```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import re
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

```
from sklearn.metrics import confusion_matrix,classification_report
from collections import Counter
from wordcloud import WordCloud
```

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
```

```
[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...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
from sklearn.model_selection import train_test_split

from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import classification_report, accuracy_score
```

```
data=pd.read_csv('/content/drive/MyDrive/resume matching datset/Resume/Resume.csv')

# Split the data into train and test sets
resume_data, resume_test_data = train_test_split(data, test_size=0.15,stratify=data['Category'], random_state=42)

# Save the test data to CSV files
resume_test_data.to_csv('test_resume_data.csv', index=False)
```

```
resume_data.head()
```

	ID	Resume_str	Resume_html	Category
1443	16066857	SENIOR EXECUTIVE CHEF Execu...	<div class="fontsize fontface vmargins hmargin...	CHEF
56	52979663	SENIOR HR Highlights ...	<div class="fontsize fontface vmargins hmargin...	HR
1131	15281412	CONSULTANT Summary Transitio...	<div class="fontsize fontface vmargins hmargin...	CONSULTANT
		SUPERVISORY LOGISTICS	<div class="fontsize fontface vmargins	

Preprocessing

- convert all characters in the string to lower case.
- remove non-english characters, punctuation and numbers.
- tokenize word
- stemming

```
resume_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2111 entries, 1443 to 1999
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ID           2111 non-null   int64
 1   Resume_str   2111 non-null   object
 2   Resume_html  2111 non-null   object
 3   Category     2111 non-null   object
dtypes: int64(1), object(3)
memory usage: 82.5+ KB
```

```
resume_data.shape
```

```
(2111, 4)
```

```
resume_data['Category'].unique()
```

```
array(['CHEF', 'HR', 'CONSULTANT', 'AVIATION', 'ENGINEERING', 'BANKING',
       'BUSINESS-DEVELOPMENT', 'ADVOCATE', 'FINANCE', 'DIGITAL-MEDIA',
       'CONSTRUCTION', 'HEALTHCARE', 'DESIGNER', 'AGRICULTURE', 'ARTS',
       'FITNESS', 'TEACHER', 'PUBLIC-RELATIONS', 'APPAREL',
       'INFORMATION-TECHNOLOGY', 'ACCOUNTANT', 'SALES', 'BPO',
       'AUTOMOBILE'], dtype=object)
```

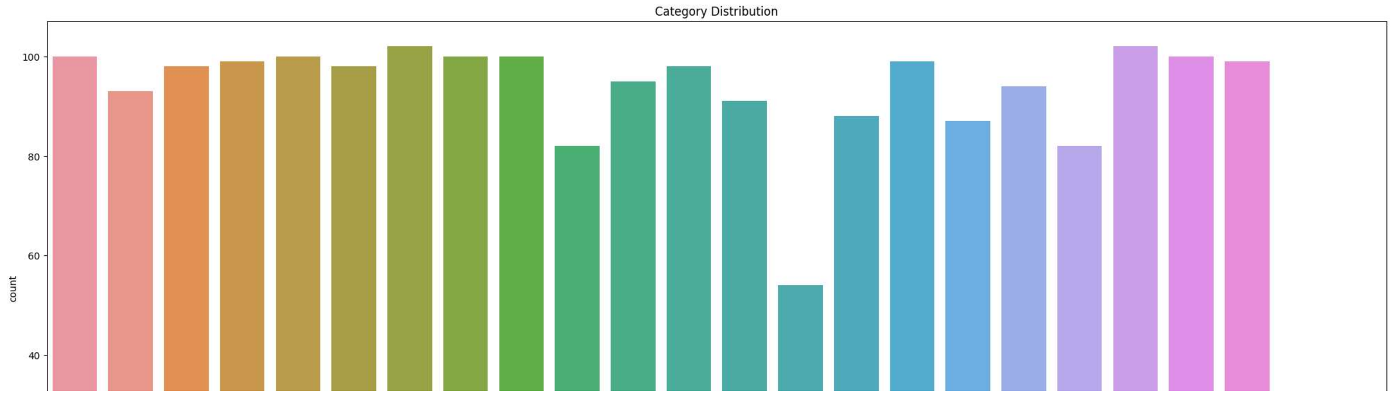
```
resume_data['Category'].value_counts()
```

```
BUSINESS-DEVELOPMENT      102
INFORMATION-TECHNOLOGY    102
CHEF                      100
ENGINEERING               100
ADVOCATE                  100
FINANCE                   100
ACCOUNTANT                100
AVIATION                   99
SALES                      99
FITNESS                    99
CONSULTANT                 98
BANKING                    98
HEALTHCARE                 98
CONSTRUCTION               95
PUBLIC-RELATIONS           94
HR                         93
DESIGNER                   91
ARTS                       88
TEACHER                    87
APPAREL                    82
DIGITAL-MEDIA              82
AGRICULTURE                54
AUTOMOBILE                 31
BPO                        19
Name: Category, dtype: int64
```

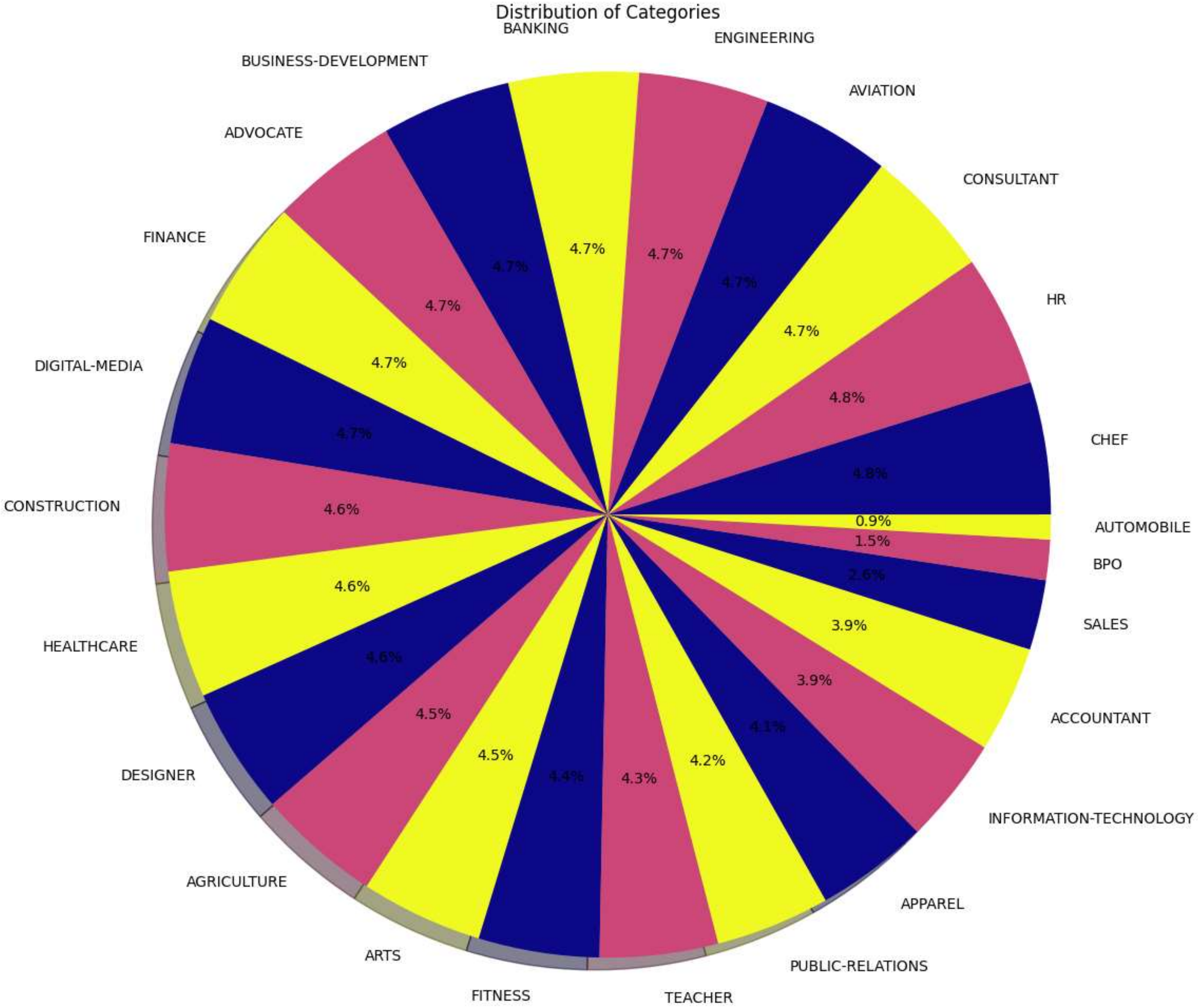
```
len(resume_data['Category'].value_counts())
```

```
24
```

