NAME: HRISHIKESH SHIVPUTRA KAMBLE

PROJECT:-

SOLVING CLASSIFICATION PREDICTION FOR "MANUFACTURING DEFECT" DATASET USING "LOGISTIC REGRESSION","NAIVES BAYES CLASSIFICATION","SUPPORT VECTOR CLASSIFIER","K NEAREST NEIGHBOUR", "DESICION TREE CLASSIFIER".

DATA:-

PRODUCTION METRICS

PRODUCTION VOLUME:-

NUMBER OF UNITS PRODUCED PER DAY.

DATA TYPE: INTEGER.

RANGE: 100 TO 1000 UNITS/DAY.

PRODUCTION COST:-

COST INCURRED FOR PRODUCTION PER DAY.

DATA TYPE: FLOAT.

RANGE: 5000 TO 20000.

SUPPLY CHAIN AND LOGISTICS

SUPPLIER QUALITY:-

QUALITY RATINGS OF SUPPLIERS.

DATA TYPE: FLOAT (%).

RANGE: 80% TO 100%.

DELIVERY DELAY:-

AVERAGE DELAY IN DELIVERY.

DATA TYPE: INTEGER (DAYS).

RANGE: 0 TO 5 DAYS.

QUALITY CONTROL AND DEFECT RATES

DEFECT RATE:-

DEFECTS PER THOUSAND UNITS PRODUCED.

DATA TYPE: FLOAT.

RANGE: 0.5 TO 5.0 DEFECTS.

QUALITY SCORE:-

OVERALL QUALITY ASSESSMENT.

DATA TYPE: FLOAT (%).

RANGE: 60% TO 100%.

MAINTENANCE AND DOWNTIME

MAINTENANCE HOURS:-

HOURS SPENT ON MAINTENANCE PER WEEK.

DATA TYPE: INTEGER.

RANGE: 0 TO 24 HOURS.

DOWNTIME PERCENTAGE:

PERCENTAGE OF PRODUCTION DOWNTIME.

DATA TYPE: FLOAT (%).

RANGE: 0% TO 5%.

INVENTORY MANAGEMENT

INVENTORY TURNOVER:-

RATIO OF INVENTORY TURNOVER.

DATA TYPE: FLOAT.

RANGE: 2 TO 10.

STOCKOUT RATE:-

RATE OF INVENTORY STOCKOUTS.

DATA TYPE: FLOAT (%).

RANGE: 0% TO 10%.

WORKFORCE PRODUCTIVITY AND SAFETY

WORKER PRODUCTIVITY:-

PRODUCTIVITY LEVEL OF THE WORKFORCE.

DATA TYPE: FLOAT (%).

RANGE: 80% TO 100%.

SAFETY INCIDENTS:-

NUMBER OF SAFETY INCIDENTS PER MONTH.

DATA TYPE: INTEGER.

RANGE: 0 TO 10 INCIDENTS.

ENERGY CONSUMPTION AND EFFICIENCY

ENERGY CONSUMPTION:-

ENERGY CONSUMED IN KWH.

DATA TYPE: FLOAT.

RANGE: 1000 TO 5000 KWH.

ENERGY EFFICIENCY:-

EFFICIENCY FACTOR OF ENERGY USAGE.

DATA TYPE: FLOAT.

RANGE: 0.1 TO 0.5.

ADDITIVE MANUFACTURING

ADDITIVE PROCESS TIME:-

TIME TAKEN FOR ADDITIVE MANUFACTURING.

DATA TYPE: FLOAT (HOURS).

RANGE: 1 TO 10 HOURS.

ADDITIVE MATERIAL COST:-

COST OF ADDITIVE MATERIALS PER UNIT.

DATA TYPE: FLOAT.

RANGE:100 TO $500.

TARGET VARIABLE

APPROACH:

1.LOAD THE REQUIRED LIBRARIES SUCH AS PANDAS,MATPLOTLIB,SEABORN ALONG WITH GIVEN DATASET.

2.PERFORM EDA ON THE GIVEN DATASET.

3.CONVERT ALL THE REQUIRED COLUMNS INTO NUMERIAL COLUMNS USING GET DUMMIES FUNCTION FROM PANDAS LIBRARY.

4.CHECK FOR CORRELATION BETWEEN FEATURES AND TARGET AND CONSIDER THE ONLY FEATURES WITH HIGHER CORRELATION.

5.IMPORT "LOGISTIC REGRESSION, NAIVES BAYES CLASSIFICATION,SUPPORT VECTOR CLASSIFIER,K NEAREST NEIGHBOUR", AND SPLIT THE GIVEN DATASET INTO TRAINING AND TESTING DATA USING TRAIN_TEST_SPLIT FUNCTION.THEN CALUCLATE ACCURACY SCORE USING SKLEARN LIBRARY BY IMPORTING METRICS.

6.ONCE WE GET ACCURACY SCORE OF ALL MODELS FOR BOTH TRAING AND TESTING DATA, CREATE A DATAFRAME AND LOAD ALL THE ACCURACY OF ALL MODEL.

