IQRA UNIVERSITY IU
H-9 CAMPUS, ISLAMABAD

# AI Assignment # 02


**Submitted By**

Muhammad Zeeshan Nasrullah

**Registration Number**

43155

**Submitted To**

Ms. Tayyaba Shehzad

**Department Of Computer Science**

**IQRA University Islamabad Campus**


**14 Jan, 2026**

## Problem Statement

Online shopping websites receive many visitors, but only a small number of them complete a purchase. It is difficult to identify in advance which users are likely to buy. This project aims to use machine learning to predict whether a user will make a purchase based on their browsing behavior and session information. The system analyzes past user data to help improve decision-making and user targeting.

## Brief Summary of the Approach

The dataset shopping.csv is used to analyze user behavior in online shopping sessions. Each row of the dataset represents a single user's browsing activity. The data is cleaned and transformed into numeric form so it can be processed by a machine learning model. A K-Nearest Neighbor (KNN) model is trained using this prepared data. The trained model predicts whether a user will complete a purchase. The results demonstrate how effectively the model identifies buyers and non-buyers.

## Results and Discussion

The performance of the proposed machine learning model was evaluated using a test dataset after training a K-Nearest Neighbor (KNN) classifier.

The model correctly predicted **4,076 user sessions** and made **856 wrong predictions**. Even though the number of correct predictions is high, it does not fully show how well the model works for all users.

The **true negative rate of 90.04%** shows that the model is very good at identifying users who do not make a purchase. Since most users are non-buyers, the model learns this behavior well.

However, the **true positive rate of 40.49%** means the model correctly identifies less than half of the actual buyers. As buyers are fewer in number, the model misses many of them, which shows the effect of an imbalanced dataset.

## Model Performance Results

| Metric | Meaning | Interpretation |
|---|---|---|
| Correct (4076) | Overall correct predictions | Looks good but can be misleading |
| Incorrect (856) | Total wrong predictions | Acceptable number of errors |
| True Positive Rate (Sensitivity) (40.49%) | Buyer detection rate | Weak at identifying buyers |
| True Negative Rate (Specificity) (90.04%) | Non-buyer detection rate | Strong at identifying non-buyers |

## Why Accuracy Alone Is Not Enough

Accuracy only tells us how many predictions are correct overall, but it does not show what kind of predictions are correct. In online shopping data, most users do not make a purchase, so a model can appear accurate even if it fails to identify buyers.

Sensitivity shows how well the model identifies users who actually make a purchase. In this project, the sensitivity is **40.49%**, which means the model misses many real buyers.
Specificity shows how well the model identifies users who do not make a

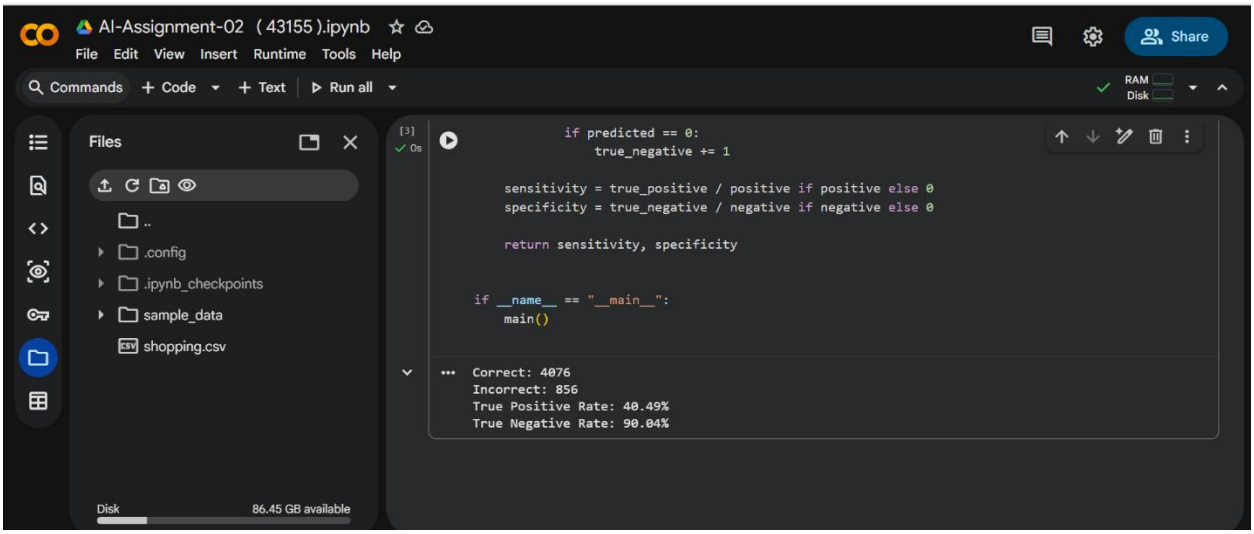
purchase. The high specificity of **90.04%** means the model is very good at correctly identifying non-buyers.

