Amazon Employee Access Challenge

CS 4661: Introduction to Data Science

Department of Computer Science

California State University, Los Angeles

<u>Team Members:</u>
Jaydeep Dhirubhai Gondaliya (Team Lead)
Nikhil Dhiman
Kinnariben Manojkumar Kotadiya
Dhruvi Shaileshbhai Desai
Drashti Shah

Index

- Accuracy
- Area Under Curve (AUC)
- Cross-Validation
- Precision
- Recall
- F1-Score
- Model Comparing

9. Challenges and Solutions

- Issues encountered and Steps taken to address challenges

10. Next Steps

- Advanced modeling plans (e.g., Gradient Boosting)
- Evaluation and validation strategies
- Timeline for project completion
- Summary of Key Considerations

11. Conclusion

12. References

- Dataset link and other resources used

1. Abstract

Brief overview of the project:

The Amazon Employee Access Challenge addresses the critical task of predicting employee access requirements within Amazon's internal network. The project leverages historical data on employee access patterns to develop a machine learning model capable of efficiently predicting future access needs. By accurately aligning resource permissions with evolving employee roles and responsibilities, the model aims to enhance security and streamline resource management. This approach minimizes unnecessary permissions, reduces manual interventions, and ensures a more efficient access control process. At its core, the project focuses on improving operational efficiency and security through data-driven predictions.

Summary of Objectives:

The primary objective of this project is to build a machine learning model capable of predicting employee access requirements for resources within Amazon's internal network. The goals of the project include:

1. **Developing a Predictive Model:** Leveraging historical data to accurately predict whether an employee should be granted access to a specific resource, minimizing both security risks and manual interventions.
2. **Optimizing Access Management:** Ensuring efficient and secure access control by reducing unnecessary permissions and aligning access decisions with employee roles and responsibilities.
3. **Improving Operational Efficiency:** Automating access granting and revocation processes to streamline resource management and minimize delays.
4. **Enhancing Security:** Proactively managing permissions to mitigate potential vulnerabilities and safeguard Amazon's internal network.
5. **Adapting to Dynamic Changes:** Ensuring the model adjusts effectively to evolving employee attributes and organizational roles.

2. Introduction

Background and context of the challenge:

In large organizations like Amazon, managing employee access to internal resources is critical to ensuring security and operational efficiency. Historically, this process involved manual reviews and approvals, which are time-consuming and prone to errors, especially as employee roles and responsibilities evolve. The Amazon Employee Access Challenge was introduced to address this issue by leveraging historical data to develop predictive models capable of automating access control decisions. The challenge not only highlights the complexities of access management in dynamic environments but also underscores the growing importance of machine learning in enhancing decision-making processes.

Importance of the Project in Access Management:

This project addresses these challenges by leveraging the power of machine learning to automate and optimize employee access management. By predicting future access requirements based on historical data, the project aims to:

- **Enhance Security:** Minimize the risk of unauthorized access by granting only necessary permissions to employees.
- **Improve Efficiency:** Streamline access control processes, reducing manual effort and associated delays.
- **Increase Agility:** Adapt quickly to changing organizational structures and employee roles.
- **Reduce Errors:** Minimize human error in access decisions, improving accuracy and consistency.
- **Gain Valuable Insights:** Uncover hidden patterns and trends in access behavior to inform strategic decisions.

3. Dataset Overview

The **Amazon Employee Access Challenge** dataset contains historical data on employee access decisions for various resources within the company, collected during 2010 and 2011. This data is essential for building a model that can automate the process of granting or denying access based on employee roles and transitions. The dataset is structured into two files: **train.csv** (used for training the model) and **test.csv** (used for predicting access decisions based on the trained model).

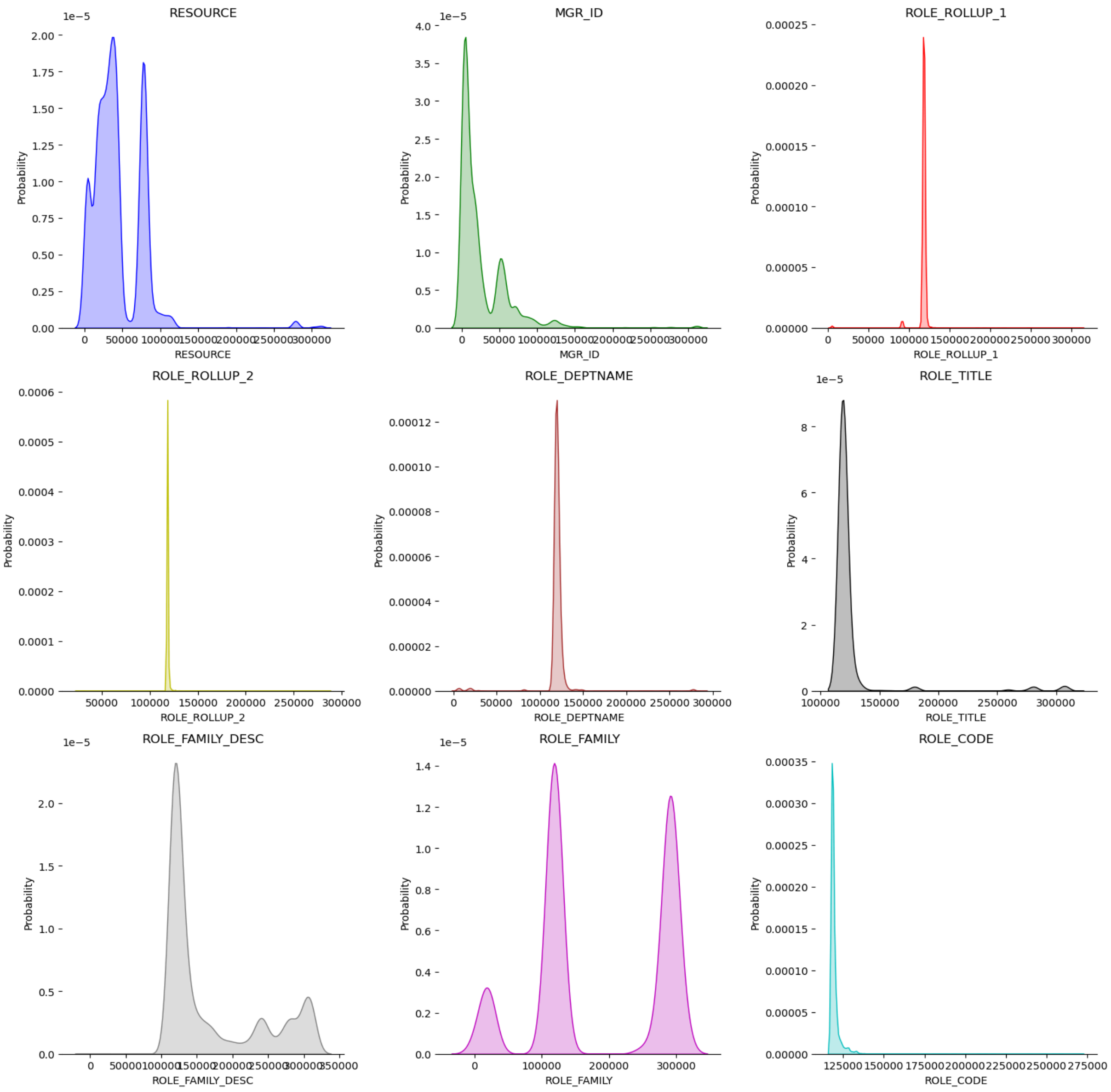
Description of the Dataset:

The dataset captures information about employee access decisions to various resources, with each record representing an access request by an employee. The historical data includes access decisions (whether access was granted or denied) along with key attributes such as the employee's role, department and other organizational details. The **train.csv** file contains labeled data, including the target variable ACTION (1 for approved, 0 for denied), and various role-related features that are used to predict the access decisions. The **test.csv** file contains similar features but lacks the ACTION variable, as it will be used to predict access decisions.

Key Features and Their Roles in Access Prediction:

1. **ACTION**: This is the target variable in the training dataset. It indicates whether an access request was approved (1) or denied (0). The goal of the model is to predict this variable based on other features.
2. **ROLE_ROLLUP**: This feature provides a hierarchical classification of an employee's role. It is important for predicting access since higher-level roles might require broader access across resources.
3. **ROLE_DEPTNAME**: This attribute specifies the department to which the employee belongs. Different departments may have distinct access needs depending on the nature of the resources they use.
4. **ROLE_TITLE**: This indicates the employee's job title or position. The title can help determine the level of access an employee should have, as some roles may require access to sensitive data or specific resources.
5. **ROLE_FAMILY:** This attribute categorizes an employee's role into broader groups based on the function or nature of the work performed. Each role family includes roles that share similar access requirements and responsibilities.
6. **ROLE_FAMILY_DESC**: This feature provides a description of the employee's role family, which groups similar roles together. Employees in the same role family often share similar access requirements.

7. **ROLE_CODE**: This attribute is a unique code representing the specific role of the employee. It is crucial for distinguishing between different job roles that might have different access needs.
8. **MGR_ID**: The manager's identifier helps to understand hierarchical relationships within the organization. The level of the manager might influence the access privileges required for their direct reports.
9. **RESOURCE**: This feature represents the unique ID of the resource to which access is being requested. The model will use this to predict whether the employee needs access to that specific resource based on their role.



Dataset

Source and Structure of the Data:

- **Source**: The dataset is provided by Kaggle as part of the Amazon Employee Access Challenge. You can access the dataset through this link.
- **Structure**: The dataset consists of two main files:
    - **train.csv**: Contains labeled data for model training, including the ACTION column (1 for approved, 0 for denied) along with role-related features such as ACTION, RESOURCE , MGR_ID, ROLE_ROLLUP_1, ROLE_ROLLUP_2, ROLE_DEPTNAME, ROLE_TITLE, ROLE_FAMILY, ROLE_FAMILY_DESC, and ROLE_CODE.
    - **test.csv**: Contains similar features to the training data but does not include the ACTION column. This is used to test the model's ability to predict access decisions based on the features.

Data Columns in train.csv:

- **ACTION**: Target variable (1 for approved, 0 for denied).
- **RESOURCE**: ID of the resource the employee is requesting access to.
- **ROLE_ROLLUP**, **ROLE_DEPTNAME**, **ROLE_TITLE**, **ROLE_FAMILY_DESC**, **ROLE_FAMILY** , **ROLE_CODE**: Role-related features for the employee.
- **MGR_ID**: Identifier of the employee's manager.

Data Columns in test.csv:

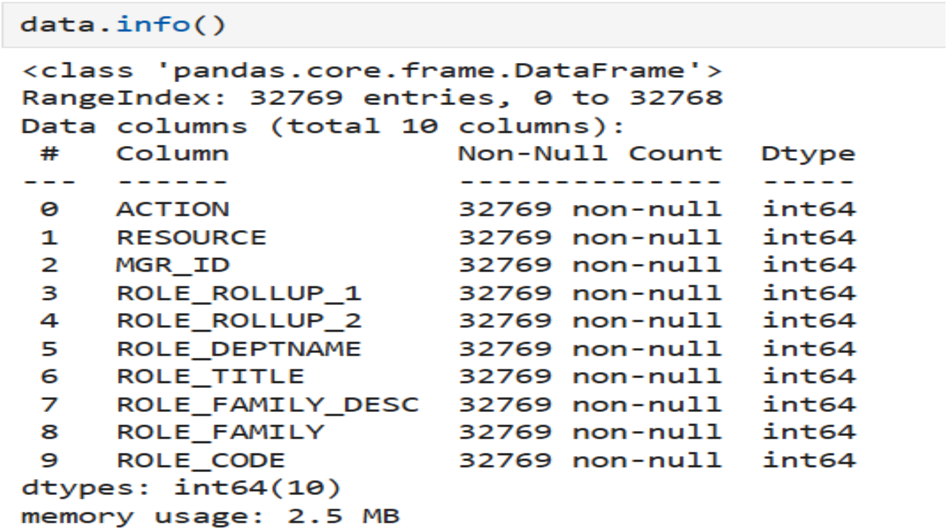
- **RESOURCE**: ID of the resource for which access is requested.
- **ROLE_ROLLUP**, **ROLE_DEPTNAME**, **ROLE_TITLE**, **ROLE_FAMILY_DESC**, **ROLE_FAMILY** **ROLE_CODE**: Role-related features for the employee.
- **MGR_ID**: Identifier of the employee's manager.

4. Exploratory Data Analysis (EDA)

In the context of the **Amazon Employee Access Challenge**, an EDA can help uncover important insights and patterns in the data, which can guide the development of an effective model for automating employee access privileges. Here is a general approach to performing EDA on the provided data, focusing on key patterns, insights, and addressing challenges like handling categorical and missing data.

Dataset Properties

The dataset contains 32,769 entries with 10 columns. All columns have complete data with no missing values, as shown below:

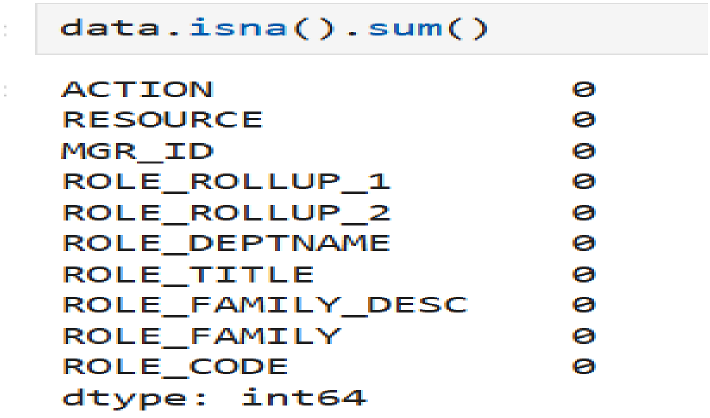


```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32769 entries, 0 to 32768
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   ACTION            32769 non-null  int64
 1   RESOURCE          32769 non-null  int64
 2   MGR_ID            32769 non-null  int64
 3   ROLE_ROLLUP_1     32769 non-null  int64
 4   ROLE_ROLLUP_2     32769 non-null  int64
 5   ROLE_DEPTNAME     32769 non-null  int64
 6   ROLE_TITLE        32769 non-null  int64
 7   ROLE_FAMILY_DESC  32769 non-null  int64
 8   ROLE_FAMILY       32769 non-null  int64
 9   ROLE_CODE         32769 non-null  int64
dtypes: int64(10)
memory usage: 2.5 MB
```

Missing Values

No missing values are present in the dataset:

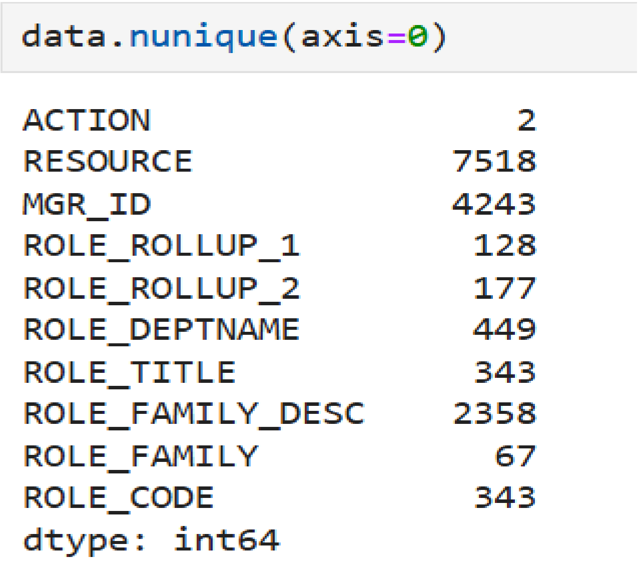


```
data.isna().sum()

ACTION              0
RESOURCE            0
MGR_ID              0
ROLE_ROLLUP_1       0
ROLE_ROLLUP_2       0
ROLE_DEPTNAME       0
ROLE_TITLE          0
ROLE_FAMILY_DESC    0
ROLE_FAMILY         0
ROLE_CODE           0
dtype: int64
```

Unique Value Counts

The unique value counts for each column are as follows:

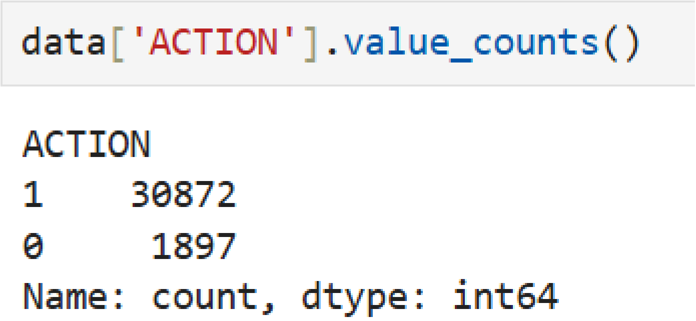


```
data.nunique(axis=0)
```

```
ACTION                  2
RESOURCE             7518
MGR_ID               4243
ROLE_ROLLUP_1         128
ROLE_ROLLUP_2         177
ROLE_DEPTNAME         449
ROLE_TITLE            343
ROLE_FAMILY_DESC     2358
ROLE_FAMILY            67
ROLE_CODE             343
dtype: int64
```

Target Variable Distribution

The target variable **ACTION** has a highly imbalanced distribution



```
data['ACTION'].value_counts()
```

```
ACTION
1    30872
0     1897
Name: count, dtype: int64
```

5. Encoding

One-Hot Encoding:

In this challenge, one-hot encoding is crucial for transforming categorical data into a numerical format suitable for machine learning models. Here's a brief overview of how to approach one-hot encoding and its role in this problem:

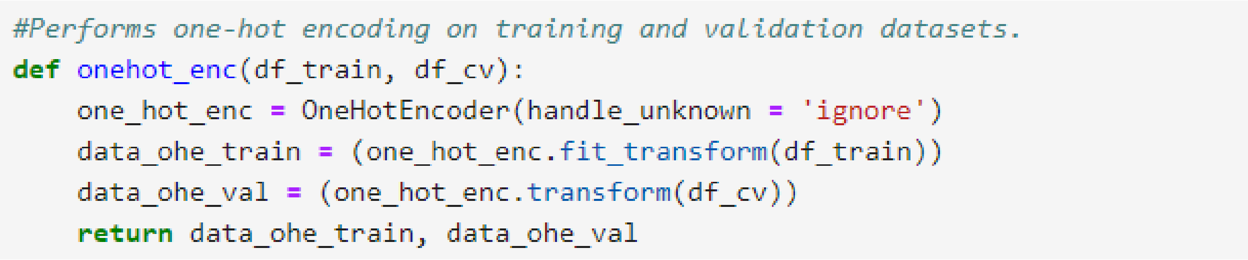
1. Why Use One-Hot Encoding?

- Even if the data is represented as integers, one-hot encoding is necessary to treat each entry in a categorical feature as distinct. For example, features like **ROLE_DEPTNAME, ROLE_TITLE,** and **ROLE_FAMILY** may have integer labels, but those integers do not represent any meaningful order.
- Logistic regression would incorrectly treat these integers as continuous values, which could lead to misinterpretation. One-hot encoding ensures that each unique entry in a categorical feature is treated as an independent, binary feature, allowing the model to understand that these are distinct categories without implying any numerical significance or ordering.

2. How One-Hot Encoding Works:

- For each unique category in a feature, a new binary column is created. If an employee has that category, the column gets a **1**; otherwise, it gets a **0**.

Example:



```
#Performs one-hot encoding on training and validation datasets.
def onehot_enc(df_train, df_cv):
    one_hot_enc = OneHotEncoder(handle_unknown = 'ignore')
    data_ohe_train = (one_hot_enc.fit_transform(df_train))
    data_ohe_val = (one_hot_enc.transform(df_cv))
    return data_ohe_train, data_ohe_val
```

3. Steps for One-Hot Encoding in the Challenge:

a. Identify Categorical Features:

- Look for columns like **ROLE_ROLLUP, ROLE_DEPTNAME, ROLE_TITLE , ROLE_FAMILY_DESC ,ROLE_CODE , RESOURCE_ID etc.**

b. Apply One-Hot Encoding:

- We can use Python libraries like **pandas** to encode the categorical features.

4. Potential Challenges:

- **High Cardinality**:
  Features like **RESOURCE** and **ROLE_CODE** may have hundreds or thousands of unique categories, leading to a significant increase in the number of columns.
  Solution:
- Apply dimensionality reduction techniques or use frequency encoding for high-cardinality features.
- Consider combining rare categories with low frequency into a single group.

5. Post-Encoding Considerations:

- **Feature Selection:** After encoding, some features might be highly correlated or irrelevant. Feature selection techniques can help remove redundant columns.
- **Model Input:** After encoding, the dataset is ready to be input into machine learning models such as decision trees, random forests, or gradient boosting machines for predicting the appropriate access levels.

6. Baseline Modeling

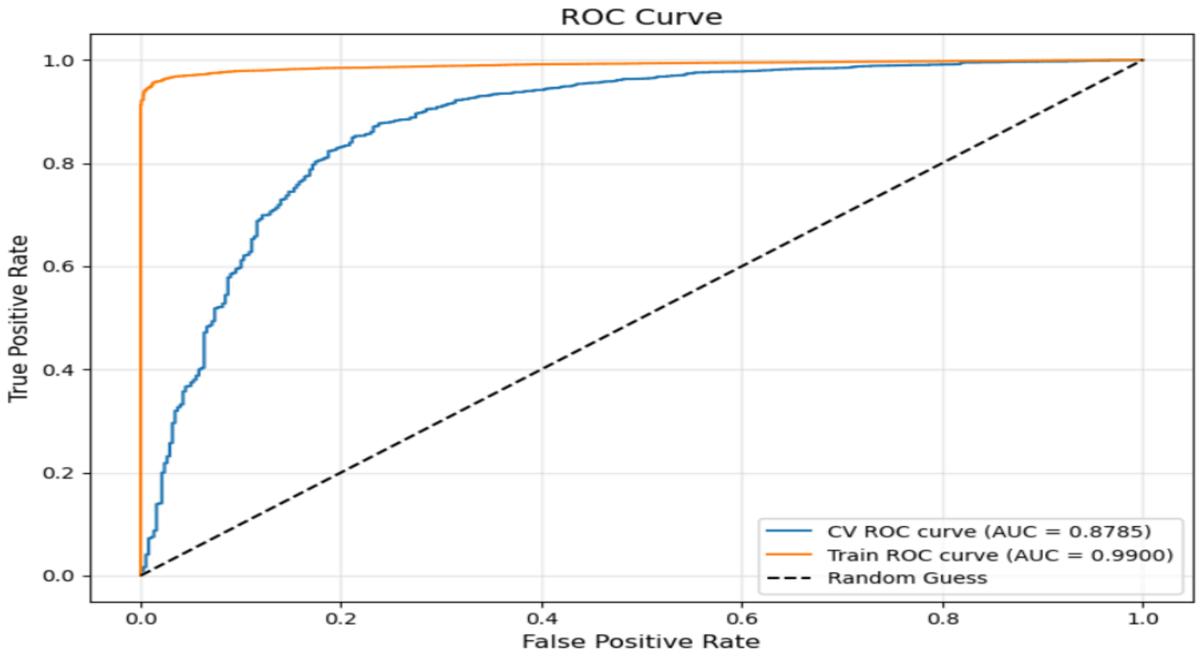
In the context of the **Amazon Employee Access Challenge**, the aim is to predict and automate the assignment of access privileges for employees based on their roles and transitions within the company. By leveraging historical data, the model will automatically adjust access permissions, which will minimize manual intervention and improve operational efficiency.

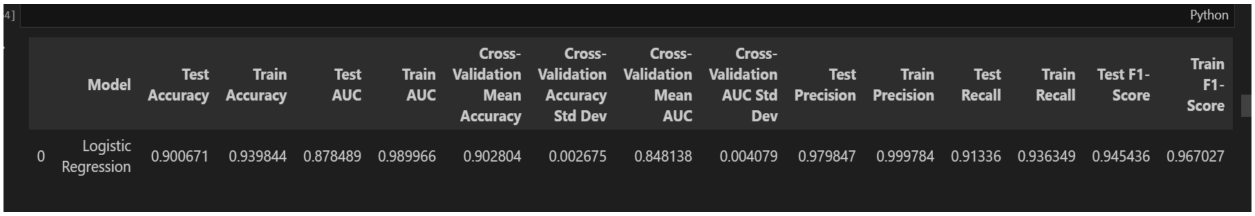
Baseline Models:

Baseline models are used to establish a starting point for model performance. In this case, we'll use Logistic Regression and Decision Trees as baseline models and then proceed with fine-tuning the Decision Trees.

Logistic Regression:

- **Overview:** Logistic Regression is a simple yet effective baseline model for classification problems. It models the probability that an instance belongs to a particular class.
- **Use Case:** In this context, Logistic Regression would predict whether an employee should have access or not, given their role and other features.
    1. **Preprocessing:** Handle missing values, normalize features, and encode categorical variables.
    2. **Model Training:** Train a Logistic Regression model on the training dataset.
    3. **Evaluation:** Evaluate the model performance using both validation and cross-validation metrics such as Accuracy, Precision, Recall, F1-Score, and AUC.



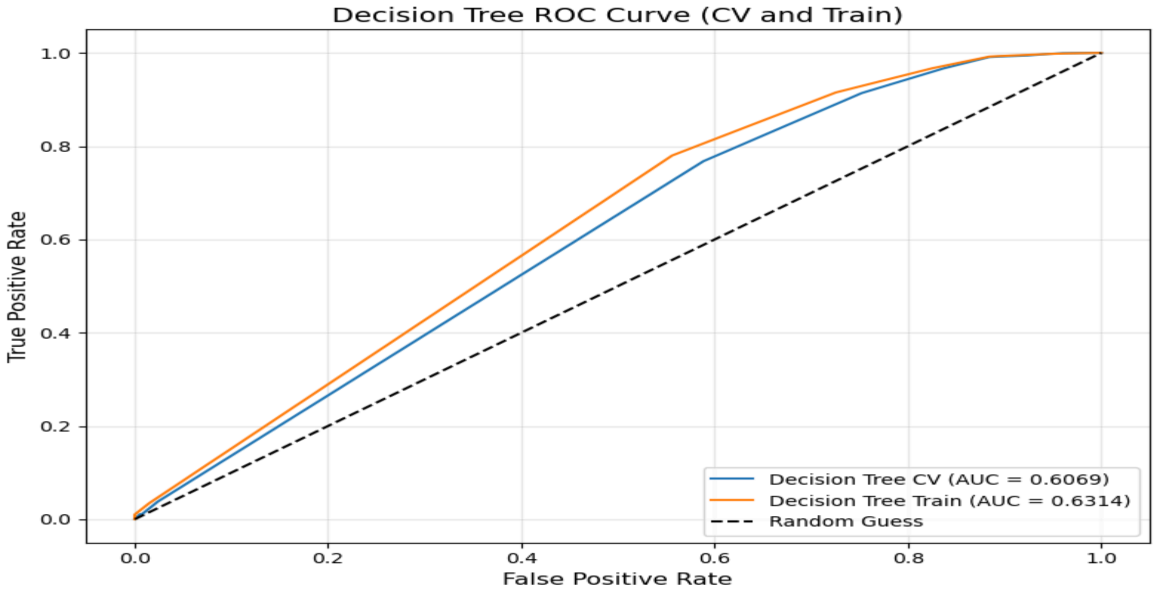


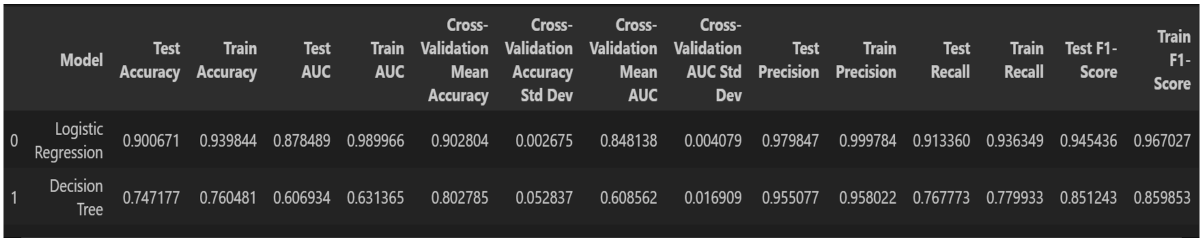
	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.900671	0.939844	0.878489	0.989966	0.902804	0.002675	0.848138	0.004079	0.979847	0.999784	0.91336	0.936349	0.945436	0.967027

Performed Cross-Validation Metrics

Decision Trees:

- **Overview:** Decision Trees are a type of supervised learning algorithm that splits the data into subsets based on feature values. It is a non-linear model that can capture complex relationships between features.
- **Use Case:** For predicting employee access, Decision Trees can be used to make decisions about access based on a given dataset.
    1. **Preprocessing:** Similar to Logistic Regression, handle missing data, and encode categorical variables.
    2. **Model Training:** Train a Decision Tree classifier on the data.
    3. **Evaluation:** Evaluate performance using common metrics like Accuracy, Precision, Recall, F1-Score, and AUC.
    4. **Cross-validation:** Use cross-validation to estimate the generalization performance of the model.





	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.900671	0.939844	0.878489	0.989966	0.902804	0.002675	0.848138	0.004079	0.979847	0.999784	0.913360	0.936349	0.945436	0.967027
1	Decision Tree	0.747177	0.760481	0.606934	0.631365	0.802785	0.052837	0.608562	0.016909	0.955077	0.958022	0.767773	0.779933	0.851243	0.859853

- **Evaluation matrix for Decision tree - As per output we can see from above scores, Decision Tree Classifiers tend to be overfitting, let's do some hyper parameter Fine-tuning.**

Fine-Tuning Decision Trees:

Fine-tuning is necessary to improve the performance of Decision Trees and avoid overfitting. Some common techniques for fine-tuning include:

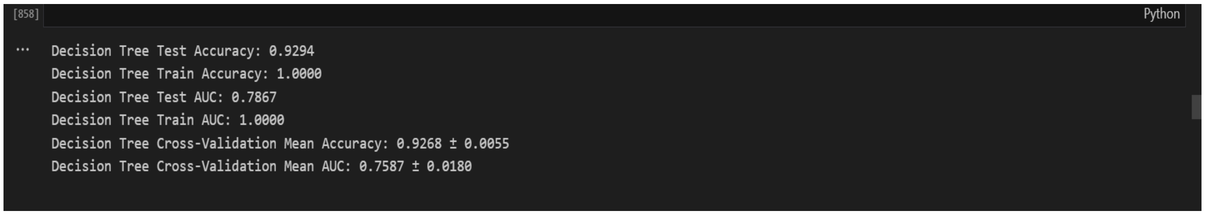
- **Hyperparameter Tuning:** Adjust hyperparameters like max depth, min samples split, and criterion.
- **Pruning:** Reduce the size of the tree to avoid overfitting.

Here's how hyperparameter tuning is done in the Decision Tree model:

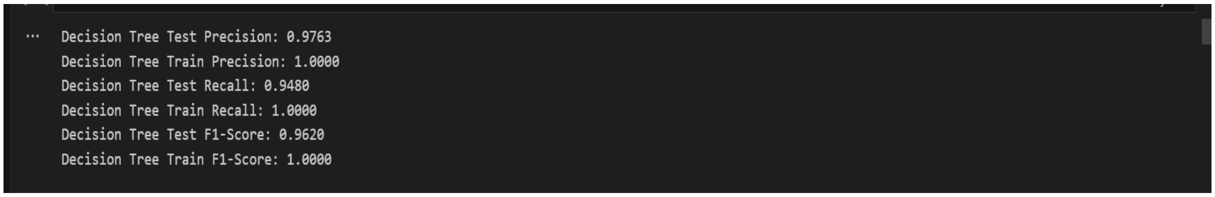
1. `max_depth=5`:
   - Limits the maximum depth of the decision tree to 5 levels.
   - Prevents overfitting by ensuring the tree doesn't grow too complex and is more likely to generalize well to unseen data.
2. `random_state=42`:
   - Sets the random seed for reproducibility of results.
   - Ensures that the model's training process (such as splitting data) is consistent across different runs, leading to stable and comparable results.
3. `class_weight='balanced'`:
   - Automatically adjusts the class weights to handle class imbalance.
   - Increases the weight of minority classes so that the model pays more attention to them during training, improving performance on imbalanced datasets.

These parameters represent a form of hyperparameter tuning aimed at improving the model's performance, preventing overfitting, and handling class imbalance effectively. Below we have given accuracy of before and after tuning.

BEFORE:

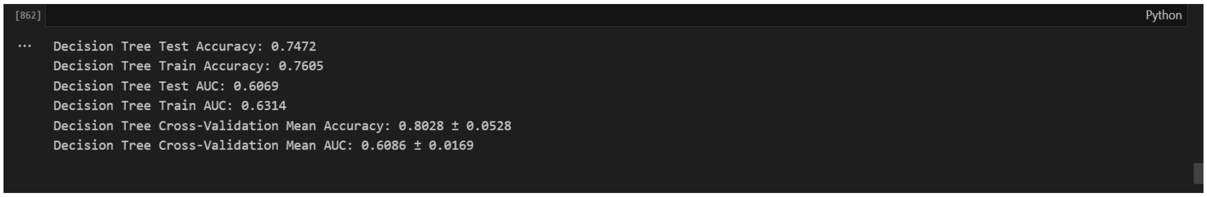


```
Decision Tree Test Accuracy: 0.9294
Decision Tree Train Accuracy: 1.0000
Decision Tree Test AUC: 0.7867
Decision Tree Train AUC: 1.0000
Decision Tree Cross-Validation Mean Accuracy: 0.9268 ± 0.0055
Decision Tree Cross-Validation Mean AUC: 0.7587 ± 0.0180
```

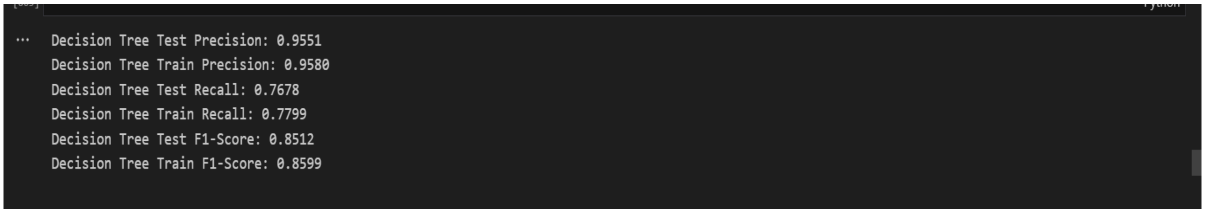


```
Decision Tree Test Precision: 0.9763
Decision Tree Train Precision: 1.0000
Decision Tree Test Recall: 0.9480
Decision Tree Train Recall: 1.0000
Decision Tree Test F1-Score: 0.9620
Decision Tree Train F1-Score: 1.0000
```

AFTER Hyperparameter Tuning:



```
Decision Tree Test Accuracy: 0.7472
Decision Tree Train Accuracy: 0.7605
Decision Tree Test AUC: 0.6069
Decision Tree Train AUC: 0.6314
Decision Tree Cross-Validation Mean Accuracy: 0.8028 ± 0.0528
Decision Tree Cross-Validation Mean AUC: 0.6086 ± 0.0169
```



```
Decision Tree Test Precision: 0.9551
Decision Tree Train Precision: 0.9580
Decision Tree Test Recall: 0.7678
Decision Tree Train Recall: 0.7799
Decision Tree Test F1-Score: 0.8512
Decision Tree Train F1-Score: 0.8599
```

Benchmarks and Performance Results:

Once all baseline models and their tuned versions are trained, evaluate their performance using the following metrics:

Accuracy: Measures the percentage of correctly predicted instances (access granted or denied).
Accuracy = Correct Predictions / Total Predictions

Precision: The proportion of true positives among all positive predictions.
Precision = TP / TP + FP

Recall: The proportion of true positives among all actual positives.
Recall = TP / TP + FN

F1-Score: The harmonic mean of precision and recall
F1-Score = 2 × Precision × Recall / Precision + Recall

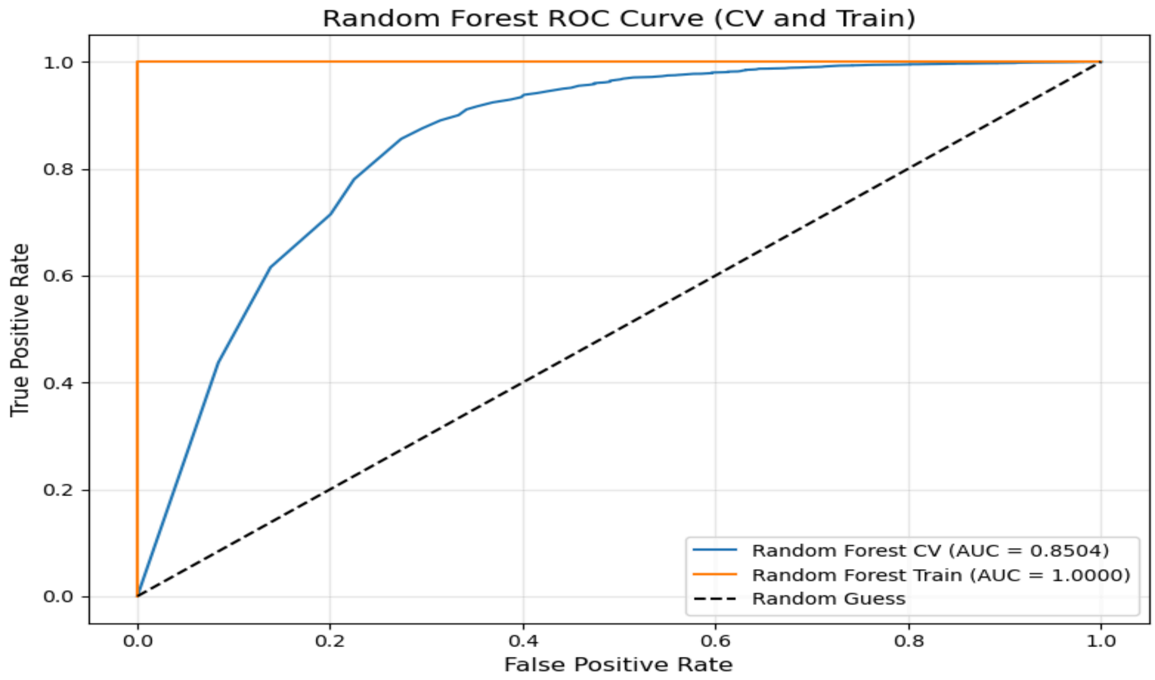
AUC (Area Under Curve): A measure of the model's ability to discriminate between classes.

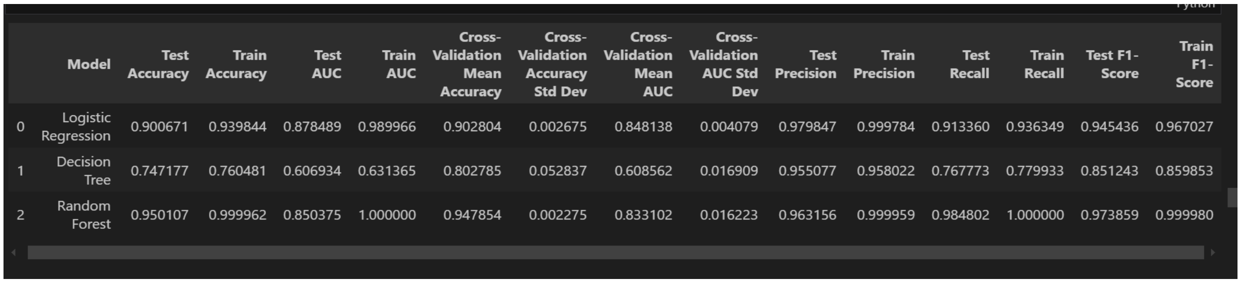
Cross-Validation: Perform K-fold cross-validation to evaluate model performance and reduce the risk of overfitting.

7. Advanced Models

Random Forest:

- **Overview:** Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy and reduce overfitting. Each tree in the forest is built using a random subset of features, which helps in making the model more robust and less prone to overfitting.
- **Use Case:** For the employee access problem, Random Forest can help capture complex relationships between employee roles, access levels, and other features in the data.



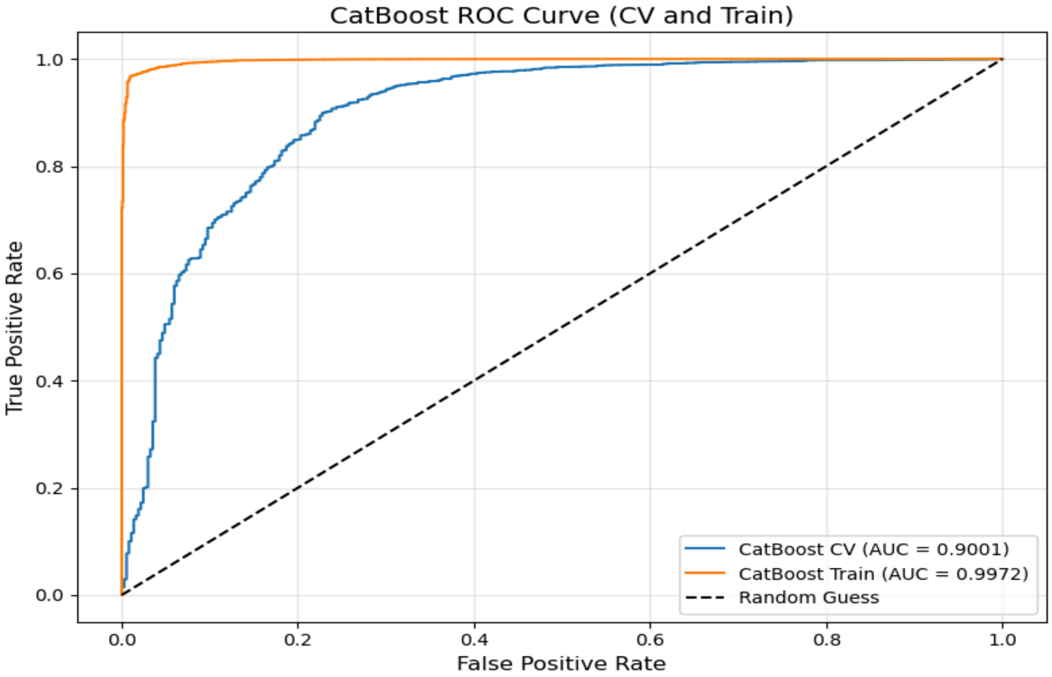


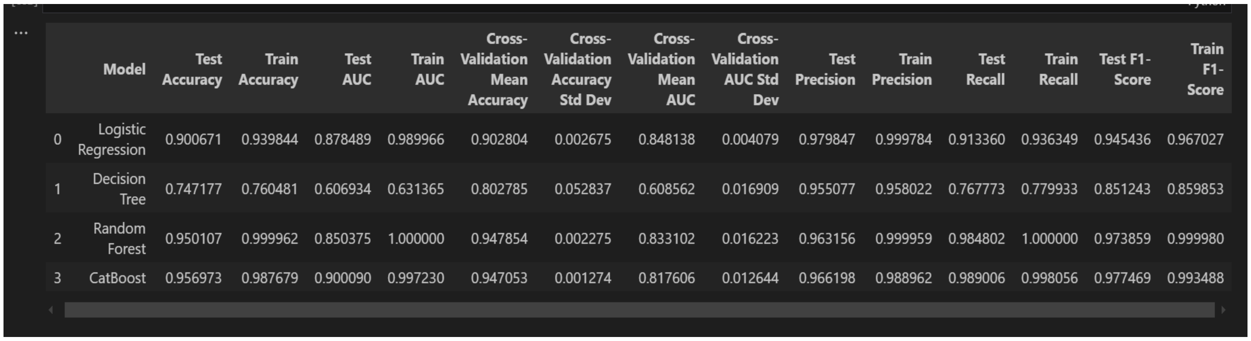
	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.900671	0.939844	0.878489	0.989966	0.902804	0.002675	0.848138	0.004079	0.979847	0.999784	0.913360	0.936349	0.945436	0.967027
1	Decision Tree	0.747177	0.760481	0.606934	0.631365	0.802785	0.052837	0.608562	0.016909	0.955077	0.958022	0.767773	0.779933	0.851243	0.859853
2	Random Forest	0.950107	0.999962	0.850375	1.000000	0.947854	0.002275	0.833102	0.016223	0.963156	0.999959	0.984802	1.000000	0.973859	0.999980

Evaluation matrix of Random Forest

CatBoost:

- **Overview:** CatBoost is a gradient boosting algorithm specifically designed to handle categorical features efficiently. It performs well even with categorical data without needing extensive preprocessing.
- **Use Case:** Since employee role data often contains categorical variables (such as job titles, department names, etc.), CatBoost would be highly effective in this scenario, as it naturally handles these variables without needing manual encoding.



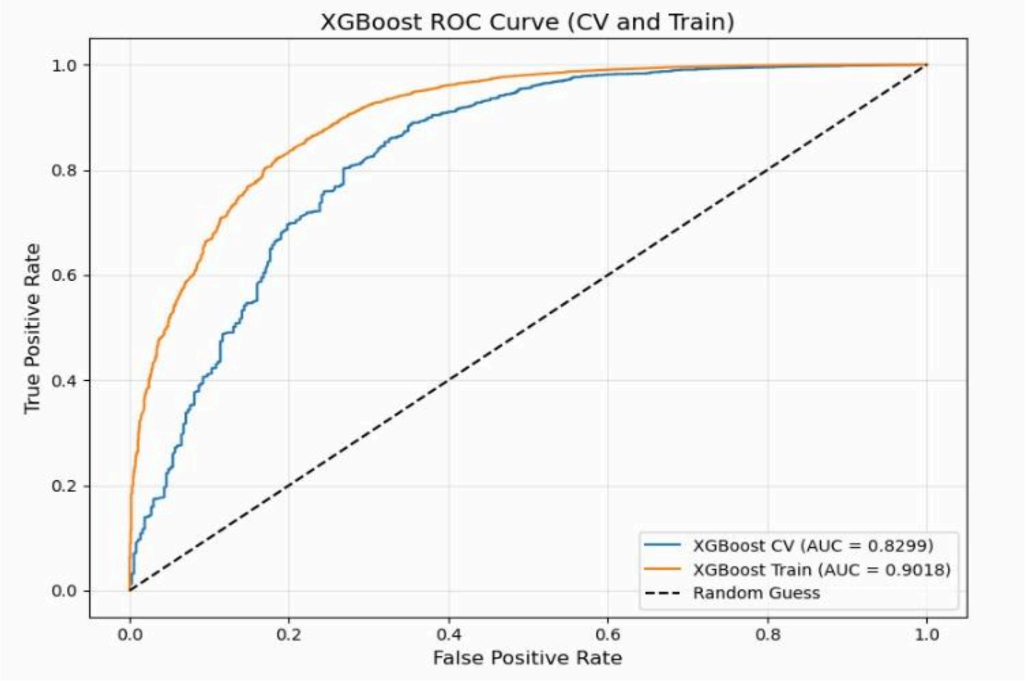


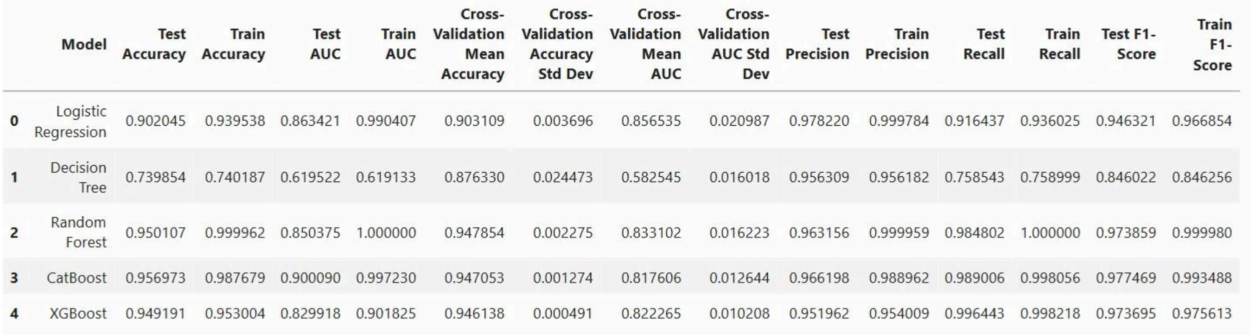
	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.900671	0.939844	0.878489	0.989966	0.902804	0.002675	0.848138	0.004079	0.979847	0.999784	0.913360	0.936349	0.945436	0.967027
1	Decision Tree	0.747177	0.760481	0.606934	0.631365	0.802785	0.052837	0.608562	0.016909	0.955077	0.958022	0.767773	0.779933	0.851243	0.859853
2	Random Forest	0.950107	0.999962	0.850375	1.000000	0.947854	0.002275	0.833102	0.016223	0.963156	0.999959	0.984802	1.000000	0.973859	0.999980
3	CatBoost	0.956973	0.987679	0.900090	0.997230	0.947053	0.001274	0.817606	0.012644	0.966198	0.988962	0.989006	0.998056	0.977469	0.993488

Evaluation matrix of CatBoost

XGBoost:

- **Overview:** XGBoost (Extreme Gradient Boosting) is another powerful gradient boosting framework that aims to optimize both speed and performance. XGBoost uses boosting techniques to combine weak learners into a strong predictive model.
- **Use Case:** XGBoost would be highly effective for predicting employee access needs by leveraging historical data and role-based features. It is often used in machine learning competitions due to its superior performance.





	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.902045	0.939538	0.863421	0.990407	0.903109	0.003696	0.856535	0.020987	0.978220	0.999784	0.916437	0.936025	0.946321	0.966854
1	Decision Tree	0.739854	0.740187	0.619522	0.619133	0.876330	0.024473	0.582545	0.016018	0.956309	0.956182	0.758543	0.758999	0.846022	0.846256
2	Random Forest	0.950107	0.999962	0.850375	1.000000	0.947854	0.002275	0.833102	0.016223	0.963156	0.999959	0.984802	1.000000	0.973859	0.999980
3	CatBoost	0.956973	0.987679	0.900090	0.997230	0.947053	0.001274	0.817606	0.012644	0.966198	0.988962	0.989006	0.998056	0.977469	0.993488
4	XGBoost	0.949191	0.953004	0.829918	0.901825	0.946138	0.000491	0.822265	0.010208	0.951962	0.954009	0.996443	0.998218	0.973695	0.975613

Evaluation matrix of XGBoost

8. Model Evaluation

In this project, we focused on predicting employee access requirements to various resources in Amazon's internal network. After developing and training several machine learning models, we used multiple evaluation metrics to assess the models' performance and determine the best one for predicting employee access requirements. The primary metrics used for evaluation were:

Accuracy:

- **Definition**: Accuracy is the most straightforward evaluation metric. It is the ratio of the number of correct predictions to the total number of predictions.
- **Formula**:
  Accuracy = Number of Correct Predictions / Total Number of Predictions
- **Importance**: Accuracy gives us a sense of how well the model performs in general. However, it can be misleading in imbalanced datasets where one class is much more prevalent than the other.
  Example: If 95% of employees do not access a particular resource, a model predicting that "no one accesses the resource" will still achieve 95% accuracy, despite being useless.

AUC (Area Under Curve):

- **Definition**: AUC refers to the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve plots the true positive rate (recall) against the false positive rate.
- **Formula**: AUC is calculated as the integral of the ROC curve.
- **Importance**: AUC is a useful metric for binary classification problems as it evaluates the model's ability to distinguish between the classes. A higher AUC indicates better model performance, as the model is better at distinguishing between positive and negative classes.
  Example: In this case, a higher AUC would indicate that the model is more effectively predicting which employees should have access to resources.

Cross-Validation:

- **Definition**: Cross-validation is a technique used to assess the performance of a model by training it on different subsets of the dataset and testing it on the remaining data. It helps in understanding how the model will generalize to unseen data.
- **Method**: Stratified K-Fold cross-validation is used, where the dataset is split into K subsets (or folds). The model is trained K times, each time using K-1 subsets for training and the remaining fold for testing. This ensures that the model is evaluated across different subsets of the data.

Importance: Cross-validation helps in reducing the risk of overfitting and provides a more reliable estimate of model performance.

Precision:

- **Definition**: Precision refers to the proportion of positive predictions that are actually correct.
- **Formula**:
  Precision = TP / TP + FP
  Where,
  TP = True Positives (correctly predicted positive instances)
  FP = False Positives (incorrectly predicted as positive)
- **Importance**: Precision is important when the cost of false positives is high. For example, if an employee is incorrectly given access to a resource, it could result in a security breach. High precision means fewer false positives.
  Example: In this context, high precision means that when the model predicts an employee should have access to a resource, they are actually likely to require that access.

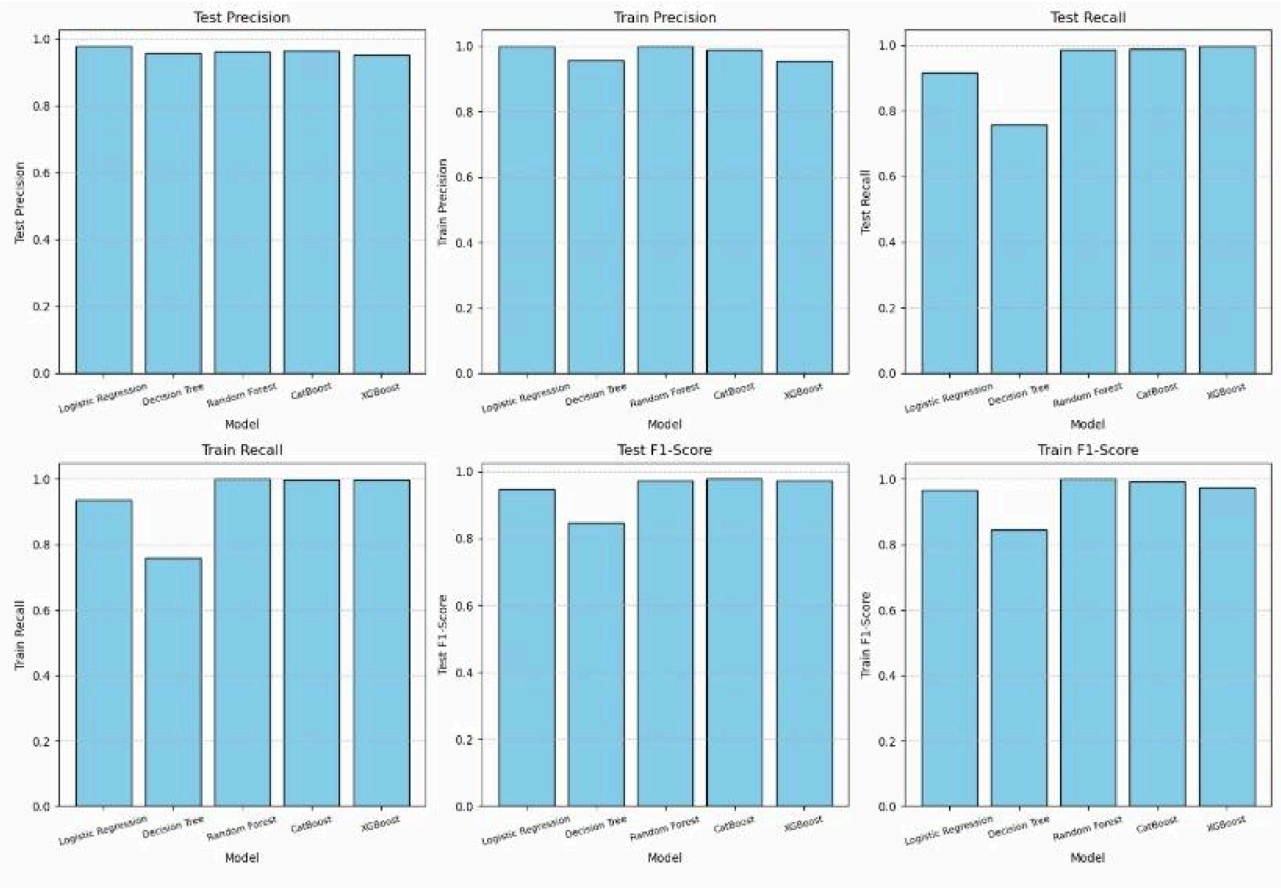
Recall (Sensitivity):

- **Definition**: Recall is the proportion of actual positive instances that were correctly identified by the model.
- **Formula**:
  Recall = TP / TP + FN
  where:
  - TP = True Positives
  - FN = False Negatives (instances that should have been predicted as positive but were missed)
- **Importance**: Recall is important when the cost of false negatives is high. In the context of access control, missing an employee's requirement for access (false negative) could result in them being unable to perform their job. High recall means fewer false negatives.
  Example: High recall means the model is good at identifying employees who need access to a resource, even if it results in a few false positives.

F1-Score:

- **Definition**: The F1-score is the harmonic mean of precision and recall. It is used when you need a balance between precision and recall.
- **Formula**:
  F1= 2 × Precision × Recall / Precision + Recall

- **Importance**: The F1-score is useful when dealing with imbalanced datasets, where you need to balance the cost of false positives and false negatives. It provides a more comprehensive evaluation than accuracy, as it considers both precision and recall.
  Example: If the model has high precision but low recall, or vice versa, the F1-score will help in comparing the trade-off between them and determining the best model.



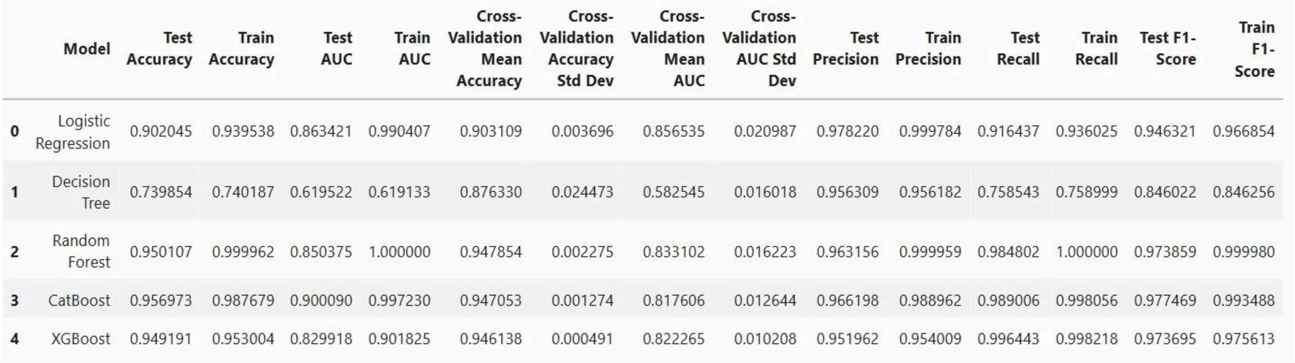
Model Comparison:

To compare the performance of different models, we use the above metrics across classifiers such as:

- **Random Forest Classifier**: A versatile model that can handle both classification and regression tasks, particularly good for high-dimensional data.
- **Logistic Regression**: A simpler linear model often used for binary classification problems.

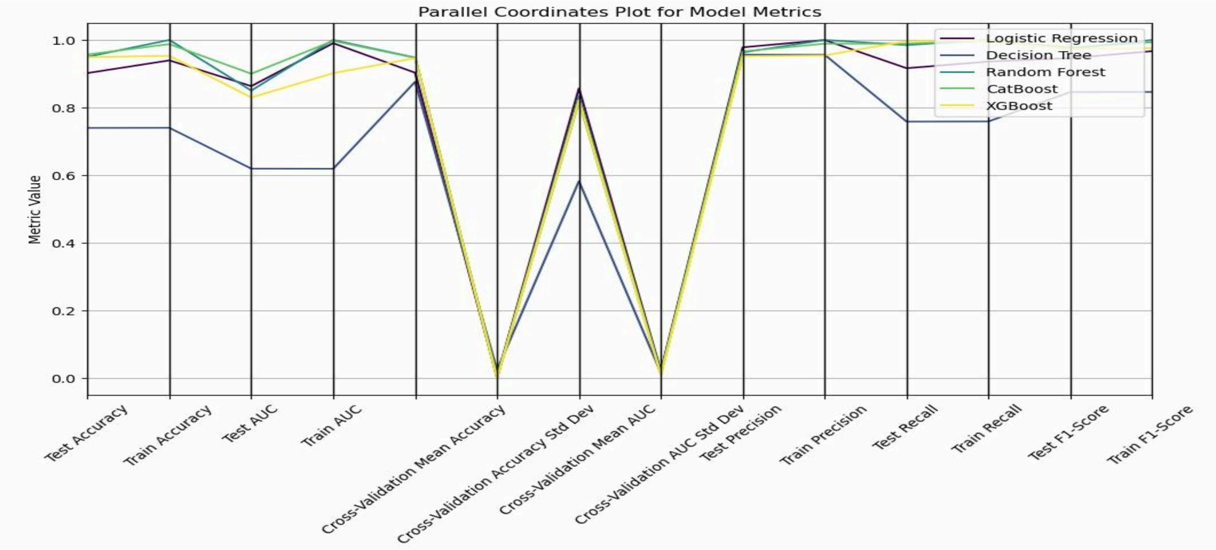
- **CatBoost Classifier**: A gradient boosting classifier known for its handling of categorical features.
- **Decision Tree Classifier**: A model that splits the data based on decision rules, helpful in understanding feature importance.
- **XGBoost** : XGBoost are effective for tabular data and typically deliver high performance in classification tasks like predicting access types.

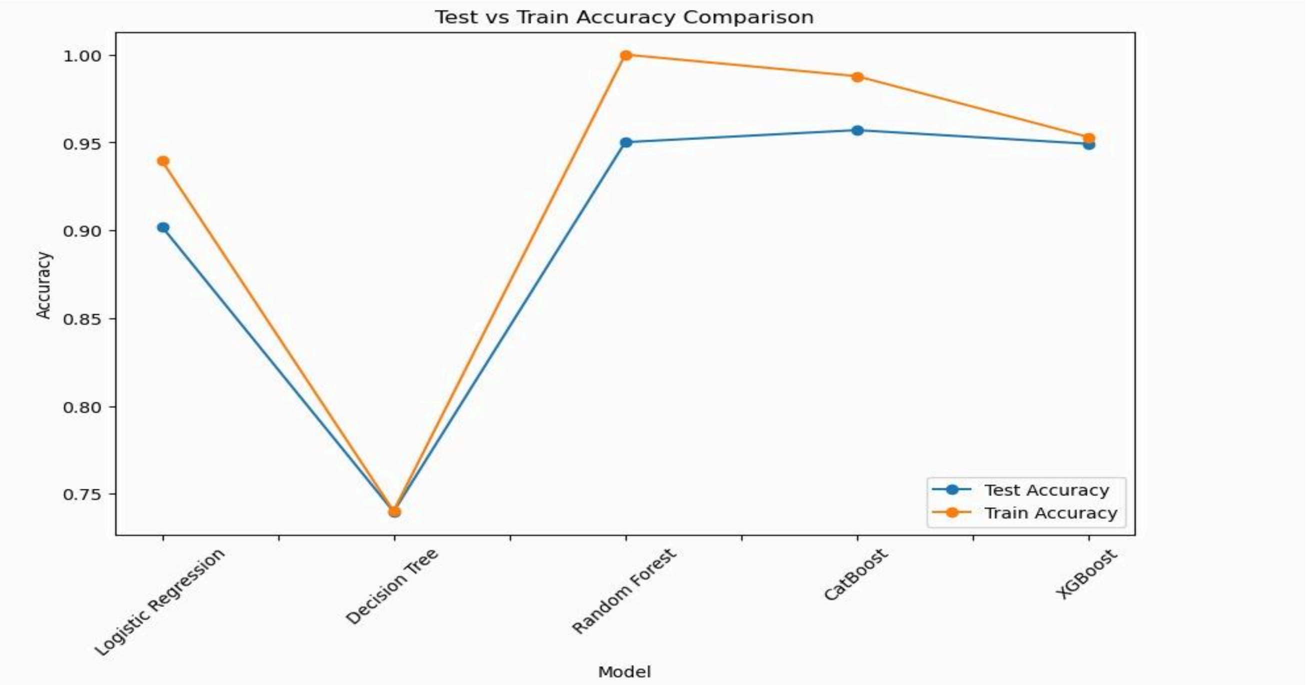
Each model is evaluated using accuracy, precision, recall, F1-score, and AUC, and cross-validation is applied to assess the robustness of the model.



	Model	Test Accuracy	Train Accuracy	Test AUC	Train AUC	Cross-Validation Mean Accuracy	Cross-Validation Accuracy Std Dev	Cross-Validation Mean AUC	Cross-Validation AUC Std Dev	Test Precision	Train Precision	Test Recall	Train Recall	Test F1-Score	Train F1-Score
0	Logistic Regression	0.902045	0.939538	0.863421	0.990407	0.903109	0.003696	0.856535	0.020987	0.978220	0.999784	0.916437	0.936025	0.946321	0.966854
1	Decision Tree	0.739854	0.740187	0.619522	0.619133	0.876330	0.024473	0.582545	0.016018	0.956309	0.956182	0.758543	0.758999	0.846022	0.846256
2	Random Forest	0.950107	0.999962	0.850375	1.000000	0.947854	0.002275	0.833102	0.016223	0.963156	0.999959	0.984802	1.000000	0.973859	0.999980
3	CatBoost	0.956973	0.987679	0.900090	0.997230	0.947053	0.001274	0.817606	0.012644	0.966198	0.988962	0.989006	0.998056	0.977469	0.993488
4	XGBoost	0.949191	0.953004	0.829918	0.901825	0.946138	0.000491	0.822265	0.010208	0.951962	0.954009	0.996443	0.998218	0.973695	0.975613

Evaluation matrix of 5 models





Test vs Train Accuracy comparison of 5 models

9. Challenges and Solutions

The **Amazon Employee Access Challenge** presents several challenges when building a model to automate access management. These challenges stem from data complexities, model performance, and ensuring that the predictions align with organizational needs. Below are some key issues encountered during the project and the solutions implemented to address them.

A. Data Imbalance (Class Imbalance in ACTION):

Challenge: The dataset is likely to have a class imbalance, where the number of access approvals (ACTION = 1) may significantly differ from the number of access denials (ACTION = 0). This imbalance can lead to biased models that favor the majority class, making predictions less accurate for the minority class.

Solution: To address the class imbalance, several techniques can be used:

- **Resampling**: Over-sample the minority class (denied access) or under-sample the majority class (approved access) to balance the dataset.
- **Synthetic Data Generation**: Use techniques like SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic examples for the minority class.
- **Class Weighting**: Apply class weights to the loss function to penalize incorrect predictions of the minority class more heavily, encouraging the model to focus on both classes equally.

B. Missing or Incomplete Data:

Challenge: Missing data, especially in key features like ROLE_DEPTNAME, ROLE_TITLE, and MANAGER_ID, can hinder the model's ability to make accurate predictions.

Solution:

- **Imputation**: Use various imputation strategies such as mean, median, or mode imputation for numerical or categorical features with missing values.
- **Predictive Imputation**: Use machine learning algorithms like k-nearest neighbors (KNN) or regression models to predict missing values based on other available features.
- **Feature Engineering**: Create new features to fill gaps, such as grouping roles into broader categories or inferring missing department names from role descriptions.

C. Handling Categorical Data:

Challenge: Several features in the dataset, such as ROLE_DEPTNAME, ROLE_TITLE, and ROLE_FAMILY_DESC, are categorical. These features need to be properly encoded for the model to process them.

Solution:

- **One-Hot Encoding**: For nominal categorical variables like ROLE_DEPTNAME, use one-hot encoding to represent each category as a separate binary column.
- **Label Encoding**: For ordinal features (if applicable), like ROLE_ROLLUP, use label encoding to assign each category a unique numerical value.
- **Embedding Techniques**: For high cardinality categorical variables, consider using embedding techniques like target encoding or neural network embeddings.

D. Feature Selection and Importance:

Challenge: The dataset contains several features, and not all may contribute equally to the model's ability to predict access decisions. Selecting the most important features without overfitting can be difficult.

Solution:

- **Feature Engineering**: Conduct exploratory data analysis (EDA) to better understand relationships between features and target. Use domain knowledge to create meaningful interactions between variables.
- **Feature Importance**: Use tree-based models like Random Forest or XGBoost to compute feature importance, helping identify which features have the most impact on the prediction.
- **Regularization**: Apply regularization techniques (e.g., L1/L2 regularization) to prevent overfitting and to reduce the dimensionality of the feature space.

E. Model Interpretability and Transparency:

Challenge: As the model is used to determine whether access should be granted or denied, it’s important that the model's decisions are interpretable, especially in an organizational context where access decisions could impact security or workflow.

Solution:

- **Explainable Models**: Use models like Logistic Regression or Decision Trees that are more interpretable compared to black-box models like deep neural networks.

- **Model-Agnostic Tools**: Utilize tools like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to explain the predictions of more complex models.
- **Human-Readable Insights**: Provide transparency by presenting not only the prediction but also insights into which features influenced the decision.

F. Overfitting and Model Generalization:

Challenge: Overfitting can occur when the model learns the noise or irrelevant patterns in the training data, making it perform poorly on unseen data.

Solution:

- **Cross-Validation**: Use cross-validation to ensure that the model generalizes well across different subsets of the data.
- **Regularization**: Implement regularization techniques (like L1/L2) in models to penalize overly complex models.
- **Early Stopping**: In case of using deep learning models, apply early stopping to prevent the model from training too long and overfitting.

G. Data Privacy and Security:

Challenge: Employee data is sensitive, and using historical access decisions could introduce security risks or concerns about privacy.

Solution:

- **Secure Data Handling**: Implement secure data handling practices, such as encryption, when storing or processing sensitive information.
- **Compliant Access Control**: Ensure that the model adheres to data protection regulations like GDPR or HIPAA, ensuring the privacy and security of employee data.

H. Predicting Access for Unseen Roles:

Challenge: The model may encounter new or unseen roles in the test set that it has not been trained on, which could reduce its performance.

Solution:

- **Use of Transfer Learning**: In more complex models, use transfer learning techniques to adapt pre-trained models to the new roles.

- **Regular Updates**: Continuously update the training data with new roles and employee transitions to keep the model current.
- **Hybrid Approaches**: Combine rule-based systems with machine learning models to handle unseen or rare cases more effectively.

10. Next Steps

After performing exploratory data analysis (EDA) and preprocessing (including one-hot encoding), the next steps involve advanced modeling, evaluation, and validation strategies. Here's a detailed plan:

Advanced Modeling Plans:

- **Hybrid and Ensemble Models: Why?**
    - Ensemble methods combine the predictions of multiple models, leveraging their individual strengths to create a more accurate and robust model. For example:
    - A Decision Tree might capture simple patterns, while Gradient Boosting can model more complex patterns.
    - Combining models (e.g., through stacking or blending) reduces the risk of overfitting or underfitting compared to using a single model.

- **AutoML Solutions: Why?**
    - AutoML automates the process of model selection, hyperparameter tuning, and pipeline optimization, saving significant time and effort.
    - Instead of manually experimenting with different algorithms, AutoML solutions explore a wide range of models (e.g., SVMs, Gradient Boosting, Neural Networks) and optimize them automatically.

Evaluation and Validation Strategies:

A. Cross-Validation

- Why Cross-Validation?
    - ○ Cross-validation helps ensure the model generalizes well to unseen data by splitting the training set into multiple folds and evaluating performance on each fold.

B. Performance Metrics

- **Accuracy:** Percentage of correctly predicted access types.
- **Precision, Recall, F1-Score:** Useful if the access types are imbalanced.
- **Confusion Matrix:** To understand where the model is making misclassifications.

C. Hyperparameter Tuning Evaluation

- After hyperparameter tuning, re-evaluate the model using the best set of parameters on the test data and check if performance has improved.

Timeline for Project Completion:

Phase 1: Data Exploration & Preprocessing (1-2 Weeks)

- **Tasks:**
  - Perform initial data analysis (EDA).
  - Handle missing values, encode categorical features, and perform feature engineering.
- **Deliverables:** Cleaned and preprocessed dataset ready for modeling.

Phase 2: Model Development (2-3 Weeks)

- **Tasks:**
  - Implement and train different models (e.g., Gradient Boosting, Random Forest, Logistic Regression).
  - Perform hyperparameter tuning and model selection.
- **Deliverables:** Trained models with hyperparameters optimized, ready for evaluation.

Phase 3: Evaluation & Validation (1 Week)

- **Tasks:**
  - Conduct cross-validation and compute performance metrics.
  - Analyze confusion matrix and fine-tune the model if necessary.
- **Deliverables:** Final evaluation report with performance metrics.

Phase 4: Model Deployment and Final Report (1 Week)

- **Tasks:**
  - Prepare model for deployment (if applicable).
  - Create a final report documenting the methodology, results, and conclusions.
- **Deliverables:** Deployed model (optional) and final project report.

The Responsibility of Each Team Member:

- Dhruvi: Conduct exploratory data analysis (EDA)
- Drashti: Build initial baseline models (e.g., logistic regression, decision trees) to establish performance benchmarks.
- Nikhil: Advanced Model Development and Experimentation
- Jaydeep: Advanced Model Development and Hyperparameter Tuning and Team Lead
- Kinnari: Model Evaluation and Validation

Summary of Key Considerations:

- **Data Quality:** Ensure thorough preprocessing, including handling missing values and encoding categorical variables.
- **Model Performance:** Focus on both accuracy and business relevance (e.g., minimizing false positives/negatives in access assignment).
- **Model Interpretability:** Depending on the application, you may want to ensure your model is interpretable, especially for explaining access decisions.

11. Conclusion

Based on the metrics and visualizations, the following key insights are drawn from the comparison of machine learning models on metrics such as Test Accuracy, Train Accuracy, Test AUC, Train AUC, Cross-Validation Accuracy, Precision, Recall, and F1-scores:

Model Performance Insights:

- **CatBoost**:
  - Emerges as the best-performing model with the highest **Test Accuracy (95.70%)** and **Test AUC (0.90)**.
  - It demonstrates a strong balance between training and testing metrics, indicating minimal overfitting and high generalizability.
- **Random Forest**:
  - Achieves a high **Test Accuracy (95.01%)** and near-perfect **Train Accuracy (99.99%)**, showcasing its effectiveness.
  - However, it exhibits slight overfitting compared to CatBoost.
- **XGBoost**:
  - Follows closely with **Test Accuracy (94.91%)** and **Test AUC (0.83)**.
  - Its performance, while strong, is slightly behind CatBoost and Random Forest.
- **Logistic Regression**:
  - Despite being a simpler model, it delivers a solid **Test Accuracy (90.46%)** and **Test AUC (0.87)**.
  - This makes it a competitive option in scenarios requiring interpretability and low complexity.
- **Decision Tree**:
  - Performs relatively poorly, with a **Test AUC (0.60)** and **Test Accuracy (84.75%)**, indicating limited capacity to capture complex patterns without ensemble techniques.
  - It also shows significant variability in cross-validation metrics, making it less reliable.

General Observations:

- **CatBoost** stands out as the most reliable model due to its high performance on both test and train datasets and low cross-validation standard deviation, reflecting consistency and robustness.
- **Ensemble models** (Random Forest, CatBoost, and XGBoost) consistently outperform simpler models (Logistic Regression and Decision Tree) across all metrics.
- **Overfitting** is observed in Random Forest, as seen by its perfect Train Accuracy and Train AUC, but it remains competitive due to its high Test Accuracy and Test Precision.

Cross-Validation Results:

- Cross-validation metrics reaffirm **CatBoost** and **XGBoost** as reliable models, with low variability (standard deviation) in accuracy and AUC scores, ensuring consistent performance across different data splits.

12. References

1. **Amazon Employee Access Challenge Data**: https://www.kaggle.com/competitions/amazon-employee-access-challenge/data
2. **Kaggle Competition Overview**: https://www.kaggle.com/competitions/amazon-employee-access-challenge/overview
3. **XGBoost Documentation**: https://xgboost.readthedocs.io/en/latest/
4. **GridSearchCV Documentation**: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
5. **Pandas One-Hot Encoding**: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
6. **Scikit-learn Classification Report**: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
7. **Scikit-learn Cross-Validation**: https://scikit-learn.org/stable/modules/cross_validation.html
8. **Feature Engineering in Machine Learning**: https://towardsdatascience.com/feature-engineering-techniques-in-machine-learning-with-python-f24e7da3f36e