7.VISUALIZATION: ONCE THE DATASET IS CREATED PLOT THE ACCURACIES OF ALL THE MODELS USING BARPLOT USING MATPLOTLIB.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
pd.set_option("Display.max_columns",100)
```

```
C:\Users\Hrishikesh\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires version
'1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [3]:
```
data=pd.read_csv(r"C:\Users\Hrishikesh\Desktop\DATA SCIENCE\manufacturing_defect_dataset.csv")
data      # LOADING THE GIVEN DATASET
```

Out[3]:

	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePercentage	Invent
0	202	13175.403783	86.648534	1	3.121492	63.463494	9	0.052343	
1	535	19770.046093	86.310664	4	0.819531	83.697818	20	4.908328	
2	960	19060.820997	82.132472	0	4.514504	90.350550	1	2.464923	
3	370	5647.606037	87.335966	5	0.638524	67.628690	8	4.692476	
4	206	7472.222236	81.989893	3	3.867784	82.728334	9	2.746726	
...	...	...	...	...	...	...	...	...	
3235	762	11325.689263	89.252385	2	2.667570	87.141681	16	0.987719	
3236	335	5598.837988	95.701437	4	0.751272	95.562997	11	0.178163	
3237	835	11736.177712	96.431554	5	4.899756	77.973442	0	4.873429	
3238	302	13664.196210	91.089782	1	4.057665	95.755591	6	0.071663	
3239	355	13563.605806	83.595956	2	2.705502	94.630965	13	4.803394	

3240 rows × 17 columns

In [4]:
```
data.isna().sum()                          # CHECKING FOR NULL VALUES
```

Out[4]:
```
ProductionVolume        0
ProductionCost          0
SupplierQuality         0
DeliveryDelay           0
DefectRate              0
QualityScore            0
MaintenanceHours        0
DowntimePercentage      0
InventoryTurnover       0
StockoutRate            0
WorkerProductivity      0
SafetyIncidents         0
EnergyConsumption       0
EnergyEfficiency        0
AdditiveProcessTime     0
AdditiveMaterialCost    0
DefectStatus            0
dtype: int64
```

In [5]:
```
data.shape                               # SHOWS NUMBER OF ROWS AND COLUMNS
```

Out[5]:
```
(3240, 17)
```

In [6]:
```
data.head()                              # DISPLAYS FIRST 5 ROWS
```

Out[6]:

	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePercentage	Inventory
0	202	13175.403783	86.648534	1	3.121492	63.463494	9	0.052343	
1	535	19770.046093	86.310664	4	0.819531	83.697818	20	4.908328	
2	960	19060.820997	82.132472	0	4.514504	90.350550	1	2.464923	
3	370	5647.606037	87.335966	5	0.638524	67.628690	8	4.692476	
4	206	7472.222236	81.989893	3	3.867784	82.728334	9	2.746726	

In [7]:
```
data.info()                      # SHOWS ALL INFORMATION REGARDING THE DATA SUCH AS NULL VALUE, COLUMNS,DATATYPES
```

