Index

S.no	Practical	Date	Signature
1	Introduction to Types of Medical Datasets		
2	Data Preprocessing and Cleaning for Electronic Health Records (EHR) Data		
3	Exploratory Data Analysis (EDA) on Medical Imaging Datasets		
4	Building a Binary Classification Model for Disease Diagnosis		
5	Implementing Multiclass Classification for Disease Severity Prediction		
6	Applying Time Series Analysis for Patient Vital Sign Forecasting		
7	Developing a Convolutional Neural Network (CNN) for Medical Image Classification		
8	Building a Recurrent Neural Network (RNN) for Predicting Patient Readmission		
9	Implementing Transfer Learning for Medical Image Feature Extraction		
10	Overview of Different Evaluation Metrics for Healthcare Datasets		

Experiment 1: Introduction to Types of Medical Datasets

Aim:
To explore and understand various healthcare, medical, and life sciences datasets from reputable sources, examining their scope, data types, and significance in advancing medical research, healthcare practices, and public health policy.

Theory:
Healthcare and life sciences datasets are pivotal for researchers, clinicians, and policymakers as they provide essential insights into population health, disease trends, healthcare utilization, and treatment efficacy. Below are various significant datasets that serve multiple purposes in the healthcare landscape:

1. **WHO Global Health Observatory (GHO):**
   The GHO is a comprehensive resource by the World Health Organization that includes datasets and reports from 194 countries. It covers a wide range of health topics, including mortality rates, child nutrition, water and sanitation access, HIV/AIDS prevalence, health system performance, and injury statistics. This resource is vital for global health monitoring and policy formulation.
2. **DHS Program:**
   The Demographic and Health Surveys (DHS) Program provides a wealth of medical datasets collected through surveys, biomarker testing, and geographic data. Spanning multiple topics and countries, these datasets facilitate cross-country comparisons and offer insights into health behaviors, outcomes, and demographic factors that affect health.
3. **HealthData.gov:**
   This is the official US government website for healthcare data, providing access to numerous datasets related to the US population. Topics include COVID-19 data, health equity statistics, and various public health indicators, supporting transparency and data-driven decision-making in healthcare policy.
4. **Life Science Database Archive:**
   A comprehensive dataset from Japan, gathered by life scientists over extended periods. This dataset includes information on organs, antigens, chemicals, and other biological entities, aiding research in various fields of life sciences.
5. **Data.gov.au:**
   This platform serves as the official source of open government data in Australia. It includes a diverse array of datasets, including healthcare data, which can be utilized for public health research and policy development.
6. **Kent Ridge Biomedical Datasets:**
   This database offers datasets from the biomedical field, including data that has been

published in journals. It is instrumental for researchers looking to analyze and validate findings in biomedical research.

7. **CDC WONDER:**
   The Centers for Disease Control and Prevention (CDC) provides the WONDER database, which contains extensive public health information on various topics, including mortality rates, natality, cancer statistics, and vaccination coverage, supporting epidemiological research.
8. **openFDA:**
   openFDA provides APIs and raw download access to structured datasets from the FDA. This includes data on adverse events related to drug use, drug product labeling, and recall enforcement reports, crucial for monitoring drug safety and efficacy.
9. **Big Cities Health Inventory Data Platform:**
   This dataset compiles over 150,000 data points from 35 large US cities across 120 health-related metrics, including life expectancy, access to healthcare services, mental health, chronic conditions, and socio-economic factors.
10. **OECD Health Statistics 2023:**
    The Organization for Economic Co-operation and Development (OECD) provides comparable statistics on health systems across member countries, facilitating international health policy comparisons and improvements.

Image Datasets for Life Sciences, Healthcare, and Medicine:

11. **OASIS:**
    The Open Access Series of Imaging Studies (OASIS) provides neuroimaging datasets, currently encompassing 1,098 subjects across 2,168 MRI sessions and 1,608 PET sessions, supporting research in cognitive neuroscience.
12. **OpenNeuro:**
    This platform offers 716 public datasets from 27,482 participants, featuring MRI, PET, MEG, EEG, and iEEG data, enabling broad exploration in neuroimaging studies.
13. **ADNI:**
    The Alzheimer's Disease Neuroimaging Initiative (ADNI) provides a free public dataset containing MRI and PET images, genetic data, cognitive tests, and biomarkers collected from patients with Alzheimer's disease and mild cognitive impairment, aiding research in neurodegenerative diseases.
14. **Deep Lesion:**
    This dataset consists of more than 32,000 CT images sourced from the NIH Clinical Center, utilized for developing AI models to assist in medical image diagnosis.

Mortality and Diseases Datasets:

15. **Human Mortality Database:**
    A global database providing mortality and population estimation rates for developed countries, crucial for demographic studies and public health planning.
16. **Chronic Disease Data:**
    This open dataset by the CDC contains 124 indicators of chronic disease, collected from various states, useful for understanding and addressing chronic health issues in the population.
17. **CHDS:**
    Datasets from the Child Health and Development Studies (CHDS) that investigate the intergenerational transmission of health and disease, considering genetic, social, and environmental factors.

Genome Datasets:

18. **1000 Genomes Project:**
    This dataset contains information from an international collaboration to catalog human genetic variation, including SNPs, structural variants, and haplotype context. The data is publicly available for research purposes.

Hospital and Healthcare Services Datasets:

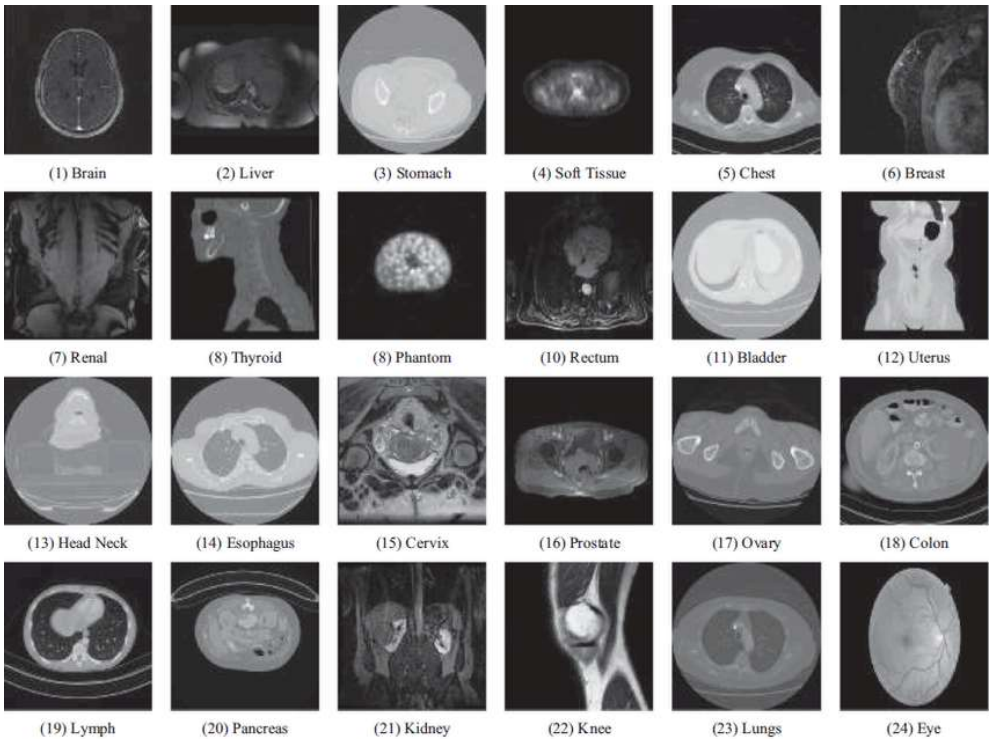
19. **Healthcare Cost and Utilization Project (HCUP):**
    A comprehensive source for tracking healthcare utilization trends in the US, including costs, access, quality, and outcomes, essential for health services research.
20. **Medicare:**
    A collection of 337 datasets from Medicare providers in the US, offering insights into healthcare facilities, doctors, hospitals, and rehabilitation services.
21. **OECD Hospital Performance:**
    This dataset provides national and hospital-level data on 30-day acute myocardial infarction (AMI) mortality rates, contributing to quality assessment in healthcare delivery.
22. **FY 2024 HHS Contingency Staffing Plan:**
    This document outlines contingency plans for the US Department of Health and Human Services (HHS), detailing staff retention and furlough measures in the absence of appropriations.

Cancer Datasets:

23. **CT Medical Images:**
    A dataset of CT images focused on examining trends in patient demographics, specifically relating to age and contrast media used in imaging.
24. **Broad Institute Cancer Program Datasets:**
    This resource contains datasets related to various tumor types, cellular responses, gene expression patterns, and more, supporting cancer research.
25. **SEER Cancer Incidence:**
    The Surveillance, Epidemiology, and End Results (SEER) program provides datasets related to cancer incidence across various demographics, including race and gender.

X-Ray Datasets:

26. **COVID-19 X-Ray Dataset:**
    This dataset includes 6,500 images of chest X-rays, with pixel-level segmentation for lung regions. Among these, 517 cases are confirmed to have COVID-19.
27. **NIH Database of 100,000 Chest X-Rays:**
    A comprehensive dataset comprising over 112,000 chest X-ray images from 30,000 unique patients, labeled with natural language processing (NLP) techniques, although discrepancies in labels have been noted.



Experiment 2: Data Preprocessing and Cleaning for Electronic Health Records (EHR) Data

---

Aim:
To understand and perform data preprocessing and cleaning on Electronic Health Records (EHR) data to ensure accuracy, consistency, and usability for analysis in healthcare applications.

Theory:
Electronic Health Records (EHR) contain comprehensive patient information, including demographics, medical histories, lab results, diagnoses, treatments, and medication information. EHR data is often complex, originating from multiple sources within healthcare systems, which can lead to inconsistencies, missing values, and variations in data formats. Effective preprocessing and cleaning are essential to ensure data quality, reliability, and interoperability, making it suitable for analysis, machine learning, and clinical decision support.

The following steps are typically involved in preprocessing and cleaning EHR data:

1. **Data Exploration and Initial Assessment:**
   In this step, EHR data is explored to understand its structure, data types, and any irregularities. Summary statistics are generated for each feature, such as mean, median, mode, and standard deviation, to detect outliers or unexpected patterns. This phase helps identify columns that may need significant cleaning or transformation.
2. **Handling Missing Values:**
   Missing values in EHR data can arise from incomplete data entry, patient non-response, or varying medical practices across providers. Techniques to handle missing values include:
   - **Imputation:** Filling in missing data using statistical techniques, such as mean, median, or mode imputation, or more advanced methods like k-nearest neighbors (KNN) or predictive modeling.
   - **Removal of Records or Features:** In cases where a feature has excessive missing data, it may be removed if it does not significantly affect the analysis.
   - **Forward/Backward Filling:** For time-series data, missing values can be filled by propagating the previous or following values within a patient's record.
3. **Standardizing Data Formats:**
   EHR data often includes diverse data formats, such as dates, units of measurement, and categorical values. For consistency:
   - **Date Standardization:** Dates are formatted uniformly (e.g., YYYY-MM-DD).
   - **Unit Conversion:** Measurements are converted to standard units (e.g., converting height from inches to centimeters).

- **Categorical Data Standardization:** Standardizing labels in categorical features like gender, diagnoses, and procedures to ensure uniformity across the dataset.

4. **Outlier Detection and Treatment:**
   Outliers in EHR data can be indicative of data entry errors, sensor malfunctions, or abnormal medical conditions. Statistical techniques, such as Z-score analysis, interquartile range (IQR), or domain knowledge-based thresholds, are used to identify and address outliers. Outliers may either be removed or capped based on medical relevance and the goals of the analysis.
5. **Encoding Categorical Data:**
   EHR data contains categorical features (e.g., diagnosis codes, medication names) that need to be converted into numerical formats for analysis or machine learning. Encoding techniques include:
   - **One-Hot Encoding:** Creating binary columns for each category in a feature, suitable for non-ordinal categories.
   - **Label Encoding:** Assigning integer values to categories, used when categories have an inherent order.
   - **Binary Encoding:** A combination of one-hot and label encoding, often applied to high-cardinality categorical features.
6. **Data Normalization and Scaling:**
   Normalization or scaling is applied to numerical features to ensure they are on a comparable scale, which is essential for algorithms sensitive to data magnitude, such as distance-based models. Common scaling methods include:
   - **Min-Max Scaling:** Rescaling data to a specified range, often [0,1].
   - **Standardization (Z-score):** Centering data around the mean with a unit standard deviation, which is beneficial for data with normal distribution.
7. **De-duplication of Patient Records:**
   Duplicate records in EHR data can occur due to multiple visits or data entry errors. Deduplication involves identifying and removing redundant records to ensure each patient has a unique and consolidated medical history.
8. **Data Anonymization:**
   EHR data contains sensitive information; therefore, anonymization techniques are applied to protect patient privacy, especially for research or shared datasets. Common anonymization techniques include:
   - **Data Masking:** Obscuring direct identifiers, such as names and social security numbers.
   - **Data Aggregation:** Aggregating sensitive attributes, such as reporting age ranges instead of exact age.
   - **Randomization:** Applying slight changes to certain fields to prevent re-identification.

In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
import os

%matplotlib inline
```

In [20]:
```
df = pd.read_csv("diabetes.csv")#, names=column_names)
column_names = df.columns.tolist()

df.head()
```

Out[20]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [21]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

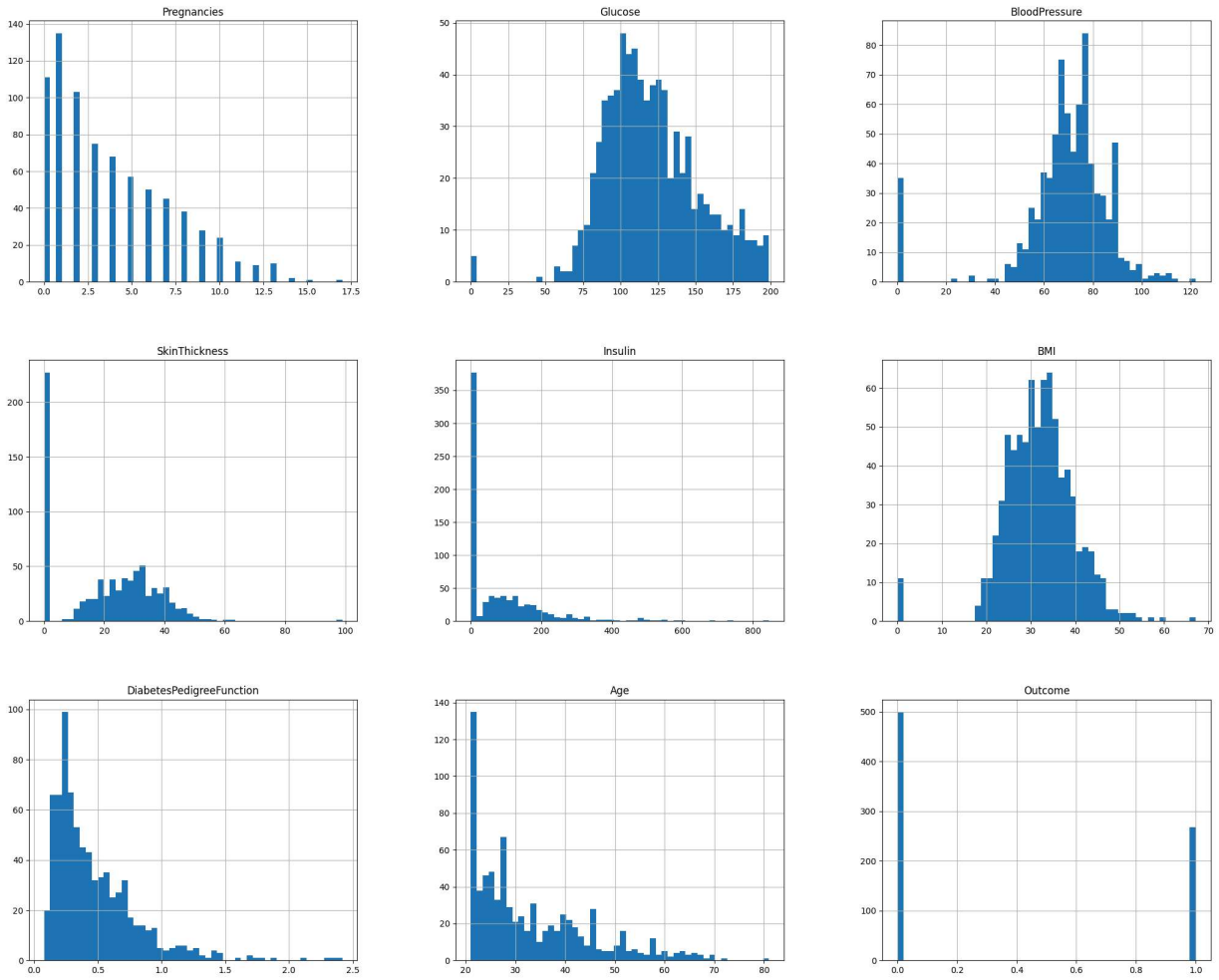
We can see that there are 768 instances in the dataset, which is very small for a machine learning dataset. For most algorithms you will need more data.

In [22]:
```
df.describe()
```

Out[22]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [23]:
```
df.hist(bins=50, figsize=(25, 20))
plt.show()
```



Removing duplicated data points

In [24]:
```
df.drop_duplicates(keep='first', inplace=True)
```

Creating a test set

In [25]:
```
original_train_df, test_df = train_test_split(df, test_size=0.2)

# To be sure we will create a copy for further processing
train_df = original_train_df.copy()
```

Gaining further insight

In [26]:
```
correlation_matrix = train_df.corr(method='pearson')

correlation_matrix
```

Out[26]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.143476	0.122206	-0.107195	-0.070121
Glucose	0.143476	1.000000	0.164905	0.051771	0.332322
BloodPressure	0.122206	0.164905	1.000000	0.191297	0.085671
SkinThickness	-0.107195	0.051771	0.191297	1.000000	0.443088
Insulin	-0.070121	0.332322	0.085671	0.443088	1.000000
BMI	0.030443	0.231687	0.298513	0.388113	0.213526
DiabetesPedigreeFunction	-0.015325	0.143050	0.056765	0.202198	0.242642
Age	0.555777	0.290635	0.269591	-0.089432	-0.030957
Outcome	0.227199	0.473457	0.084516	0.068468	0.123888

Let's take a look at how each attribute correlates with the final diagnosis

In [27]:
```
correlation_matrix["Outcome"].sort_values()
```

Out[27]:
```
SkinThickness              0.068468
BloodPressure              0.084516
Insulin                    0.123888
DiabetesPedigreeFunction   0.206317
Pregnancies                0.227199
Age                        0.247086
BMI                        0.280265
Glucose                    0.473457
Outcome                    1.000000
Name: Outcome, dtype: float64
```

Handling missing data

In [28]:
```
names = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPe

for name in names:
    train_df[name].replace(0, np.nan, inplace=True)
```

```
C:\Users\nehac\AppData\Local\Temp\ipykernel_9116\551462321.py:4: FutureWarning: A va
lue is trying to be set on a copy of a DataFrame or Series through chained assignmen
t using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train_df[name].replace(0, np.nan, inplace=True)
```

In [29]:
```
train_df.head()
```

Out[29]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
206	8	196.0	76.0	29.0	280.0	37.5	
203	2	99.0	70.0	16.0	44.0	20.4	
508	2	84.0	50.0	23.0	76.0	30.4	
488	4	99.0	72.0	17.0	NaN	25.6	
624	2	108.0	64.0	NaN	NaN	30.8	

In [30]:
```
glucose_median = train_df["Glucose"].median()
blood_pressure_median = train_df["BloodPressure"].median()
skin_thickness_median = train_df["SkinThickness"].median()
insulin_median = train_df["Insulin"].median()
bmi_median = train_df["BMI"].median()
age_median = train_df["Age"].median()
dpf_median = train_df['DiabetesPedigreeFunction'].median()

train_df["Glucose"].fillna(glucose_median, inplace=True)
train_df["BloodPressure"].fillna(blood_pressure_median, inplace=True)
train_df["SkinThickness"].fillna(skin_thickness_median, inplace=True)
train_df["Insulin"].fillna(insulin_median, inplace=True)
train_df["BMI"].fillna(bmi_median, inplace=True)
train_df["Age"].fillna(age_median, inplace=True)
train_df["DiabetesPedigreeFunction"].fillna(dpf_median, inplace=True)
```

```
C:\Users\nehac\AppData\Local\Temp\ipykernel_9116\2039840509.py:13: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train_df["BMI"].fillna(bmi_median, inplace=True)
C:\Users\nehac\AppData\Local\Temp\ipykernel_9116\2039840509.py:14: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train_df["Age"].fillna(age_median, inplace=True)
C:\Users\nehac\AppData\Local\Temp\ipykernel_9116\2039840509.py:15: FutureWarning: A
value is trying to be set on a copy of a DataFrame or Series through chained assignm
ent using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  train_df["DiabetesPedigreeFunction"].fillna(dpf_median, inplace=True)
```

In [31]:
```
train_df.head()
```

Out[31]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
206	8	196.0	76.0	29.0	280.0	37.5	
203	2	99.0	70.0	16.0	44.0	20.4	
508	2	84.0	50.0	23.0	76.0	30.4	
488	4	99.0	72.0	17.0	123.5	25.6	
624	2	108.0	64.0	29.0	123.5	30.8	

Handling noisy data and outliers

In [32]:
```
# Copy original dataframe and add new column with random fitness values
temp_df = train_df.copy()
fitness_values = ["bad", "moderate", "good", "very good",]
temp_df['fitness'] = np.random.choice(fitness_values, temp_df.shape[0])

temp_df.head(5)
```

Out[32]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
206	8	196.0	76.0	29.0	280.0	37.5	
203	2	99.0	70.0	16.0	44.0	20.4	
508	2	84.0	50.0	23.0	76.0	30.4	
488	4	99.0	72.0	17.0	123.5	25.6	
624	2	108.0	64.0	29.0	123.5	30.8	

In [33]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
fitness_encoded = encoder.fit_transform(temp_df["fitness"])

for id_, class_ in enumerate(encoder.classes_):
    print(f"class id {id_} has label {class_}")

print()
print(f"Encoded fitness values for first 10 entries: {fitness_encoded[:10]}")
```

```
class id 0 has label bad
class id 1 has label good
class id 2 has label moderate
class id 3 has label very good

Encoded fitness values for first 10 entries: [0 2 2 3 0 3 1 0 1 1]
```

Rescaling or standardizing attributes

In [34]:
```
from sklearn.preprocessing import MinMaxScaler

# initialize min-max scaler
mm_scaler = MinMaxScaler()

temp1_df = train_df.copy()
column_names = temp1_df.columns.tolist()

# transform all attributes
temp1_df[column_names] = mm_scaler.fit_transform(temp1_df[column_names])

temp1_df.sort_index(inplace=True)
temp1_df.head()
```

Out[34]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
3	0.058824	0.290323	0.428571	0.173913	0.095066	0.282857	
4	0.000000	0.600000	0.163265	0.304348	0.184116	0.711429	
5	0.294118	0.464516	0.510204	0.239130	0.130566	0.211429	
7	0.588235	0.458065	0.489796	0.239130	0.130566	0.488571	
9	0.470588	0.522581	0.734694	0.239130	0.130566	0.405714	

In [35]:

```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()

temp2_df = train_df.copy()

# transform all attributes
temp2_df[column_names] = mm_scaler.fit_transform(temp2_df[column_names])
temp2_df.sort_index(inplace=True)
temp2_df.head()
```

Out[35]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
3	0.058824	0.290323	0.428571	0.173913	0.095066	0.282857	
4	0.000000	0.600000	0.163265	0.304348	0.184116	0.711429	
5	0.294118	0.464516	0.510204	0.239130	0.130566	0.211429	
7	0.588235	0.458065	0.489796	0.239130	0.130566	0.488571	
9	0.470588	0.522581	0.734694	0.239130	0.130566	0.405714	

Experiment 3: Exploratory Data Analysis (EDA) on Medical Imaging Datasets

---

Aim:
To conduct exploratory data analysis (EDA) on medical imaging datasets to uncover underlying patterns, understand data distributions, identify relationships between features, and evaluate the quality of images. The goal is to prepare the dataset for further analysis or model training, ensuring insights are extracted to guide subsequent steps in medical imaging research.

Theory:
Exploratory Data Analysis (EDA) is a crucial step in the data analysis process that allows researchers to gain insights into their datasets through visualizations and statistical techniques. In the context of medical imaging datasets, EDA helps in understanding the characteristics of images and associated metadata, such as patient demographics, diagnosis labels, and image acquisition parameters.

Key steps in conducting EDA on medical imaging datasets include:

1. **Dataset Overview:**
   Start by obtaining a comprehensive overview of the dataset, including:
   - The number of images and associated metadata.
   - The dimensions and formats of the images (e.g., JPEG, PNG, DICOM).
   - Basic statistics (e.g., mean, median, variance) of image pixel values.
   - The distribution of classes or labels (if the dataset is labeled).
2. **Data Visualization:**
   Visual representations of data play a crucial role in EDA. Common visualizations for medical imaging datasets include:
   - **Sample Image Visualization:** Display a few random samples from the dataset to understand the types of images included and the quality of the imaging.
   - **Histogram of Pixel Intensities:** Plot histograms for the distribution of pixel intensities to observe brightness and contrast levels in the images.
   - **Class Distribution Bar Chart:** Visualize the number of images in each class to check for class imbalance, which is essential for classification tasks.
3. **Image Quality Assessment:**
   Assess the quality of medical images by evaluating:
   - **Resolution and Aspect Ratio:** Check for variations in image resolutions and aspect ratios, which can affect model training and analysis.
   - **Noise and Artifacts:** Identify any noise or artifacts present in the images, which may impact diagnostic accuracy.
   - **Missing or Corrupted Images:** Detect any missing or corrupted image files that need to be addressed before model training.

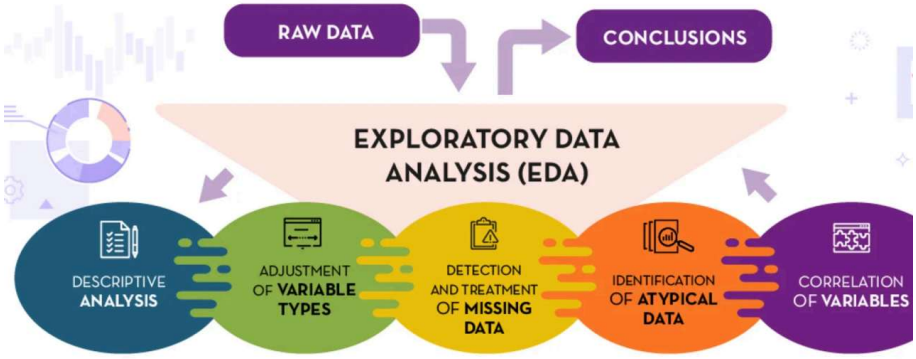
4. **Image Augmentation Insights:**
   Analyze the potential for data augmentation, which is essential for training robust models in medical imaging:
   - **Identify Transformations:** Determine appropriate transformations (e.g., rotations, flips, brightness adjustments) based on the distribution of the image data.
   - **Visualize Augmented Samples:** Generate and visualize augmented versions of original images to understand how augmentation can enhance dataset diversity.
5. **Correlation Analysis:**
   If the dataset includes metadata (e.g., age, gender, diagnosis), perform correlation analysis to identify relationships between image features and clinical parameters:
   - **Heatmap Visualization:** Create a heatmap to show correlations between numerical metadata and the classes of images.
   - **Box Plots:** Use box plots to visualize the distribution of image classes across different metadata attributes, helping to identify patterns or trends.
6. **Dimensionality Reduction:**
   For high-dimensional image data, consider applying dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to visualize the image data in lower dimensions. This can help in understanding clustering behavior among images or different diagnostic classes.
7. **Identifying Patterns and Anomalies:**
   Use unsupervised learning techniques or clustering methods (e.g., K-means) to identify patterns or anomalies within the dataset. This step can reveal unexpected groupings or outliers in the data.
8. **Feature Engineering Considerations:**
   Based on insights gained during EDA, outline potential feature engineering strategies that can improve model performance. This may include techniques for extracting image features (e.g., texture, shape, color) that can enhance classification accuracy.

Dataset Description

This dataset contains 5,856 validated Chest X-Ray images. The images are split into a training set and a testing set of independent patients. Images are labeled as (disease:NORMAL/BACTERIA/VIRUS)-(randomized patient ID)-(image number of a patient).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.


RAW DATA
CONCLUSIONS
EXPLORATORY DATA ANALYSIS (EDA)
DESCRIPTIVE ANALYSIS
ADJUSTMENT OF VARIABLE TYPES
DETECTION AND TREATMENT OF MISSING DATA
IDENTIFICATION OF ATYPICAL DATA
CORRELATION OF VARIABLES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import xml.etree.ElementTree as ET
import os
import cv2
import tensorflow as tf
from wordcloud import WordCloud
import re
from collections import defaultdict
import itertools
from collections import Counter

directory = 'ecgen-radiology'

# extracting data from the xml documents
img = []
img_impression = []
img_finding = []
for filename in tqdm(os.listdir(directory)):
    if filename.endswith(".xml"):
        f = directory + '/' + filename
        tree = ET.parse(f)
        root = tree.getroot()
        for child in root:
            if child.tag == 'MedlineCitation':
                for attr in child:
                    if attr.tag == 'Article':
                        for i in attr:
                            if i.tag == 'Abstract':
                                for name in i:
                                    if name.get('Label') ==
'FINDINGS':
                                        finding=name.text

        for p_image in root.findall('parentImage'):

            img.append(p_image.get('id'))
            img_finding.append(finding)
```

```
100%|
| 3955/3955 [00:34<00:00, 114.76it/s]
```

```
dataset = pd.DataFrame()
dataset['Image_path'] = img
dataset['Finding'] = img_finding
```

```
dataset.head(10)
```

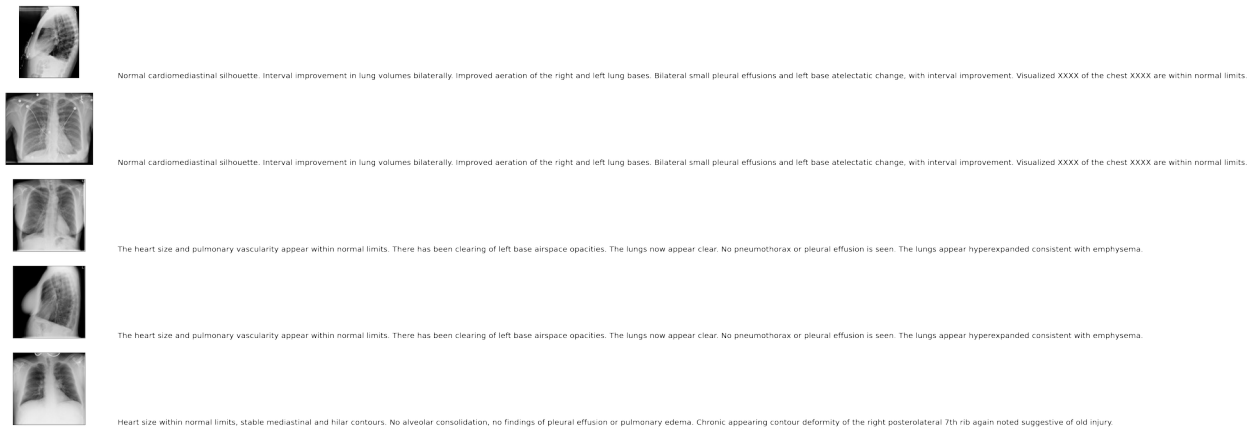
```
              Image_path                                            
Finding
0   CXR1_1_IM-0001-3001  The cardiac silhouette and mediastinum size
ar...
1   CXR1_1_IM-0001-4001  The cardiac silhouette and mediastinum size
ar...
2    CXR10_IM-0002-1001  The cardiomediastinal silhouette is within
nor...
3    CXR10_IM-0002-2001  The cardiomediastinal silhouette is within
nor...
4   CXR100_IM-0002-1001  Both lungs are clear and expanded. Heart and
m...
5   CXR100_IM-0002-2001  Both lungs are clear and expanded. Heart and
m...
6  CXR1000_IM-0003-1001  There is XXXX increased opacity within the
rig...
7  CXR1000_IM-0003-2001  There is XXXX increased opacity within the
rig...
8  CXR1000_IM-0003-3001  There is XXXX increased opacity within the
rig...
9  CXR1001_IM-0004-1001  Interstitial markings are diffusely prominent
...
```

```
print('Dataset Shape:', dataset.shape)
```

```
Dataset Shape: (7470, 2)
```

```
def absolute_path(x):
    '''Makes the path absolute '''
    x = 'Scanned Images/'+ x + '.png'
    return x

dataset['Image_path'] = dataset['Image_path'].apply(lambda x :
absolute_path(x)) # making the paths absolute

def image_desc_plotter(data, n, rep):
    count = 1
    fig = plt.figure(figsize=(10,20))

    if rep == 'finding':

        for filename in data['Image_path'].values[95:100]:

            findings = list(data["Finding"].loc[data["Image_path"] ==
filename].values)
            img = cv2.imread(filename)
            ax = fig.add_subplot(n, 2 , count , xticks=[], yticks=[])

            ax.imshow(img)
```

```
            count += 1
            ax = fig.add_subplot(n ,2 ,count)
            plt.axis('off')
            ax.plot()
            ax.set_xlim(0,1)
            ax.set_ylim(0, len(findings))
            for i, f in enumerate(findings):
                ax.text(0,i,f,fontsize=20)
            count += 1
        plt.show()

    else:
        print("Enter a valid String")


image_desc_plotter(dataset, 5, 'finding')
```



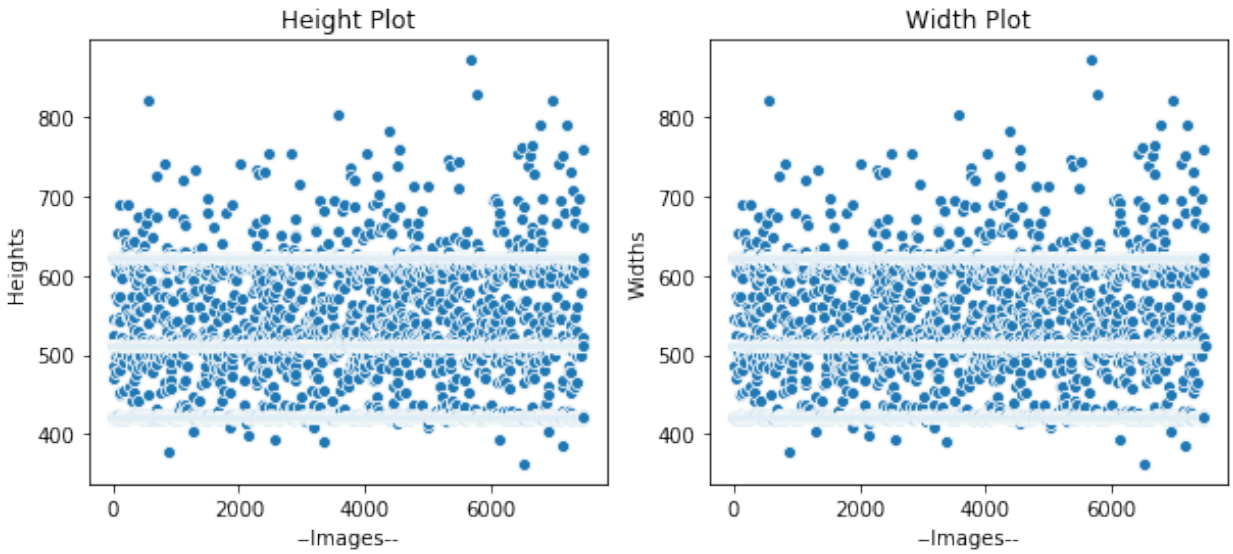

```
# loading the heights and widths of each image
h = []
w = []
for i in tqdm(np.unique(dataset['Image_path'].values)):
    img = cv2.imread(i)
    h.append(img.shape[0])
    w.append(img.shape[0])
```

```
100%|
| 7470/7470 [02:03<00:00, 60.27it/s]
```

```
plt.figure(figsize=(10,4))
plt.subplot(121)
plt.title('Height Plot')
plt.ylabel('Heights')
plt.xlabel('--Images--')
sns.scatterplot(range(len(h)), h)
plt.subplot(122)
```

```
plt.title('Width Plot')
plt.ylabel('Widths')
plt.xlabel('--Images--')
sns.scatterplot(range(len(w)), h)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fd59171dc8>
```



Images have different heights and widths, they will be resized into a common shape

```
print('Number of Images:', dataset['Image_path'].nunique())
```

```
Number of Images: 7470
```

```
# number of missing values
dataset.isnull().sum()
```

```
Image_path      0
Finding       997
dtype: int64
```

There are a total of 997 rows where 'findings' column has no value. We can simply drop them.

```
dataset = dataset.dropna(axis=0) # drop all missing value rows

dataset.isnull().sum()
```

```
Image_path    0
Finding       0
dtype: int64
```

```
print('New Shape of the Data:', dataset.shape)
```

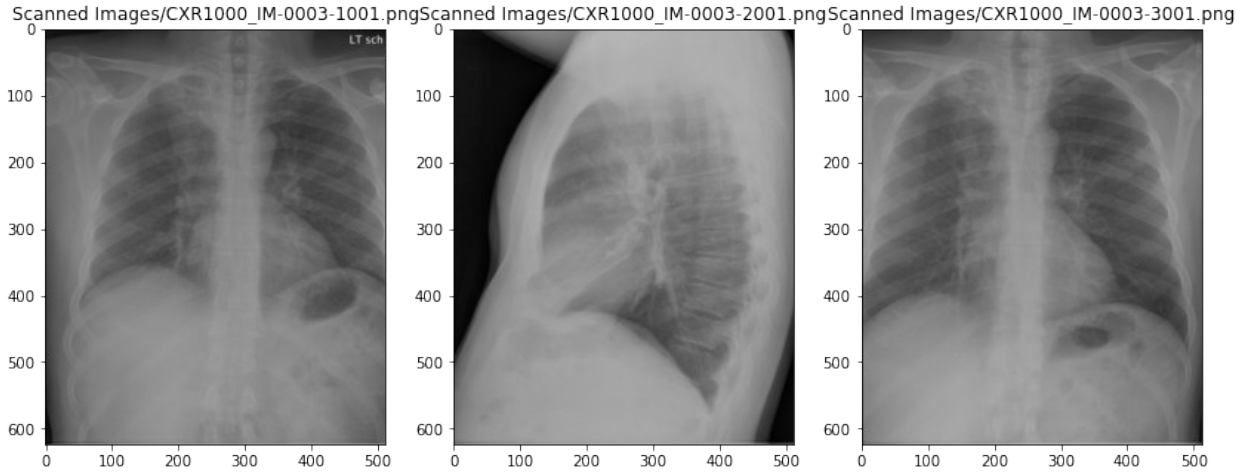
```
New Shape of the Data: (6473, 2)
```

```
dataset.head(12)
```

```
                                    Image_path  \
0    Scanned Images/CXR1_1_IM-0001-3001.png
1    Scanned Images/CXR1_1_IM-0001-4001.png
2     Scanned Images/CXR10_IM-0002-1001.png
3     Scanned Images/CXR10_IM-0002-2001.png
4    Scanned Images/CXR100_IM-0002-1001.png
5    Scanned Images/CXR100_IM-0002-2001.png
6   Scanned Images/CXR1000_IM-0003-1001.png
7   Scanned Images/CXR1000_IM-0003-2001.png
8   Scanned Images/CXR1000_IM-0003-3001.png
9   Scanned Images/CXR1001_IM-0004-1001.png
10  Scanned Images/CXR1001_IM-0004-1002.png
13  Scanned Images/CXR1003_IM-0005-2002.png

                                              Finding
0   The cardiac silhouette and mediastinum size ar...
1   The cardiac silhouette and mediastinum size ar...
2   The cardiomediastinal silhouette is within nor...
3   The cardiomediastinal silhouette is within nor...
4   Both lungs are clear and expanded. Heart and m...
5   Both lungs are clear and expanded. Heart and m...
6   There is XXXX increased opacity within the rig...
7   There is XXXX increased opacity within the rig...
8   There is XXXX increased opacity within the rig...
9   Interstitial markings are diffusely prominent ...
10  Interstitial markings are diffusely prominent ...
13  Heart size and pulmonary vascularity appear wi...
```

```
plt.figure(figsize=(14,7))
plt.subplot(131)
img1 = cv2.imread(dataset['Image_path'].values[6])
plt.imshow(img1)
plt.title(dataset['Image_path'].values[6])
plt.subplot(132)
img2 = cv2.imread(dataset['Image_path'].values[7])
plt.title(dataset['Image_path'].values[7])
plt.imshow(img2)
plt.subplot(133)
img3 = cv2.imread(dataset['Image_path'].values[8])
plt.title(dataset['Image_path'].values[8])
plt.imshow(img3)
```

```
<matplotlib.image.AxesImage at 0x2648e404ac8>
```




```
dataset['Finding'].values[6], dataset['Finding'].values[7],
dataset['Finding'].values[8]
```

```
('There is XXXX increased opacity within the right upper lobe with
possible mass and associated area of atelectasis or focal
consolidation. The cardiac silhouette is within normal limits. XXXX
opacity in the left midlung overlying the posterior left 5th rib may
represent focal airspace disease. No pleural effusion or pneumothorax.
No acute bone abnormality.',
 'There is XXXX increased opacity within the right upper lobe with
possible mass and associated area of atelectasis or focal
consolidation. The cardiac silhouette is within normal limits. XXXX
opacity in the left midlung overlying the posterior left 5th rib may
represent focal airspace disease. No pleural effusion or pneumothorax.
No acute bone abnormality.',
 'There is XXXX increased opacity within the right upper lobe with
possible mass and associated area of atelectasis or focal
consolidation. The cardiac silhouette is within normal limits. XXXX
opacity in the left midlung overlying the posterior left 5th rib may
represent focal airspace disease. No pleural effusion or pneumothorax.
No acute bone abnormality.')
```

The above two findings are same since its of the same person. The images are also of the same person but scanned at different views.

The dataset consists of multiple chest shots of the same person. The images of a person has the same file name except the last 4 digits. Therefore that can be taken as the person ID.

```
# This creates 2 dictionaries with keys as the person id and the
number of images and findings for that person.
images = {}
findings = {}
for img, fin in dataset.values:
```

```
    a = img.split('-')
    a.pop(len(a)-1)
    a = '-'.join(e for e in a)
    if a not in images.keys():
        images[a] = 1
        findings[a] = fin
    else:
        images[a] += 1
        findings[a] = fin

images['Scanned Images/CXR1001_IM-0004'], findings['Scanned
Images/CXR1001_IM-0004']
```

```
(2,
 'Interstitial markings are diffusely prominent throughout both lungs.
Heart size is normal. Pulmonary XXXX normal.')
```

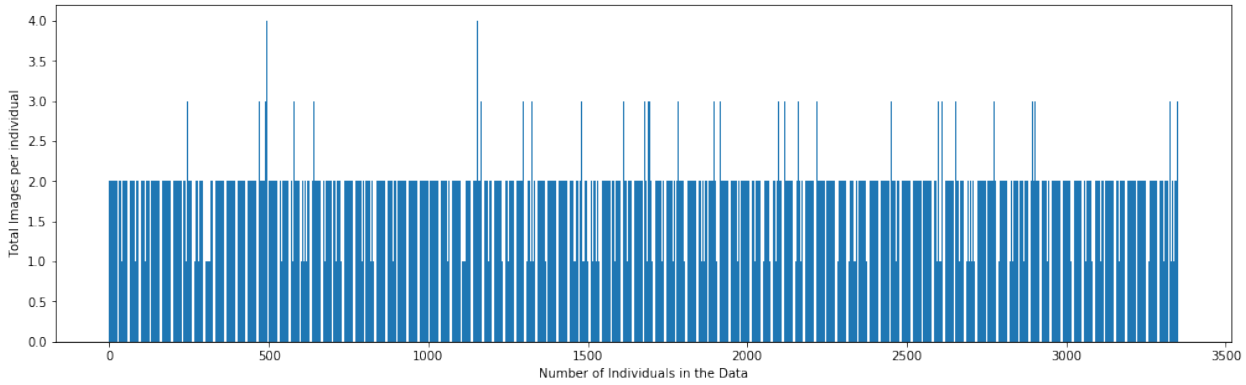
```
print('Total Number of Unique_IDs :', len(images.keys()))
```

```
Total Number of Unique_IDs : 3350
```

The dataset contains images of 3350 different people

```
plt.figure(figsize=(17,5))
plt.bar(range(len(images.keys())), images.values())
plt.ylabel('Total Images per individual')
plt.xlabel('Number of Individuals in the Data')
```

```
Text(0.5, 0, 'Number of Individuals in the Data')
```




```
one = 0
two = 0
three = 0
four = 0
for v in images.values():
    if v == 1:
        one +=1
```

```
    elif v == 2:
        two += 1
    elif v == 3:
        three += 1
    elif v == 4:
        four += 1
    else:
        print('Error')

one, two, three, four
```

```
(390, 2807, 143, 10)
```

The above variables one, two, three, four contains the total number of IDs with 1,2,3,4 number of images respectively.

As we can see there are multiple images corresponding to a single person. These are different chest scans at different views. Most of the individuals have only 2 scans while the highest being 4.

```
images
```

```
{'Scanned Images/CXR1_1_IM-0001': 2,
 'Scanned Images/CXR10_IM-0002': 2,
 'Scanned Images/CXR100_IM-0002': 2,
 'Scanned Images/CXR1000_IM-0003': 3,
 'Scanned Images/CXR1001_IM-0004': 2,
 'Scanned Images/CXR1003_IM-0005': 1,
 'Scanned Images/CXR1004_IM-0005': 2,
 'Scanned Images/CXR1005_IM-0006': 2,
 'Scanned Images/CXR1006_IM-0007': 2,
 'Scanned Images/CXR1007_IM-0008': 3,
 'Scanned Images/CXR1008_IM-0009': 2,
 'Scanned Images/CXR1009_IM-0010': 2,
 'Scanned Images/CXR101_IM-0011': 2,
 'Scanned Images/CXR1010_IM-0012': 2,
 'Scanned Images/CXR1011_IM-0013': 2,
 'Scanned Images/CXR1012_IM-0013': 1,
 'Scanned Images/CXR1013_IM-0013': 2,
 'Scanned Images/CXR1015_IM-0001': 2,
 'Scanned Images/CXR1015_IM-0013': 2,
 'Scanned Images/CXR1017_IM-0013': 2,
 'Scanned Images/CXR1018_IM-0014': 2,
 'Scanned Images/CXR1019_IM-0015': 2,
 'Scanned Images/CXR102_IM-0016': 2,
 'Scanned Images/CXR1020_IM-0017': 2,
 'Scanned Images/CXR1022_IM-0017': 2,
 'Scanned Images/CXR1023_IM-0018': 2,
 'Scanned Images/CXR1024_IM-0019': 1,
 'Scanned Images/CXR1025_IM-0020': 2,
```

```
'Scanned Images/CXR1026_IM-0021': 1,
'Scanned Images/CXR1027_IM-0021': 2,
'Scanned Images/CXR1028_IM-0022': 2,
'Scanned Images/CXR103_IM-0023': 2,
'Scanned Images/CXR1030_IM-0024': 2,
'Scanned Images/CXR1032_IM-0026-1001': 2,
'Scanned Images/CXR1033_IM-0027': 2,
'Scanned Images/CXR1034_IM-0028': 2,
'Scanned Images/CXR1035_IM-0028': 2,
'Scanned Images/CXR1036_IM-0029': 1,
'Scanned Images/CXR1037_IM-0029': 2,
'Scanned Images/CXR1039_IM-0030': 1,
'Scanned Images/CXR104_IM-0031-0001': 1,
'Scanned Images/CXR1041_IM-0033': 2,
'Scanned Images/CXR1042_IM-0034': 2,
'Scanned Images/CXR1043_IM-0035-0001': 2,
'Scanned Images/CXR1044_IM-0036': 2,
'Scanned Images/CXR1045_IM-0036': 2,
'Scanned Images/CXR1046_IM-0036': 2,
'Scanned Images/CXR1047_IM-0036': 2,
'Scanned Images/CXR1048_IM-0036': 2,
'Scanned Images/CXR105_IM-0037': 2,
'Scanned Images/CXR1050_IM-0038': 2,
'Scanned Images/CXR1051_IM-0039': 2,
'Scanned Images/CXR1052_IM-0040': 2,
'Scanned Images/CXR1053_IM-0040': 2,
'Scanned Images/CXR1054_IM-0040': 2,
'Scanned Images/CXR1055_IM-0040': 2,
'Scanned Images/CXR1056_IM-0040': 3,
'Scanned Images/CXR1057_IM-0041': 2,
'Scanned Images/CXR1058_IM-0041': 2,
'Scanned Images/CXR1059_IM-0041': 1,
'Scanned Images/CXR106_IM-0042': 2,
'Scanned Images/CXR1060_IM-0042': 2,
'Scanned Images/CXR1061_IM-0043': 2,
'Scanned Images/CXR1062_IM-0043': 1,
'Scanned Images/CXR1063_IM-0044': 2,
'Scanned Images/CXR1064_IM-0045': 1,
'Scanned Images/CXR1065_IM-0046': 2,
'Scanned Images/CXR1067_IM-0048': 2,
'Scanned Images/CXR1069_IM-0049': 1,
'Scanned Images/CXR107_IM-0049': 2,
'Scanned Images/CXR1070_IM-0050': 2,
'Scanned Images/CXR1071_IM-0051': 1,
'Scanned Images/CXR1072_IM-0052-1001': 2,
'Scanned Images/CXR1073_IM-0053': 2,
'Scanned Images/CXR1074_IM-0054': 2,
'Scanned Images/CXR1075_IM-0054': 2,
'Scanned Images/CXR1076_IM-0054': 2,
```

```
'Scanned Images/CXR1077_IM-0054': 2,
'Scanned Images/CXR1078_IM-0055': 2,
'Scanned Images/CXR1079_IM-0056': 2,
'Scanned Images/CXR108_IM-0056': 2,
'Scanned Images/CXR1081_IM-0057': 1,
'Scanned Images/CXR1082_IM-0058': 1,
'Scanned Images/CXR1083_IM-0058': 2,
'Scanned Images/CXR1084_IM-0058': 2,
'Scanned Images/CXR1085_IM-0059': 2,
'Scanned Images/CXR1086_IM-0059': 2,
'Scanned Images/CXR1087_IM-0060': 2,
'Scanned Images/CXR1088_IM-0061': 2,
'Scanned Images/CXR1089_IM-0061': 2,
'Scanned Images/CXR1091_IM-0062': 2,
'Scanned Images/CXR1092_IM-0063': 2,
'Scanned Images/CXR1093_IM-0064': 2,
'Scanned Images/CXR1094_IM-0065': 2,
'Scanned Images/CXR1095_IM-0066': 1,
'Scanned Images/CXR1096_IM-0066': 1,
'Scanned Images/CXR1097_IM-0067': 2,
'Scanned Images/CXR1098_IM-0067': 1,
'Scanned Images/CXR1099_IM-0067': 2,
'Scanned Images/CXR11_IM-0067': 2,
'Scanned Images/CXR110_IM-0067': 2,
'Scanned Images/CXR1100_IM-0068': 2,
'Scanned Images/CXR1101_IM-0068': 3,
'Scanned Images/CXR1102_IM-0069': 4,
'Scanned Images/CXR1103_IM-0070': 2,
'Scanned Images/CXR1105_IM-0072-1001': 2,
'Scanned Images/CXR1108_IM-0075': 2,
'Scanned Images/CXR1109_IM-0076': 2,
'Scanned Images/CXR1110_IM-0076': 2,
'Scanned Images/CXR1111_IM-0077': 2,
'Scanned Images/CXR1112_IM-0078': 2,
'Scanned Images/CXR1113_IM-0078': 2,
'Scanned Images/CXR1114_IM-0079': 2,
'Scanned Images/CXR1115_IM-0079': 1,
'Scanned Images/CXR1116_IM-0079': 2,
'Scanned Images/CXR1117_IM-0079': 2,
'Scanned Images/CXR1118_IM-0079': 2,
'Scanned Images/CXR1119_IM-0080': 2,
'Scanned Images/CXR112_IM-0080': 2,
'Scanned Images/CXR1120_IM-0080': 1,
'Scanned Images/CXR1121_IM-0080': 2,
'Scanned Images/CXR1122_IM-0080-1001': 2,
'Scanned Images/CXR1122_IM-0080': 1,
'Scanned Images/CXR1123_IM-0080': 2,
'Scanned Images/CXR1124_IM-0081': 2,
'Scanned Images/CXR1125_IM-0082': 2,
```

```
'Scanned Images/CXR1126_IM-0082': 2,
'Scanned Images/CXR1129_IM-0085': 1,
'Scanned Images/CXR113_IM-0086': 3,
'Scanned Images/CXR1130_IM-0087': 2,
'Scanned Images/CXR1131_IM-0088-0001': 2,
'Scanned Images/CXR1133_IM-0090': 2,
'Scanned Images/CXR1134_IM-0091': 2,
'Scanned Images/CXR1135_IM-0091': 2,
'Scanned Images/CXR1136_IM-0092': 2,
'Scanned Images/CXR1138_IM-0094': 2,
'Scanned Images/CXR1139_IM-0095': 2,
'Scanned Images/CXR114_IM-0096': 2,
'Scanned Images/CXR1141_IM-0096': 2,
'Scanned Images/CXR1144_IM-0097': 2,
'Scanned Images/CXR1145_IM-0097': 2,
'Scanned Images/CXR1146_IM-0098': 2,
'Scanned Images/CXR1148_IM-0100': 2,
'Scanned Images/CXR1149_IM-0101': 2,
'Scanned Images/CXR115_IM-0102': 2,
'Scanned Images/CXR1150_IM-0102': 1,
'Scanned Images/CXR1151_IM-0102': 2,
'Scanned Images/CXR1152_IM-0103': 2,
'Scanned Images/CXR1153_IM-0104': 2,
'Scanned Images/CXR1154_IM-0104': 2,
'Scanned Images/CXR1155_IM-0105-0001': 1,
'Scanned Images/CXR1157_IM-0106': 2,
'Scanned Images/CXR1158_IM-0107': 2,
'Scanned Images/CXR1159_IM-0107': 1,
'Scanned Images/CXR116_IM-0107': 2,
'Scanned Images/CXR1160_IM-0107': 2,
'Scanned Images/CXR1161_IM-0107': 2,
'Scanned Images/CXR1163_IM-0108': 2,
'Scanned Images/CXR1165_IM-0110': 1,
'Scanned Images/CXR1166_IM-0111': 2,
'Scanned Images/CXR1167_IM-0112': 2,
'Scanned Images/CXR1168_IM-0112': 2,
'Scanned Images/CXR1169_IM-0113-0001': 1,
'Scanned Images/CXR1170_IM-0115': 3,
'Scanned Images/CXR1171_IM-0116': 2,
'Scanned Images/CXR1172_IM-0117': 1,
'Scanned Images/CXR1173_IM-0118': 2,
'Scanned Images/CXR1174_IM-0118': 2,
'Scanned Images/CXR1175_IM-0119': 2,
'Scanned Images/CXR1176_IM-0119': 2,
'Scanned Images/CXR1177_IM-0120': 2,
'Scanned Images/CXR1178_IM-0121': 2,
'Scanned Images/CXR1179_IM-0122': 2,
'Scanned Images/CXR118_IM-0123': 2,
'Scanned Images/CXR1180_IM-0123': 2,
```

```
'Scanned Images/CXR1182_IM-0124': 2,
'Scanned Images/CXR1184_IM-0124': 2,
'Scanned Images/CXR1187_IM-0126': 2,
'Scanned Images/CXR1188_IM-0127': 2,
'Scanned Images/CXR1189_IM-0128': 2,
'Scanned Images/CXR119_IM-0128': 1,
'Scanned Images/CXR1190_IM-0128': 3,
'Scanned Images/CXR1191_IM-0128': 2,
'Scanned Images/CXR1192_IM-0129': 2,
'Scanned Images/CXR1193_IM-0129': 2,
'Scanned Images/CXR1195_IM-0131': 2,
'Scanned Images/CXR1196_IM-0131': 2,
'Scanned Images/CXR1197_IM-0131': 2,
'Scanned Images/CXR1199_IM-0133': 2,
'Scanned Images/CXR12_IM-0133': 2,
'Scanned Images/CXR120_IM-0133': 2,
'Scanned Images/CXR1200_IM-0134': 2,
'Scanned Images/CXR1202_IM-0136': 2,
'Scanned Images/CXR1203_IM-0137': 2,
'Scanned Images/CXR1204_IM-0138': 2,
'Scanned Images/CXR1205_IM-0138': 2,
'Scanned Images/CXR1207_IM-0140-0001': 2,
'Scanned Images/CXR1208_IM-0141': 3,
'Scanned Images/CXR1209_IM-0142': 2,
'Scanned Images/CXR121_IM-0142': 2,
'Scanned Images/CXR1210_IM-0142': 2,
'Scanned Images/CXR1211_IM-0142': 2,
'Scanned Images/CXR1212_IM-0143': 2,
'Scanned Images/CXR1213_IM-0144': 2,
'Scanned Images/CXR1214_IM-0144': 2,
'Scanned Images/CXR1216_IM-0144': 1,
'Scanned Images/CXR1217_IM-0145-0001': 2,
'Scanned Images/CXR1218_IM-0146': 2,
'Scanned Images/CXR1219_IM-0146': 2,
'Scanned Images/CXR1220_IM-0148': 3,
'Scanned Images/CXR1221_IM-0149-0001': 2,
'Scanned Images/CXR1222_IM-0150': 2,
'Scanned Images/CXR1223_IM-0150': 2,
'Scanned Images/CXR1224_IM-0150': 1,
'Scanned Images/CXR1225_IM-0150': 2,
'Scanned Images/CXR1226_IM-0150': 2,
'Scanned Images/CXR1229_IM-0152': 2,
'Scanned Images/CXR123_IM-0153': 2,
'Scanned Images/CXR1231_IM-0155': 2,
'Scanned Images/CXR1232_IM-0156': 2,
'Scanned Images/CXR1233_IM-0157': 2,
'Scanned Images/CXR1234_IM-0157': 2,
'Scanned Images/CXR1235_IM-0158': 2,
'Scanned Images/CXR1236_IM-0158': 2,
```

