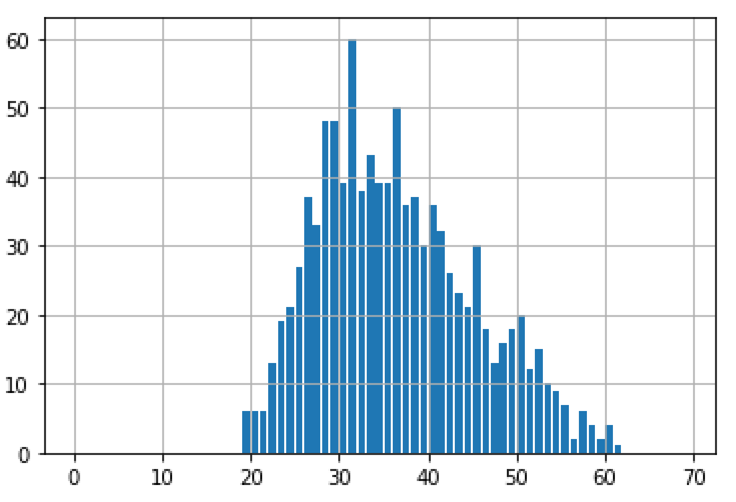
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
Daily Time Spent on Site    1000 non-null float64
Age                         1000 non-null int64
Area Income                 1000 non-null float64
Daily Internet Usage        1000 non-null float64
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 object
Clicked on Ad               1000 non-null int64
dtypes: float64(3), int64(3), object(4)
memory usage: 62.6+ KB
```

In [7]:
```python
data.describe()
```

Out[7]:

	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

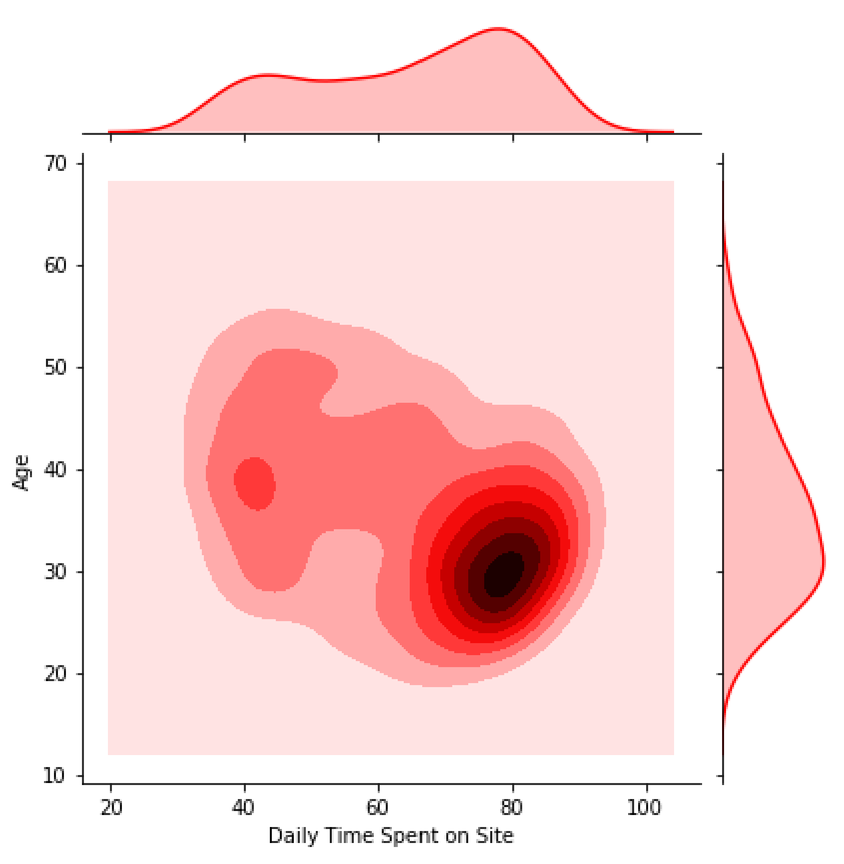
In [8]:
```python
plt.hist(data['Age'],bins=range(70),rwidth=0.8,density=False,)
plt.grid()
plt.show()
```



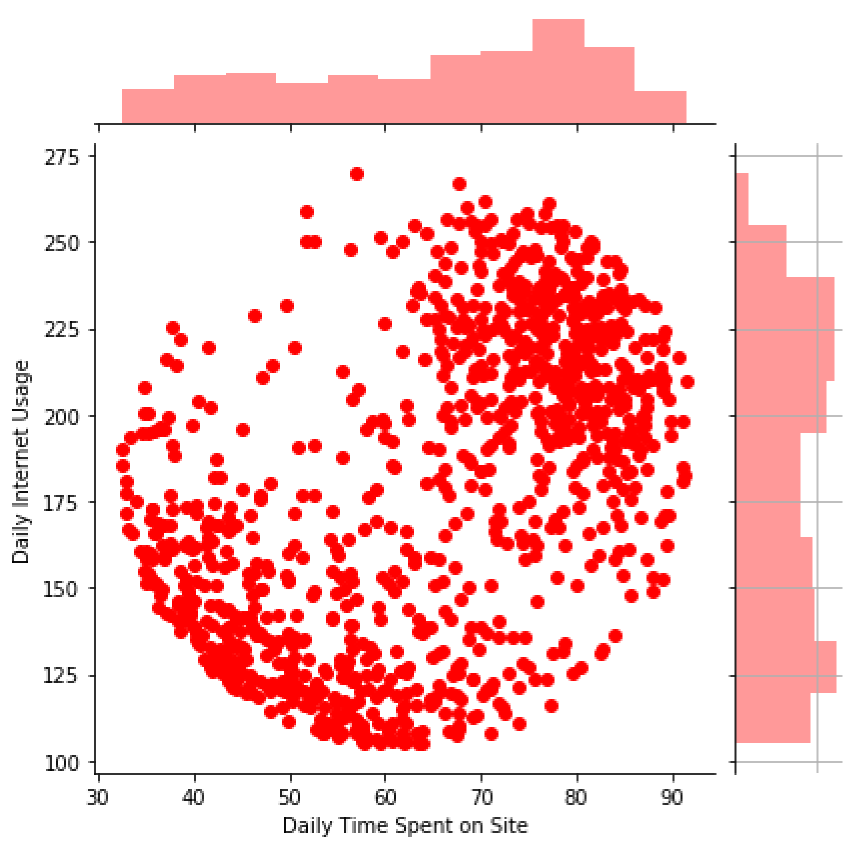
Create a jointplot showing the kde distributions of Daily Time spent on site vs. Age.

In [9]:
```python
sns.jointplot(x='Daily Time Spent on Site',y='Age',data=data,kind='kde',color='red')
```

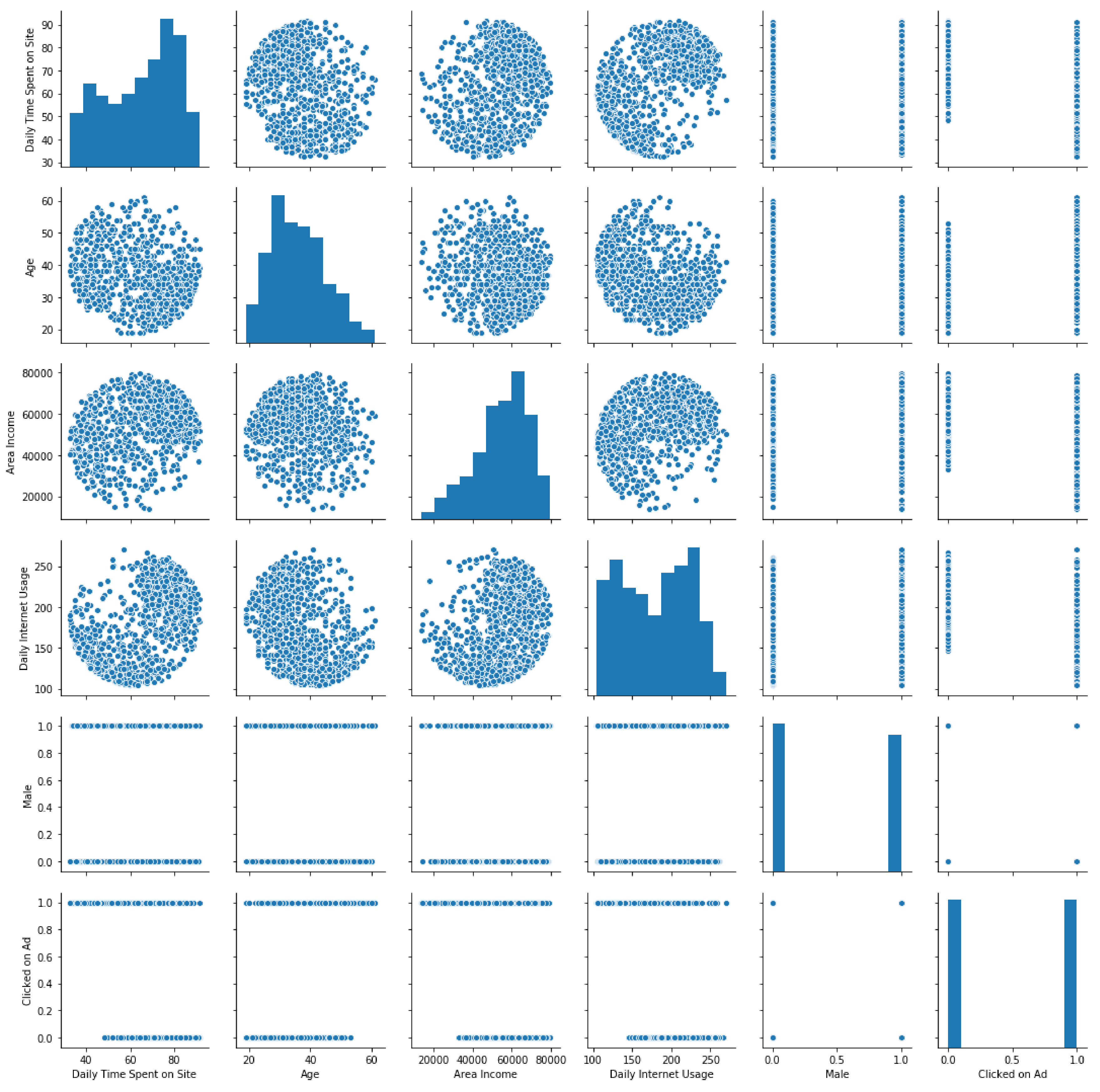
Out[9]: <seaborn.axisgrid.JointGrid at 0x5637330>



In [16]:
```python
sns.jointplot(x='Daily Time Spent on Site',y='Daily Internet Usage',data=data,kind='scatter',color='red')
plt.grid()
plt.show()
```



In [17]:
```python
sns.pairplot(data)
plt.show()
```



In [21]:
```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [33]:
```python
x= data.drop(columns=(['Clicked on Ad','City','Country','Timestamp','Ad Topic Line'])).to_numpy()
y = data['Clicked on Ad'].to_numpy()
```

In [34]:
```python
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=7,train_size=0.80)
```

In [35]:
```python
x_train.shape,x_test.shape
```

Out[35]: ((800, 5), (200, 5))

In [36]:
```python
model = LogisticRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

In [37]:
```python
from sklearn.metrics import accuracy_score,classification_report
```

In [38]:
```python
print('Accuracy score :-',accuracy_score(y_test,y_pred))
```

```
Accuracy score :- 0.9
```

In [39]:
```python
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.92      0.90        97
           1       0.92      0.88      0.90       103

    accuracy                           0.90       200
   macro avg       0.90      0.90      0.90       200
weighted avg       0.90      0.90      0.90       200
```

```
In [1]: from IPython.display import Image
        Image(filename='C:/Users/Microsoft/Desktop/pandas/na.GIF')
```

Out[1]: <IPython.core.display.Image object>

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

What are the types of logistic regression

- Binary (eg. Tumor Malignant or Benign)
- Multi-linear functions failsClass (eg. Cats, dogs or Sheep's)
- Ordinal Logistic Regression: the target variable has three or more ordinal categories such as restaurant or product rating from 1 to 5.

Logistic Regression.

Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.

We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function.

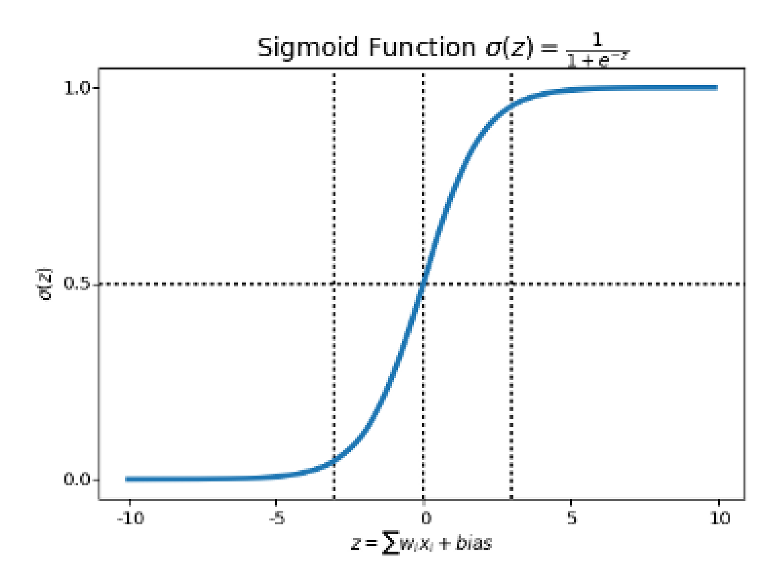
The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression.

What is the Sigmoid Function?

In order to map predicted values to probabilities, we use the Sigmoid function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities

```
In [3]: from IPython.display import Image
        Image(filename='C:/Users/Microsoft/Desktop/pandas/ca.png')
```

Out[3]:




```
In [4]: from IPython.display import Image
        Image(filename='C:/Users/Microsoft/Desktop/pandas/ka.png')
```

Out[4]:

$$f(x) = \frac{1}{1+e^{-(x)}}$$

Hypothesis Representation

When using linear regression we used a formula of the hypothesis i.e.

$h\Theta(x) = \beta_0 + \beta_1 X$

For logistic regression we are going to modify it a little bit i.e.

$\sigma(Z) = \sigma(\beta_0 + \beta_1 X)$

We have expected that our hypothesis will give values between 0 and 1.

$Z = \beta_0 + \beta_1 X$

$h\Theta(x) = \text{sigmoid}(Z)$

i.e. $h\Theta(x) = 1/(1 + e\text{^}-(\beta_0 + \beta_1 X)$

```
In [5]: from IPython.display import Image
        Image(filename='C:/Users/Microsoft/Desktop/pandas/ha.png')
```

Out[5]:

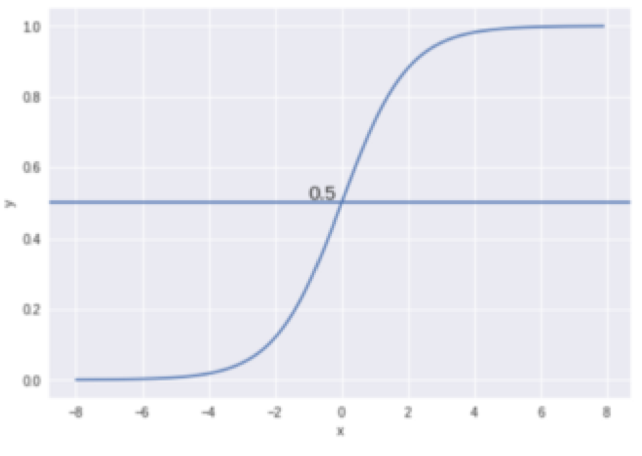
$$h\theta(X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

Decision Boundary.

We expect our classifier to give us a set of outputs or classes based on probability when we pass the inputs through a prediction function and returns a probability score between 0 and 1. For Example, We have 2 classes, let's take them like cats and dogs(1 — dog , 0 — cats). We basically decide with a threshold value above which we classify values into Class 1 and of the value goes below the threshold then we classify it in Class 2.

```
In [7]: from IPython.display import Image
        Image(filename='C:/Users/Microsoft/Desktop/pandas/lss.png')
```

Out[7]:



As shown in the above graph we have chosen the threshold as 0.5, if the prediction function returned a value of 0.7 then we would classify this observation as Class 1(DOG). If our prediction returned a value of 0.2 then we would classify the observation as Class 2(CAT).

Cost Funtion.

MSE measures the average squared difference between an observation's actual and predicted values. The output is a single number representing the cost, or score, associated with our current set of weights. Our goal is to minimize MSE to improve the accuracy of our model.

We learnt about the cost function J(θ) in the Linear regression, the cost function represents optimization objective i.e. we create a cost function and minimize it so that we can develop an accurate model with minimum error.

```
In [16]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/ua.png',height=200,width=400)
```

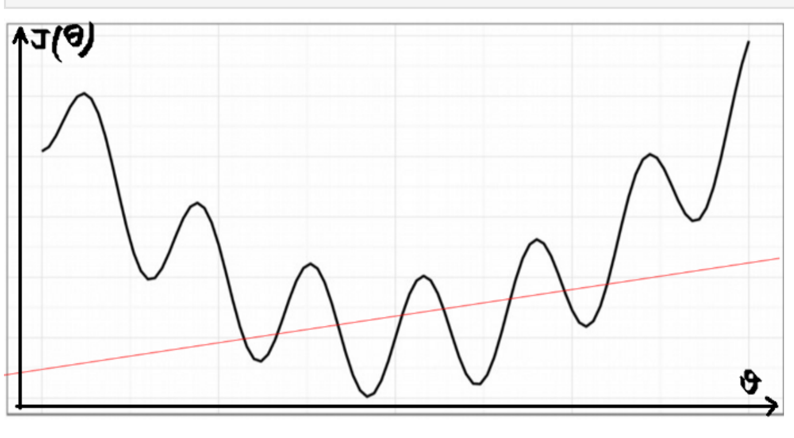
Out[16]:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2.$$

If we try to use the cost function of the linear regression in 'Logistic Regression' then it would be of no use as it would end up being a non-convex function with many local minimums, in which it would be very difficult to minimize the cost value and find the global minimum.

```
In [17]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/dda.png',height=200,width=400)
```

Out[17]:



For logistic regression, the Cost function is defined as:

```
-log(hθ(x))   if y  = 1
-log(1-hθ(x)) if y  = 0
```

Gradient Descent.

We minimized J(Θ) by trial and error above — just trying lots of values and visually inspecting the resulting graph. There must be a better way? Queue gradient descent. Gradient Descent is a general function for minimizing a function, in this case the Mean Squared Error cost function.

Gradient Descent basically just does what we were doing by hand — change the theta values, or parameters, bit by bit, until we hopefully arrived a minimum.

Now the question arises, how do we reduce the cost value. Well, this can be done by using Gradient Descent. The main goal of Gradient descent is to minimize the cost value. i.e. min J(θ)

Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.

```
In [21]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/sv.png',height=200,width=250)
```

Out[21]:

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$$

```
In [20]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/xxa.jpeg',height=200,width=400)
```

Out[20]:

Want $\min_\theta J(\theta)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update all θ_j)

$\}$

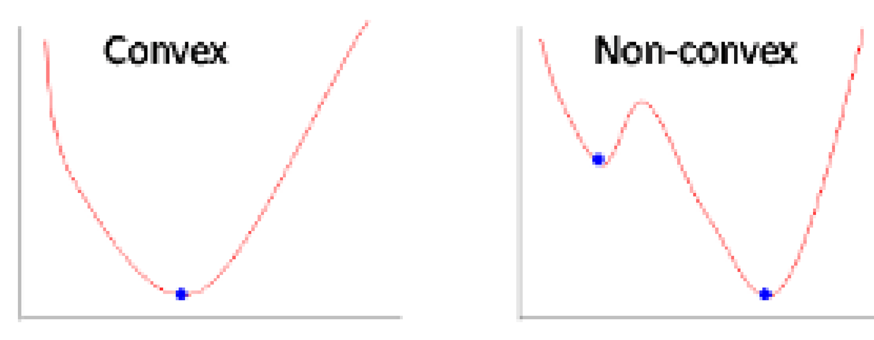
Gradient descent has an analogy in which we have to imagine ourselves at the top of a mountain valley and left stranded and blindfolded, our objective is to reach the bottom of the hill. Feeling the slope of the terrain around you is what everyone would do. Well, this action is analogous to calculating the gradient descent, and taking a step is analogous to one iteration of the update to the parameters.

Why cost function which has been used for linear can not be used for logistic?

Linear regression uses mean squared error as its cost function. If this is used for logistic regression, then it will be a non-convex function of parameters (theta). Gradient descent will converge into global minimum only if the function is convex.

```
In [23]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/za.png',height=500,width=500)
```

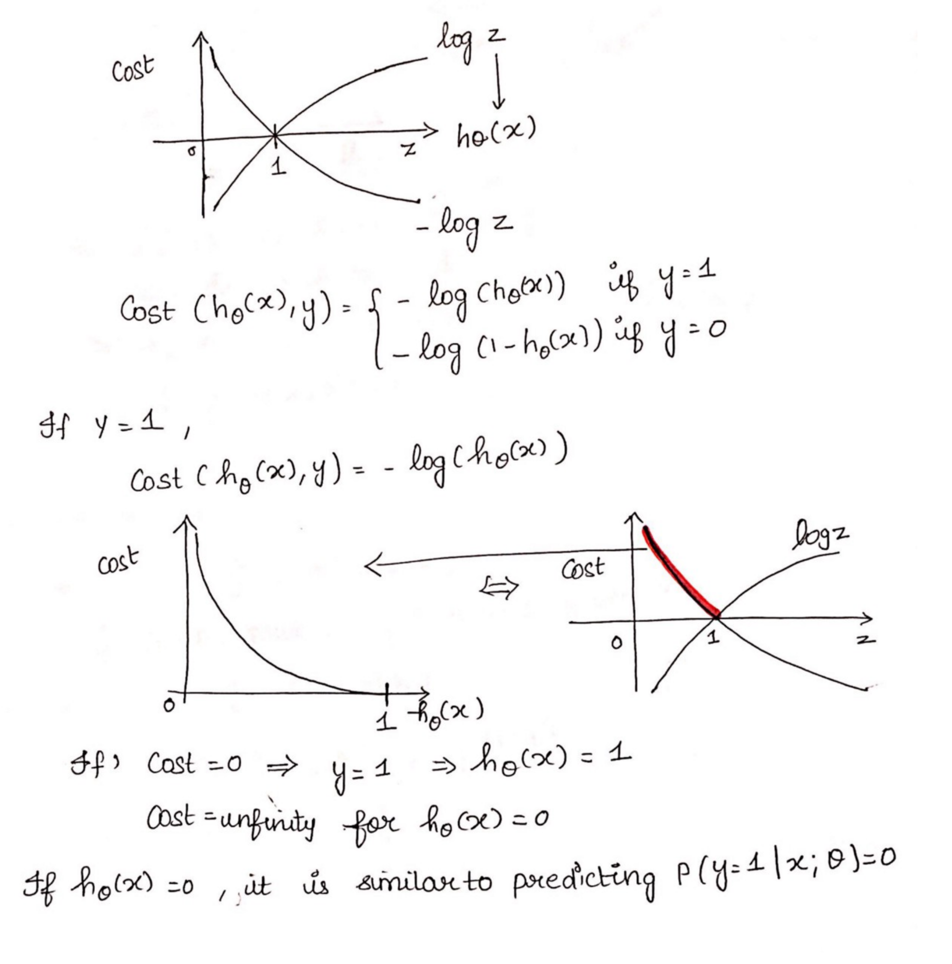
Out[23]:



Cost function explanation.

```
In [24]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/qo.jpeg',height=500,width=500)
```

Out[24]:



Simplified cost function

```
In [26]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/oa.png',height=500,width=500)
```

Out[26]:

Cost($h_\Theta(x)$, y) = -y log($h_\Theta(x)$) – (1-y) log (1- $h_\Theta(x)$)

If y = 1, (1-y) term will become zero, therefore – log ($h_\Theta(x)$) alone will be present

If y = 0, (y) term will become zero, therefore – log (1- $h_\Theta(x)$) alone will be present

Why this cost function?

```
In [27]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/ja.jpeg',height=500,width=500)
```

Out[27]:

Let us consider,

* $\hat{y} = P(y=1|x)$

$\hat{y}$ is the probability that $y = 1$, given x

* $1-\hat{y} = P(y=0|x)$

* $P(y|x) = \hat{y}^{y} \cdot (1-\hat{y})^{(1-y)}$

If $y = 1 \Rightarrow P(y|x) = \hat{y}$

```
In [31]: from IPython.display import Image
         Image(filename='C:/Users/Microsoft/Desktop/pandas/vac.jpeg',height=500,width=500)
```

Out[31]:

$\Rightarrow \log(\hat{y}^{y} \cdot (1-\hat{y})^{(1-y)})$

$\Rightarrow y \log \hat{y} + (1-y) \log(1-\hat{y})$

$\Rightarrow -L(\hat{y}, y)$

$\boxed{\log P(y|x) = -L(\hat{y}, y)}$

This negative function is because when we train, we need to maximize the probability by minimizing loss function. Decreasing the cost will increase the maximum likelihood assuming that samples are drawn from an identically independent distribution.

This implementation is for binary logistic regression. For data with more than 2 classes, **softmax regression** has to be used.

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

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('iris.csv')
```

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

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [4]:
```
data.tail()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.3+ KB
```

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

Out[6]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

OneHotEncoding

In works as an dummies variable,it use to crate columns according catrgories avalible in the specific variable but not avalible in the.