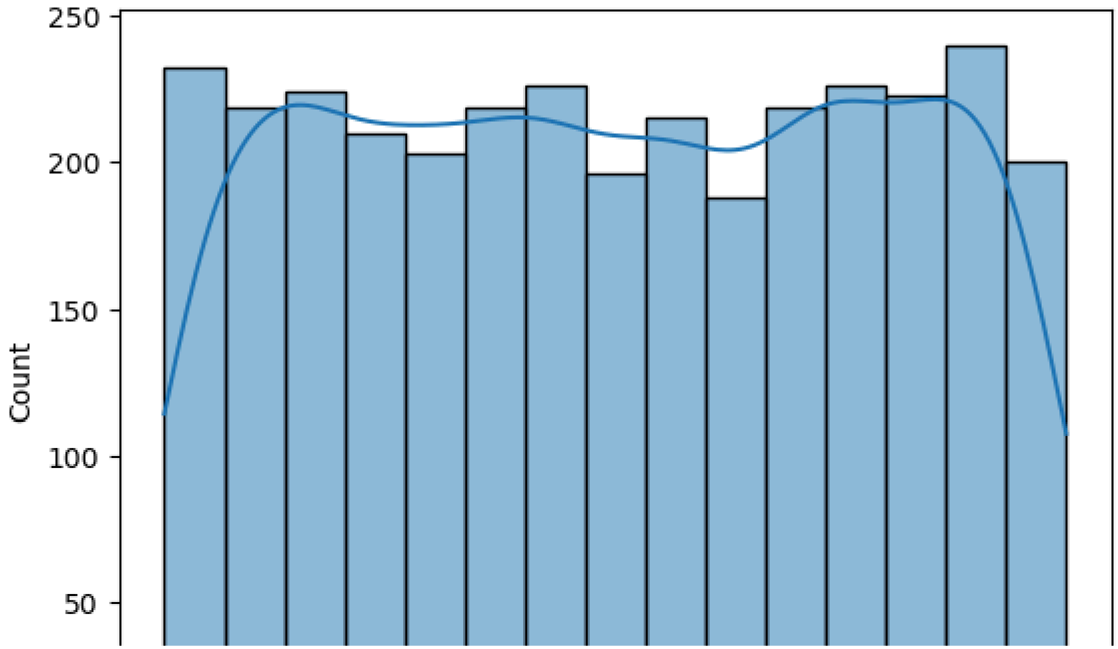
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3240 entries, 0 to 3239
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   ProductionVolume      3240 non-null   int64  
 1   ProductionCost        3240 non-null   float64
 2   SupplierQuality       3240 non-null   float64
 3   DeliveryDelay         3240 non-null   int64  
 4   DefectRate            3240 non-null   float64
 5   QualityScore          3240 non-null   float64
 6   MaintenanceHours      3240 non-null   int64  
 7   DowntimePercentage    3240 non-null   float64
 8   InventoryTurnover     3240 non-null   float64
 9   StockoutRate          3240 non-null   float64
 10  WorkerProductivity    3240 non-null   float64
 11  SafetyIncidents       3240 non-null   int64  
 12  EnergyConsumption     3240 non-null   float64
 13  EnergyEfficiency      3240 non-null   float64
 14  AdditiveProcessTime   3240 non-null   float64
 15  AdditiveMaterialCost  3240 non-null   float64
 16  DefectStatus          3240 non-null   int64  
dtypes: float64(12), int64(5)
memory usage: 430.4 KB
```

In [8]:
```
data.describe()                # SHOWS THE ALL DETAILS REGARDING ALL NUMERICAL COLUMNS
```

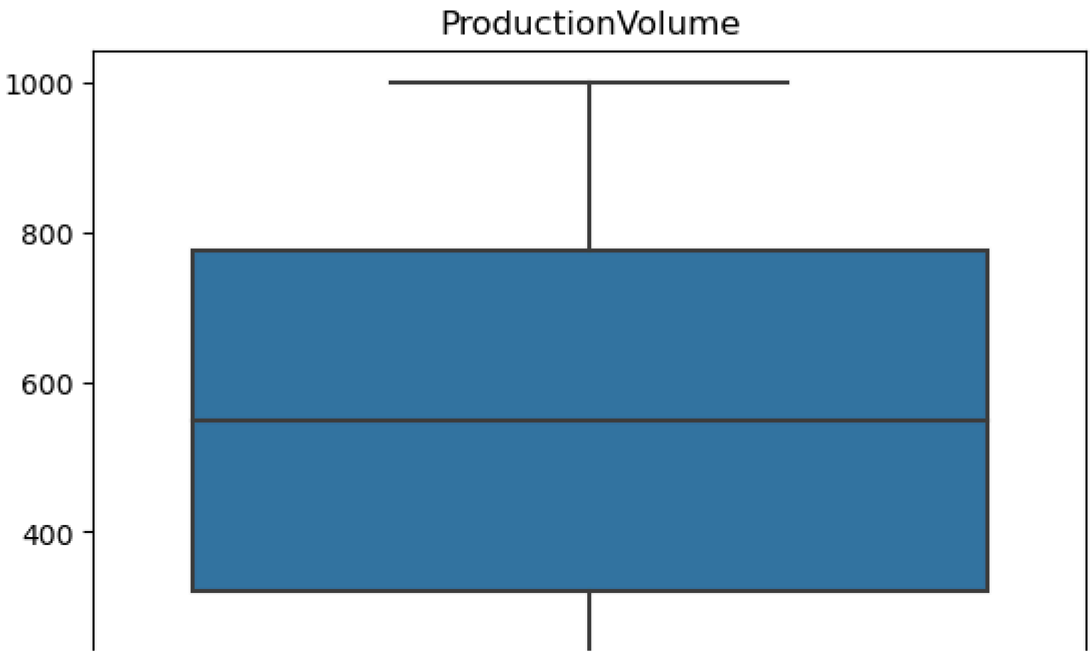
Out[8]:

	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePercentage	Inve
count	3240.000000	3240.000000	3240.000000	3240.000000	3240.000000	3240.000000	3240.000000	3240.000000	
mean	548.523148	12423.018476	89.833290	2.558951	2.749116	80.134272	11.476543	2.501373	
std	262.402073	4308.051904	5.759143	1.705804	1.310154	11.611750	6.872684	1.443684	
min	100.000000	5000.174521	80.004820	0.000000	0.500710	60.010098	0.000000	0.001665	
25%	322.000000	8728.829280	84.869219	1.000000	1.598033	70.103420	5.750000	1.264597	
50%	549.000000	12405.204656	89.704861	3.000000	2.708775	80.265312	12.000000	2.465151	
75%	775.250000	16124.462428	94.789936	4.000000	3.904533	90.353822	17.000000	3.774861	
max	999.000000	19993.365549	99.989214	5.000000	4.998529	99.996993	23.000000	4.997591	

In [9]:
```
for i in data.columns:
    sns.histplot(data[i],kde=True)    # PLOT HISTPLOT TO SEE DATA DISTRIBUTION
    plt.show()
```



In [10]:
```
for i in data.columns:                # CHECKING FOR OUTLIERS USING BOX PLOT
    sns.boxplot(data[i])
    plt.title(i)
    plt.show()
```

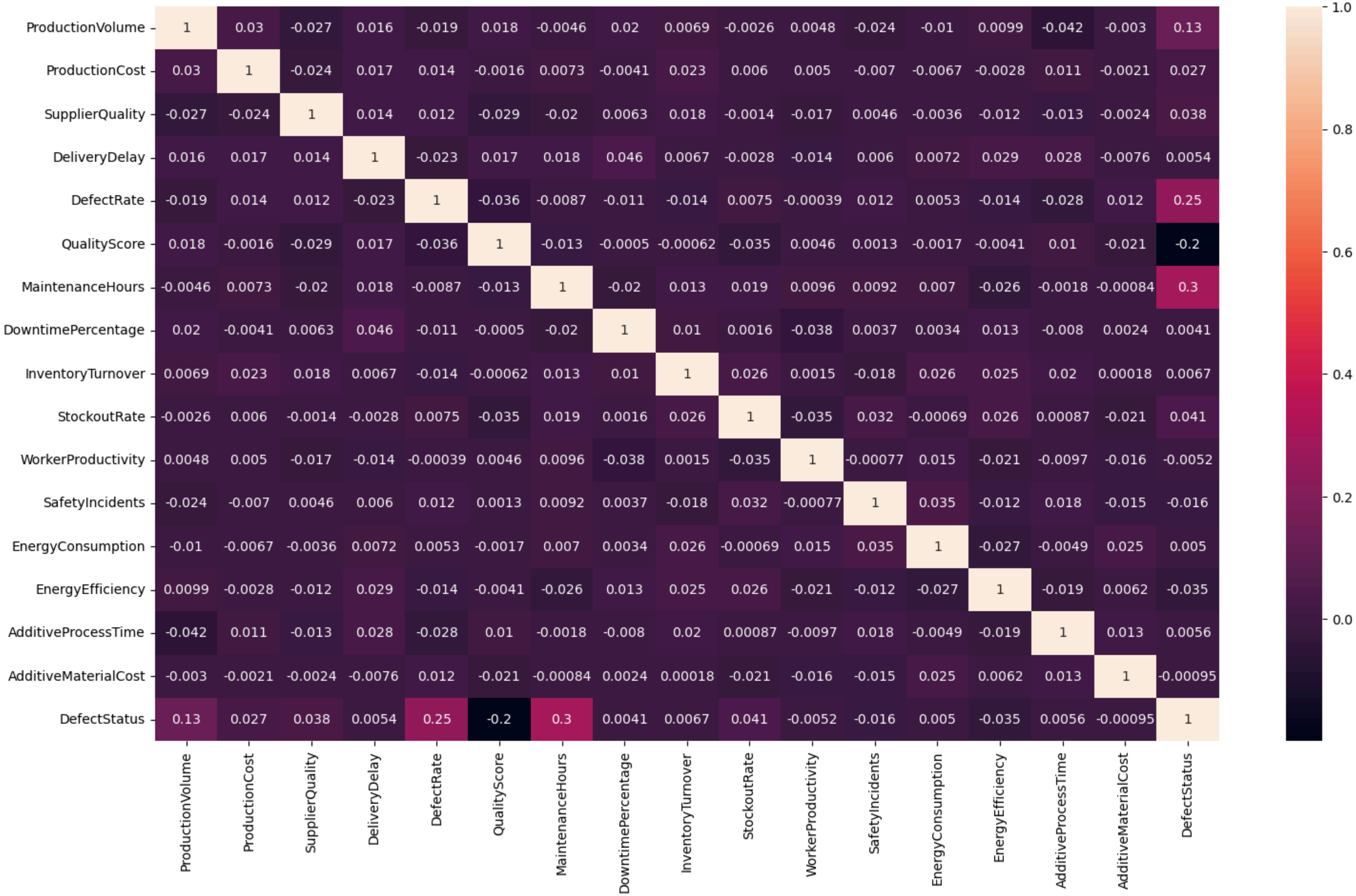


In [11]:
```
data.corr()*100            #CHECKING FOR CORRELATION BETWEEN FEATURES AND TARGET
```

Out[11]:

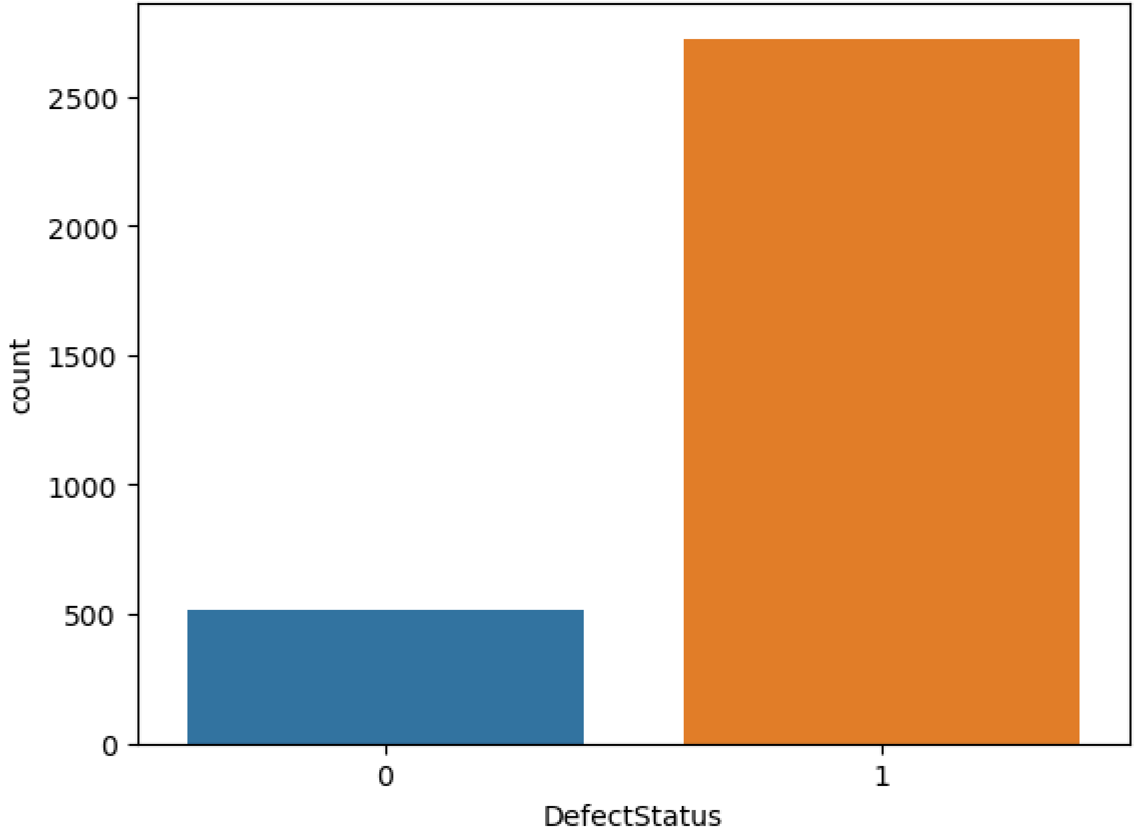
	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePer
ProductionVolume	100.000000	2.958397	-2.655870	1.619300	-1.935997	1.782617	-0.455379	'
ProductionCost	2.958397	100.000000	-2.410342	1.736470	1.442831	-0.160269	0.733253	-(
SupplierQuality	-2.655870	-2.410342	100.000000	1.423341	1.215710	-2.932955	-1.962564	(
DeliveryDelay	1.619300	1.736470	1.423341	100.000000	-2.302384	1.726761	1.814430	4
DefectRate	-1.935997	1.442831	1.215710	-2.302384	100.000000	-3.634961	-0.868677	-'
QualityScore	1.782617	-0.160269	-2.932955	1.726761	-3.634961	100.000000	-1.336578	-(
MaintenanceHours	-0.455379	0.733253	-1.962564	1.814430	-0.868677	-1.336578	100.000000	-2
DowntimePercentage	1.990492	-0.407842	0.629765	4.624665	-1.120780	-0.050486	-2.049382	10(
InventoryTurnover	0.694672	2.274864	1.822809	0.668538	-1.414767	-0.061775	1.275795	'
StockoutRate	-0.263723	0.600573	-0.139270	-0.276676	0.754714	-3.504899	1.925128	(
WorkerProductivity	0.475361	0.503017	-1.738914	-1.386851	-0.038756	0.459116	0.962166	-3
SafetyIncidents	-2.419484	-0.695803	0.455226	0.602982	1.219621	0.129290	0.917376	(
EnergyConsumption	-1.021279	-0.672774	-0.358159	0.723315	0.529744	-0.169381	0.701653	(
EnergyEfficiency	0.992340	-0.277833	-1.161360	2.946756	-1.416829	-0.414694	-2.648641	'
AdditiveProcessTime	-4.239337	1.107507	-1.250659	2.808467	-2.842590	0.998600	-0.181974	-(
AdditiveMaterialCost	-0.298013	-0.211446	-0.242994	-0.760067	1.159566	-2.122258	-0.084454	(
DefectStatus	12.897325	2.672044	3.818395	0.542464	24.574565	-19.921873	29.710679	(

In [12]:
```
plt.figure(figsize=(18,10))
sns.heatmap(data.corr(),annot=True)
plt.show()
```



In [13]:
```
sns.countplot(data=data,x="DefectStatus")
