Importing Libraries:

numpy: A library for numerical operations in Python.

pandas: A library for data manipulation and analysis, particularly for working with tabular data.

matplotlib.pyplot: A plotting library for creating visualizations in Python.

warnings: A module to control the display of warning messages in Python.

scikit-learn (sklearn): A machine learning library in Python.

Suppressing Warnings: The warnings.filterwarnings('ignore') line is used to suppress warning messages that might be generated during the execution of the code.

Adding a New Column: The code adds a new column called 'cleaned_resume' to the DataFrame resumeDataSet. This column will be used to store the processed or cleaned resume text.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
#from sklearn.naive_bayes import MultinomiaINB
#from sklearn.multiclass import oneVsRestClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from pandas.plotting import scatter_matrix
from sklearn import metrics

resumeDataSet = pd.read_csv('/content/sample_data/UpdatedResumeDataSet.csv', encoding = 'utf-8')
resumeDataSet['cleaned_resume'] = ''
resumeDataSet.head()
```

	Category	Resume	cleaned_resume
0	Data Science	Skills * Programming Languages: Python (pandas...	
1	Data Science	Education Details \r\nMay 2013 to May 2017 B.E...	
2	Data Science	Areas of Interest Deep Learning, Control Syste...	
3	Data Science	Skills â□¢ R â□¢ Python â□¢ SAP HANA â□¢ Table...	
4	Data Science	Education Details \r\n MCA YMCAUST, Faridab...	

Printing the distinct categories of resumes present in the **'Category'** column of the resumeDataSet DataFrame.

It uses the **.unique()** method to retrieve the unique values in the 'Category' column and then prints them to the console. Each unique value represents a different category of resume.

Here's what the code does step by step:

resumeDataSet['Category']: This extracts the 'Category' column from the resumeDataSet DataFrame, which contains the categories of the resumes.

.unique(): This method is used on the 'Category' column to retrieve an array of unique values (distinct categories) in that column.

```
print ("Displaying the distinct categories of resume")
print(resumeDataSet['Category'].unique())
```

```
Displaying the distinct categories of resume
['Data Science' 'HR' 'Advocate' 'Arts' 'Web Designing'
 'Mechanical Engineer' 'Sales' 'Health and fitness' 'Civil Engineer'
 'Java Developer' 'Business Analyst' 'SAP Developer' 'Automation Testing'
 'Electrical Engineering' 'Operations Manager' 'Python Developer'
 'DevOps Engineer' 'Network Security Engineer' 'PMO' 'Database' 'Hadoop'
 'ETL Developer' 'DotNet Developer' 'Blockchain' 'Testing']
```

The distinct categories of resume and the number of records belonging to each category:

resumeDataSet['Category']: This extracts the 'Category' column from the resumeDataSet DataFrame, which contains the categories of the resumes.

.value_counts(): This method is used on the 'Category' column to count the occurrences of each unique value (category) in that column.

```
print("Displaying the distinct categories of resume and the number of records belonging to each catgeories")
print(resumeDataSet['Category'].value_counts())
```

```
Displaying the distinct categories of resume and the number of records belonging to each catgeories
Java Developer               84
Testing                      70
DevOps Engineer              55
Python Developer             48
Web Designing                45
HR                           44
Hadoop                       42
Blockchain                   40
ETL Developer                40
Operations Manager           40
Data Science                 40
Sales                        40
Mechanical Engineer          40
Arts                         36
Database                     33
Electrical Engineering       30
Health and fitness           30
PMO                          30
Business Analyst             28
DotNet Developer             28
Automation Testing           26
Network Security Engineer    25
SAP Developer                24
Civil Engineer               24
Advocate                     20
Name: Category, dtype: int64
```

import seaborn as sns: This line imports the seaborn library, which provides high-level interface functions for creating attractive and informative statistical graphics.

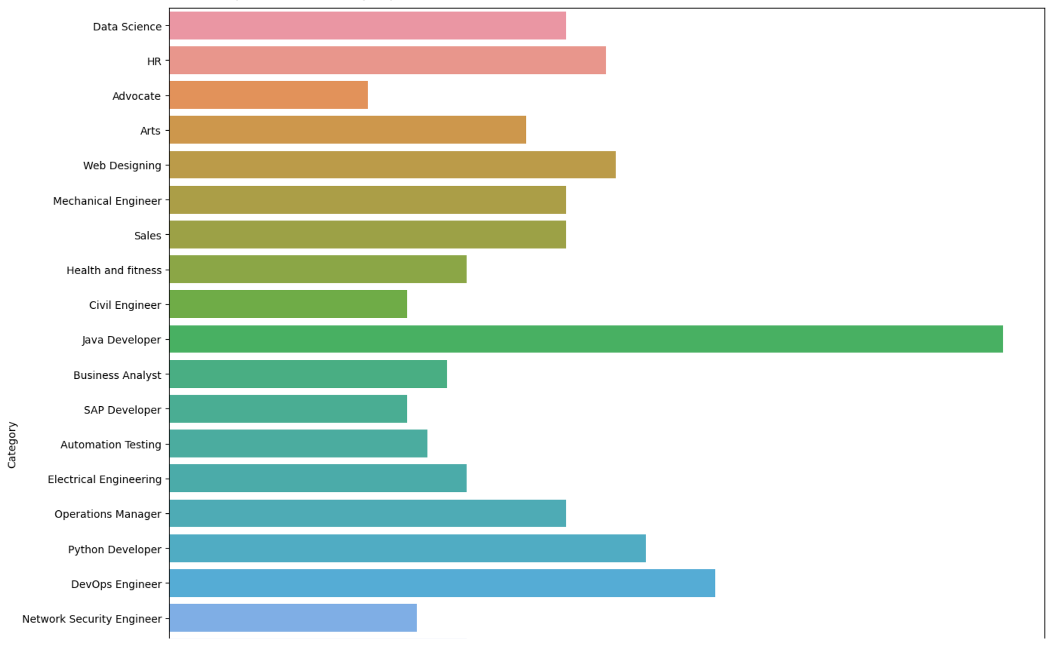
plt.figure(figsize=(15, 15)): This sets the figure size for the plot, making it 15x15 inches.

plt.xticks(rotation=90): This rotates the x-axis labels by 90 degrees to make them more readable when dealing with long category names.

sns.countplot(y="Category", data=resumeDataSet): This line creates a countplot using seaborn's countplot function. It plots the number of occurrences of each category from the 'Category' column of the resumeDataSet DataFrame on the y-axis.

```
import seaborn as sns
plt.figure(figsize=(15,15))
plt.xticks(rotation=90)
sns.countplot(y="Category", data=resumeDataSet)
```

```
<Axes: xlabel='count', ylabel='Category'>
```



from matplotlib.gridspec import GridSpec: This imports the GridSpec module from matplotlib, which allows for the creation of custom grid layouts for plots.

plt.figure(1, figsize=(25, 25)): This creates a new figure with a size of 25x25 inches to accommodate the pie chart.

the_grid = GridSpec(2, 2): This sets up a 2x2 grid layout for placing the pie chart within the figure.

cmap = plt.get_cmap('coolwarm'): This sets the colormap 'coolwarm' for coloring the pie chart wedges.

colors = [cmap(i) for i in np.linspace(0, 1, 3)]: This generates a list of three colors from the 'coolwarm' colormap that will be used to color the pie chart wedges.

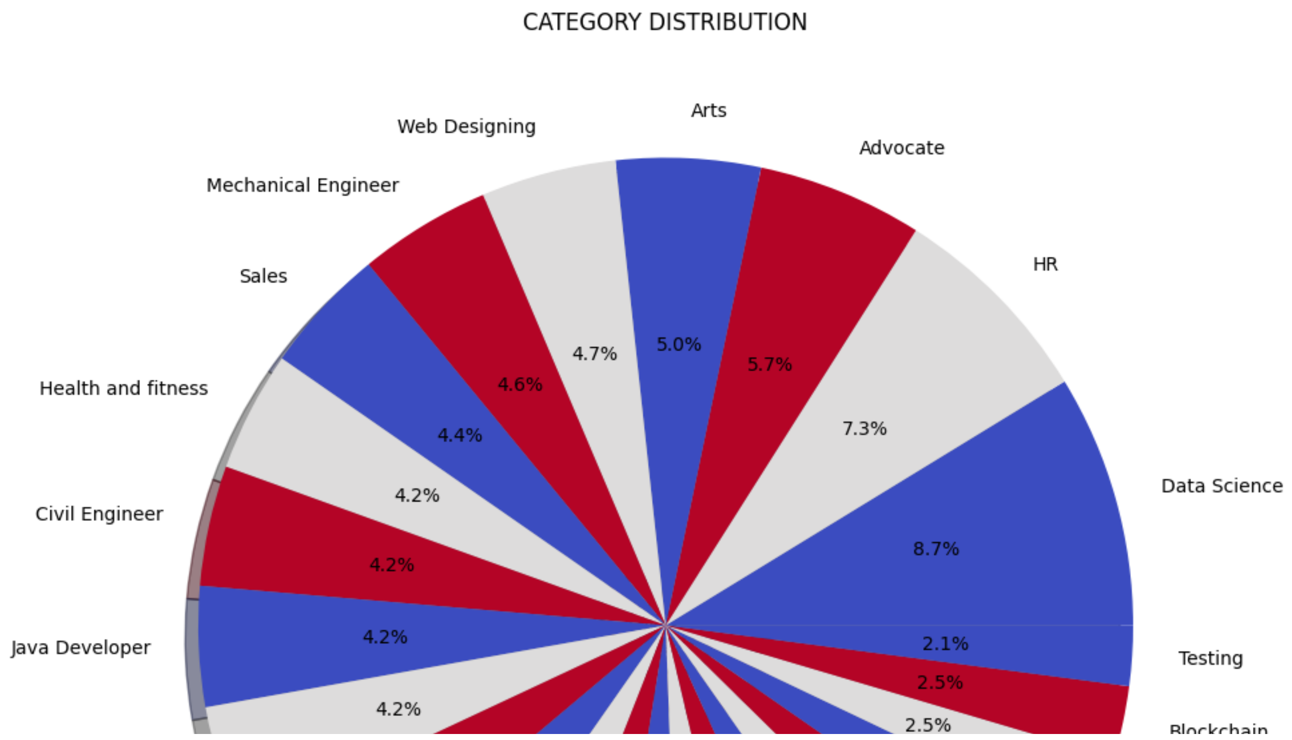
plt.subplot(the_grid[0, 1], aspect=1, title='CATEGORY DISTRIBUTION'): This creates a subplot at the position (0, 1) within the grid. The aspect ratio is set to 1 to make the pie chart a perfect circle. The title 'CATEGORY DISTRIBUTION' is added to the plot.

source_pie = plt.pie(targetCounts, labels=targetLabels, autopct='%1.1f%%', shadow=True, colors=colors): This creates the pie chart using the pie function. It takes the targetCounts as the data to be plotted, targetLabels as the category labels for each wedge, and autopct='%1.1f%%' to display the percentage values on the chart. The shadow=True adds a shadow effect to the pie chart, and the colors parameter sets the colors of the wedges.

```
from matplotlib.gridspec import GridSpec
targetCounts = resumeDataSet['Category'].value_counts()
targetLabels = resumeDataSet['Category'].unique()
#Make square figures and axes
plt.figure(1,figsize=(25,25))
the_grid = GridSpec(2,2)

cmap = plt.get_cmap('coolwarm')
colors = [cmap(i) for i in np.linspace(0,1,3)]
plt.subplot(the_grid[0,1], aspect=1, title='CATEGORY DISTRIBUTION')

source_pie = plt.pie(targetCounts, labels=targetLabels, autopct='%1.1f%%', shadow=True, colors = colors)
plt.show()
```



import re: This imports the Python re module, which provides support for working with regular expressions.

re.sub('http\S+\s*', '', resumeText): This removes URLs (starting with 'http') from the resume text. **re.sub('RT|cc', '', resumeText)**: This removes 'RT' (Retweet) and 'cc' (carbon copy) mentions from the text.

re.sub('#\S+', '', resumeText): This removes hashtags and any text that follows them (e.g., #job, #resume).

re.sub('@\S+', ' ', resumeText): This removes Twitter handles (@username) and replaces them with a space.

re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[]^_{|}~"""), ' ', resumeText)`: This removes special characters and replaces them with spaces.

re.sub(r'[^\x00-\x7f]', r' ', resumeText): This removes non-ASCII characters from the text.

re.sub('\s+', ' ', resumeText): This replaces multiple consecutive spaces with a single space.

resumeDataSet['cleaned_resume'] = resumeDataSet.Resume.apply(lambda x: cleanResume(x)): This line applies the cleanResume function to each element in the 'Resume' column of the resumeDataSet DataFrame using the apply method. The cleaned text is then stored in the newly created 'cleaned_resume' column.

```
import re
def cleanResume(resumeText):
  resumeText = re.sub('http\S+\s*', '', resumeText)
  resumeText = re.sub('RT|cc', '', resumeText)
  resumeText = re.sub('#\S+', '', resumeText)
  resumeText = re.sub('@\S+', ' ', resumeText)
  resumeText = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~"""), ' ', resumeText)
  resumeText = re.sub(r'[^\x00-\x7f]',r' ', resumeText)
  resumeText = re.sub('\s+', ' ', resumeText)
  return resumeText

resumeDataSet['cleaned_resume'] = resumeDataSet.Resume.apply(lambda x: cleanResume(x))
```

nltk.download('stopwords') and nltk.download('punkt'): These lines download the necessary NLTK resources for stopwords and tokenization, respectively. The downloads are required only once.

from nltk.corpus import stopwords: This imports the stopwords corpus from NLTK, which contains common words that are usually removed from text processing tasks.

**oneSetOfStopWords = set(stopwords.words('english')+['``',"''"])**: This creates a set of stopwords for the English language, adding two additional tokens ('``' and "''") that are sometimes present in text data.

totalWords = []: This initializes an empty list to store all the words from the cleaned resume data.

Sentences = resumeDataSet['Resume'].values: This extracts all the resume texts from the 'Resume' column of the resumeDataSet DataFrame and stores them in the Sentences variable.

cleanedSentences = "": This initializes an empty string to store the concatenated cleaned resume text.

The code then runs a loop from 0 to 159 (160 resumes) to process each resume text:

cleanedText = cleanResume(Sentences[i]): The function cleanResume is called to clean the current resume text, and the result is stored in cleanedText.

cleanedSentences += cleanedText: The cleaned resume text is concatenated to the cleanedSentences string. The cleaned text is then tokenized into words using nltk.word_tokenize(). Each word is checked to see if it is not a stopword or punctuation, and if so, it is added to the totalWords list.

wordfreqdist = nltk.FreqDist(totalWords): This creates a frequency distribution of words in the totalWords list using the FreqDist() function from NLTK.

wc = WordCloud().generate(cleanedSentences): This creates a WordCloud object using the WordCloud library and generates the word cloud from the cleanedSentences.

plt.imshow(wc, interpolation='bilinear'): This displays the word cloud using plt.imshow(), with 'bilinear' interpolation for smoother visualization.

plt.axis("off"): This removes the axis ticks and labels from the plot.

````
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
import string
from wordcloud import WordCloud

oneSetOfStopWords = set(stopwords.words('english')+['``',"''"])
totalWords =[]
Sentences = resumeDataSet['Resume'].values
cleanedSentences = ""
for i in range(0,160):
    cleanedText = cleanResume(Sentences[i])
    cleanedSentences += cleanedText
    requiredWords = nltk.word_tokenize(cleanedText)
    for word in requiredWords:
        if word not in oneSetOfStopWords and word not in string.punctuation:
            totalWords.append(word)

wordfreqdist = nltk.FreqDist(totalWords)
mostcommon = wordfreqdist.most_common(50)
print(mostcommon)

wc = WordCloud().generate(cleanedSentences)
plt.figure(figsize=(15,15))
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
````

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
```

The code you provided is using the LabelEncoder class from scikit-learn to convert categorical variables in the resumeDataSet DataFrame into numeric representations. The LabelEncoder assigns a unique integer to each unique category in the specified column. **le.fit_transform(resumeDataSet[i])**: This fits the LabelEncoder to the data in the 'Category' column and transforms the categories into numeric labels.

resumeDataSet[i] = le.fit_transform(resumeDataSet[i]): This replaces the original categorical values in the 'Category' column with their corresponding integer labels.

```
from sklearn.preprocessing import LabelEncoder

var_mod = ['Category']
le = LabelEncoder()
for i in var_mod:
    resumeDataSet[i] = le.fit_transform(resumeDataSet[i])
```

from sklearn.feature_extraction.text import TfidfVectorizer: This imports the TfidfVectorizer class from scikit-learn, which is used to convert the raw text data into a numerical feature matrix using the Term Frequency-Inverse Document Frequency (TF-IDF) representation. **from scipy.sparse import hstack**: This imports the hstack function from SciPy, which is used to horizontally stack sparse matrices

word_vectorizer = TfidfVectorizer(sublinear_tf=True, stop_words='english', max_features=1500): This creates an instance of the TfidfVectorizer class with specific settings:

sublinear_tf=True: Applies sublinear scaling to the term frequency to make the representation less sensitive to absolute frequency values. stop_words='english': Removes common English stop words from the text data during vectorization. **max_features=1500:** Limits the maximum number of features (vocabulary size) to 1500. The most frequently occurring words will be included in the vocabulary.

word_vectorizer.fit(requiredText): This fits the TfidfVectorizer to the cleaned resume text in the requiredText array, building the vocabulary and calculating the TF-IDF weights.

WordFeatures = word_vectorizer.transform(requiredText): This transforms the cleaned resume text into a TF-IDF feature matrix using the fitted TfidfVectorizer. The resulting WordFeatures will be a sparse matrix representing the TF-IDF values for each resume.

X_train, X_test, y_train, y_test = train_test_split(WordFeatures, requiredTarget, random_state=0, test_size=0.2): This splits the data into training and testing sets. It uses the train_test_split function with the following arguments:

WordFeatures: The TF-IDF feature matrix representing the input text data. requiredTarget: The target variable (resume categories). **random_state=0:** A fixed random seed for reproducibility. test_size=0.2: The proportion (20%) of the data to be allocated for testing, while the remaining 80% will be used for training. print(X_train.shape): This prints the shape of the training feature matrix X_train, indicating the number of samples (resumes) and the number of features (TF-IDF dimensions).

print(X_test.shape): This prints the shape of the testing feature matrix X_test, indicating the number of samples (resumes) and the number of features (TF-IDF dimensions).

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack

requiredText = resumeDataSet['cleaned_resume'].values
requiredTarget = resumeDataSet['Category'].values

word_vectorizer = TfidfVectorizer(
    sublinear_tf=True,
    stop_words='english',
    max_features=1500)
word_vectorizer.fit(requiredText)
WordFeatures = word_vectorizer.transform(requiredText)

print ("Feature completed .....")

X_train,X_test,y_train,y_test = train_test_split(WordFeatures,requiredTarget,random_state=0, test_size=0.2)
print(X_train.shape)
print(X_test.shape)
```

```
Feature completed .....
(769, 1500)
(193, 1500)
```

from sklearn.multiclass import OneVsRestClassifier: This imports the OneVsRestClassifier class from scikit-learn, which allows us to perform multiclass classification using multiple binary classifiers (e.g., KNeighborsClassifier).

clf = OneVsRestClassifier(KNeighborsClassifier()): This creates an instance of OneVsRestClassifier with KNeighborsClassifier() as the base classifier. The KNeighborsClassifier() is a k-Nearest Neighbors classifier, which is suitable for multi-class classification tasks.

clf.fit(X_train, y_train): This trains the OneVsRestClassifier on the training data X_train and their corresponding target labels y_train.

prediction = clf.predict(X_test): This uses the trained classifier to predict the target labels for the test data X_test.

print('Accuracy of KNeighbors Classifier on training set: {:.2f}'.format(clf.score(X_train, y_train))): This calculates and prints the accuracy of the classifier on the training set.

print('Accuracy of KNeighbors Classifier on test set: {:.2f}'.format(clf.score(X_test, y_test))): This calculates and prints the accuracy of the classifier on the test set.

print("\n Classification report for classifier %s:\n%s\n" % (clf, metrics.classification_report(y_test, prediction))): This generates and prints a classification report that includes metrics such as precision, recall, F1-score, and support for each class. The classification report is based on the predictions made by the classifier on the test set (y_test) and the true target labels (prediction).

```
from sklearn.multiclass import OneVsRestClassifier
clf = OneVsRestClassifier(KNeighborsClassifier())
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
print('Accuracy of KNeighbors Classifier on training set: {:.2f}'.format(clf.score(X_train, y_train)))
print('Accuracy of KNeighbors Classifier on test set: {:.2f}'.format(clf.score(X_test, y_test)))

print("\n Classification report for classifier %s:\n%s\n" % (clf, metrics.classification_report(y_test, prediction)))
```

```
Accuracy of KNeighbors Classifier on training set: 0.99
Accuracy of KNeighbors Classifier on test set: 0.99

 Classification report for classifier OneVsRestClassifier(estimator=KNeighborsClassifier()):
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         3
           2       1.00      0.80      0.89         5
           3       1.00      1.00      1.00         9
           4       1.00      1.00      1.00         6
           5       0.83      1.00      0.91         5
           6       1.00      1.00      1.00         9
           7       1.00      1.00      1.00         7
           8       1.00      0.91      0.95        11
           9       1.00      1.00      1.00         9
          10       1.00      1.00      1.00         8
          11       0.90      1.00      0.95         9
          12       1.00      1.00      1.00         5
          13       1.00      1.00      1.00         9
          14       1.00      1.00      1.00         7
          15       1.00      1.00      1.00        19
          16       1.00      1.00      1.00         3
          17       1.00      1.00      1.00         4
          18       1.00      1.00      1.00         5
          19       1.00      1.00      1.00         6
          20       1.00      1.00      1.00        11
          21       1.00      1.00      1.00         4
          22       1.00      1.00      1.00        13
          23       1.00      1.00      1.00        15
          24       1.00      1.00      1.00         8

    accuracy                           0.99       193
   macro avg       0.99      0.99      0.99       193
weighted avg       0.99      0.99      0.99       193
```

```
https://thecleverprogrammer.com/2020/12/06/resume-screening-with-python/
```

0s completed at 1:51 AM