Label Encoder

It will just covert the text to numeric.

Get_dummies.

The Dummy Variable trap is a scenario in which the independent variables are multicollinear - a scenario in which two or more variables are highly correlated; in simple terms one variable can be predicted from the others. To demonstrate the Dummy Variable Trap, take the case of gender (male/female) as an example.

It is mainly use for nominal.

In [22]:
```
dumi = pd.get_dummies(data['species'])
```

In [23]:
```
dumi.head()
```

Out[23]:

	setosa	versicolor	virginica
0	1	0	0
1	1	0	0
2	1	0	0
3	1	0	0
4	1	0	0

In [24]:
```
dum = pd.get_dummies(data['species'],drop_first=True)
```

In [25]:
```
dum.head()
```

Out[25]:

	versicolor	virginica
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0

In [39]:
```
df = pd.concat([dum,data],axis=1)
```

In [42]:
```
df.head()
```

Out[42]:

	versicolor	virginica	sepal_length	sepal_width	petal_length	petal_width	species
0	0	0	5.1	3.5	1.4	0.2	setosa
1	0	0	4.9	3.0	1.4	0.2	setosa
2	0	0	4.7	3.2	1.3	0.2	setosa
3	0	0	4.6	3.1	1.5	0.2	setosa
4	0	0	5.0	3.6	1.4	0.2	setosa

LabelEncoder

Encode categorical features using an ordinal encoding scheme. Encode categorical features as a one-hot numeric array. LabelEncoder can be used to normalize labels. It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels.

In [44]:
```
data.head()
```

Out[44]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [46]:
```
from sklearn.preprocessing import LabelEncoder
```

It is use for ordered.

In [47]:
```
label = LabelEncoder()
data['species'] = label.fit_transform(data['species'])
```

In [51]:
```
data['species'].value_counts()
```

Out[51]:
```
2    50
1    50
0    50
Name: species, dtype: int64
```

Either we can use the get_dummies or OneHotEncoding.

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

In [4]:
```
data = pd.read_csv('Social_Network_Ads (2).csv')
```

In [5]:
```
data.head()
```

Out[5]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

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

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

In [7]:
```
data.shape
```

Out[7]:
```
(400, 5)
```

In [8]:
```
data.describe()
```

Out[8]:

	User ID	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000
mean	1.569154e+07	37.655000	69742.500000	0.357500
std	7.165832e+04	10.482877	34096.960282	0.479864
min	1.556669e+07	18.000000	15000.000000	0.000000
25%	1.562676e+07	29.750000	43000.000000	0.000000
50%	1.569434e+07	37.000000	70000.000000	0.000000
75%	1.575036e+07	46.000000	88000.000000	1.000000
max	1.581524e+07	60.000000	150000.000000	1.000000

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
User ID            400 non-null int64
Gender             400 non-null object
Age                400 non-null int64
EstimatedSalary    400 non-null int64
Purchased          400 non-null int64
dtypes: int64(4), object(1)
memory usage: 14.1+ KB
```

In [10]:
```
data.nunique()
```

Out[10]:
```
User ID            400
Gender               2
Age                 43
EstimatedSalary    117
Purchased            2
dtype: int64
```

In [11]:
```
data.isnull().sum()
```

Out[11]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [21]:
```
x= data[['Age','EstimatedSalary']].to_numpy()

y = data['Purchased'].to_numpy()
```

In [22]:
```
from sklearn.model_selection import train_test_split
```

In [23]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.80,random_state=42)
```

In [26]:
```
x_train.shape,x_test.shape
```

Out[26]:
```
((320, 2), (80, 2))
```

In [27]:
```
x_train = x_train.astype('float')
```

In [28]:
```
x_test = x_test.astype('float')
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
```

In [30]:
```
scaler  = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test  = scaler.transform(x_test)
```

In [31]:
```
from sklearn.linear_model import LogisticRegression
```

In [33]:
```
import warnings
warnings.filterwarnings('ignore')

model = LogisticRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
```

In [43]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

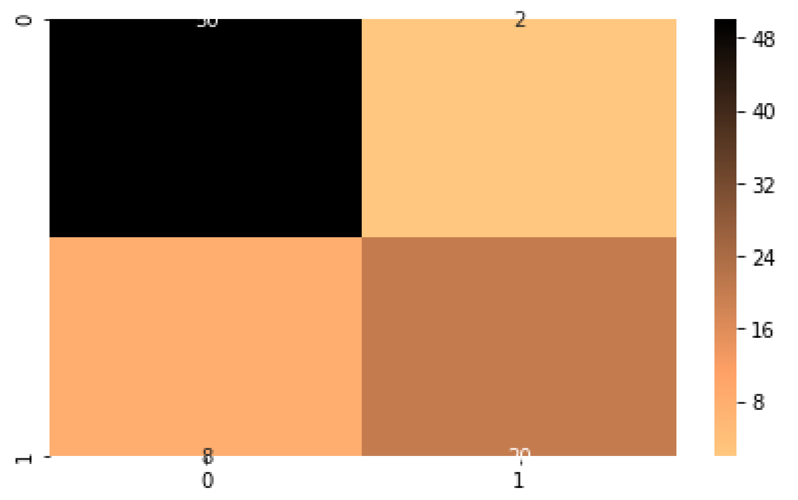
In [37]:
```
print('Accuracy :',accuracy_score(y_test,y_pred))
```

```
Accuracy : 0.875
```

In [38]:
```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[38]:
```
array([[50,  2],
       [ 8, 20]], dtype=int64)
```

In [47]:
```
plt.figure(figsize=(7,4))
sns.heatmap(mat,annot=True,cmap='copper_r')
plt.show()
```



In [48]:
```
report=print(classification_report(y_test,y_pred))
report
```

```
              precision    recall  f1-score   support

           0       0.86      0.96      0.91        52
           1       0.91      0.71      0.80        28

    accuracy                           0.88        80
   macro avg       0.89      0.84      0.85        80
weighted avg       0.88      0.88      0.87        80
```

Imbalace data

- A classfication dataset with skewed class proportion is called imbalance.
- Classes that make up large proportion of the datasets are called majority classes.
- Classes that make up small proportion of the datasets are called majority classes.

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```python
data = pd.read_csv('pima.csv')
```

In [5]:
```python
data.head()
```

Out[5]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [6]:
```python
data.tail()
```

Out[6]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

In [7]:
```python
data.isnull().sum()
```

Out[7]:
```
Preg     0
Plas     0
Pres     0
skin     0
test     0
mass     0
pedi     0
age      0
class    0
dtype: int64
```

In [8]:
```python
data.shape
```

Out[8]: (768, 9)

In [9]:
```python
data.describe()
```

Out[9]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

In [10]:
```python
data.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
Preg     768 non-null int64
Plas     768 non-null int64
Pres     768 non-null int64
skin     768 non-null int64
test     768 non-null int64
mass     768 non-null float64
pedi     768 non-null float64
age      768 non-null int64
class    768 non-null int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [11]:
```python
data.nunique()
```

Out[11]:
```
Preg      17
Plas     136
Pres      47
skin      51
test     186
mass     248
pedi     517
age       52
class      2
dtype: int64
```

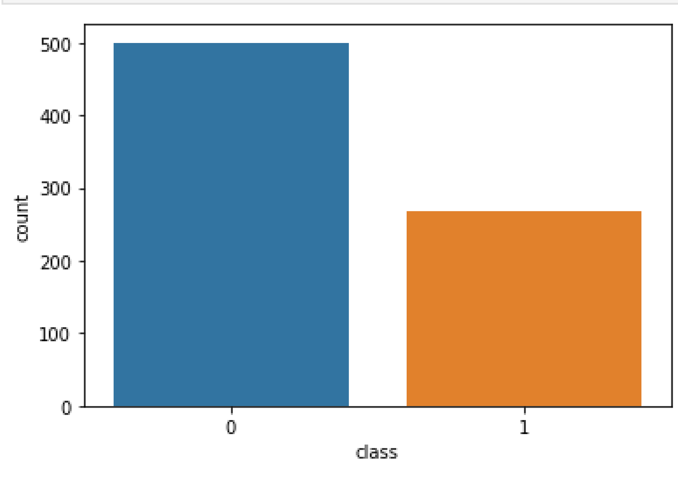
In [12]:
```python
data.head()
```

Out[12]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

From the we see that the class is not balance in nature.

In [13]:
```python
sns.countplot(data['class'])
plt.show()
```



In [14]:
```python
x = data.drop(columns='class').to_numpy()
y = data['class']
```

In [15]:
```python
from sklearn.model_selection import train_test_split
```

In [16]:
```python
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state=7,train_size=0.7)
```

In [17]:
```python
x_train.shape,x_test.shape
```

Out[17]: ((537, 8), (231, 8))

In [18]:
```python
from sklearn.linear_model import LogisticRegression
```

In [19]:
```python
print('Before sampling Count of level 1 :-',y_train.value_counts()[1])
print('Before Sampling Count of level 0 :-',y_train.value_counts()[0])
```

```
Before sampling Count of level 1 :- 184
Before Sampling Count of level 0 :- 353
```

In [20]:
```python
#!pip install imblearn

from imblearn import over_sampling,under_sampling
```

```
Using TensorFlow backend.
```

In [21]:
```python
from imblearn.over_sampling import SMOTE
```

In [22]:
```python
sm = SMOTE(sampling_strategy=1,random_state=1,k_neighbors=5)
#sampling_strategy=1 means it will fill complete label 1 and make it balance
#k=5
```

In [23]:
```python
x_train_res,y_train_res = sm.fit_sample(x_train,y_train.ravel())
#ravel -1d and 2d format and it will take only in one form
```

In [24]:
```python
print('After upsampling Count of level 1 :-',sum(y_train_res==1))
print('After upSampling Count of level 0 :-',sum(y_train_res==0))
```

```
After upsampling Count of level 1 :- 353
After upSampling Count of level 0 :- 353
```

In [25]:
```python
x_train_res.shape,y_train_res.shape
```

Out[25]: ((706, 8), (706,))

In [26]:
```python
from sklearn.linear_model import LogisticRegression
```

In [33]:
```python
model = LogisticRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)

from sklearn.metrics import accuracy_score,classification_report,r2_score,confusion_matrix
```

In [34]:
```python
print('Accuracy :-',accuracy_score(y_test,y_pred))
```

```
Accuracy :- 0.7489177489177489
```

In [35]:
```python
model = LogisticRegression()
model.fit(x_train_res,y_train_res)
y_predict = model.predict(x_test)
```

In [30]:
```python
print('Accuracy :-',accuracy_score(y_test,y_predict))
```

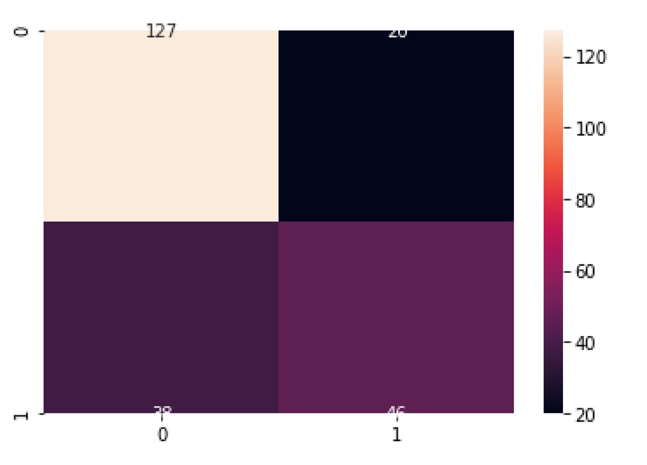
```
Accuracy :- 0.7532467532467533
```

In [36]:
```python
mat= confusion_matrix(y_test,y_pred)
mat
```

Out[36]:
```
array([[127,  20],
       [ 38,  46]], dtype=int64)
```

In [40]:
```python
sns.heatmap(mat,annot=True, fmt='d')
```

Out[40]: <matplotlib.axes._subplots.AxesSubplot at 0xf114850>



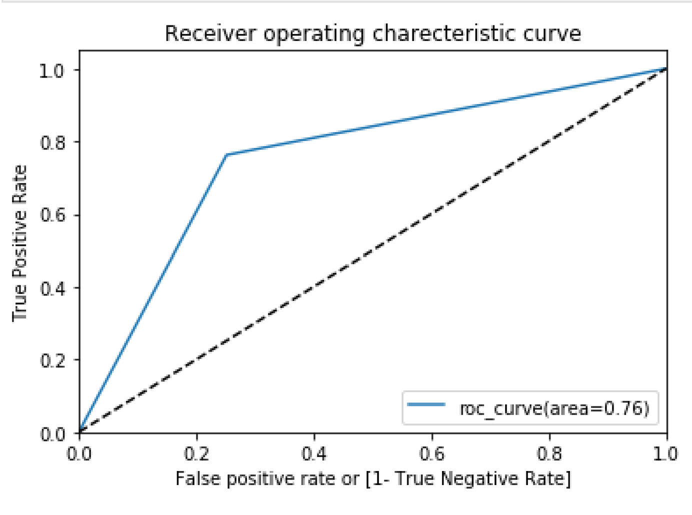
In [37]:
```python
print('Classification Report before Upsampling')
print(classification_report(y_test,y_pred))
print('Classification Report After Upsampling')
```

