A Project Report

On

Intelligent Implementation of ReBAC models using Machine Learning

BY

**RUTHIK REDDY CHITTI**

**2022A7PS0204H**

Under the supervision of

**PROF. BARSHA MITRA**

**SUBMITTED IN FULLFILLMENT OF THE REQUIREMENTS OF**

**CS F376: DESIGN PROJECT**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**

**HYDERABAD CAMPUS**

**(MAY, 2025**)

# ACKNOWLEDGMENTS

Firstly, I wish to thank Prof.Barsha Mitra for providing me the opportunity to work on this project, I express my sincere thanks for her guidance and constant supervision, valuable inputs regarding the project, and also for supporting in all aspects for smooth functioning of the project.

I thank my fellow team members of this project, who were wonderful people to work with, for their constant encouragement. Their coordination made this project doable.

RUTHIK REDDY CHITTI 2022A7PS0204H



**Birla Institute of Technology and Science-Pilani,**

**Hyderabad Campus**

**Certificate**

This is to certify that the project report entitled “Intelligent Implementation of ReBAC models using Machine Learning” submitted by Mr. Ruthik Reddy Chitti (ID No.2022A7PS0204H) in

Fulfillment of the requirements of the course CS F376, Design Project Course, embodies the work done by him under my supervision and guidance.

**Date: 06-05-2025 Prof. Barsha Mitra)**

BITS- Pilani, Hyderabad Campus

# ABSTRACT

Access control modelling is important in the modern computing environment in defining security policies for resources. Studying various access control models is essential for efficient saccess management. Increasing complexity made it more obvious that more intelligent access control models should be developed. As we have traditional methods that do not satisfy our needs, we need to adapt to changes involving security challenges, data privacy, and easy access. Also, further owing to the growth of AI, this study helps as a base to develop further.

RBAC(Role-Based Access Control) and ABAC(Attribute-Based Access Control) are earlier methods that are widely used, but they couldn’t meet our requirements with dynamic environments where data changes in realtime. So, we started extending the concept to Relationship access Control (ReBAC). This model works on the relationships between users and resources to grant access. This project focusses on researching more on ReBAC models and their application in real-time. Further, policy languages like B-Language and other policy mining techniques are studied. Finally, we understand ReBAC models with their application and its advantage over previous models like RBAC and ABAC.

Later, we worked on converting Datasets(ReBAC) to extract policies from them. On the basis of those, testing and data scrutiny reveal some patterns among policies. Then, some methods are used to increase accuracy and other parameters. Later, we also found out that policies were inconsistent. Inconsistent policies are removed to and datasets are tested again. Overall, we get policies and then decision trees for the dataset. This way, we get our policy extraction done.

Then, from the ReBAC datasets, we built a decision tree, and then policies from the tree for each dataset. We worked on three datasets: Uni 1\_ReBAC, Uni 2\_ReBAC, and Company\_ReBAC. Later, based on the data we extracted false positive policies. Now, we start making analysis on misclassified samples, where clustering(unsupervised learning) is used in the analysis.

After getting similarities among mis-classified samples, we now try to relate these observations with false positives and try to get a view on false positives and also datapoints leading to such things.

**CONTENTS**

Title page…………………………………………………….……..…………….1

Acknowledgements………………………………………………..……………..2

Certificate……………………………………………………………..………....3

Abstract……………………………………………………………….………....4

1. ABAC……….……………………………………………………..………....6

2.ReBAC………………………………………………….……………………..8

1. Abstract view of UURAC …….…………………………………..………...9
2. B Language………………………………………………………………….14
3. RBAC Datasets to ReBAC Datasets…………………………………………16
4. Policy Extraction…………………………………………………………….20

7. Clustering Among Misclassified Samples…………………………………. 28

8. Experimental Results………………………………………………………..34

Conclusion……………………………………………………………………..36

References……………………………………………………………………..37

# Attribute-Based Access Control(ABAC)

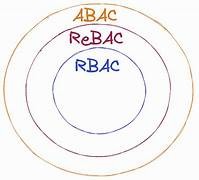
The attribute-based access control (ABAC) model uses attributes for access, and decisions are made based on the results.

**Key Words:**

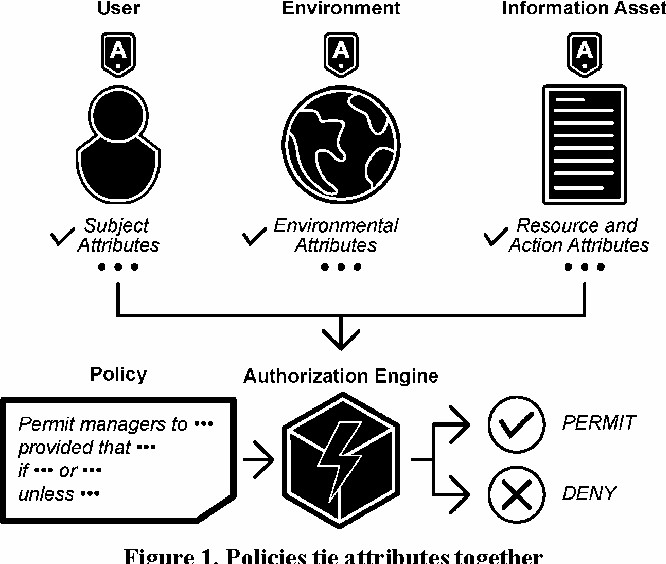
1. **Logical Object/Resource**: The resources requiring access and examples are directories, folders, etc**.**
2. **Subject:** Entity requesting to operate on object.

 The subject can be one of two types: A person or an NPE(Non-Person entity).

1. **Policy:** They convey Rules. Which are used for access control.



In ABAC, attributes are given as input for access control. The attributes mainly consist of Subject, Object, Environment, and Policies. The subject attributes come from various sources. Local administration point where attributes are managed locally for smaller areas, whereas there are enterprise levels for global access. Similarly, for Object attributes, there are local(For local data control) and Enterprise Object Management(global data control).



Policies are of various types, and here it is brief descriptions about them.

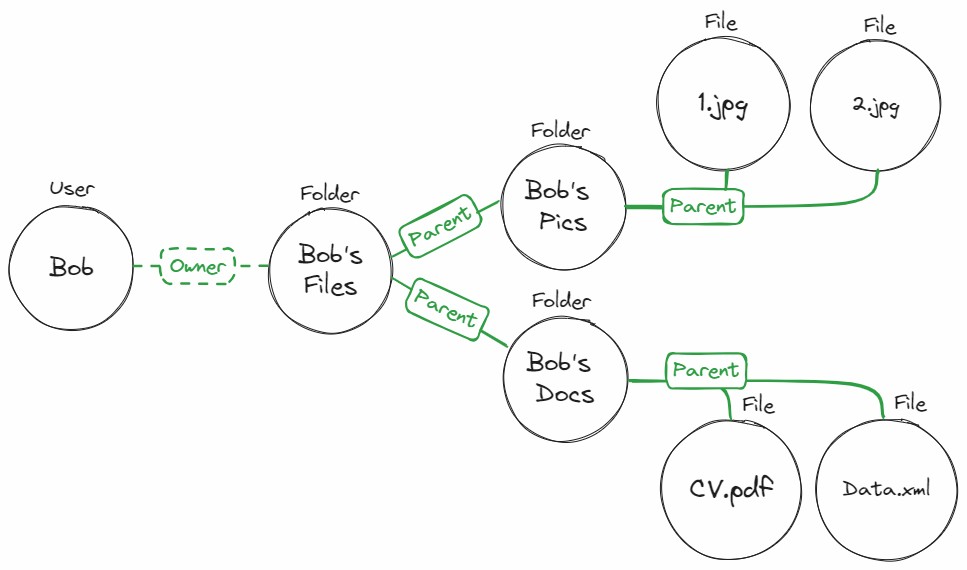
* 1. **NLPs:** These policies are human-readable specifying access rules. This must be translated into DPs for enforcement.
  2. **DPs(Digital policy):** These policies are machine-enforceable and directly control access decisions.
  3. **MPs(Meta Policy):** Policies that manage DPs resolve conflicts and assign priorities.
  4. **DPM(Digital Policy Management):**The overall system for managing DPs and DPs in an enterprise, ensuring they are implemented, updated, and deconflicted properly.

# Relationship-based access control (ReBAC)

Relationship-based access control (ReBAC) grants permissions based on relationships between resources, users, and other entities.

**Types of Relationship:**

1. **Data ownership:** Users have permission to edit posts they created.
2. **Parent-child** (e.g., organization-repository hierarchy)
3. **Groups** (e.g., team memberships)
4. **Hierarchies** (e.g., nested organizational structures)



This figure shows the relationships clearly, and then we design policies based on the relationships between them for access control.

ReBAC is a subset of Attribute-Based Access Control(ABAC) and is also viewed as an extension of Role-Based Access Control (RBAC). This approach helps us to be more intuitive and maintains logic perfectly, but while implementing relationships must be defined properly. The users and resources are defined as nodes in graph structure where the edges define the relationship between them and based on policies we decide the authorization of resources to the incoming request from node.

## ABSTRACT VIEW OF UURAC

The study on various models started with UURAC from a paper by A User-to-User Relationship-based Access Control Model for Online Social Networks by Cheng, which developed access control in OSNs.

