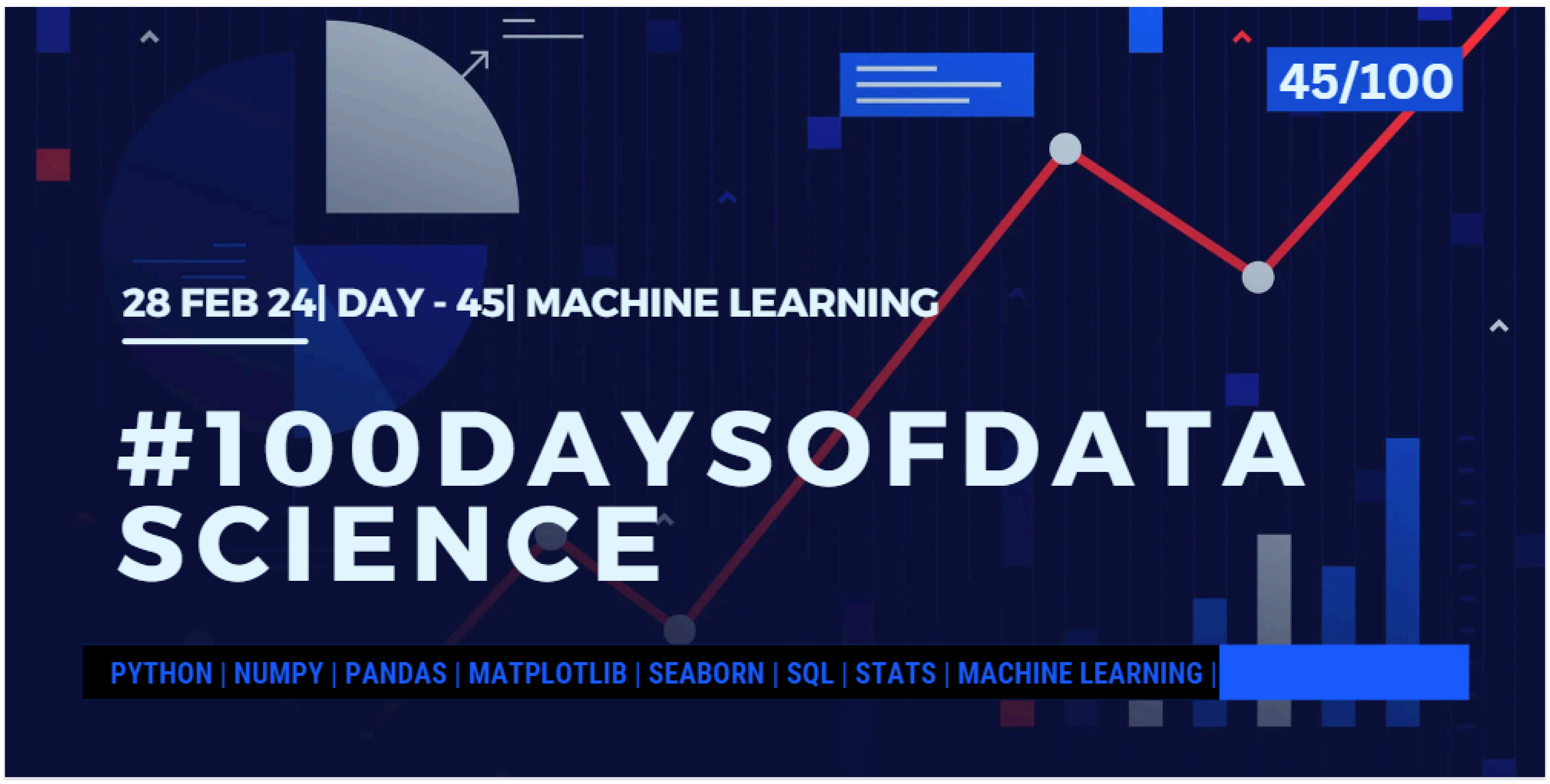
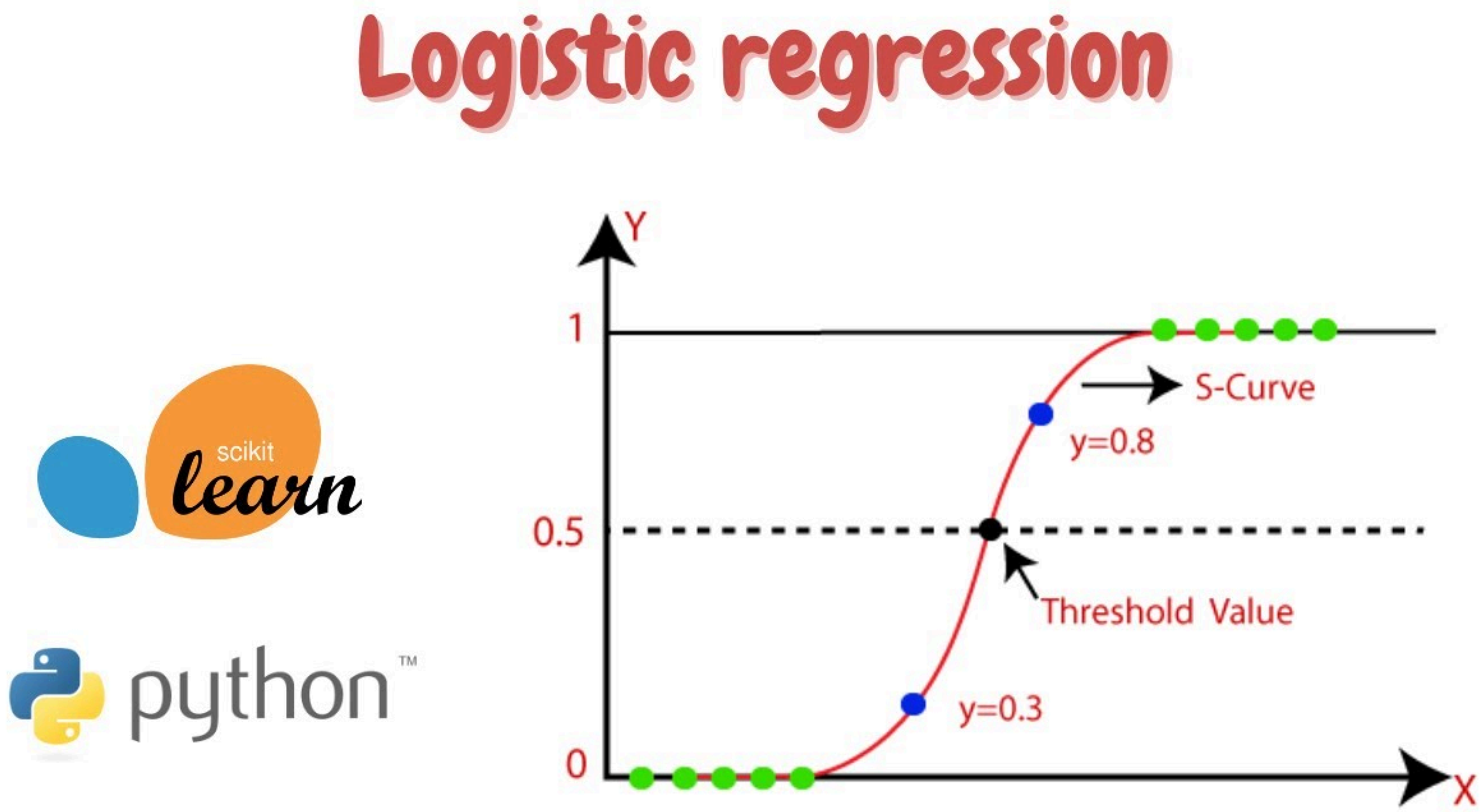


Logistic Regression



- Logistic regression is a supervised machine learning algorithm in which we have labelled data x and y are given mostly it is used for classification problem that to binary classification.
- Practically, it is used to classify observations into different categories. Hence, its output is discrete in nature.
- Logistic Regression is also called Logit Regression. It is based on probability metric.
- It is one of the most simple, straightforward and versatile classification algorithms which is used to solve classification problems.
- Logistic regression allows us to implement regression to find out or to measure the relationship between the independent and dependent variable.
- It determines the probability of observations to belong to either of the two classes internally it used sigmoid function where we substitute the regression equation and reach to logit function(logistics regression equation) by taking the the log of outs ratio the algorithm gives regression equation which predicts the value of an observation to belong to class 0 and class 1.

- **Sigmoid**
  This predicted response value, denoted by z is then converted into a probability value that lie between 0 and 1. We use the sigmoid function in order to map predicted values to probability values. This sigmoid function then maps any real value into a probability value between 0 and 1. The sigmoid function has an S shaped curve. It is also called sigmoid curve.

- **Outs ratio**
  Probability of success upon probability of failure

- **Logit regression equation**
  $\log(p/1-p) = \beta_0 + \beta_1 x_1$

- **Decision boundary**
  This probability value is then mapped to a discrete class which is either “0” or “1”. In order to map this probability value to a discrete class (pass/fail, yes/no, true/false), we select a threshold value. This threshold value is called Decision boundary. Above this threshold value, we will map the probability values into class 1 and below which we will map values into class 0.

Mathematically, it can be expressed as follows:-

- $p \geq 0.5$ => class = 1
- $p < 0.5$ => class = 0

Types of Logistic Regression

Logistic Regression model can be classified into three groups based on the target variable categories. These three groups are described below:-

1. Binary Logistic Regression:
   - In Binary Logistic Regression, the target variable has two possible categories. The common examples of categories are yes or no, good or bad, true or false, spam or no spam and pass or fail.
2. Multinomial Logistic Regression:
   - In Multinomial Logistic Regression, the target variable has three or more categories which are not in any particular order. So, there are three or more nominal categories. The examples include the type of categories of fruits - apple, mango, orange and banana.
3. Ordinal Logistic Regression:
   - In Ordinal Logistic Regression, the target variable has three or more ordinal categories. So, there is intrinsic order involved with the categories. For example, the student performance can be categorized as poor, average, good and excellent.

In [1]:
```
### Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.linear_model import SGDClassifier
```

In [2]:
```
### Import the Dataset
df = pd.read_csv(r'C:\Users\hp\Desktop\100DaysOfDataScience\Day 45\User_Data.csv')
df.head()
```

Out[2]:

	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 [3]:
```
df.shape ### Checking Shape
```

Out[3]: (400, 5)

In [4]:
```
df.describe() ### Get information of the Dataset
```

Out[4]:

	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 [5]:
```
df.columns ### Checking Columns
```

Out[5]: Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')

In [6]:
```
df.info() ### Checking Information About a DataFrame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   User ID          400 non-null    int64 
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64 
 3   EstimatedSalary  400 non-null    int64 
 4   Purchased        400 non-null    int64 
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [7]:
```
df.isnull().sum() ### Checking Null Values in the Data
```

Out[7]:
```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [8]:
```
df1 = pd.DataFrame.copy(df)
df1.shape
```

Out[8]: (400, 5)

In [9]:
```
for i in df1.columns:
    print({i:df1[i].unique()}) ### Checking Unique values in each columns
```

```
{'User ID': array([15624510, 15810944, 15668575, 15603246, 15804002, 15728773,
       15598044, 15694829, 15600575, 15727311, 15570769, 15606274,
       15746139, 15704987, 15628972, 15697686, 15733883, 15617482,
       15704583, 15621083, 15649487, 15736760, 15714658, 15599081,
       15705113, 15631159, 15792818, 15633531, 15744529, 15669656,
       15581198, 15729054, 15573452, 15776733, 15724858, 15713144,
       15690188, 15689425, 15671766, 15782806, 15764419, 15591915,
       15772798, 15792008, 15715541, 15639277, 15798850, 15776348,
       15727696, 15793813, 15694395, 15764195, 15744919, 15671655,
       15654901, 15649136, 15775562, 15807481, 15642885, 15789109,
       15814004, 15673619, 15595135, 15583681, 15605000, 15718071,
       15679760, 15654574, 15577178, 15595324, 15756932, 15726358,
       15595228, 15782530, 15592877, 15651983, 15746737, 15774179,
       15667265, 15655123, 15595917, 15668385, 15709476, 15711218,
       15798659, 15663939, 15694946, 15631912, 15768816, 15682268,
       15684801, 15636428, 15809823, 15699284, 15786993, 15709441,
       15710257, 15582492, 15575694, 15756820, 15766289, 15593014,
       15584545, 15675949, 15672091, 15801658, 15706185, 15789863,
       15720943, 15697997, 15665416, 15660200, 15619653, 15773447,
       15739160, 15689237, 15679297, 15591433, 15642725, 15701962,
       15811613, 15741049, 15724423, 15574305, 15678168, 15697020,
       15610801, 15745232, 15722758, 15792102, 15675185, 15801247,
       15725660, 15638963, 15800061, 15578006, 15668504, 15687491,
       15610403, 15741094, 15807909, 15666141, 15617134, 15783029,
       15622833, 15746422, 15750839, 15749130, 15779862, 15767871,
       15679651, 15576219, 15699247, 15619087, 15605327, 15610140,
       15791174, 15602373, 15762605, 15598840, 15744279, 15670619,
       15599533, 15757837, 15697574, 15578738, 15762228, 15614827,
       15789815, 15579781, 15587013, 15570932, 15794661, 15581654,
       15644296, 15614420, 15609653, 15594577, 15584114, 15673367,
       15685576, 15774727, 15694288, 15603319, 15759066, 15814816,
       15724402, 15571059, 15674206, 15715160, 15730448, 15662067,
       15779581, 15662901, 15689751, 15667742, 15738448, 15680243,
       15745083, 15708228, 15628523, 15708196, 15735549, 15809347,
       15660866, 15766609, 15654230, 15794566, 15800890, 15697424,
       15724536, 15735878, 15707596, 15657163, 15622478, 15779529,
       15636023, 15582066, 15666675, 15732987, 15789432, 15663161,
       15694879, 15593715, 15575002, 15622171, 15795224, 15685346,
       15691808, 15721007, 15794253, 15694453, 15813113, 15614187,
       15619407, 15646227, 15660541, 15753874, 15617877, 15772073,
       15701537, 15736228, 15780572, 15769596, 15586996, 15722061,
       15638003, 15775590, 15730688, 15753102, 15810075, 15723373,
       15795298, 15584320, 15724161, 15750056, 15609637, 15794493,
       15569641, 15815236, 15811177, 15680587, 15672821, 15767681,
       15600379, 15801336, 15721592, 15581282, 15746203, 15583137,
       15680752, 15688172, 15791373, 15589449, 15692819, 15727467,
       15734312, 15764604, 15613014, 15759684, 15609669, 15685536,
       15750447, 15663249, 15638646, 15734161, 15631070, 15761950,
       15649668, 15713912, 15586757, 15596522, 15625395, 15760570,
       15566689, 15725794, 15673539, 15705298, 15675791, 15747043,
       15736397, 15678201, 15720745, 15637593, 15598070, 15787550,
       15603942, 15733973, 15596761, 15652400, 15717893, 15622585,
       15733964, 15753861, 15747097, 15594762, 15667417, 15684861,
       15742204, 15623502, 15774872, 15611191, 15674331, 15619465,
       15575247, 15695679, 15713463, 15785170, 15796351, 15639576,
       15693264, 15589715, 15769902, 15587177, 15814553, 15601550,
       15664907, 15612465, 15810800, 15665760, 15588080, 15776844,
       15717560, 15629739, 15729908, 15716781, 15646936, 15768151,
       15579212, 15721835, 15800515, 15591279, 15587419, 15750335,
       15699619, 15606472, 15778368, 15671387, 15573926, 15709183,
       15577514, 15778830, 15768072, 15768293, 15654456, 15807525,
       15574372, 15671249, 15779744, 15624755, 15611430, 15774744,
       15629885, 15708791, 15793890, 15646091, 15596984, 15800215,
       15577806, 15749381, 15683758, 15670615, 15715622, 15707634,
       15806901, 15775335, 15724150, 15627220, 15672330, 15668521,
       15807837, 15592570, 15748589, 15635893, 15757632, 15691863,
       15706071, 15654296, 15755018, 15594041], dtype=int64)}
{'Gender': array(['Male', 'Female'], dtype=object)}
{'Age': array([19, 35, 26, 27, 32, 25, 20, 18, 29, 47, 45, 46, 48, 49, 31, 21, 28,
       33, 30, 23, 24, 22, 59, 34, 39, 38, 37, 42, 40, 36, 41, 58, 55, 52,
       60, 56, 53, 50, 51, 57, 44, 43, 54], dtype=int64)}
{'EstimatedSalary': array([ 19000,  20000,  43000,  57000,  76000,  58000,  84000, 150000,
        33000,  65000,  80000,  52000,  86000,  18000,  82000,  25000,
        26000,  28000,  29000,  22000,  49000,  41000,  23000,  30000,
        74000, 137000,  16000,  44000,  90000,  27000,  72000,  31000,
        17000,  51000, 108000,  15000,  79000,  54000, 135000,  89000,
        32000,  83000,  55000,  48000, 117000,  87000,  66000, 120000,
        63000,  68000, 113000, 112000,  42000,  88000,  62000, 118000,
        85000,  81000,  50000, 116000, 123000,  73000,  37000,  59000,
       149000,  21000,  35000,  71000,  61000,  75000,  53000, 107000,
        96000,  45000,  47000, 100000,  38000,  69000, 148000, 115000,
        34000,  60000,  70000,  36000,  39000, 134000, 101000, 130000,
       114000, 142000,  78000, 143000,  91000, 144000, 102000, 126000,
       133000, 147000, 104000, 146000, 122000,  97000,  95000, 131000,
        77000, 125000, 106000, 141000,  93000, 138000, 119000, 105000,
        99000, 129000,  46000,  64000, 139000], dtype=int64)}
{'Purchased': array([0, 1], dtype=int64)}
```

In [10]:
```
### Finding categorical variables

colname = [var for var in df.columns if df1[var].dtype=='O']
print('There are {} categorical variables\n'.format(len(colname)))
print('The categorical variables are :', colname)
```

```
There are 1 categorical variables

The categorical variables are : ['Gender']
```

In [11]:
```
### Converting all categorical data into numerical data
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for x in colname:
    df1[x]=le.fit_transform(df1[x])
    le_name_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
    print("Feature",x)
    print("Mapping", le_name_mapping)
```

```
Feature Gender
Mapping {'Female': 0, 'Male': 1}
```

In [12]:
```
df2 = df1.copy()
df2.columns
```

Out[12]:
```
Index(['User ID', 'Gender', 'Age', 'EstimatedSalary', 'Purchased'], dtype='object')
```

In [13]:
```python
### Spliting Data into X and y
X = df2.iloc[:,1:4]
y = df2.iloc[:,-1]
print('X:',X.shape)
print('*' * 17)
print('y:',y.shape)
```

```
X: (400, 3)
*****************
y: (400,)
```

In [14]:
```python
### Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
#x = scaler.fit_transform(x)
print(X)
```

```
[[ 1.02020406 -1.78179743 -1.49004624]
 [ 1.02020406 -0.25358736 -1.46068138]
 [-0.98019606 -1.11320552 -0.78528968]
 ...
 [-0.98019606  1.17910958 -1.46068138]
 [ 1.02020406 -0.15807423 -1.07893824]
 [-0.98019606  1.08359645 -0.99084367]]
```

In [15]:
```python
y = y.astype(int) #convert y in to integer always perform this operation
```

In [16]:
```python
### Spliting into Training and Testing Data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=10)

print("X_train: ",X_train.shape)
print("X_test: ",X_test.shape)
print("y_train: ",y_train.shape)
print("y_test: ",y_test.shape)
```

```
X_train:  (320, 3)
X_test:  (80, 3)
y_train:  (320,)
y_test:  (80,)
```

In [17]:
```python
#create a model object
lr = LogisticRegression()
#train the model object
lr.fit(X_train,y_train)
#predict using the model
y_pred = lr.predict(X_test)
print(y_pred)
```

```
[0 0 1 1 0 1 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 1 0 0 0 0 1 1 0 1
 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1 1 0 1 0 0 0 1 0 0
 0 0 0 0 1 0]
```

In [18]:
```python
lr.score(X_train,y_train)
```

Out[18]:
```
0.846875
```

In [19]:
```python
print(list(zip(df2[:-1],lr.coef_.ravel())))
print(lr.intercept_)
```

```
[('User ID', 0.2005590457098062), ('Gender', 2.1832648113729274), ('Age', 1.0595280271232759)]
[-1.01288431]
```

In [20]:
```python
# Checking confusion matrix for the model
cfm = confusion_matrix(y_test,y_pred)
dff = pd.DataFrame(cfm)
dff.style.set_properties(**{"background-color": "#F3FFFF","color":"black","border": "2px  solid black"})
```

Out[20]:

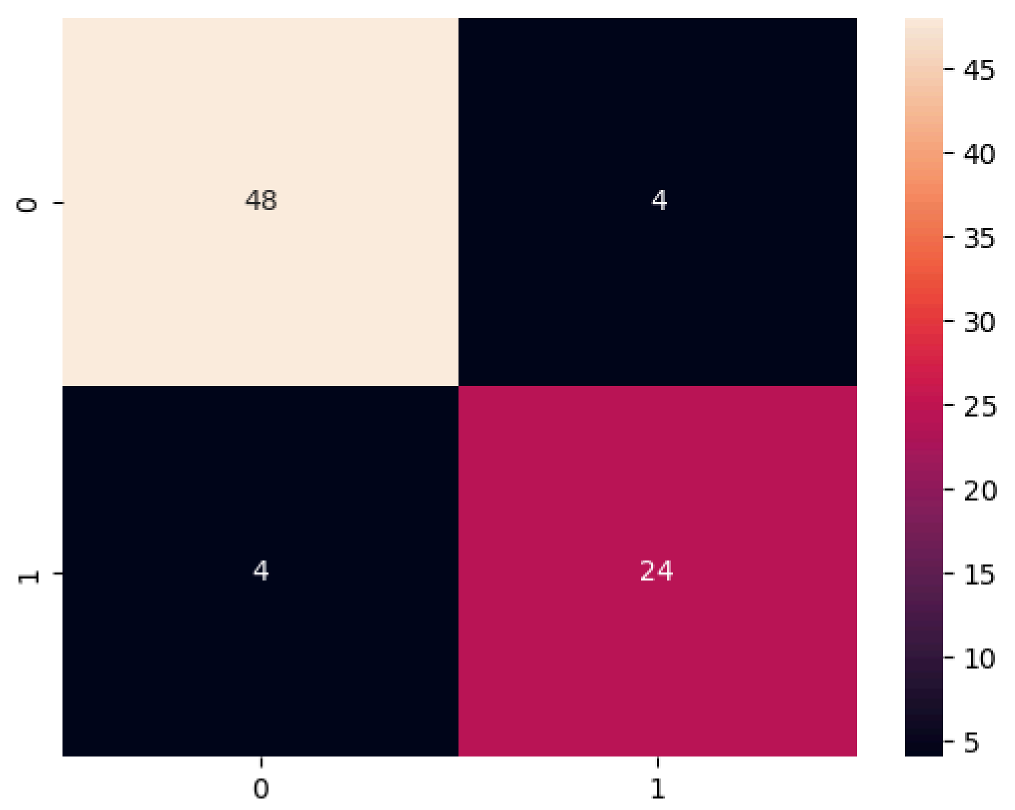
	0	1
0	48	4
1	4	24

In [21]:
```python
print('\nTrue Positives(TP) = ', cfm[0,0])
print('\nTrue Negatives(TN) = ', cfm[1,1])
print('\nFalse Positives(FP) = ', cfm[0,1])
print('\nFalse Negatives(FN) = ', cfm[1,0])
```

```
True Positives(TP) =  48

True Negatives(TN) =  24

False Positives(FP) =  4

False Negatives(FN) =  4
```

In [22]:
```python
# Heatmap of Confusion matrix
sns.heatmap(pd.DataFrame(cfm), annot=True)
```

Out[22]:
```
<Axes: >
```



In [23]:
```python
# Checking classification report score for the model
cr = classification_report(y_test,y_pred)
print("Classification report: ")
print(cr)

# Checking accuracy score for the model
acc = accuracy_score(y_test,y_pred)
print("Accuracy of the model: ",acc)
```

```
Classification report: 
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        52
           1       0.86      0.86      0.86        28

    accuracy                           0.90        80
   macro avg       0.89      0.89      0.89        80
weighted avg       0.90      0.90      0.90        80

Accuracy of the model:  0.9
```

In [ ]: