



SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

FACULTY OF ENGINEERING & TECHNOLOGY

(Formerly SRM University, Under section 3 of UGC Act, 1956)

**S.R.M. NAGAR, KATTANKULATHUR –603 203, KANCHEEPURAM
DISTRICT**

SCHOOL OF COMPUTING

DEPARTMENT OF NETWORKING AND COMMUNICATIONS

Course Code: 18CSE305J

Course Name: Artificial Intelligence

LAB REPORT

NAME: *****

REG. NO.: *****

SECTION: I2

CSE - Cloud Computing

TABLE OF CONTENT

SR. NO.	NAME OF EXPERIMENT	PAGE NO.
1	Implementation of Toy problem using Python	3
2	Developing agent programs for real-world problems - Graph coloring problem	4
3	Constraint Satisfaction Problem Crypt arithmetic puzzle	6
4	Implementation and analysis of BFS and DFS	8
5	Developing Best first search and A* algorithm for real-world problem	10
6	Implementation of Min-Max algorithm for an application	12
7	Implementation of Unification algorithm & Implementation of Resolution	13
8	Implementation of uncertain methods for an application	15
9	Implementation of learning algorithms for an application	16
10	To Implement NLP programs	20
11	Applying deep learning methods to solve an application	22

Experiment 1: Implementation of Toy problem using Python

Date:06-01-2022

AIM:

Implementation of Toy problem using Python

ALGORITHM:

1. Move forward with 1000 bananas – I will eat up 1 banana on the way forward
2. Leave 998 bananas after 1 km and return with 1 banana – will eat up 1 banana on the way back
3. Pick up the next 1000 bananas and move forward – I will eat up 1 banana on the way forward
4. Leave 998 bananas after 1 km and return with 1 banana - will eat up 1 banana on the way back
5. Will carry the last 1000 bananas from point a and move forward – will eat up 1 banana

OUTPUT:

```
In [1]: total=int(input('Enter no. of bananas at starting'))
distance=int(input('Enter distance you want to cover'))
load_capacity=int(input('Enter max load capacity of your camel'))
lose=0
start=total
for i in range(distance):
    while start>0:
        start=start-load_capacity
        if start<1:
            lose=lose-1
            lose=lose+2
        lose=lose-1
        start=total-lose
        if start==0:
            break
    print(start)

Enter no. of bananas at starting:1000
Enter distance you want to cover:1000
Enter max load capacity of your camel:1000
533
```

RESULT: Implementation of the Toy problem using python has been completed.

Experiment 2: Developing agent programs for real-world problems - Graph coloring problem

Date: 11-01-2022

AIM

Check whether a given graph is Bipartite or not.

ALGORITHM

Algorithm to check if a graph is Bipartite:

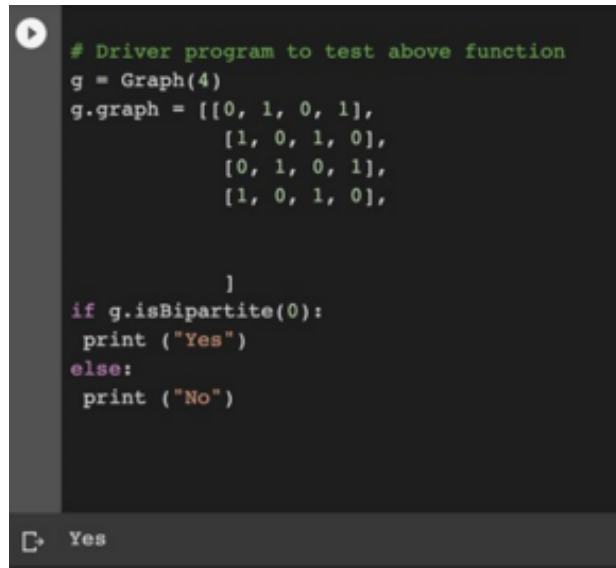
One approach is to check whether the graph is 2-colorable or not using the backtracking algorithm in coloring problem.

Following is a simple algorithm to find out whether a given graph is Bipartite or not using Breadth-First Search (BFS).