The policy administration is by users and can be both inward and outward. The main crux of UURAC lies here: It is Relationship-Based. The whole case is depicted as a Graph in which users and resources are treated as nodes and connected by edges where actions are defined. The Social Graphs G=<U, E, Σ>.U represents nodes and E as edges where sigma has all the policies defined. Later, algorithms are developed for further processing.

Algorithm-1:

The Access Evaluation algorithm is responsible for giving access to user(ua) to perform action a on target (ut/rt).The algorithm starts with the **Policy Collection Phase**, where it gathers all the relevant policies for the user who wants access (*ua*) and the target resource or user (*ut/rt*). These policies can come from system rules, user-defined permissions, or relationships in social graphs. Once collected, the **Policy Evaluation** step extracts graph rules and applies them to see if the access request meets the necessary conditions.

Next, in the **Execution** phase, the algorithm runs through the policies to determine if access can be granted. Finally, during the **Results**

**Combination** step, all the results are put together. If even one rule is not met, the access request is denied, ensuring the system remains secure.

Here are a few example policies:

* P\_Alice <poke, (ua, (f\*,3))>: This policy allows Alice to "poke" under certain relationship conditions.
* P\_harry <poke-1 (ut, (f\*,2))>: Specifies when Harry can receive a "poke."
* <poke, (ua, (f\*,5))>: A more general rule for the "poke" action with broader use.

Algorithm-2: Path checker.

This algorithm plays a crucial role in translating policy rules into actionable paths using graph theory. It works by converting regular expressions (RegEx) into Deterministic Finite Automata (DFA).The algorithm starts with **Path Initialization**, where it sets up the DFA based on the extracted graph rules. Once initialized, it proceeds to **Graph Exploration**, leveraging **Depth-First Search with Backtracking (DFST)** to traverse possible paths within the graph. This approach ensures that all potential paths are explored, even if it requires backtracking to previously visited nodes. During, the Validation step evaluates if a valid path exists between the nodes given. if exists return TRUE, else FALSE is returned.

Algorithm-3: Depth-First Search with Backtracking (DFST):

The **Depth-First Search with Backtracking (DFST)** algorithm is all about finding a valid path between a starting point (*u*) and a target (*t*) within a graph (*G*). It uses a method called **recursive depth-first search (DFS)**, combined with **backtracking**, to make sure it explores every possible path.

The algorithm works by diving deep into the graph, visiting each connected node one by one. If it hits a dead end or finds an invalid path, it simply backtracks—meaning it goes back to the previous node and tries a different route. This back-and-forth process helps uncover all potential paths through the graph, boosting the chances of finding a valid one.

During the **Validation** step, the algorithm checks if the discovered path matches the rules set by the **Deterministic Finite Automata (DFA)** created in the **Path Checker** algorithm. If it does, the algorithm confirms a valid access path by returning **TRUE**.

To speed things up, the algorithm also uses a **Breadth-First Search (BFS)** approach. Unlike DFS, which goes deep down one path at a time, BFS looks at multiple paths simultaneously. This parallel exploration makes finding valid paths faster and more efficient, especially when dealing with large and complex graphs.

By blending **DFS, backtracking**, and **BFS**, this algorithm strikes a good balance between thoroughness and efficiency, ensuring that it doesn't miss any potential access routes while keeping the process quick and reliable.

Differences Between Algorithm-2 and Algorithm-3:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Algorithm 2** | **Algorithm 3** |
| Purpose | Initializes DFA | Performs actual DFS to final path |
| Method | Calls DFST | Recursively explores graph |
| Input | Graph path, hopcount  , start node, target  node | A node to explore |
| Output | TRUE if a valid path exists, FALSE otherwise | TRUE if a valid path exists, FALSE otherwise |
| Technique | DFA conversion | DFS with backtracking |

.

ReBAC: A Flexible Approach to Access Control

Philip W. L. Fong's work on Relationship-Based Access Control (ReBAC) introduces a great framework for access control in social networks and also can be expanded. The advantages are:

**User-defined policies**: ReBAC empowers users to set their own access rules, rendering a good level of flexibility:

1. **Multiple relationship types**: ReBAC supports various types of connections.
2. **Real-time updates**: The system can handle Dynamic changes..

Key Components of ReBAC

ReBAC uses a graph-based model represented as G = (V, {Ri}i∈I), where:

* V: Set of users
* Ri: Different types of relationship.

>This model allows for reverse relationships and easy addition or removal of connections.

ReBAC incorporates three main operations for managing system states:

1. **Push Operation**: Introduces a new context (c1) with optional initial relationships (Δ), inheriting rules from a parent context (c2).
2. **Pop Operation**: Removes a context, cleaning up associated permissions and relationships.
3. **Edge Operation**: Modifies relationships within a context by adding (Δ1) or removing (Δ2) connections.

These operations allow for dynamic updates to the system, ensuring that access control remains flexible and responsive to changing needs.

**Example:**

* A new emergency department is created in a hospital.
* The emergency department inherits rules from the existing hospital.
* If a doctor can access patient records in the hospital, they can also access them in the emergency department.

**RBAC v/s ReBAC:**

In the **RBAC** control model, permissions are assigned to roles(users are assigned roles). Access is determined based on the role a user has. In **ReBAC**, it extends RBAC by defining access based on the relationship between entities. The major difference here is that access is granted based on who the user is related to.

# B Language

B Language is a policy language designed by Fong. Everything in this language is set in graphs as users at nodes and relationships between them with edges. The language is designed with various rules to make and denote the policies, and various notations are used to make policy-making easy and feasible.

Notations used in the language are expanded and defined more to make us define more policies and also helps us and makes us easy to define more

rules. Let's start with basic notation, (G,u,v)::=ϕ. Now, in this notation, G defines the graph where I is the owner and v is the accessor, and they both possess the relationship prescribed by policy ϕ. Now, let's analyze it carefully by defining more definations or say notations. In the description, T denotes any pair of vertices, implying if (G,u,v)::=T, then both u, the owner, and v the accessor, have a relationship or say are connected without any policy. Then comes ‘a, ’ which means bot accessor and owner are the same. (G, u, v) |= a iff u = v. This explains to us the access is granted if and only if u = v. i.e, the owner and accessor are the same.

Now, let's start making things complicated. <i>φ asserts that the owner has an i-neighbor that is related to the accessor in the manner specified by φ; In other words, that every individual related to the owner via an i relationship is also related to the accessor in a manner specified by φ. G, u, v |= <i> φ iff there exists u ʹ ∈ V such that (u, uʹ ) ∈ Ri and G, uʹ , v |= φ.

Let us look at some examples so that things will be clear.

**Examples:**

**Note:** the idiom “<i> a” asserts that successful accessors must be related to the owner directly via an i relationship

1. **Grant access to the owner’s child. u**(owner)----v(child)

Now, let's say there is a relationship called parent. And, given access is granted to the owner's child, i.e, the owner is the father. We generally describe relationships from an accessor perspective, so here we write a negation. <**-**parent> a saying accessor is child(-parent) of the owner.

1. **“Grant access to grandparents.”**

“<parent> <parent> a”. Here, it is clear and straightforward that the accessor is the grandparent, so it is the parent of the parent, which makes things simple to define policy.

* defining some new rules. <R>φ “there exists a relationship R such that φ is true
* [R] φ “for all relationships R, φ is true

“Grant access to a sibling who is not married.” The given policy is a bit complicated, so we use the above-defined rules. <sibling>(a ∧ [spouse] ⊥) The first part gives us a sibling relationship, while the inner thing makes us the accessor (sibling) has no spouse. or simply for all relationships, a sibling has should not be spouse.

“Grant access to a married sibling.” The given policy defines to have a

relation with a married sibling. <sibling>(a ∧ <spouse> ⊤) there exists a sibling where any of his relation is spouse. So, out of many relationships, the accessor has a relationship type of spouse, and he is the sibling to the owner.

In this way, B language has defined its own notations and started policymaking with supporting complex policies. The notations are many, but here, only some of them are presented.

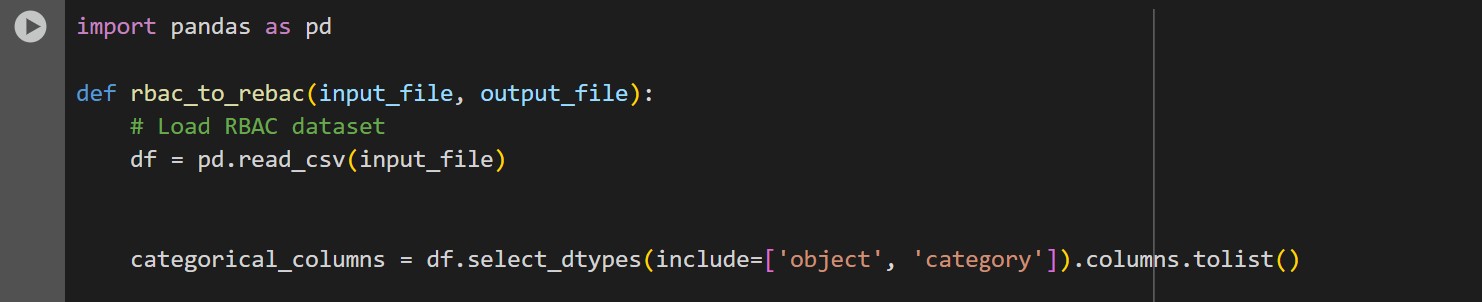
# RBAC Datasets to ReBAC Datasets

The RBAC datasets are converted into ReBAC datasets using a technique from machine language that uses one-hot encoding. The very first thing is we analyze the given dataset and make the count and types of rows in the

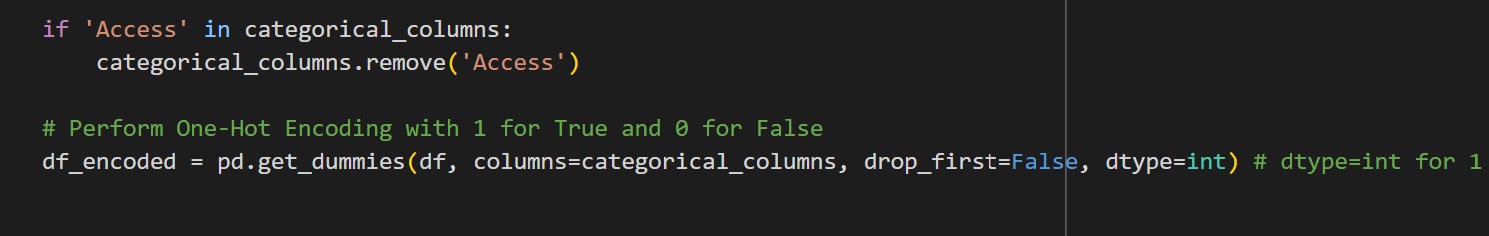
given dataset.



