**Methodology:**

* 1. **Data Pre-processing**

**3.1.1. Exploring column ‘Category’**

We used the .value\_counts() function whose task is to compute a tabular representation by counting the occurrences of unique values within the specified column ‘Category’.

**3.1.2. Label Encoding**

We have used the 'LabelEncoder' class from the scikit-learn library which is a tool commonly used for encoding categorical data into numerical format suitable for machine learning models. An instance of the LabelEncoder class is created and assigned to the variable 'encoder'.

This object will be used to transform categorical columns in the DataFrame. Then using the ‘fit\_transform’ function we have converted the categorical values from both the columns to numerical columns. This function first fits the data in required parameters and then converts (transforms) it into its modified version.

**3.1.3. Data Visualization**

We have also generated a vertical bar plot showing the distribution of categories in the 'Category' column of your DataFrame. Each bar represents the count of occurrences of a specific category, and the x-axis labels are rotated for better visibility. It is helpful for understanding the distribution and frequency of different categories in your dataset.

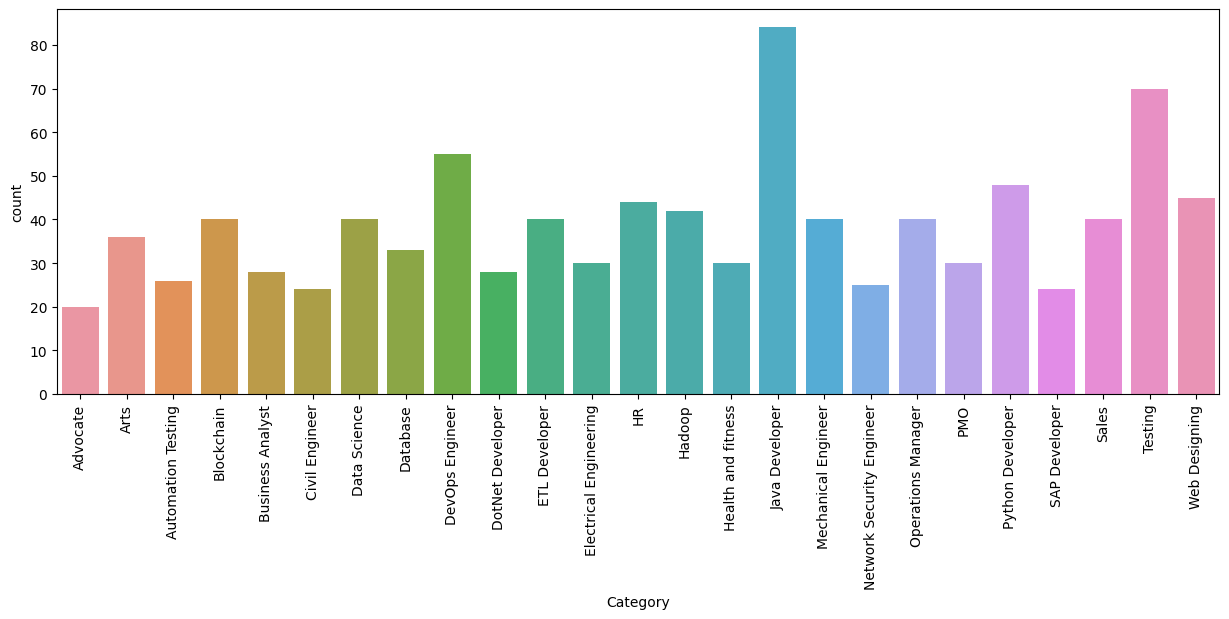


Fig 3.1: Bar graph showing count vs. Category.

Here, we created a visually appealing pie chart representing the distribution of categories in the 'Category' column of your DataFrame (df). Each wedge in the pie corresponds to a unique category, and the size of each wedge is proportional to the count of that category.

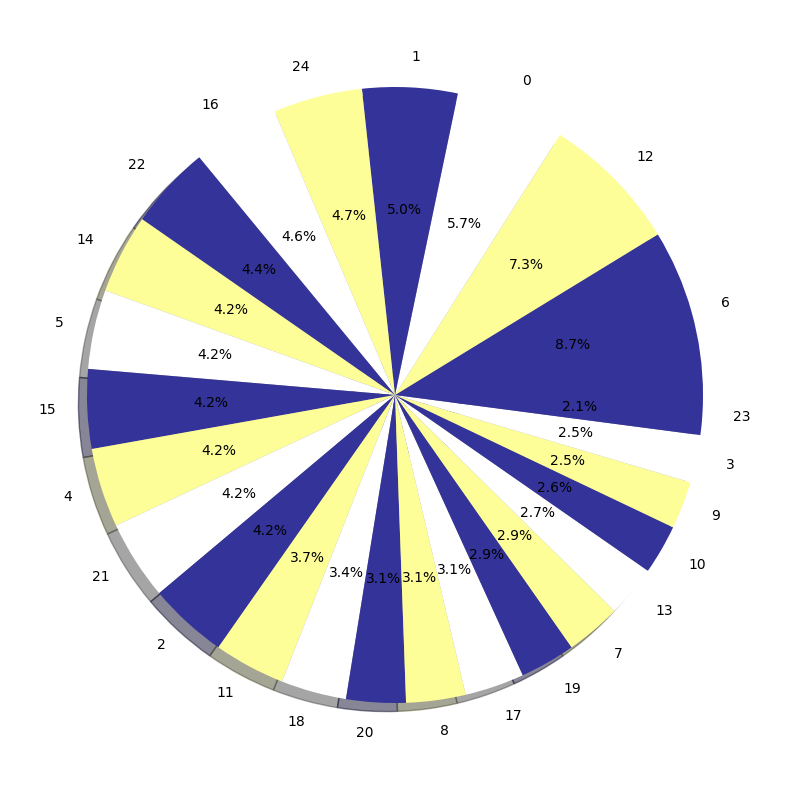


Fig 3.2: Pie chart of the encoded column ‘Category’.

**3.1.4. Data Cleaning**

We have defined a Python function called cleanResume that takes a text input (txt) and performs several cleaning operations on it. It removes any URLs, RT/cc, hashtags, mentions, non-ASCII characters, multiple spaces within single spaces, and special characters and then returns a cleaned text. We have also replaced the word “Resume” with “ “ spaces to produce a more clean text.

* 1. **Feature Engineering**

**3.2.1. Vectorization**

We have used the TfidfVectorizer from scikit-learn to convert a collection of text data (in the 'Resume' column of your DataFrame df) into a numerical format known as TF-IDF (Term Frequency-Inverse Document Frequency).

TF-IDF (Term Frequency-Inverse Document Frequency)- It is a statistical measure used in natural language processing to evaluate the importance of a word within a collection of documents (a corpus). It considers both the frequency of the word within a single document and its rarity across the entire corpus.

**‘tfidf.fit(df['Resume'])’** is used to 'fit' the TfidfVectorizer to the 'Resume' data in the DataFrame. It analyzes the text to identify unique words and their frequencies across all the resumes.

**‘tfidf.transform(df['Resume'])’** is used to convert the text data into numerical vectors where each vector represents a document (resume) and the values in the vectors correspond to the TF-IDF weights of words in each document.

* 1. **Train Test Split**

In this we have taken our feature data (requiredText) and target data 'Category', and split them into two sets: one for training your machine learning model (X\_train, y\_train) and one for testing the model's performance (X\_test, y\_test). The split is done with 80% of the data used for training and 20% for testing, and the random state ensures reproducibility.

* 1. **Machine Learning Techniques**

**3.4.1. OneVsRestClassifier approach**

Implementing the One-vs-Rest (OvR) classifier approach with base classifiers is valuable in multiclass classification due to its simplicity, interpretability, and flexibility in choosing diverse base classifiers such as Decision Trees, KNearest Neighbours or Random Forests.

The OvR strategy provides an effective means to extend binary classifiers to handle multiple classes, allows for ensemble learning, handles class imbalances, and supports parallelization for scalability and faster training. Moreover, it accommodates incremental learning and probabilistic outputs, striking a practical balance between model interpretability and predictive performance.

**3.4.2. KNearest Neighbour Classifier**

We set up a K-Nearest Neighbors classifier using the One-vs-Rest approach for multiclass classification. It trains the classifier, makes predictions on a test set, and evaluates the accuracy of the model in predicting the correct classes.

Employing a One-vs-Rest (OvR) classifier with K-Nearest Neighbors (KNN) as the base classifier offers advantages for multiclass classification. KNN inherently supports multiclass tasks, and the OvR strategy extends this capability, providing simplicity and ease of implementation. KNN is effective in capturing complex, non-linear relationships and does not assume a specific shape of decision boundaries. Additionally, its ability to handle imbalanced class distributions and adapt to varying class sizes, along with the straightforward interpretability of individual KNN models, makes it a suitable choice for scenarios where transparency in classification decisions is valued.

**3.4.3. Decision Tree Classifier**

We have used the One-vs-Rest classifier using the Decision Tree as a base classifier. This means it will train multiple binary classifiers, each focusing on distinguishing one class from the rest.

Using a Decision Tree as the base classifier in a One-vs-Rest (OvR) strategy is advantageous due to its inherent ability to handle multiclass problems, simplicity, and interpretability. Decision Trees are effective in capturing non-linear relationships, providing insight into feature importance, and accommodating mixed data types without extensive preprocessing. Their compatibility with ensemble methods, robustness to irrelevant features, ability to handle imbalanced data, and ease of implementation make them a versatile choice for OvR classification tasks. Additionally, the transparent nature of Decision Trees is valuable when interpretability is a priority, allowing for an intuitive understanding of the model's decision-making process.

**3.4.4. Random Forest Classifier**

We have trained a Random Forest model using a One-vs-Rest approach for multi-class classification, make predictions on a test set and evaluate the accuracy of the model. Random Forest's ensemble nature can lead to improved performance over a single Decision Tree. The accuracy score indicates how well the model performs in correctly predicting the classes.

Utilizing the One-vs-Rest classifier with a Random Forest as the base classifier offers several advantages, including inherent support for multiclass classification, the ensemble's ability to improve generalization and reduce overfitting, the capacity to capture complex and non-linear relationships, provision of feature importance insights, robustness to noisy data and imbalanced class distributions, and ease of implementation with parallelization benefits. Random Forest's ensemble nature, coupled with its generalization across diverse datasets, makes it a powerful choice for multiclass classification tasks, striking a balance between predictive performance and interpretability trade-offs.

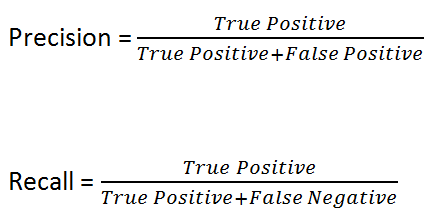
**3.4.5. Pickle Model**

It refers to a Python module that is used for serializing and deserializing Python objects. In machine learning, it is often employed to save trained models to a file, allowing them to be easily reloaded for later use. Serialization is the process of converting a Python object into a byte stream. This byte stream can be saved to a file or sent over a network. In the context of machine learning, this Python object is typically a trained model.

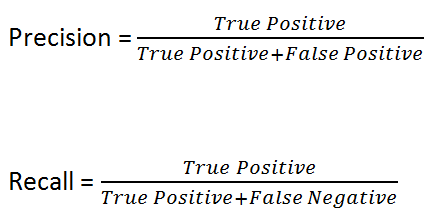
**3.5 CV and performance measurement, comparative analysis**

We have provided a reusable function to print precision, recall, and F1 score for assessing the performance of all classifiers, applied to all classifiers ie. K-Nearest Neighbors, Decision Tree and Random Forest. Additionally, it outputs the confusion matrix, offering a detailed breakdown of classification results.

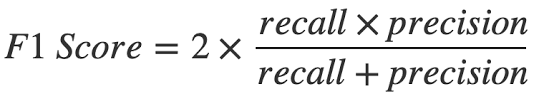
**Precision-** It is a metric used in classification to measure the accuracy of positive predictions made by a model. It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model, including both correct and incorrect positive predictions. The formula for precision is:



**Recall-** Is known as Sensitivity or True Positive Rate, is a metric used in classification to measure the ability of a model to correctly identify all relevant instances of a particular class. It is defined as the ratio of true positive predictions to the total number of actual positive instances. The formula for recall is:



**F1 score**- It is a metric used to assess the balance between precision and recall in a binary or multiclass classification problem. It is particularly useful when there is an uneven class distribution. The formula for the F1 score is:



**3.6 Analysis of bias and variance**

To calculate bias-variance, we have used the bias-variance decomposition function from the mlxtend library. It decomposes the mean squared error (mse) of the model into bias and variance components.

For the Decision Tree model, Random forest model and K Nearest Neighbour model, the decomposition is calculated using training and testing data, and the results are printed.

The components are:

**Mean squared error-** It is a metric used to measure the average squared difference between the predicted values and the actual values in a regression problem. It quantifies the overall accuracy of a model's predictions by calculating the average of the squared differences between predicted and actual values for each data point. Lower the MSE higher the accuracy of the entire model.

**Bias component-** Represents the error introduced by approximating a real-world problem with a simplified model. High bias may indicate that the model is not complex enough to capture the underlying patterns in the data.

**Variance component-** Represents the error introduced by the model's sensitivity to variations in the training data. A high variance may indicate overfitting, where the model is too complex and captures noise instead of general patterns in the data.

Results and Discussion:

Following is the result after using all the classifiers on the dataset-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Classifiers | Accuracy | Precision | Recall | F1-Score |
| KNeighborsClassifier | 0.984455 | 0.9874 | 0.9845 | 0.9839 |
| RandomForestClassifier | 0.994818 | 0.9955 | 0.9948 | 0.9949 |
| DecisionTreeClassifier | 0.994818 | 0.9957 | 0.9948 | 0.9950 |

Table 4.1: Result of all classifiers on the dataset

Following is the result after performing cross-validation-

|  |  |
| --- | --- |
| Classifiers | Cross Validation Accuracy |
| KNeighborsClassifier | 0.978162 |
| RandomForestClassifier | 0.994802 |
| DecisionTreeClassifier | 0.991688 |

Table 4.2: Result of all classifiers after performing cross-validation

Following is the result after performing bias-variance decomposition-

|  |  |  |  |
| --- | --- | --- | --- |
| Classifiers | 0-1 loss | Bias | Variance |
| KNeighborsClassifier | 0.0467 | 0.02515 | 0.03251 |
| RandomForestClassifier | 0.01738 | 0.003144 | 0.01545 |
| DecisionTreeClassifier | 0.0207 | 0.003144 | 0.02062 |

Table 4.3: Result after performing bias-variance decomposition