```

Out[13]: <Axes: xlabel='DefectStatus', ylabel='count'>



FROM ABOVE GRAPH I COME TO KNOW THAT THERE IS SOME BIASNESS IN DEFECT STATUS THAT NON DEFECTIVE ENTERIES ARE RARE.

In [14]:
```
data["MaintenanceHours"].sum()
```

Out[14]: 37184

TOTAL MAINTENANCE HOUR IS 37184 HRS

In [15]:
```
data["QualityScore"].mean()
```

Out[15]: 80.13427211241357

THE AVERAGE QUALITY SCORE FOR PRODUCTS IS 80

In [16]:
```
data[["ProductionVolume","ProductionCost"]].sum().reset_index()
```

Out[16]:

	index	0
0	ProductionVolume	1.777215e+06
1	ProductionCost	4.025058e+07

TOTAL PRODUCTION COST FOR TOTAL PRODUCTION(1777215 QTY) IS RS.40250579

In [17]:
```
data["SafetyIncidents"].sum()
```

Out[17]: 14877

TOTAL NUMBER OF SAFETY INCIDENT OCCURED ARE 14877

In [18]:
```
data["DowntimePercentage"].mean()
```

Out[18]: 2.501373151490477

AVERAGE DOWNTIME FOR GIVEN DATASET IS 2.5%

In [19]:
```
data["EnergyConsumption"].sum()
```

Out[19]: 9682722.028902918

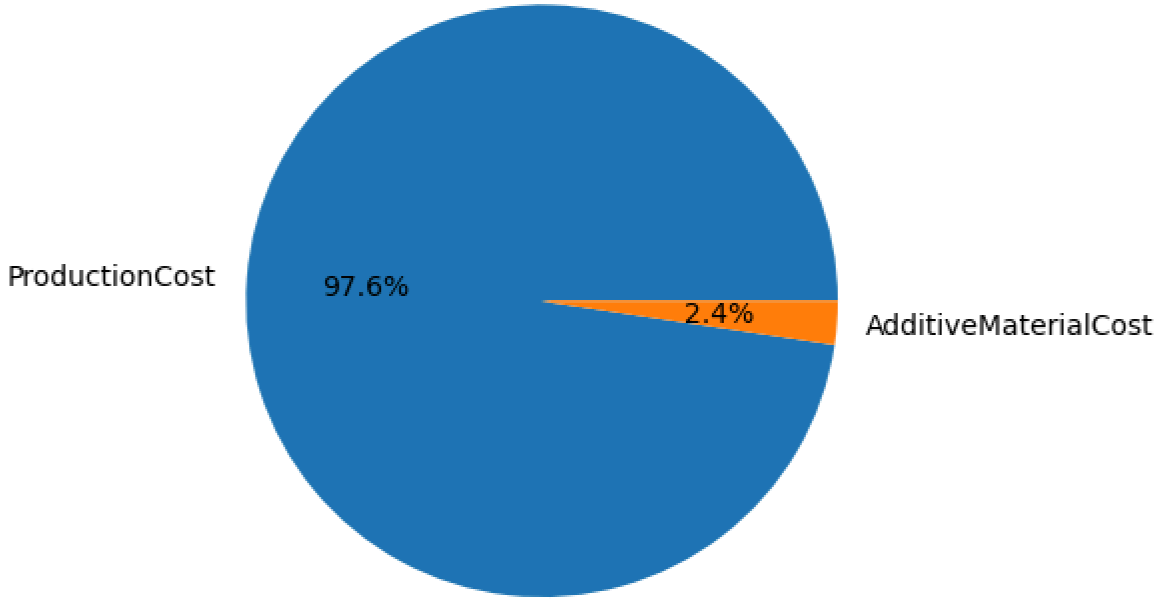
TOTAL ENERGY CONSUMED IN KWH IS 9682722 KWH

In [20]:
```
o=data[["ProductionCost","AdditiveMaterialCost"]].sum().reset_index()
o=np.around(o,2)
o
```

Out[20]:

	index	0
0	ProductionCost	40250579.86
1	AdditiveMaterialCost	970430.15

In [21]:
```
plt.pie(o[0],labels=o["index"],autopct="%1.1f%%")
plt.show()
```



In [22]:
```
data["DefectRate"].mean()
```

Out[22]:
```
2.749116379386996
```

AVERAGE DEFECTIVE RATE IS 2.75%

In [ ]:
```

```

In [23]:
```
FEATURE=data[['ProductionVolume', 'ProductionCost', 'SupplierQuality',
       'DeliveryDelay', 'DefectRate', 'QualityScore', 'MaintenanceHours',
       'DowntimePercentage', 'InventoryTurnover', 'StockoutRate',
       'WorkerProductivity', 'SafetyIncidents', 'EnergyConsumption',
       'EnergyEfficiency', 'AdditiveProcessTime', 'AdditiveMaterialCost']]
TARGET=data['DefectStatus']
                        # STORE DATA INTO FEATURES AND TARGET ACCORDINGLY
```

In [24]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(FEATURE,TARGET,train_size=.85)
  # SPLIT THE DATASET INTO TRAIN AND TESTING DATA
```