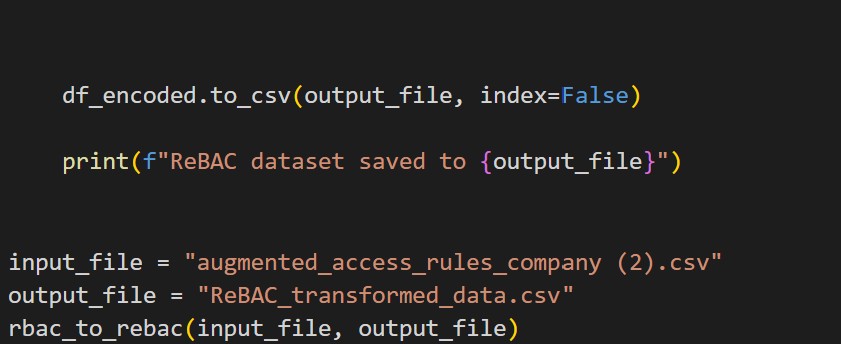
The rows that are defined in the dataset are used to define the relationship. Each unique value in the row is now made into the relationship, and if you find a relationship, it will be represented by 1 else 0. The special value NaN is also treated and made into a special column. So, our entire task is to make such a dataset with the main logic, which is defined as the existence of relationship is denoted by 1 else 0.



So, we start our task by converting given data into a dataframe using the pandas library. Then, columns with text-based(categorical values are selected and converted to a list. The data with dataframe containing columns with text-based(categorical values) are selected and converted to a list(pandas index objects are immutable, tolist () to convert to list for modification.



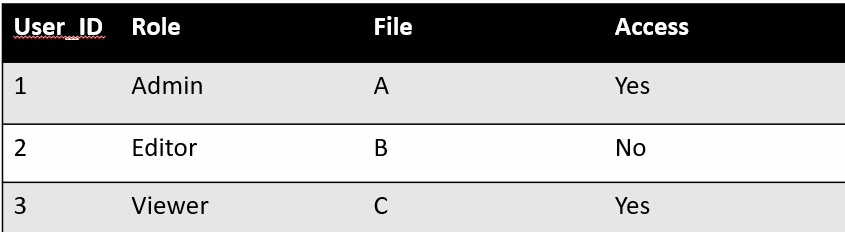
The task is made very simple by using a function/method from the pandas library. get\_dummies().This method first finds all the unique categorical values in the rows and then creates new columns and then writes binary value corresponding to the original RBAC dataset. This function makes our task of creating a new column, which is a tedious task.



And, at last we convert back dataframe into csv file

Lets take a small example where we convert RBAC into ReBAC dataset:

For instance lets take 3 users with access as follows:

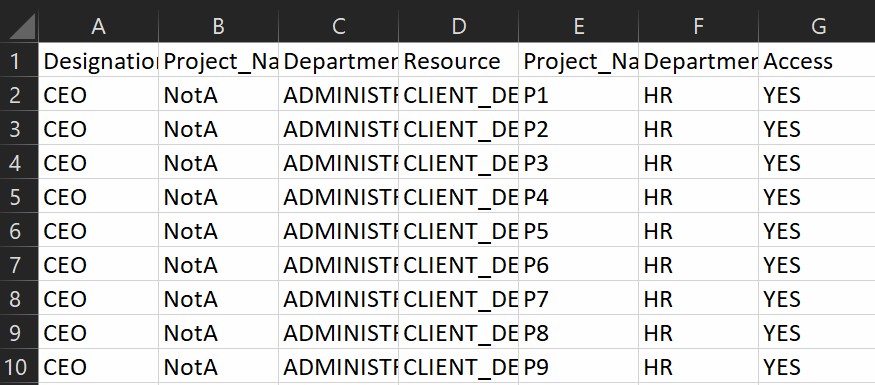


The ReBAC for the following example becomes:

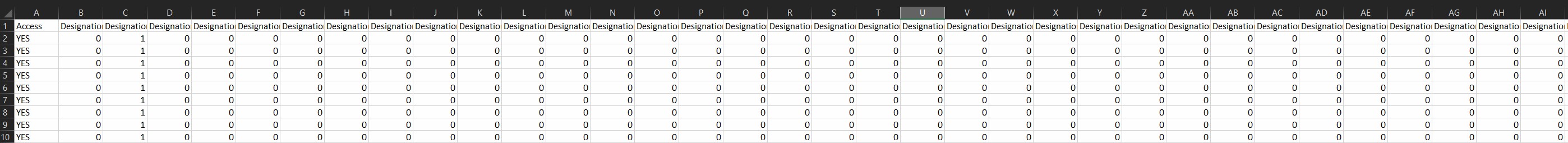
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| User\_I  D | Role\_admi n | Role\_edito  r | Role\_viewe  r | File\_  A | File\_  B | File\_  C | Acces s |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | Yes |
| 2 | 0 | 1 | 0 | 0 | 1 | 0 | No |
| 3 | 0 | 0 | 1 | 0 | 0 | 1 | Yes |

Now, instead of roles, we here consider relationships by converting them into different datasets where columns are defined based on unique values in the column.

Initial Data Sets taken:



After Conversion:



Here, the dataset is very large, with almost 2 lakh entries and rows of nearly 70.

**ADVANTAGES:**

There are many advantages of converting RBAC datasets into ReBAC datasets, which helps us in many ways. It gives us Dynamic access management where permissions are non-static and permissions can be granted or revoked based on real-time relationships, whereas in RBAC, it is static. Fine-grained Access control can be achieved through ReBAC.The important point is that we can define more complex policies, which are not possible using RBAC datatsets

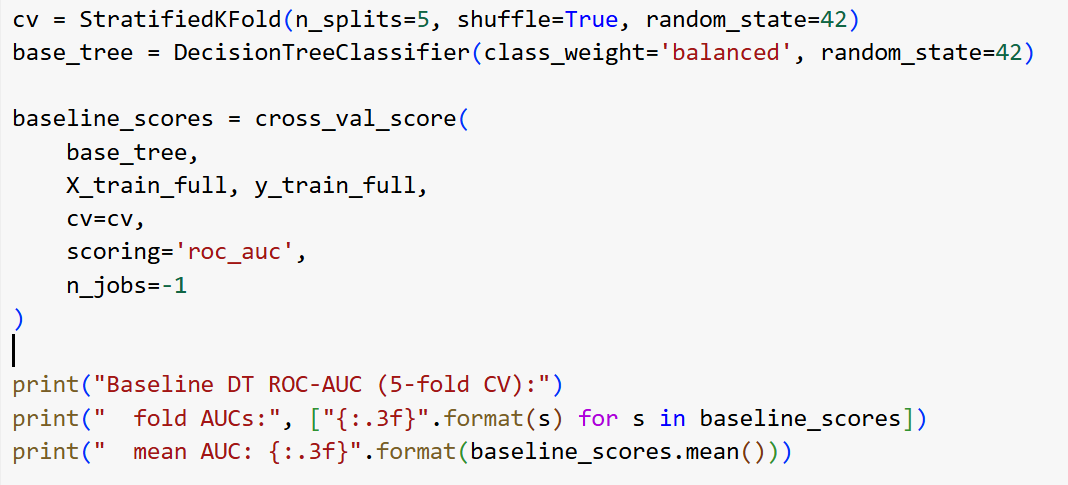
**POLICY EXTRACTION**

Now, we have converted our RBAC(or any other datasets) into ReBAC. The prime objective is to extract policies/rules from the given data. So, to extract the policies and apply them in the future, we converted the data into the ReBAC dataset. In the future, these policies will be used to give access. Now, we will run through the code to extract policies, which are described below in a structured and systematic manner.