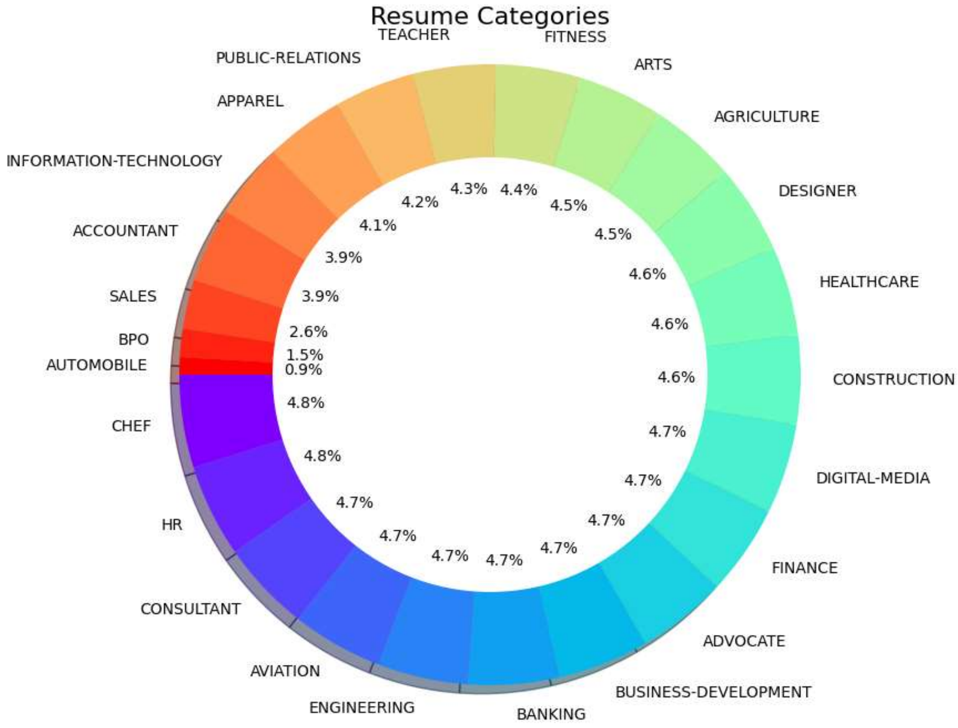
```
plt.figure(figsize=(25, 10))
sns.countplot(x=resume_data['Category'])
plt.xticks(rotation=150)
plt.title('Category Distribution')
plt.show()
```



```
counts=resume_data['Category'].value_counts()
labels=resume_data['Category'].unique()

plt.figure(figsize=(15,12))
plt.pie(counts,labels= labels, autopct='%1.1f%%',shadow=True, colors=plt.cm.plasma(np.linspace(0,1,3)))
plt.title('Distribution of Categories')
plt.axis('equal')
plt.show()
```



```
colors = plt.cm.rainbow(np.linspace(0, 1, len(labels)))
plt.figure(figsize=(12, 8))
plt.pie(counts, labels=labels, autopct='%1.1f%%', shadow=True, colors=colors, startangle=180)
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
plt.axis('equal')

plt.title("Resume Categories", fontsize=16)
plt.show()
```

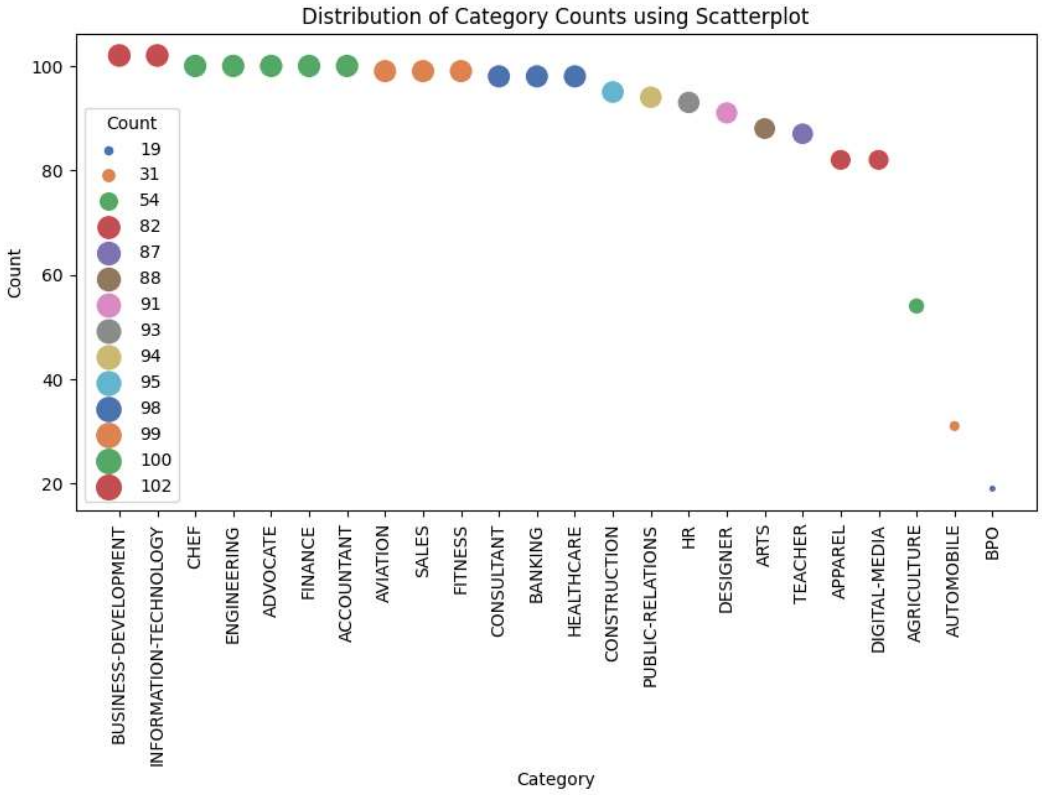


```
import plotly.graph_objects as go
counts=resume_data['Category'].value_counts()
labels=resume_data['Category'].unique()
fig = go.Figure(data=[go.Pie(labels=labels, values=counts)])
fig.update_layout(title='Resume Categories')
fig.show()
```

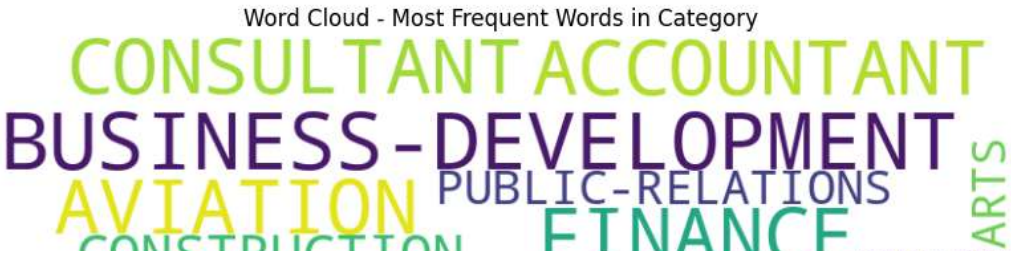
Resume Categories

■ CHEF

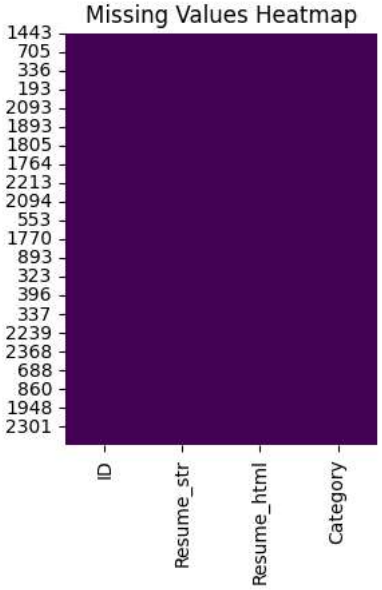
```
category_counts = resume_data['Category'].value_counts()
category_counts_df = category_counts.reset_index()
category_counts_df.columns = ['Category', 'Count']

plt.figure(figsize=(10, 5))
sns.scatterplot(data=category_counts_df, x="Category", y="Count" ,size="Count",hue="Count", sizes=(20, 200), hue_norm=(0, 7), palette="deep",
plt.xlabel('Category')
plt.ylabel('Count')
plt.title('Distribution of Category Counts using Scatterplot')
plt.xticks(rotation=90)
plt.show()
```



```
combined_text = ' '.join(resume_data['Category'])
word_counts = Counter(combined_text.split())
wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(word_counts)
plt.figure(figsize=(10, 6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud - Most Frequent Words in Category')
plt.show()
```



```
missing_values = resume_data.isnull().sum()
plt.figure(figsize=(3, 4))
sns.heatmap(resume_data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```



```
resume_data.isnull().sum()
```

```
ID             0
Resume_str     0
Resume_html    0
Category       0
dtype: int64
```

```
print(resume_data['Resume_str'][0])
```

```
              HR ADMINISTRATOR/MARKETING ASSOCIATE

HR ADMINISTRATOR       Summary     Dedicated Customer Service Manager with 15+ years of experience in Hospitality and Customer Service M

HR Administrator     Dec 2013    to    Current      Company Name    -    City   ,    State     Helps to develop policies, directs and coordir
```

```
def resume_cleaning(text):
    cleaned_text = re.sub(r'<.*?>', ' ', text)
    cleaned_text = re.sub('[^a-zA-Z]', ' ', cleaned_text)
    cleaned_text = re.sub(r'[^\w\s]|_', ' ', cleaned_text)
    cleaned_text = re.sub(r'\d+', ' ', cleaned_text)
    cleaned_text = re.sub(r'\s+', ' ', cleaned_text).strip()
    cleaned_text=re.sub('http\S+\s', " ", cleaned_text)
    cleaned_text = cleaned_text.lower()
    words = word_tokenize(cleaned_text)
    stop_words = set(stopwords.words('english'))
    filtered_words = [word for word in words if word not in stop_words]
    words = word_tokenize(cleaned_text)
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in filtered_words]

    cleaned_text = ' '.join(stemmed_words)
    return cleaned_text
```

```
resume_data['Cleaned_Resume']=resume_data['Resume_str'].apply(lambda x:resume_cleaning(x))
```

```
resume_data.head()
```

	ID	Resume_str	Resume_html	Category	Cleaned_Resume
1443	16066857	SENIOR EXECUTIVE CHEF Execu...	<div class="fontsize fontface vmargins hmargin...	CHEF	senior execut chef execut profil seek employ e...
56	52979663	SENIOR HR Highlights ...	<div class="fontsize fontface vmargins hmargin...	HR	senior hr highlight safeti managementemploye e...
1131	15281412	CONSULTANT Summary Transitio...	<div class="fontsize fontface vmargins hmargin...	CONSULTANT	consult summari transit applic develop secur p...
2473	[illegible]	SUPERVISORY LOGISTICS	<div class="fontsize fontface vmargins	AVIATION	supervisori logist manag specialist

```
empty_rows = resume_data[resume_data['Cleaned_Resume'] == '']
print(empty_rows)
```

```
           ID            Resume_str  \
656  12632728

                                          Resume_html             Category  \
656  <div class="fontsize fontface vmargins hmargin...  BUSINESS-DEVELOPMENT

    Cleaned_Resume
656
```

```
resume_data=resume_data.drop(['Resume_str', 'Resume_html'], axis=1)
resume_data = resume_data.drop(empty_rows.index)
```

```
resume_data.Cleaned_Resume[0]
```

```
'hr administr market associ hr administr summari dedic custom servic manag year experi hospit custom servic manag respect builder leade
r custom focus team strive instil share enthusiast commit custom servic highlight focus custom satisfact team manag market savvi confli
ct resolut techniqu train develop skill multi tasker client relat specialist accomplish missouri dot supervisor train certif certifi ih
g custom loyalti market segment hilton worldwid gener manag train certif accomplish trainer cross server hospit system hilton onq micro
opera pm fidelio opera reserv system or holidex complet cours seminar custom servic sale strategi inventori control loss prevent safeti
time manag leadership perform assess experi hr administr market associ hr administr dec current compani name citi state help develop po
lici direct coordin activ employ compens labor relat benefit train employe servic prepar employe separ notic relat document keep record
```

```
categories = np.sort(resume_data['Category'].unique())
categories
# create new df for corpus and category
df_categories = [resume_data[resume_data['Category'] == category].loc[:, ['Cleaned_Resume', 'Category']] for category in categories]
```

```
def wordcloud(df):
    txt = ' '.join(txt for txt in resume_data['Cleaned_Resume'])
    wordcloud = WordCloud(
        height=2000,
        width=4000
    ).generate(txt)

    return wordcloud
```

```
plt.figure(figsize=(32, 20))

for i, category in enumerate(categories):
    wc = wordcloud(df_categories[i])

    plt.subplot(5, 5, i + 1).set_title(category)
    plt.imshow(wc)
    plt.axis('off')
    plt.plot()

plt.show()
plt.close()
```

```
def remove_extra_word(text):

    extra_word=['compani', 'name', 'citi', 'state', 'work', 'manag'] # extra words
    words = text.split()  # Split the text into words

    # Filter out the extra words
    filter_word = [word for word in words if word not in extra_word]

    filter_text = ' '.join(filter_word)

    return filter_text
```

```
# apply resume_data['Cleaned_Resume']

resume_data['Cleaned_Resume']=resume_data['Cleaned_Resume'].apply(lambda x:remove_extra_word(x))
```

```
plt.figure(figsize=(32, 20))

for i, category in enumerate(categories):
    wc = wordcloud(df_categories[i])

    plt.subplot(5, 5, i + 1).set_title(category)
    plt.imshow(wc)
    plt.axis('off')
    plt.plot()

plt.show()
plt.close()
```

```
from sklearn.preprocessing import LabelEncoder
encoder=LabelEncoder()
resume_data['Category']=encoder.fit_transform(resume_data['Category'])
```

```
resume_data.head()
```

```
resume_data.Category.unique()
```

```
X_train, X_valid, y_train, y_valid = train_test_split(resume_data['Cleaned_Resume'], resume_data['Category'], test_size=0.15, random_state=42
```

```
# Print the sizes of the split datasets
print("Train data size:", X_train.shape)
print("Validation data size:", X_valid.shape)
```

```
train_size = X_train.shape[0]
val_size = X_valid.shape[0]
test_size=resume_test_data.shape[0]  # our test data, which is separate from the full data

# Labels for the pie chart
labels = ['Train', 'Validation', 'Test']

# Sizes of the pie slices
sizes = [train_size, val_size, test_size]

# Colors for each slice
colors = ['yellow', 'oran', 'green']
# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(sizes, labels=labels, colors=colors, shadow=True, autopct='%1.1f%%', startangle=140)
plt.title('Dataset Proportions')
plt.axis('equal')
plt.show()
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
tfidf=TfidfVectorizer(stop_words='english',max_features=800)
```

```
tfidf_train_vectors = tfidf.fit_transform(X_train)
tfidf_valid_vectors =tfidf.transform(X_valid)
```

```
tfidf_train_vectors.shape
```

```
tfidf.get_feature_names_out()
```

```
accuracy_lis=[]
model_lis=[]
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
RF = RandomForestClassifier()

RF.fit(tfidf_train_vectors,y_train)
# Predict on validation data
y_val_pred = RF.predict(tfidf_valid_vectors)
```

```
# Print classification report for validation data
print("Classification Report (Validation Data):\n")
print(classification_report(y_valid, y_val_pred))

accuracy=accuracy_score(y_valid, y_val_pred)
print("Accuracy is : ", accuracy)

# store info
model_lis.append("Random Forest Classifier")
accuracy_lis.append(accuracy*100)
```

```
from sklearn.linear_model import LogisticRegression
```

```
LR = LogisticRegression()

LR.fit(tfidf_train_vectors,y_train)
# Predict on validation data
y_val_pred = LR.predict(tfidf_valid_vectors)

# Print classification report for validation data
print("Classification Report (Validation Data):\n")
print(classification_report(y_valid, y_val_pred))

print("Accuracy is : ", accuracy_score(y_valid, y_val_pred))

# store info
model_lis.append("Logistic Regression")
accuracy_lis.append(accuracy_score(y_valid, y_val_pred)*100)
```

```
from sklearn.neighbors import KNeighborsClassifier
```

```
k = 24 # Number of neighbors
knn_classifier = KNeighborsClassifier(n_neighbors=k)

# Train the KNN classifier
knn_classifier.fit(tfidf_train_vectors,y_train)

# Predict on validation data
y_val_pred = knn_classifier.predict(tfidf_valid_vectors)

# Print classification report for validation data
print("Classification Report (Validation Data):\n")
print(classification_report(y_valid, y_val_pred))

print("Accuracy is : ", accuracy_score(y_valid, y_val_pred))

# store info
model_lis.append("K Nearest Neighbors")
accuracy_lis.append(accuracy_score(y_valid, y_val_pred)*100)
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
nb_classifier = MultinomialNB()

# Train the KNN classifier
nb_classifier.fit(tfidf_train_vectors,y_train)
```

```
# Predict on validation data
y_val_pred = nb_classifier.predict(tfidf_valid_vectors)

# Print classification report for validation data
print("Classification Report (Validation Data):\n")
print(classification_report(y_valid, y_val_pred))

print("Accuracy is : ", accuracy_score(y_valid, y_val_pred))

# store info
model_lis.append("Naive Bayes")
accuracy_lis.append(accuracy_score(y_valid, y_val_pred)*100)
```

```
from sklearn.svm import SVC

# Initialize SVM classifier
svm_classifier = SVC()

# Train the classifier
svm_classifier.fit(tfidf_train_vectors,y_train)

# Predict on validation data
y_val_pred = svm_classifier.predict(tfidf_valid_vectors)

# Print classification report for validation data
print("Classification Report (Validation Data):\n")
print(classification_report(y_valid, y_val_pred))

print("Accuracy is : ", accuracy_score(y_valid, y_val_pred))

# store info
model_lis.append("Support Vector Machine")
accuracy_lis.append(accuracy_score(y_valid, y_val_pred)*100)
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.utils import to_categorical
```

```
tfidf_train_arrays = tfidf_train_vectors.toarray()
tfidf_valid_arrays = tfidf_valid_vectors.toarray()

# Build a simple neural network model
num_classes = 24

y_train_label = to_categorical(y_train, num_classes=num_classes)
y_valid_label = to_categorical(y_valid, num_classes=num_classes)
# Build a more complex neural network model
model = Sequential()
model.add(Dense(1000, input_dim=tfidf_train_arrays.shape[1]))
model.add(Dense(500, activation='relu'))
model.add(Dense(200, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))  # Use softmax for multi-class classification

# Compile the model with a lower learning rate
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=['accuracy'])

# Train the model with more epochs
history = model.fit(tfidf_train_arrays, y_train_label, epochs=50, batch_size=32, validation_data=(tfidf_valid_arrays, y_valid_label))

# Evaluate the model on the validation set
loss, accuracy = model.evaluate(tfidf_valid_arrays, y_valid_label)
print(f"Validation loss: {loss:.4f}")
print(f"Validation accuracy: {accuracy:.4f}")

# store info
model_lis.append("Artificial Neural Network")
accuracy_lis.append(accuracy_score(y_valid, y_val_pred)*100)
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])

plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])

plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'], loc='upper left')
plt.show()
```

```
accuracy_data = pd.DataFrame({'model': model_lis, 'accuracy': accuracy_lis})

# Plot the data
plt.figure(figsize=(10, 6))
plt.bar(accuracy_data['model'], accuracy_data['accuracy'])
plt.xlabel('Model Name')
plt.ylabel('Accuracy')
plt.xticks(rotation=45)
plt.title('Overview of the models and accuracy')
plt.show()
```

```
accuracy_data
```

```
import pickle
pickle.dump(tfidf,open('tfidf.pkl', 'wb'))
pickle.dump(RF,open('best_clf.pkl', 'wb'))
```

```
category_mapping = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))
```

```
category_mapping
```