1. Assign a RED color to the source vertex (putting it into set U).
2. Color all the neighbors with BLUE color (putting into set V).
3. Color all neighbors with RED color (putting into set U).
4. This way, assign a color to all vertices such that it satisfies all the constraints of the way coloring problem where $m = 2$.
5. While assigning colors, if we find a neighbor which is colored with the same color as the current vertex, then the graph cannot be colored with 2 vertices (or the graph is not Bipartite)

OUTPUT

The given graph is Bipartite as only 2 colors were used.



```
# Driver program to test above function
g = Graph(4)
g.graph = [[0, 1, 0, 1],
           [1, 0, 1, 0],
           [0, 1, 0, 1],
           [1, 0, 1, 0],

           ]
if g.isBipartite(0):
    print ("Yes")
else:
    print ("No")
```

Yes

RESULT: The given graph is Bipartite as only 2 colors were used.

Experiment 3: Constraint Satisfaction Problem Crypt arithmetic puzzle

Date: 28-01-2022

AIM:

Provide a code to solve a crypt arithmetic puzzle using constraint satisfaction.

ALGORITHM:

For this problem, we will define a node, which contains a letter and its corresponding values.

isValid(nodeList, count, word1, word2, word3)

Input – A list of nodes, the number of elements in the node list and three words.

Output – True if the sum of the value for word1 and word2 is same as word3 value.

Begin

 m := 1

 for each letter i from right to left of word1, do

 ch := word1[i]

 for all elements j in the nodeList, do

 if nodeList[j].letter = ch, then

 break

 done

 val1 := val1 + (m * nodeList[j].value)

 m := m * 10

 done

 m := 1

 for each letter i from right to left of word2, do

 ch := word2[i]

 for all elements j in the nodeList, do

 if nodeList[j].letter = ch, then

 break

 done

 val2 := val2 + (m * nodeList[j].value)

 m := m * 10

 done

 m := 1

 for each letter i from right to left of word3, do

 ch := word3[i]

 for all elements j in the nodeList, do

 if nodeList[j].letter = ch, then

 break

done

val3 := val3 + (m * nodeList[j].value)

m := m * 10

done

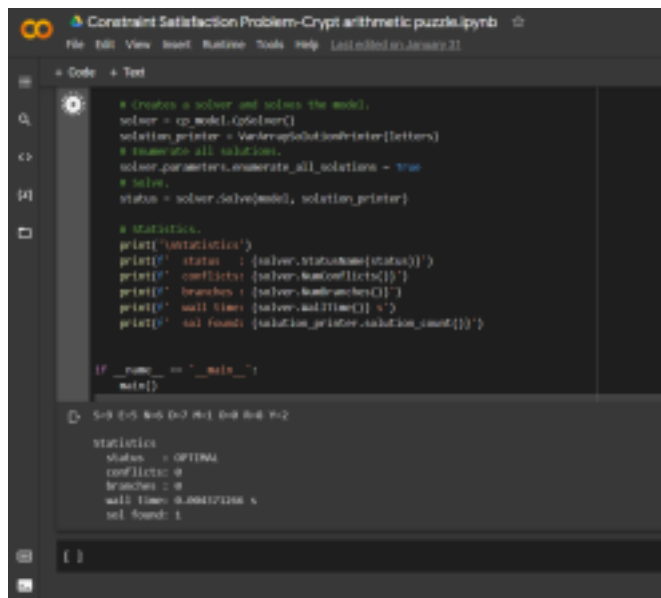
if val3 = (val1 + val2), then

return true

return false

End

OUTPUT:



The screenshot shows a Jupyter Notebook titled "Constraint Satisfaction Problem-Crypt arithmetic puzzle.ipynb". The code cell contains the following Python code:

```
# Creates a solver and solves the model.
solver = cp.model.Cpsolver()
solution_printer = VarkitOptSolPrinter([letters])
# Enumerate all solutions.
solver.parameters.enumerate_all_solutions = True
# Solve.
status = solver.Solve(model, solution_printer)

# Statistics.
print("Statistics")
print("  status : {solver.statusname(status)}")
print("  conflicts: {solver.numconflicts()}")
print("  branches : {solver.numbranches()}")
print("  wall time: {solver.wallTime()} s")
print("  sol found: {solution_printer.solution_count()}")

if __name__ == '__main__':
    main()
```