**6.1 DECISION TREE**

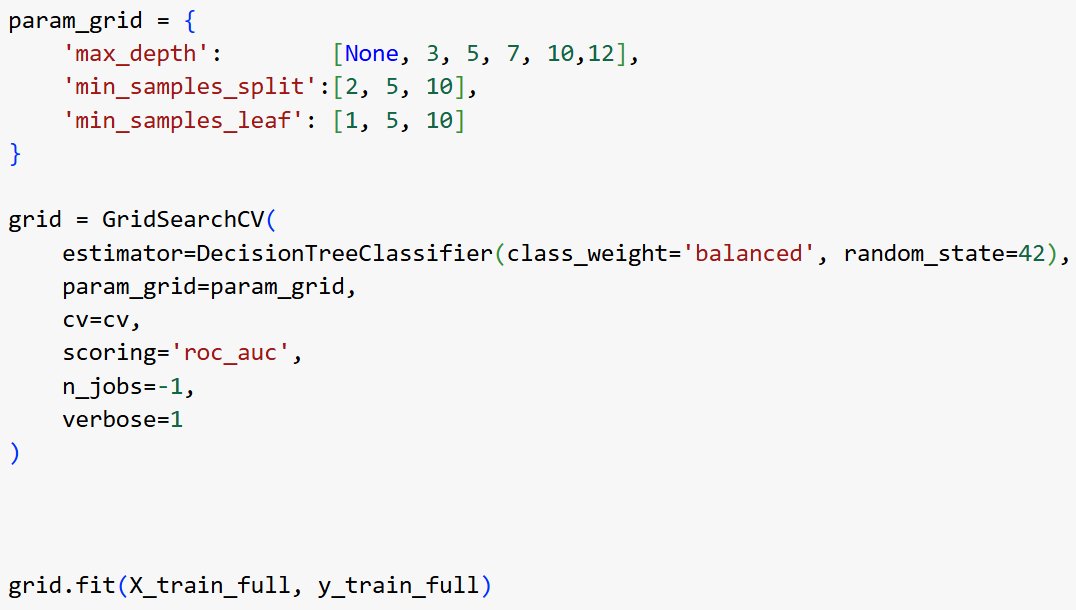
We start our analysis by using a decision tree, but the major issue is getting good accuracy and precision in the decision tree. So, parameters are to be adjusted such that we get the maximum of them. Here, cross-validation is used to get a quick baseline model. We split this into two phases.

PHASE-1:



Here, the dataset is split into 5 folds(randomizes before shuffling). After this, the baseline model performance using ROC-AUC as metric is computed. Finally, the ROC-AUC score for each fold is displayed along with their average which helps us for the future improvements.

PHASE 2:



The 3 parameters, namely:

max\_depth: represents the maximum depth of decision tree.

min\_samples\_split: minimum number of samples required to split an internode node

min\_Sample\_leaf: minimum number of samples required to form a leaf

A param\_grid is created using these parameters with various values and now systematically searches through all combinations of the specified parameters in the param\_grid.

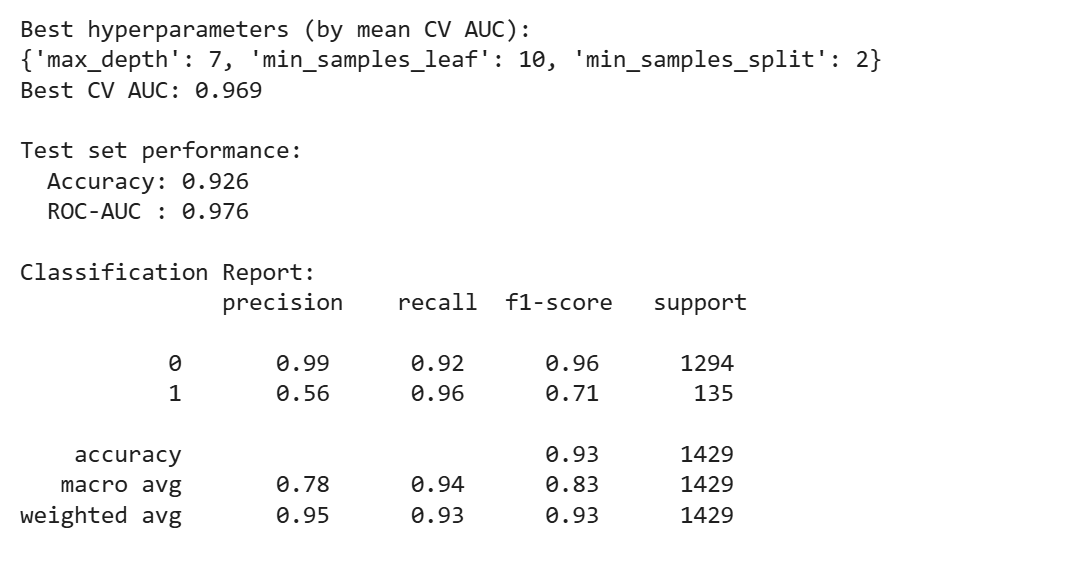
The grid search process executes, evaluating each hyperparameter combination via cross-validation to find the best-performing parameters.

In short:

Step 1 (Baseline): Quickly makes a baseline performance metric.

Step 2 (Optimization): Employs GridSearchCV to find optimal hyperparameters, refining the model for better performance

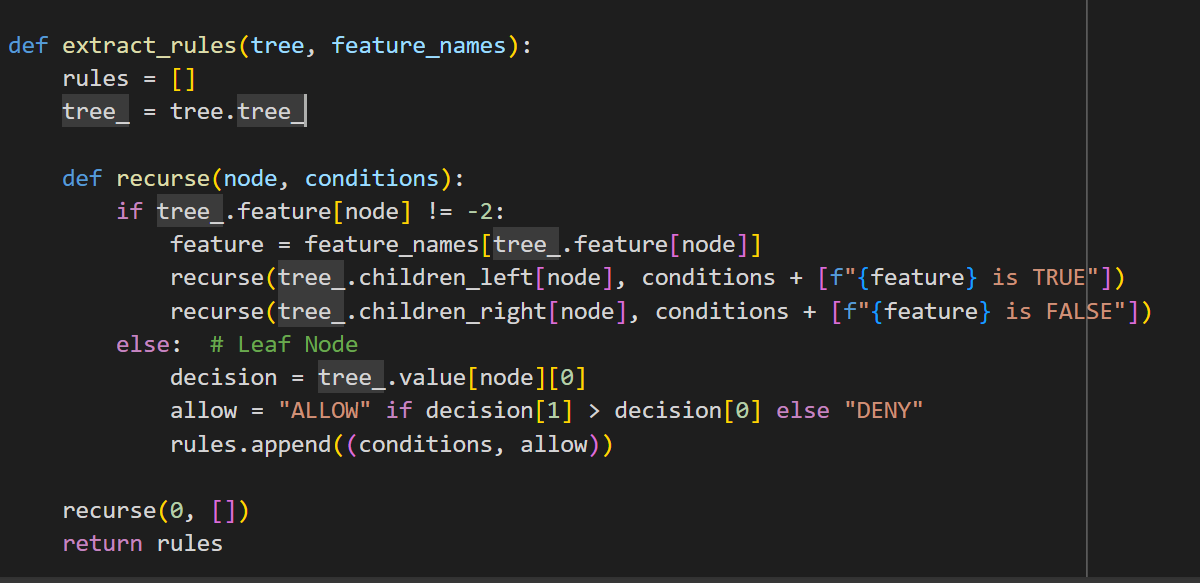
Now, finally decision tree is made with parameters obtained from the above work.



**6.2 POLICIES/RULES**

After the decision tree is created, we now extract policies from it. Two functions are created to extract policies.

**1.extract\_rules():** This function provides a **transparent, interpretable set of access control policies** derived from the trained decision tree. It is particularly useful when explaining model behavior to non-technical stakeholders or validating that learned policies align with organizational rules.



In this code, the rules are extracted, in the recurse function,the policies is defined such that TRUE for left sub-tree and FALSE for right sub-tree while going through tree.



2. format\_rules(rules): This function is used to format rules that is make it human readable that are extracted in the above function.So we use some keywords and format them.

INPUT: (['Feature1 is TRUE', 'Feature2 is FALSE'], 'ALLOW')

OUTPUT: [

"If Role\_Admin is TRUE, and Resource\_Private is FALSE, then access is allow",

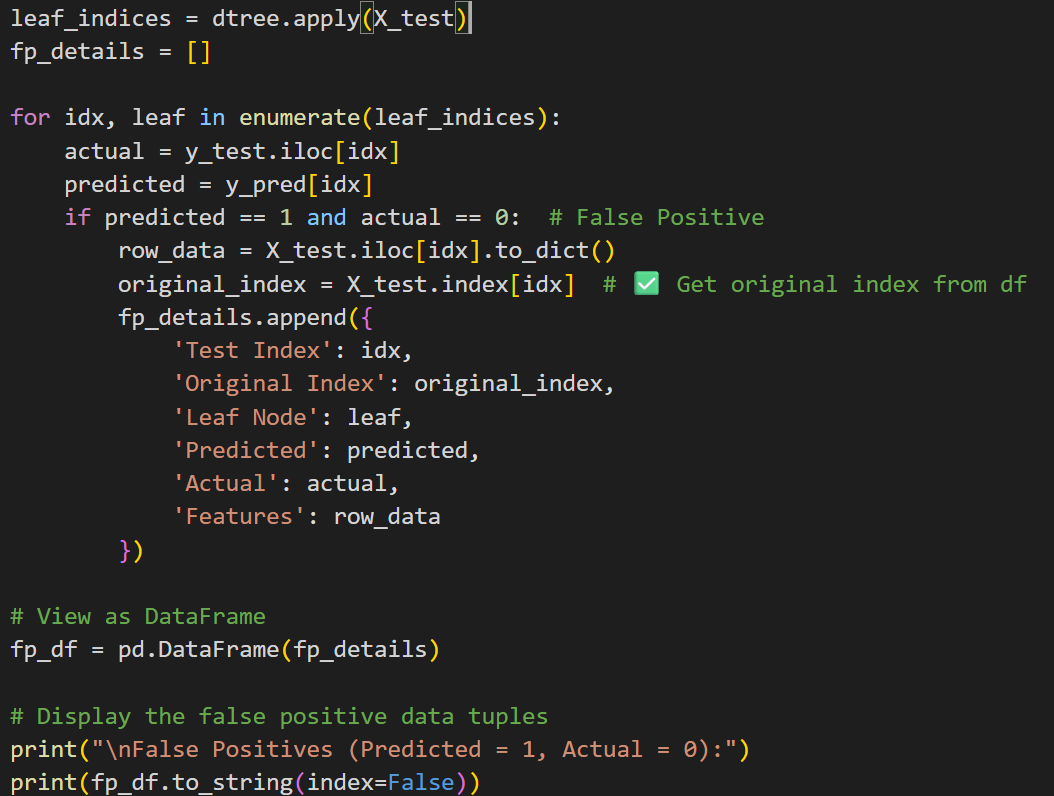
"If Role\_Student is FALSE, and Action\_Edit is TRUE, then access is deny",

]

**6.3 IDENTIFYING FALSE POSITIVE GRANTS IN DECISION TREE**

The actual part of our work starts now. Now, after policies are extracted we start to get false positives from them. Because we now focus on reducing these false positives and increasing the accuracy, precision and other parameters.

In access control systems, false positives represent a serious security risk. A false positive occurs when the model mistakenly grants access (predicts "Allow") to a user who should have been denied (actual label is "Deny"). This section describes how such false positives are identified and reported from the decision tree model's predictions.



The above code describes how to identify false positives. The trained decision tree is iterated with each test sample and compared with actual label. while comparing test index and original dataset index, leaf node they reached in a tree and all input features used for prediction are stored in a dataframe and displayed.

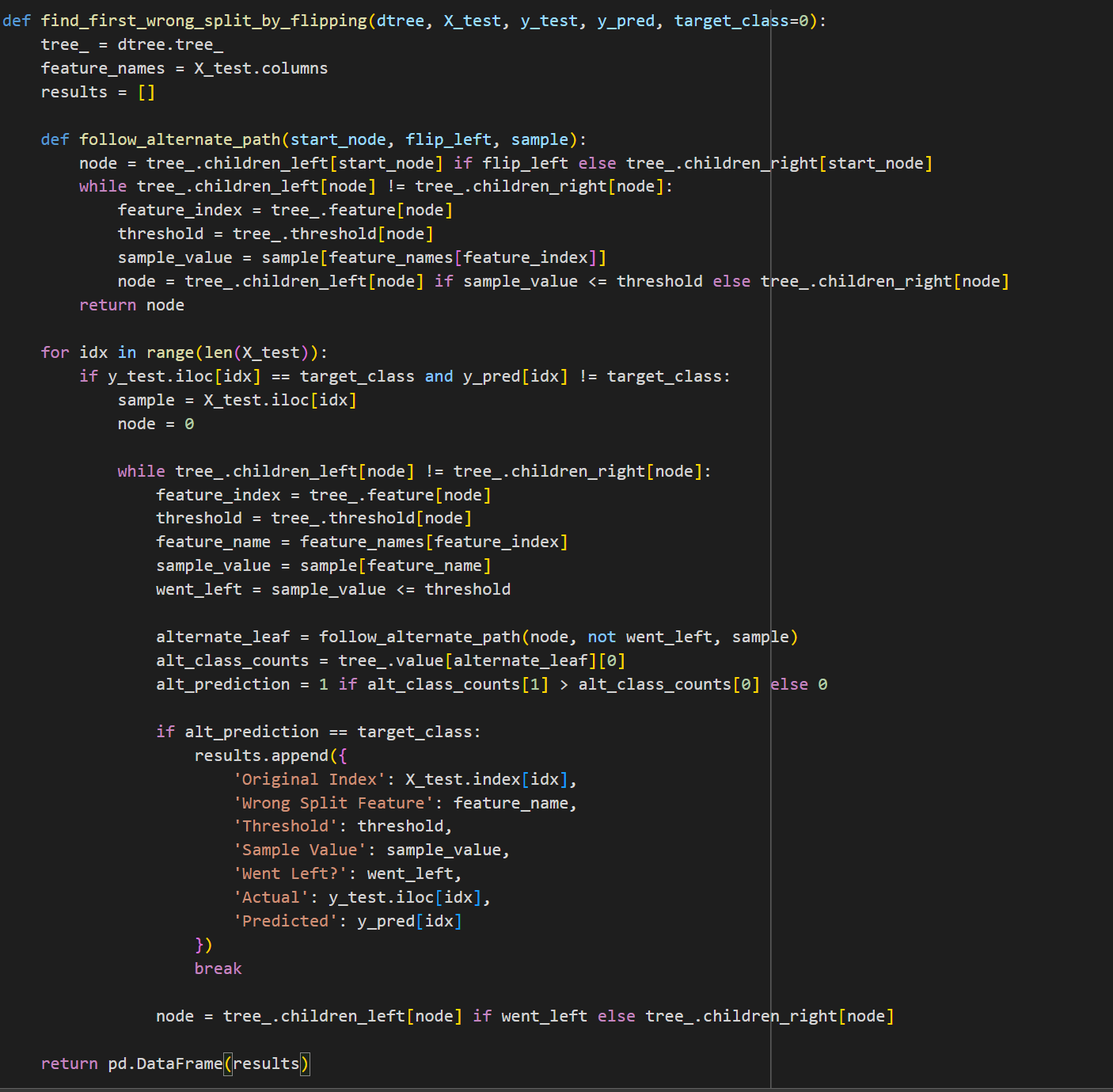
This analysis helps us to :

🡪Detection of over-permissive rules

🡪Insight into which feature combinations are most error-prone.

**6.4 ANALYZING FIRST WRONG SPLIT IN FALSE POSITIVES**

In cases where the decision tree failed, it is helpful to identify the first split that caused the error. This analysis simulates what would happen if the tree had approached the opposite path and checks if early wrong decision was made misclassification.



This code snippet describes the approach we wanted in order to analyze the false positives.

The main purpose is to find the earliest point in the tree that leads to misprediction. The follow\_alternate\_path(..) function gives a node and a sample, it stimulates going in the opposite direction. For every sample that was misclassified, the function checks each internal node along the path and reports to us.

Finally, a dataframe listing the original test index, the feature that caused the wrong split is printed.

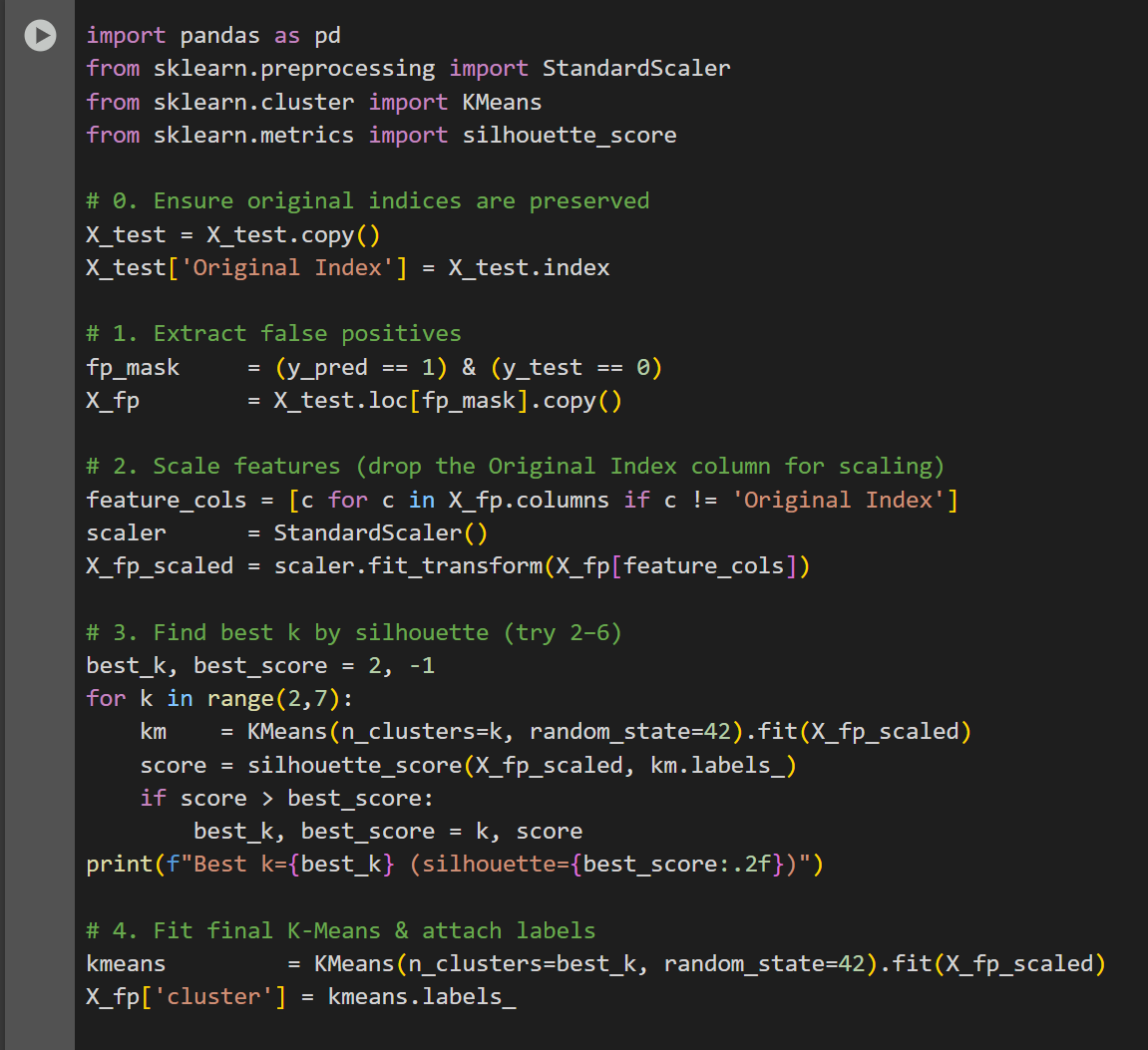
This analysis is not found much useful, as we get every wrong split is led by the root node itself, which might sound correct, but if you look in depth, this isn’t true, because this might lead us to change the root node, which will not solve our issue. So, this analysis is not accurate .

So, we now try to find similarities among False policies by making clusters among them, as this might lead us to draw some conclusions on policies/rules we got.

**7. Clustering Among Misclassified Samples**

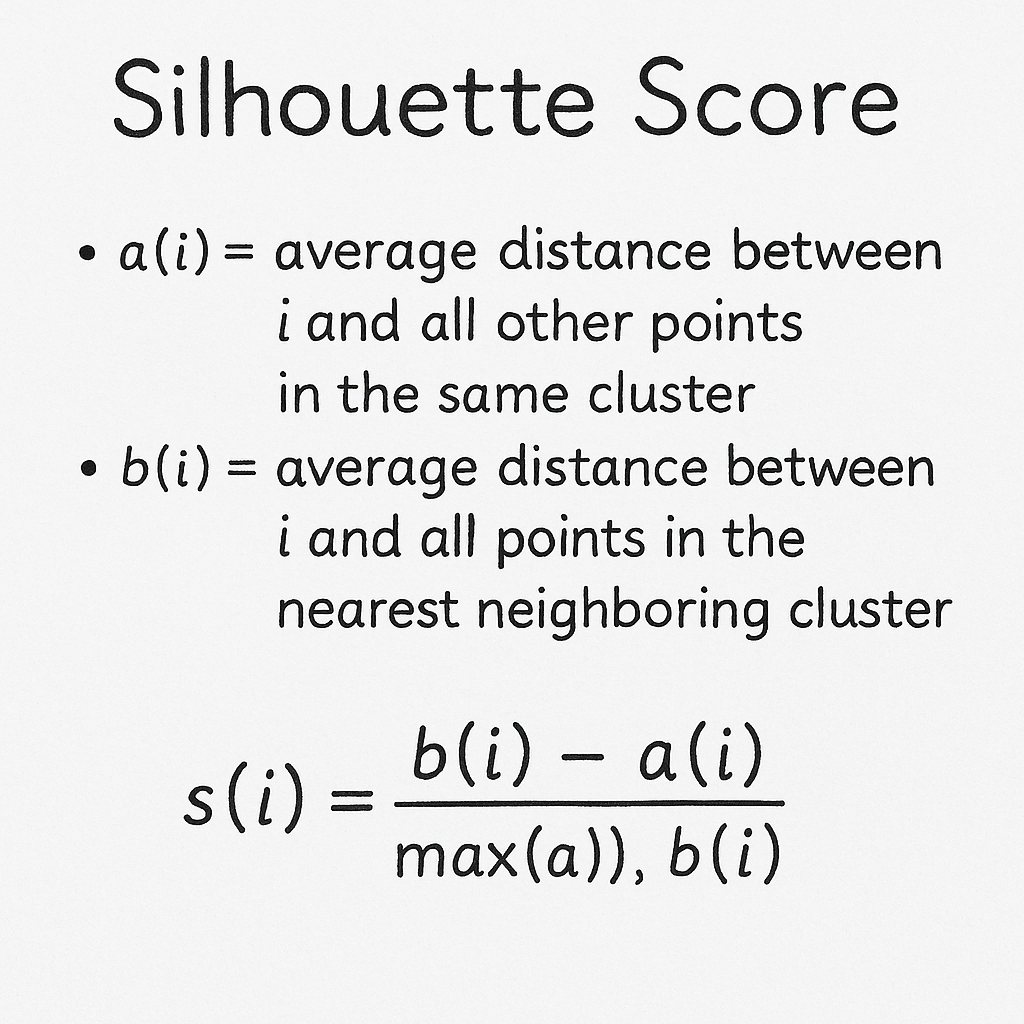
**7.1 Clustering using K-means**

To get a better view of misclassified samples, we use unsupervised learning(clustering) to group similar cases. This helps us to get insight into multiple access errors stemming from the same policy. And also gives us a clearer understanding of systematic patterns in errors.

****

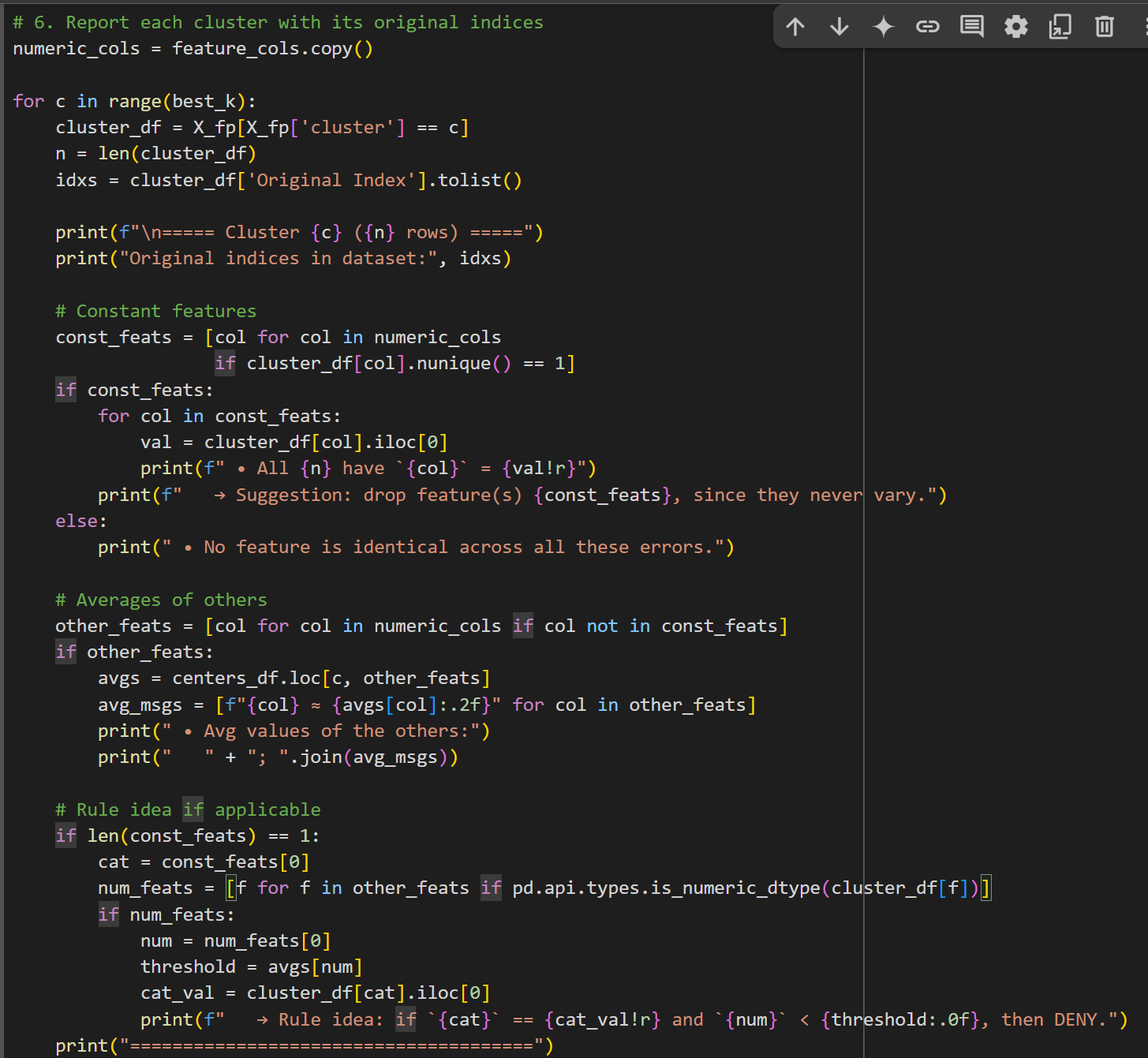
1. In the first step, we preserve original indices, so that it is useful for reference in future proceedings.
2. As predicted above, we extract false positives.
3. Scaling is done using StandardScaler() because K-means clustering is based on distance, so it is most important to perform scaling.This also boosts the performance of clustering.
4. Now, the important step is to consider how many clusters. We decide it based on the Silhouette Score ( -1 to 1, 1 being best). Selects the best k based on the Silhouette score. The range of k is chosen as 2 to 7 because k>7 makes the model overfit.
5. Perform clustering with an optimal number of clusters and add labels to the false positive DataFrame.

For a given data point i:



****

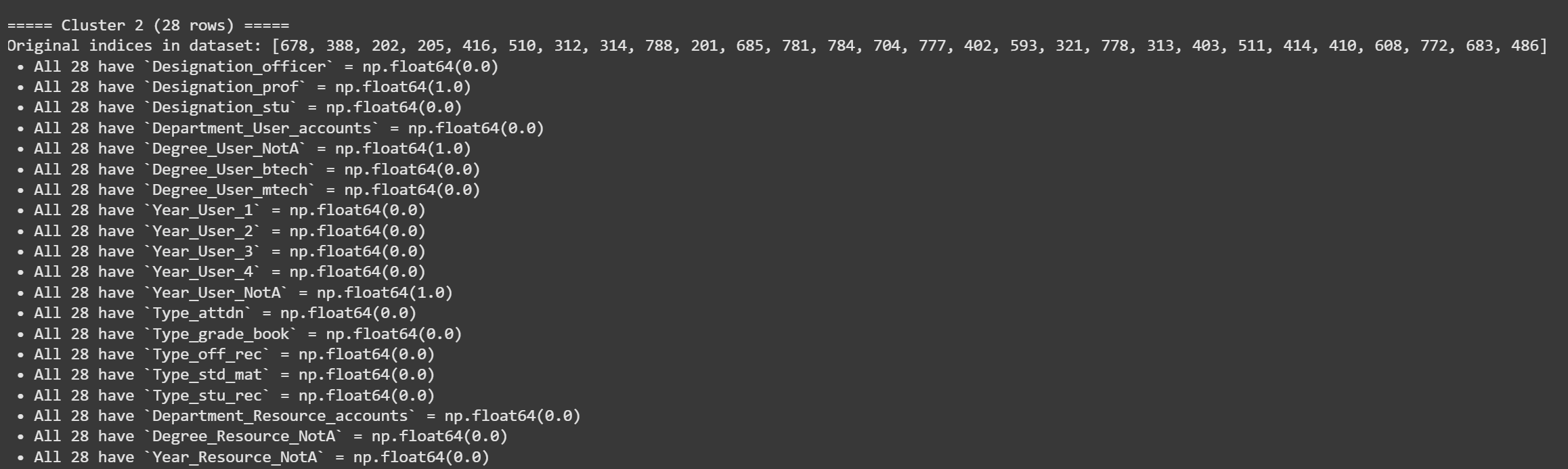
6. This code snippet is used to display how many false positives were assigned to each cluster. And also converts back to the original feature values.

****

7. The final part were our actual analysis is done. It lists original indices in the dataset, identifies constant features (same value for all samples within each cluster) candidates for rule generation.

**Eg:**

It is for a cluster taken from dataset

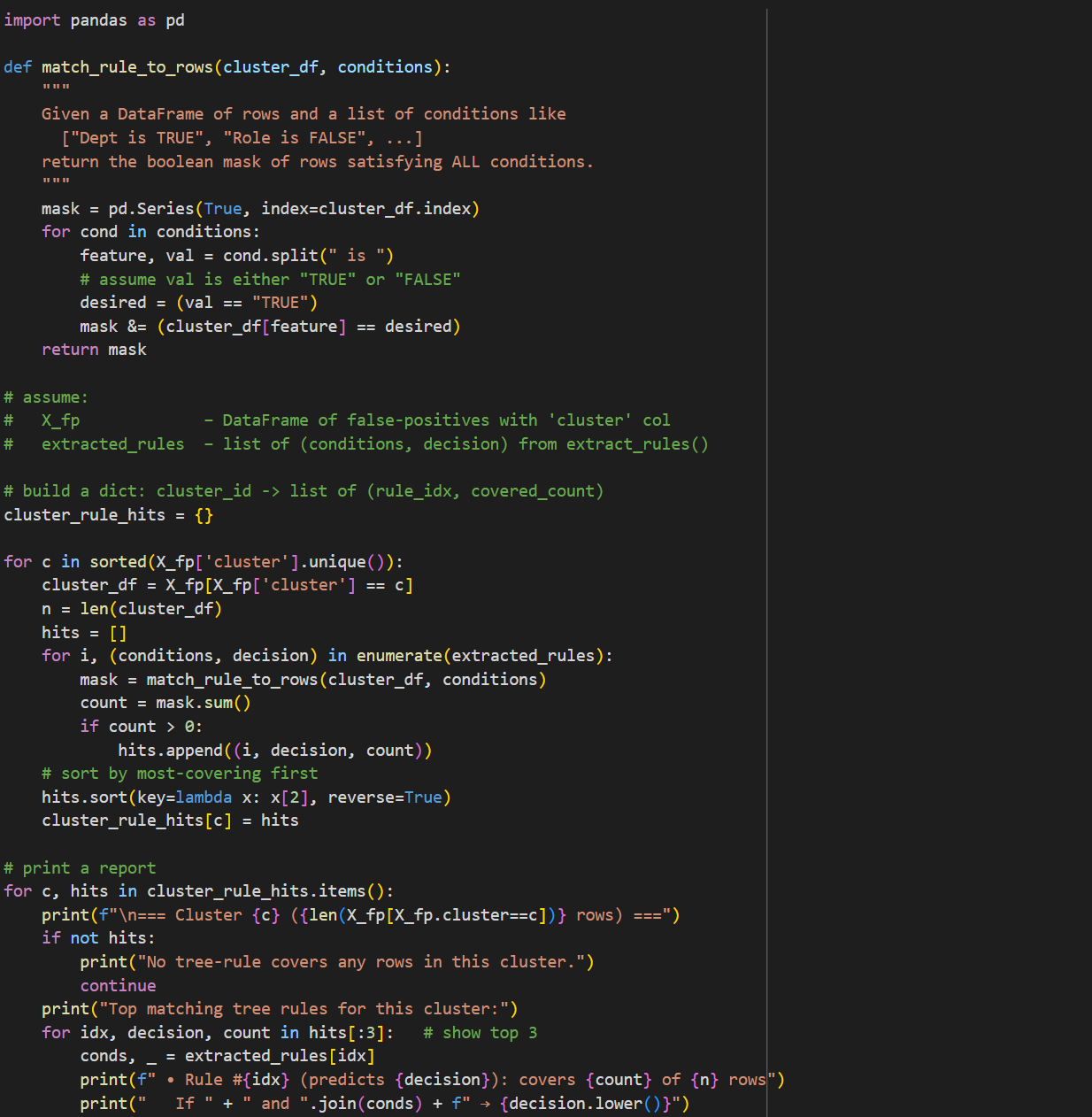
****

**7.2 Matching Decision Tree Rules to False Positives**

This section attempts to link rules extracted from the decision tree to clusters of FPs found earlier. By doing this we can:

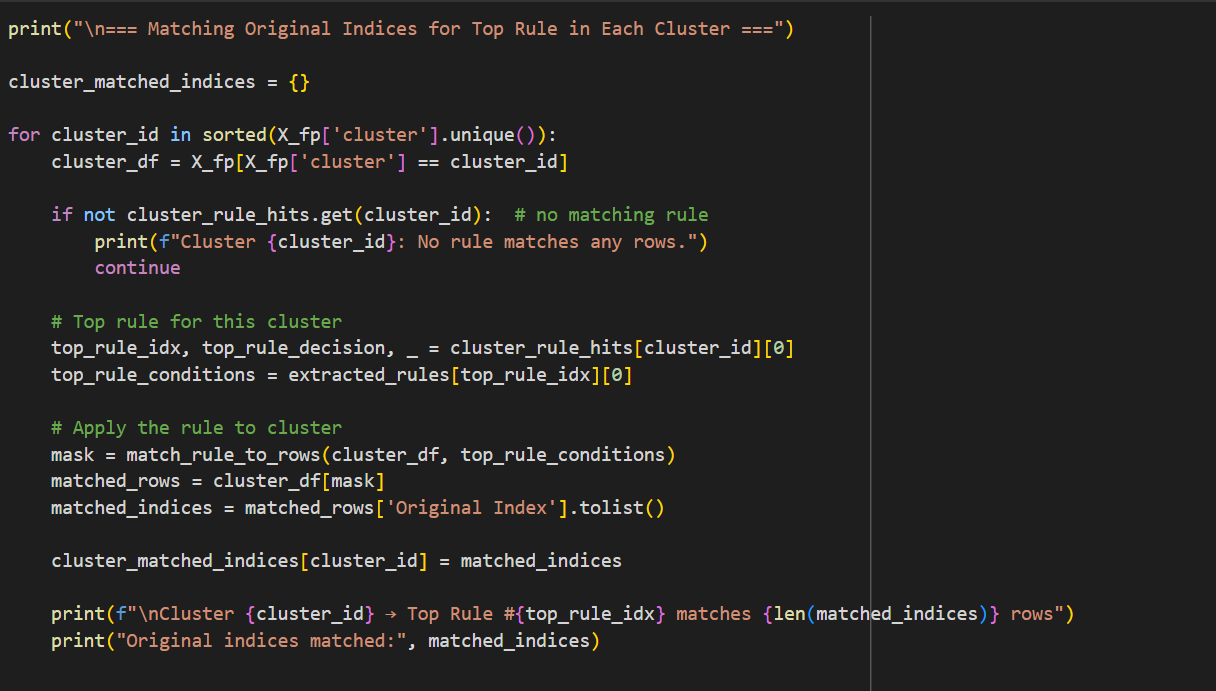
* Understand which access rules are contributing to misclassifications.
* Suggest patches to policy logic.

The code snippet is as follows:



Lets start with the match\_rule\_to\_row() function, which matches a rule to rows in clusters(list of TRUE/FALSE). Only return TRUE if all are satisfied. Then, Loop over clusters, for each cluster of FP’s, we check how many rows are matched by each decision rule. For each rule that covers part of a cluster, the number of matching rows is recorded. then, sorting and reporting the top 3 most-covering rules per cluster is done

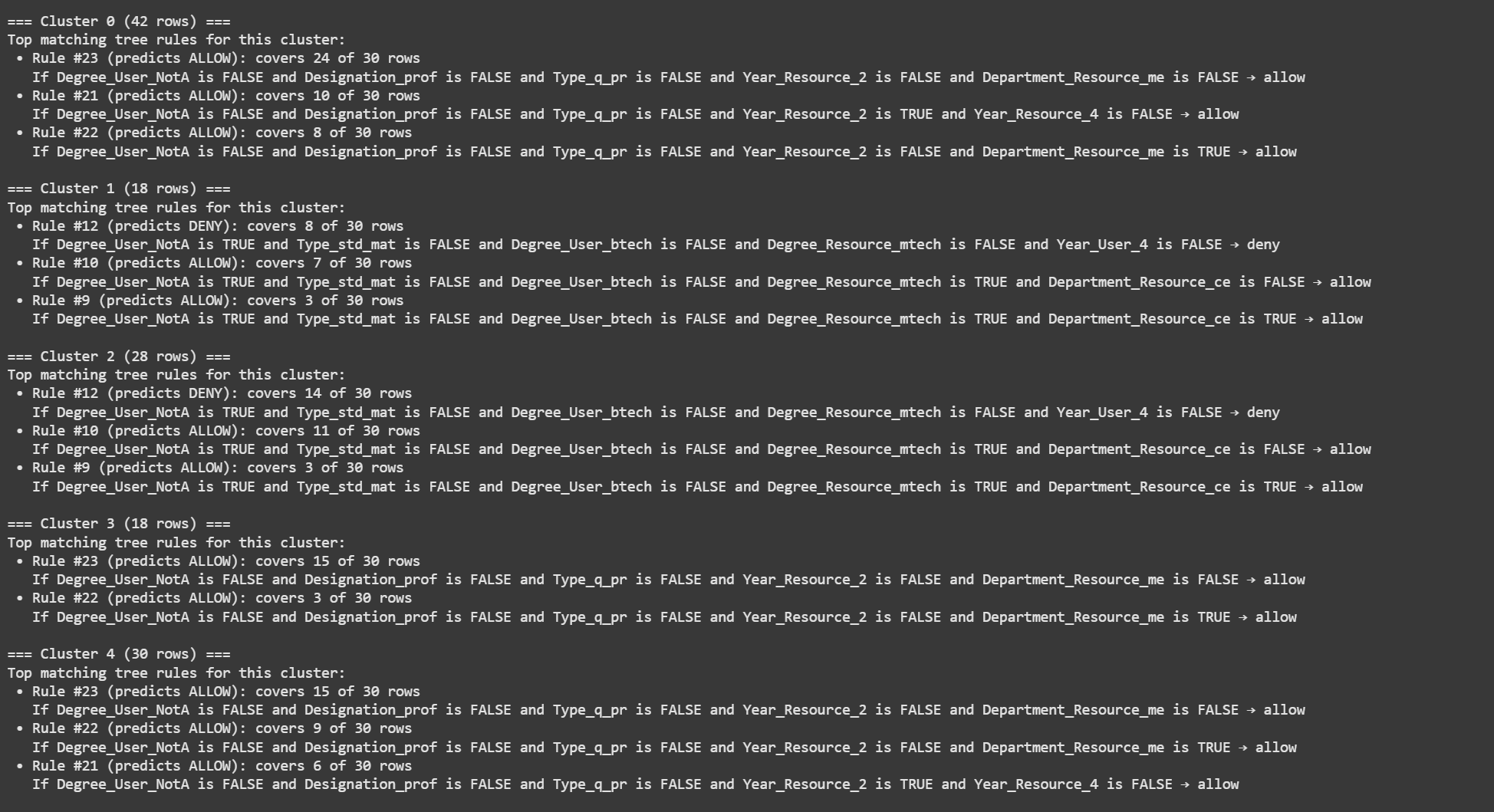
An extension to above code:



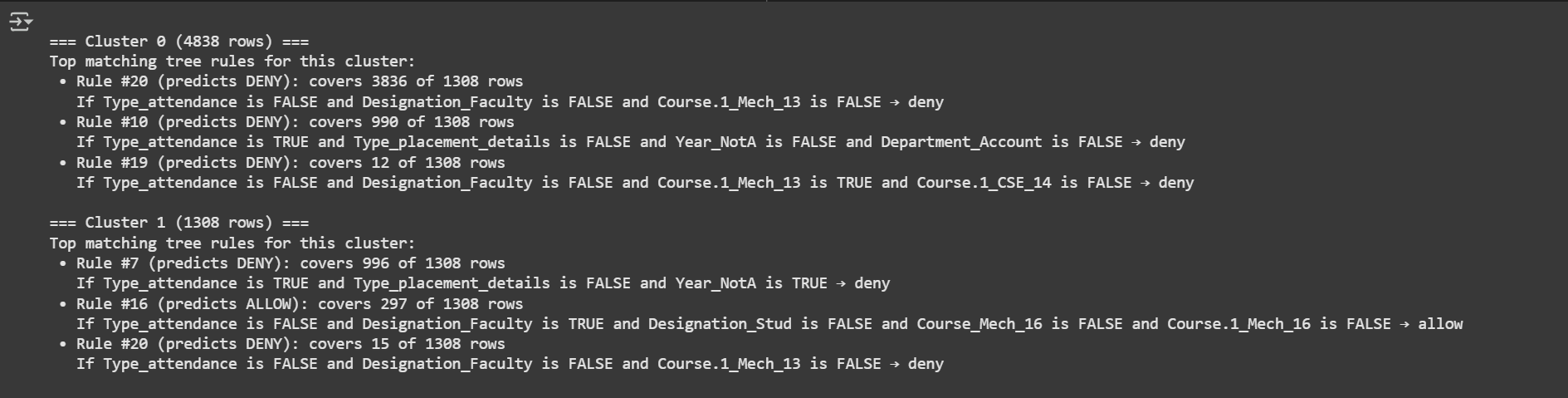
In this code, we print the original indices for top match rule which can be extended to al other rules**.**

**Experimental Results**

# Uni 1\_ReBAC



# Uni 2\_ReBAC



# CONCLUSION

The work given us a detailed view of RBAC, ReBAC models, and their differences. Later, we worked on the policy language B, which made us understand defining policies. we worked on converting the RBAC dataset into ReBAC datasets. We choose to use binary values for conversion, which is one of the effective methods. Many efficient algorithms have been developed to improve efficiency, further, we will study policy mining and other effective methods to understand ReBAC more deeply and extend it to use in real-time.

Later, after working on ReBAC datasets, we extracted false positives from decision tree. Now, after analysis is made on mis-classified datapoints and then draw similar columns from them,vlater we related false positives corresponding to such datapoints. So, Finally this analysis helped us to draw policies from a dataset(from RBAC) then remove false positives and also the datapoints leading to such thing.

This analysis can also be extended to false negatives, here in this report we stick to false positive because in the context of access control false positives is most vital thing to be eliminated that is no one withnot access should be denied access.

# REFERENCES

1.Oso. (n.d.). Relationship-Based Access Control (ReBAC). Oso Academy. from https://www.osohq.com/academy/relationship-based-access-control-rebac

2. Permit.io. (n.d.). ReBAC Overview – Build Policies. Permit.io Documentation. from https://docs.permit.io/how-to/build-policies/rebac/overview/

3.Cheng, Y., Park, J., &amp; Sandhu, R. (n.d.). A User-to-User

Relationship-Based Access

Control Model for Online Social Networks. Institute for Cyber Security, University of Texas at San Antonio.

4. G. Bruns, P. W. Fong, I. Siahaan, and M. Huth. Relationship-based access control: its

expression and enforcement through hybrid logic. In ACM CODASPY, 2012.

5.. B. Carminati, E. Ferrari, R. Heatherly, M. Kantarcioglu, and B. Thuraisingham.

Asemantic web based framework for social network access control. In

ACM SAC MAT,

2009

1. PhilipW. L. Fong. 2011. Relationship-based access control: protection model and policy language. In Proc. First ACM Conference on Data and Application Security and Privacy (CODASPY). ACM, 191–202
2. Maarten Decat, Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. 2014. .eworkforce management case study: functional analysis and access control requirements.CW Reports CW655. Department of Computer Science, KU Leuven. h.ps://lirias.kuleuven.be/handle/123456789/440203
3. Jasper Bogaerts, Maarten Decat, Bert Lagaisse, and Wouter Joosen. 2015. Entity-Based Access Control: supporting more expressive access control policies. In Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC 2015). ACM, 291–300. h.ps://lirias.kuleuven.be/handle/123456789/521795