In [25]:
```
from sklearn.preprocessing import MinMaxScaler
M=MinMaxScaler()        # IMPORT MINMAX SCALER FOR SCALING
```

In [26]:
```
x_train[[ 'ProductionCost', 'SupplierQuality','WorkerProductivity','EnergyConsumption','AdditiveMaterialCost','Quality
x_test[[ 'ProductionCost', 'SupplierQuality','WorkerProductivity','EnergyConsumption','AdditiveMaterialCost','QualityS
```

In [27]:
```
x_train
```

Out[27]:

	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePercentage	Invent
750	314	0.741229	0.864223	0	1.767439	0.494821	14	4.147022	
854	612	0.974625	0.368772	5	2.414440	0.889253	3	2.617827	
618	424	0.877477	0.423033	3	0.590979	0.829542	17	3.377587	
874	747	0.544718	0.149785	0	1.162732	0.532157	8	0.951805	
511	669	0.673261	0.421389	4	1.645821	0.488925	13	1.836956	
...	...	...	...	...	...	...	...	...	
2459	433	0.672544	0.113234	4	4.586594	0.945415	3	1.787886	
956	871	0.357180	0.832350	3	3.009391	0.354431	9	0.111644	
1917	342	0.399689	0.550879	4	1.543858	0.759708	19	0.423848	
2770	201	0.861798	0.892530	4	4.870485	0.633665	17	1.952001	
808	737	0.395299	0.927068	1	1.241016	0.341955	12	3.511869	

2754 rows × 16 columns

In [28]:
```
x_test
```

Out[28]:

	ProductionVolume	ProductionCost	SupplierQuality	DeliveryDelay	DefectRate	QualityScore	MaintenanceHours	DowntimePercentage	Invent
2592	404	0.059633	0.678411	5	4.814604	0.349189	6	3.591952	
978	488	0.754596	0.966031	0	2.945905	0.113183	17	4.538595	
2000	139	0.255441	0.613640	0	0.556960	0.631937	10	0.953749	
2893	242	0.009904	0.209442	3	3.896071	0.024530	13	1.358449	
2391	474	0.161985	0.861545	1	3.258825	0.066955	11	3.646038	
...	...	...	...	...	...	...	...	...	
2493	594	0.220576	0.209329	3	4.897323	0.719271	2	0.893239	
1083	220	0.514076	0.788409	4	2.235725	0.399279	11	0.928073	
1414	904	0.485303	0.727489	3	0.908772	0.117463	3	1.765946	
2832	545	0.880303	0.121743	5	4.426865	0.641324	22	3.126963	
1618	813	0.572856	0.075345	5	2.632025	0.941877	16	1.760580	

486 rows × 16 columns

In [30]:
```
import tensorflow as tf
from tensorflow import keras
```

In [31]:
```
model=keras.Sequential([
    keras.layers.Dense(16,input_shape=(16,),activation='relu'),#input layer
    keras.layers.Dropout(0.1),
    keras.layers.Dense(15,activation='relu'),#hidden layer
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10,activation='relu'),#hidden layer
    keras.layers.Dense(1,activation='sigmoid')])#output layer)


model.compile(optimizer='adam',
              loss='binary_crossentropy',
             metrics=['accuracy'])
```

In [32]:
```
model.fit(x_train,y_train,epochs=50)
```

```
Epoch 1/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7243 - loss: 3.9227
Epoch 2/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7184 - loss: 1.4627
Epoch 3/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7540 - loss: 1.0775
Epoch 4/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7659 - loss: 0.8670
Epoch 5/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7721 - loss: 0.7054
Epoch 6/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8014 - loss: 0.5783
Epoch 7/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8140 - loss: 0.5848
Epoch 8/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8124 - loss: 0.5507
Epoch 9/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8153 - loss: 0.5390
Epoch 10/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8340 - loss: 0.4754
Epoch 11/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8131 - loss: 0.5041
Epoch 12/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8226 - loss: 0.4952
Epoch 13/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8301 - loss: 0.4781
Epoch 14/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8176 - loss: 0.4768
Epoch 15/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8269 - loss: 0.4666
Epoch 16/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8508 - loss: 0.4315
Epoch 17/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8301 - loss: 0.4519
Epoch 18/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8174 - loss: 0.4689
Epoch 19/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8359 - loss: 0.4389
Epoch 20/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8391 - loss: 0.4318
Epoch 21/50
```

```
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8287 - loss: 0.4441
Epoch 22/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8350 - loss: 0.4396
Epoch 23/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8249 - loss: 0.4433
Epoch 24/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8236 - loss: 0.4405
Epoch 25/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8423 - loss: 0.4119
Epoch 26/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8387 - loss: 0.4189
Epoch 27/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8326 - loss: 0.4140
Epoch 28/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8396 - loss: 0.4012
Epoch 29/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8333 - loss: 0.4195
Epoch 30/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8429 - loss: 0.4057
Epoch 31/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8313 - loss: 0.4112
Epoch 32/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8336 - loss: 0.4008
Epoch 33/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8413 - loss: 0.3916
Epoch 34/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8398 - loss: 0.3912
Epoch 35/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8356 - loss: 0.3904
Epoch 36/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8235 - loss: 0.4056
Epoch 37/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8476 - loss: 0.3698
Epoch 38/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8439 - loss: 0.3768
Epoch 39/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8467 - loss: 0.3636
Epoch 40/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8500 - loss: 0.3650
Epoch 41/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8417 - loss: 0.3805
Epoch 42/50
```

```
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8306 - loss: 0.3962
Epoch 43/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8388 - loss: 0.3911
Epoch 44/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8398 - loss: 0.3652
Epoch 45/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8408 - loss: 0.3794
Epoch 46/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8360 - loss: 0.3756
Epoch 47/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8465 - loss: 0.3767
Epoch 48/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8498 - loss: 0.3474
Epoch 49/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8431 - loss: 0.3873
Epoch 50/50
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8401 - loss: 0.3684
```

Out[32]:

```
<keras.src.callbacks.history.History at 0x11db09d7990>
```

In [33]:
```
T=model.predict(x_test)
T
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

Out[33]:
```
array([[0.75250596],
       [0.9203568 ],
       [0.8428492 ],
       [0.91548973],
       [0.78184664],
       [0.91170174],
       [0.86980355],
       [0.8342533 ],
       [0.8301319 ],
       [0.57468915],
       [0.8848917 ],
       [0.98116946],
       [0.9379451 ],
       [0.9147485 ],
       [0.8482009 ],
       [0.649905  ],
       [0.9540807 ],
       [0.94738686]
```

In [34]:
```
pred=[]
for i in T:
    if i>0.5:
        pred.append(1)
    else:
        pred.append(0)
```

In [35]:
```
pred
```

Out[35]:
```
[1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
```

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

accuracy_score(y_test,pred)
```

Out[37]:
```
0.8436213991769548
```

In [38]:
```
confusion_matrix(y_test,pred)
```

Out[38]:
```
array([[  1,  75],
       [  1, 409]], dtype=int64)
```

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

```
              precision    recall  f1-score   support

           0       0.50      0.01      0.03        76
           1       0.85      1.00      0.91       410

    accuracy                           0.84       486
   macro avg       0.67      0.51      0.47       486
weighted avg       0.79      0.84      0.78       486
```

In [40]:
```
model.evaluate(x_train,y_train)
```

```
87/87 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.8405 - loss: 0.3694
```

Out[40]:
```
[0.3585779368877411, 0.8427741527557373]
```

In [41]:
```
model.evaluate(x_test,y_test)
```

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 952us/step - accuracy: 0.8447 - loss: 0.3470
```

Out[41]:
```
[0.3662823438644409, 0.8436213731765747]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

LOGISTIC REGRESSION:-

In [ ]:

```
from sklearn.linear_model import LogisticRegression
L=LogisticRegression()
L.fit(x_train,y_train)     #IMPORT LOGISTIC REGRESSION AND FIT
```

In [ ]:

```
L1=L.score(x_train,y_train)*100    #TRAINING ACCURACY
L1
```

In [ ]:

```
L2=L.score(x_test,y_test)*100      #TESTING ACCURACY
L2
```

In [ ]:

```

```

SUPPORT VECTOR CLASSIFIER:-

In [ ]:

```
from sklearn.svm import SVC
S=SVC()
S.fit(x_train,y_train)
```

In [ ]:

```
S1=S.score(x_train,y_train)*100     #TRAINING ACCURACY
S1
```

In [ ]:

```
S2=S.score(x_test,y_test)*100      #TESTING ACCURACY
S2
```

KNEIGHBOR CLASSIFIER:-

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
K=KNeighborsClassifier()
K.fit(x_train,y_train)          #IMPORT KNN AND FIT
```

In [ ]:
```
K1=K.score(x_train,y_train)*100     #TRAINING ACCURACY
K1
```

In [ ]:
```
K2=K.score(x_test,y_test)*100       #TESTING ACCURACY
K2
```

In [ ]:
```
```

NAIVES BAYES:-

In [ ]:
```
from sklearn.naive_bayes import GaussianNB,ComplementNB,MultinomialNB,BernoulliNB
G=GaussianNB()
C=ComplementNB()
M=MultinomialNB()
B=BernoulliNB()                            #IMPORT NAIVES BAYES AND FIT
```

GaussianNB:-

In [ ]:
```
G.fit(x_train,y_train)
```

In [ ]:
```
G1=G.score(x_train,y_train)*100
G1                                 #TRAINING ACCURACY
```

In [ ]:
```
G2=G.score(x_test,y_test)*100    #TESTING ACCURACY
G2
```

BernoulliNB:-

In [ ]:
```
B.fit(x_train,y_train)
```

In [ ]:
```
B1=B.score(x_train,y_train)*100     #TRAINING ACCURACY
B1
```

In [ ]:
```
B2=B.score(x_test,y_test)*100       #TESTING ACCURACY
B2
```

ComplementNB:-

In [ ]:
```
C.fit(x_train,y_train)
```

In [ ]:
```
C1=C.score(x_train,y_train)*100       #TRAINING ACCURACY
C1
```

In [ ]:
```
C2=C.score(x_test,y_test)*100         #TESTING ACCURACY
C2
```

In [ ]:
```

```

MultinomialNB:-

In [ ]:
```
M.fit(x_train,y_train)
```

In [ ]:
```
M1=M.score(x_train,y_train)*100     #TRAINING ACCURACY
M1
```

In [ ]:
```
M2=M.score(x_train,y_train)*100    #TESTING ACCURACY
M2
```

In [ ]:

DECISION TREE CLASSIFIER:-

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
D=DecisionTreeClassifier()        #IMPORT DECISION TREE CLASSIFIER AND FIT
```

In [ ]:
```
D.fit(x_train,y_train)
```

In [ ]:
```
DT1=D.score(x_train,y_train)*100        #TRAINING ACCURACY
DT1
```

In [ ]:
```
DT2=D.score(x_test,y_test)*100
DT2                                   #TESTING ACCURACY
```

In [ ]:

ADABOOST:-

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
A=AdaBoostClassifier()
```

In [ ]:
```
A.fit(x_train,y_train)
```

In [ ]:
```
A1=A.score(x_train,y_train)*100
A1
```

In [ ]:
```
A2=A.score(x_test,y_test)*100
A2
```

ACCURACY GRAPH:-

In [ ]:
```
A={"METHODS":["LOGISTIC REGRESSION","SVC","GAUSSIAN NB","BERNOULLI NB","COMPLEMENT NB","MULTINOMIAL NB","K NEAREST NEI
A=pd.DataFrame(A)
A=np.around(A,2)
A                #CREATE A DATAFRAME FOR ALL ACCURACIES
```

In [ ]:
```
plt.bar(A["METHODS"],A["TRAIN ACCURACY"],width=0.3,label="TRAINING ACCURACY")
plt.bar(A["METHODS"],A["TEST ACCURACY"],align="edge",width=0.3,label="TESTING ACCURACY")
plt.legend(bbox_to_anchor=[1,0,0,1])
plt.xlabel("METHODS------->")
plt.ylabel("ACCURACY------->")
plt.xticks(rotation=90)
plt.show()                   # PLOT THE ACCURACY CHART BETWEEN ALL MODELS ACCURACIES
```

CONCLUSION:

FROM THE ABOVE BAR CHART IT IS CLEAR THAT ADABOOST BEST FOR CLASSIFICATION FOR THIS DATASET .

In [ ]: