Medical Text Transcription Classification ML Model with XAI Integration

July 17, 2024

[1]:

```python
# Import necessary libraries
import os  # For interacting with the operating system
import re  # For regular expressions
import warnings  # For managing warnings
import random  # For generating random numbers

import numpy as np  # For numerical operations
import pandas as pd  # For data manipulation and analysis
import matplotlib  # For plotting and visualization
import matplotlib.pyplot as plt  # For plotting and visualization
import seaborn as sns  # For statistical data visualization

from wordcloud import WordCloud  # For generating word clouds

# Natural Language Toolkit (NLTK) libraries for text processing
from nltk.tokenize import WhitespaceTokenizer, word_tokenize  # For tokenizing text
from nltk.stem import WordNetLemmatizer  # For lemmatizing text
from nltk.corpus import stopwords  # For accessing stop words

# Scikit-learn libraries for machine learning and data processing
from sklearn import preprocessing  # For preprocessing data
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer  # For text vectorization
from sklearn.decomposition import PCA, TruncatedSVD  # For dimensionality reduction
from sklearn.ensemble import RandomForestClassifier  # For random forest classification
from sklearn.svm import SVC  # For support vector classification
from xgboost import XGBClassifier  # For XGBoost classification
from sklearn.neighbors import KNeighborsClassifier  # For k-nearest neighbors classification
from sklearn.linear_model import LogisticRegression  # For logistic regression
from sklearn.tree import DecisionTreeClassifier  # For decision tree classification
from sklearn.metrics import classification_report, confusion_matrix, roc_curve,␣
 ↪auc, roc_auc_score  # For model evaluation metrics
from sklearn.preprocessing import LabelEncoder, StandardScaler, label_binarize ␣
 ↪# For preprocessing data
from sklearn.model_selection import train_test_split, KFold  # For splitting␣
 ↪data and cross-validation

# Imbalanced-learn library for handling imbalanced datasets
from imblearn.over_sampling import SMOTE  # For synthetic minority␣
 ↪over-sampling technique

# PyTorch libraries for deep learning
import torch  # For PyTorch operations
from torch.utils.data import TensorDataset, DataLoader, RandomSampler,␣
 ↪SequentialSampler  # For PyTorch data handling
from transformers import BertTokenizer, BertForSequenceClassification,␣
 ↪RobertaTokenizer, RobertaForSequenceClassification, AdamW,␣
 ↪get_linear_schedule_with_warmup  # For using BERT, BioBERT, ClinicalBERT and␣
 ↪RoBERTa models
```

[2]:

```
# Mount Google Drive to access files
from google.colab import drive
drive.mount('/content/drive')

# Read a CSV file from the specified path in Google Drive
df = pd.read_csv("/content/drive/My Drive/dataset/mtsamples.csv")

# Display the first few rows of the DataFrame
df.head()
```

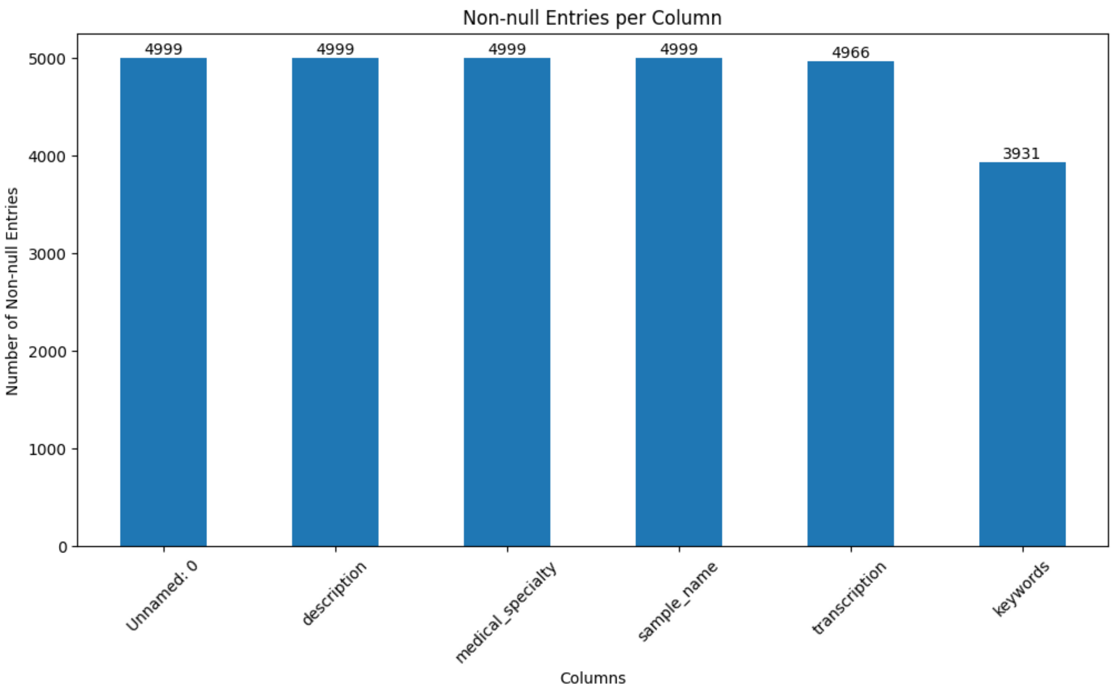
```
Mounted at /content/drive
```

[2]:

```
   Unnamed: 0                                        description  \
0           0   A 23-year-old white female presents with comp…
1           1           Consult for laparoscopic gastric bypass.
2           2           Consult for laparoscopic gastric bypass.
3           3                             2-D M-Mode. Doppler.
4           4                                 2-D Echocardiogram

            medical_specialty                                sample_name  \
0        Allergy / Immunology                         Allergic Rhinitis
1                  Bariatrics   Laparoscopic Gastric Bypass Consult - 2
2                  Bariatrics   Laparoscopic Gastric Bypass Consult - 1
3  Cardiovascular / Pulmonary                    2-D Echocardiogram - 1
4  Cardiovascular / Pulmonary                    2-D Echocardiogram - 2

                                       transcription  \
0  SUBJECTIVE:,  This 23-year-old white female pr…
1  PAST MEDICAL HISTORY:, He has difficulty climb…
2  HISTORY OF PRESENT ILLNESS: , I have seen ABC …
3  2-D M-MODE: , ,1.  Left atrial enlargement wit…
4  1.  The left ventricular cavity size and wall …

                                            keywords
0  allergy / immunology, allergic rhinitis, aller…
1  bariatrics, laparoscopic gastric bypass, weigh…
2  bariatrics, laparoscopic gastric bypass, heart…
3  cardiovascular / pulmonary, 2-d m-mode, dopple…
4  cardiovascular / pulmonary, 2-d, doppler, echo…
```

[3]:

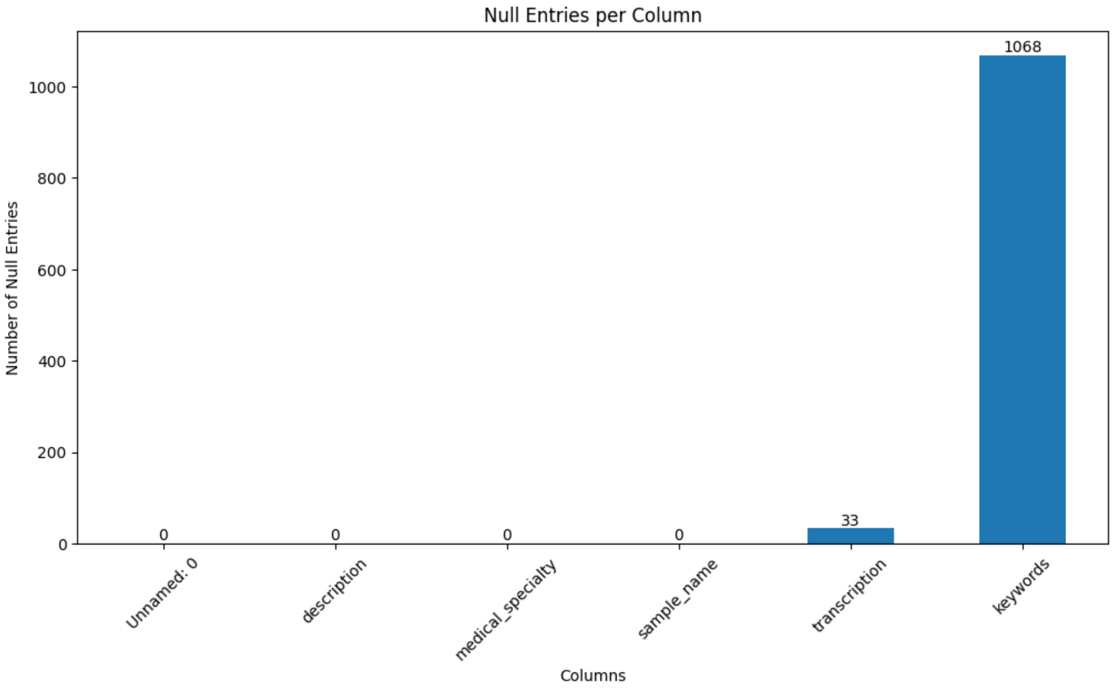
```python
# Calculate the count of non-null entries for each column in the dataframe
non_null_counts = df.count()

# Set the size of the figure for the plot
plt.figure(figsize=(12, 6))

# Create a bar plot of the non-null counts
ax = non_null_counts.plot(kind='bar')

# Set the title of the plot
plt.title('Non-null Entries per Column')

# Label the x-axis
plt.xlabel('Columns')

# Label the y-axis
plt.ylabel('Number of Non-null Entries')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Add labels to each bar in the plot
for i in ax.containers:
    ax.bar_label(i)

# Display the plot
plt.show()
```



[4]:

```python
# Calculate the number of null entries in each column of the dataframe
null_counts = df.isnull().sum()

# Set the figure size for the plot
plt.figure(figsize=(12, 6))

# Create a bar plot of the null counts per column
ax = null_counts.plot(kind='bar')

# Set the title of the plot
plt.title('Null Entries per Column')

# Set the label for the x-axis
plt.xlabel('Columns')

# Set the label for the y-axis
plt.ylabel('Number of Null Entries')

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Add labels to the bars in the plot
for i in ax.containers:
    ax.bar_label(i)
```

```
# Display the plot
plt.show()
```



```
[5]: def trim(df):
         """
         Cleans the DataFrame by performing the following operations:
         - Strips whitespace from column names
         - Drops duplicate rows
         - Converts column names to lowercase
         - Replaces spaces in column names with underscores
         - Strips whitespace from categorical data (object dtype columns)
         """
         # Strip whitespace from column names
         df.columns = df.columns.str.strip()
         # Drop duplicate rows
         df = df.drop_duplicates()
         # Convert column names to lowercase
         df.columns = df.columns.str.lower()
         # Replace spaces in column names with underscores
         df.columns = df.columns.str.replace(' ', '_')

         # Select columns with object dtype (categorical data)
         df_obj = df.select_dtypes(['object'])
         # Strip whitespace from categorical data
```

```python
    df[df_obj.columns] = df_obj.apply(lambda x: x.str.strip())

    # Print status messages
    print("All column names have been stripped, converted to lowercase, and spaces replaced with underscores.")
    print("Duplicate rows have been dropped.")
    print("Whitespace has been stripped from categorical data.")
    return df

# Set pandas option to display max column width
pd.set_option('display.max_colwidth', 255)

# Drop the 'Unnamed: 0' column from the DataFrame
df.drop('Unnamed: 0', axis=1, inplace=True)

# Apply the trim function to clean the DataFrame
df = trim(df)

def vc(df, column, r=False):
    """
    Computes value counts and percentages for a specified column.

    Parameters:
    df (pd.DataFrame): The DataFrame to analyze.
    column (str): The column name to compute value counts for.
    r (bool): If True, returns the resulting DataFrame. If False, prints and displays it.

    Returns:
    pd.DataFrame: Value counts and percentages if r is True.
    """
    # Compute value counts and convert to DataFrame
    vc_df = df.reset_index().groupby([column]).size().to_frame('count')
    # Compute percentages
    vc_df['percentage (%)'] = vc_df['count'].div(sum(vc_df['count'])).mul(100)
    # Sort by percentage in descending order
    vc_df = vc_df.sort_values(by=['percentage (%)'], ascending=False)

    if r:
        return vc_df
    else:
        # Print status message and display DataFrame
        print(f'STATUS: Value counts of "{column}"...')
        display(vc_df)

def shape(df, df_name):
    """
    Prints the dimensions (number of rows and columns) of the DataFrame.

    Parameters:
    df (pd.DataFrame): The DataFrame whose dimensions are to be printed.
    df_name (str): The name of the DataFrame to be printed in the message.
    """
    # Print dimensions of the DataFrame
    print(f'STATUS: Dimension of "{df_name}" = {df.shape}')

# Display the first 3 rows of the DataFrame
df.head(3)
```

```
All column names have been stripped, converted to lowercase, and spaces replaced
with underscores.
Duplicate rows have been dropped.
Whitespace has been stripped from categorical data.
```

```
[5]:                                                   description  \
0  A 23-year-old white female presents with complaint of allergies.
1                         Consult for laparoscopic gastric bypass.
2                         Consult for laparoscopic gastric bypass.

      medical_specialty                                 sample_name  \
0  Allergy / Immunology                           Allergic Rhinitis
1            Bariatrics  Laparoscopic Gastric Bypass Consult - 2
2            Bariatrics  Laparoscopic Gastric Bypass Consult - 1

    transcription  \
0  SUBJECTIVE:,  This 23-year-old white female presents with complaint of
allergies.  She used to have allergies when she lived in Seattle but she thinks
they are worse here.  In the past, she has tried Claritin, and Zyrtec.  Both
worked for short time b…
1  PAST MEDICAL HISTORY:, He has difficulty climbing stairs, difficulty with
airline seats, tying shoes, used to public seating, and lifting objects off the
floor.  He exercises three times a week at home and does cardio.  He has
difficulty walking two b…
2  HISTORY OF PRESENT ILLNESS: , I have seen ABC today.  He is a very pleasant
gentleman who is 42 years old, 344 pounds.  He is 5'9".  He has a BMI of 51.  He
has been overweight for ten years since the age of 33, at his highest he was 358
pounds, at hi…

                                                                          keywords
0
allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,
1     bariatrics, laparoscopic gastric bypass, weight loss programs, gastric
bypass, atkin's diet, weight watcher's, body weight, laparoscopic gastric,
weight loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lost
2  bariatrics, laparoscopic gastric bypass, heart attacks, body weight,
pulmonary embolism, potential complications, sleep study, weight loss, gastric
bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart, pounds,
weight,
```

[6]:

```python
# Print the dimensions of the dataframe (number of rows and columns)
print(df.shape)
```

```
(4999, 5)
```

[8]:

```python
# Select specific columns from the DataFrame
df = df[['transcription', 'keywords', 'medical_specialty']]

# Display the first few rows of the DataFrame to verify the selection
df.head()
```

[8]:

```
        transcription  \
0  SUBJECTIVE:,  This 23-year-old white female presents with complaint of
allergies.  She used to have allergies when she lived in Seattle but she thinks
they are worse here.  In the past, she has tried Claritin, and Zyrtec.  Both
worked for short time b…
1  PAST MEDICAL HISTORY:, He has difficulty climbing stairs, difficulty with
airline seats, tying shoes, used to public seating, and lifting objects off the
floor.  He exercises three times a week at home and does cardio.  He has
difficulty walking two b…
2  HISTORY OF PRESENT ILLNESS: , I have seen ABC today.  He is a very pleasant
gentleman who is 42 years old, 344 pounds.  He is 5'9".  He has a BMI of 51.  He
has been overweight for ten years since the age of 33, at his highest he was 358
pounds, at hi…
3  2-D M-MODE: , ,1.  Left atrial enlargement with left atrial diameter of 4.7
cm.,2.  Normal size right and left ventricle.,3.  Normal LV systolic function
with left ventricular ejection fraction of 51%.,4.  Normal LV diastolic
function.,5.  No pericard…
4  1.  The left ventricular cavity size and wall thickness appear normal.  The
wall motion and left ventricular systolic function appears hyperdynamic with
estimated ejection fraction of 70% to 75%.  There is near-cavity obliteration
seen.  There also ap…

             keywords  \
0
allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,
1                          bariatrics, laparoscopic gastric bypass, weight loss
programs, gastric bypass, atkin's diet, weight watcher's, body weight,
laparoscopic gastric, weight loss, pounds, months, weight, laparoscopic, band,
loss, diets, overweight, lost
2                      bariatrics, laparoscopic gastric bypass, heart attacks,
body weight, pulmonary embolism, potential complications, sleep study, weight
loss, gastric bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass,
heart, pounds, weight,
3  cardiovascular / pulmonary, 2-d m-mode, doppler, aortic valve, atrial
enlargement, diastolic function, ejection fraction, mitral, mitral valve,
pericardial effusion, pulmonary valve, regurgitation, systolic function,
tricuspid, tricuspid valve, normal lv
4  cardiovascular / pulmonary, 2-d, doppler, echocardiogram, annular, aortic
root, aortic valve, atrial, atrium, calcification, cavity, ejection fraction,
mitral, obliteration, outflow, regurgitation, relaxation pattern, stenosis,
systolic function, tric…

            medical_specialty
0        Allergy / Immunology
1                  Bariatrics
2                  Bariatrics
3  Cardiovascular / Pulmonary
4  Cardiovascular / Pulmonary
```

##Exploratory Data Analysis

[9]:
```python
# Create a copy of the DataFrame to avoid modifying the original DataFrame
df_1 = df.copy()
```

[10]:
```python
# Calculate the counts of unique values in the 'medical_specialty' column of the DataFrame 'df_1'
value_counts = df_1['medical_specialty'].value_counts()

# Print the resulting counts
print(value_counts)
```

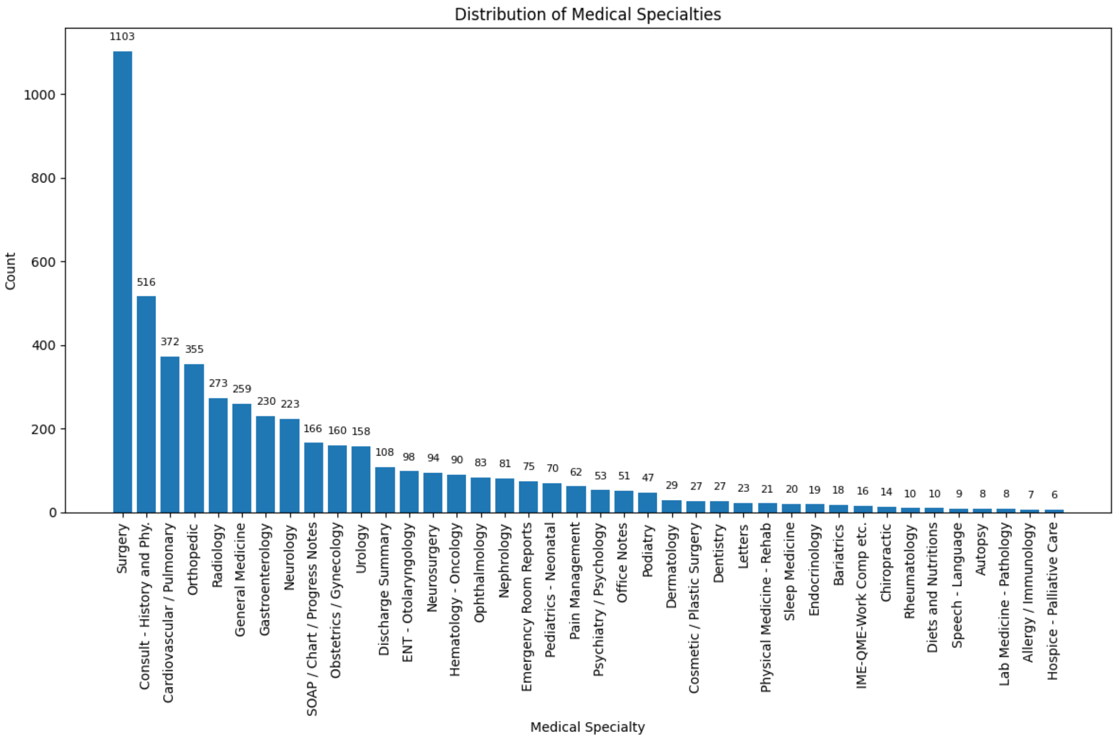
```
medical_specialty
Surgery                          1103
Consult - History and Phy.        516
Cardiovascular / Pulmonary        372
Orthopedic                        355
Radiology                         273
General Medicine                  259
Gastroenterology                  230
Neurology                         223
SOAP / Chart / Progress Notes     166
Obstetrics / Gynecology           160
Urology                           158
Discharge Summary                 108
ENT - Otolaryngology               98
Neurosurgery                       94
```

```
Hematology - Oncology              90
Ophthalmology                      83
Nephrology                         81
Emergency Room Reports             75
Pediatrics - Neonatal              70
Pain Management                    62
Psychiatry / Psychology            53
Office Notes                       51
Podiatry                           47
Dermatology                        29
Cosmetic / Plastic Surgery         27
Dentistry                          27
Letters                            23
Physical Medicine - Rehab          21
Sleep Medicine                     20
Endocrinology                      19
Bariatrics                         18
IME-QME-Work Comp etc.             16
Chiropractic                       14
Rheumatology                       10
Diets and Nutritions               10
Speech - Language                   9
Autopsy                             8
Lab Medicine - Pathology            8
Allergy / Immunology                7
Hospice - Palliative Care           6
Name: count, dtype: int64
```

#Analyze Categorical Variable

```
[11]: # Calculate the count of each unique value in the 'medical_specialty' column of␣
       ↪the dataframe df_1
      value_counts = df_1['medical_specialty'].value_counts()

      # Set the size of the plot
      plt.figure(figsize=(12, 8))

      # Create a bar plot with the index (medical specialties) on the x-axis and the␣
       ↪counts on the y-axis
      plt.bar(value_counts.index, value_counts)

      # Add a title to the plot
      plt.title('Distribution of Medical Specialties')

      # Label the x-axis
      plt.xlabel('Medical Specialty')

      # Label the y-axis
```

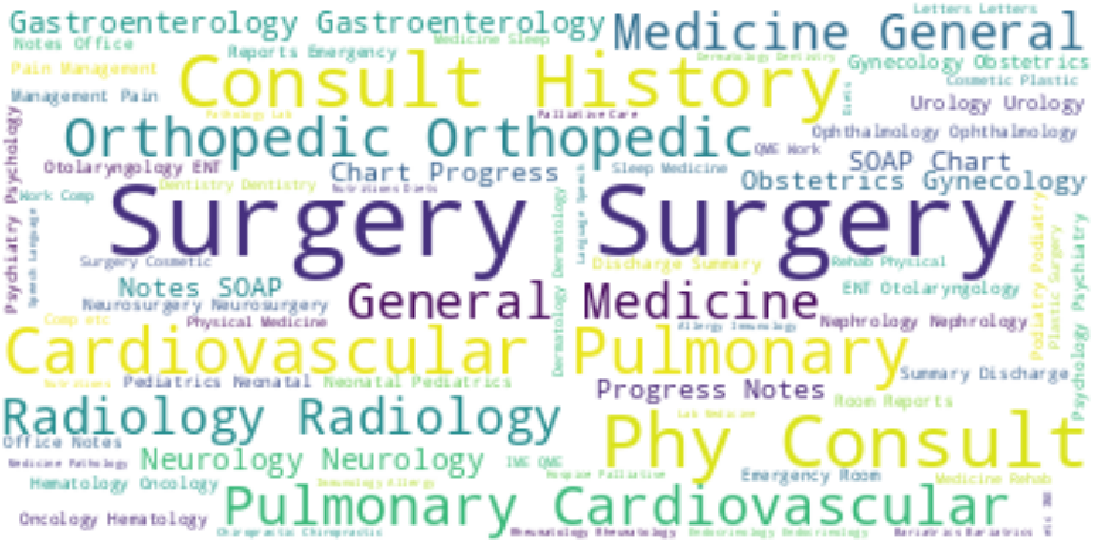
```
plt.ylabel('Count')

# Add the count labels on top of each bar
for i, count in enumerate(value_counts):
    # Position the text slightly above the bar
    plt.text(i, count + 0.02 * max(value_counts), str(count), ha='center',␣
 ↪va='bottom', fontsize=8)

# Rotate the x-axis labels by 90 degrees for better readability
plt.xticks(rotation=90)

# Adjust the plot to ensure everything fits without overlap
plt.tight_layout()

# Display the plot
plt.show()
```



Word cloud of the medical specialty column

[12]:
```
# Convert the 'medical_specialty' column in the dataframe to string type
df_1['medical_specialty'] = df_1['medical_specialty'].astype(str)
```

```
# Concatenate all values in the 'medical_specialty' column into a single string with spaces in between
text = ' '.join(df_1['medical_specialty'])

# Generate a word cloud from the concatenated text with a white background
wordcloud = WordCloud(background_color='white').generate(text)

# Set up the plot with a specific figure size
plt.figure(figsize=(10, 10))

# Display the word cloud image using bilinear interpolation
plt.imshow(wordcloud, interpolation='bilinear')

# Remove axis labels
plt.axis('off')

# Show the plot
plt.show()
```



Total word count of transcription column by medical specialty

```
[13]: # Initialize empty lists to store medical specialties and their corresponding word counts
medical_specialty_list = []
word_count_list = []

# Loop through each unique medical specialty in the DataFrame 'df_1'
for medical_specialty in df_1['medical_specialty'].unique():
    # Filter the DataFrame to only include rows with the current medical␣
  ↪specialty
    df_filter = df_1.loc[df_1['medical_specialty'] == medical_specialty]
    # Calculate the total word count for the current specialty by summing the␣
  ↪word counts of all transcriptions
    word_count_temp = df_filter['transcription'].str.split().str.len().sum()
    # Append the current medical specialty and its word count to their␣
  ↪respective lists
    medical_specialty_list.append(medical_specialty)
    word_count_list.append(word_count_temp)

# Create a new DataFrame with the collected medical specialties and word counts
word_count_df = pd.DataFrame({'Medical Specialty': medical_specialty_list,␣
  ↪'Word Count': word_count_list})
# Ensure the 'Word Count' column is of integer type
word_count_df['Word Count'] = word_count_df['Word Count'].astype('int')
# Sort the DataFrame by word count in descending order
word_count_df = word_count_df.sort_values('Word Count', ascending=False)
# Reset the index of the DataFrame and drop the old index
word_count_df.reset_index(drop=True)
```

[13]:

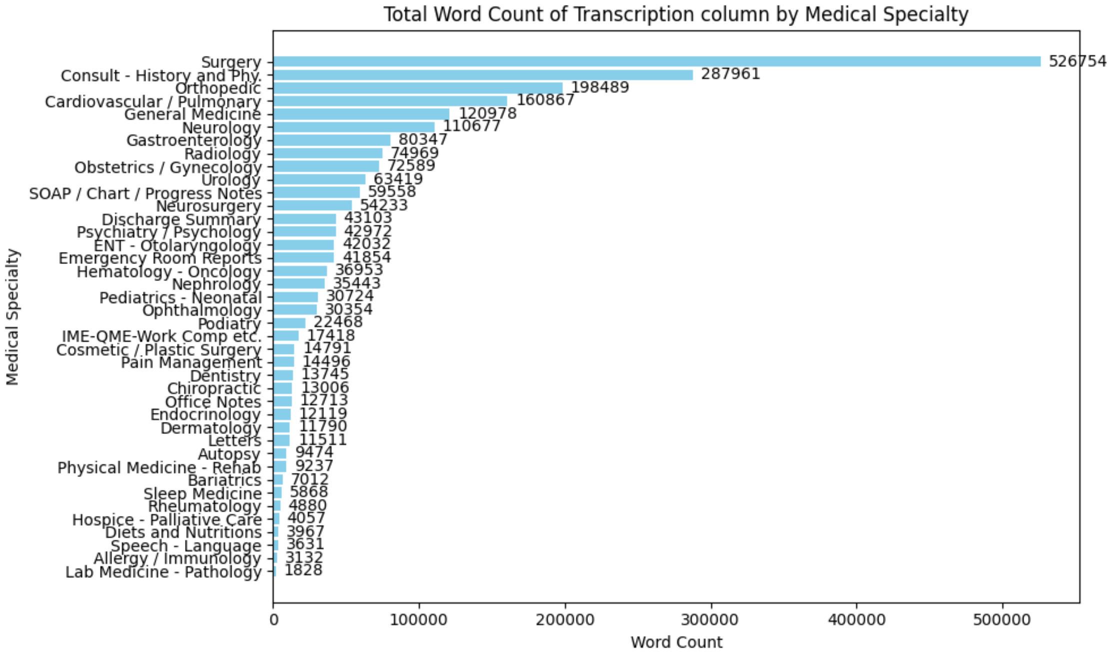
```
                Medical Specialty  Word Count
0                         Surgery      526754
1      Consult - History and Phy.      287961
2                      Orthopedic      198489
3      Cardiovascular / Pulmonary      160867
4                General Medicine      120978
5                       Neurology      110677
6                Gastroenterology       80347
7                       Radiology       74969
8         Obstetrics / Gynecology       72589
9                         Urology       63419
10  SOAP / Chart / Progress Notes       59558
11                   Neurosurgery       54233
12              Discharge Summary       43103
13        Psychiatry / Psychology       42972
14           ENT - Otolaryngology       42032
15         Emergency Room Reports       41854
16          Hematology - Oncology       36953
17                     Nephrology       35443
18          Pediatrics - Neonatal       30724
19                  Ophthalmology       30354
20                       Podiatry       22468
21         IME-QME-Work Comp etc.       17418
22     Cosmetic / Plastic Surgery       14791
23                Pain Management       14496
24                      Dentistry       13745
```

```
25                  Chiropractic       13006
26                  Office Notes       12713
27                 Endocrinology       12119
28                   Dermatology       11790
29                       Letters       11511
30                       Autopsy        9474
31      Physical Medicine - Rehab       9237
32                    Bariatrics        7012
33                Sleep Medicine        5868
34                  Rheumatology        4880
35      Hospice - Palliative Care       4057
36          Diets and Nutritions        3967
37             Speech - Language        3631
38          Allergy / Immunology        3132
39       Lab Medicine - Pathology       1828
```

[14]:

```
# Initialize empty lists to store medical specialties and their corresponding␣
 ↪word counts
medical_specialty_list = []
word_count_list = []

# Loop through each unique medical specialty in the DataFrame 'df_1'
for medical_specialty in df_1['medical_specialty'].unique():
    # Filter the DataFrame to only include rows with the current medical␣
 ↪specialty
    df_filter = df_1.loc[df_1['medical_specialty'] == medical_specialty]
    # Calculate the total word count for the current specialty by summing the␣
 ↪word counts of all transcriptions
    word_count_temp = df_filter['transcription'].str.split().str.len().sum()
    # Append the current medical specialty and its word count to their␣
 ↪respective lists
    medical_specialty_list.append(medical_specialty)
    word_count_list.append(word_count_temp)

# Create a new DataFrame with the collected medical specialties and word counts
word_count_df = pd.DataFrame({'Medical Specialty': medical_specialty_list,␣
 ↪'Word Count': word_count_list})
# Ensure the 'Word Count' column is of integer type
word_count_df['Word Count'] = word_count_df['Word Count'].astype('int')
# Sort the DataFrame by word count in descending order
word_count_df = word_count_df.sort_values('Word Count', ascending=False)
# Reset the index of the DataFrame and drop the old index
word_count_df = word_count_df.reset_index(drop=True)

# Import matplotlib for plotting
import matplotlib.pyplot as plt
```

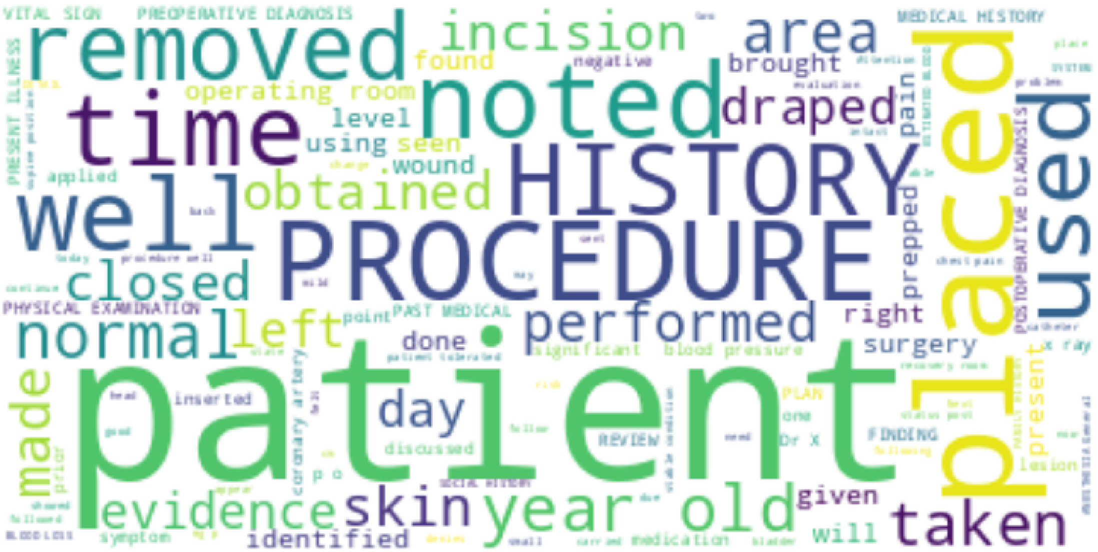
```
# Set the figure size for the plot
plt.figure(figsize=(10, 6))

# Create a horizontal bar plot
bars = plt.barh(y=word_count_df['Medical Specialty'], width=word_count_df['Word␣
 ↪Count'], color='skyblue')

# Add word counts as text on the bars
for bar in bars:
    width = bar.get_width()
    plt.annotate(f'{int(width)}',
                 xy=(width, bar.get_y() + bar.get_height() / 2),
                 xytext=(5, 0),
                 textcoords='offset points',
                 ha='left',
                 va='center')

# Label the axes and title the plot
plt.xlabel('Word Count')
plt.ylabel('Medical Specialty')
plt.title('Total Word Count of Transcription column by Medical Specialty')

# Invert y-axis to have the specialty with the highest word count at the top
plt.gca().invert_yaxis()

# Adjust spacing between subplots
plt.tight_layout()

# Display the plot
plt.show()
```



Word cloud of the transcription column

[15]:

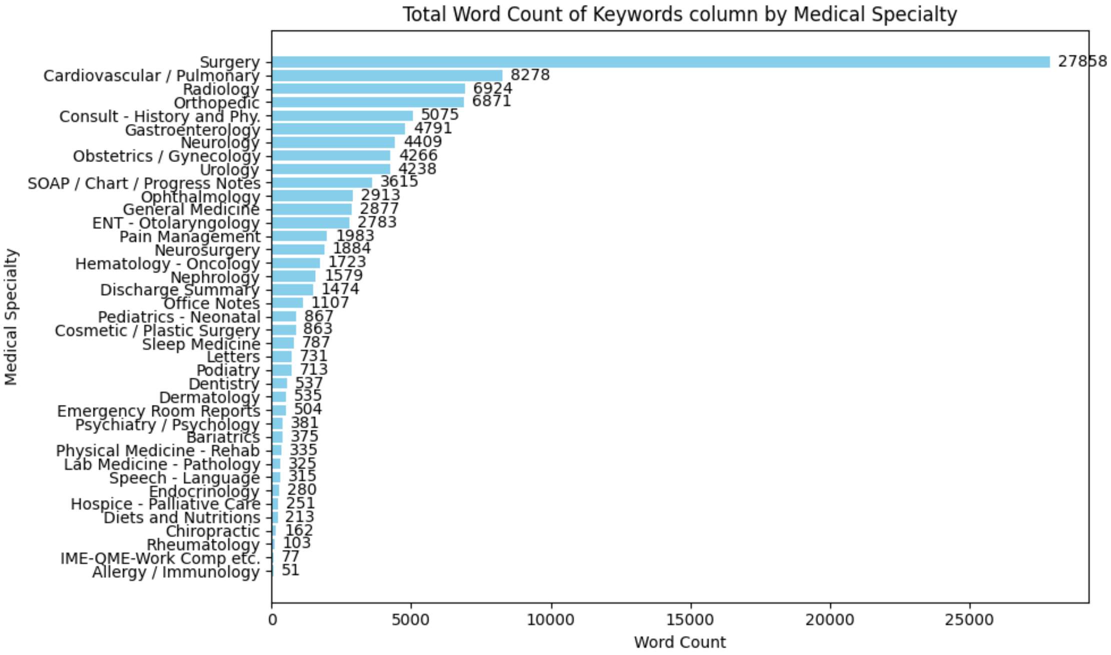
```
# Convert the 'transcription' column to string type to ensure compatibility for␣
 ↪text processing
df_1['transcription'] = df_1['transcription'].astype(str)

# Concatenate all the strings in the 'transcription' column into a single text
text = ' '.join(df_1['transcription'])

# Generate a word cloud with the specified background color using the␣
 ↪concatenated text
wordcloud = WordCloud(background_color='white').generate(text)

# Plot the word cloud
plt.figure(figsize=(10, 10))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



Total Word Count of keywords column by Medical Specialty

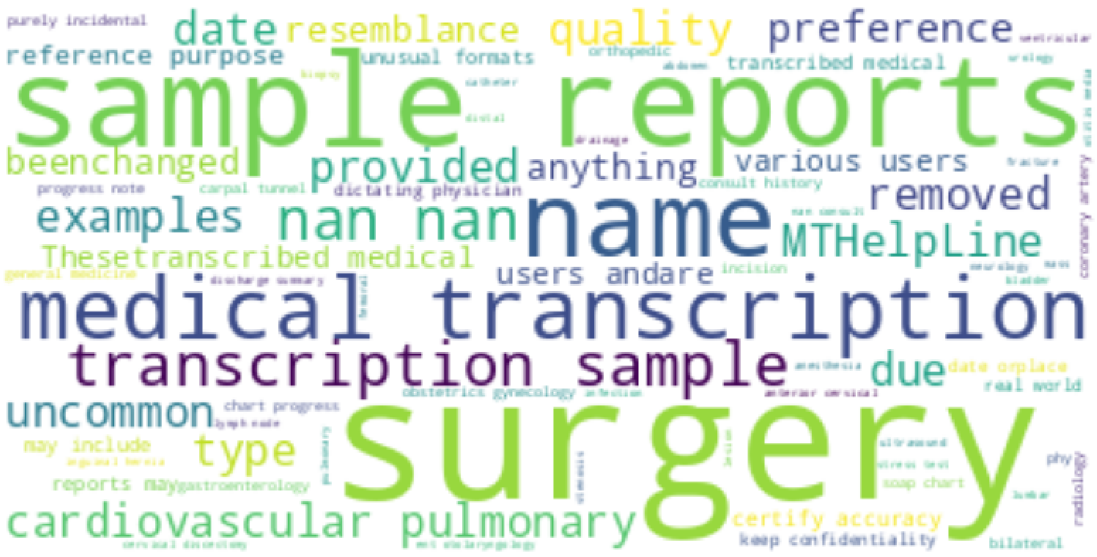
```
[16]: # Initialize empty lists to store medical specialties and corresponding word␣
       ↪counts
      medical_specialty_list = []
      word_count_list = []

      # Drop rows where 'keywords' column is NaN
      df_filtered = df_1.dropna(subset=['keywords'])

      # Iterate over unique medical specialties in the filtered DataFrame
      for medical_specialty in df_filtered['medical_specialty'].unique():
          # Calculate word count based on medical specialty
          if medical_specialty == 'autopsy':
              word_count_temp = df_filtered.loc[df_filtered['medical_specialty'] ==␣
       ↪medical_specialty, 'keywords'].str.split().str.len().sum()
          else:
              df_filter = df_filtered[df_filtered['medical_specialty'] ==␣
       ↪medical_specialty]
              word_count_temp = df_filter['keywords'].str.split().str.len().sum()

          # Append medical specialty and corresponding word count to lists
          medical_specialty_list.append(medical_specialty)
          word_count_list.append(word_count_temp)

      # Create a DataFrame from the lists of medical specialties and word counts
      word_count_df = pd.DataFrame({'Medical Specialty': medical_specialty_list,␣
       ↪'Word Count': word_count_list})
```

```
# Convert 'Word Count' column to integer type
word_count_df['Word Count'] = word_count_df['Word Count'].astype('int')

# Sort DataFrame by 'Word Count' in descending order
word_count_df = word_count_df.sort_values('Word Count', ascending=False)

# Reset index of the sorted DataFrame
word_count_df = word_count_df.reset_index(drop=True)

# Create a horizontal bar plot
plt.figure(figsize=(10, 6))
bars = plt.barh(y=word_count_df['Medical Specialty'], width=word_count_df['Word Count'], color='skyblue')

# Annotate each bar with its value
for bar in bars:
    width = bar.get_width()
    plt.annotate(f'{int(width)}',
                 xy=(width, bar.get_y() + bar.get_height() / 2),
                 xytext=(5, 0),
                 textcoords='offset points',
                 ha='left',
                 va='center')

# Add labels and title to the plot
plt.xlabel('Word Count')
plt.ylabel('Medical Specialty')
plt.title('Total Word Count of Keywords column by Medical Specialty')

# Invert y-axis to display highest word count at the top
plt.gca().invert_yaxis()

# Adjust layout to prevent cropping of labels
plt.tight_layout()

# Display the plot
plt.show()
```



Word cloud of the keywords column

[17]:
```
# Convert the 'keywords' column in df_1 to string type
df_1['keywords'] = df_1['keywords'].astype(str)

# Join all values in the 'keywords' column into a single string separated by␣
 ↪spaces
text = ' '.join(df_1['keywords'])

# Generate a WordCloud object with white background
wordcloud = WordCloud(background_color='white').generate(text)

# Plot the WordCloud
plt.figure(figsize=(10, 10))  # Set the size of the plot
plt.imshow(wordcloud, interpolation='bilinear')  # Display the WordCloud with␣
 ↪bilinear interpolation
plt.axis('off')  # Turn off axis labels
plt.show()  # Display the plot
```



##Data Preprocessing

```
[18]: # Printing the shape of the DataFrame 'df'
      print(f'data shape is: {df.shape}')

      # Checking and counting the number of missing values (null values) in each column of the DataFrame 'df',
      # then sorting them in descending order based on the count of missing values.
      df.isnull().sum().sort_values(ascending=False)
```

```
data shape is: (4999, 3)
```

```
[18]: keywords            1068
      transcription         33
      medical_specialty      0
      dtype: int64
```

```
[19]: # Remove transcription rows that is empty
      df = df[df['transcription'].notna()]
```

```
[20]: # Fill missing values in the 'keywords' column with an empty string
      df['keywords'] = df['keywords'].fillna('')
```

```
<ipython-input-20-31feb3f19d44>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
```

```
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['keywords'] = df['keywords'].fillna('')
```

[21]:
```
# Print the shape of the DataFrame 'df'
print(f'data shape is: {df.shape}')

# Check and display the count of null values in each column of 'df', sorted in␣
 ↪descending order
df.isnull().sum().sort_values(ascending=False)
```

```
data shape is: (4966, 3)
```

[21]:
```
transcription        0
keywords             0
medical_specialty    0
dtype: int64
```

#Concatenation and Subsetting

[22]:
```
# Concatenate 'keywords' and 'transcription' columns and store the result in a␣
 ↪new 'transcription' column
df['transcription'] = df['keywords'] + df['transcription']
df.head()
```

```
<ipython-input-22-625e380a33d0>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['transcription'] = df['keywords'] + df['transcription']
```

[22]:
```
      transcription  \
0  allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,SUBJECTIVE:,  This
23-year-old white female presents with complaint of allergies.  She used to have
allergies when she l…
1  bariatrics, laparoscopic gastric bypass, weight loss programs, gastric
bypass, atkin's diet, weight watcher's, body weight, laparoscopic gastric,
weight loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostPAST MEDICAL HISTOR…
2  bariatrics, laparoscopic gastric bypass, heart attacks, body weight,
pulmonary embolism, potential complications, sleep study, weight loss, gastric
bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart, pounds,
weight,HISTORY OF PRESE…
3  cardiovascular / pulmonary, 2-d m-mode, doppler, aortic valve, atrial
enlargement, diastolic function, ejection fraction, mitral, mitral valve,
pericardial effusion, pulmonary valve, regurgitation, systolic function,
tricuspid, tricuspid valve, normal…
4  cardiovascular / pulmonary, 2-d, doppler, echocardiogram, annular, aortic
root, aortic valve, atrial, atrium, calcification, cavity, ejection fraction,
mitral, obliteration, outflow, regurgitation, relaxation pattern, stenosis,
systolic function, tric…

         keywords  \
0
allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,
1                        bariatrics, laparoscopic gastric bypass, weight loss
programs, gastric bypass, atkin's diet, weight watcher's, body weight,
laparoscopic gastric, weight loss, pounds, months, weight, laparoscopic, band,
loss, diets, overweight, lost
2                     bariatrics, laparoscopic gastric bypass, heart attacks,
body weight, pulmonary embolism, potential complications, sleep study, weight
loss, gastric bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass,
heart, pounds, weight,
3  cardiovascular / pulmonary, 2-d m-mode, doppler, aortic valve, atrial
enlargement, diastolic function, ejection fraction, mitral, mitral valve,
pericardial effusion, pulmonary valve, regurgitation, systolic function,
tricuspid, tricuspid valve, normal lv
4  cardiovascular / pulmonary, 2-d, doppler, echocardiogram, annular, aortic
root, aortic valve, atrial, atrium, calcification, cavity, ejection fraction,
mitral, obliteration, outflow, regurgitation, relaxation pattern, stenosis,
systolic function, tric…

            medical_specialty
0        Allergy / Immunology
1                  Bariatrics
2                  Bariatrics
3  Cardiovascular / Pulmonary
4  Cardiovascular / Pulmonary
```

[23]:

```python
# Select only the 'transcription' and 'medical_specialty' columns from the DataFrame 'df'
df = df[['transcription', 'medical_specialty']]
df.head()
```

[23]:

```
    transcription  \
0  allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,SUBJECTIVE:,  This
23-year-old white female presents with complaint of allergies.  She used to have
allergies when she l…
1  bariatrics, laparoscopic gastric bypass, weight loss programs, gastric
bypass, atkin's diet, weight watcher's, body weight, laparoscopic gastric,
weight loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostPAST MEDICAL HISTOR…
2  bariatrics, laparoscopic gastric bypass, heart attacks, body weight,
pulmonary embolism, potential complications, sleep study, weight loss, gastric
bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart, pounds,
weight,HISTORY OF PRESE…
3  cardiovascular / pulmonary, 2-d m-mode, doppler, aortic valve, atrial
enlargement, diastolic function, ejection fraction, mitral, mitral valve,
pericardial effusion, pulmonary valve, regurgitation, systolic function,
tricuspid, tricuspid valve, normal…
4  cardiovascular / pulmonary, 2-d, doppler, echocardiogram, annular, aortic
root, aortic valve, atrial, atrium, calcification, cavity, ejection fraction,
mitral, obliteration, outflow, regurgitation, relaxation pattern, stenosis,
systolic function, tric…

            medical_specialty
0        Allergy / Immunology
1                 Bariatrics
2                 Bariatrics
3  Cardiovascular / Pulmonary
4  Cardiovascular / Pulmonary
```

[24]:

```
# Print the shape of the DataFrame 'df'
print(f'data shape is: {df.shape}')

# Check and display the count of null values in each column of 'df', sorted in␣
 ↪descending order
df.isnull().sum().sort_values(ascending=False)
```

```
data shape is: (4966, 2)
```

[24]:

```
transcription        0
medical_specialty    0
dtype: int64
```

Word count of the transcription column by medical specialty after concatenation

[25]:

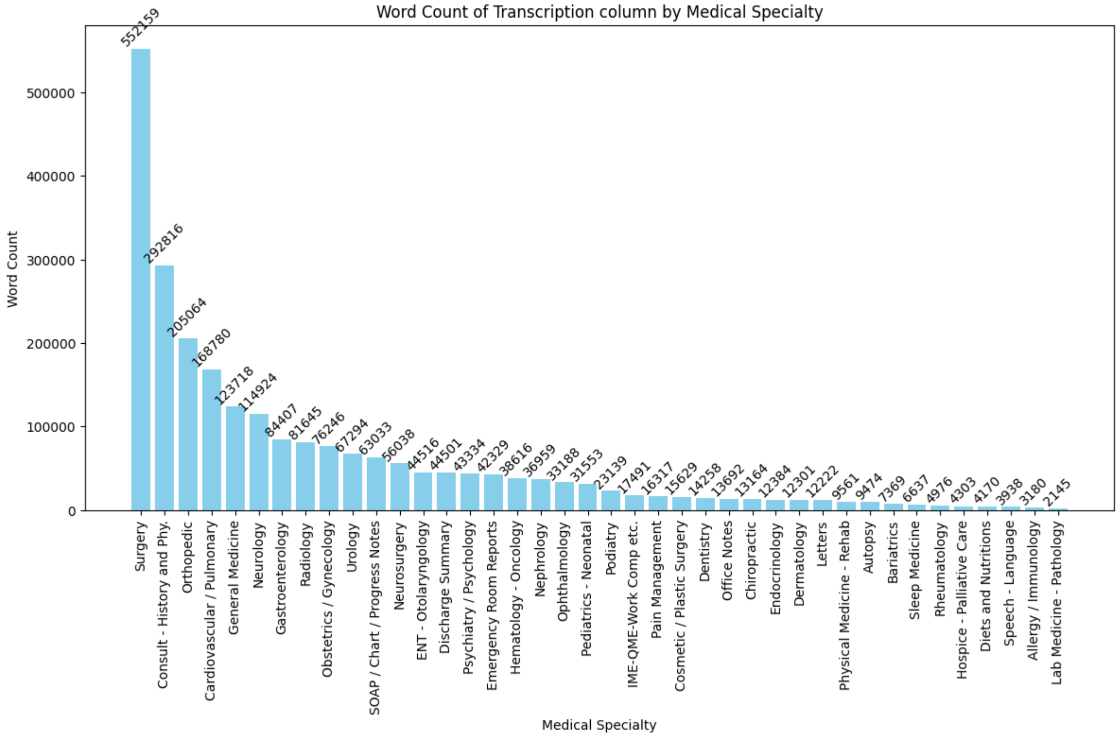
```
# Initialize lists to store medical specialties and their corresponding word␣
 ↪counts
medical_specialty_list = []
word_count_list = []

# Iterate over unique values in 'medical_specialty' column
for medical_specialty in df['medical_specialty'].unique():
    # Filter DataFrame for the current medical specialty
    df_filter = df.loc[df['medical_specialty'] == medical_specialty]
    # Calculate total word count for transcriptions in the current medical␣
 ↪specialty
    word_count_temp = df_filter['transcription'].str.split().str.len().sum()

    # Append medical specialty and its word count to lists
    medical_specialty_list.append(medical_specialty)
    word_count_list.append(word_count_temp)

# Create a DataFrame from the lists
word_count_df = pd.DataFrame({'Medical Specialty': medical_specialty_list,␣
 ↪'Word Count': word_count_list})

# Convert 'Word Count' to integer type
word_count_df['Word Count'] = word_count_df['Word Count'].astype('int')

# Sort DataFrame by 'Word Count' in descending order
word_count_df = word_count_df.sort_values('Word Count', ascending=False)

# Reset index for the DataFrame
word_count_df = word_count_df.reset_index(drop=True)

# Display the resulting DataFrame (note: printing the result is not shown in␣
 ↪the original code)
word_count_df
```

```
[25]:
                Medical Specialty  Word Count
0                         Surgery      552159
1      Consult - History and Phy.      292816
2                      Orthopedic      205064
3      Cardiovascular / Pulmonary      168780
4                General Medicine      123718
5                       Neurology      114924
6                Gastroenterology       84407
7                       Radiology       81645
8         Obstetrics / Gynecology       76246
9                         Urology       67294
10  SOAP / Chart / Progress Notes       63033
11                   Neurosurgery       56038
12           ENT - Otolaryngology       44516
13              Discharge Summary       44501
14        Psychiatry / Psychology       43334
15         Emergency Room Reports       42329
16          Hematology - Oncology       38616
17                     Nephrology       36959
18                  Ophthalmology       33188
19          Pediatrics - Neonatal       31553
20                       Podiatry       23139
```

```
21         IME-QME-Work Comp etc.       17491
22                Pain Management       16317
23     Cosmetic / Plastic Surgery       15629
24                      Dentistry       14258
25                   Office Notes       13692
26                   Chiropractic       13164
27                  Endocrinology       12384
28                    Dermatology       12301
29                        Letters       12222
30      Physical Medicine - Rehab        9561
31                        Autopsy        9474
32                     Bariatrics        7369
33                 Sleep Medicine        6637
34                   Rheumatology        4976
35      Hospice - Palliative Care        4303
36           Diets and Nutritions        4170
37              Speech - Language        3938
38           Allergy / Immunology        3180
39       Lab Medicine - Pathology        2145
```

[26]:

```python
# Plotting the bar graph
plt.figure(figsize=(12, 8))  # Set the figure size
bars = plt.bar(word_count_df['Medical Specialty'], word_count_df['Word Count'],␣
 ↪color='skyblue')  # Create bars for each medical specialty
plt.xlabel('Medical Specialty')  # Label for the x-axis
plt.ylabel('Word Count')  # Label for the y-axis
plt.title('Word Count of Transcription column by Medical Specialty')  # Title␣
 ↪of the plot
plt.xticks(rotation=90)  # Rotate x-axis labels for better visibility

# Adding the word count on top of each bar with rotation
for bar in bars:
    yval = bar.get_height()  # Get the height (word count) of each bar
    plt.text(
        bar.get_x() + bar.get_width()/2,  # x-coordinate for placing the text␣
 ↪at the center of the bar
        yval,  # y-coordinate for placing the text (same as bar height)
        int(yval),  # Text to display (converted to integer for neatness)
        ha='center',  # Horizontal alignment of the text
        va='bottom',  # Vertical alignment of the text
        rotation=45  # Rotate the text for better alignment
    )

plt.tight_layout()  # Adjust layout to prevent clipping of labels

# Show the plot
plt.show()
```



[27]:

```
# Calculate the total word count of all transcriptions before data preprocessing
total_word_count = df['transcription'].str.split().str.len().sum()

# Print the total word count
print(f'The word count of all transcriptions before data preprocessing is:␣
 ↪{int(total_word_count)}')
```

```
The word count of all transcriptions before data preprocessing is: 2407470
```

0.1 Converting to Lowercase

[28]:

```
def lower(df, attribute):
    # Apply lowercase conversion using lambda function to the specified␣
 ↪attribute column
    df[attribute] = df[attribute].apply(lambda x: str(x).lower() if pd.
 ↪notnull(x) else x)
    return df

# Example usage:
# Apply lowercase conversion to 'transcription' column of DataFrame 'df'
df = lower(df, 'transcription')

# Display the first 3 rows of the modified DataFrame
```

```
df.head(3)
```

```
[28]:     transcription  \
0  allergy / immunology, allergic rhinitis, allergies, asthma, nasal sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergic,subjective:,  this
23-year-old white female presents with complaint of allergies.  she used to have
allergies when she l…
1  bariatrics, laparoscopic gastric bypass, weight loss programs, gastric
bypass, atkin's diet, weight watcher's, body weight, laparoscopic gastric,
weight loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast medical histor…
2  bariatrics, laparoscopic gastric bypass, heart attacks, body weight,
pulmonary embolism, potential complications, sleep study, weight loss, gastric
bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart, pounds,
weight,history of prese…

      medical_specialty
0  Allergy / Immunology
1            Bariatrics
2            Bariatrics
```

##Removing Punctuation and Numbers

```
[29]: # Ignore warnings (optional)
warnings.filterwarnings('ignore')

def remove_punc_num(df, attribute):
    # Ensure the attribute column is treated as string
    df[attribute] = df[attribute].astype(str)

    # Remove standalone digits
    df[attribute] = df[attribute].apply(lambda x: re.sub(r'\b\d+\b', '', x))

    # Remove alphanumeric sequences with digits
    df[attribute] = df[attribute].apply(lambda x: re.sub(r'\w*\d\w*', '', x))

    # Remove punctuation
    df[attribute] = df[attribute].apply(lambda x: re.sub(r'[^\w\s]', '', x))

    # Remove extra spaces and convert to lowercase
    df[attribute] = df[attribute].apply(lambda x: " ".join(re.
 ↪findall(r'\b\w+\b', x.lower())))

    # Display the cleaned DataFrame
    print("Head of DataFrame after cleaning:")
    print(df.head(3))  # Display the first 3 rows of the DataFrame
    return df


# Call the function to clean the 'transcription' column of DataFrame 'df'
df_cleaned = remove_punc_num(df, 'transcription')
```

```
Head of DataFrame after cleaning:
    transcription  \
0  allergy immunology allergic rhinitis allergies asthma nasal sprays rhinitis
nasal erythematous allegra sprays allergicsubjective this yearold white female
presents with complaint of allergies she used to have allergies when she lived
in seattle but sh…
1  bariatrics laparoscopic gastric bypass weight loss programs gastric bypass
atkins diet weight watchers body weight laparoscopic gastric weight loss pounds
months weight laparoscopic band loss diets overweight lostpast medical history
he has difficulty…
2  bariatrics laparoscopic gastric bypass heart attacks body weight pulmonary
embolism potential complications sleep study weight loss gastric bypass
anastomosis loss sleep laparoscopic gastric bypass heart pounds weighthistory of
present illness i have …

      medical_specialty
0  Allergy / Immunology
1            Bariatrics
2            Bariatrics
```

#Tokenization

[30]:
```
# Initialize WhitespaceTokenizer
tk = WhitespaceTokenizer()

def tokenise(df, attribute):
    # Tokenize each row of the specified attribute column using␣
 ↪WhitespaceTokenizer
    df['tokenised'] = df.apply(lambda row: tk.tokenize(str(row[attribute])),␣
 ↪axis=1)
    return df


# Call the tokenise function to tokenize the 'transcription' column of␣
 ↪DataFrame 'df'
df = tokenise(df, 'transcription')

# Create a copy of the DataFrame for experimentation
df_experiment = df.copy()

# Display the first 3 rows of the DataFrame after tokenization
df.head(3)
```

[30]:
```
    transcription  \
0  allergy immunology allergic rhinitis allergies asthma nasal sprays rhinitis
nasal erythematous allegra sprays allergicsubjective this yearold white female
presents with complaint of allergies she used to have allergies when she lived
in seattle but sh…
1  bariatrics laparoscopic gastric bypass weight loss programs gastric bypass
atkins diet weight watchers body weight laparoscopic gastric weight loss pounds
months weight laparoscopic band loss diets overweight lostpast medical history
he has difficulty…
2  bariatrics laparoscopic gastric bypass heart attacks body weight pulmonary
embolism potential complications sleep study weight loss gastric bypass
anastomosis loss sleep laparoscopic gastric bypass heart pounds weighthistory of
present illness i have …

      medical_specialty  \
0  Allergy / Immunology
1            Bariatrics
2            Bariatrics

        tokenised
0  [allergy, immunology, allergic, rhinitis, allergies, asthma, nasal, sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergicsubjective, this,
yearold, white, female, presents, with, complaint, of, allergies, she, used, to,
have, allergies, whe…
1  [bariatrics, laparoscopic, gastric, bypass, weight, loss, programs, gastric,
bypass, atkins, diet, weight, watchers, body, weight, laparoscopic, gastric,
weight, loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast, medi…
2  [bariatrics, laparoscopic, gastric, bypass, heart, attacks, body, weight,
pulmonary, embolism, potential, complications, sleep, study, weight, loss,
gastric, bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart,
pounds, weighthistory…
```

#Stop Words Removal

[31]:
```python
import nltk

# Download NLTK stopwords dataset
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[31]:
```
True
```

[32]:
```python
# Showing the list of the English stop words, it has a number of 179 stop words␣
 ↪in this list
```

```
stop = stopwords.words('english')
print(f"There are {len(stop)} stop words \n")
print(stop)
```

```
There are 179 stop words

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

[33]:

```
# Function to remove stop words
def remove_stop_words(df, attribute):
    stop = set(stopwords.words('english'))  # Use set for faster lookup
    df['stop_words_removal'] = df[attribute].apply(lambda x: [word for word in␣
 ↪x if word.lower() not in stop])
    return df

# Remove stop words from the 'tokenised' column
df = remove_stop_words(df, 'tokenised')
df.head(2)
```

[33]:

```
    transcription  \
0  allergy immunology allergic rhinitis allergies asthma nasal sprays rhinitis
nasal erythematous allegra sprays allergicsubjective this yearold white female
presents with complaint of allergies she used to have allergies when she lived
in seattle but sh…
1  bariatrics laparoscopic gastric bypass weight loss programs gastric bypass
atkins diet weight watchers body weight laparoscopic gastric weight loss pounds
months weight laparoscopic band loss diets overweight lostpast medical history
he has difficulty…
```

```
      medical_specialty  \
0  Allergy / Immunology
1            Bariatrics

        tokenised  \
0  [allergy, immunology, allergic, rhinitis, allergies, asthma, nasal, sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergicsubjective, this,
yearold, white, female, presents, with, complaint, of, allergies, she, used, to,
have, allergies, whe…
1  [bariatrics, laparoscopic, gastric, bypass, weight, loss, programs, gastric,
bypass, atkins, diet, weight, watchers, body, weight, laparoscopic, gastric,
weight, loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast, medi…

stop_words_removal
0  [allergy, immunology, allergic, rhinitis, allergies, asthma, nasal, sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergicsubjective, yearold,
white, female, presents, complaint, allergies, used, allergies, lived, seattle,
thinks, worse, pas…
1  [bariatrics, laparoscopic, gastric, bypass, weight, loss, programs, gastric,
bypass, atkins, diet, weight, watchers, body, weight, laparoscopic, gastric,
weight, loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast, medi…
```

#Lemmatization

[34]:
```
import nltk

# Download NLTK datasets
nltk.download('punkt')
nltk.download('wordnet')

# Initialize WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

# Function to lemmatize text
def lemmatize_text(input_text):
    lemmatized_words = []
    words = word_tokenize(input_text)
    lemmatized_words.extend([wordnet_lemmatizer.lemmatize(word) for word in␣
 ↪words])
    return lemmatized_words


# Function to lemmatize a column of text
def lemmatize(df, input_column, output_column):
    df[output_column] = df[input_column].apply(lambda x: ', '.
 ↪join(lemmatize_text(' '.join(x))))
    return df


# Assuming 'df' is your DataFrame and 'stop_words_removal' is the column with␣
 ↪the text to be lemmatized
df = lemmatize(df, 'stop_words_removal', 'lemmatized_transcription')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
```

[35]:

```
df.head(2)
```

[35]:

```
     transcription  \
0  allergy immunology allergic rhinitis allergies asthma nasal sprays rhinitis
nasal erythematous allegra sprays allergicsubjective this yearold white female
presents with complaint of allergies she used to have allergies when she lived
in seattle but sh…
1  bariatrics laparoscopic gastric bypass weight loss programs gastric bypass
atkins diet weight watchers body weight laparoscopic gastric weight loss pounds
months weight laparoscopic band loss diets overweight lostpast medical history
he has difficulty…

      medical_specialty  \
0  Allergy / Immunology
1            Bariatrics

        tokenised  \
0  [allergy, immunology, allergic, rhinitis, allergies, asthma, nasal, sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergicsubjective, this,
yearold, white, female, presents, with, complaint, of, allergies, she, used, to,
have, allergies, whe…
1  [bariatrics, laparoscopic, gastric, bypass, weight, loss, programs, gastric,
bypass, atkins, diet, weight, watchers, body, weight, laparoscopic, gastric,
weight, loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast, medi…

stop_words_removal  \
0  [allergy, immunology, allergic, rhinitis, allergies, asthma, nasal, sprays,
rhinitis, nasal, erythematous, allegra, sprays, allergicsubjective, yearold,
white, female, presents, complaint, allergies, used, allergies, lived, seattle,
thinks, worse, pas…
1  [bariatrics, laparoscopic, gastric, bypass, weight, loss, programs, gastric,
bypass, atkins, diet, weight, watchers, body, weight, laparoscopic, gastric,
weight, loss, pounds, months, weight, laparoscopic, band, loss, diets,
overweight, lostpast, medi…

lemmatized_transcription
0  allergy, immunology, allergic, rhinitis, allergy, asthma, nasal, spray,
rhinitis, nasal, erythematous, allegra, spray, allergicsubjective, yearold,
white, female, present, complaint, allergy, used, allergy, lived, seattle,
think, worse, past, tried, c…
1  bariatrics, laparoscopic, gastric, bypass, weight, loss, program, gastric,
bypass, atkins, diet, weight, watcher, body, weight, laparoscopic, gastric,
weight, loss, pound, month, weight, laparoscopic, band, loss, diet, overweight,
lostpast, medical, h…
```

[36]:

```python
# Drop columns 'transcription', 'tokenised', 'stop_words_removal'
df = df.drop(['transcription', 'tokenised', 'stop_words_removal'], axis=1)

# Display the first 2 rows of the modified DataFrame
df.head(2)
```

[36]:

```
      medical_specialty  \
0  Allergy / Immunology
1            Bariatrics

lemmatized_transcription
0  allergy, immunology, allergic, rhinitis, allergy, asthma, nasal, spray,
rhinitis, nasal, erythematous, allegra, spray, allergicsubjective, yearold,
white, female, present, complaint, allergy, used, allergy, lived, seattle,
think, worse, past, tried, c…
1  bariatrics, laparoscopic, gastric, bypass, weight, loss, program, gastric,
bypass, atkins, diet, weight, watcher, body, weight, laparoscopic, gastric,
weight, loss, pound, month, weight, laparoscopic, band, loss, diet, overweight,
lostpast, medical, h…
```

[37]:

```python
# Rename the column 'lemmatized_transcription' to 'preprocessed_transcription'
df.rename(columns={'lemmatized_transcription': 'preprocessed_transcription'},␣
 ↪inplace=True)

# Display the first few rows of the DataFrame to verify the rename operation
df.head()
```

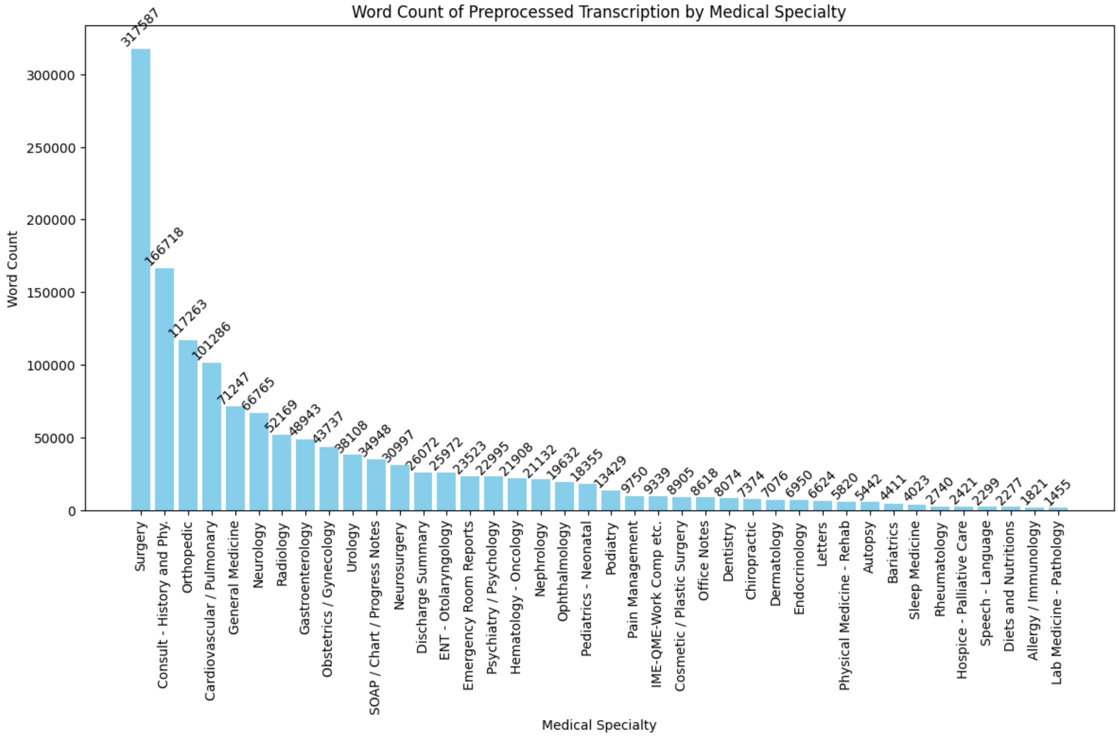
[37]:

```
            medical_specialty  \
0        Allergy / Immunology
1                  Bariatrics
2                  Bariatrics
3  Cardiovascular / Pulmonary
4  Cardiovascular / Pulmonary

preprocessed_transcription
```

```
0  allergy, immunology, allergic, rhinitis, allergy, asthma, nasal, spray,
rhinitis, nasal, erythematous, allegra, spray, allergicsubjective, yearold,
white, female, present, complaint, allergy, used, allergy, lived, seattle,
think, worse, past, tried, c…
1  bariatrics, laparoscopic, gastric, bypass, weight, loss, program, gastric,
bypass, atkins, diet, weight, watcher, body, weight, laparoscopic, gastric,
weight, loss, pound, month, weight, laparoscopic, band, loss, diet, overweight,
lostpast, medical, h…
2  bariatrics, laparoscopic, gastric, bypass, heart, attack, body, weight,
pulmonary, embolism, potential, complication, sleep, study, weight, loss,
gastric, bypass, anastomosis, loss, sleep, laparoscopic, gastric, bypass, heart,
pound, weighthistory, pr…
3  cardiovascular, pulmonary, mmode, doppler, aortic, valve, atrial,
enlargement, diastolic, function, ejection, fraction, mitral, mitral, valve,
pericardial, effusion, pulmonary, valve, regurgitation, systolic, function,
tricuspid, tricuspid, valve, nor…
4  cardiovascular, pulmonary, doppler, echocardiogram, annular, aortic, root,
aortic, valve, atrial, atrium, calcification, cavity, ejection, fraction,
mitral, obliteration, outflow, regurgitation, relaxation, pattern, stenosis,
systolic, function, tricu…
```

[38]:

```python
# Calculate word count for each preprocessed_transcription
df['word_count'] = df['preprocessed_transcription'].apply(lambda x: len(str(x).split()))

# Aggregate word counts by category (assuming the category column is 'medical_specialty')
word_counts_by_category = df.groupby('medical_specialty')['word_count'].sum().reset_index()

# Sort the DataFrame by word counts in descending order
word_counts_by_category = word_counts_by_category.sort_values(by='word_count', ascending=False)

# Plotting the bar graph
plt.figure(figsize=(12, 8))
bars = plt.bar(word_counts_by_category['medical_specialty'], word_counts_by_category['word_count'], color='skyblue')
plt.xlabel('Medical Specialty')
plt.ylabel('Word Count')
plt.title('Word Count of Preprocessed Transcription by Medical Specialty')
plt.xticks(rotation=90)

# Adding the word count on top of each bar with rotation
for bar in bars:
    yval = bar.get_height()
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        yval,
        int(yval),
        ha='center',
        va='bottom',
        rotation=45
    )

plt.tight_layout()

# Show the plot
plt.show()
```



##Appling Domian Knowledge

[39]:
```
filtered_data_categories = df.copy()

# Strip whitespace from 'medical_specialty' column values

filtered_data_categories['medical_specialty'] =␣
 ↪filtered_data_categories['medical_specialty'].apply(lambda x: x.strip())

# List of categories to exclude
```

```
categories_to_exclude = [
    'Surgery', 'SOAP / Chart / Progress Notes', 'Office Notes',
    'Consult - History and Phy.', 'Emergency Room Reports',
    'Discharge Summary', 'Pain Management', 'General Medicine', 'Letters'
]

# Exclude specified categories
for category in categories_to_exclude:
    mask = filtered_data_categories['medical_specialty'] == category
    filtered_data_categories = filtered_data_categories[~mask]

# Adjusting categories
filtered_data_categories['medical_specialty'] =␣
 ↪filtered_data_categories['medical_specialty'].replace({
    'Neurosurgery': 'Neurology',
    'Nephrology': 'Urology'
})

# Group by 'medical_specialty' and get the size of each group
grouped_data = filtered_data_categories.groupby('medical_specialty').size()

# Sort the groups by size in descending order
sorted_grouped_data = grouped_data.sort_values(ascending=False)

i = 1
print('============Reduced Categories======================')
for catName, count in sorted_grouped_data.items():
    print(f'Cat: {i} {catName} : {count}')
    i += 1
print('============Reduced Categories======================')

# Select relevant columns and drop rows with NaN values in␣
 ↪'preprocessed_transcription'
df = filtered_data_categories[['preprocessed_transcription',␣
 ↪'medical_specialty']]
df = df.dropna(subset=['preprocessed_transcription'])
print(f'data shape is: {df.shape}')
```

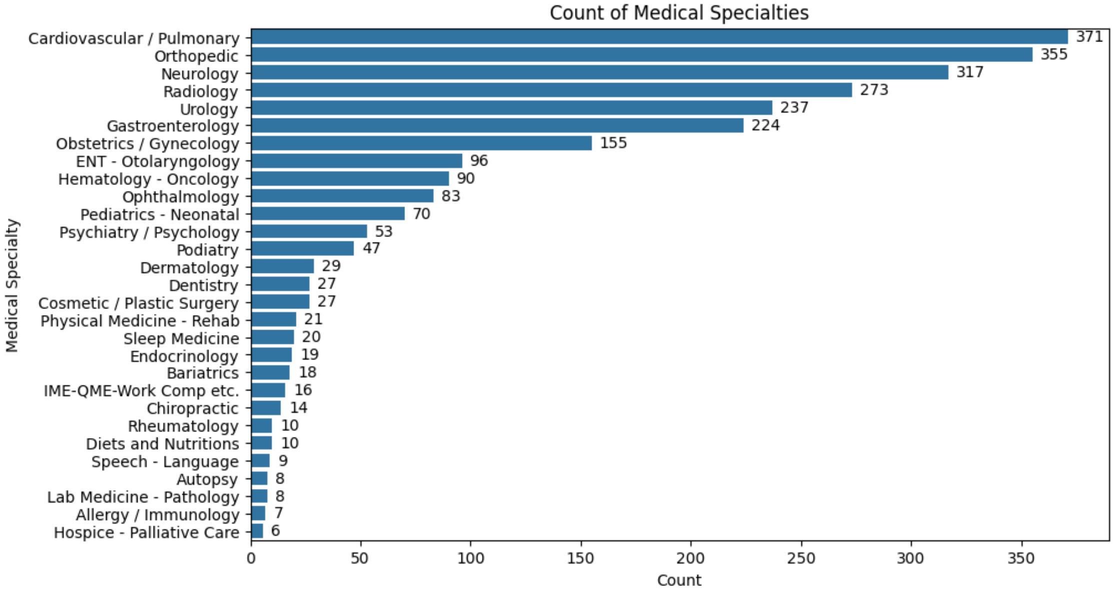
```
============Reduced Categories======================
Cat: 1 Cardiovascular / Pulmonary : 371
Cat: 2 Orthopedic : 355
Cat: 3 Neurology : 317
Cat: 4 Radiology : 273
Cat: 5 Urology : 237
Cat: 6 Gastroenterology : 224
Cat: 7 Obstetrics / Gynecology : 155
```

```
Cat: 8 ENT - Otolaryngology : 96
Cat: 9 Hematology - Oncology : 90
Cat: 10 Ophthalmology : 83
Cat: 11 Pediatrics - Neonatal : 70
Cat: 12 Psychiatry / Psychology : 53
Cat: 13 Podiatry : 47
Cat: 14 Dermatology : 29
Cat: 15 Cosmetic / Plastic Surgery : 27
Cat: 16 Dentistry : 27
Cat: 17 Physical Medicine - Rehab : 21
Cat: 18 Sleep Medicine : 20
Cat: 19 Endocrinology : 19
Cat: 20 Bariatrics : 18
Cat: 21 IME-QME-Work Comp etc. : 16
Cat: 22 Chiropractic : 14
Cat: 23 Diets and Nutritions : 10
Cat: 24 Rheumatology : 10
Cat: 25 Speech - Language : 9
Cat: 26 Lab Medicine - Pathology : 8
Cat: 27 Autopsy : 8
Cat: 28 Allergy / Immunology : 7
Cat: 29 Hospice - Palliative Care : 6
============Reduced Categories======================
data shape is: (2620, 2)
```

Count of medical specialties after applying domain knowledge

[40]:
```python
# Set the figure size
plt.figure(figsize=(10, 6))

# Create the count plot using Seaborn
ax = sns.countplot(y='medical_specialty',  # Plotting against␣
 ↪'medical_specialty' on the y-axis
                   data=filtered_data_categories,  # Using data from␣
 ↪'filtered_data_categories'
                   order=filtered_data_categories['medical_specialty'].
 ↪value_counts().index)  # Order by value counts

# Set plot title and labels
plt.title('Count of Medical Specialties')
plt.xlabel('Count')
plt.ylabel('Medical Specialty')

# Annotate each bar with its count
for p in ax.patches:
    ax.annotate(f'{int(p.get_width())}',  # Text to display (count of each bar)
                (p.get_width(), p.get_y() + p.get_height() / 2),  # Position to␣
 ↪annotate
```

```
                ha='left', va='center',  # Alignment of the text
                xytext=(5, 0),  # Offset of the text from the annotation point
                textcoords='offset points')  # Offset in points

# Display the plot
plt.show()
```



##Category Filtering and Adjustment

[41]:
```
# Count the occurrences of each medical specialty
counts = df['medical_specialty'].value_counts()

# Filter the DataFrame to keep only rows where 'medical_specialty' appears 50␣
↪times or more
df = df[~df['medical_specialty'].isin(counts[counts < 50].index)]
```

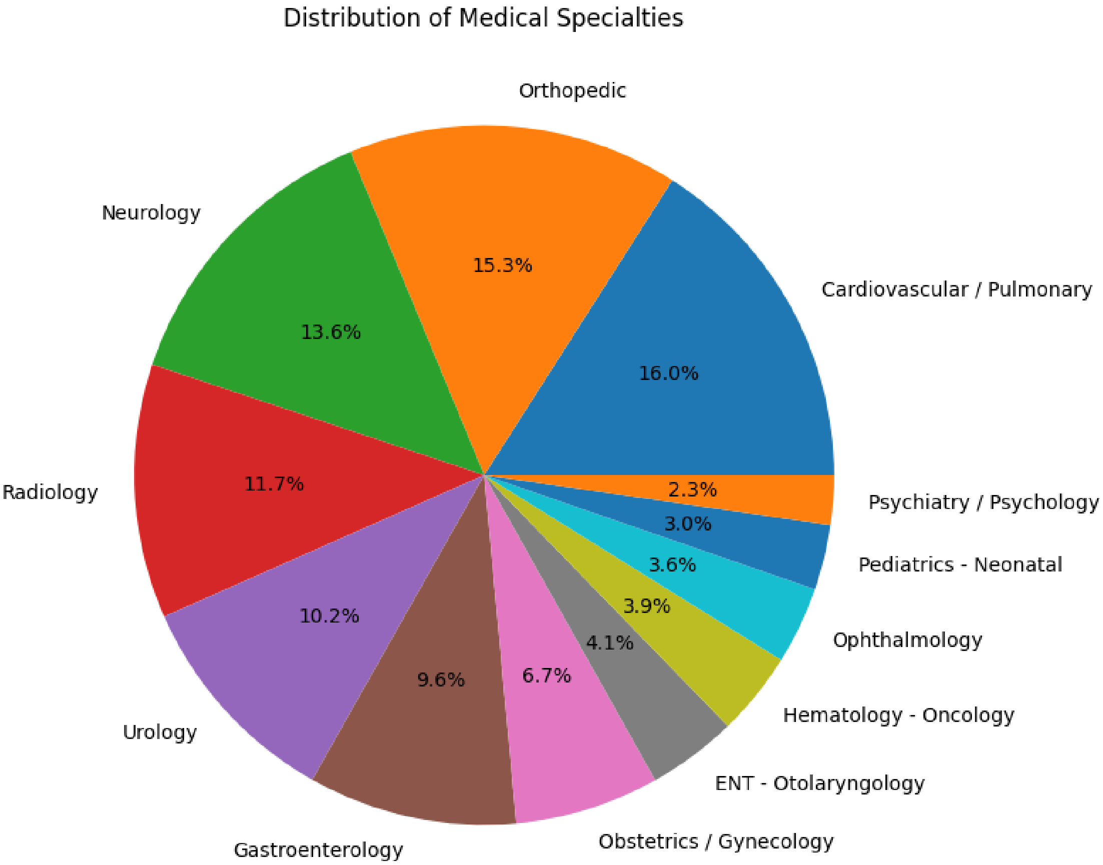
Finalized medical specialty and respective count

[42]:
```
# Check the value counts again to see the cleaned data
print(df['medical_specialty'].value_counts())
```

```
medical_specialty
Cardiovascular / Pulmonary    371
Orthopedic                    355
Neurology                     317
Radiology                     273
Urology                       237
Gastroenterology              224
```

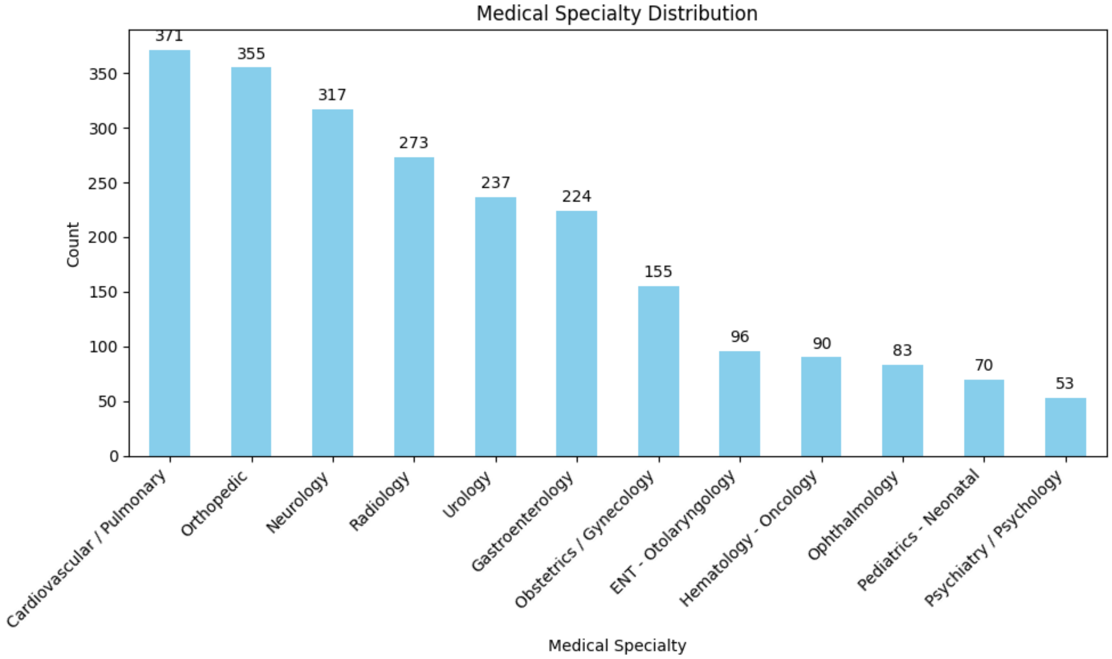
```
Obstetrics / Gynecology       155
ENT - Otolaryngology           96
Hematology - Oncology          90
Ophthalmology                  83
Pediatrics - Neonatal          70
Psychiatry / Psychology        53
Name: count, dtype: int64
```

```
[43]: # Plotting a pie chart
      df['medical_specialty'].value_counts().plot(kind='pie', autopct='%1.1f%%',␣
       ↪figsize=(10, 8))
      plt.title('Distribution of Medical Specialties')
      plt.ylabel('')
      plt.show()
```



Bar graph of finalized medical specialty distribution

[44]:
```
# Calculate the counts of each medical specialty
medical_specialty_counts = df['medical_specialty'].value_counts()

# Plotting the bar graph
plt.figure(figsize=(10, 6))  # Set the figure size
bars = medical_specialty_counts.plot(kind='bar', color='skyblue')  # Plotting a␣
 ↪bar chart with skyblue color

plt.title('Medical Specialty Distribution')  # Adding a title to the plot
plt.xlabel('Medical Specialty')  # Labeling the x-axis
plt.ylabel('Count')  # Labeling the y-axis
plt.xticks(rotation=45, ha='right')  # Rotating x-axis labels for better␣
 ↪readability

# Adding the counts on top of each bar
for bar in bars.patches:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval + 5, round(yval),␣
 ↪ha='center', va='bottom')

plt.tight_layout()
plt.show()  # Displaying the plot
```



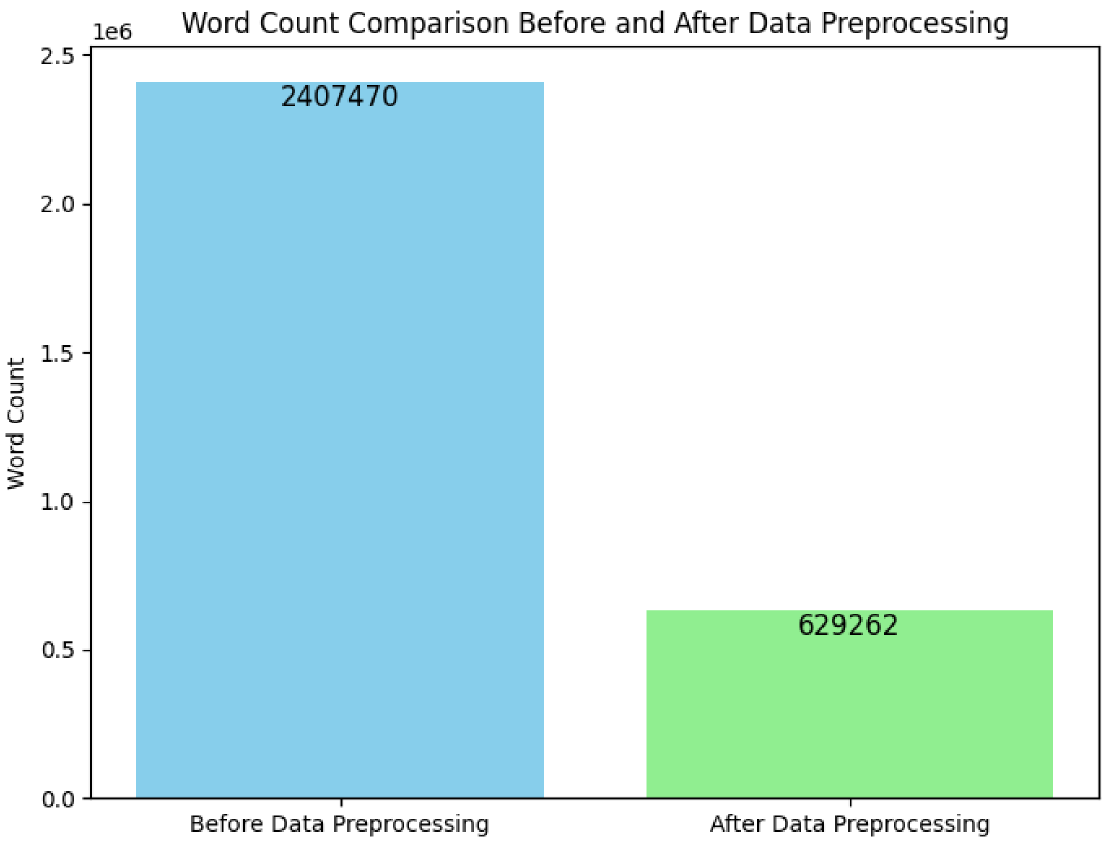
[45]:
```
# Calculate total word count of preprocessed transcriptions
```

```
total_word_count_preprocessing = df['preprocessed_transcription'].str.split().
 ↪str.len().sum()

# Print the total word count after preprocessing
print(f'The word count of transcription after Data Preprocessing is:␣
 ↪{int(total_word_count_preprocessing)}')

# Calculate percentage reduction in word count after preprocessing
percentage_reduction = round((total_word_count -␣
 ↪total_word_count_preprocessing) / total_word_count * 100, 2)

# Print the percentage reduction
print(f'{percentage_reduction}% less word')
```

```
The word count of transcription after Data Preprocessing is: 629262
73.86% less word
```

[46]:

```
# Word counts before and after normalization
word_counts = [total_word_count, total_word_count_preprocessing]
labels = ['Before Data Preprocessing', 'After Data Preprocessing']

# Plotting
plt.figure(figsize=(8, 6))
bars = plt.bar(labels, word_counts, color=['skyblue', 'lightgreen'])
plt.title('Word Count Comparison Before and After Data Preprocessing')
plt.ylabel('Word Count')

# Adding the word count on top of each bar
for bar, count in zip(bars, word_counts):
    plt.text(bar.get_x() + bar.get_width()/2,
             bar.get_height() - 100000,  # Adjusting the height of the text
             f'{count}',
             ha='center',
             va='bottom',
             fontsize=12,
             color='black')

plt.show()
```



#Label Encoding and Flattening

[47]:
```
from sklearn import preprocessing

# Initialize LabelEncoder
le = preprocessing.LabelEncoder()

# Fit LabelEncoder with unique values from 'medical_specialty' and transform to␣
↪numerical labels
le.fit(df['medical_specialty'])

# Encode 'medical_specialty' column with numerical labels
df['encoded_target'] = le.transform(df['medical_specialty'])

# Display the first few rows of the DataFrame with encoded labels
df.head()
```

[47]:
```
preprocessed_transcription  \
3   cardiovascular, pulmonary, mmode, doppler, aortic, valve, atrial,
enlargement, diastolic, function, ejection, fraction, mitral, mitral, valve,
pericardial, effusion, pulmonary, valve, regurgitation, systolic, function,
tricuspid, tricuspid, valve, nor…
4   cardiovascular, pulmonary, doppler, echocardiogram, annular, aortic, root,
aortic, valve, atrial, atrium, calcification, cavity, ejection, fraction,
mitral, obliteration, outflow, regurgitation, relaxation, pattern, stenosis,
systolic, function, tricu…
7   cardiovascular, pulmonary, echocardiogram, cardiac, function, doppler,
echocardiogram, multiple, view, aortic, valve, coronary, artery, descending,
aorta, great, vessel, heart, hypertrophy, interatrial, septum, intracardiac,
pericardial, effusion, tri…
9   cardiovascular, pulmonary, ejection, fraction, lv, systolic, function,
cardiac, chamber, regurgitation, tricuspid, normal, lv, systolic, function,
normal, lv, systolic, ejection, fraction, estimated, normal, lv, lv, systolic,
systolic, function, funct…
11  cardiovascular, pulmonary, study, doppler, tricuspid, regurgitation, heart,
pressure, stenosis, ventricular, heart, ventricle, tricuspid, regurgitationd,
study, mild, aortic, stenosis, widely, calcified, minimally, restricted, mild,
left, ventricular,…

             medical_specialty  encoded_target
3   Cardiovascular / Pulmonary               0
4   Cardiovascular / Pulmonary               0
7   Cardiovascular / Pulmonary               0
9   Cardiovascular / Pulmonary               0
11  Cardiovascular / Pulmonary               0
```

[48]:

```
df.shape
```

[48]:

```
(2324, 3)
```

[49]:

```
# function to flatten one list
def flat_list(unflat_list):
    flatted = [item for sublist in unflat_list for item in sublist]
    return flatted

def to_list(df, attribute):
    # Select the normalised transcript column
    df_transcription = df[[attribute]]
    # To convert the attribute into list format, but it has inner list. So it␣
 ↪cannot put into the CountVectoriser
    unflat_list_transcription = df_transcription.values.tolist()
    # Let's use back the function defined above, "flat_list", to flatten the␣
 ↪list
    flat_list_transcription = flat_list(unflat_list_transcription)
    return flat_list_transcription
flat_list_transcription = to_list(df, 'preprocessed_transcription')
```

#Baseline Models (Traditional ML) Development

#ML model with Bag of Words Representation and PCA

[50]:

```python
# Step 1: Bag of Words Representation
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(df['preprocessed_transcription'])

# Step 2: Dimension Reduction with PCA
pca = PCA(n_components=100)  # You can adjust the number of components as needed
X_pca = pca.fit_transform(X.toarray())

# Step 3: Handling Imbalanced Data with SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_pca, df['encoded_target'])

# Step 4: Splitting Data into Train and Test Sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# list of category names from the "medical_specialty" column
category_names = df['medical_specialty'].unique().tolist()

# Step 5: Model Building and Evaluation
classifiers = {
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(),
    "Logistic Regression": LogisticRegression()
}

for name, classifier in classifiers.items():
    print(f"Training and evaluating {name}...")
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print(f"Results for {name}:")
    print(classification_report(y_test, y_pred, target_names=category_names))
    print("\n" + "="*60 + "\n")
```

```
Training and evaluating Random Forest…
Results for Random Forest:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.65      0.69      0.67        74
                 Neurology       0.91      0.94      0.92        77
                   Urology       0.83      0.76      0.79        76
                 Radiology       0.82      0.87      0.85        79
   Psychiatry / Psychology       0.43      0.44      0.43        66
```

```
     Pediatrics - Neonatal       0.84      0.93      0.88        74
                Orthopedic       0.93      1.00      0.96        55
             Ophthalmology       0.72      0.67      0.69        79
   Obstetrics / Gynecology       0.78      0.80      0.79        71
     Hematology - Oncology       0.97      0.99      0.98        73
          Gastroenterology       0.28      0.24      0.26        87
      ENT - Otolaryngology       0.84      0.79      0.81        80

                  accuracy                           0.75       891
                 macro avg       0.75      0.76      0.75       891
              weighted avg       0.74      0.75      0.75       891


============================================================

Training and evaluating Support Vector Machine…
Results for Support Vector Machine:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.77      0.73      0.75        74
                 Neurology       0.91      0.83      0.87        77
                   Urology       0.88      0.78      0.83        76
                 Radiology       0.77      0.89      0.82        79
   Psychiatry / Psychology       0.54      0.55      0.54        66
     Pediatrics - Neonatal       0.89      0.89      0.89        74
                Orthopedic       0.95      1.00      0.97        55
             Ophthalmology       0.66      0.65      0.65        79
   Obstetrics / Gynecology       0.81      0.77      0.79        71
     Hematology - Oncology       0.93      0.95      0.94        73
          Gastroenterology       0.44      0.54      0.49        87
      ENT - Otolaryngology       0.91      0.79      0.85        80

                  accuracy                           0.77       891
                 macro avg       0.79      0.78      0.78       891
              weighted avg       0.78      0.77      0.78       891


============================================================

Training and evaluating XGBoost…
Results for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.69      0.69      0.69        74
                 Neurology       0.93      0.92      0.93        77
                   Urology       0.91      0.79      0.85        76
                 Radiology       0.83      0.94      0.88        79
   Psychiatry / Psychology       0.43      0.45      0.44        66
```

```
     Pediatrics - Neonatal      0.86      0.93      0.90        74
                Orthopedic      0.96      1.00      0.98        55
             Ophthalmology      0.70      0.62      0.66        79
   Obstetrics / Gynecology      0.79      0.83      0.81        71
     Hematology - Oncology      0.97      0.99      0.98        73
          Gastroenterology      0.37      0.33      0.35        87
      ENT - Otolaryngology      0.81      0.82      0.82        80

                  accuracy                          0.77       891
                 macro avg      0.77      0.78      0.77       891
              weighted avg      0.77      0.77      0.77       891


============================================================

Training and evaluating Logistic Regression…
Results for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary      0.74      0.73      0.73        74
                 Neurology      0.92      0.90      0.91        77
                   Urology      0.82      0.86      0.84        76
                 Radiology      0.80      0.87      0.84        79
   Psychiatry / Psychology      0.52      0.47      0.49        66
     Pediatrics - Neonatal      0.86      0.92      0.89        74
                Orthopedic      0.96      1.00      0.98        55
             Ophthalmology      0.72      0.66      0.69        79
   Obstetrics / Gynecology      0.81      0.79      0.80        71
     Hematology - Oncology      0.95      1.00      0.97        73
          Gastroenterology      0.50      0.48      0.49        87
      ENT - Otolaryngology      0.84      0.84      0.84        80

                  accuracy                          0.79       891
                 macro avg      0.79      0.79      0.79       891
              weighted avg      0.78      0.79      0.78       891


============================================================
```

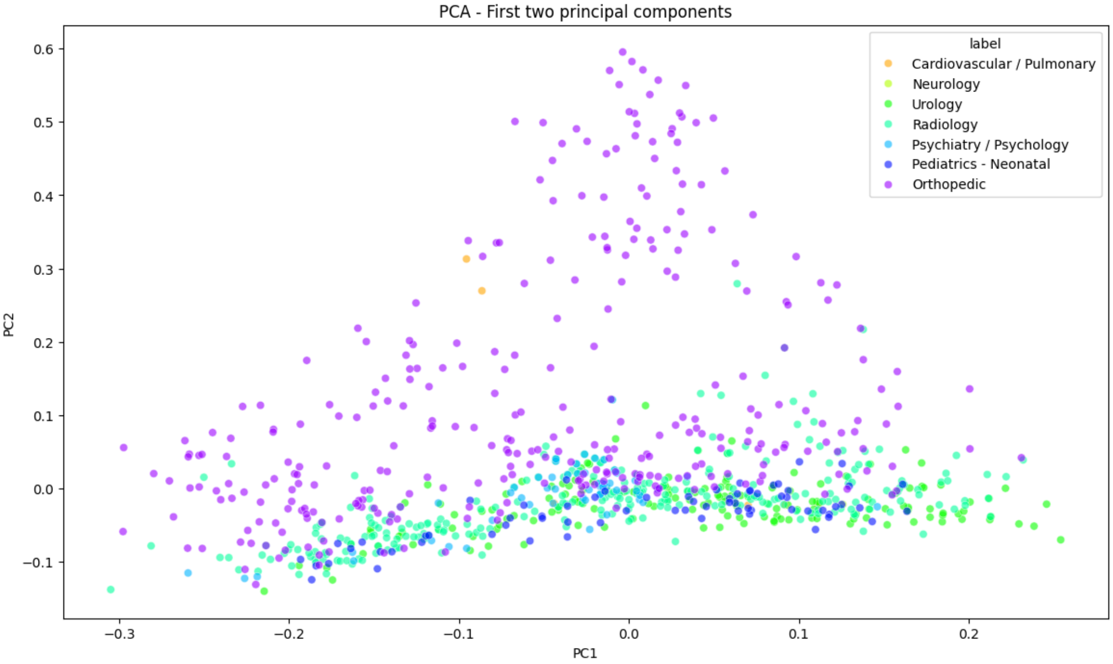
#ML model with TF-IDF Word Representation with PCA

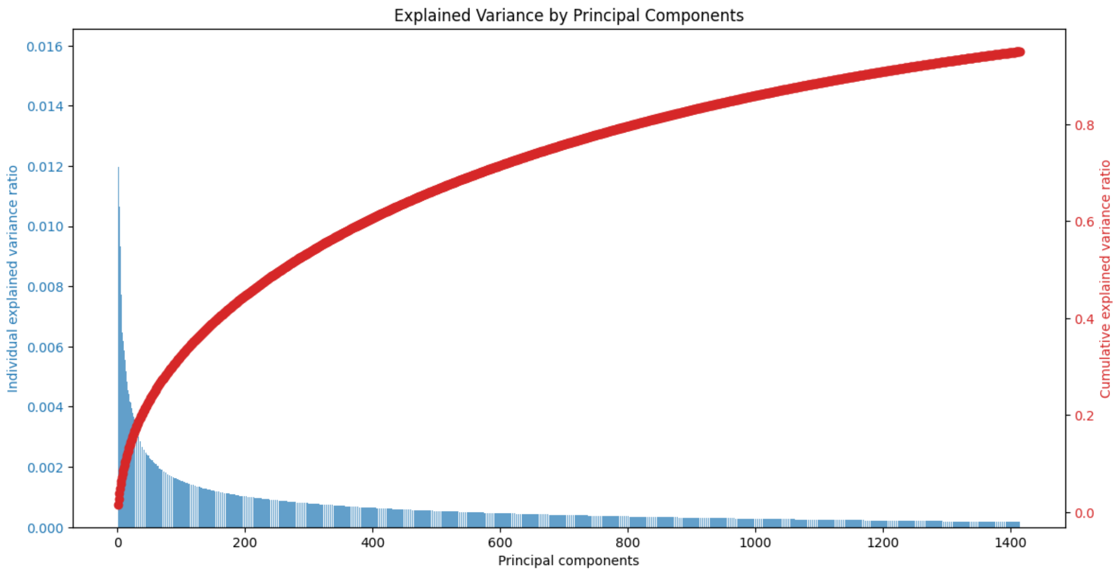
[51]:

```
# Step 1: TF-IDF Word Representation

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Apply TF-IDF vectorization to the text data
```

```
X_tfidf = tfidf_vectorizer.fit_transform(flat_list_transcription)

# Step 2: PCA for Dimension Reduction

# Initialize PCA with 95% variance explained
pca = PCA(n_components=0.95)

# Perform PCA on the TF-IDF transformed data
X_pca = pca.fit_transform(X_tfidf.toarray())

# Calculate explained variance ratio and cumulative explained variance
explained_variance = pca.explained_variance_ratio_
cumulative_explained_variance = explained_variance.cumsum()

# Visualizing PCA results

# Create a dataframe with the first two principal components and labels
pca_df = pd.DataFrame(data=X_pca, columns=[f'PC{i+1}' for i in range(X_pca.
 ↪shape[1])])
pca_df['label'] = df['medical_specialty']  # assuming 'medical_specialty' is␣
 ↪the label

# Plot the first two principal components
plt.figure(figsize=(14, 8))
sns.scatterplot(
    x='PC1', y='PC2',
    hue='label',
    palette=sns.color_palette("hsv", len(pca_df['label'].unique())),
    data=pca_df,
    legend="full",
    alpha=0.6
)
plt.title('PCA - First two principal components')
plt.show()

# Plot the explained variance ratio with secondary y-axis for cumulative␣
 ↪explained variance
fig, ax1 = plt.subplots(figsize=(12, 6))

color = 'tab:blue'
ax1.set_xlabel('Principal components')
ax1.set_ylabel('Individual explained variance ratio', color=color)
ax1.bar(range(1, len(explained_variance) + 1), explained_variance, alpha=0.7,␣
 ↪align='center', color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx()  # instantiate a second axes that shares the same x-axis
```

```
color = 'tab:red'
ax2.set_ylabel('Cumulative explained variance ratio', color=color)
ax2.plot(range(1, len(cumulative_explained_variance) + 1),␣
 ↪cumulative_explained_variance, color=color, marker='o')
ax2.tick_params(axis='y', labelcolor=color)

fig.tight_layout()
plt.title('Explained Variance by Principal Components')
plt.show()
```





[52]:

```
# Step 3: Handling Imbalanced Data with SMOTE

# Apply SMOTE for oversampling
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_pca, df['medical_specialty']) ␣
 ↪# assuming 'medical_specialty' is the label

# Step 4: Splitting Data into Train and Test Sets (80% train, 20% test)

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.2, random_state=42)

# Step 5: Building and Evaluating Classifiers

# Define classifiers to evaluate
classifiers = {
    'Random Forest': RandomForestClassifier(),
    'Support Vector Machine': SVC(),
    'XGBoost': XGBClassifier(),
    'Logistic Regression': LogisticRegression()
}

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical labels
y_train_encoded = label_encoder.fit_transform(y_train)
```

```python
y_test_encoded = label_encoder.transform(y_test)

# Train and evaluate each classifier
for name, clf in classifiers.items():
    clf.fit(X_train, y_train_encoded)
    y_pred = clf.predict(X_test)
    print(f"Classifier: {name}")
    print(classification_report(y_test_encoded, y_pred,␣
 ↪target_names=label_encoder.classes_))
    print("\n" + "="*60 + "\n")
```

```
Classifier: Random Forest
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.64      0.70      0.67        74
      ENT - Otolaryngology       0.94      0.95      0.94        77
          Gastroenterology       0.85      0.83      0.84        76
     Hematology - Oncology       0.89      0.89      0.89        79
                 Neurology       0.49      0.41      0.45        66
   Obstetrics / Gynecology       0.82      0.95      0.88        74
             Ophthalmology       0.98      1.00      0.99        55
                Orthopedic       0.72      0.70      0.71        79
     Pediatrics - Neonatal       0.85      0.87      0.86        71
   Psychiatry / Psychology       1.00      0.97      0.99        73
                 Radiology       0.26      0.24      0.25        87
                   Urology       0.88      0.89      0.88        80

                  accuracy                           0.77       891
                 macro avg       0.78      0.78      0.78       891
              weighted avg       0.77      0.77      0.77       891


============================================================

Classifier: Support Vector Machine
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.66      0.76      0.70        74
      ENT - Otolaryngology       0.93      0.88      0.91        77
          Gastroenterology       0.89      0.84      0.86        76
     Hematology - Oncology       0.89      0.92      0.91        79
                 Neurology       0.54      0.55      0.54        66
   Obstetrics / Gynecology       0.84      0.92      0.88        74
             Ophthalmology       0.96      1.00      0.98        55
                Orthopedic       0.79      0.71      0.75        79
     Pediatrics - Neonatal       0.87      0.92      0.89        71
   Psychiatry / Psychology       1.00      0.97      0.99        73
```

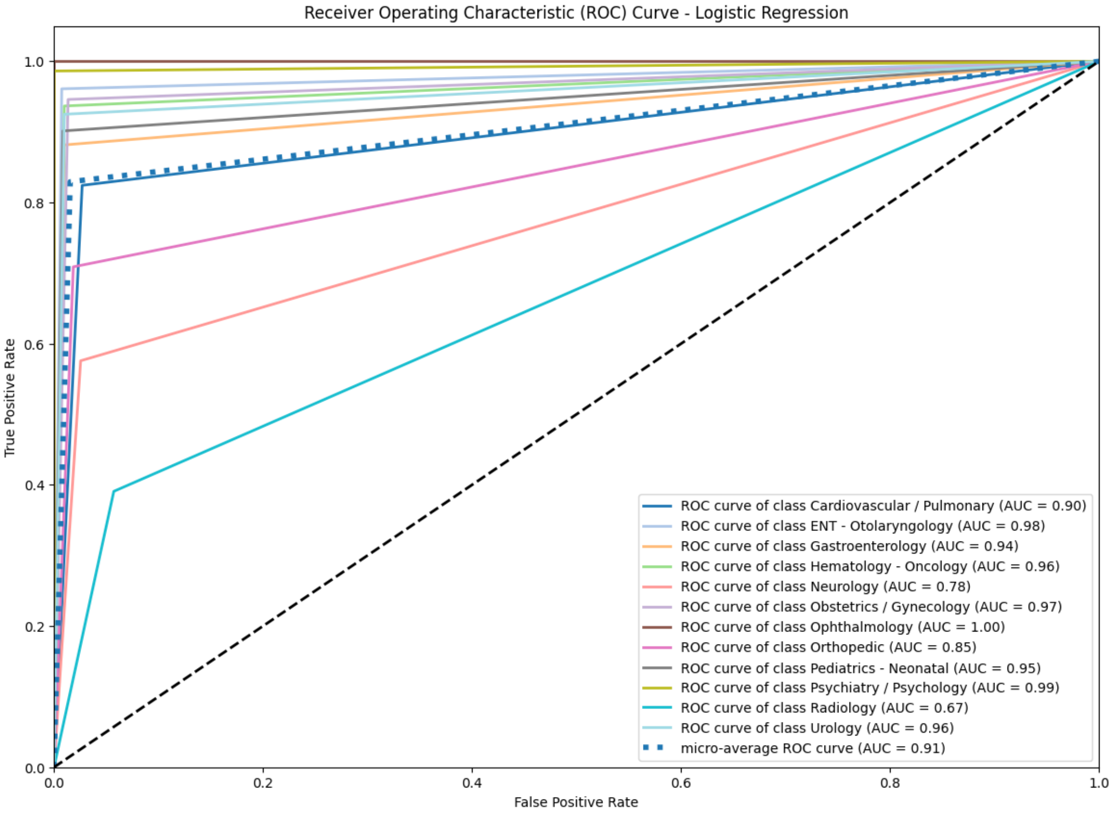
```
                 Radiology       0.29      0.25      0.27        87
                   Urology       0.88      0.89      0.88        80

                  accuracy                           0.79       891
                 macro avg       0.79      0.80      0.80       891
              weighted avg       0.79      0.79      0.79       891


============================================================

Classifier: XGBoost
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.67      0.76      0.71        74
      ENT - Otolaryngology       0.92      0.94      0.93        77
          Gastroenterology       0.87      0.88      0.88        76
     Hematology - Oncology       0.88      0.89      0.88        79
                 Neurology       0.51      0.45      0.48        66
   Obstetrics / Gynecology       0.86      0.91      0.88        74
             Ophthalmology       0.98      1.00      0.99        55
                Orthopedic       0.74      0.70      0.72        79
     Pediatrics - Neonatal       0.85      0.94      0.89        71
   Psychiatry / Psychology       0.99      0.97      0.98        73
                 Radiology       0.40      0.36      0.38        87
                   Urology       0.91      0.88      0.89        80

                  accuracy                           0.80       891
                 macro avg       0.80      0.81      0.80       891
              weighted avg       0.79      0.80      0.79       891


============================================================

Classifier: Logistic Regression
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.73      0.82      0.78        74
      ENT - Otolaryngology       0.93      0.96      0.94        77
          Gastroenterology       0.89      0.88      0.89        76
     Hematology - Oncology       0.90      0.94      0.92        79
                 Neurology       0.64      0.58      0.61        66
   Obstetrics / Gynecology       0.86      0.95      0.90        74
             Ophthalmology       1.00      1.00      1.00        55
                Orthopedic       0.79      0.71      0.75        79
     Pediatrics - Neonatal       0.90      0.90      0.90        71
   Psychiatry / Psychology       1.00      0.99      0.99        73
                 Radiology       0.42      0.39      0.41        87
                   Urology       0.90      0.93      0.91        80
```

```
    accuracy                           0.83       891
   macro avg       0.83      0.84      0.83       891
weighted avg       0.82      0.83      0.83       891


============================================================
```

#AUC and ROC of TF-IDF Word Representation with PCA (Logistic Regression)

[53]:

```
# Number of classes
n_classes = len(label_encoder.classes_)

# Binarize the output
y_test_binarized = label_binarize(y_test_encoded, classes=np.arange(n_classes))
y_pred_binarized = label_binarize(y_pred, classes=np.arange(n_classes))

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_pred_binarized[:,␣
 ↪i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_binarized.ravel(),␣
 ↪y_pred_binarized.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot all ROC curves
plt.figure(figsize=(14, 10))

colors = plt.cm.get_cmap('tab20', n_classes)

for i, color in zip(range(n_classes), colors.colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'ROC curve of class {label_encoder.classes_[i]} (AUC =␣
 ↪{roc_auc[i]:0.2f})')

plt.plot(fpr["micro"], tpr["micro"], linestyle=':', linewidth=4,
         label=f'micro-average ROC curve (AUC = {roc_auc["micro"]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.show()
```



#ML model with Bag-of-n-grams (1,1) Word Representation and TruncatedSVD

[54]:

```
# Step 1: N-gram Representation
vectorizer = CountVectorizer(ngram_range=(1, 1))
X = vectorizer.fit_transform(flat_list_transcription)
y = df['encoded_target']

# Step 2: Imbalanced Data Handling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 3: Dimensionality Reduction with TruncatedSVD
svd = TruncatedSVD(n_components=100)
X_svd = svd.fit_transform(X_resampled)
```

```python
# Step 4: Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X_svd, y_resampled, test_size=0.2, random_state=42)

# Extract unique category names from the "medical_specialty" column
category_names = df['medical_specialty'].unique()

# Step 5: Model Building
models = {
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(),
    "Logistic Regression": LogisticRegression()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=category_names)
    print(f"Classification Report for {name}:\n{report}")
    print("\n" + "="*60 + "\n")
```

```
Classification Report for Random Forest:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.61      0.69      0.65        74
                 Neurology       0.91      0.96      0.94        77
                   Urology       0.82      0.78      0.80        76
                 Radiology       0.74      0.80      0.77        79
   Psychiatry / Psychology       0.42      0.38      0.40        66
     Pediatrics - Neonatal       0.82      0.84      0.83        74
                Orthopedic       0.96      1.00      0.98        55
             Ophthalmology       0.69      0.65      0.67        79
   Obstetrics / Gynecology       0.81      0.90      0.85        71
     Hematology - Oncology       0.96      0.99      0.97        73
          Gastroenterology       0.25      0.22      0.23        87
      ENT - Otolaryngology       0.82      0.75      0.78        80

                  accuracy                           0.74       891
                 macro avg       0.73      0.75      0.74       891
              weighted avg       0.73      0.74      0.73       891


============================================================


Classification Report for Support Vector Machine:
                            precision    recall  f1-score   support
```

```
Cardiovascular / Pulmonary       0.65      0.72      0.68        74
                 Neurology       0.92      0.86      0.89        77
                   Urology       0.86      0.67      0.76        76
                 Radiology       0.72      0.90      0.80        79
   Psychiatry / Psychology       0.59      0.55      0.57        66
     Pediatrics - Neonatal       0.80      0.74      0.77        74
                Orthopedic       0.98      1.00      0.99        55
             Ophthalmology       0.75      0.62      0.68        79
   Obstetrics / Gynecology       0.54      0.82      0.65        71
     Hematology - Oncology       0.93      0.95      0.94        73
          Gastroenterology       0.40      0.40      0.40        87
      ENT - Otolaryngology       0.93      0.70      0.80        80

                  accuracy                           0.73       891
                 macro avg       0.76      0.74      0.74       891
              weighted avg       0.75      0.73      0.74       891


============================================================

Classification Report for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.62      0.72      0.66        74
                 Neurology       0.91      0.95      0.93        77
                   Urology       0.82      0.79      0.81        76
                 Radiology       0.81      0.82      0.82        79
   Psychiatry / Psychology       0.48      0.45      0.47        66
     Pediatrics - Neonatal       0.83      0.85      0.84        74
                Orthopedic       0.96      1.00      0.98        55
             Ophthalmology       0.76      0.71      0.73        79
   Obstetrics / Gynecology       0.83      0.85      0.84        71
     Hematology - Oncology       0.97      0.99      0.98        73
          Gastroenterology       0.28      0.28      0.28        87
      ENT - Otolaryngology       0.83      0.74      0.78        80

                  accuracy                           0.75       891
                 macro avg       0.76      0.76      0.76       891
              weighted avg       0.75      0.75      0.75       891


============================================================

Classification Report for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.71      0.77      0.74        74
```

```
               Neurology       0.90      0.92      0.91        77
                 Urology       0.85      0.88      0.86        76
               Radiology       0.82      0.91      0.86        79
  Psychiatry / Psychology       0.58      0.56      0.57        66
    Pediatrics - Neonatal       0.88      0.89      0.89        74
              Orthopedic       0.98      1.00      0.99        55
           Ophthalmology       0.79      0.72      0.75        79
  Obstetrics / Gynecology       0.90      0.87      0.89        71
    Hematology - Oncology       0.91      0.99      0.95        73
        Gastroenterology       0.46      0.40      0.43        87
     ENT - Otolaryngology       0.88      0.81      0.84        80

                accuracy                           0.80       891
               macro avg       0.80      0.81      0.81       891
            weighted avg       0.80      0.80      0.80       891


============================================================
```

#ML model with Bag-of-n-grams (1,2) Word Representation and TruncatedSVD

[55]:

```python
# Step 1: N-gram Representation
vectorizer = CountVectorizer(ngram_range=(1, 2))
X = vectorizer.fit_transform(flat_list_transcription)
y = df['encoded_target']

# Step 2: Imbalanced Data Handling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 3: Dimensionality Reduction with TruncatedSVD
svd = TruncatedSVD(n_components=100)
X_svd = svd.fit_transform(X_resampled)

# Step 4: Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X_svd, y_resampled,␣
 ↪test_size=0.2, random_state=42)

# Extract unique category names from the "medical_specialty" column
category_names = df['medical_specialty'].unique()

# Step 5: Model Building
models = {
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(),
    "Logistic Regression": LogisticRegression()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=category_names)
    print(f"Classification Report for {name}:\n{report}")
    print("\n" + "="*60 + "\n")
```

```
Classification Report for Random Forest:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.56      0.66      0.61        74
                 Neurology       0.92      0.92      0.92        77
                   Urology       0.81      0.76      0.78        76
                 Radiology       0.81      0.82      0.82        79
   Psychiatry / Psychology       0.49      0.42      0.46        66
     Pediatrics - Neonatal       0.81      0.86      0.84        74
                Orthopedic       0.95      1.00      0.97        55
             Ophthalmology       0.71      0.68      0.70        79
   Obstetrics / Gynecology       0.76      0.83      0.79        71
     Hematology - Oncology       0.94      0.99      0.96        73
          Gastroenterology       0.26      0.23      0.25        87
      ENT - Otolaryngology       0.82      0.76      0.79        80

                  accuracy                           0.74       891
                 macro avg       0.74      0.75      0.74       891
              weighted avg       0.73      0.74      0.73       891


============================================================

Classification Report for Support Vector Machine:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.65      0.72      0.68        74
                 Neurology       0.93      0.86      0.89        77
                   Urology       0.85      0.70      0.77        76
                 Radiology       0.72      0.86      0.79        79
   Psychiatry / Psychology       0.62      0.56      0.59        66
     Pediatrics - Neonatal       0.81      0.77      0.79        74
                Orthopedic       0.92      1.00      0.96        55
             Ophthalmology       0.71      0.63      0.67        79
   Obstetrics / Gynecology       0.56      0.80      0.66        71
     Hematology - Oncology       0.91      0.95      0.93        73
          Gastroenterology       0.42      0.40      0.41        87
      ENT - Otolaryngology       0.90      0.69      0.78        80
```

```
                  accuracy                           0.74       891
                 macro avg       0.75      0.74      0.74       891
              weighted avg       0.75      0.74      0.74       891


============================================================

Classification Report for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.61      0.69      0.65        74
                 Neurology       0.92      0.90      0.91        77
                   Urology       0.81      0.82      0.81        76
                 Radiology       0.81      0.81      0.81        79
   Psychiatry / Psychology       0.52      0.45      0.48        66
     Pediatrics - Neonatal       0.82      0.85      0.83        74
                Orthopedic       0.95      1.00      0.97        55
             Ophthalmology       0.72      0.70      0.71        79
   Obstetrics / Gynecology       0.82      0.82      0.82        71
     Hematology - Oncology       0.96      0.99      0.97        73
          Gastroenterology       0.30      0.29      0.29        87
      ENT - Otolaryngology       0.79      0.76      0.78        80

                  accuracy                           0.75       891
                 macro avg       0.75      0.76      0.75       891
              weighted avg       0.74      0.75      0.74       891


============================================================

Classification Report for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.70      0.82      0.76        74
                 Neurology       0.90      0.94      0.92        77
                   Urology       0.81      0.86      0.83        76
                 Radiology       0.84      0.91      0.87        79
   Psychiatry / Psychology       0.56      0.53      0.55        66
     Pediatrics - Neonatal       0.88      0.89      0.89        74
                Orthopedic       0.98      1.00      0.99        55
             Ophthalmology       0.76      0.71      0.73        79
   Obstetrics / Gynecology       0.83      0.82      0.82        71
     Hematology - Oncology       0.88      0.99      0.93        73
          Gastroenterology       0.49      0.38      0.43        87
      ENT - Otolaryngology       0.92      0.81      0.86        80

                  accuracy                           0.80       891
```

```
       macro avg       0.80      0.80      0.80       891
    weighted avg       0.79      0.80      0.79       891


============================================================
```

#ML model with Bag-of-n-grams (2,2) Word Representation and TruncatedSVD

[56]:
```python
# Step 1: N-gram Representation
vectorizer = CountVectorizer(ngram_range=(2, 2))
X = vectorizer.fit_transform(flat_list_transcription)
y = df['encoded_target']

# Step 2: Imbalanced Data Handling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 3: Dimensionality Reduction with TruncatedSVD
svd = TruncatedSVD(n_components=100, random_state=42)
X_svd = svd.fit_transform(X_resampled)

# Step 4: Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X_svd, y_resampled, test_size=0.2, random_state=42)

# Extract unique category names from the "medical_specialty" column
# Assuming df['medical_specialty'] contains the actual category names corresponding to y
category_names = df['medical_specialty'].unique()

# Step 5: Model Building
models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=category_names)
    print(f"Classification Report for {name}:\n{report}")
    print("\n" + "="*60 + "\n")
```

```
Classification Report for Random Forest:
                            precision    recall  f1-score   support
```

```
Cardiovascular / Pulmonary       0.61      0.69      0.65        74
                 Neurology       0.86      0.90      0.88        77
                   Urology       0.63      0.68      0.65        76
                 Radiology       0.73      0.77      0.75        79
   Psychiatry / Psychology       0.47      0.41      0.44        66
     Pediatrics - Neonatal       0.80      0.82      0.81        74
                Orthopedic       0.88      0.95      0.91        55
             Ophthalmology       0.71      0.63      0.67        79
   Obstetrics / Gynecology       0.76      0.83      0.79        71
     Hematology - Oncology       0.80      0.96      0.87        73
          Gastroenterology       0.31      0.28      0.29        87
      ENT - Otolaryngology       0.76      0.53      0.62        80

                  accuracy                           0.69       891
                 macro avg       0.69      0.70      0.70       891
              weighted avg       0.69      0.69      0.69       891


============================================================

Classification Report for Support Vector Machine:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.58      0.64      0.61        74
                 Neurology       0.85      0.68      0.75        77
                   Urology       0.49      0.26      0.34        76
                 Radiology       0.65      0.42      0.51        79
   Psychiatry / Psychology       0.53      0.42      0.47        66
     Pediatrics - Neonatal       0.84      0.51      0.64        74
                Orthopedic       0.58      0.76      0.66        55
             Ophthalmology       0.58      0.56      0.57        79
   Obstetrics / Gynecology       0.21      0.90      0.34        71
     Hematology - Oncology       0.85      0.30      0.44        73
          Gastroenterology       0.34      0.18      0.24        87
      ENT - Otolaryngology       0.86      0.31      0.46        80

                  accuracy                           0.48       891
                 macro avg       0.61      0.50      0.50       891
              weighted avg       0.61      0.48      0.50       891


============================================================

Classification Report for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.64      0.69      0.66        74
```

```
                   Neurology       0.91      0.90      0.90        77
                     Urology       0.68      0.68      0.68        76
                   Radiology       0.80      0.81      0.81        79
     Psychiatry / Psychology       0.40      0.38      0.39        66
       Pediatrics - Neonatal       0.78      0.84      0.81        74
                  Orthopedic       0.94      0.93      0.94        55
               Ophthalmology       0.67      0.65      0.66        79
     Obstetrics / Gynecology       0.77      0.85      0.81        71
       Hematology - Oncology       0.79      0.97      0.87        73
            Gastroenterology       0.31      0.30      0.31        87
        ENT - Otolaryngology       0.80      0.56      0.66        80

                    accuracy                           0.70       891
                   macro avg       0.71      0.71      0.71       891
                weighted avg       0.70      0.70      0.70       891


============================================================

Classification Report for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.72      0.72      0.72        74
                 Neurology       0.91      0.94      0.92        77
                   Urology       0.60      0.49      0.54        76
                 Radiology       0.78      0.80      0.79        79
   Psychiatry / Psychology       0.53      0.47      0.50        66
     Pediatrics - Neonatal       0.84      0.73      0.78        74
                Orthopedic       0.81      0.85      0.83        55
             Ophthalmology       0.74      0.68      0.71        79
   Obstetrics / Gynecology       0.52      0.80      0.63        71
     Hematology - Oncology       0.63      0.95      0.76        73
          Gastroenterology       0.48      0.33      0.39        87
      ENT - Otolaryngology       0.81      0.62      0.70        80

                  accuracy                           0.69       891
                 macro avg       0.70      0.70      0.69       891
              weighted avg       0.70      0.69      0.68       891


============================================================
```

#ML model with Bag-of-n-grams (2,3) Word Representation and TruncatedSVD

[57]:
```
# Step 1: N-gram Representation
vectorizer = CountVectorizer(ngram_range=(2, 3))
X = vectorizer.fit_transform(flat_list_transcription)
```

```
y = df['encoded_target']

# Step 2: Imbalanced Data Handling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 3: Dimensionality Reduction with TruncatedSVD
svd = TruncatedSVD(n_components=100)
X_svd = svd.fit_transform(X_resampled)

# Step 4: Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X_svd, y_resampled,␣
 ↪test_size=0.2, random_state=42)

# Extract unique category names from the "medical_specialty" column
# Create a mapping from encoded target to category name
encoded_target_to_category = dict(zip(df['encoded_target'],␣
 ↪df['medical_specialty']))

# Ensure the category names are in the correct order of the encoded targets
category_names = [encoded_target_to_category[i] for i in␣
 ↪sorted(encoded_target_to_category)]

# Step 5: Model Building
models = {
    "Random Forest": RandomForestClassifier(),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(),
    "Logistic Regression": LogisticRegression()
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=category_names)
    print(f"Classification Report for {name}:\n{report}")
    print("\n" + "="*60 + "\n")
```

```
Classification Report for Random Forest:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.59      0.66      0.62        74
      ENT - Otolaryngology       0.84      0.86      0.85        77
          Gastroenterology       0.58      0.63      0.60        76
     Hematology - Oncology       0.68      0.70      0.69        79
                 Neurology       0.43      0.39      0.41        66
   Obstetrics / Gynecology       0.77      0.72      0.74        74
```

```
             Ophthalmology       0.87      0.95      0.90        55
                Orthopedic       0.69      0.62      0.65        79
     Pediatrics - Neonatal       0.78      0.79      0.78        71
   Psychiatry / Psychology       0.75      0.95      0.84        73
                 Radiology       0.28      0.24      0.26        87
                   Urology       0.64      0.53      0.58        80

                  accuracy                           0.66       891
                 macro avg       0.66      0.67      0.66       891
              weighted avg       0.65      0.66      0.65       891


============================================================

Classification Report for Support Vector Machine:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.56      0.61      0.58        74
      ENT - Otolaryngology       0.83      0.52      0.64        77
          Gastroenterology       0.35      0.28      0.31        76
     Hematology - Oncology       0.65      0.38      0.48        79
                 Neurology       0.53      0.42      0.47        66
   Obstetrics / Gynecology       0.83      0.41      0.55        74
             Ophthalmology       0.51      0.65      0.58        55
                Orthopedic       0.62      0.57      0.59        79
     Pediatrics - Neonatal       0.19      0.90      0.31        71
   Psychiatry / Psychology       0.76      0.26      0.39        73
                 Radiology       0.30      0.11      0.17        87
                   Urology       0.88      0.26      0.40        80

                  accuracy                           0.44       891
                 macro avg       0.58      0.45      0.46       891
              weighted avg       0.59      0.44      0.45       891


============================================================

Classification Report for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.69      0.73      0.71        74
      ENT - Otolaryngology       0.86      0.87      0.86        77
          Gastroenterology       0.59      0.67      0.63        76
     Hematology - Oncology       0.78      0.75      0.76        79
                 Neurology       0.41      0.39      0.40        66
   Obstetrics / Gynecology       0.82      0.80      0.81        74
             Ophthalmology       0.98      0.96      0.97        55
                Orthopedic       0.67      0.63      0.65        79
     Pediatrics - Neonatal       0.75      0.82      0.78        71
   Psychiatry / Psychology       0.79      0.95      0.86        73
                 Radiology       0.35      0.32      0.33        87
                   Urology       0.76      0.60      0.67        80

                  accuracy                           0.70       891
                 macro avg       0.70      0.71      0.70       891
              weighted avg       0.70      0.70      0.70       891


============================================================

Classification Report for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.72      0.69      0.70        74
      ENT - Otolaryngology       0.88      0.88      0.88        77
          Gastroenterology       0.59      0.45      0.51        76
     Hematology - Oncology       0.67      0.71      0.69        79
                 Neurology       0.51      0.45      0.48        66
   Obstetrics / Gynecology       0.79      0.70      0.74        74
             Ophthalmology       0.76      0.80      0.78        55
                Orthopedic       0.71      0.62      0.66        79
     Pediatrics - Neonatal       0.38      0.83      0.52        71
   Psychiatry / Psychology       0.72      0.79      0.75        73
                 Radiology       0.42      0.26      0.32        87
                   Urology       0.80      0.59      0.68        80

                  accuracy                           0.64       891
                 macro avg       0.66      0.65      0.64       891
              weighted avg       0.66      0.64      0.64       891


============================================================
```

#ML model with Bag-of-n-grams (3,3) Word Representation and TruncatedSVD

[58]:

```
# Step 1: N-gram Representation
vectorizer = CountVectorizer(ngram_range=(3, 3))
X = vectorizer.fit_transform(flat_list_transcription)
y = df['encoded_target']

# Step 2: Imbalanced Data Handling
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 3: Dimensionality Reduction with TruncatedSVD
```

```
svd = TruncatedSVD(n_components=100, random_state=42)
X_svd = svd.fit_transform(X_resampled)

# Step 4: Splitting Data
X_train, X_test, y_train, y_test = train_test_split(X_svd, y_resampled,␣
 ↪test_size=0.2, random_state=42)

# Extract unique category names from the "medical_specialty" column
# Create a mapping from encoded target to category name
encoded_target_to_category = dict(zip(df['encoded_target'],␣
 ↪df['medical_specialty']))

# Ensure the category names are in the correct order of the encoded targets
category_names = [encoded_target_to_category[i] for i in␣
 ↪sorted(encoded_target_to_category)]

# Step 5: Model Building
models = {
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machine": SVC(),
    "XGBoost": XGBClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    report = classification_report(y_test, y_pred, target_names=category_names)
    print(f"Classification Report for {name}:\n{report}")
    print("\n" + "="*60 + "\n")
```

```
Classification Report for Random Forest:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.49      0.46      0.47        74
      ENT - Otolaryngology       0.65      0.57      0.61        77
          Gastroenterology       0.43      0.38      0.40        76
     Hematology - Oncology       0.63      0.58      0.61        79
                 Neurology       0.35      0.35      0.35        66
   Obstetrics / Gynecology       0.67      0.62      0.64        74
             Ophthalmology       0.63      0.87      0.73        55
                Orthopedic       0.56      0.58      0.57        79
     Pediatrics - Neonatal       0.67      0.48      0.56        71
   Psychiatry / Psychology       0.46      0.90      0.61        73
                 Radiology       0.20      0.16      0.18        87
                   Urology       0.63      0.42      0.51        80
```

```
                  accuracy                           0.52       891
                 macro avg       0.53      0.53      0.52       891
              weighted avg       0.53      0.52      0.51       891


============================================================

Classification Report for Support Vector Machine:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.57      0.41      0.47        74
      ENT - Otolaryngology       0.30      0.18      0.23        77
          Gastroenterology       0.29      0.13      0.18        76
     Hematology - Oncology       0.44      0.05      0.09        79
                 Neurology       0.44      0.21      0.29        66
   Obstetrics / Gynecology       1.00      0.18      0.30        74
             Ophthalmology       0.22      0.22      0.22        55
                Orthopedic       0.55      0.46      0.50        79
     Pediatrics - Neonatal       0.12      0.92      0.21        71
   Psychiatry / Psychology       0.78      0.10      0.17        73
                 Radiology       0.25      0.05      0.08        87
                   Urology       0.72      0.16      0.27        80

                  accuracy                           0.25       891
                 macro avg       0.47      0.25      0.25       891
              weighted avg       0.48      0.25      0.25       891


============================================================

Classification Report for XGBoost:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.49      0.46      0.48        74
      ENT - Otolaryngology       0.73      0.57      0.64        77
          Gastroenterology       0.43      0.42      0.42        76
     Hematology - Oncology       0.64      0.59      0.62        79
                 Neurology       0.34      0.30      0.32        66
   Obstetrics / Gynecology       0.67      0.59      0.63        74
             Ophthalmology       0.67      0.89      0.77        55
                Orthopedic       0.57      0.65      0.61        79
     Pediatrics - Neonatal       0.63      0.46      0.54        71
   Psychiatry / Psychology       0.45      0.92      0.61        73
                 Radiology       0.21      0.18      0.20        87
                   Urology       0.63      0.41      0.50        80

                  accuracy                           0.53       891
                 macro avg       0.54      0.54      0.53       891
             weighted avg       0.54      0.53      0.52       891


============================================================

Classification Report for Logistic Regression:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.65      0.58      0.61        74
      ENT - Otolaryngology       0.62      0.39      0.48        77
          Gastroenterology       0.46      0.30      0.37        76
     Hematology - Oncology       0.48      0.14      0.22        79
                 Neurology       0.49      0.32      0.39        66
   Obstetrics / Gynecology       0.76      0.43      0.55        74
             Ophthalmology       0.40      0.49      0.44        55
                Orthopedic       0.71      0.56      0.62        79
     Pediatrics - Neonatal       0.26      0.41      0.32        71
   Psychiatry / Psychology       0.25      0.99      0.40        73
                 Radiology       0.43      0.26      0.33        87
                   Urology       0.61      0.29      0.39        80

                  accuracy                           0.42       891
                 macro avg       0.51      0.43      0.43       891
              weighted avg       0.51      0.42      0.42       891


============================================================
```

#Advanced Model Development

#BioBERT (‘dmis-lab/biobert-v1.1’)

[59]:
```python
# Set up device for PyTorch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Split data into features and target
X = df['preprocessed_transcription']
y = df['medical_specialty']

# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Load BioBERT tokenizer
tokenizer = BertTokenizer.from_pretrained('dmis-lab/biobert-v1.1',␣
 ↪do_lower_case=True)
```

```
# Tokenize and encode sequences
encoded_data = tokenizer(X.tolist(), padding=True, truncation=True,␣
 ↪max_length=128, return_tensors='pt')

# Split data into train and test sets
train_inputs, test_inputs, train_labels, test_labels =␣
 ↪train_test_split(encoded_data['input_ids'],
                                                                        ␣
 ↪y_encoded,
                                                                        ␣
 ↪random_state=42,
                                                                        ␣
 ↪test_size=0.2,
                                                                        ␣
 ↪stratify=y_encoded)

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
train_inputs_resampled, train_labels_resampled = smote.
 ↪fit_resample(train_inputs, train_labels)

# Convert data to PyTorch tensors
train_inputs_tensor = torch.tensor(train_inputs_resampled)
test_inputs_tensor = torch.tensor(test_inputs)
train_labels_tensor = torch.tensor(train_labels_resampled)
test_labels_tensor = torch.tensor(test_labels)

# Create DataLoader for train and test sets
train_data = TensorDataset(train_inputs_tensor, train_labels_tensor)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=32)

test_data = TensorDataset(test_inputs_tensor, test_labels_tensor)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=32)

# Load BioBERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('dmis-lab/biobert-v1.1',␣
 ↪num_labels=len(label_encoder.classes_))
model.to(device)

# Set up optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
epochs = 4
total_steps = len(train_dataloader) * epochs
```

```
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,␣
 ↪num_training_steps=total_steps)

# Train the model
model.train()
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        batch = tuple(t.to(device) for t in batch)
        inputs = {'input_ids': batch[0],
                  'labels': batch[1]}
        optimizer.zero_grad()
        outputs = model(**inputs)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    avg_train_loss = total_loss / len(train_dataloader)
    print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss:␣
 ↪{avg_train_loss}')

# Evaluate the model
model.eval()
predictions, true_labels = [], []
for batch in test_dataloader:
    batch = tuple(t.to(device) for t in batch)
    inputs = {'input_ids': batch[0],
              'labels': batch[1]}
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    label_ids = inputs['labels'].cpu().numpy()
    predictions.append(logits)
    true_labels.append(label_ids)

predictions = np.concatenate(predictions, axis=0)
true_labels = np.concatenate(true_labels, axis=0)
predicted_labels = np.argmax(predictions, axis=1)
class_names = label_encoder.classes_

# Print classification report with method name included
print("Classification Report for BioBERT:")
print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names))
```

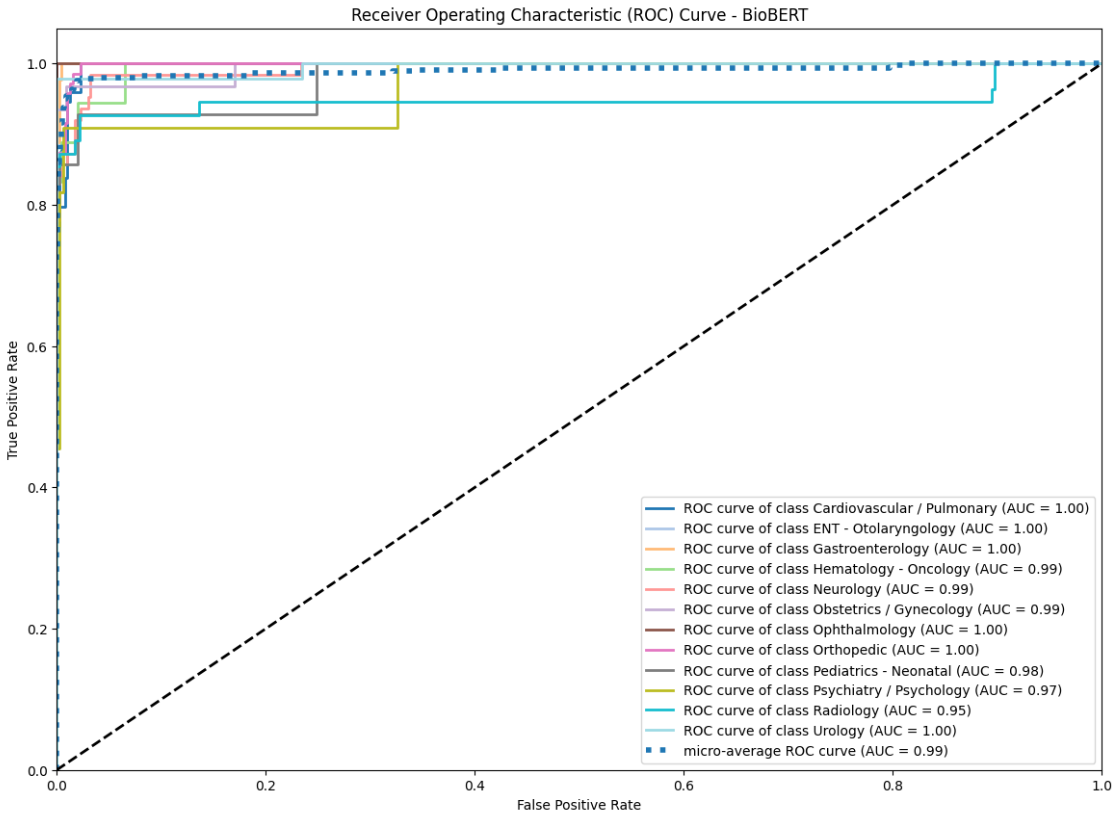
```
tokenizer_config.json:   0%|          | 0.00/49.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/462 [00:00<?, ?B/s]

pytorch_model.bin:   0%|          | 0.00/433M [00:00<?, ?B/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
We strongly recommend passing in an `attention_mask` since your input_ids may be
padded. See https://huggingface.co/docs/transformers/troubleshooting#incorrect-
output-when-padding-tokens-arent-masked.

Epoch 1/4, Average Training Loss: 1.6693773828446865
Epoch 2/4, Average Training Loss: 0.620629906787404
Epoch 3/4, Average Training Loss: 0.4752154874482325
Epoch 4/4, Average Training Loss: 0.4098977278252797
Classification Report for BioBERT:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.95      0.95      0.95        74
      ENT - Otolaryngology       1.00      1.00      1.00        19
          Gastroenterology       0.96      1.00      0.98        45
     Hematology - Oncology       0.88      0.83      0.86        18
                 Neurology       0.84      0.90      0.87        63
   Obstetrics / Gynecology       0.88      0.97      0.92        31
             Ophthalmology       1.00      1.00      1.00        17
                Orthopedic       0.90      0.93      0.92        71
     Pediatrics - Neonatal       0.92      0.86      0.89        14
   Psychiatry / Psychology       0.88      0.64      0.74        11
                 Radiology       0.98      0.85      0.91        55
                   Urology       0.98      0.98      0.98        47

                  accuracy                           0.93       465
                 macro avg       0.93      0.91      0.92       465
              weighted avg       0.93      0.93      0.93       465
```

#AUC and ROC Plot of BioBERT Model

[60]:

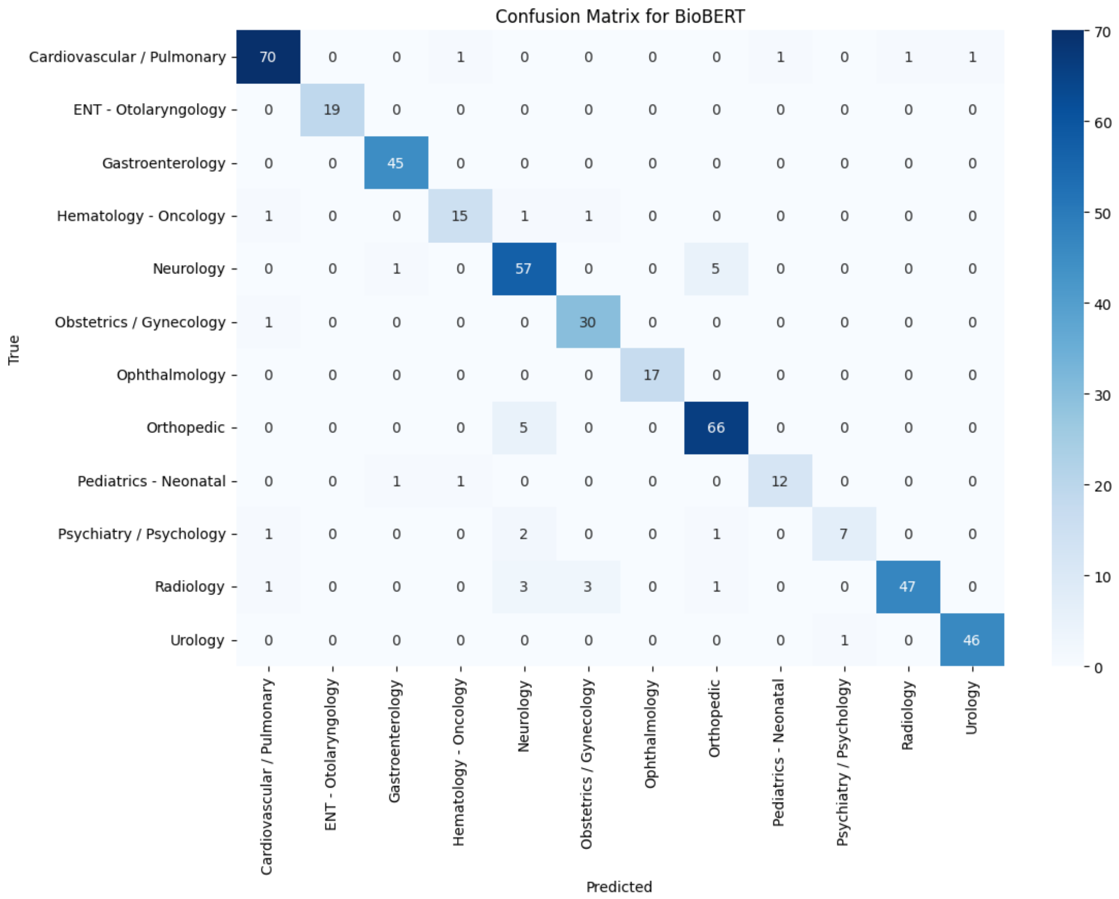
```
# Compute ROC curve and ROC area for each class
fpr = dict()
```

```
tpr = dict()
roc_auc = dict()
n_classes = len(class_names)  # Number of classes

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve((true_labels == i), predictions[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(label_binarize(true_labels,␣
 ↪classes=np.arange(n_classes)).ravel(),
                                          predictions.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# Plot all ROC curves
plt.figure(figsize=(14, 10))
colors = plt.cm.get_cmap('tab20', n_classes)

for i, color in zip(range(n_classes), colors.colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label=f'ROC curve of class {class_names[i]} (AUC = {roc_auc[i]:0.
 ↪2f})')

plt.plot(fpr["micro"], tpr["micro"], linestyle=':', linewidth=4,
         label=f'micro-average ROC curve (AUC = {roc_auc["micro"]:0.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - BioBERT')
plt.legend(loc="lower right")
plt.show()
```



#Confusion Matrix of BioBERT Model

[61]:
```
# Calculate the confusion matrix
conf_matrix = confusion_matrix(true_labels, predicted_labels)

# Plot the confusion matrix
plt.figure(figsize=(12, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',␣
 ↪xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for BioBERT')
plt.show()
```



#K-Fold Cross-Validation of BioBERT Model

[62]:
```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
kfold_scores = []

for fold, (train_index, val_index) in enumerate(kfold.
 ↪split(encoded_data['input_ids'], y_encoded)):
    print(f'Fold {fold + 1}/{kfold.n_splits}')

    # Split data into train and validation sets
    train_inputs_fold = encoded_data['input_ids'][train_index]
    train_labels_fold = y_encoded[train_index]
    val_inputs_fold = encoded_data['input_ids'][val_index]
    val_labels_fold = y_encoded[val_index]

    # Convert to PyTorch tensors
    train_inputs_tensor = torch.tensor(train_inputs_fold)
    val_inputs_tensor = torch.tensor(val_inputs_fold)
    train_labels_tensor = torch.tensor(train_labels_fold)
```

```python
    val_labels_tensor = torch.tensor(val_labels_fold)

    # Create DataLoader for train and validation sets
    train_data_fold = TensorDataset(train_inputs_tensor, train_labels_tensor)
    train_sampler_fold = RandomSampler(train_data_fold)
    train_dataloader_fold = DataLoader(train_data_fold, sampler=train_sampler_fold, batch_size=32)

    val_data_fold = TensorDataset(val_inputs_tensor, val_labels_tensor)
    val_sampler_fold = SequentialSampler(val_data_fold)
    val_dataloader_fold = DataLoader(val_data_fold, sampler=val_sampler_fold, batch_size=32)

    # Reload model for each fold
    model = BertForSequenceClassification.from_pretrained('dmis-lab/biobert-v1.1', num_labels=len(label_encoder.classes_))
    model.to(device)
    optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
    scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

    # Training loop
    model.train()
    for epoch in range(epochs):
        total_loss = 0
        for batch in train_dataloader_fold:
            batch = tuple(t.to(device) for t in batch)
            inputs = {'input_ids': batch[0],
                      'labels': batch[1]}
            optimizer.zero_grad()
            outputs = model(**inputs)
            loss = outputs.loss
            total_loss += loss.item()
            loss.backward()
            torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            optimizer.step()
            scheduler.step()

        avg_train_loss = total_loss / len(train_dataloader_fold)
        print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss: {avg_train_loss}')

    # Evaluation loop
    model.eval()
    predictions, true_labels = [], []
    for batch in val_dataloader_fold:
        batch = tuple(t.to(device) for t in batch)
        inputs = {'input_ids': batch[0],
                  'labels': batch[1]}
        with torch.no_grad():
            outputs = model(**inputs)
        logits = outputs.logits
        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
        true_labels.append(label_ids)

    predictions = np.concatenate(predictions, axis=0)
    true_labels = np.concatenate(true_labels, axis=0)
    predicted_labels = np.argmax(predictions, axis=1)

    # Calculate metrics for this fold
    fold_report = classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names, output_dict=True)
    kfold_scores.append(fold_report)

    # Print fold metrics
    print("Classification Report for Fold:")
    print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names))
    print("\n" + "="*60 + "\n")

# Aggregate K-Fold results
avg_metrics = {
    'precision': np.mean([fold['weighted avg']['precision'] for fold in␣
 ↪kfold_scores]),
    'recall': np.mean([fold['weighted avg']['recall'] for fold in␣
 ↪kfold_scores]),
    'f1-score': np.mean([fold['weighted avg']['f1-score'] for fold in␣
 ↪kfold_scores])
}

print("Average Metrics across all Folds:")
print(avg_metrics)
```

```
Fold 1/5
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
Epoch 1/4, Average Training Loss: 1.893898162801387
Epoch 2/4, Average Training Loss: 0.6543802331059666
Epoch 3/4, Average Training Loss: 0.32252542409351315
Epoch 4/4, Average Training Loss: 0.1966225555387594
Classification Report for Fold:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.94      0.99      0.96        74
      ENT - Otolaryngology       1.00      1.00      1.00        14
          Gastroenterology       1.00      0.91      0.95        44
     Hematology - Oncology       0.83      0.90      0.86        21
                 Neurology       0.82      0.86      0.84        63
   Obstetrics / Gynecology       0.94      0.97      0.95        32
             Ophthalmology       1.00      1.00      1.00        16
                Orthopedic       0.91      0.92      0.92        65
     Pediatrics - Neonatal       1.00      0.73      0.84        11
   Psychiatry / Psychology       0.93      0.81      0.87        16
                 Radiology       0.97      0.94      0.96        70
                   Urology       0.97      0.97      0.97        39

                  accuracy                           0.93       465
                 macro avg       0.94      0.92      0.93       465
              weighted avg       0.93      0.93      0.93       465


============================================================

Fold 2/5

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 1.7774644362724434
Epoch 2/4, Average Training Loss: 0.6324314500315714
Epoch 3/4, Average Training Loss: 0.31388682139627005
Epoch 4/4, Average Training Loss: 0.2173106681744931
Classification Report for Fold:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.97      0.99      0.98        70
      ENT - Otolaryngology       0.95      0.95      0.95        21
          Gastroenterology       0.95      0.97      0.96        36
     Hematology - Oncology       0.93      0.87      0.90        15
                 Neurology       0.91      0.89      0.90        65
   Obstetrics / Gynecology       0.91      0.97      0.94        33
             Ophthalmology       1.00      1.00      1.00        19
                Orthopedic       0.97      0.96      0.97        73
     Pediatrics - Neonatal       0.81      0.81      0.81        16
   Psychiatry / Psychology       0.86      0.75      0.80         8
                 Radiology       0.95      0.94      0.95        66
                   Urology       0.98      1.00      0.99        43

                  accuracy                           0.95       465
                 macro avg       0.93      0.92      0.93       465
              weighted avg       0.95      0.95      0.95       465


============================================================

Fold 3/5

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 1.8254361991154946
Epoch 2/4, Average Training Loss: 0.6619001639596487
Epoch 3/4, Average Training Loss: 0.33570041166523756
Epoch 4/4, Average Training Loss: 0.21859307675543477
Classification Report for Fold:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.96      0.93      0.95        74
      ENT - Otolaryngology       0.93      1.00      0.96        26
          Gastroenterology       0.96      0.93      0.94        54
     Hematology - Oncology       0.88      0.88      0.88        17
                 Neurology       0.89      0.96      0.92        68
   Obstetrics / Gynecology       0.92      0.96      0.94        23
             Ophthalmology       0.94      1.00      0.97        15
                Orthopedic       0.95      0.95      0.95        66
     Pediatrics - Neonatal       0.81      0.76      0.79        17
   Psychiatry / Psychology       0.71      0.71      0.71         7
                 Radiology       0.94      0.88      0.91        50
                   Urology       0.96      0.94      0.95        48

                  accuracy                           0.93       465
                 macro avg       0.90      0.91      0.91       465
              weighted avg       0.93      0.93      0.93       465


============================================================

Fold 4/5

Some weights of BertForSequenceClassification were not initialized from the
```

```
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 1.7640899692551564
Epoch 2/4, Average Training Loss: 0.6257501029867237
Epoch 3/4, Average Training Loss: 0.29930050014439274
Epoch 4/4, Average Training Loss: 0.19988448640047493
Classification Report for Fold:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.91      0.96      0.94        75
      ENT - Otolaryngology       1.00      0.89      0.94        19
          Gastroenterology       0.93      0.97      0.95        38
     Hematology - Oncology       0.95      0.78      0.86        23
                 Neurology       0.84      0.92      0.88        64
   Obstetrics / Gynecology       0.96      0.93      0.95        28
             Ophthalmology       1.00      1.00      1.00        16
                Orthopedic       0.93      0.90      0.91        71
     Pediatrics - Neonatal       0.79      0.85      0.81        13
   Psychiatry / Psychology       0.82      0.82      0.82        11
                 Radiology       0.97      0.89      0.93        44
                   Urology       0.97      0.97      0.97        63

                  accuracy                           0.92       465
                 macro avg       0.92      0.91      0.91       465
              weighted avg       0.93      0.92      0.92       465


============================================================

Fold 5/5

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at dmis-lab/biobert-v1.1 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 1.677667657197532
Epoch 2/4, Average Training Loss: 0.5729492114762128
Epoch 3/4, Average Training Loss: 0.3026561723169634
Epoch 4/4, Average Training Loss: 0.20843907527751843
Classification Report for Fold:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.92      0.92      0.92        78
      ENT - Otolaryngology       0.94      0.94      0.94        16
```

```
          Gastroenterology       0.98      0.98      0.98        52
     Hematology - Oncology       0.81      0.93      0.87        14
                 Neurology       0.83      0.84      0.83        57
   Obstetrics / Gynecology       0.97      0.95      0.96        39
             Ophthalmology       0.94      0.94      0.94        17
                Orthopedic       0.88      0.96      0.92        80
     Pediatrics - Neonatal       0.75      0.46      0.57        13
   Psychiatry / Psychology       1.00      0.64      0.78        11
                 Radiology       0.90      0.86      0.88        43
                   Urology       0.98      1.00      0.99        44

                  accuracy                           0.91       464
                 macro avg       0.91      0.87      0.88       464
              weighted avg       0.91      0.91      0.91       464


============================================================

Average Metrics across all Folds:
{'precision': 0.9287861952034742, 'recall': 0.9277039302929182, 'f1-score':
0.927142847241272}
```

#Misclassification Analysis of BioBERT Model

[63]:

```
# Calculate the number of entries for each class
class_counts = np.bincount(true_labels)

# Calculate the number of misclassifications for each class
misclassified_counts = np.bincount(true_labels[true_labels !=␣
 ↪predicted_labels], minlength=len(class_names))

# Calculate accuracy for each class
accuracy_per_class = (class_counts - misclassified_counts) / class_counts * 100

# Create a DataFrame to display the results
misclassification_analysis_df = pd.DataFrame({
    'Medical Specialties': class_names,
    'Number of Entries': class_counts,
    'Number of  BioBERT Misclassifications Errors': misclassified_counts,
    'Accuracy (%)': accuracy_per_class
})

# Display the DataFrame
print(misclassification_analysis_df)
```

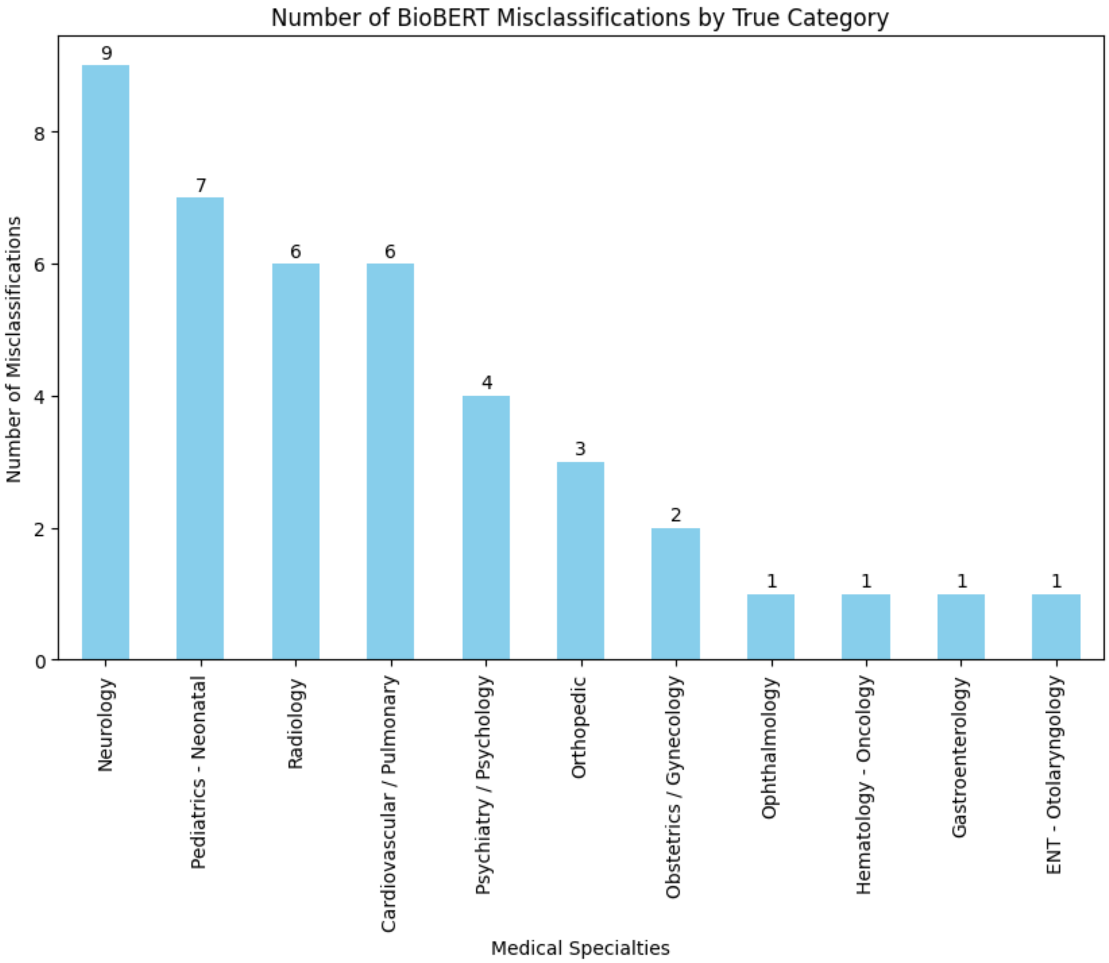
```
          Medical Specialties  Number of Entries  \
0   Cardiovascular / Pulmonary                 78
1         ENT - Otolaryngology                 16
```

```
2              Gastroenterology                 52
3         Hematology - Oncology                 14
4                     Neurology                 57
5       Obstetrics / Gynecology                 39
6                 Ophthalmology                 17
7                    Orthopedic                 80
8         Pediatrics - Neonatal                 13
9       Psychiatry / Psychology                 11
10                    Radiology                 43
11                      Urology                 44

    Number of  BioBERT Misclassifications Errors  Accuracy (%)
0                                              6     92.307692
1                                              1     93.750000
2                                              1     98.076923
3                                              1     92.857143
4                                              9     84.210526
5                                              2     94.871795
6                                              1     94.117647
7                                              3     96.250000
8                                              7     46.153846
9                                              4     63.636364
10                                             6     86.046512
11                                             0    100.000000
```

#Number of BioBERT Misclassifications by True Category

[64]:

```python
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()
    return probs

# Select misclassified examples
misclassified_indices = np.where(predicted_labels != true_labels)[0]
misclassified_texts = df['preprocessed_transcription'].
  ↪iloc[misclassified_indices].tolist()
misclassified_true_labels = true_labels[misclassified_indices]
misclassified_predicted_labels = predicted_labels[misclassified_indices]

# Convert misclassified labels to their category names
misclassified_true_categories = [class_names[label] for label in␣
  ↪misclassified_true_labels]
misclassified_predicted_categories = [class_names[label] for label in␣
  ↪misclassified_predicted_labels]

# Bar graph of misclassified categories
misclassified_counts = pd.Series(misclassified_true_categories).value_counts()

# Plotting
plt.figure(figsize=(10, 6))
bars = misclassified_counts.plot(kind='bar', color='skyblue')
plt.title('Number of BioBERT Misclassifications by True Category')
plt.xlabel('Medical Specialties')
plt.ylabel('Number of Misclassifications')

# Add numbers on top of the bars
for bar in bars.patches:
    plt.text(
        bar.get_x() + bar.get_width() / 2,
        bar.get_height() + 0.05,
        int(bar.get_height()),
        ha='center',
        va='bottom',
        color='black',
        fontsize=10
    )

plt.show()
```



#Explainable AI (XAI) Technique

#Integration of XAI with BioBERT Model

[65]:
```
pip install shap
```

```
Collecting shap
  Downloading shap-0.46.0-cp310-cp310-
manylinux_2_12_x86_64.manylinux2010_x86_64.manylinux_2_17_x86_64.manylinux2014_x
86_64.whl (540 kB)
                            540.1/540.1

kB 7.3 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-
packages (from shap) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from shap) (1.11.4)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-
packages (from shap) (1.2.2)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from shap) (2.0.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-
packages (from shap) (4.66.4)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-
packages (from shap) (24.1)
Collecting slicer==0.0.8 (from shap)
  Downloading slicer-0.0.8-py3-none-any.whl (15 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages
(from shap) (0.58.1)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-
packages (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.42,>=0.41.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba->shap) (0.41.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->shap) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->shap) (2024.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn->shap) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn->shap) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.8.2->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.46.0 slicer-0.0.8
```

[66]:

```python
# SHAP library for explainable AI
import shap  # For SHapley Additive exPlanations
```

#SHAP Visualization of Correctly Classified Examples

#SHAP Visualization of Sample Index ‘0’

[67]:

```python
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )
    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 0
single_input = df['preprocessed_transcription'].iloc[sample_index]

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the selected input
shap_values = explainer([single_input])

# Map predicted labels to category names
predicted_label = np.argmax(predict_proba([single_input])[0])
predicted_category = class_names[predicted_label]

# Display true medical specialty (target variable) and predicted category
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category)

# Visualize the results
shap.initjs()
shap.plots.text(shap_values[0])
```

```
  0%|          | 0/498 [00:00<?, ?it/s]
```

```
PartitionExplainer explainer: 2it [00:15, 15.50s/it]
```

```
True Medical Specialty:
Cardiovascular / Pulmonary

Predicted Category:
Cardiovascular / Pulmonary
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#SHAP Visualization of Sample Index ‘1’

[68]:
```python
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 1
single_input = df['preprocessed_transcription'].iloc[sample_index]

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the selected input
shap_values = explainer([single_input])

# Map predicted labels to category names
predicted_label = np.argmax(predict_proba([single_input])[0])
predicted_category = class_names[predicted_label]

# Display true medical specialty (target variable) and predicted category
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category)

# Visualize the results
shap.initjs()
shap.plots.text(shap_values[0])
```

```
Token indices sequence length is longer than the specified maximum sequence
length for this model (525 > 512). Running this sequence through the model will
result in indexing errors
```

```
  0%|          | 0/498 [00:00<?, ?it/s]
```

```
PartitionExplainer explainer: 2it [00:20, 20.50s/it]
```

```
True Medical Specialty:
Cardiovascular / Pulmonary

Predicted Category:
Cardiovascular / Pulmonary
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#SHAP Visualization of Sample Index ‘825’

[69]:

```python
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 825
single_input = df['preprocessed_transcription'].iloc[sample_index]
```

```
# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the selected input
shap_values = explainer([single_input])

# Map predicted labels to category names
predicted_label = np.argmax(predict_proba([single_input])[0])
predicted_category = class_names[predicted_label]

# Display true medical specialty (target variable) and predicted category
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category)

# Visualize the results
shap.initjs()
shap.plots.text(shap_values[0])
```

```
  0%|          | 0/498 [00:00<?, ?it/s]
```

```
PartitionExplainer explainer: 2it [00:13, 13.13s/it]
```

```
True Medical Specialty:
Orthopedic

Predicted Category:
Orthopedic
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#SHAP Visualization of Sample Index ‘116’

[70]:

```
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )
    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 116
single_input = df['preprocessed_transcription'].iloc[sample_index]

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the selected input
shap_values = explainer([single_input])

# Map predicted labels to category names
predicted_label = np.argmax(predict_proba([single_input])[0])
predicted_category = class_names[predicted_label]

# Display true medical specialty (target variable) and predicted category
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category)

# Visualize the results
shap.initjs()
shap.plots.text(shap_values[0])
```

```
  0%|          | 0/498 [00:00<?, ?it/s]

PartitionExplainer explainer: 2it [00:11, 11.57s/it]

True Medical Specialty:
Urology

Predicted Category:
Urology


<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

1 SHAP Visualization of Misclassified Sample Index '263'

[71]:
```
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 263
single_input = df['preprocessed_transcription'].iloc[sample_index]

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the selected input
shap_values = explainer([single_input])

# Map predicted labels to category names
predicted_label = np.argmax(predict_proba([single_input])[0])
predicted_category = class_names[predicted_label]

# Display true medical specialty (target variable) and predicted category
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category)

# Visualize the results
shap.initjs()
shap.plots.text(shap_values[0])
```

```
  0%|          | 0/498 [00:00<?, ?it/s]
```

```
True Medical Specialty:
Radiology

Predicted Category:
Orthopedic
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#Perturbation Testing of SHAP XAI Technique

#Removing the Most Important Words Identified by SHAP

[72]:

```
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 1
single_input = df['preprocessed_transcription'].iloc[sample_index]

# Words to remove based on SHAP output with commas included
words_to_remove = [
    "cardiovascular,", "pulmonary,", "doppler,", "echocardiogram,", "annular,",
    "aortic,", "root,", "aortic,", "valve,", "atrial,", "atrium,",␣
  ↪"calcification,",
    "cavity,", "ejection,", "fraction,", "mitral,", "obliteration,", "outflow,",
    "regurgitation,", "relaxation,", "pattern,", "stenosis,", "systolic,",␣
  ↪"function,",
    "tricuspid,", "valve,", "ventricular,", "ventricular,", "cavity,", "wall,",
    "motion,", "pulmonary,", "left,", "ventricular,", "cavity,", "size,",␣
  ↪"wall,",
    "thickness,", "appear,", "normal,", "wall,", "motion,", "left,",␣
  ↪"ventricular,",
    "systolic,", "function,", "appears,", "hyperdynamic,", "estimated,",␣
  ↪"ejection,",
    "fraction,", "near,", "cavity,", "obliteration,"
]

# Remove the words from the input text
modified_input = single_input
for word in words_to_remove:
    modified_input = modified_input.replace(word, '')

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)

# Explain the prediction for the modified input
shap_values_modified = explainer([modified_input])

# Map predicted labels to category names for modified input
predicted_label_modified = np.argmax(predict_proba([modified_input])[0])
predicted_category_modified = class_names[predicted_label_modified]

# Display true medical specialty (target variable) and predicted category for␣
  ↪modified input
print("\nAfter Removing the Most Important Words:")
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category_modified)

# Visualize the results for modified input
shap.initjs()
shap.plots.text(shap_values_modified[0])
```

```
  0%|          | 0/498 [00:00<?, ?it/s]
```

```
After Removing the Most Important Words:
True Medical Specialty:
Cardiovascular / Pulmonary
```

```
Predicted Category:
Cardiovascular / Pulmonary
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#Adding Noisy Words

[73]:

```python
# Define the class names
class_names = label_encoder.classes_

# Create a predict function for SHAP
def predict_proba(texts):
    encoded_inputs = tokenizer.batch_encode_plus(
        texts,
        max_length=128,
        padding=True,
        truncation=True,
        return_tensors='pt'
    )

    input_ids = encoded_inputs['input_ids'].to(device)
    attention_mask = encoded_inputs['attention_mask'].to(device)

    with torch.no_grad():
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probs = torch.nn.functional.softmax(logits, dim=1).cpu().numpy()

    return probs

# Select a single input from the test data
sample_index = 1
single_input = df['preprocessed_transcription'].iloc[sample_index]

# Words to remove based on SHAP output with commas included
words_to_remove = [
    "cardiovascular,", "pulmonary,", "doppler,", "echocardiogram,", "annular,",
    "aortic,", "root,", "aortic,", "valve,", "atrial,", "atrium,",␣
  ↪"calcification,",
    "cavity,", "ejection,", "fraction,", "mitral,", "obliteration,", "outflow,",
    "regurgitation,", "relaxation,", "pattern,", "stenosis,", "systolic,",␣
  ↪"function,",
    "tricuspid,", "valve,", "ventricular,", "ventricular,", "cavity,", "wall,",
    "motion,", "pulmonary,", "left,", "ventricular,", "cavity,", "size,",␣
  ↪"wall,",
    "thickness,", "appear,", "normal,", "wall,", "motion,", "left,",␣
  ↪"ventricular,",
    "systolic,", "function,", "appears,", "hyperdynamic,", "estimated,",␣
 ↪"ejection,",
    "fraction,", "near,", "cavity,", "obliteration,"
]

# Remove the words from the input text
modified_input = single_input
for word in words_to_remove:
    modified_input = modified_input.replace(word, '')

# Define a function to add noise to the text
def add_noise(text):
    # Domain-specific irrelevant terms
    irrelevant_terms = ["aspirin", "antibiotic", "bandage", "nausea",␣
 ↪"headache"]

    # Split text into words
    words = text.split()

    # List to store noisy words
    added_noisy_words = []

    # Introduce random noise: add irrelevant terms and slight misspellings
    noisy_words = []
    for word in words:
        if random.random() < 0.1:  # 10% chance to add an irrelevant term
            new_word = random.choice(irrelevant_terms)
            noisy_words.append(new_word)
            added_noisy_words.append(new_word)
        if random.random() < 0.1:  # 10% chance to slightly alter the word
            new_word = word[:-1] + random.choice('abcdefghijklmnopqrstuvwxyz')
            noisy_words.append(new_word)
            added_noisy_words.append(new_word)
        else:
            noisy_words.append(word)

    # Join words back into a string
    noisy_text = ' '.join(noisy_words)

    return noisy_text, added_noisy_words

# Add noise to the modified input
noisy_modified_input, added_noisy_words = add_noise(modified_input)

# SHAP Explainer expects list of strings for transformer models
explainer = shap.Explainer(predict_proba, tokenizer, output_names=class_names)
```

```
# Explain the prediction for the noisy modified input
shap_values_noisy_modified = explainer([noisy_modified_input])

# Map predicted labels to category names for noisy modified input
predicted_label_noisy_modified = np.
 ↪argmax(predict_proba([noisy_modified_input])[0])
predicted_category_noisy_modified = class_names[predicted_label_noisy_modified]

# Display true medical specialty (target variable) and predicted category for␣
 ↪noisy modified input
print("\nAfter Adding Noise to the Input:")
print("True Medical Specialty:")
print(df['medical_specialty'].iloc[sample_index])
print("\nPredicted Category:")
print(predicted_category_noisy_modified)

# Display the noisy words added
print("\nNoisy Words Added:")
print(added_noisy_words)

# Visualize the results for noisy modified input
shap.initjs()
shap.plots.text(shap_values_noisy_modified[0])
```

```
After Adding Noise to the Input:
True Medical Specialty:
Cardiovascular / Pulmonary

Predicted Category:
Cardiovascular / Pulmonary

Noisy Words Added:
['alsor', 'increasedq', 'nausea', 'antibiotic', 'consistentp', 'nausea',
'wellx', 'headache', 'bandage', 'mmp', 'supportivey', 'structuref', 'nausea',
'nausea', 'aspirin', 'aspirin', 'missedc', 'studz']
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

#ClinicalBERT (‘emilyalsentzer/Bio_ClinicalBERT’)

```
[74]: # Set up device for PyTorch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Split data into features and target
X = df['preprocessed_transcription']
y = df['medical_specialty']
```

```
# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Load Clinical BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('emilyalsentzer/Bio_ClinicalBERT',␣
 ↪do_lower_case=True)

# Tokenize and encode sequences
encoded_data = tokenizer(X.tolist(), padding=True, truncation=True,␣
 ↪max_length=128, return_tensors='pt')

# Split data into train and test sets
train_inputs, test_inputs, train_labels, test_labels =␣
 ↪train_test_split(encoded_data['input_ids'],
                                                                        ␣
 ↪y_encoded,
                                                                        ␣
 ↪random_state=42,
                                                                        ␣
 ↪test_size=0.2,
                                                                        ␣
 ↪stratify=y_encoded)

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
train_inputs_resampled, train_labels_resampled = smote.
 ↪fit_resample(train_inputs, train_labels)

# Convert data to PyTorch tensors
train_inputs_tensor = torch.tensor(train_inputs_resampled)
test_inputs_tensor = torch.tensor(test_inputs)
train_labels_tensor = torch.tensor(train_labels_resampled)
test_labels_tensor = torch.tensor(test_labels)

# Create DataLoader for train and test sets
train_data = TensorDataset(train_inputs_tensor, train_labels_tensor)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=32)

test_data = TensorDataset(test_inputs_tensor, test_labels_tensor)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=32)

# Load Clinical BERT model for sequence classification
```

```
model = BertForSequenceClassification.from_pretrained('emilyalsentzer/
 ↪Bio_ClinicalBERT', num_labels=len(label_encoder.classes_))
model.to(device)

# Set up optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
epochs = 4
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,␣
 ↪num_training_steps=total_steps)

# Train the model
model.train()
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        batch = tuple(t.to(device) for t in batch)
        inputs = {'input_ids': batch[0],
                  'labels': batch[1]}
        optimizer.zero_grad()
        outputs = model(**inputs)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    avg_train_loss = total_loss / len(train_dataloader)
    print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss:␣
 ↪{avg_train_loss}')

# Evaluate the model
model.eval()
predictions, true_labels = [], []
for batch in test_dataloader:
    batch = tuple(t.to(device) for t in batch)
    inputs = {'input_ids': batch[0],
              'labels': batch[1]}
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    label_ids = inputs['labels'].cpu().numpy()
    predictions.append(logits)
    true_labels.append(label_ids)
```

```
predictions = np.concatenate(predictions, axis=0)
true_labels = np.concatenate(true_labels, axis=0)
predicted_labels = np.argmax(predictions, axis=1)
class_names = label_encoder.classes_

# Print classification report with method name included
print("Classification Report for ClinicalBERT:")
print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names))
```

```
vocab.txt:   0%|          | 0.00/213k [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/385 [00:00<?, ?B/s]
```

```
pytorch_model.bin:   0%|          | 0.00/436M [00:00<?, ?B/s]
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at emilyalsentzer/Bio_ClinicalBERT and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
Epoch 1/4, Average Training Loss: 1.8036292142101698
Epoch 2/4, Average Training Loss: 0.6997512030814376
Epoch 3/4, Average Training Loss: 0.5137819602553334
Epoch 4/4, Average Training Loss: 0.4420144919838224
Classification Report for ClinicalBERT:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.94      0.91      0.92        74
      ENT - Otolaryngology       1.00      0.95      0.97        19
          Gastroenterology       0.92      1.00      0.96        45
     Hematology - Oncology       0.94      0.89      0.91        18
                 Neurology       0.84      0.89      0.86        63
   Obstetrics / Gynecology       0.86      0.97      0.91        31
             Ophthalmology       1.00      0.94      0.97        17
                Orthopedic       0.93      0.94      0.94        71
     Pediatrics - Neonatal       0.79      0.79      0.79        14
   Psychiatry / Psychology       0.89      0.73      0.80        11
                 Radiology       0.94      0.85      0.90        55
                   Urology       0.96      0.96      0.96        47

                  accuracy                           0.92       465
                 macro avg       0.92      0.90      0.91       465
              weighted avg       0.92      0.92      0.92       465
```

#BERT (‘bert-base-uncased’)

[75]:

```
# Set up device for PyTorch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Split data into features and target
X = df['preprocessed_transcription']
y = df['medical_specialty']

# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Load BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',␣
 ↪do_lower_case=True)

# Tokenize and encode sequences
encoded_data = tokenizer(X.tolist(), padding=True, truncation=True,␣
 ↪max_length=128, return_tensors='pt')

# Split data into train and test sets
train_inputs, test_inputs, train_labels, test_labels =␣
 ↪train_test_split(encoded_data['input_ids'],
                                                                                     ␣
 ↪y_encoded,
                                                                                     ␣
 ↪random_state=42,
                                                                                     ␣
 ↪test_size=0.2,
                                                                                     ␣
 ↪stratify=y_encoded)

# Convert data to PyTorch tensors
train_inputs_tensor = torch.tensor(train_inputs)
test_inputs_tensor = torch.tensor(test_inputs)
train_labels_tensor = torch.tensor(train_labels)
test_labels_tensor = torch.tensor(test_labels)

# Create DataLoader for train and test sets
train_data = TensorDataset(train_inputs_tensor, train_labels_tensor)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=32)

test_data = TensorDataset(test_inputs_tensor, test_labels_tensor)
test_sampler = SequentialSampler(test_data)
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=32)
```

```
# Load BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',␣
 ↪num_labels=len(label_encoder.classes_))
model.to(device)

# Set up optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
epochs = 4
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0,␣
 ↪num_training_steps=total_steps)

# Train the model
model.train()
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        batch = tuple(t.to(device) for t in batch)
        inputs = {'input_ids': batch[0],
                  'labels': batch[1]}
        optimizer.zero_grad()
        outputs = model(**inputs)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    avg_train_loss = total_loss / len(train_dataloader)
    print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss:␣
 ↪{avg_train_loss}')

# Evaluate the model
model.eval()
predictions, true_labels = [], []
for batch in test_dataloader:
    batch = tuple(t.to(device) for t in batch)
    inputs = {'input_ids': batch[0],
              'labels': batch[1]}
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    label_ids = inputs['labels'].cpu().numpy()
    predictions.append(logits)
    true_labels.append(label_ids)
```

```python
predictions = np.concatenate(predictions, axis=0)
true_labels = np.concatenate(true_labels, axis=0)
predicted_labels = np.argmax(predictions, axis=1)
class_names = label_encoder.classes_

# Print classification report with method name included
print("Classification Report for BERT:")
print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names))
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/570 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 2.0324573436025846
Epoch 2/4, Average Training Loss: 0.8897511072583117
Epoch 3/4, Average Training Loss: 0.5055445887274661
Epoch 4/4, Average Training Loss: 0.37606183527889897
Classification Report for BERT:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.94      0.97      0.95        74
      ENT - Otolaryngology       1.00      0.95      0.97        19
          Gastroenterology       0.91      0.96      0.93        45
     Hematology - Oncology       0.79      0.83      0.81        18
                 Neurology       0.81      0.87      0.84        63
   Obstetrics / Gynecology       0.88      0.97      0.92        31
             Ophthalmology       1.00      0.94      0.97        17
                Orthopedic       0.91      0.94      0.92        71
     Pediatrics - Neonatal       0.92      0.79      0.85        14
   Psychiatry / Psychology       1.00      0.73      0.84        11
                 Radiology       0.98      0.85      0.91        55
                   Urology       0.98      0.91      0.95        47

                  accuracy                           0.91       465
                 macro avg       0.93      0.89      0.91       465
              weighted avg       0.92      0.91      0.91       465
```

#RoBERTa (‘roberta-base’)

[76]:
```
# Set up device for PyTorch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Split data into features and target
X = df['preprocessed_transcription']
y = df['medical_specialty']

# Encode target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Load RoBERTa tokenizer
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')

# Tokenize and encode sequences
encoded_data = tokenizer(X.tolist(), padding=True, truncation=True,␣
 ↪max_length=128, return_tensors='pt')

# Split data into train and test sets
train_inputs, test_inputs, train_labels, test_labels =␣
 ↪train_test_split(encoded_data['input_ids'],
                                                                               ␣
 ↪y_encoded,
                                                                               ␣
 ↪random_state=42,
                                                                               ␣
 ↪test_size=0.2,
                                                                               ␣
 ↪stratify=y_encoded)

# Convert data to PyTorch tensors
train_inputs_tensor = torch.tensor(train_inputs)
test_inputs_tensor = torch.tensor(test_inputs)
train_labels_tensor = torch.tensor(train_labels)
test_labels_tensor = torch.tensor(test_labels)

# Create DataLoader for train and test sets
train_data = TensorDataset(train_inputs_tensor, train_labels_tensor)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=32)

test_data = TensorDataset(test_inputs_tensor, test_labels_tensor)
test_sampler = SequentialSampler(test_data)
```

```
test_dataloader = DataLoader(test_data, sampler=test_sampler, batch_size=32)

# Load RoBERTa model for sequence classification
model = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=len(label_encoder.classes_))
model.to(device)

# Set up optimizer and scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
epochs = 4
total_steps = len(train_dataloader) * epochs
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

# Train the model
model.train()
for epoch in range(epochs):
    total_loss = 0
    for batch in train_dataloader:
        batch = tuple(t.to(device) for t in batch)
        inputs = {'input_ids': batch[0],
                  'labels': batch[1]}
        optimizer.zero_grad()
        outputs = model(**inputs)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        optimizer.step()
        scheduler.step()

    avg_train_loss = total_loss / len(train_dataloader)
    print(f'Epoch {epoch + 1}/{epochs}, Average Training Loss: {avg_train_loss}')

# Evaluate the model
model.eval()
predictions, true_labels = [], []
for batch in test_dataloader:
    batch = tuple(t.to(device) for t in batch)
    inputs = {'input_ids': batch[0],
              'labels': batch[1]}
    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    logits = logits.detach().cpu().numpy()
    label_ids = inputs['labels'].cpu().numpy()
    predictions.append(logits)
    true_labels.append(label_ids)

predictions = np.concatenate(predictions, axis=0)
true_labels = np.concatenate(true_labels, axis=0)
predicted_labels = np.argmax(predictions, axis=1)
class_names = label_encoder.classes_

# Print classification report with method name included
print("Classification Report for RoBERTa:")
print(classification_report(true_labels, predicted_labels,␣
 ↪target_names=class_names))
```

```
tokenizer_config.json:   0%|          | 0.00/25.0 [00:00<?, ?B/s]

vocab.json:   0%|          | 0.00/899k [00:00<?, ?B/s]

merges.txt:   0%|          | 0.00/456k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.36M [00:00<?, ?B/s]

config.json:   0%|          | 0.00/481 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/499M [00:00<?, ?B/s]

Some weights of RobertaForSequenceClassification were not initialized from the
model checkpoint at roberta-base and are newly initialized:
['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias',
'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Epoch 1/4, Average Training Loss: 1.8289603065636197
Epoch 2/4, Average Training Loss: 0.542975342374737
Epoch 3/4, Average Training Loss: 0.29224961591979204
Epoch 4/4, Average Training Loss: 0.21937284069293636
Classification Report for RoBERTa:
                            precision    recall  f1-score   support

Cardiovascular / Pulmonary       0.95      0.96      0.95        74
      ENT - Otolaryngology       1.00      1.00      1.00        19
          Gastroenterology       0.94      1.00      0.97        45
     Hematology - Oncology       0.89      0.89      0.89        18
                 Neurology       0.82      0.87      0.85        63
   Obstetrics / Gynecology       0.88      0.97      0.92        31
             Ophthalmology       1.00      0.94      0.97        17
                Orthopedic       0.91      0.94      0.92        71
     Pediatrics - Neonatal       0.92      0.79      0.85        14
    Psychiatry / Psychology       0.89      0.73      0.80        11
                 Radiology       0.98      0.85      0.91        55
                   Urology       1.00      0.96      0.98        47
                  accuracy                           0.92       465
                 macro avg       0.93      0.91      0.92       465
              weighted avg       0.93      0.92      0.92       465
```