```
Classification Report before Upsampling
              precision    recall  f1-score   support

           0       0.77      0.86      0.81       147
           1       0.70      0.55      0.61        84

    accuracy                           0.75       231
   macro avg       0.73      0.71      0.71       231
weighted avg       0.74      0.75      0.74       231

Classification Report After Upsampling
```

In [41]:
```python
from sklearn.metrics import roc_auc_score,roc_curve
```

In [42]:
```python
def draw_roc(actual,predicted):
    fpr,tpr,threshold=roc_curve(actual,predicted)
    auc_score=roc_auc_score(actual,predicted)
    plt.figure(figsize=(6,4))
    plt.plot(fpr,tpr,label='roc_curve(area=%0.2f)'%auc_score)
    plt.plot([0,1],[0,1],'k--')
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.05])
    plt.title('Receiver operating chatecteristic curve')
    plt.xlabel('False positive rate or [1- True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.show()
    return fpr,tpr,threshold
```

In [43]:
```python
a,b,c= draw_roc(y_test,y_predict)
plt.show()
```



Down-Sampling.

In [44]:
```python
non_diab_indices = data[data['class']==0].index
#get the record numbers of non-diab cases

no_diab = len(data[data['class']==0])
#how many non_diab cases
print(no_diab)

diab_indices=data[data['class']==1].index
#record number of the diabetics cases

diab = len(data[data['class']==1])
#how many diabetic cases
print(diab)
```

```
500
268
```

In [45]:
```python
random_indices = np.random.choice(non_diab_indices,no_diab-200,replace=False)
#randomly pick up 300 non diab indice
```

In [46]:
```python
down_sample_indices = np.concatenate([diab_indices,random_indices])
#combining diab and non diab (after doing dampling)
```

In [47]:
```python
pima_df_down_sample = data.loc[down_sample_indices]
#extract all those
pima_df_down_sample.shape
pima_df_down_sample.groupby(['class']).count()
```

Out[47]:

	Preg	Plas	Pres	skin	test	mass	pedi	age
class								
0	300	300	300	300	300	300	300	300
1	268	268	268	268	268	268	268	268

In [49]:
```python
pima_df_down_sample.head()
```

Out[49]:

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
2	8	183	64	0	0	23.3	0.672	32	1
4	0	137	40	35	168	43.1	2.288	33	1
6	3	78	50	32	88	31.0	0.248	26	1
8	2	197	70	45	543	30.5	0.158	53	1

In [55]:
```python
x = pima_df_down_sample.drop(columns=['class']).to_numpy()
y = pima_df_down_sample['class'].to_numpy()
```

In [56]:
```python
x_train1,x_test1,y_train1,y_test1 = train_test_split(x,y,random_state=0,train_size=0.70)
```

In [57]:
```python
x_train1.shape,x_test1.shape
```

Out[57]: ((397, 8), (171, 8))

In [58]:
```python
model = LogisticRegression()
model.fit(x_train1,y_train1)
```

Out[58]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [59]:
```python
y_predi = model.predict(x_test)
```

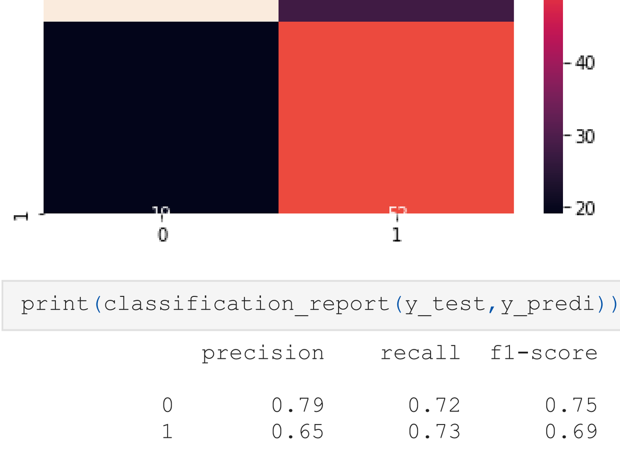
In [61]:
```python
print('Accuarcy :-',accuracy_score(y_test,y_predi))
```

```
Accuarcy :- 0.7251461988304093
```

In [63]:
```python
matrix = confusion_matrix(y_test,y_predi)
matrix
```

Out[63]:
```
array([[72, 28],
       [19, 52]], dtype=int64)
```

In [65]:
```python
sns.heatmap(matrix,annot=True)
plt.show()
```



In [67]:
```python
print(classification_report(y_test,y_predi))
```

```
              precision    recall  f1-score   support

           0       0.79      0.72      0.75       100
           1       0.65      0.73      0.69        71

    accuracy                           0.73       171
   macro avg       0.72      0.73      0.72       171
weighted avg       0.73      0.73      0.73       171
```

Always recommend to use the upsampling because it will loss the data.

Logistcs Regression

- Logistics Regression is Supervised machine learning algorthm use for the prediction of classes or catgorical entity.It mainly use for the Decision making purposes.
- Logistics Regression is Regression because in that we have to predict the dependent variable which is in the form of classes i.e yes,no or 0 or 1 with the help of the independednt variable.
- It works like a classification because it handle the classes in the dependent variable.
- Logistics helps to take decision on the basis of the Probablitry.
- It uses the Sigmoid Function which is helps to maintaiin probablity inbetween 0 and 1.In that it will gives the probablity inbetween 0 and 1 in respect of the input variable which is form of categories on and the independent variable.It means it helps to map the probablities inbetween the 0 and 1 according the independent features.
- Sigmoid funtion uses the Threshold which help logistics to take decision suppose we having the threshold =0.5 and our output probablity is 0.9 then it will map it to 1 and shows that this case belongs to 1.
- Similarly if we got the 0.4 then it map to 0 and suggest us to maintain that case as 0.
    - **Simoid function helps to map the Prediction into probablity inbetween 0 and 1.**

Apply Logistics Regression

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

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
data = pd.read_csv('bank-additional-full.csv',sep=';')
```

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

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0	nonexistent
2	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0	nonexistent
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0	nonexistent
4	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0	nonexistent

5 rows × 21 columns

In [4]:
```
data.columns
```

Out[4]:
```
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y'],
      dtype='object')
```

In [5]:
```
data.shape
```

Out[5]:
```
(41188, 21)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
age               41188 non-null int64
job               41188 non-null object
marital           41188 non-null object
education         41188 non-null object
default           41188 non-null object
housing           41188 non-null object
loan              41188 non-null object
contact           41188 non-null object
month             41188 non-null object
day_of_week       41188 non-null object
duration          41188 non-null int64
campaign          41188 non-null int64
pdays             41188 non-null int64
previous          41188 non-null int64
poutcome          41188 non-null object
emp.var.rate      41188 non-null float64
cons.price.idx    41188 non-null float64
cons.conf.idx     41188 non-null float64
euribor3m         41188 non-null float64
nr.employed       41188 non-null float64
y                 41188 non-null object
dtypes: float64(5), int64(5), object(11)
memory usage: 4.9+ MB
```

In [7]:
```
data.describe()
```

Out[7]:

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.emplo
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664	-40.502600	3.621291	5167.035
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840	4.628198	1.734447	72.251
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000	-50.800000	0.634000	4963.600
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000	-42.700000	1.344000	5099.100
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000	-41.800000	4.857000	5191.000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000	-36.400000	4.961000	5228.100
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000	-26.900000	5.045000	5228.100

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

In [9]:
```
dataframe = pd.DataFrame({'Dtye':data.dtypes,'Unique':data.nunique(),'Duplicated':data.duplicated().sum()})
```

In [10]:
```
dataframe
```

Out[10]:

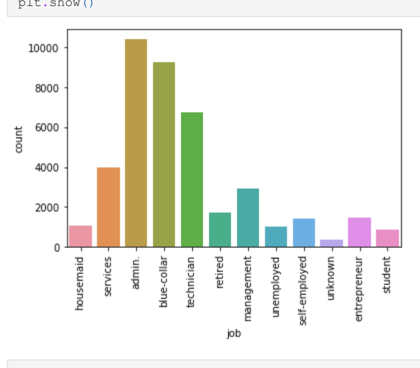
	Dtye	Unique	Duplicated
age	int64	78	12
job	object	12	12
marital	object	4	12
education	object	8	12
default	object	3	12
housing	object	3	12
loan	object	3	12
contact	object	2	12
month	object	10	12
day_of_week	object	5	12
duration	int64	1544	12
campaign	int64	42	12
pdays	int64	27	12
previous	int64	8	12
poutcome	object	3	12
emp.var.rate	float64	10	12
cons.price.idx	float64	26	12
cons.conf.idx	float64	26	12
euribor3m	float64	316	12
nr.employed	float64	11	12
y	object	2	12

EDA

In [11]:
```
data['job'].value_counts().sort_values(ascending=False).head()
```

Out[11]:
```
admin.         10422
blue-collar     9254
technician      6743
services        3969
management      2924
Name: job, dtype: int64
```

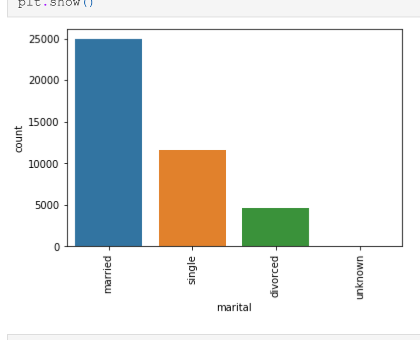
In [12]:
```
sns.countplot(data['job'])
plt.xticks(rotation=90)
plt.show()
```



In [13]:
```
data['marital'].value_counts()
```

Out[13]:
```
married     24928
single      11568
divorced     4612
unknown        80
Name: marital, dtype: int64
```

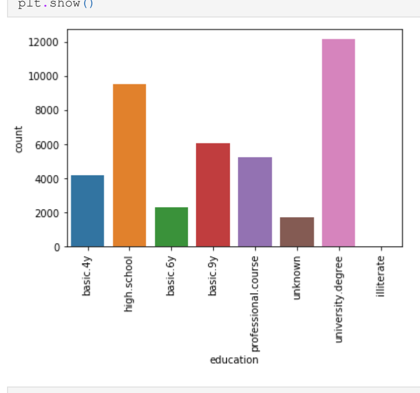
In [14]:
```
sns.countplot(data['marital'])
plt.xticks(rotation=90)
plt.show()
```



In [15]:
```
data['education'].value_counts()
```

Out[15]:
```
university.degree      12168
high.school             9515
basic.9y                6045
professional.course     5243
basic.4y                4176
basic.6y                2292
unknown                 1731
illiterate                18
Name: education, dtype: int64
```

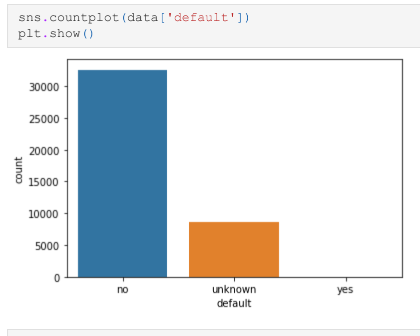
In [16]:
```
sns.countplot(data['education'])
plt.xticks(rotation=90)
plt.show()
```



In [17]:
```
data['default'].value_counts()
```

Out[17]:
```
no         32588
unknown     8597
yes            3
Name: default, dtype: int64
```

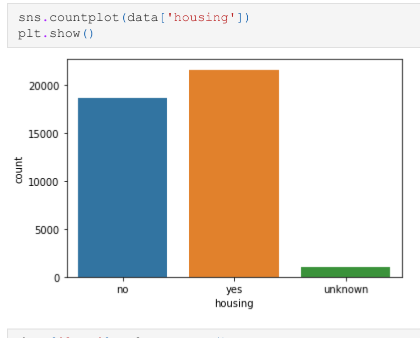
In [18]:
```
sns.countplot(data['default'])
plt.show()
```



In [19]:
```
data['housing'].value_counts()
```

Out[19]:
```
yes        21576
no         18622
unknown      990
Name: housing, dtype: int64
```

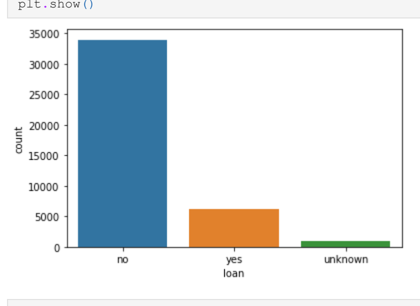
In [20]:
```
sns.countplot(data['housing'])
plt.show()
```



In [21]:
```
data['loan'].value_counts()
```

Out[21]:
```
no         33950
yes         6248
unknown      990
Name: loan, dtype: int64
```

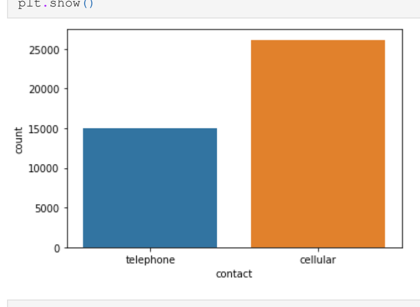
In [22]:
```
sns.countplot(data['loan'])
plt.show()
```



In [23]:
```
data['contact'].value_counts()
```

Out[23]:
```
cellular     26144
telephone    15044
Name: contact, dtype: int64
```

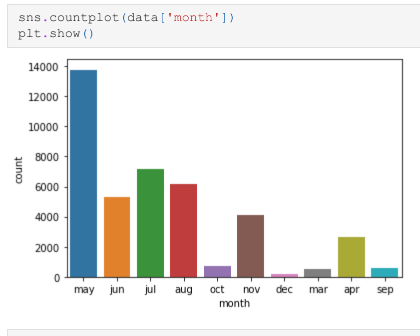
In [24]:
```
sns.countplot(data['contact'])
plt.show()
```



In [25]:
```
data['month'].value_counts()
```

Out[25]:
```
may    13769
jul     7174
aug     6178
jun     5318
nov     4101
apr     2632
oct      718
sep      570
mar      546
dec      182
Name: month, dtype: int64
```

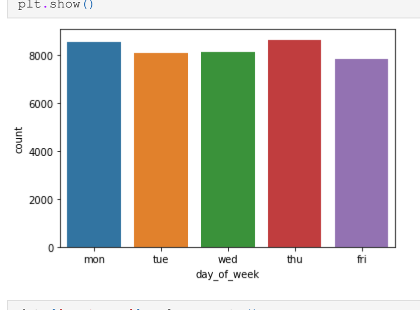
In [26]:
```
sns.countplot(data['month'])
plt.show()
```



In [27]:
```
data['day_of_week'].value_counts()
```

Out[27]:
```
thu    8623
mon    8514
wed    8134
tue    8090
fri    7827
Name: day_of_week, dtype: int64
```

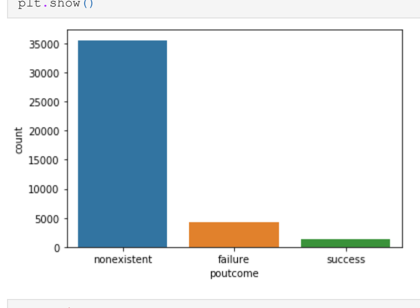
In [28]:
```
sns.countplot(data['day_of_week'])
plt.show()
```



In [29]:
```
data['poutcome'].value_counts()
```

Out[29]:
```
nonexistent    35563
failure         4252
success         1373
Name: poutcome, dtype: int64
```

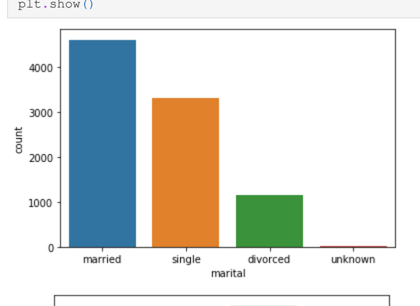
In [30]:
```
sns.countplot(data['poutcome'])
plt.show()
```

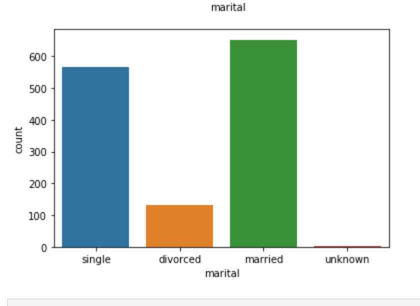


In [31]:
```
data['job'].value_counts()
```

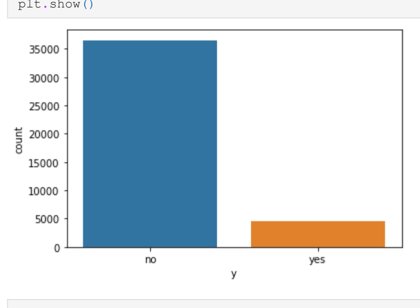
Out[31]:
```
admin.           10422
blue-collar       9254
technician        6743
services          3969
management        2924
retired           1720
entrepreneur      1456
self-employed     1421
housemaid         1060
unemployed        1014
student            875
unknown            330
Name: job, dtype: int64
```

In [32]:
```
sns.countplot(data[((data['job']=='admin.') & (data['y']=='no'))]['marital'])
plt.show()
sns.countplot(data[((data['job']=='admin.') & (data['y']=='yes'))]['marital'])
plt.show()
```

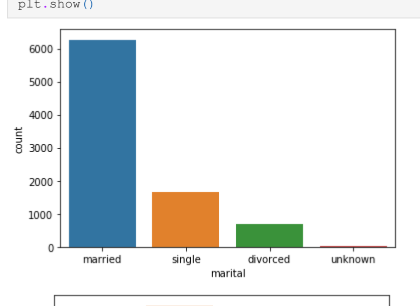


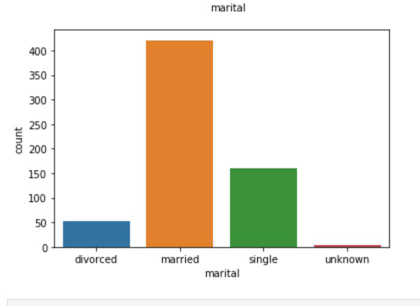


In [33]:
```
sns.countplot(data['y'])
plt.show()
```

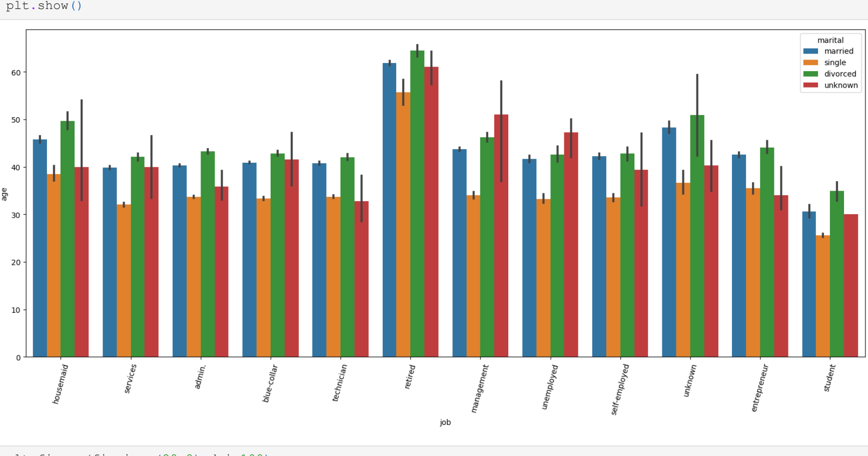


In [34]:
```
sns.countplot(data[((data['job']=='blue-collar') & (data['y']=='no'))]['marital'])
plt.show()
sns.countplot(data[((data['job']=='blue-collar') & (data['y']=='yes'))]['marital'])
plt.show()
```

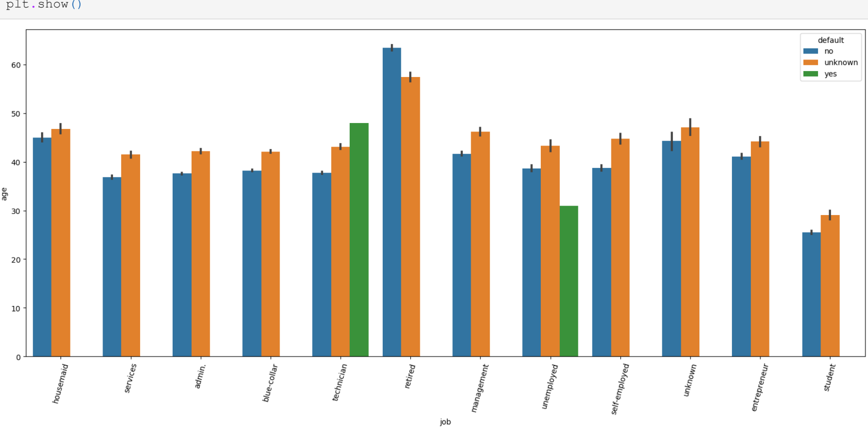




In [35]:
```
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['job'],y=data['age'],hue=data['marital'])
plt.xticks(rotation=75)
plt.show()
```

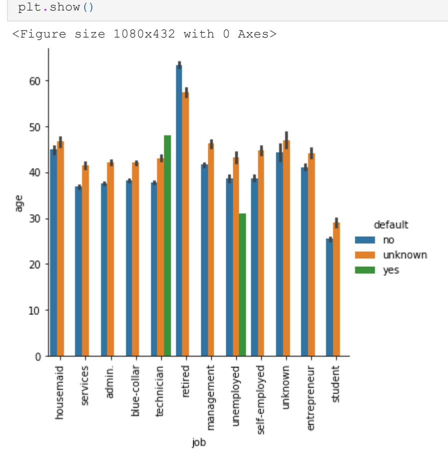


In [36]:
```
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['job'],y=data['age'],hue=data['default'])
plt.xticks(rotation=75)
plt.show()
```

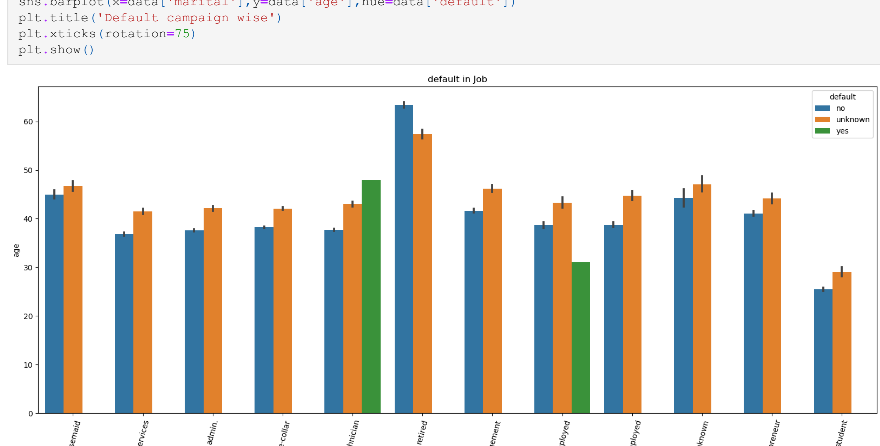


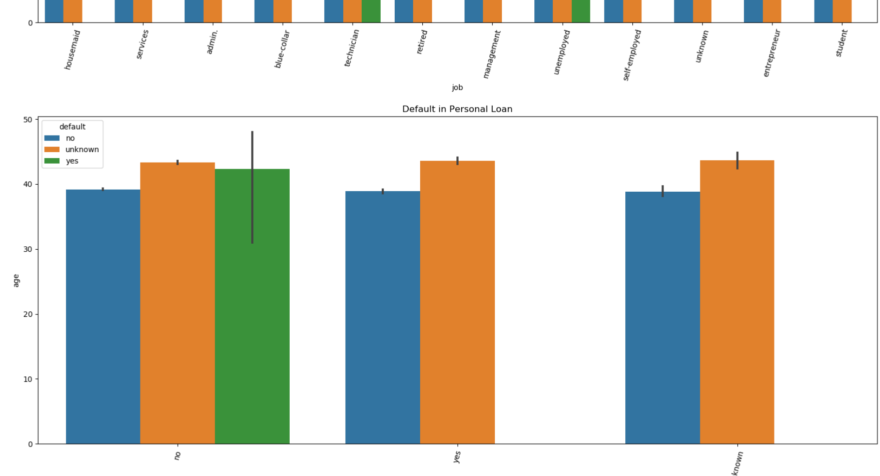
In [37]:
```
plt.figure(figsize=(15,6))
sns.catplot(x='job',y='age',kind='bar',hue='default',data=data)
plt.xticks(rotation=90)
plt.show()
```

