**Data Preprocessing and Model Implementation Documentation**

* **Importing the Dataset from Drive:**

To work with files stored in Google Drive, the google.colab module mounts Google Drive, providing access to files directly from a Google Colab notebook.

* **Reading the Dataset into DataFrames:**

Pandas is utilized to read Excel files into DataFrames. This step involves reading two datasets from Excel files into DataFrame objects(df1,df2), which will be combined and processed later.

* **Merging the Common Columns from Both Datasets:**

The data frames (df1 and df2) contain differently named columns and alignment, so, merged them by defining common\_cols and using the Pandas functions, rename() and reindex() to correctly align and merge the columns into a combined\_df.

* **Removing Unnecessary Columns:**

This step involves cleaning the data by removing columns that are not relevant to the analysis. This helps in reducing the complexity of the dataset, focusing the analysis on the important features and improving the performance.

* **Plotting the Distribution of Target Labels Using Matplotlib:**

Visualizing the distribution of target labels is an essential step in understanding the class imbalance in the dataset. Matplotlib is used to create histograms, providing insights into the frequency of each class. Here, the IT\_Support has the highest number of instances.

* **Downloading the NLTK Stopwords:**

NLTK's stopwords are downloaded to filter out common words that do not contribute meaningful information to the text analysis. Stopwords are words like "the", "is", and "in" that appear frequently but do not carry significant meaning. Removing them helps in reducing noise in the data.

* **Initializing the WordNetLemmatizer and Defining the Process Functions:**

Lemmatization is the process of converting words to their base form. Initializing the WordNetLemmatizer and defining preprocessing functions involves converting text to lowercase and lemmatizing words. This step standardizes the text data, ensuring consistency and reducing the dimensionality of the dataset by treating different forms of a word as a single term.

* **Creating a New Column 'text' to Combine 'title' and 'description' Columns:**

Combining multiple text columns into a single column simplifies the processing by consolidating all relevant text into one place. This combined text is used for feature extraction, ensuring that all available textual information is utilized.

**TF-IDF Vectorization**

* TF-IDF (Term Frequency-Inverse Document Frequency) vectorization is a technique used to convert textual data into numerical features so that the model can process textual data.
* It works by assigning a weight to each word in a document based on its frequency in that document (term frequency) and how unique or rare it is across the entire dataset (inverse document frequency).
* This helps in highlighting important words that are not too common across all documents but are significant in individual documents.
* The process involves first calculating the term frequency of each word in a document, then adjusting these frequencies by the inverse document frequency to down-weight common words and up-weight rare ones. The resulting TF-IDF vectors represent the importance of each word in the context of the entire dataset.

**Synthetic Minority Over-sampling Technique (SMOTE)**

* SMOTE (Synthetic Minority Over-sampling Technique) is an over-sampling method used to address class imbalance in datasets. It generates synthetic samples for the minority class rather than duplicating existing samples. This helps make the classifier more robust and balanced by providing a more even distribution of class labels.
* For each instance in the minority class, SMOTE identifies its k nearest neighbors from the same class. This is typically done using Euclidean distance in feature space.
* One of these k nearest neighbors is randomly selected.
* A synthetic sample is created by selecting a point along the line segment joining the minority class instance and its chosen neighbor. This is done by choosing a random point between the two instances.
* Synthetic Sample=Minority Instance+Random Value×(Neighbor−Minority Instance)
* This random value typically lies between 0 and 1.
* This process is repeated until the desired level of balancing is achieved, i.e., until the minority class reaches the specified number of samples.

The minority target classes HR, PMO, and Finance are upsampled, to 4000 instances each, using the SMOTE technique after the TF-IDF vectorization to achieve the balance between the classes.

Also, the Majority class, IT\_Support is reduced to 4000 instances using the RandomDownsampler.

This balances the dataset more evenly.

**In the workflow, TF-IDF vectorization is applied before SMOTE to transform the text data into numerical form, which is then used for balancing the dataset with SMOTE and training the machine learning model.  
  
Logistic Regression**A logistic regression model is trained and used to make predictions on a test set.

The max\_iter parameter specifies the maximum number of iterations the solver will take to converge. The max\_iter is set to a value of 1000, instead of the default value of 100, to ensure the model has enough opportunities to converge.

**BERT contextual embeddings instead of TF-IDF vectorization.**The training curve for the SMOTE + TF-IDF + Logistic Regression model exhibited an undesirable trend of increasing rather than decreasing, indicating the model was not learning effectively from the data.

* To address this, a transition was made to leverage BERT contextual embeddings. Unlike TF-IDF, which represents words based on their frequency and inverse document frequency, BERT contextual embeddings capture the meaning of words within the context of the entire sentence. This is achieved through BERT's deep bidirectional training on large-scale corpora, enabling it to understand and encode semantic relationships and contexts effectively.
* A BERT tokenizer is initialized along with the model pre-trained on a large corpus (in this case, 'bert-base-uncased').
* The get\_bert\_embeddings function processes text inputs by tokenizing them into BERT-compatible format (padding and truncating to a maximum length), obtaining embeddings from the BERT model's last hidden state, and averaging them across tokens to produce contextual embeddings for the entire text.

**After using the contextual embeddings and training Logistic regression model, an accuracy of 94% was achieved.**