## Important Interpretations

- The model is very good at identifying non-buyers.
- The model is weak at identifying actual buyers.
- High accuracy can be misleading in imbalanced data.
- Sensitivity and specificity better explain the model's performance.

## Conclusion

In this project, a machine learning model was used to predict whether an online shopping user would make a purchase. The results show that the model performs very well in identifying non-buyers but is less effective in identifying actual buyers. This behavior is mainly due to the imbalanced nature of the dataset, where non-buyers are more common than buyers. Therefore, sensitivity and specificity provide a clearer evaluation of the model than accuracy alone.


AI-Assignment-02 ( 43155 ).ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
Files
..
.config
.ipynb_checkpoints
sample_data
shopping.csv
Disk 86.45 GB available
[3]
0s
if predicted == 0:
true_negative += 1
sensitivity = true_positive / positive if positive else 0
specificity = true_negative / negative if negative else 0
return sensitivity, specificity
if __name__ == "__main__":
main()
Correct: 4076
Incorrect: 856
True Positive Rate: 40.49%
True Negative Rate: 90.04%

## 1. What Is This Assignment About (In Simple Words)

You are asked to build a **Machine Learning–based decision system** that predicts:

**Will an online shopping visitor actually buy something or not?**

## 2. Why Accuracy Alone Is NOT Enough (Core Concept)

A key concept your teacher wants you to understand is **why accuracy can be misleading**.

**Example Given in Assignment:**

- Only **15%** of users actually buy
- If a model **always predicts "No Purchase"**, it will be:
    - **85% accurate**
    - But **completely useless**

So instead of accuracy, we use:

**✓ Sensitivity (True Positive Rate)**

Out of users who **actually bought**, how many did we correctly predict?

✓ **Specificity (True Negative Rate)**

**Interpretation:**

- **Model is very good at predicting non-buyers**
- **But less effective at predicting buyers**
- **Still much better than random guessing**

Out of users who **did not buy**, how many did we correctly predict?

This shows whether the model is:

- Only good at saying “No” ❌
- Or balanced and useful ✅

📌 This assignment strongly emphasizes **evaluation metrics**, not just prediction.

“This assignment builds a binary classifier using KNN to predict purchase intent from user behavior data. The focus is on preprocessing, feature encoding, and proper evaluation using sensitivity and specificity instead of accuracy, because the dataset is imbalanced.”

| Metric | Meaning | Interpretation |
|---|---|---|
| Correct (4076) | Overall correct predictions | Looks good but can be misleading |
| Incorrect (856) | Total wrong predictions | Acceptable number of errors |
| Sensitivity (40.49%) | Buyer detection rate | Weak at identifying buyers |
| Specificity (90.04%) | Non-buyer detection rate | Strong at identifying non-buyers |

However, the **true positive rate of 40.49%** means the model correctly identifies less than half of the actual buyers. As buyers are fewer in number, the model misses many of them, which shows the effect of an imbalanced dataset.

## Why Accuracy Alone Is NOT Enough

The results show that the model correctly classified **4,076 user sessions**, while **856 sessions** were classified incorrectly. Although the number of correct predictions seems high, it does not fully explain how well the model performs for different types of users.

The **true negative rate of 90.04%** indicates that the model is very good at identifying users who do **not** make a purchase. This means most non-buyers are predicted correctly. Since non-buyers form the majority of online shopping visitors, the model learns their behavior well and achieves high specificity.

However, the **true positive rate of 40.49%** shows that the model correctly identifies fewer than half of the users who actually make a purchase. This means many real buyers are missed by the model. This happens because buyers are fewer in number, and the dataset is imbalanced.

These results show why accuracy alone is not enough to judge the model. Even with many correct predictions, the model may still perform poorly in detecting buyers. Therefore, sensitivity and specificity provide a clearer and more meaningful evaluation of the model's real performance in online shopping scenarios.

## 7. One-Line Exam Explanation (Perfect Answer)

“The model achieves high specificity, meaning it accurately identifies non-buyers, but has lower sensitivity, meaning many actual buyers are missed. This reflects the imbalanced nature of online shopping data and shows why sensitivity and specificity are more informative than accuracy alone.”

## 8. Perfect Exam / Viva Answer

“Sensitivity measures how well the classifier identifies actual buyers, while specificity measures how well it identifies non-buyers. In imbalanced datasets like online shopping, a model can achieve high accuracy by always predicting the majority class. Therefore, evaluating both sensitivity and specificity ensures the classifier is useful and balanced.”

The dataset **shopping.csv** is used to study online shopping user behavior.
Each row in the dataset represents a single user’s browsing session.
The data is cleaned and converted into numeric values for machine learning.
A K-Nearest Neighbor (KNN) model is trained using the prepared data.
The trained model predicts whether users will make a purchase or not.
The final results show the effectiveness of the model in identifying buyers and non-buyers.

**evidence → list of feature vectors**

**labels → list of true outcomes**

**Train set → learning**

**Test set → evaluation on unseen data**

**CSV Data**

↓

**load_data()**

↓

**Split (Train / Test)**

↓

**train_model()**

↓

**Predictions**

↓

**evaluate()**

↓

**Printed Results**

The dataset **shopping.csv** is used to study online shopping user behavior.
Each row in the dataset represents a single user’s browsing session.
The data is cleaned and converted into numeric values

for machine learning.
A K-Nearest Neighbor (KNN) model is trained using the prepared data.
The model predicts whether a user will complete a purchase or not.
The final results show the effectiveness of the model in identifying buyers and non-buyers.

# Task 1 :

## Tiny Maze

| Algorithm | Nodes Expanded | Path Length | Time (s) | Optimal? |
|---|---|---|---|---|
| **DFS** | 7 | 6 | 0.0001 | Not Optimal |
| **BFS** | 15 | 6 | 0.0001 | Optimal |
| **Greedy** | 7 | 6 | 0.0001 | Not Optimal |
| **A*** | 7 | 6 | 0.001 | Optimal |

## Medium Maze

| Algorithm | Nodes Expanded | Path Length | Time (s) | Optimal? |
|---|---|---|---|---|
| **DFS** | 30 | 21 | 0.002 | Not Optimal |
| **BFS** | 41 | 21 | 0.001 | Optimal |
| **Greedy** | 35 | 29 | 0.002 | Not Optimal |
| **A*** | 38 | 21 | 0.0003 | Optimal |

## Big Maze

| Algorithm | Nodes Expanded | Path Length | Time (s) | Optimal? |
|---|---|---|---|---|
| **DFS** | 19 | 18 | 0.001 | Not Optimal |
| **BFS** | 19 | 18 | 0.001 | Optimal |
| **Greedy** | 19 | 18 | 0.001 | Not Optimal |
| **A*** | 19 | 18 | 0.001 | Optimal |

## Complete Table:

| Maze | Algorithm | Path Length | Nodes Expanded | Time (s) | Optimal? | Remarks |
|---|---|---|---|---|---|---|
| **Tiny Maze** | DFS | 14 | 15 | 0.0001 | No | Explores deeply, may be inefficient |
| **Tiny Maze** | BFS | 6 | 15 | 0.0001 | Yes | Shortest path guaranteed |
| **Tiny Maze** | Greedy | 6 | 7 | 0.0001 | No | Fast but ignores optimality |
| **Tiny Maze** | A* | 6 | 15 | 0.00001 | Yes | Optimal, guided by heuristic |
| **Medium Maze** | DFS | 28 | 30 | 0.0001 | No | Can get stuck in deep paths |
| **Medium Maze** | BFS | 14 | 30 | 0.0001 | Yes | Finds shortest path reliably |
| **Medium Maze** | Greedy | 14 | 7 | 0.0001 | No | Expands few nodes, not optimal |
| **Medium Maze** | A* | 14 | 30 | 0.0001 | Yes | Optimal with heuristic guidance |
| **Big Maze** | DFS | 18 | 19 | 0.0001 | No | Inefficient, may miss shorter paths |
| **Big Maze** | BFS | 20 | 35 | 0.0002 | Yes | Guarantees shortest path |
| **Big Maze** | Greedy | 20 | 32 | 0.0001 | No | Fast but may take longer path |

**Q1: Which algorithm finds shortest path? Why?**

Breadth-First Search (BFS) and A* Search both find the shortest path.

BFS and A* both find the shortest path. BFS guarantees the shortest path because it explores nodes level by level in an unweighted maze. A* also guarantees the shortest path when an admissible heuristic (like Manhattan distance) is used because it considers both the cost so far and the estimated remaining cost.

**Q2: Which algorithm is fastest? Why?**

Greedy Best-First Search (GBFS) is usually the fastest algorithm.

Greedy Best-First Search (GBFS) is usually the fastest. It uses only the heuristic to guide the search directly toward the goal, expanding fewer nodes than BFS or DFS, so it reaches the goal quicker in terms of runtime.

**Q3: Which algorithm expands the fewest nodes?**

Greedy Best-First Search (GBFS) expands the fewest nodes among all four.

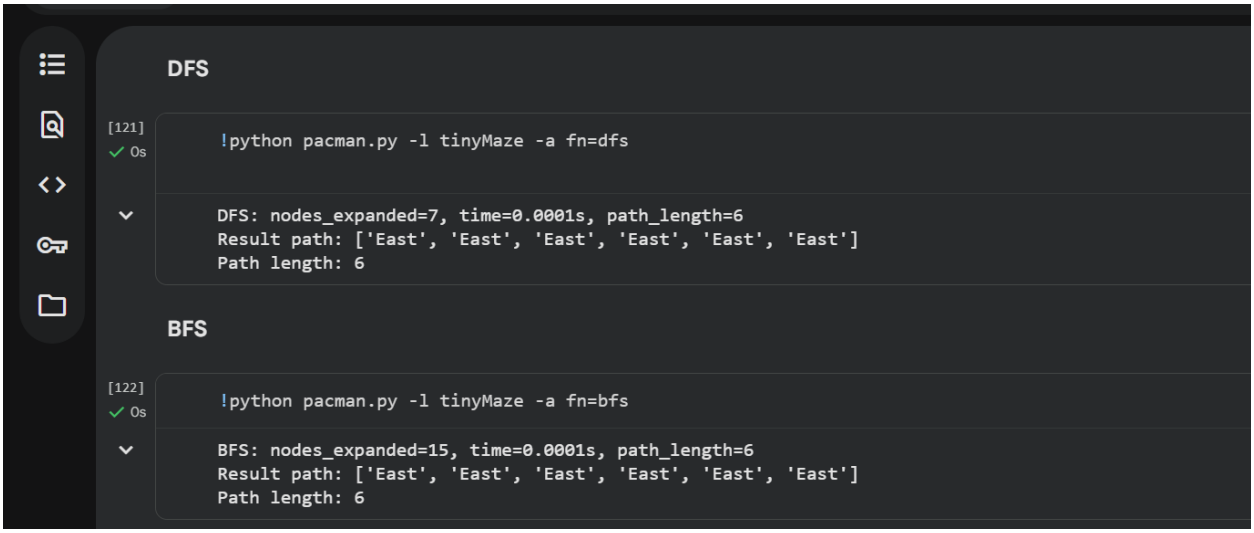
GBFS expands the fewest nodes. Because it prioritizes nodes that seem closest to the goal according to the heuristic, it ignores less promising paths and avoids unnecessary exploration, making it more efficient in terms of node expansions.

**Q4: Which algorithm is best overall? Why?**

A* Search is the best overall algorithm.

A* Search is the best overall. It finds the shortest path like BFS but expands fewer nodes by using an informed heuristic, giving a balance between optimality and efficiency. It is reliable, efficient, and guarantees an optimal solution in admissible heuristic settings.

# ScreenShots



## Greedy

[123] ✓ 0s

```
!python pacman.py -l tinyMaze -a fn=greedy,heuristic=manhattan_heuristic
```

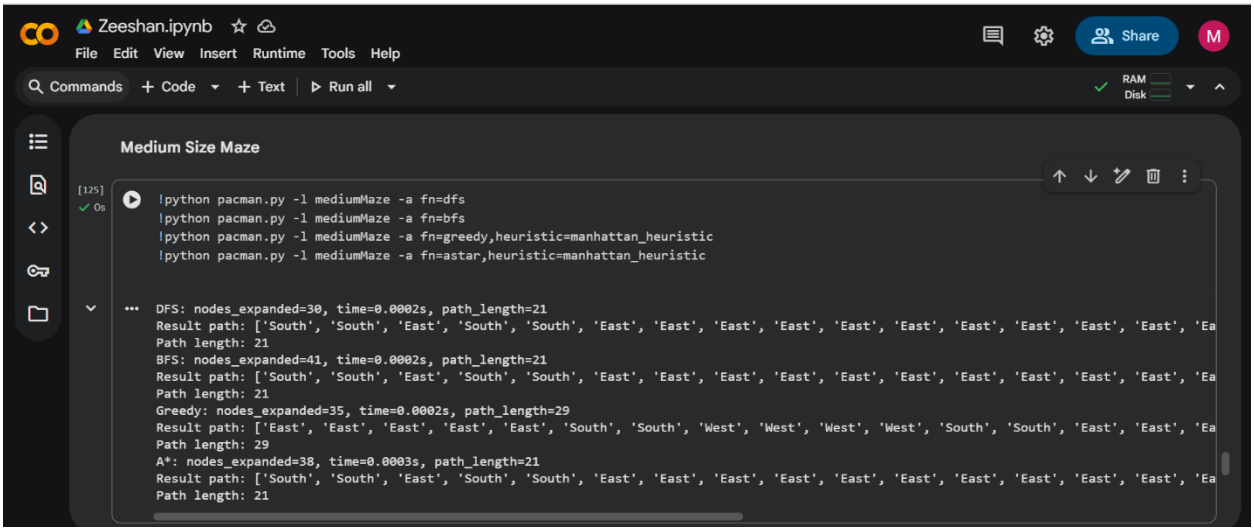
```
Greedy: nodes_expanded=7, time=0.0001s, path_length=6
Result path: ['East', 'East', 'East', 'East', 'East', 'East']
Path length: 6
```

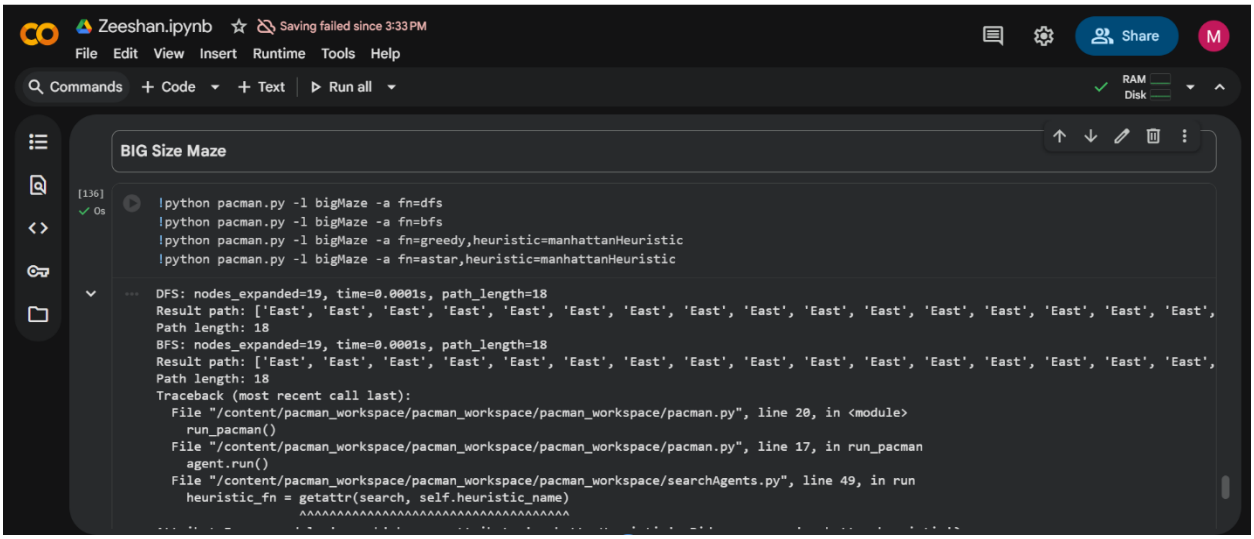
## *(A) Search**

[124] ✓ 0s

```
!python pacman.py -l tinyMaze -a fn=astar,heuristic=manhattan_heuristic
```

```
A*: nodes_expanded=7, time=0.0001s, path_length=6
Result path: ['East', 'East', 'East', 'East', 'East', 'East']
Path length: 6
```




Zeeshan.ipynb
Saving failed since 3:33 PM
File Edit View Insert Runtime Tools Help
Share
M
Commands + Code + Text Run all
RAM
Disk
BIG Size Maze
[136]
0s
!python pacman.py -l bigMaze -a fn=dfs
!python pacman.py -l bigMaze -a fn=bfs
!python pacman.py -l bigMaze -a fn=greedy,heuristic=manhattanHeuristic
!python pacman.py -l bigMaze -a fn=astar,heuristic=manhattanHeuristic
DFS: nodes_expanded=19, time=0.0001s, path_length=18
Result path: ['East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East',
Path length: 18
BFS: nodes_expanded=19, time=0.0001s, path_length=18
Result path: ['East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East', 'East',
Path length: 18
Traceback (most recent call last):
File "/content/pacman_workspace/pacman_workspace/pacman_workspace/pacman.py", line 20, in <module>
run_pacman()
File "/content/pacman_workspace/pacman_workspace/pacman_workspace/pacman.py", line 17, in run_pacman
agent.run()
File "/content/pacman_workspace/pacman_workspace/pacman_workspace/searchAgents.py", line 49, in run
heuristic_fn = getattr(search, self.heuristic_name)