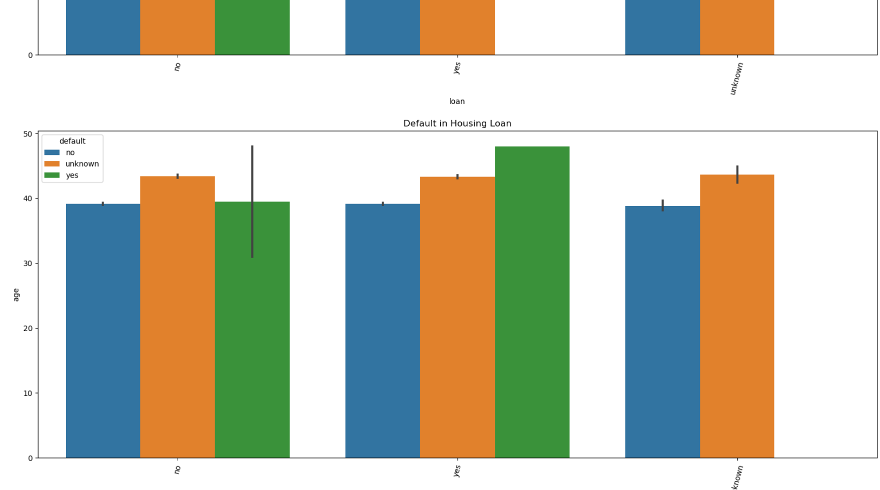
```
<Figure size 1080x432 with 0 Axes>
```

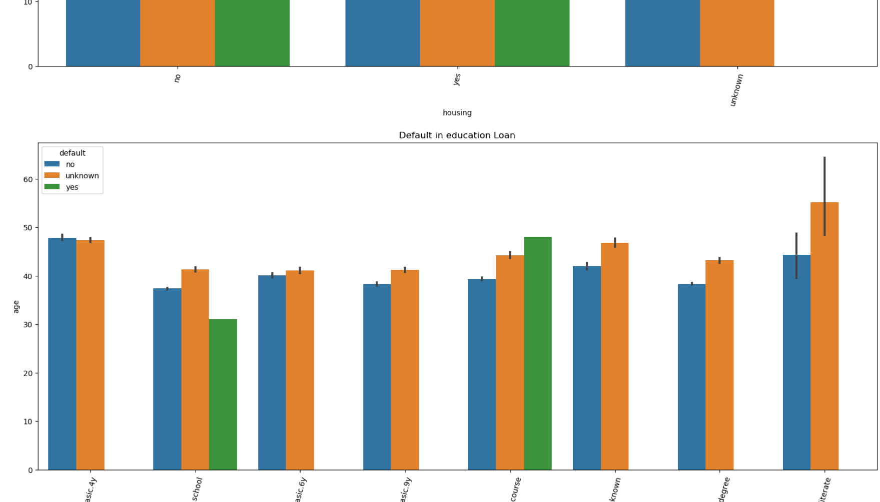


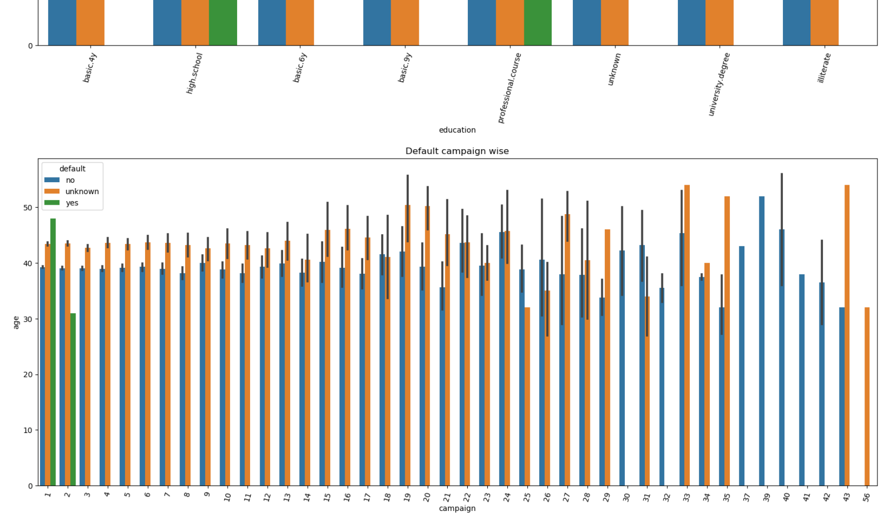
In [38]:
```
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['job'],y=data['age'],hue=data['default'])
plt.title('default in Job')
plt.xticks(rotation=75)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['loan'],y=data['age'],hue=data['default'])
plt.title('Default in Personal Loan')
plt.xticks(rotation=75)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['housing'],y=data['age'],hue=data['default'])
plt.title('Default in Housing Loan')
plt.xticks(rotation=75)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['education'],y=data['age'],hue=data['default'])
plt.title('Default in education Loan')
plt.xticks(rotation=75)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['campaign'],y=data['age'],hue=data['default'])
plt.title('Default campaign wise')
plt.xticks(rotation=75)
plt.show()
plt.figure(figsize=(20,8),dpi=100)
sns.barplot(x=data['marital'],y=data['age'],hue=data['default'])
plt.title('Default campaign wise')
plt.xticks(rotation=75)
plt.show()
```

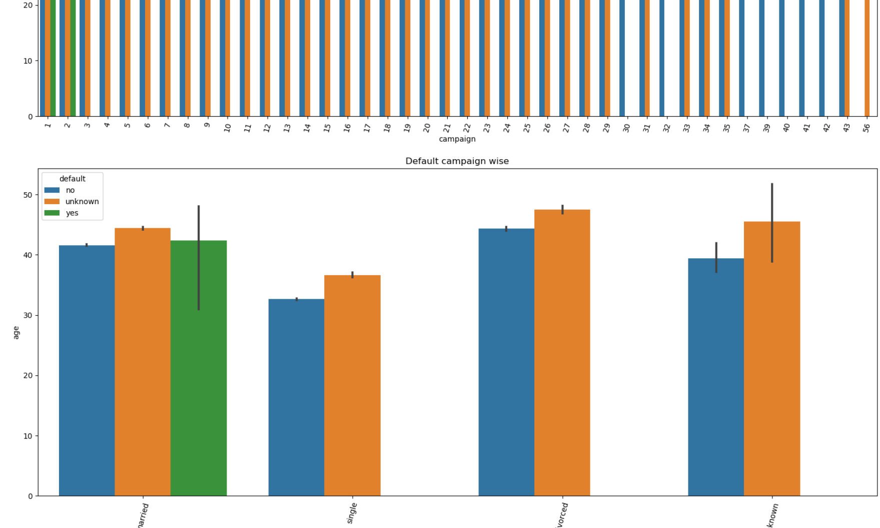




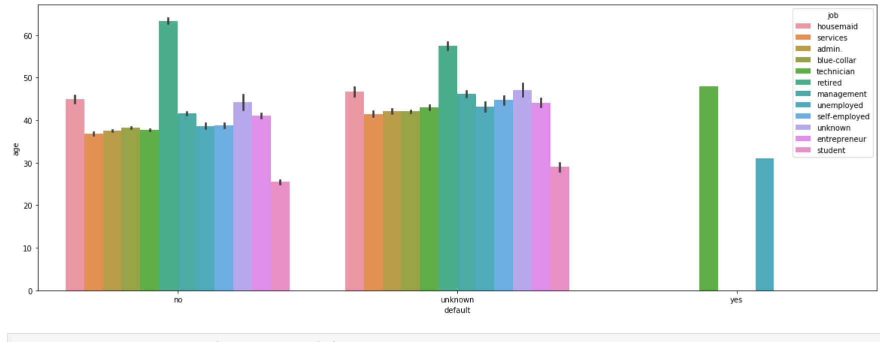




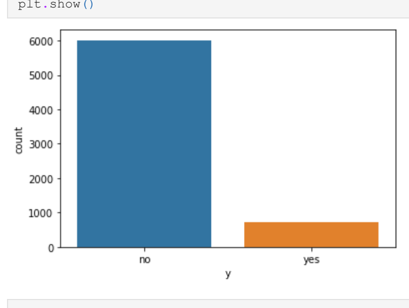




In [39]:
```
plt.figure(figsize=(20,7))
sns.barplot(data['default'],data['age'],hue=data['job'])
plt.show()
```



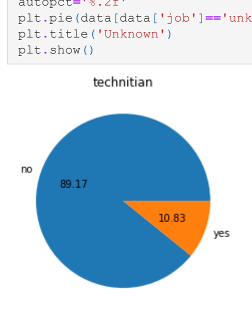
```
In [40]: sns.countplot(data[data['job']=='technician']['y'])
         plt.show()
```

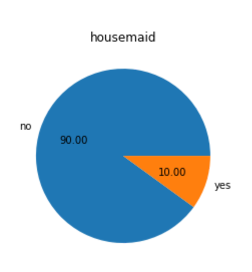


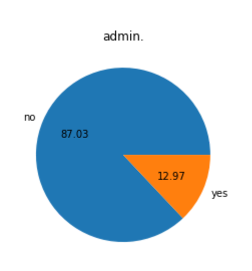
```
In [41]: data[data['job']=='technician']['y'].value_counts()
```

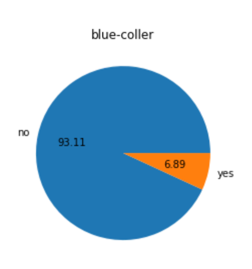
```
Out[41]: no     6013
         yes     730
         Name: y, dtype: int64
```

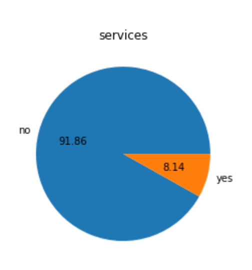
```
In [42]: label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='technician']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('technitian')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='housemaid']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('housemaid')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='admin.']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('admin.')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='blue-collar']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('blue-coller')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='services']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('services')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='management']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('management')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='retired']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Retired')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='entrepreneur']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Entrepreneur')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='self-employed']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Self-employed')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='unemployed']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Un-employed')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='student']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Student')
         plt.show()
         label= ['no','yes']
         autopct='%.2f'
         plt.pie(data[data['job']=='unknown']['y'].value_counts(),labels=label,autopct=autopct)
         plt.title('Unknown')
         plt.show()
```

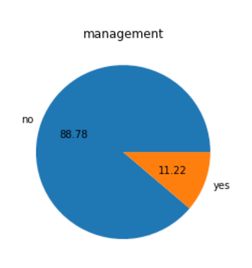


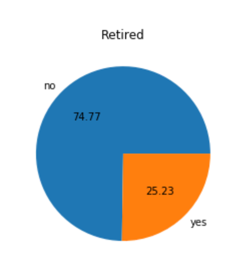


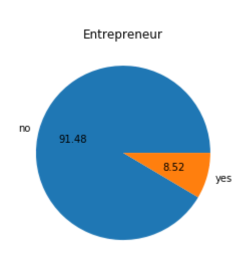


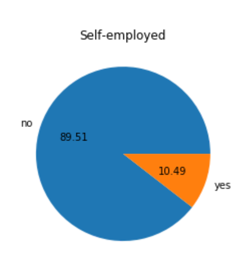


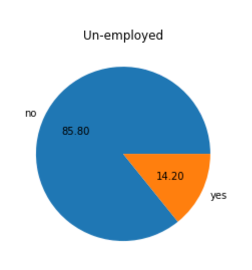


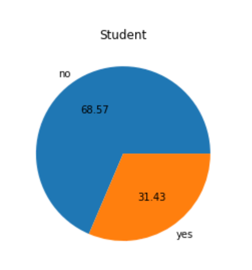


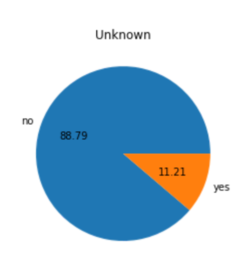




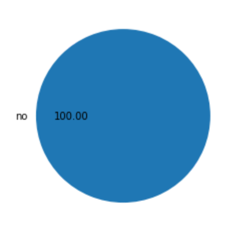




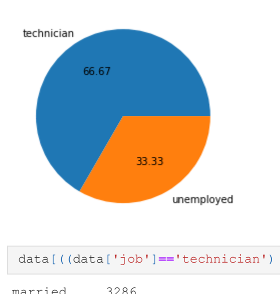





```
In [43]: label= ['no']
         autopct='%.2f'
         plt.pie(data[data['default']=='yes']['y'].value_counts(),labels=label,autopct=autopct)
         plt.show()
```



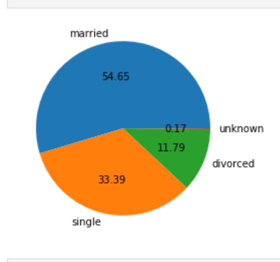

```
In [44]: label= ['technician','unemployed']
         autopct='%.2f'
         plt.pie(data[data['default']=='yes']['job'].value_counts(),labels=label,autopct=autopct)
         plt.show()
```



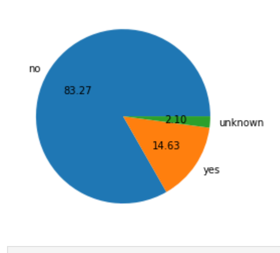

```
In [45]: data[((data['job']=='technician') & (data['y']=='no'))]['marital'].value_counts()
```

```
Out[45]: married     3286
         single      2008
         divorced     709
         unknown       10
         Name: marital, dtype: int64
```

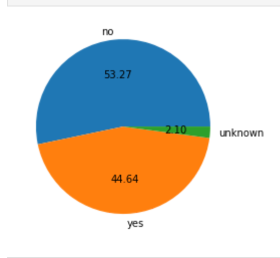
```
In [46]: label= ['married','single','divorced','unknown']
         autopct='%.2f'
         plt.pie(data[((data['job']=='technician') & (data['y']=='no'))]['marital'].value_counts(),labels=label,autopct=
         plt.show()
```




```
In [47]: label= ['no','yes','unknown']
         autopct='%.2f'
         plt.pie(data[((data['job']=='technician') & (data['y']=='no'))]['loan'].value_counts(),labels=label,autopct=aut
         plt.show()
```




```
In [48]: label= ['no','yes','unknown']
         autopct='%.2f'
         plt.pie(data[((data['job']=='technician') & (data['y']=='no'))]['housing'].value_counts(),labels=label,autopct=
         plt.show()
```

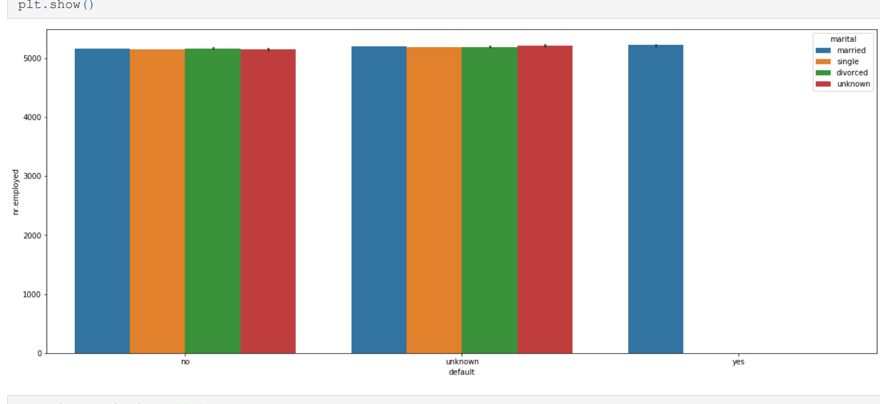



```
In [49]: dataframe
```

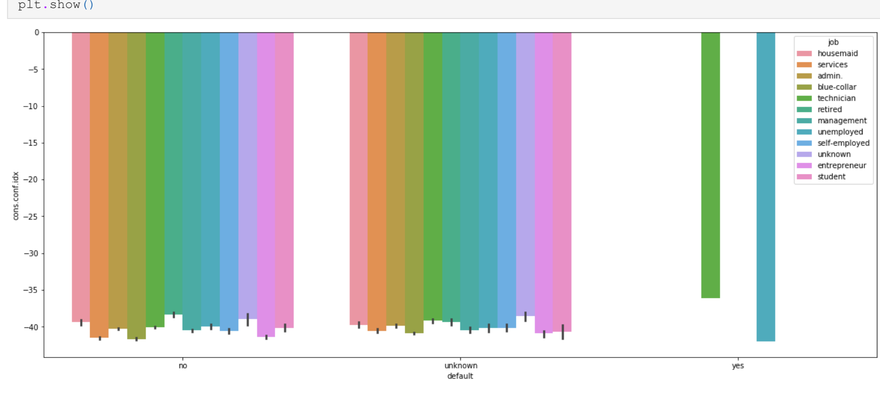
Out[49]:

	Dtye	Unique	Duplicated
age	int64	78	12
job	object	12	12
marital	object	4	12
education	object	8	12
default	object	3	12
housing	object	3	12
loan	object	3	12
contact	object	2	12
month	object	10	12
day_of_week	object	5	12
duration	int64	1544	12
campaign	int64	42	12
pdays	int64	27	12
previous	int64	8	12
poutcome	object	3	12
emp.var.rate	float64	10	12
cons.price.idx	float64	26	12
cons.conf.idx	float64	26	12
euribor3m	float64	316	12
nr.employed	float64	11	12
y	object	2	12

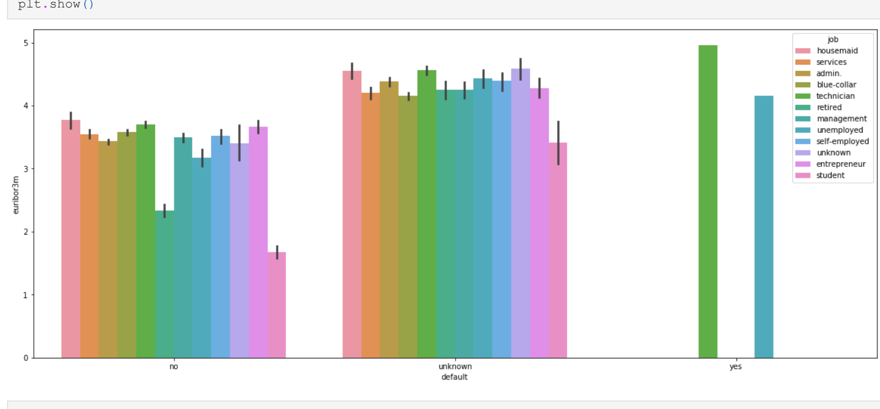
```
In [50]: plt.figure(figsize=(20,8))
         sns.barplot(y=data['nr.employed'],x =data['default'],hue=data['marital'])
         plt.show()
```



```
In [51]: plt.figure(figsize=(20,8))
         sns.barplot(y=data['cons.conf.idx'],x =data['default'],hue=data['job'])
         plt.show()
```

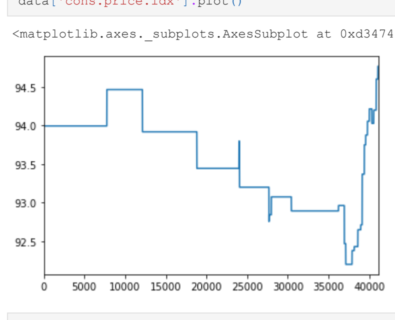


```
In [52]: plt.figure(figsize=(20,8))
         sns.barplot(y=data['euribor3m'],x =data['default'],hue=data['job'])
         plt.show()
```

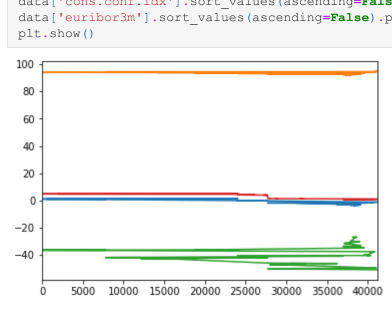


```
In [53]: data['cons.price.idx'].plot()
```

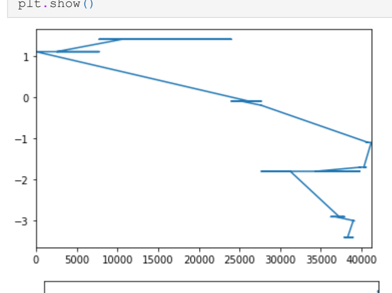
```
Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0xd347430>
```

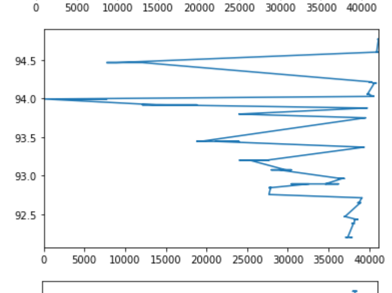


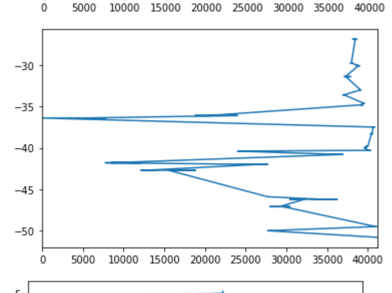
```
In [54]: data['emp.var.rate'].sort_values(ascending=False).plot()
         data['cons.price.idx'].sort_values(ascending=False).plot()
         data['cons.conf.idx'].sort_values(ascending=False).plot()
         data['euribor3m'].sort_values(ascending=False).plot()
         plt.show()
```

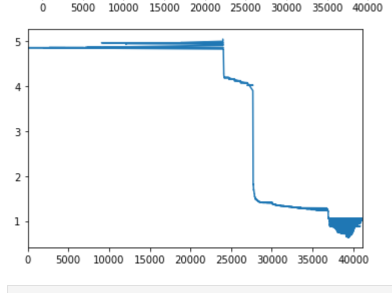


```
In [55]: data['emp.var.rate'].sort_values(ascending=False).plot()
         plt.show()
         data['cons.price.idx'].sort_values(ascending=False).plot()
         plt.show()
         data['cons.conf.idx'].sort_values(ascending=False).plot()
         plt.show()
         data['euribor3m'].sort_values(ascending=False).plot()
         plt.show()
```



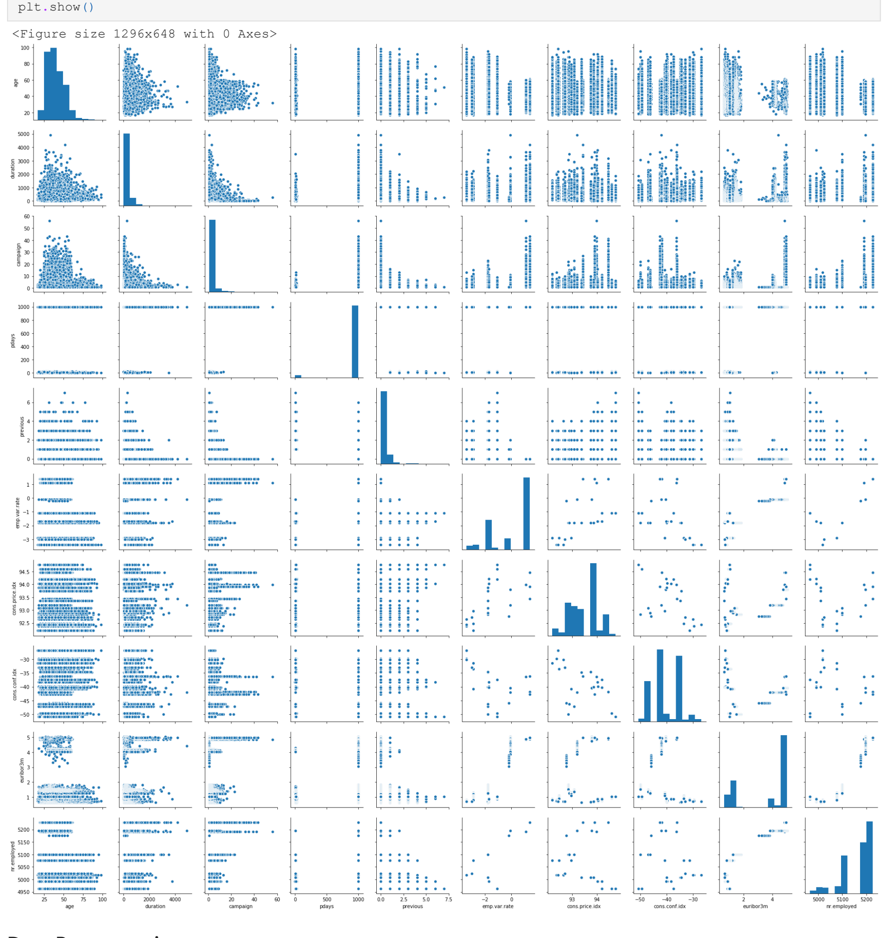






```
In [56]: plt.figure(figsize=(18,9))
         sns.pairplot(data)
         plt.show()
```

```
<Figure size 1296x648 with 0 Axes>
```



Data Preprocessing.

```
In [57]: data.iloc[:,10:]
```

Out[57]:

	duration	campaign	pdays	previous	poutcome	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	261	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
1	149	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
2	226	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
3	151	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
4	307	1	999	0	nonexistent	1.1	93.994	-36.4	4.857	5191.0	no
...	...	...	...	...	...	...	...	...	...	...	...
41183	334	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	yes
41184	383	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	no
41185	189	2	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	no
41186	442	1	999	0	nonexistent	-1.1	94.767	-50.8	1.028	4963.6	yes
41187	239	3	999	1	failure	-1.1	94.767	-50.8	1.028	4963.6	no

41188 rows × 11 columns

```
In [58]: hi = {'no':0,'yes':1}
         data['y']= data['y'].map(hi)
```

```
In [59]: df = pd.get_dummies(data['marital'],drop_first=True)
         dff = pd.get_dummies(data['default'],drop_first=True)
         dfff = pd.get_dummies(data['housing'],drop_first=True)
         dffff = pd.get_dummies(data['loan'],drop_first=True)
         df1 = pd.get_dummies(data['poutcome'],drop_first=True)
```

```
In [60]: data1 = pd.concat([df,dff,dfff,dffff,df1],axis=1)
```

```
In [61]: data.set_index(data['month'])
```

Out[61]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous
month														
may	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	...	1	999	0
may	57	services	married	high.school	unknown	no	no	telephone	may	mon	...	1	999	0
may	37	services	married	high.school	no	yes	no	telephone	may	mon	...	1	999	0
may	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	...	1	999	0
may	56	services	married	high.school	no	no	yes	telephone	may	mon	...	1	999	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
nov	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	1	999	0
nov	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0
nov	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	...	2	999	0
nov	44	technician	married	professional.course	no	no	no	cellular	nov	fri	...	1	999	0
nov	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	...	3	999	1

41188 rows × 21 columns

```
In [62]: data.drop(columns=['marital','education','job','default','loan','contact','day_of_week','poutcome'],inplace=Tru
```

```
In [63]: data.drop(columns=['housing'],inplace=True)
```

```
In [64]: df = pd.concat([data1,data],axis=1)
```

```
In [65]: df.iloc[:,10:]
```

Out[65]:

	success	age	month	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	y
0	0	56	may	261	1	999	0	1.1	93.994	-36.4	4.857	5191.0	0
1	0	57	may	149	1	999	0	1.1	93.994	-36.4	4.857	5191.0	0
2	0	37	may	226	1	999	0	1.1	93.994	-36.4	4.857	5191.0	0
3	0	40	may	151	1	999	0	1.1	93.994	-36.4	4.857	5191.0	0
4	0	56	may	307	1	999	0	1.1	93.994	-36.4	4.857	5191.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
41183	0	73	nov	334	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1
41184	0	46	nov	383	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	0
41185	0	56	nov	189	2	999	0	-1.1	94.767	-50.8	1.028	4963.6	0
41186	0	44	nov	442	1	999	0	-1.1	94.767	-50.8	1.028	4963.6	1
41187	0	74	nov	239	3	999	1	-1.1	94.767	-50.8	1.028	4963.6	0

41188 rows × 13 columns

```
In [66]: df.drop(columns=['month'],inplace=True)
```

```
In [67]: x = df.drop(columns=['y']).to_numpy()
         y = df['y'].to_numpy()
```

```
In [68]: x.shape,y.shape
```

```
Out[68]: ((41188, 21), (41188,))
```

```
In [69]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
```

```
In [70]: x_train,x_test,y_train,y_test = train_test_split(x,y,random_state =0,stratify=y,train_size=0.80)
```

```
In [71]: x_train.shape,y_train.shape
```

```
Out[71]: ((32950, 21), (32950,))
```

```
In [72]: print(len(y_train[y_train==1]))
         print(len(y_train[y_train==0]))
```

```
3712
29238
```

```
In [73]: from imblearn.over_sampling import SMOTE
```

```
Using TensorFlow backend.
```

```
In [74]: sm = SMOTE(sampling_strategy=1,random_state=0,k_neighbors=5,n_jobs=-1)
```

```
In [75]: x_train_res,y_train_res = sm.fit_sample(x_train,y_train.ravel())
```

```
In [76]: x_train_res.shape,y_train_res.shape
```

```
Out[76]: ((58476, 21), (58476,))
```

```
In [77]: print(len(y_train_res[y_train_res==1]))
         print(len(y_train_res[y_train_res==0]))
```

```
29238
29238
```

```
In [78]: from sklearn.preprocessing import StandardScaler
```

```
In [79]: scaler = StandardScaler()
         x_train_scalled=scaler.fit_transform(x_train_res)
         x_test_scalled =scaler.transform(x_test)
```

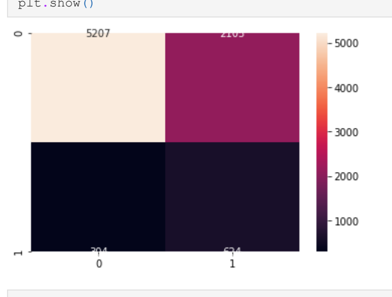
```
In [80]: model = LogisticRegression()
         model.fit(x_train_res,y_train_res)
         y_pred = model.predict(x_test_scalled)
         print('Accuarcy :',accuracy_score(y_test,y_pred))
```

```
Accuarcy : 0.7078174314153921
```

```
In [81]: mat= confusion_matrix(y_test,y_pred)
         mat
```

```
Out[81]: array([[5207, 2103],
                [ 304,  624]], dtype=int64)
```

```
In [82]: sns.heatmap(mat,annot=True,fmt='d')
         plt.show()
```



```
In [83]: print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.94      0.71      0.81      7310
           1       0.23      0.67      0.34       928

    accuracy                           0.71      8238
   macro avg       0.59      0.69      0.58      8238
weighted avg       0.86      0.71      0.76      8238
```

Feature Selection By the Correlation.

In [84]:
```
corr = np.abs(df.corr()['y'])
corr
```

Out[84]:
```
married          0.043398
single           0.054133
unknown          0.005211
unknown          0.099293
yes              0.003041
unknown          0.002270
yes              0.011743
unknown          0.002270
yes              0.004466
nonexistent      0.193507
success          0.316269
age              0.030399
duration         0.405274
campaign         0.066357
pdays            0.324914
previous         0.230181
emp.var.rate     0.298334
cons.price.idx   0.136211
cons.conf.idx    0.054878
euribor3m        0.307771
nr.employed      0.354678
y                1.000000
Name: y, dtype: float64
```

In [85]:
```
good_corr = corr[corr>0.3]
good_corr.drop(columns=['y'],inplace=True)
```

In [86]:
```
corrdata = df[good_corr.index]
```

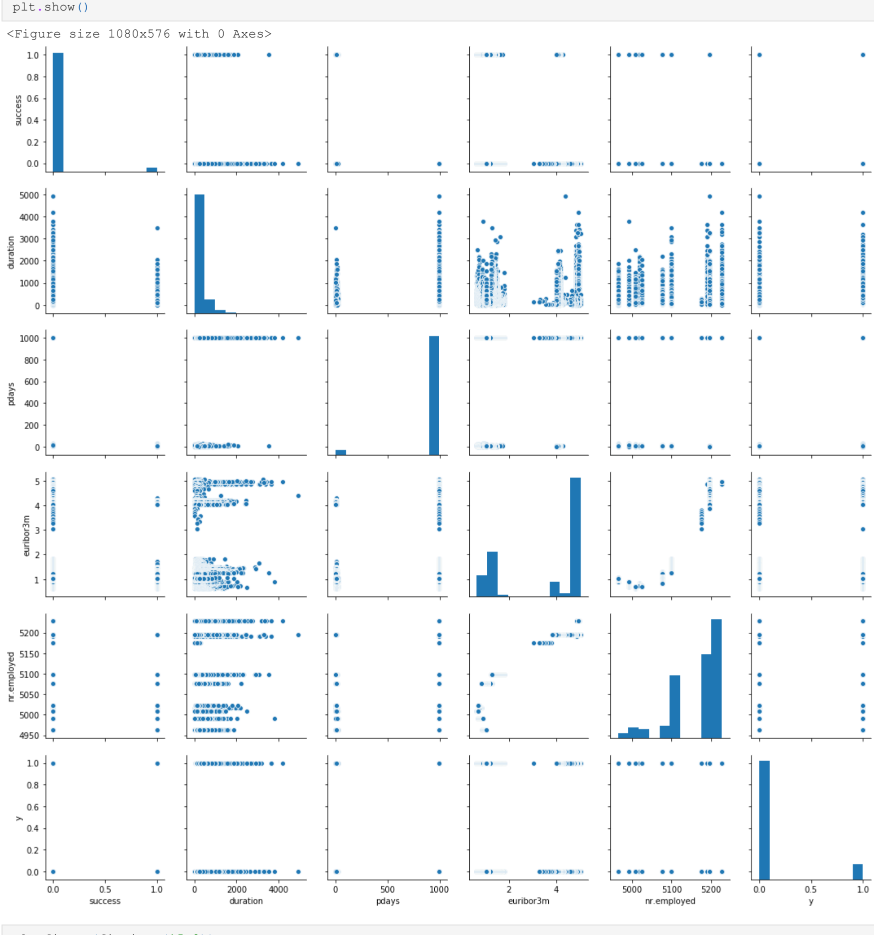
In [87]:
```
corrdata.head()
```

Out[87]:

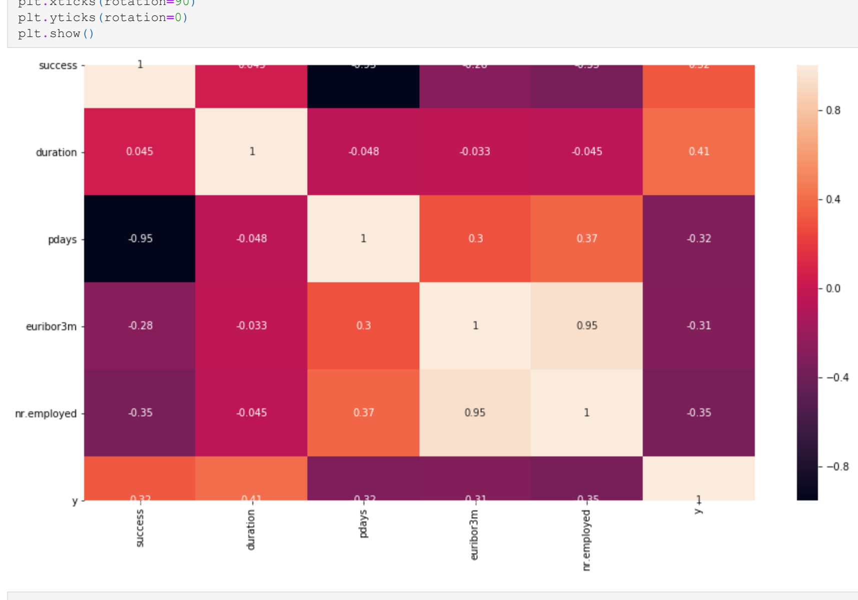
	success	duration	pdays	euribor3m	nr.employed	y
0	0	261	999	4.857	5191.0	0
1	0	149	999	4.857	5191.0	0
2	0	226	999	4.857	5191.0	0
3	0	151	999	4.857	5191.0	0
4	0	307	999	4.857	5191.0	0

In [88]:
```
plt.figure(figsize=(15,8))
sns.pairplot(corrdata)
plt.show()
```

```
<Figure size 1080x576 with 0 Axes>
```



In [89]:
```
plt.figure(figsize=(15,8))
sns.heatmap(corrdata.corr(),annot=True)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.show()
```



In [90]:
```
x = corrdata.drop(columns=['y'])
y = corrdata['y']
```

In [91]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.80,stratify=y,random_state=42)
```

In [92]:
```
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

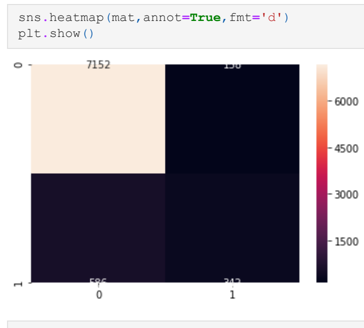
In [93]:
```
model = LogisticRegression()
model.fit(x_train,y_train)
y_pred = model.predict(x_test)
print('Accuracy :',accuracy_score(y_test,y_pred))
```

```
Accuracy : 0.909686817188638
```

In [94]:
```
mat = confusion_matrix(y_test,y_pred)
mat
```

Out[94]:
```
array([[7152,  158],
       [ 586,  342]], dtype=int64)
```

In [95]:
```
sns.heatmap(mat,annot=True,fmt='d')
plt.show()
```



In [96]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      7310
           1       0.68      0.37      0.48       928

    accuracy                           0.91      8238
   macro avg       0.80      0.67      0.71      8238
weighted avg       0.90      0.91      0.90      8238
```

In [97]:
```
data1 = pd.DataFrame({'Actaul':y_test,'Predicted':y_pred})
data1
```

Out[97]:

	Actaul	Predicted
14455	0	0
36380	0	0
40076	0	0
10778	0	0
27939	0	0
...	...	...
33359	1	1
34998	0	0
39861	0	1
3920	0	0
33395	0	0

8238 rows × 2 columns

In [ ]: