Introduction:

We have been entrusted with data from the Memorial Sloan Kettering Cancer Center (MSKCC) that revolves around cancer tumors and the multitude of genetic mutations associated with them. The challenge lies in distinguishing the driver mutations that have the most significant impact on tumor growth. Currently, pathologists manually sift through vast amounts of data, relying on text-based clinical literature to categorize each genetic mutation. This process is not only time-consuming but also resource-intensive. Our objective is to develop a Machine Learning model that streamlines the classification of genetic mutations, expediting the identification of crucial mutations for more prompt and accurate patient treatment.

It is imperative that our model is interpretable, aiding pathologists in making informed decisions. Given the high stakes involved, with potential life-threatening consequences for patients, precision is paramount. Therefore, the model we construct must demonstrate exceptional accuracy to minimize the risk of misinterpretations that could have severe implications for patient outcomes.

Importing Libraries

In [46]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import string

# Text Processing libraries
import re
from wordcloud import WordCloud
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import normalize
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE

# Metrices
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import log_loss

# Visualizer
import plotly.express as px

# Models
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split

import plotly.offline as py
py.init_notebook_mode(connected=True)
import warnings
warnings.filterwarnings('ignore')
```

Data Study

In [2]:

```
df_variants = pd.read_csv('training_variants.csv')
```

```
df_variants.head(10)
```

Out[2]:

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4
5	5	CBL	V391I	4
6	6	CBL	V430M	5
7	7	CBL	Deletion	1
8	8	CBL	Y371H	4
9	9	CBL	C384R	4

In [3]:

```
df_text = pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"],skiprows=1)
df_text.head(10)
```

Out[3]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...
5	5	Oncogenic mutations in the monomeric Casitas B...
6	6	Oncogenic mutations in the monomeric Casitas B...
7	7	CBL is a negative regulator of activated recep...
8	8	Abstract Juvenile myelomonocytic leukemia (JM...
9	9	Abstract Juvenile myelomonocytic leukemia (JM...

In [4]:

```
df_variants.isnull().sum()
```

Out[4]:

```
ID           0
Gene         0
Variation    0
Class        0
dtype: int64
```

In [5]:

```
df_text.isnull().sum()
```

Out[5]:

```
ID      0
TEXT    5
```

```
TEXT    5
dtype: int64
```

In [6]:

```
df_text.shape
```

Out[6]:

```
(3321, 2)
```

In [7]:

```
df_variants.shape
```

Out[7]:

```
(3321, 4)
```

In [8]:

```
df_text = df_text.sample(frac=0.33, replace=True, random_state=42)
df_variants = df_variants.sample(frac=0.33, replace=True, random_state=42)
print(df_text.shape)
print(df_variants.shape)
```

```
(1096, 2)
(1096, 4)
```

Text Pre Process:

In [9]:

```
Stopwords = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def pre_process(text):
    text = str(text).lower()
    # remove HTML tags
    text = re.sub('<.*?>', ' ', text)
    # remove punctuations
    text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text)
    # replace special characters with space
    text = re.sub('[^a-zA-Z0-9\n]', ' ', text)
    # remove multiple spaces
    text = re.sub(r'\s+', ' ', text)
    # tokenize text
    text_tokens = word_tokenize(text)
    # remove stopwords
    text_tokens = [word for word in text_tokens
                   if word not in Stopwords]
    # lemmatize the words
    text_tokens = [lemmatizer.lemmatize(word)
                   for word in text_tokens]
    # Join them
    processed_text = ' '.join(text_tokens)

    return processed_text
```

In [10]:

```
df_text['TEXT'] = df_text['TEXT'].apply(pre_process)
df = pd.merge(df_variants, df_text, on='ID')
```

In [11]:

```
df.head()
```

Out[11]:

	ID	Gene	Variation	Class	TEXT

	ID	Gene	Variation	Class	TEXT
0	3174	RAB35	F161L	7	shrna screen gene affect akt phosphorylation i...
1	860	ABL1	F317L	2	abstract mutation bcr abl kinase domain may ca...
2	1294	HRAS	Q22K	2	several group shown noonan syndrome n omim 163...
3	1130	MET	D1010H	2	met proto oncogene receptor tyrosine kinase ge...
4	1095	MAP3K1	E1286V	5	langerhans cell histiocytosis lch understood n...

In [12]:

```
df['TEXT'].isnull().sum()
```

Out[12]:

```
0
```

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
ID            0
Gene          0
Variation     0
Class         0
TEXT          0
dtype: int64
```

In [14]:

```
df['Class'].value_counts()
```

Out[14]:

```
Class
7    453
4    303
1    219
2    191
6    140
5     98
3     32
9     24
8     12
Name: count, dtype: int64
```

Visualizations

In [15]:

```
plot_class = df['Class'].value_counts().sort_index()
fig = px.bar(plot_class,title='Frequency Distribution for all classes',text_auto=True)
fig.show()
```

In [16]:

```
plot_gene = df['Gene'].value_counts()
fig = px.bar(plot_gene, title='Frequency Distribution for all genes')
fig.show()
```

In [17]:

```
fig = px.ecdf(plot_gene)
fig.show()
```

In [18]:

```
top_genes = df['Gene'].value_counts().head(10)
fig = px.bar(top_genes, title='''Frequency Distribution for most common genes''')
fig.show()
```

Observations:

1. The dataset is imbalanced. Class 7 has most data-points and Class 8 has the least.
2. BRCA1, TP53, EGFR are the most common Genes

Feature Engineering

In [19]:

```
df.head()
```

Out[19]:

	ID	Gene	Variation	Class	TEXT
0	3174	RAB35	F161L	7	shrna screen gene affect akt phosphorylation i...
1	860	ABL1	F317L	2	abstract mutation bcr abl kinase domain may ca...
2	1294	HRAS	Q22K	2	several group shown noonan syndrome n omim 163...
3	1130	MET	D1010H	2	met proto oncogene receptor tyrosine kinase ge...
4	1095	MAP3K1	E1286V	5	langerhans cell histiocytosis lch understood n...

In [20]:

```
df['Text_Length'] = df['TEXT'].apply(lambda x: len(x))
df['Word_Count'] = df['TEXT'].apply(lambda x: len(x.split()))
df['Gene_Variation_TEXT'] = df['Gene'] + '_' + df['Variation'] + " " + df['TEXT']
df.head()
```

Out[20]:

	ID	Gene	Variation	Class	TEXT	Text_Length	Word_Count	Gene_Variation_TEXT
0	3174	RAB35	F161L	7	shrna screen gene affect akt phosphorylation i...	15268	2027	RAB35_F161L shrna screen gene affect akt phosp...
1	860	ABL1	F317L	2	abstract mutation bcr abl kinase domain may ca...	14285	1914	ABL1_F317L abstract mutation bcr abl kinase do...
2	1294	HRAS	Q22K	2	several group shown noonan syndrome n omim 163...	8584	1111	HRAS_Q22K several group shown noonan syndrome ...
3	1130	MET	D1010H	2	met proto oncogene receptor tyrosine kinase ge...	187961	26599	MET_D1010H met proto oncogene receptor tyrosin...
4	1095	MAP3K1	E1286V	5	langerhans cell histiocytosis lch understood n...	17879	2586	MAP3K1_E1286V langerhans cell histiocytosis lc...

In [21]:

```
df_copy = df
```

In [22]:

```
df.shape
```

Out[22]:

```
(1472, 8)
```

Modelling

Univariate analysis on TEXT columns

In [23]:

```
X = df['TEXT']
y = df['Class']
```

In [24]:

```
X.shape
```

Out[24]:

```
(1472,)
```

In [25]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=100)
bow_X = vectorizer.fit_transform(X)
final_X = bow_X
#print(bow_X[:1])
```

In [26]:

```
oversample = SMOTE()

X_resampled, y_resampled = oversample.fit_resample(final_X, y)
X_resampled.shape
```

Out[26]:

```
(4077, 100)
```

In [27]:

```
y_resampled.value_counts()
```

Out[27]:

```
Class
7    453
2    453
5    453
4    453
9    453
3    453
6    453
1    453
8    453
Name: count, dtype: int64
```

In [28]:

```
X_train, X_test, y_train, y_test = train_test_split(
    final_X, y, test_size=0.2, random_state=0)
```

In [29]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[29]:

```
((1177, 100), (295, 100), (1177,), (295,))
```

RandomForest

In [30]:

```
from sklearn.metrics import accuracy_score, f1_score
for i in range(50, 201, 50):
    print('n_estimators: ', i)
    clf = RandomForestClassifier(n_estimators=i, n_jobs=-1)

    # Training
    clf.fit(X_train, y_train)
    # Test the training data
    y_pred_train = clf.predict(X_train)
    accuracy_train = accuracy_score(y_pred_train,y_train)
```

```
    f1_train = f1_score(y_pred_train,y_train, average='weighted')


    # Test the test data
    y_pred_test = clf.predict(X_test)
    accuracy_test = accuracy_score(y_pred_test,y_test)
    f1_test = f1_score(y_pred_test,y_test, average='weighted')


    print('train accuracy: ',accuracy_train,'test accuracy: ',accuracy_test)
    print('train f1 score: ',f1_train,'test f1 score: ',f1_test)
```

```
n_estimators:  50
train accuracy:  0.945624468988955 test accuracy:  0.7694915254237288
train f1 score:  0.9459047327079815 test f1 score:  0.773965906534416
n_estimators:  100
train accuracy:  0.945624468988955 test accuracy:  0.7559322033898305
train f1 score:  0.9463514338208725 test f1 score:  0.7613635623208302
n_estimators:  150
train accuracy:  0.945624468988955 test accuracy:  0.7661016949152543
train f1 score:  0.9464064737692404 test f1 score:  0.7711490072985043
n_estimators:  200
train accuracy:  0.945624468988955 test accuracy:  0.7694915254237288
train f1 score:  0.9465033606902027 test f1 score:  0.7742145200331363
```

Univariate Analysis on Gene and Variation Columns

In [31]:

```
from sklearn.preprocessing import OneHotEncoder


gene_encoder = OneHotEncoder(sparse=False)  # sparse=False to get a dense array
encoded_gene = gene_encoder.fit_transform(df['Gene'].values.reshape(-1, 1))
gene_columns = [f"Gene_{gene}" for gene in gene_encoder.get_feature_names_out(['Gene'])]
encoded_gene_df = pd.DataFrame(encoded_gene, columns=gene_columns)

variation_encoder = OneHotEncoder(sparse=False)
encoded_variation = variation_encoder.fit_transform(df['Variation'].values.reshape(-1, 1
))
variation_columns = [f"Variation_{variation}" for variation in variation_encoder.get_fea
ture_names_out(['Variation'])]
encoded_variation_df = pd.DataFrame(encoded_variation, columns=variation_columns)
```

Gene

In [32]:

```
X = encoded_gene_df
y = df['Class']
```

In [33]:

```
X.shape, y.shape
```

Out[33]:

```
((1472, 177), (1472,))
```

In [34]:

```
oversample = SMOTE()

X_resampled, y_resampled = oversample.fit_resample(X, y)
X_resampled.shape
```

Out[34]:

```
(4077, 177)
```

In [35]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

In [36]:

```
from sklearn.metrics import accuracy_score, f1_score
for i in range(50, 201, 50):
    print('n_estimators: ', i)
    clf = RandomForestClassifier(n_estimators=i, n_jobs=-1)

    # Training
    clf.fit(X_train, y_train)
    # Test the training data
    y_pred_train = clf.predict(X_train)
    accuracy_train = accuracy_score(y_pred_train,y_train)
    f1_train = f1_score(y_pred_train,y_train, average='weighted')


    # Test the test data
    y_pred_test = clf.predict(X_test)
    accuracy_test = accuracy_score(y_pred_test,y_test)
    f1_test = f1_score(y_pred_test,y_test, average='weighted')


    print('train accuracy: ',accuracy_train,'test accuracy: ',accuracy_test)
    print('train f1 score: ',f1_train,'test f1 score: ',f1_test)
```

```
n_estimators:  50
train accuracy:  0.7051826677994902 test accuracy:  0.5796610169491525
train f1 score:  0.7271770209958094 test f1 score:  0.6177635720116642
n_estimators:  100
train accuracy:  0.7051826677994902 test accuracy:  0.5694915254237288
train f1 score:  0.727197425832304 test f1 score:  0.6065014999670942
n_estimators:  150
train accuracy:  0.7051826677994902 test accuracy:  0.5932203389830508
train f1 score:  0.7291296057147816 test f1 score:  0.6239726618297096
n_estimators:  200
train accuracy:  0.7051826677994902 test accuracy:  0.5796610169491525
train f1 score:  0.7292696971684556 test f1 score:  0.6167789284616848
```

Variation

In [37]:

```
X = encoded_variation_df
y = df['Class']
```

In [38]:

```
oversample = SMOTE()

X_resampled, y_resampled = oversample.fit_resample(X, y)
X_resampled.shape
```

Out[38]:

```
(4077, 855)
```

In [39]:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=0)
```

In [40]:

```
from sklearn.metrics import accuracy_score, f1_score
for i in range(50, 201, 50):
```

```
    print('n_estimators: ', i)
    clf = RandomForestClassifier(n_estimators=i, n_jobs=-1)

    # Training
    clf.fit(X_train, y_train)
    # Test the training data
    y_pred_train = clf.predict(X_train)
    accuracy_train = accuracy_score(y_pred_train,y_train)
    f1_train = f1_score(y_pred_train,y_train, average='weighted')


    # Test the test data
    y_pred_test = clf.predict(X_test)
    accuracy_test = accuracy_score(y_pred_test,y_test)
    f1_test = f1_score(y_pred_test,y_test, average='weighted')


    print('train accuracy: ',accuracy_train,'test accuracy: ',accuracy_test)
    print('train f1 score: ',f1_train,'test f1 score: ',f1_test)
```

```
n_estimators:  50
train accuracy:  0.9762107051826678 test accuracy:  0.6305084745762712
train f1 score:  0.9760875604284007 test f1 score:  0.6305115930404276
n_estimators:  100
train accuracy:  0.9830076465590484 test accuracy:  0.6406779661016949
train f1 score:  0.9828604222947543 test f1 score:  0.64016663056344
n_estimators:  150
train accuracy:  0.9830076465590484 test accuracy:  0.6576271186440678
train f1 score:  0.9830499998289368 test f1 score:  0.6596613789933343
n_estimators:  200
train accuracy:  0.9830076465590484 test accuracy:  0.6576271186440678
train f1 score:  0.9830499998289368 test f1 score:  0.6596613789933343
```

Multivariate Analysis

In [41]:

```
X = df['Gene_Variation_TEXT']
y = df['Class']
```

In [42]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(max_features=100)
bow_X = vectorizer.fit_transform(X)
final_X = bow_X
```

In [43]:

```
oversample = SMOTE()

X_resampled, y_resampled = oversample.fit_resample(final_X, y)
X_resampled.shape
```

Out[43]:

```
(4077, 100)
```

In [44]:

```
X_train, X_test, y_train, y_test = train_test_split(
    final_X, y, test_size=0.2, random_state=0)
```

In [45]:

```
from sklearn.metrics import accuracy_score, f1_score
for i in range(50, 201, 50):
    print('n_estimators: ', i)
```

```
    clf = RandomForestClassifier(n_estimators=i, n_jobs=-1)

    # Training
    clf.fit(X_train, y_train)
    # Test the training data
    y_pred_train = clf.predict(X_train)
    accuracy_train = accuracy_score(y_pred_train,y_train)
    f1_train = f1_score(y_pred_train,y_train, average='weighted')


    # Test the test data
    y_pred_test = clf.predict(X_test)
    accuracy_test = accuracy_score(y_pred_test,y_test)
    f1_test = f1_score(y_pred_test,y_test, average='weighted')


    print('train accuracy: ',accuracy_train,'test accuracy: ',accuracy_test)
    print('train f1 score: ',f1_train,'test f1 score: ',f1_test)
```

```
n_estimators:  50
train accuracy:  0.945624468988955 test accuracy:  0.7694915254237288
train f1 score:  0.946049067442206 test f1 score:  0.7733850482836372
n_estimators:  100
train accuracy:  0.945624468988955 test accuracy:  0.7627118644067796
train f1 score:  0.9459337923154081 test f1 score:  0.7662846043173872
n_estimators:  150
train accuracy:  0.945624468988955 test accuracy:  0.7694915254237288
train f1 score:  0.9462667300548241 test f1 score:  0.7751412018217388
n_estimators:  200
train accuracy:  0.945624468988955 test accuracy:  0.7661016949152543
train f1 score:  0.9464538481606171 test f1 score:  0.771101867774866
```

In [60]:

```
X = df['Gene_Variation_TEXT']  # Assuming 'TEXT' is the column containing text data
y = df['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [61]:

```
X_train.shape, y_train.shape
```

Out[61]:

```
((1177,), (1177,))
```

In [62]:

```
tfidf_vectorizer = TfidfVectorizer(max_features=500)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

In [63]:

```
svm_model = SVC(kernel='linear', C=1.0)
svm_model.fit(X_train_tfidf, y_train)
```

Out[63]:

```
▾ SVC
SVC(kernel='linear')
```

In [64]:

```
y_pred_train = svm_model.predict(X_train_tfidf)
y_pred_test = svm_model.predict(X_test_tfidf)
```

In [65]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Training Set
print("Training Set Accuracy:", accuracy_score(y_train, y_pred_train))
print("Classification Report (Training Set):\n", classification_report(y_train, y_pred_t
rain))
print("Confusion Matrix (Training Set):\n", confusion_matrix(y_train, y_pred_train))

# Test Set
print("\nTest Set Accuracy:", accuracy_score(y_test, y_pred_test))
print("Classification Report (Test Set):\n", classification_report(y_test, y_pred_test))
print("Confusion Matrix (Test Set):\n", confusion_matrix(y_test, y_pred_test))
```

```
Training Set Accuracy: 0.7434154630416313
Classification Report (Training Set):
              precision    recall  f1-score   support

           1       0.69      0.73      0.71       171
           2       0.79      0.41      0.54       148
           3       0.83      0.18      0.29        28
           4       0.79      0.80      0.80       243
           5       0.59      0.53      0.56        74
           6       0.95      0.73      0.82       124
           7       0.70      0.97      0.81       359
           8       0.00      0.00      0.00         8
           9       0.93      0.64      0.76        22

    accuracy                           0.74      1177
   macro avg       0.70      0.55      0.59      1177
weighted avg       0.75      0.74      0.73      1177

Confusion Matrix (Training Set):
 [[125   2   0  19   8   3  14   0   0]
 [  6  61   0   3   0   0  78   0   0]
 [  0   0   5   9   4   1   9   0   0]
 [ 25   1   1 194   8   0  13   0   1]
 [ 11   2   0   6  39   1  15   0   0]
 [  4   3   0   8   7  90  12   0   0]
 [  6   5   0   1   0   0 347   0   0]
 [  0   3   0   4   0   0   1   0   0]
 [  3   0   0   1   0   0   4   0  14]]

Test Set Accuracy: 0.6813559322033899
Classification Report (Test Set):
              precision    recall  f1-score   support

           1       0.64      0.62      0.63        48
           2       0.71      0.28      0.40        43
           3       1.00      0.25      0.40         4
           4       0.74      0.70      0.72        60
           5       0.71      0.50      0.59        24
           6       0.75      0.75      0.75        16
           7       0.65      0.96      0.78        94
           8       0.00      0.00      0.00         4
           9       1.00      1.00      1.00         2

    accuracy                           0.68       295
   macro avg       0.69      0.56      0.58       295
weighted avg       0.68      0.68      0.65       295

Confusion Matrix (Test Set):
 [[30  0  0  9  1  2  6  0  0]
 [ 1 12  0  0  0  0 30  0  0]
 [ 0  0  1  1  1  0  1  0  0]
 [ 8  1  0 42  2  1  6  0  0]
 [ 5  0  0  3 12  1  3  0  0]
 [ 0  0  0  1  1 12  2  0  0]
 [ 3  1  0  0  0  0 90  0  0]
 [ 0  3  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  2]]
```

In [66]:

```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {'kernel': ['linear', 'rbf'], 'C': [0.1, 1, 10, 100]}
grid_search = GridSearchCV(SVC(), param_grid, cv=5)
grid_search.fit(X_train_tfidf, y_train)

best_svm_model = grid_search.best_estimator_
```

In [67]:

```
grid_search.best_params_
```

Out[67]:

```
{'C': 10, 'kernel': 'rbf'}
```

In [71]:

```
svm_model = SVC(kernel='rbf', C=10)
svm_model.fit(X_train_tfidf, y_train)
```

Out[71]:

SVC
SVC(C=10)

In [72]:

```
y_pred_train = svm_model.predict(X_train_tfidf)
y_pred_test = svm_model.predict(X_test_tfidf)
```

In [73]:

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Training Set
print("Training Set Accuracy:", accuracy_score(y_train, y_pred_train))
print("Classification Report (Training Set):\n", classification_report(y_train, y_pred_t
rain))
print("Confusion Matrix (Training Set):\n", confusion_matrix(y_train, y_pred_train))

# Test Set
print("\nTest Set Accuracy:", accuracy_score(y_test, y_pred_test))
print("Classification Report (Test Set):\n", classification_report(y_test, y_pred_test))
print("Confusion Matrix (Test Set):\n", confusion_matrix(y_test, y_pred_test))
```

```
Training Set Accuracy: 0.9320305862361937
Classification Report (Training Set):
              precision    recall  f1-score   support

           1       0.88      0.93      0.91       171
           2       0.93      0.93      0.93       148
           3       0.94      0.54      0.68        28
           4       0.96      0.93      0.95       243
           5       0.78      0.82      0.80        74
           6       0.95      0.94      0.94       124
           7       0.96      0.98      0.97       359
           8       1.00      1.00      1.00         8
           9       1.00      1.00      1.00        22

    accuracy                           0.93      1177
   macro avg       0.93      0.90      0.91      1177
weighted avg       0.93      0.93      0.93      1177

Confusion Matrix (Training Set):
 [[159   1   0   1   7   3   0   0   0]
 [  0 137   0   0   1   1   9   0   0]
 [  0   0  15   7   1   1   4   0   0]
 [ 14   1   0 226   2   0   0   0   0]
 [  6   3   1   0  61   0   3   0   0]
 [  1   2   0   1   4 116   0   0   0]
 [  0   3   0   0   2   1 353   0   0]
 [  0   0   0   0   0   0   0   8   0]
```

```
 [  0   0   0   0   0   0   0   0  22]]

Test Set Accuracy: 0.7694915254237288
Classification Report (Test Set):
              precision    recall  f1-score   support

           1       0.69      0.69      0.69        48
           2       0.77      0.70      0.73        43
           3       0.75      0.75      0.75         4
           4       0.79      0.75      0.77        60
           5       0.77      0.71      0.74        24
           6       0.86      0.75      0.80        16
           7       0.78      0.88      0.83        94
           8       1.00      0.75      0.86         4
           9       0.50      0.50      0.50         2

    accuracy                           0.77       295
   macro avg       0.77      0.72      0.74       295
weighted avg       0.77      0.77      0.77       295

Confusion Matrix (Test Set):
 [[33  0  1 10  1  0  3  0  0]
 [ 1 30  0  0  0  0 12  0  0]
 [ 0  0  3  1  0  0  0  0  0]
 [ 8  0  0 45  1  1  5  0  0]
 [ 4  0  0  0 17  1  2  0  0]
 [ 0  1  0  1  1 12  1  0  0]
 [ 1  7  0  0  2  0 83  0  1]
 [ 0  1  0  0  0  0  0  3  0]
 [ 1  0  0  0  0  0  0  0  1]]
```

In [74]:

```
from sklearn.linear_model import LogisticRegression
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train_tfidf, y_train)
```

Out[74]:

```
▾ LogisticRegression
LogisticRegression(max_iter=1000)
```

In [75]:

```
y_pred_train = logreg_model.predict(X_train_tfidf)
y_pred_test = logreg_model.predict(X_test_tfidf)
```

In [76]:

```
print("Training Set Accuracy:", accuracy_score(y_train, y_pred_train))
print("Test Set Accuracy:", accuracy_score(y_test, y_pred_test))
```

```
Training Set Accuracy: 0.7187765505522515
Test Set Accuracy: 0.6610169491525424
```

In [79]:

```
print("Training Set F1_score:", f1_score(y_train, y_pred_train, average='weighted'))
print("Test Set F1_score:", f1_score(y_test, y_pred_test, average='weighted'))
```

```
Training Set F1_score: 0.6944706612130505
Test Set F1_score: 0.6304002073396894
```

In [ ]:

```
```