The output of the code is:

```
Sol 0-5 8-6 0-2 8-1 8-8 8-8 8-2

Statistics
status : OPTIMAL
conflicts: 0
branches : 0
wall time: 0.004171268 s
sol found: 1
```

RESULT: The Crypt Arithmetic puzzle has been solved successfully using Python

Date: 11-02-2022

To find the longest path that one can go through in this grid using DFS, given a binary tree find the depth of binary tree using BFS.

ALGORITHM(BFS):

- Step 1: SET STATUS = 1 (ready state) for each node in G
- Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting for state)
- Step 3: Repeat Steps 4 and 5 until QUEUE is empty
- Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).
- Step 5: Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set
 - their STATUS = 2
 - (waiting state)
- [END OF LOOP]
- Step 6: EXIT

```

DFS.py x DFS.py x bash - "p-172-34-6-00" x
thameschl_12:~/environment/R4191180800W15/11-02-2012 $ python3 DFS.py
Homepage
AboutAuthor
RecipesIndex
Summary
Contact
Veg
BreakFastIndex
LunchIndex
DinnerIndex
Dill
Dosa
RiceVariety
Lambat
Card
Chappathi
Bann
Phulka
AlooKutthaMasala
Time for DFS : 0.00628785594921818125 seconds
thameschl_12:~/environment/R4191180800W15/11-02-2012 $

```


DFS

ALGORITHM(DFS):

The steps involved in the BFS algorithm to explore a graph are given as follows -

- **Step 1:** SET STATUS = 1 (ready state) for each node in G
- **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting for state)
- **Step 3:** Repeat Steps 4 and 5 until STACK is empty
- **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
- **Step 5:** Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]
- **Step 6:** EXIT

OUTPUT:



```
thhemath_12@ip-172-31-8-66:~$ python3 BFS.py
Homepage
AboutUs.htm
Summary
Contact
RecipesIndex
veg
BreakfastIndex
Idlis
Dosa
LunchIndex
RiceVariety
Curd
DinnerIndex
Chappathi
Rice
Phulkas
Alcohol.htm
Time for DFS : 0.0001827504875927344 seconds
thhemath_12@ip-172-31-8-66:~$
```

RESULT: Implementation and analysis of BFS and DFS have been completed using python.

Experiment 5: Developing Best first search and A* algorithm for real-world problem

Date: 11-02-2022

AIM:

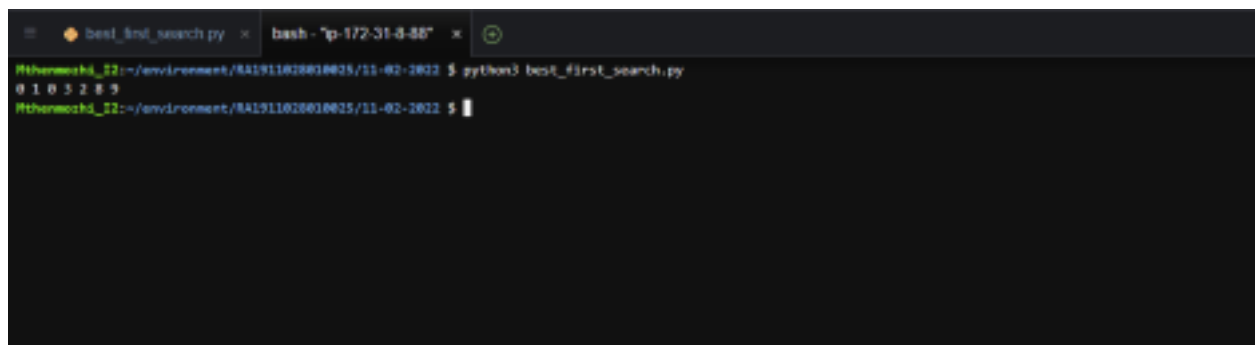
Given a 2D grid having several obstacles, start from the source cell and find a path to the goal cell using the A-star algorithm.

BEST FIRST SEARCH

ALGORITHM:

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until the GOAL node is reached
 1. If the OPEN list is empty, then EXIT the loop returning 'False'
 2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
 3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
 4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
 5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function $f(n)$

OUTPUT:



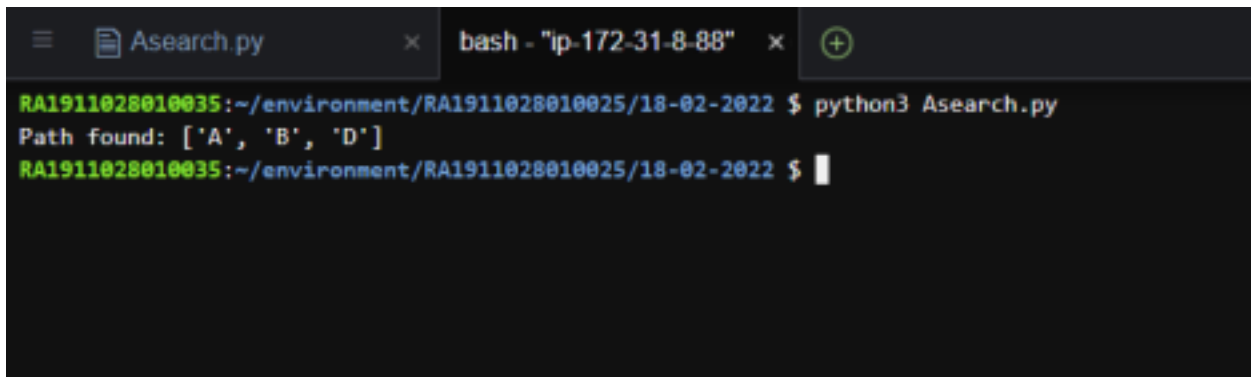
```
best_first_search.py x bash - "p-172-31-8-88" x
#thermashd_12:~/environment/BA1911020010025/11-02-2022 $ python3 best_first_search.py
0 1 0 5 2 8 9
#thermashd_12:~/environment/BA1911020010025/11-02-2022 $
```

A* SEARCH ALGORITHM

ALGORITHM:

- Step 2: If the OPEN list is empty, Stop and return failure.
- Step 3: Remove the node n , from the OPEN list which has the lowest value of $h(n)$, and place it in the CLOSED list.
- Step 4: Expand the node n , and generate the successors of node n .
- Step 5: Check each successor of node n , and find whether any node is a goal node or not. If any successor node is the goal node, then return success and terminate the search, else proceed to Step 6.
- Step 6: For each successor node, the algorithm checks for evaluation function $f(n)$, and then check if the node has been in either the OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list.
- Step 7: Return to Step 2.

OUTPUT:



```
RA1911028010035:~/environment/RA1911028010025/18-02-2022 $ python3 Asearch.py
Path found: ['A', 'B', 'D']
RA1911028010035:~/environment/RA1911028010025/18-02-2022 $
```

RESULT: Implementation of Best first search and A* algorithm is completed using python

Experiment 6: Implementation of Min-Max algorithm for an application

Date: 18-02-2022

AIM:

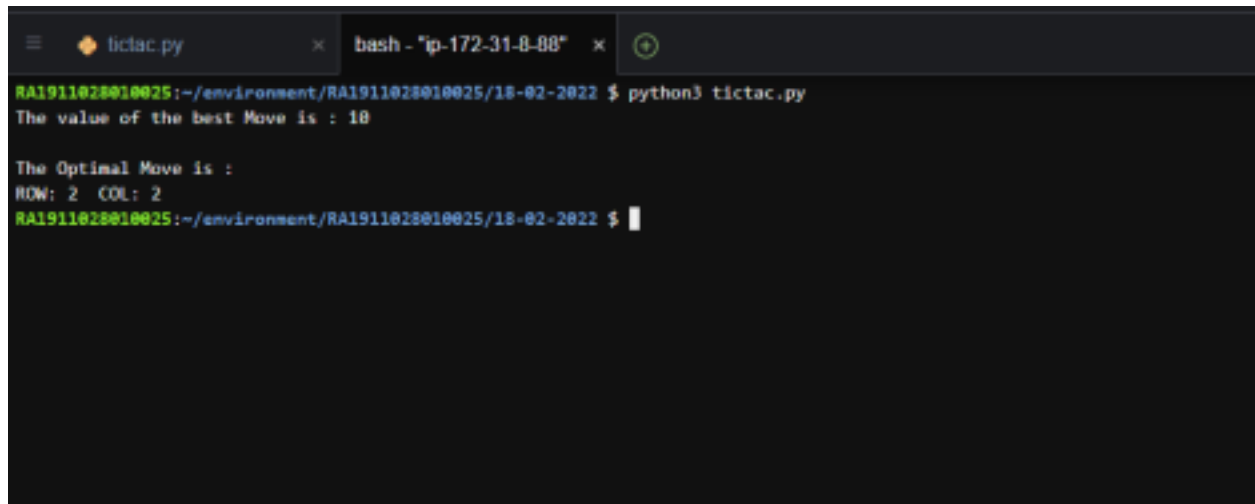
TicTac Toe Problem

ALGORITHM:

Perfect play for a deterministic, 2-player game

- Max tries to maximize its the score
- Min tries to minimize Max's score (Min)
- Goal: move to the position of highest minimax value
- Identify the best achievable payoff against the best play

OUTPUT:



```
tictac.py x bash - "ip-172-31-0-88" x +
RA1911028010025:~/environment/RA1911028010025/18-02-2022 $ python3 tictac.py
The value of the best Move is : 10

The Optimal Move is :
ROW: 2 COL: 2
RA1911028010025:~/environment/RA1911028010025/18-02-2022 $
```

RESULT: Implementation of Min-Max Algorithm for an application is completed using python.

Experiment 7a,7b: Implementation of Unification algorithm & Implementation of Resolution

Date: 28-03-2022

UNIFICATION

AIM:

To implement a unification algorithm

ALGORITHM:

Initialize the substitution set to be empty.

Recursively unify atomic sentences:

Check for Identical expression matches.

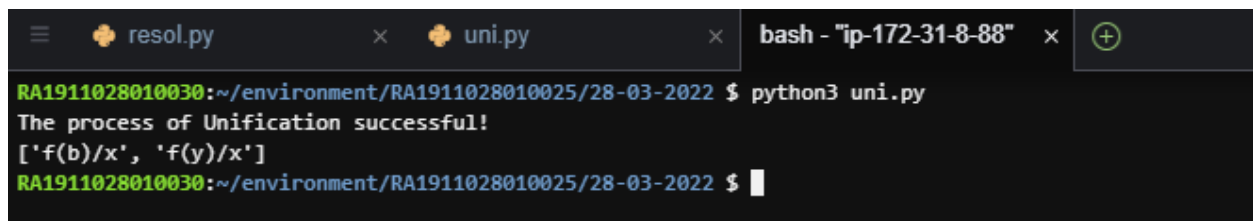
If one expression is a variable v I, and the other is a term that does not contain variable v I, then:

Substitute t_i / v I in the existing substitutions

Add t_i / v I to the substitution setlist.

If both the expressions are functions, then the function name must be similar, and the number of arguments must be the same in both expressions.

OUTPUT:



```
RA1911028010030:~/environment/RA1911028010025/28-03-2022 $ python3 uni.py
The process of Unification successful!
['f(b)/x', 'f(y)/x']
RA1911028010030:~/environment/RA1911028010025/28-03-2022 $
```

RESOLUTION

AIM:

To implement a resolution algorithm.

ALGORITHM:

Resolution is used, if there are various statements given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

- 1) Conversion of facts into first-order logic.
- 2) Convert FOL statements into CNF
- 3) Negate the statement which needs to prove (proof by contradiction)

4) Draw a resolution graph (unification).

OUTPUT:

```
RA1911028010030:~/environment/RA1911028010025/28-03-2022 $ python3 resol.py  
['TRUE', 'FALSE']  
RA1911028010030:~/environment/RA1911028010025/28-03-2022 $
```

RESULT: Implementation of the Unification algorithm & Implementation of Resolution has been completed.

Experiment 8: Implementation of uncertain methods for an application

Date: 04-04-2022

BAYES THEOREM

AIM:

Implementation of uncertain methods for an application - Bayes Theorem

ALGORITHM:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

where A and B are events and $P(B) \neq 0$.

- We are trying to find the probability of event A, given that event B is true. Event B is also termed as evidence.
- $P(A)$ is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- $P(A|B)$ is a posteriori probability of B, i.e. probability of event after evidence is seen.

OUTPUT:

```
+ Code + Text
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the Naive Bayes model on the training set
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
ac = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
cm

array([[55,  2],
       [ 4, 20]])
```

RESULT: Implementation of uncertain methods like the Bayes Theorem is completed.

Experiment 9: Implementation of learning algorithms for an application

Date: 04-04-2022

AIM

To Implement learning algorithms for an application using Python.

UNSUPERVISED LEARNING

- **K-NEAREST NEIGHBOR(KNN)**

ALGORITHM:

The K-NN working can be explained based on the below algorithm:

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready

OUTPUT:



- **K-MEANS CLUSTERING**

ALGORITHM:

The working of the K-Means algorithm is explained in the below steps:

- Step-1: Select the number K to decide the number of clusters.
- Step-2: Select random K points or centroids. (It can be other from the input dataset).
- Step-3: Assign each data point to its closest centroid, which will form the predefined K clusters.
- Step-4: Calculate the variance and place a new centroid in each cluster.
- Step-5: Repeat the third steps, which means reassigning each datapoint to the new closest centroid of each cluster.
- Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.
- Step-7: The model is ready.

OUTPUT:



SUPERVISED LEARNING

- **LINEAR REGRESSION**

ALGORITHM:

$$Y = a_0 + a_1x + \varepsilon$$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

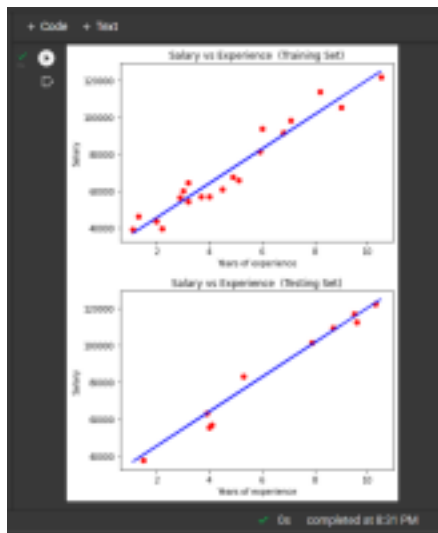
a₀= intercept of the line (Gives an additional degree of freedom)

a₁ = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

OUTPUT:



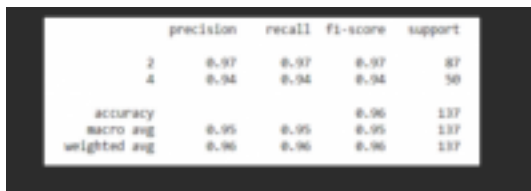
- **LOGISTIC REGRESSION**

ALGORITHM:

Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.

OUTPUT:



	precision	recall	f1-score	support
2	0.97	0.97	0.97	87
4	0.94	0.94	0.94	50
accuracy			0.96	137
macro avg	0.95	0.95	0.95	137
weighted avg	0.96	0.96	0.96	137

RESULT: Implementation of learning algorithms for an application was completed using python.

Experiment 10: To Implement NLP programs

Date: 12/04/2022

AIM

To Implement Natural Language Processing algorithms using Python.

ALGORITHM

There are the following five phases of NLP:

Phases of NLP

1. Lexical Analysis and Morphological

The first phase of NLP is the Lexical Analysis. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. It divides the whole text into paragraphs, sentences, and words.

2. Syntactic Analysis (Parsing)

Syntactic Analysis is used to check grammar and word arrangements and shows the relationship among the words.

Example: Agra goes to the Poonam

In the real world, Agra goes to the Poonam, which does not make any sense, so this sentence is rejected by the Syntactic analyzer.

3. Semantic Analysis

is concerned with the meaning representation. It mainly focuses on the literal meaning of words, phrases, and sentences.

4. Discourse Integration

depends upon the sentences that proceed and also invokes the meaning of the sentences that follow it.

5. Pragmatic Analysis

Pragmatic is the fifth and last phase of NLP. It helps you to discover the intended effect by applying a set of rules that characterize cooperative dialogues.

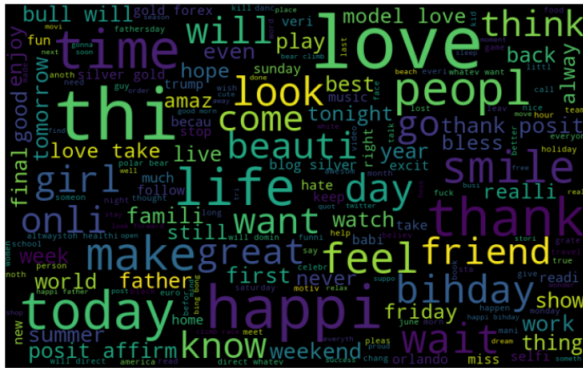
OUTPUT:

In [15]:

```
# visualize the frequent words
all_words = " ".join([sentence for sentence in df['clean_tweet']])

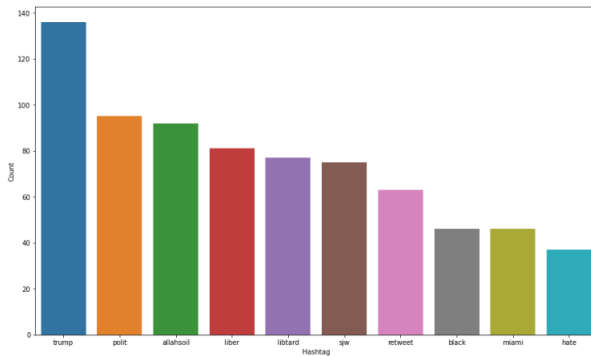
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=500, random_state=42, max_font_size=100).generate(a

# plot the graph
plt.figure(figsize=(15,8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [27]:

```
# select top 10 hashtags
d = d.nlargest(columns='Count', n=10)
plt.figure(figsize=(15,9))
sns.barplot(data=d, x='Hashtag', y='Count')
plt.show()
```



RESULT: Implementation of Natural Language Processing completed using Python.

Experiment 11: Applying deep learning methods to solve an application

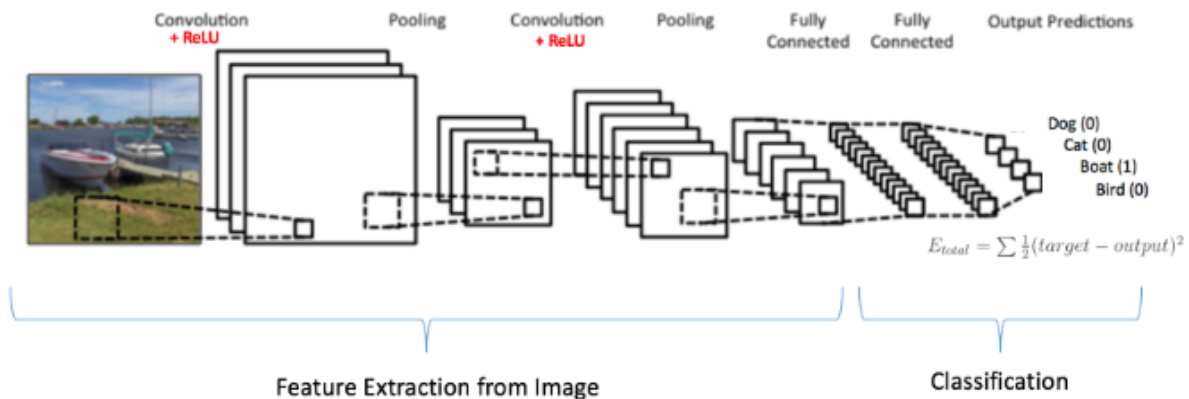
Date: 12-04-2022

AIM:

To implement deep learning methods to solve an application

ALGORITHM:

Every image is an cumulative arrangement of dots (a pixel) arranged in a special order. If you change the order or color of a pixel, the image would change as well.



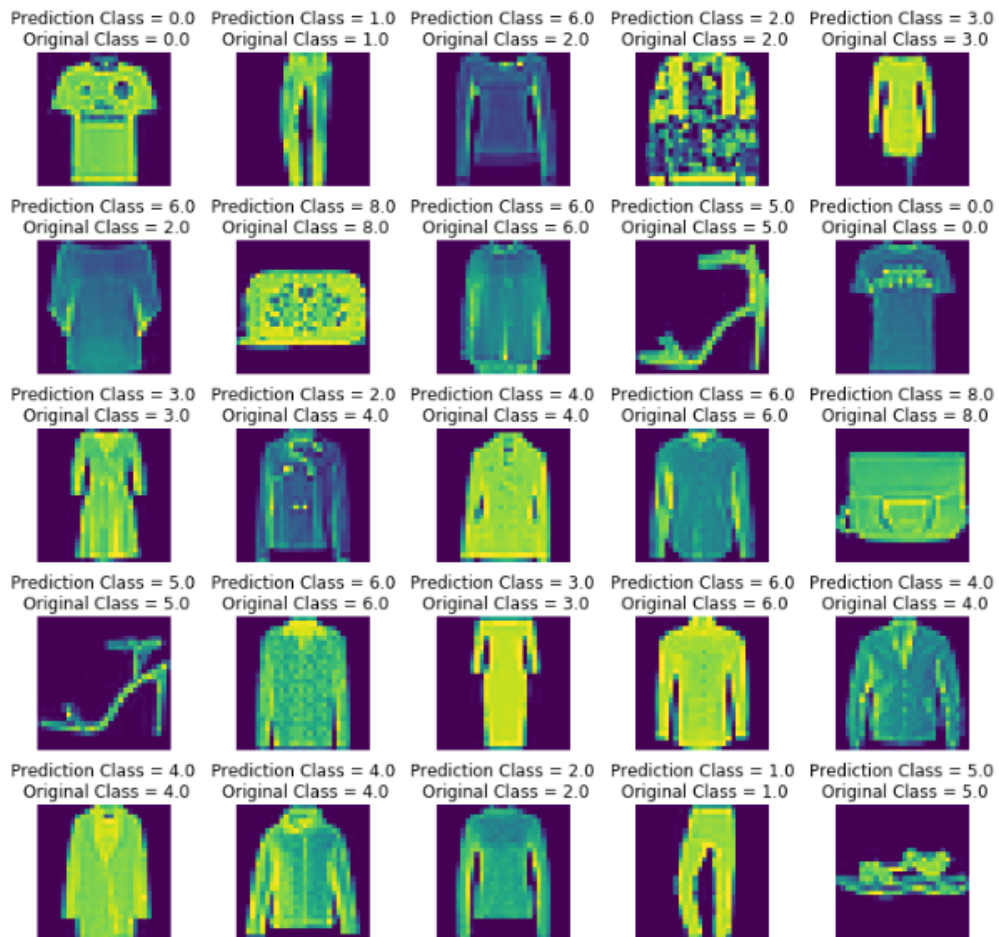
Three basic components to define a basic convolutional neural network.

OUTPUT:

```
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel()

for i in np.arange(0, L * W):
    axes[i].imshow(x_test[i].reshape(28,28))
    axes[i].set_title(f"Prediction Class = {predicted_classes[i]:0.1f}\n Original Class = {y_test[i]:0.1f}")
    axes[i].axis('off')

plt.subplots_adjust(wspace=0.5)
```



RESULT: Implementation of deep learning methods successfully completed using python.