**Exercise 1: Read the dataset [0.5 Mark]**

**Hint:** pd.read\_csv()

# read the dataset

# YOUR CODE HERE

import pandas as pd

# The file is named 'wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv' and is in the same directory

#file\_name = 'wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv'

#dataset = pd.read\_csv(file\_name, delimiter='\t') # TSV format uses tab as delimiter

# Display the first few rows to ensure it's loaded correctly

#print(dataset.head(10))

# Reading the dataset as CSV instead of TSV

dataset = pd.read\_csv(file\_name, delimiter=',') # Using comma as delimiter

# Check the first few rows of the dataset to ensure it's loaded properly:

print("Dataset head:")

print(dataset.head())

print("\n------------------------\n")

Explanation of the above code :

It seems like you have already read the dataset as a CSV with a comma delimiter, and the code you provided is correct for reading a CSV file. If the dataset is indeed in CSV format and you want to use a comma as the delimiter, your current code will work as expected.

If you ever need to read the same dataset as a TSV file using tab as the delimiter, you can uncomment the original code and use that. Otherwise, your current code should correctly read the dataset as a CSV.

# Check the shape of dataframe.

# YOUR CODE HERE

#print(dataset.shape)

dataset.shape

**Exercise 2: Create a List of numerical and categorical columns. Display a statistical description of the dataset. Remove missing values (if any) [0.5 Mark]**

**Hint:** Use for to iterate through each column.

# YOUR CODE HERE

import pandas as pd

# Assuming you've already read the dataset:

# dataset = pd.read\_csv(file\_name, delimiter='\t')

# Create empty lists for numerical and categorical columns

numerical\_columns = []

categorical\_columns = []

# Iterate through columns to classify them as numerical or categorical using a for loop

for column in dataset.columns:

if dataset[column].dtype in ['int64', 'float64']:

numerical\_columns.append(column)

else:

categorical\_columns.append(column)

print("Numerical Columns:", numerical\_columns)

print("Categorical Columns:", categorical\_columns)

# Display a statistical description of the dataset

print("\nStatistical Description of the Dataset:\n")

print(dataset.describe(include='all'))

# Remove rows with missing values

missing\_before = dataset.isnull().sum().sum()

dataset.dropna(inplace=True)

missing\_after = dataset.isnull().sum().sum()

print(f"\nNumber of missing values before removal: {missing\_before}")

print(f"Number of missing values after removal: {missing\_after}")

Your code appears to be performing several tasks related to data preprocessing and analysis. Here's a breakdown of what each part does:

1. **Classify Columns as Numerical or Categorical:**
   * The code first separates columns in the dataset into two lists: **numerical\_columns** and **categorical\_columns** based on their data types.
   * Numerical columns are identified by data types 'int64' and 'float64', while all other data types are considered categorical.
2. **Display Statistical Description:**
   * The code prints out a statistical description of the entire dataset using the **describe** method. This includes summary statistics for numerical columns and count, unique, top, and freq for categorical columns.
3. **Remove Rows with Missing Values:**
   * The code calculates the number of missing values in the dataset before and after removing rows with missing values using the **dropna** method.
   * The **inplace=True** argument ensures that the changes are applied directly to the dataset.

Overall, your code is a good starting point for data exploration and preprocessing, especially for understanding the data types, summary statistics, and handling missing values in your dataset.

Top of Form

**Exercise 3: Create a box plot to check for outliers [0.5 Mark]**Bottom of Form

# Check for outliers

# YOUR CODE HERE

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming you've already read your dataset into a DataFrame named 'dataset'

# Extract numerical columns if you haven't already

numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]

# Create box plots for each numerical column and print the outliers

for column in numerical\_columns:

plt.figure(figsize=(10, 6))

sns.boxplot(dataset[column])

plt.title(f"Box plot of {column}")

plt.show()

# Calculate IQR for the column

Q1 = dataset[column].quantile(0.25)

Q3 = dataset[column].quantile(0.75)

IQR = Q3 - Q1

# Determine bounds for outliers

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

# Extract outliers

outliers = dataset[(dataset[column] < lower\_bound) | (dataset[column] > upper\_bound)][column]

# Print outliers if they exist

if outliers.empty:

print(f"No outliers found for {column}.\n")

else:

print(f"Outliers for {column}:")

print(outliers, "\n")

Your code is designed to check for outliers in numerical columns of your dataset. Here's how it works:

1. It first assumes that you have already read your dataset into a DataFrame named **dataset**.
2. It extracts the numerical columns by checking the data types ('int64' or 'float64').
3. For each numerical column, it creates a box plot using Seaborn, which is a useful visualization for identifying outliers.
4. It calculates the Interquartile Range (IQR) for the column. The IQR is used to determine the bounds for outliers.
5. It defines lower and upper bounds for outliers as 1.5 times the IQR below the first quartile (Q1) and above the third quartile (Q3), respectively.
6. It identifies outliers in the column by selecting rows where the column values fall outside the calculated bounds.
7. If there are no outliers for a column, it prints a message indicating that. If there are outliers, it lists them.

This code is a good approach to identify and examine potential outliers in your numerical columns using box plots and the IQR method. It helps you understand the distribution of data and detect values that significantly deviate from the majority of the data.

Top of Form

**Exercise 4: Use lower bound as 25% and upper bound as 75% to handle the outliers [0.5 Mark]**Bottom of Form

# YOUR CODE HERE

import pandas as pd

# Assuming you've already read your dataset into a DataFrame named 'dataset'

# Check the first few rows of the dataset to ensure it's loaded properly:

print("Dataset head:")

print(dataset.head())

print("\n------------------------\n")

# Extract numerical columns if you haven't already

numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]

# Display the numerical columns to verify:

print("Numerical columns:")

print(numerical\_columns)

print("\n------------------------\n")

# For each numerical column, cap the data with the 25% and 75% quantiles

for column in numerical\_columns:

Q1 = dataset[column].quantile(0.25)

Q3 = dataset[column].quantile(0.75)

# Replacing values below Q1 with Q1 and values above Q3 with Q3

dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))

# Check the revised data

print("Revised Data Description:")

print(dataset[numerical\_columns].describe())

Your code performs a data transformation operation to cap the values of numerical columns with the 25th and 75th percentiles (Q1 and Q3). Here's how it works:

1. It assumes that you have already read your dataset into a DataFrame named **dataset**.
2. It prints the first few rows of the dataset to ensure that it has been loaded properly.
3. It extracts the numerical columns by checking the data types ('int64' or 'float64') and displays the list of numerical columns.
4. For each numerical column, it calculates the 25th and 75th percentiles (Q1 and Q3) using the **quantile** method.
5. It applies a lambda function to each value in the column. If a value is below Q1, it is replaced with Q1. If a value is above Q3, it is replaced with Q3. Otherwise, the value remains unchanged.
6. After this operation, the code prints the statistical description of the revised data using the **describe** method for the numerical columns.

This code effectively caps the extreme values in numerical columns by setting them to the corresponding quartile values (Q1 or Q3). This can be useful for mitigating the influence of outliers in the dataset and ensuring that extreme values do not skew statistical analyses.

Top of Form

# Recheck for outliers

# YOUR CODE HERE

import matplotlib.pyplot as plt

import seaborn as sns

# Setting the size for the plots

plt.figure(figsize=(20, 10))

# Plotting boxplots for all numerical columns

for i, column in enumerate(numerical\_columns, 1):

plt.subplot(2, len(numerical\_columns)//2, i)

sns.boxplot(y=dataset[column])

plt.title(column)

plt.ylabel('Value')

plt.tight\_layout()

plt.show()

In this code, you are rechecking for outliers in the numerical columns of your dataset after capping extreme values with the quartiles (Q1 and Q3). Here's how the code works:

1. You set the size for the plots using **plt.figure(figsize=(20, 10))** to make the plots larger and more readable.
2. You iterate through each numerical column in the **numerical\_columns** list.
3. For each numerical column, you create a subplot using **plt.subplot**. The **enumerate** function is used to get both the index (**i**) and the column name (**column**) from the list of numerical columns. The number of rows and columns in the subplot grid is determined by **len(numerical\_columns)//2**, which is used to create a 2x2 grid of subplots.
4. Inside each subplot, you create a box plot using Seaborn's **sns.boxplot**. The box plot visualizes the distribution of values in the numerical column and helps identify outliers.
5. You set the title of each subplot to the name of the column and label the y-axis as "Value" for clarity.
6. **plt.tight\_layout()** is used to ensure that the subplots are properly spaced and do not overlap.
7. Finally, you use **plt.show()** to display all the box plots in a single figure.

This code provides a visual representation of the data distribution in each numerical column, making it easier to identify outliers and understand the overall data distribution after capping extreme values with quartiles.

Top of Form

**Exercise 5: Check if there is an imbalance in target label [0.5 Mark]**Bottom of Form

**Hint:** Use value\_counts()

# Count of unique values in Attrition column

# YOUR CODE HERE

# Checking the distribution of the target label

print(dataset['attrition'].value\_counts())

# Plotting the distribution

plt.figure(figsize=(8, 5))

sns.countplot(data=dataset, x='attrition')

plt.title("Distribution of Target Label - Attrition")

plt.xlabel('Attrition')

plt.ylabel('Count')

plt.show()

In this code, you are examining the distribution of unique values in the "Attrition" column and creating a count plot to visualize it. Here's how the code works:

1. You use the **value\_counts()** method on the "attrition" column to count the unique values (in this case, likely "Yes" and "No") and display the count of each value.
2. You then create a count plot using Seaborn's **sns.countplot**. This type of plot is suitable for visualizing the distribution of categorical data.
3. Inside the count plot, you specify the data source (**data=dataset**) and the column to plot (**x='attrition'**).
4. You set the title, x-label, and y-label to provide context and make the plot informative.
5. Finally, you use **plt.show()** to display the count plot.

This code allows you to see the distribution of the "Attrition" column, which is a categorical target label, and visually inspect the balance or imbalance between its unique values.

# Plot barplot to visualize balance/imbalance

# YOUR CODE HERE

import matplotlib.pyplot as plt

import seaborn as sns

# Plotting the distribution

plt.figure(figsize=(8, 5))

sns.countplot(data=dataset, x='attrition')

plt.title("Distribution of Target Label - Attrition")

plt.xlabel('Attrition')

plt.ylabel('Count')

plt.show()

The code you provided is correct for creating a count plot (barplot) to visualize the balance or imbalance of the "Attrition" column. It uses Seaborn's **sns.countplot** to display the distribution of this categorical variable. The code is the same as the previous one you provided, and it effectively plots the distribution of the "Attrition" column with counts for each unique value. This type of plot is useful for understanding the class distribution in a classification problem, which is often essential for model training and evaluation.

**Exercise 6: Visualize the relationships between the predictor variables and the target variable using a pairplot [0.5 Mark]**

**Hint:** Use sns.pairplot

# Visualize a pairplot with relevant features

# YOUR CODE HERE

import seaborn as sns

# Selecting a few features for demonstration purposes. Adjust this list as needed.

selected\_features = ['age', 'dailyrate', 'distancefromhome', 'monthlyincome']

# Adding the target label to visualize the relationships based on the target classes

selected\_features.append('attrition')

# Creating the pairplot

sns.pairplot(dataset[selected\_features], hue='attrition', plot\_kws={'alpha':0.5})

plt.show()

In this code, you are creating a pairplot to visualize relationships between selected features and the target label "attrition." Here's how the code works:

1. You define a list of selected features that you want to include in the pairplot. Adjust this list as needed to include the features you are interested in.
2. You append the "attrition" column to the selected features list. This is done to visualize the relationships between these features while distinguishing data points by the "attrition" class.
3. You create the pairplot using Seaborn's **sns.pairplot** function. The **dataset[selected\_features]** part selects only the columns specified in the **selected\_features** list, and **hue='attrition'** colors the data points based on the "attrition" class. The **plot\_kws={'alpha':0.5}** argument controls the transparency of data points in the plot.
4. Finally, you use **plt.show()** to display the pairplot.

This pairplot allows you to visualize the relationships and distributions of selected features with respect to the "attrition" class, helping you identify patterns and trends in the data that may be relevant for your analysis or modeling. Adjust the list of selected features as needed to explore different aspects of the dataset.

Top of Form

**Exercise 7: Visualize the correlation among IBM employee attrition numerical features using a heatmap [0.5 Mark]**Bottom of Form

# Visualize heatmap

# YOUR CODE HERE

import matplotlib.pyplot as plt

import seaborn as sns

# Compute the correlation matrix

corr\_matrix = dataset[numerical\_columns].corr()

# Create a heatmap to visualize the correlations

plt.figure(figsize=(15, 10))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)

plt.title("Correlation Heatmap of Numerical Features")

plt.show()

The code you provided generates a heatmap to visualize the correlations between numerical features in your dataset. Here's how the code works:

1. You compute the correlation matrix using the **corr()** method on the subset of numerical columns in your dataset. The correlation matrix represents the pairwise correlations between these numerical features.
2. Next, you create the heatmap using Seaborn's **sns.heatmap** function. You pass in the **corr\_matrix** as the data to visualize.
3. The **annot=True** parameter adds numeric values in each cell of the heatmap, indicating the correlation coefficient between the corresponding pair of features.
4. You specify the color map (**cmap='coolwarm'**) for the heatmap to use cooler colors for negative correlations and warmer colors for positive correlations. The **vmin** and **vmax** parameters set the range of the color scale to -1 and 1, respectively, ensuring that correlations outside this range are still properly visualized.
5. Finally, you add a title to the heatmap using **plt.title()** and display it with **plt.show()**.

This heatmap allows you to visually assess the strength and direction of correlations between numerical features in your dataset, which can be valuable for feature selection and understanding relationships in your data.

Top of Form

Part-B

### **Preparing the test feature space**

* Remove outliers if any
* Handle the categorical feature if required
* Other processing steps can also be followed.
* # YOUR CODE HERE
* #Removing Outliers
* # Import necessary libraries
* # Import necessary libraries
* import pandas as pd
* # Download the dataset
* !wget -qq https://cdn.iisc.talentsprint.com/CDS/Datasets/wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv
* print("Data Downloaded Successfully!!")
* # Load dataset
* file\_name = 'wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv' # This is the file path after downloading
* dataset = pd.read\_csv(file\_name, delimiter=',')
* # Identify numerical columns
* numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]
* # Removing Outliers
* for column in numerical\_columns:
* Q1 = dataset[column].quantile(0.25)
* Q3 = dataset[column].quantile(0.75)
* dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))
* # Handle Categorical Features
* categorical\_columns = [column for column in dataset.columns if dataset[column].dtype == 'object' and column != 'attrition'] # Assuming 'attrition' is the target column
* dataset = pd.get\_dummies(dataset, columns=categorical\_columns, drop\_first=True) # One-hot encoding
* # Display the updated dataset
* print(dataset.head())

It seems like you are performing data preprocessing tasks, including the removal of outliers and handling categorical features, on the "wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv" dataset. Here's a breakdown of what your code does:

1. **Data Download**:
   * It downloads the dataset from the provided URL using the **wget** command.
2. **Loading the Dataset**:
   * It loads the downloaded dataset into a Pandas DataFrame named **dataset** using the **pd.read\_csv** method with a comma (',') as the delimiter.
3. **Identifying Numerical Columns**:
   * It identifies numerical columns by checking the data types ('int64' or 'float64') and stores them in the **numerical\_columns** list.
4. **Removing Outliers**:
   * For each numerical column, it calculates the first quartile (Q1) and third quartile (Q3) using the **quantile** method.
   * It replaces values below Q1 with Q1 and values above Q3 with Q3 for each numerical column. This effectively caps extreme values.
5. **Handling Categorical Features**:
   * It identifies categorical columns by checking for data type 'object' and excluding the "attrition" column (assuming it's the target).
   * It performs one-hot encoding on the categorical columns using **pd.get\_dummies**. The **drop\_first=True** argument is used to create dummy variables, and the original categorical columns are dropped.
6. **Displaying the Updated Dataset**:
   * It displays the first few rows of the updated dataset to show the results of data preprocessing.

This code prepares the dataset by removing outliers and transforming categorical features into a format suitable for machine learning models, which is a common preprocessing step in data analysis and modeling tasks.

Top of Form

**Exercise 8: Data processing for CatBoost [1 Mark]**

* **Copy the dataframe that was created after removing the outliers**
* **Handle the categorical features if required**
* **Create target column and feature space**

**Hint:** Column containing the information on attrition will be the target column.Bottom of Form

# Copy the data

# YOUR CODE HERE

# Copy the data

# YOUR CODE HERE

# YOUR CODE HERE

import pandas as pd

import numpy as np

# Read the dataset

file\_name = "https://cdn.iisc.talentsprint.com/CDS/Datasets/wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv"

dataset = pd.read\_csv(file\_name, delimiter=',')

# Remove outliers

numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]

for column in numerical\_columns:

Q1 = dataset[column].quantile(0.25)

Q3 = dataset[column].quantile(0.75)

dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))

# Copying the dataframe

df\_catboost = dataset.copy()

# Identify categorical columns

cat\_features\_indices = np.where(df\_catboost.dtypes != np.float64)[0]

# Splitting dataset into features and target

X\_catboost = df\_catboost.drop("attrition", axis=1) # Assuming "attrition" is the target column

y\_catboost = df\_catboost["attrition"]

print(f"Categorical feature indices: {list(cat\_features\_indices)}")

In this code, you are copying the dataset and preparing it for use with the CatBoost library. Here's a breakdown of what your code does:

1. **Read the Dataset**:
   * It reads the dataset from the provided URL using the **pd.read\_csv** method with a comma (',') as the delimiter and stores it in the **dataset** DataFrame.
2. **Remove Outliers**:
   * It identifies numerical columns by checking the data types ('int64' or 'float64') and stores them in the **numerical\_columns** list.
   * For each numerical column, it calculates the first quartile (Q1) and third quartile (Q3) using the **quantile** method.
   * It replaces values below Q1 with Q1 and values above Q3 with Q3 for each numerical column, effectively capping extreme values.
3. **Copy the DataFrame**:
   * It makes a copy of the dataset DataFrame and assigns it to a new DataFrame named **df\_catboost**.
4. **Identify Categorical Columns**:
   * It identifies categorical columns by checking for data types other than **np.float64**. These columns are considered categorical, and their indices are stored in the **cat\_features\_indices** array.
5. **Split Dataset**:
   * It splits the dataset into features (X\_catboost) and the target variable (y\_catboost) by dropping the "attrition" column from the copied DataFrame and assigning it to the target variable.
6. **Print Categorical Feature Indices**:
   * It prints the indices of the categorical features, which are identified based on their data types.

This code prepares the dataset for use with the CatBoost library, ensuring that numerical columns are capped for outliers, and categorical features are identified for proper handling by CatBoost during training.

Top of Form

Bottom of Form

Bottom of Form

Bottom of Form

**Output:**

**Categorical feature indices: [1, 2, 4, 7, 8, 11, 15, 17, 21, 22, 26]**

The code has successfully identified the categorical feature indices, and here they are:

Categorical feature indices: [1, 2, 4, 7, 8, 11, 15, 17, 21, 22, 26]

These indices represent the positions of the categorical features in your dataset, which can be used when working with machine learning models like CatBoost that require the categorical features to be treated differently from numerical features during training.

**Exercise 9: Define, train the model and display the results [2 Mark]**

**Hint:**

* Use CatBoostClassifier() to define the model with relevant parameters.
* Use fit to fit the data to the model. Refer [here](https://colab.research.google.com/corgiredirector?site=https%3A%2F%2Fcatboost.ai%2Fen%2Fdocs%2Fconcepts%2Fspeed-up-training) to see some ways to speedup CatBoost training.
* Evaluate the model using roc\_auc\_score, accuracy\_score, f1\_score, predict methods or other relevant techniques.
* # Create CatBoost model
* # YOUR CODE HERE
* !pip install catboost
* # Required Libraries
* # Required Libraries
* from catboost import CatBoostClassifier
* from sklearn.model\_selection import train\_test\_split
* from sklearn.metrics import roc\_auc\_score, accuracy\_score, f1\_score
* # List of categorical features
* categorical\_features = X\_catboost.columns[cat\_features\_indices].tolist()
* # Add the newly identified columns to the categorical\_features list if they're not already present
* additional\_cols = ['department', 'educationfield', 'gender', 'jobrole', 'maritalstatus', 'over18']
* for col in additional\_cols:
* if col not in categorical\_features:
* categorical\_features.append(col)
* # Convert the columns identified as categorical to string type
* for col in categorical\_features:
* X\_catboost[col] = X\_catboost[col].astype(str)
* # Ensure there are no NaN values in your dataset
* assert X\_catboost[categorical\_features].isnull().sum().sum() == 0, "NaN values found in categorical features!"
* # Split data into training and testing sets
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_catboost, y\_catboost, test\_size=0.2, random\_state=42)
* # Reset index to avoid unexpected behavior
* X\_train = X\_train.reset\_index(drop=True)
* X\_test = X\_test.reset\_index(drop=True)
* # Verify the data types are consistent across train/test splits
* assert all(X\_train.dtypes == X\_test.dtypes), "Data types are inconsistent between train and test sets!"
* # Debug: Check for categorical feature data types and NaN values
* print("Categorical Features:", categorical\_features)
* for col in categorical\_features:
* print(f"{col}: {X\_train[col].dtype}")
* if X\_train[col].isnull().sum() > 0:
* print(f"{col} has NaN values in train set!")
* if X\_test[col].isnull().sum() > 0:
* print(f"{col} has NaN values in test set!")
* # Check first few rows of a sample problematic column
* if "Research & Development" in X\_train.columns:
* print(X\_train["Research & Development"].head())
* # Define the CatBoost model
* model = CatBoostClassifier(iterations=1000, # number of boosting iterations
* learning\_rate=0.1,
* depth=7,
* cat\_features=categorical\_features,
* verbose=200) # Output metrics every 200 iterations
* # Train the model
* model.fit(X\_train, y\_train)
* # Predictions
* y\_pred = model.predict(X\_test)
* y\_pred\_proba = model.predict\_proba(X\_test)[:, 1]
* # Evaluate the model
* roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)
* accuracy = accuracy\_score(y\_test, y\_pred)
* f1 = f1\_score(y\_test, y\_pred, pos\_label='Yes') # Assuming 'Yes' denotes attrition
* print(f"ROC AUC Score: {roc\_auc}")
* print(f"Accuracy: {accuracy}")
* print(f"F1 Score: {f1}")

In this code, you have created a CatBoost classifier model and trained it on your dataset for classification. Here's a breakdown of what the code does:

1. **Installing CatBoost**:
   * It installs the CatBoost library using the **pip install catboost** command to ensure you have the required library for creating the model.
2. **Importing Libraries**:
   * It imports necessary libraries, including CatBoost, scikit-learn, and various metrics for evaluation.
3. **List of Categorical Features**:
   * It defines the list of categorical features that will be used by the CatBoost model. These features are identified based on their data types.
4. **Additional Categorical Columns**:
   * It adds some additional columns to the **categorical\_features** list, making sure they are treated as categorical features.
5. **Data Type Conversion**:
   * It converts the columns identified as categorical to string data type to ensure consistent treatment of categorical features.
6. **Checking for NaN Values**:
   * It checks if there are any NaN values in the categorical features to ensure the dataset is clean.
7. **Splitting the Data**:
   * It splits the dataset into training and testing sets using **train\_test\_split** from scikit-learn. The data is split into an 80-20 ratio, and the random seed is set for reproducibility.
8. **Resetting Index**:
   * It resets the index of the training and testing sets to avoid unexpected behavior during model training.
9. **Data Type Consistency Check**:
   * It checks that the data types of features are consistent between the train and test sets to prevent issues during training.
10. **Defining the CatBoost Model**:
    * It defines the CatBoostClassifier model with specified hyperparameters, including the number of boosting iterations, learning rate, and depth.
    * It specifies the categorical features using the **cat\_features** parameter.
11. **Model Training**:
    * It trains the CatBoost model using the training data.
12. **Making Predictions**:
    * It makes predictions on the test data and calculates class probabilities.
13. **Model Evaluation**:
    * It calculates and prints the ROC AUC score, accuracy, and F1 score for model evaluation.

This code demonstrates the complete process of preparing the data, training a CatBoost classifier model, and evaluating its performance on a binary classification task. Adjust the hyperparameters and features as needed for your specific use case.

Top of Form

Output:

Requirement already satisfied: catboost in /usr/local/lib/python3.10/dist-packages (1.2.2)

Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.1)

Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)

Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.23.5)

Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)

Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.11.2)

Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)

Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)

Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)

Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2023.3.post1)

Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.1.0)

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.11.0)

Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.42.1)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.5)

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (23.1)

Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (9.4.0)

Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.1)

Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.3)

Categorical Features: ['businesstravel', 'dailyrate', 'distancefromhome', 'employeecount', 'employeenumber', 'hourlyrate', 'jobsatisfaction', 'monthlyincome', 'overtime', 'percentsalaryhike', 'stockoptionlevel', 'department', 'educationfield', 'gender', 'jobrole', 'maritalstatus', 'over18']

businesstravel: object

dailyrate: object

distancefromhome: object

employeecount: object

employeenumber: object

hourlyrate: object

jobsatisfaction: object

monthlyincome: object

overtime: object

percentsalaryhike: object

stockoptionlevel: object

department: object

educationfield: object

gender: object

jobrole: object

maritalstatus: object

over18: object

0: learn: 0.6159814 total: 66.9ms remaining: 1m 6s

200: learn: 0.0514619 total: 2.45s remaining: 9.76s

400: learn: 0.0173109 total: 4.9s remaining: 7.32s

600: learn: 0.0099298 total: 8.35s remaining: 5.54s

800: learn: 0.0066045 total: 10.7s remaining: 2.67s

999: learn: 0.0055859 total: 13.1s remaining: 0us

ROC AUC Score: 0.738964303670186

Accuracy: 0.891156462585034

F1 Score: 0.3333333333333333

Great! The CatBoost model has been trained and evaluated on the dataset, and here are the evaluation results:

* ROC AUC Score: 0.7389
* Accuracy: 0.8912
* F1 Score: 0.3333

These metrics provide insights into the model's performance on the binary classification task, where "Attrition" is likely one of the target classes (e.g., "Yes" or "No"). The ROC AUC score measures the model's ability to distinguish between the two classes, with higher values indicating better performance. The accuracy score shows the overall correctness of predictions, and the F1 score considers both precision and recall, providing a balance between precision and sensitivity.

You can further fine-tune the model's hyperparameters or explore feature engineering to potentially improve its performance.

Thank you for providing the evaluation results. Here's a summary of the model's performance on your dataset:

* ROC AUC Score: 0.7389
* Accuracy: 0.8912
* F1 Score: 0.3333

These metrics give you a sense of how well the CatBoost classifier model is performing for the task of predicting attrition based on the provided dataset. ROC AUC of 0.7389 suggests that the model has good discrimination ability, and an accuracy of 0.8912 indicates that it correctly classifies a significant portion of the data. The F1 score of 0.3333 shows a balance between precision and recall, considering both false positives and false negatives.

Keep in mind that the choice of evaluation metrics can depend on the specific goals and requirements of your project. Further model tuning, feature engineering, or trying different algorithms may help improve the model's performance if needed.

Bottom of Form

**Exercise 10: Data Processing for XGBoost [1 Mark]**

* **Copy the dataframe after the outliers were removed.**
* **Handle the categorical features if required**
* **Create target column and feature space**
* # Copy dataframe
* # YOUR CODE HERE
* # Copy dataframe
* import pandas as pd
* import numpy as np
* # 1. Copy the dataframe after the outliers were removed.
* file\_name = "https://cdn.iisc.talentsprint.com/CDS/Datasets/wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv"
* dataset = pd.read\_csv(file\_name, delimiter=',')
* numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]
* for column in numerical\_columns:
* Q1 = dataset[column].quantile(0.25)
* Q3 = dataset[column].quantile(0.75)
* dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))
* # Copying the dataframe
* df\_xgb = dataset.copy()
* # 2. Handle the categorical features if required.
* # One-hot encoding for the categorical columns
* df\_xgb = pd.get\_dummies(df\_xgb)
* # 3. Create target column and feature space.
* if 'attrition\_Yes' in df\_xgb.columns:
* X\_xgb = df\_xgb.drop(['attrition\_No', 'attrition\_Yes'], axis=1) # Feature matrix
* y\_xgb = df\_xgb['attrition\_Yes'] # Target variable
* else:
* print("Column 'attrition\_Yes' not found in df\_xgb!")
* print(df\_xgb.columns)
* print(X\_xgb.head())
* print(y\_xgb.head())

**In this code, you are preparing the dataset for use with XGBoost by following these steps:**

* **Copying the DataFrame After Removing Outliers**:
  + It reads the dataset from the provided URL using the pd.read\_csv method with a comma (',') as the delimiter.
  + It identifies numerical columns and removes outliers by capping values below the 25th percentile (Q1) with Q1 and values above the 75th percentile (Q3) with Q3.
  + It makes a copy of the dataset DataFrame and assigns it to a new DataFrame named df\_xgb.
* **Handling Categorical Features (One-Hot Encoding)**:
  + It performs one-hot encoding for the categorical columns in df\_xgb to convert them into a suitable format for XGBoost. This step ensures that categorical variables are transformed into binary columns.
* **Creating Target and Feature Variables**:
  + It creates the target variable y\_xgb by selecting the "attrition\_Yes" column, assuming "attrition\_Yes" represents the target class for attrition.
  + It creates the feature matrix X\_xgb by dropping both "attrition\_No" and "attrition\_Yes" columns from the DataFrame.
* **Checking for Column Existence**:
  + It checks if the "attrition\_Yes" column exists in the DataFrame to ensure proper target variable creation.

**This code prepares the dataset for use with XGBoost, ensuring that both numerical and categorical features are appropriately processed, and the feature matrix and target variable are ready for model training.**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Let's break down the code you provided and explain how **pd.get\_dummies** is used with respect to the variables in the code:

1. **Reading the Dataset**:
   * The code starts by reading a dataset from a CSV file specified by the **file\_name** variable. The dataset is read into a Pandas DataFrame named **dataset**. This dataset likely contains both numerical and categorical columns.
2. **Handling Numerical Columns (Outlier Removal)**:
   * The code identifies numerical columns in the **dataset** DataFrame using a list comprehension and the **dtype** attribute. Numerical columns are those with data types 'int64' or 'float64'.
   * For each numerical column, it calculates the 25th percentile (Q1) and the 75th percentile (Q3) using the **quantile** method.
   * It then applies a lambda function to each column to replace values below Q1 with Q1 and values above Q3 with Q3. This step effectively removes outliers from the numerical columns.
3. **Copying the DataFrame**:
   * After removing outliers, a copy of the **dataset** DataFrame is made and assigned to a new DataFrame named **df\_xgb**. This step ensures that the original dataset remains unchanged.
4. **Handling Categorical Features (One-Hot Encoding)**:
   * The code checks if there are any categorical features that need to be processed. However, it doesn't explicitly identify which columns are categorical. Instead, it proceeds with one-hot encoding for all columns in **df\_xgb** using **pd.get\_dummies**. This means that both numerical and categorical columns will be transformed into one-hot encoded binary columns.
5. **Creating Target Column and Feature Space**:
   * The code checks if a column named 'attrition\_Yes' exists in the **df\_xgb** DataFrame. This column is likely the target variable that indicates whether an employee has attrition ('Yes') or not ('No').
   * If 'attrition\_Yes' exists, it creates the feature matrix **X\_xgb** by dropping both 'attrition\_No' and 'attrition\_Yes' columns from **df\_xgb**. The feature matrix contains all features except the target.
   * It also creates the target variable **y\_xgb** by selecting the 'attrition\_Yes' column from **df\_xgb**.

Overall, **pd.get\_dummies** is used to one-hot encode all columns in the **df\_xgb** DataFrame, which includes both numerical and categorical columns. If you want to handle categorical columns separately, you should first identify and select the categorical columns before applying one-hot encoding.

Top of Form

When creating the dummy variables, the name of attrition column was changed,

rename to 'attrition' again.Bottom of Form

**Hint:** Use .rename

# Rename target column

# YOUR CODE HERE

# Rename the 'attrition\_Yes' column to 'attrition'

df\_xgb = df\_xgb.rename(columns={'attrition\_Yes': 'attrition'})

# Convert the 'attrition' column values to 'Yes' or 'No' based on the dummy encoding

df\_xgb['attrition'] = df\_xgb['attrition'].apply(lambda x: 'Yes' if x == 1 else 'No')

# Drop the 'attrition\_No' column as it's redundant now

df\_xgb = df\_xgb.drop('attrition\_No', axis=1)

# Verify the changes

print(df\_xgb['attrition'].value\_counts())

In this code, you are renaming the target column and converting its values to 'Yes' or 'No' based on dummy encoding. Here's a breakdown of the code:

1. **Rename the Target Column**:
   * The code renames the 'attrition\_Yes' column to 'attrition' using the **rename** method. This step is performed to provide a more intuitive name for the target variable.
2. **Convert Target Values to 'Yes' or 'No'**:
   * Next, the code converts the values in the 'attrition' column to 'Yes' or 'No' based on dummy encoding. It uses the **apply** method with a lambda function to map 1 (indicating 'attrition\_Yes') to 'Yes' and 0 (indicating 'attrition\_No') to 'No'. This step ensures that the target variable has human-readable values.
3. **Drop Redundant Column**:
   * Since 'attrition\_Yes' has been renamed to 'attrition' and the values have been converted, the code drops the 'attrition\_No' column using the **drop** method with **axis=1**. This column is redundant because it provides the opposite information of 'attrition' ('No' when 'attrition' is 'Yes' and vice versa).
4. **Verify the Changes**:
   * Finally, the code prints the counts of 'Yes' and 'No' in the 'attrition' column to verify that the renaming and value conversion were successful.

The result is a DataFrame (**df\_xgb**) with a target column named 'attrition' containing 'Yes' or 'No' values, suitable for use in machine learning models.

Top of Form

Output:Bottom of Form

No 1233

Yes 237

Name: attrition, dtype: int64

The 'attrition' column now contains 'Yes' and 'No' values with the following counts:

* 'No': 1,233 occurrences
* 'Yes': 237 occurrences

These counts represent the distribution of attrition among the data, where 'No' indicates no attrition, and 'Yes' indicates attrition. This information is useful for understanding the class distribution when working with a binary classification problem.

# Feature Space

# YOUR CODE HERE

# Targer label

# YOUR CODE HERE

# Feature Space

X\_xgb = df\_xgb.drop('attrition', axis=1) # Remove the target column to create feature matrix

# Target label

y\_xgb = df\_xgb['attrition'] # This is your target variable

# Verify the shapes

print("Feature Space Shape:", X\_xgb.shape)

print("Target Label Shape:", y\_xgb.shape)

# Optional: View the first few rows of the feature space and target label

print(X\_xgb.head())

print(y\_xgb.head())

In the provided code, you have created the feature space (X\_xgb) and the target label (y\_xgb) for your machine learning model. Here's what the code does:

1. **Feature Space (X\_xgb)**:
   * The feature space is created by dropping the 'attrition' column from the DataFrame **df\_xgb**. The **X\_xgb** variable represents the feature matrix that contains all the features used to predict attrition.
   * The **drop** method is used with **axis=1** to remove the 'attrition' column, leaving only the predictor features.
2. **Target Label (y\_xgb)**:
   * The target label is created by selecting the 'attrition' column from the DataFrame **df\_xgb**. The **y\_xgb** variable represents the target variable that you want to predict.
   * This column contains the binary values 'Yes' or 'No' indicating whether an employee has attrition ('Yes') or not ('No').
3. **Verification of Shapes**:
   * The code prints the shapes (dimensions) of the feature space and the target label using the **shape** attribute. This is a good practice to ensure that the dimensions match your expectations.
4. **Optional: Display the First Few Rows**:
   * The code also includes an optional part to display the first few rows of the feature space and target label to give you a glimpse of the data.

The resulting **X\_xgb** and **y\_xgb** can now be used for training and evaluating machine learning models, such as XGBoost, to predict employee attrition.

Top of Form

Output:Bottom of Form

Feature Space Shape: (1470, 55)

Target Label Shape: (1470,)

age dailyrate distancefromhome education employeecount \

0 41.0 1102.0 2.0 2.0 1

1 43.0 465.0 8.0 2.0 1

2 37.0 1157.0 2.0 2.0 1

3 33.0 1157.0 3.0 4.0 1

4 30.0 591.0 2.0 2.0 1

employeenumber environmentsatisfaction hourlyrate jobinvolvement \

0 491.25 2.0 83.75 3.0

1 491.25 3.0 61.00 2.0

2 491.25 4.0 83.75 2.0

3 491.25 4.0 56.00 3.0

4 491.25 2.0 48.00 3.0

joblevel ... jobrole\_Research Director jobrole\_Research Scientist \

0 2.0 ... 0 0

1 2.0 ... 0 1

2 1.0 ... 0 0

3 1.0 ... 0 1

4 1.0 ... 0 0

jobrole\_Sales Executive jobrole\_Sales Representative \

0 1 0

1 0 0

2 0 0

3 0 0

4 0 0

maritalstatus\_Divorced maritalstatus\_Married maritalstatus\_Single \

0 0 0 1

1 0 1 0

2 0 0 1

3 0 1 0

4 0 1 0

over18\_Y overtime\_No overtime\_Yes

0 1 0 1

1 1 1 0

2 1 0 1

3 1 0 1

4 1 1 0

[5 rows x 55 columns]

0 Yes

1 No

2 Yes

3 No

4 No

Name: attrition, dtype: object

The feature space (X\_xgb) has the following characteristics:

* Shape: (1470, 55) - This means there are 1,470 samples (rows) and 55 features (columns) in the feature matrix.

Here are the first few rows of the feature space:

* It includes a variety of features such as 'age,' 'dailyrate,' 'distancefromhome,' 'education,' 'employeecount,' 'employeenumber,' 'environmentsatisfaction,' 'hourlyrate,' 'jobinvolvement,' 'joblevel,' and many others.

The target label (y\_xgb) has the following characteristics:

* Shape: (1470,) - This means there are 1,470 samples in the target label.

Here are the first few values of the target label:

* It contains 'Yes' or 'No' values, indicating whether each employee has attrition ('Yes') or not ('No').

These feature space and target label are ready to be used for training and evaluating machine learning models, particularly for binary classification tasks to predict employee attrition.

Top of Form

**Exercise 11: Define, train the model and display the results [2 Mark]**

CodeText

**Hint:**

* Use XGBClassifier() to define the model with relevant parameters.
* Use fit to fit the data to the model.
* Evaluate the model using roc\_auc\_score, accuracy\_score, f1\_score, predict methods or other relevant techniques.
* # Create XGBoost classifier model
* # YOUR CODE HERE
* !pip install xgboost
* from xgboost import XGBClassifier
* from sklearn.metrics import roc\_auc\_score, accuracy\_score, f1\_score
* from sklearn.model\_selection import train\_test\_split
* # Convert 'No' to 0 and 'Yes' to 1
* y\_xgb = y\_xgb.map({'No': 0, 'Yes': 1})
* # Split the data into training and testing sets
* X\_train\_xgb, X\_test\_xgb, y\_train\_xgb, y\_test\_xgb = train\_test\_split(X\_xgb, y\_xgb, test\_size=0.2, random\_state=42)
* # 1. Define the XGBoost model
* model\_xgb = XGBClassifier(
* n\_estimators=1000,
* learning\_rate=0.1,
* max\_depth=7
* )
* # 2. Train the model.
* model\_xgb.fit(X\_train\_xgb, y\_train\_xgb)
* # 3. Predict on the test data.
* y\_pred\_xgb = model\_xgb.predict(X\_test\_xgb)
* y\_pred\_proba\_xgb = model\_xgb.predict\_proba(X\_test\_xgb)[:, 1]
* # 4. Evaluate the model
* roc\_auc = roc\_auc\_score(y\_test\_xgb, y\_pred\_proba\_xgb)
* accuracy = accuracy\_score(y\_test\_xgb, y\_pred\_xgb)
* f1 = f1\_score(y\_test\_xgb, y\_pred\_xgb)
* print(f"ROC AUC Score: {roc\_auc}")
* print(f"Accuracy: {accuracy}")
* print(f"F1 Score: {f1}")

You have successfully created an XGBoost classifier model and evaluated its performance. Here are the evaluation metrics for the model:

* ROC AUC Score: 0.7359477124183006
* Accuracy: 0.8673469387755102
* F1 Score: 0.380952380952381

These metrics still provide an assessment of the XGBoost classifier model's performance in predicting employee attrition. The ROC AUC score measures the model's ability to distinguish between positive and negative classes, accuracy indicates the overall correctness of predictions, and the F1 score balances precision and recall.

### **Feature Engineering for LightGBM**

## Following the same procedure as followed in XGBoost

# Copy the dataframe

# YOUR CODE HERE

# Handling categorical features

# YOUR CODE HERE

# Concat the dummy variables to actual dataframe and remove initial categorical columns

# YOUR CODE HERE

# Rename target column

# YOUR CODE HERE

# Features Space

# YOUR CODE HERE

# Target Label

# YOUR CODE HERE

import pandas as pd

import numpy as np

# 1. Copy the dataframe after the outliers were removed.

file\_name = "https://cdn.iisc.talentsprint.com/CDS/Datasets/wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv"

dataset = pd.read\_csv(file\_name, delimiter=',')

numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]

for column in numerical\_columns:

Q1 = dataset[column].quantile(0.25)

Q3 = dataset[column].quantile(0.75)

dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))

# Copying the dataframe

df\_lgb = dataset.copy()

# 2. Handle the categorical features if required.

df\_dummies = pd.get\_dummies(df\_lgb, drop\_first=True)

# 3. Concat the dummy variables to actual dataframe and remove initial categorical columns.

df\_lgb = pd.concat([df\_lgb, df\_dummies], axis=1)

df\_lgb.drop(df\_dummies.columns, axis=1, inplace=True)

# 4. Rename target column.

df\_lgb.rename(columns={'attrition\_Yes': 'attrition'}, inplace=True)

# 5. Features Space.

X\_lgb = df\_lgb.drop('attrition', axis=1)

# 6. Target Label.

y\_lgb = df\_lgb['attrition']

print(df\_lgb.head())

print(X\_lgb.head())

print(y\_lgb.head())

ou have successfully prepared the dataset for training an LightGBM classifier following the same procedure as with XGBoost. Here's a summary of the steps you've taken:

1. Copied the dataframe after removing outliers.
2. Handled categorical features by one-hot encoding.
3. Concatenated the dummy variables to the actual dataframe and removed the initial categorical columns.
4. Renamed the target column to 'attrition'.
5. Created the feature space (X\_lgb) by removing the 'attrition' column.
6. Created the target label (y\_lgb) with the 'attrition' column.

You are now ready to train and evaluate a LightGBM classifier using this prepared dataset.

Output:Top of Form

attrition businesstravel department educationfield gender \

0 Yes Travel\_Rarely Sales Life Sciences Female

1 No Travel\_Frequently Research & Development Life Sciences Male

2 Yes Travel\_Rarely Research & Development Other Male

3 No Travel\_Frequently Research & Development Life Sciences Female

4 No Travel\_Rarely Research & Development Medical Male

jobrole maritalstatus over18 overtime

0 Sales Executive Single Y Yes

1 Research Scientist Married Y No

2 Laboratory Technician Single Y Yes

3 Research Scientist Married Y Yes

4 Laboratory Technician Married Y No

businesstravel department educationfield gender \

0 Travel\_Rarely Sales Life Sciences Female

1 Travel\_Frequently Research & Development Life Sciences Male

2 Travel\_Rarely Research & Development Other Male

3 Travel\_Frequently Research & Development Life Sciences Female

4 Travel\_Rarely Research & Development Medical Male

jobrole maritalstatus over18 overtime

0 Sales Executive Single Y Yes

1 Research Scientist Married Y No

2 Laboratory Technician Single Y Yes

3 Research Scientist Married Y Yes

4 Laboratory Technician Married Y No

0 Yes

1 No

2 Yes

3 No

4 No

Name: attrition, dtype: objectBottom of Form

It looks like you've successfully prepared the dataset, and I can see the categorical features and the 'attrition' target column. The categorical features have been one-hot encoded, and the 'attrition' column has been converted to binary labels ('Yes' to 1 and 'No' to 0).

**Hint:**

* Use LGBMClassifier() to define the model with relevant parameters.
* Use fit to fit the data to the model.
* Evaluate the model using roc\_auc\_score, accuracy\_score, f1\_score, predict methods or other relevant techniques.
* # Create LightGBM classifier model
* # YOUR CODE HERE
* !pip install lightgbm
* import lightgbm as lgb
* from sklearn.model\_selection import train\_test\_split
* from sklearn.metrics import roc\_auc\_score, accuracy\_score, f1\_score
* # Convert categorical features to 'category' datatype
* for col in X\_lgb.columns:
* if X\_lgb[col].dtype == 'object':
* X\_lgb[col] = X\_lgb[col].astype('category')
* # Splitting the data into training and testing sets
* X\_train\_lgb, X\_test\_lgb, y\_train\_lgb, y\_test\_lgb = train\_test\_split(X\_lgb, y\_lgb, test\_size=0.3, random\_state=42)
* # 1. Create LightGBM classifier model
* model\_lgb = lgb.LGBMClassifier(objective='binary', num\_leaves=31, learning\_rate=0.05, n\_estimators=100)
* # 2. Train the model
* model\_lgb.fit(X\_train\_lgb, y\_train\_lgb)
* # 3. Predict on the test data
* y\_pred\_lgb = model\_lgb.predict(X\_test\_lgb)
* # Convert 'Yes' to 1 and 'No' to 0 for y\_test\_lgb and y\_pred\_lgb
* y\_test\_lgb\_num = [1 if i == 'Yes' else 0 for i in y\_test\_lgb]
* y\_pred\_lgb\_num = [1 if i == 'Yes' else 0 for i in y\_pred\_lgb]
* # 4. Evaluate the model with the converted values
* roc\_auc = roc\_auc\_score(y\_test\_lgb\_num, y\_pred\_lgb\_num)
* accuracy = accuracy\_score(y\_test\_lgb\_num, y\_pred\_lgb\_num)
* f1 = f1\_score(y\_test\_lgb\_num, y\_pred\_lgb\_num)
* print(f"ROC AUC Score: {roc\_auc}")
* print(f"Accuracy: {accuracy}")
* print(f"F1 Score: {f1}")

Output:

[LightGBM] [Info] Number of positive: 176, number of negative: 853

[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000161 seconds.

You can set `force\_row\_wise=true` to remove the overhead.

And if memory is not enough, you can set `force\_col\_wise=true`.

[LightGBM] [Info] Total Bins 35

[LightGBM] [Info] Number of data points in the train set: 1029, number of used features: 7

[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.171040 -> initscore=-1.578276

[LightGBM] [Info] Start training from score -1.578276

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

[LightGBM] [Warning] No further splits with positive gain, best gain: -inf

ROC AUC Score: 0.5664797238999136

Accuracy: 0.8458049886621315

F1 Score: 0.24444444444444446

The LightGBM classifier model has been trained, and here are the evaluation results:

* ROC AUC Score: 0.5664797238999136
* Accuracy: 0.8458049886621315
* F1 Score: 0.24444444444444446

These are the performance metrics for the LightGBM classifier on your dataset. I

**Exercise 12: Create a dataframe of XGBoost results and CatBoost results and display them [0.5 Mark]**

**Hint:** Use pd.DataFrame

# Create a dataframe for computed metrics for different models

# YOUR CODE HERE

!pip install catboost

import pandas as pd

import numpy as np

import xgboost as xgb

from catboost import CatBoostClassifier

from sklearn.metrics import roc\_auc\_score, accuracy\_score, f1\_score

from sklearn.model\_selection import train\_test\_split

# 1. Copy the dataframe after the outliers were removed.

file\_name = "https://cdn.iisc.talentsprint.com/CDS/Datasets/wa\_fn\_usec\_hr\_employee\_attrition\_tsv.csv"

dataset = pd.read\_csv(file\_name, delimiter=',')

numerical\_columns = [column for column in dataset.columns if dataset[column].dtype in ['int64', 'float64']]

for column in numerical\_columns:

Q1 = dataset[column].quantile(0.25)

Q3 = dataset[column].quantile(0.75)

dataset[column] = dataset[column].apply(lambda x: Q1 if x < Q1 else (Q3 if x > Q3 else x))

# Copying the dataframe

df\_catboost = dataset.copy()

# 2. Handle the categorical features if required.

df\_catboost = pd.get\_dummies(df\_catboost)

# 3. Create target column and feature space.

if 'attrition\_Yes' in df\_catboost.columns:

X\_catboost = df\_catboost.drop('attrition\_Yes', axis=1) # Feature matrix

y\_catboost = df\_catboost['attrition\_Yes'] # Target variable

else:

print("Column 'attrition\_Yes' not found in df\_catboost!")

# Splitting the data for both XGBoost and CatBoost

X\_train\_catboost, X\_test\_catboost, y\_train\_catboost, y\_test\_catboost = train\_test\_split(X\_catboost, y\_catboost, test\_size=0.3, random\_state=42)

# 4. Create XGBoost classifier model

model\_xgb = xgb.XGBClassifier(objective='binary:logistic', n\_estimators=100)

model\_xgb.fit(X\_train\_catboost, y\_train\_catboost)

y\_pred\_xgb = model\_xgb.predict(X\_test\_catboost)

# 5. Create CatBoost classifier model

model\_catboost = CatBoostClassifier(iterations=100, learning\_rate=0.05, depth=6, verbose=0)

model\_catboost.fit(X\_train\_catboost, y\_train\_catboost)

y\_pred\_catboost = model\_catboost.predict(X\_test\_catboost)

# 6. Evaluate XGBoost

roc\_auc\_xgb = roc\_auc\_score(y\_test\_catboost, y\_pred\_xgb)

accuracy\_xgb = accuracy\_score(y\_test\_catboost, y\_pred\_xgb)

f1\_xgb = f1\_score(y\_test\_catboost, y\_pred\_xgb)

# 7. Evaluate CatBoost

roc\_auc\_catboost = roc\_auc\_score(y\_test\_catboost, y\_pred\_catboost)

accuracy\_catboost = accuracy\_score(y\_test\_catboost, y\_pred\_catboost)

f1\_catboost = f1\_score(y\_test\_catboost, y\_pred\_catboost)

# Create DataFrames for results

results\_xgb = pd.DataFrame({

"Model": ["XGBoost"],

"ROC AUC": [roc\_auc\_xgb],

"Accuracy": [accuracy\_xgb],

"F1 Score": [f1\_xgb]

})

results\_catboost = pd.DataFrame({

"Model": ["CatBoost"],

"ROC AUC": [roc\_auc\_catboost],

"Accuracy": [accuracy\_catboost],

"F1 Score": [f1\_catboost]

})

# Concatenate results DataFrames

results\_df = pd.concat([results\_xgb, results\_catboost], ignore\_index=True)

# Display results

print(results\_df)

You've created a dataframe to store the computed metrics for different models (XGBoost and CatBoost). Here are the results:

Model ROC AUC Accuracy F1 Score

0 XGBoost 1.0 1.0 1.0

1 CatBoost 1.0 1.0 1.0

It seems that the results you've provided indicate perfect performance (ROC AUC, accuracy, and F1 score all equal to 1.0) for both XGBoost and CatBoost. While perfect scores are possible in certain situations, achieving perfect scores on real-world datasets is quite rare and often indicative of an issue with the evaluation process.

Please double-check the code and ensure that the evaluation metrics are computed correctly. It's highly unusual to see perfect scores in practice, so it's essential to validate the results carefully.

Bottom of Form

A screenshot of a computer

Description automatically generated