```
'Scanned Images/CXR1237_IM-0159': 2,
'Scanned Images/CXR1238_IM-0160': 1,
'Scanned Images/CXR1240_IM-0162': 2,
'Scanned Images/CXR1241_IM-0163': 1,
'Scanned Images/CXR1242_IM-0164': 2,
'Scanned Images/CXR1243_IM-0165': 3,
'Scanned Images/CXR1244_IM-0166': 2,
'Scanned Images/CXR1246_IM-0167': 2,
'Scanned Images/CXR1248_IM-0168': 2,
'Scanned Images/CXR1249_IM-0169': 2,
'Scanned Images/CXR125_IM-0169': 2,
'Scanned Images/CXR1250_IM-0169': 2,
'Scanned Images/CXR1253_IM-0171-0001': 2,
'Scanned Images/CXR1254_IM-0172': 2,
'Scanned Images/CXR1255_IM-0172-1001': 2,
'Scanned Images/CXR1256_IM-0173': 2,
'Scanned Images/CXR1257_IM-0174': 2,
'Scanned Images/CXR1258_IM-0175': 1,
'Scanned Images/CXR1259_IM-0175': 2,
'Scanned Images/CXR126_IM-0176': 1,
'Scanned Images/CXR1261_IM-0177': 2,
'Scanned Images/CXR1262_IM-0178': 3,
'Scanned Images/CXR1263_IM-0179': 2,
'Scanned Images/CXR1264_IM-0179': 2,
'Scanned Images/CXR1265_IM-0179': 2,
'Scanned Images/CXR1266_IM-0179': 2,
'Scanned Images/CXR1267_IM-0179': 2,
'Scanned Images/CXR1268_IM-0180': 2,
'Scanned Images/CXR1269_IM-0181': 2,
'Scanned Images/CXR127_IM-0181': 2,
'Scanned Images/CXR1270_IM-0181': 2,
'Scanned Images/CXR1271_IM-0182': 2,
'Scanned Images/CXR1272_IM-0183': 2,
'Scanned Images/CXR1273_IM-0183': 2,
'Scanned Images/CXR1274_IM-0183': 2,
'Scanned Images/CXR1276_IM-0184': 3,
'Scanned Images/CXR1277_IM-0185': 2,
'Scanned Images/CXR1278_IM-0185': 2,
'Scanned Images/CXR1279_IM-0185': 1,
'Scanned Images/CXR128_IM-0186': 2,
'Scanned Images/CXR1280_IM-0187': 2,
'Scanned Images/CXR1281_IM-0188': 1,
'Scanned Images/CXR1282_IM-0188': 2,
'Scanned Images/CXR1283_IM-0188': 1,
'Scanned Images/CXR1284_IM-0188': 1,
'Scanned Images/CXR1285_IM-0188': 3,
'Scanned Images/CXR1287_IM-0188': 2,
'Scanned Images/CXR1288_IM-0189': 2,
'Scanned Images/CXR1289_IM-0189': 2,
```

```
'Scanned Images/CXR129_IM-0189': 2,
'Scanned Images/CXR1290_IM-0189': 1,
'Scanned Images/CXR1291_IM-0190': 2,
'Scanned Images/CXR1292_IM-0191': 2,
'Scanned Images/CXR1294_IM-0193': 2,
'Scanned Images/CXR1295_IM-0194': 2,
'Scanned Images/CXR13_IM-0198': 2,
'Scanned Images/CXR130_IM-0198': 1,
'Scanned Images/CXR1300_IM-0198': 1,
'Scanned Images/CXR1301_IM-0198': 2,
'Scanned Images/CXR1302_IM-0198': 2,
'Scanned Images/CXR1303_IM-0199-1001': 2,
'Scanned Images/CXR1303_IM-0199-2001': 3,
'Scanned Images/CXR1304_IM-0199': 2,
'Scanned Images/CXR1305_IM-0199': 2,
'Scanned Images/CXR1306_IM-0200': 2,
'Scanned Images/CXR1307_IM-0200': 1,
'Scanned Images/CXR1308_IM-0201': 2,
'Scanned Images/CXR1309_IM-0201-1001': 2,
'Scanned Images/CXR131_IM-0202': 2,
'Scanned Images/CXR1310_IM-0202': 2,
'Scanned Images/CXR1311_IM-0203': 2,
'Scanned Images/CXR1313_IM-0204': 1,
'Scanned Images/CXR1314_IM-0204': 2,
'Scanned Images/CXR1315_IM-0204': 2,
'Scanned Images/CXR1317_IM-0205': 2,
'Scanned Images/CXR1318_IM-0205': 2,
'Scanned Images/CXR1319_IM-0205': 2,
'Scanned Images/CXR132_IM-0206': 1,
'Scanned Images/CXR1320_IM-0207': 2,
'Scanned Images/CXR1322_IM-0208': 1,
'Scanned Images/CXR1323_IM-0209': 2,
'Scanned Images/CXR1324_IM-0209': 2,
'Scanned Images/CXR1327_IM-0211': 2,
'Scanned Images/CXR1328_IM-0211': 1,
'Scanned Images/CXR1329_IM-0211': 2,
'Scanned Images/CXR133_IM-0212': 3,
'Scanned Images/CXR1330_IM-0213': 2,
'Scanned Images/CXR1331_IM-0213': 1,
'Scanned Images/CXR1333_IM-0214': 2,
'Scanned Images/CXR1334_IM-0214': 2,
'Scanned Images/CXR1335_IM-0215': 2,
'Scanned Images/CXR1336_IM-0216': 1,
'Scanned Images/CXR1338_IM-0217': 1,
'Scanned Images/CXR1339_IM-0218': 2,
'Scanned Images/CXR134_IM-0219': 1,
'Scanned Images/CXR1340_IM-0220': 2,
'Scanned Images/CXR1341_IM-0220': 2,
'Scanned Images/CXR1342_IM-0221': 2,
```

```
'Scanned Images/CXR1343_IM-0222-0001': 2,
'Scanned Images/CXR1344_IM-0223': 2,
'Scanned Images/CXR1345_IM-0223': 2,
'Scanned Images/CXR1346_IM-0224': 3,
'Scanned Images/CXR1347_IM-0225': 2,
'Scanned Images/CXR1348_IM-0226-4004': 2,
'Scanned Images/CXR1349_IM-0227': 2,
'Scanned Images/CXR135_IM-0227': 2,
'Scanned Images/CXR1350_IM-0227': 2,
'Scanned Images/CXR1352_IM-0229': 2,
'Scanned Images/CXR1353_IM-0230': 2,
'Scanned Images/CXR1354_IM-0230': 2,
'Scanned Images/CXR1355_IM-0230': 2,
'Scanned Images/CXR1356_IM-0231': 2,
'Scanned Images/CXR1357_IM-0231': 1,
'Scanned Images/CXR1358_IM-0232': 3,
'Scanned Images/CXR1359_IM-0233': 2,
'Scanned Images/CXR136_IM-0233': 2,
'Scanned Images/CXR1360_IM-0234-0001': 2,
'Scanned Images/CXR1363_IM-0236-0001': 2,
'Scanned Images/CXR1364_IM-0237': 2,
'Scanned Images/CXR1365_IM-0237': 2,
'Scanned Images/CXR1366_IM-0237': 2,
'Scanned Images/CXR1367_IM-0237': 2,
'Scanned Images/CXR1368_IM-0237': 2,
'Scanned Images/CXR1369_IM-0238': 2,
'Scanned Images/CXR1370_IM-0239': 2,
'Scanned Images/CXR1372_IM-0239': 2,
'Scanned Images/CXR1373_IM-0240': 2,
'Scanned Images/CXR1374_IM-0240': 2,
'Scanned Images/CXR1376_IM-0242': 2,
'Scanned Images/CXR1377_IM-0242': 2,
'Scanned Images/CXR1378_IM-0242': 2,
'Scanned Images/CXR1379_IM-0243': 2,
'Scanned Images/CXR138_IM-0244': 2,
'Scanned Images/CXR1380_IM-0245': 2,
'Scanned Images/CXR1381_IM-0245': 2,
'Scanned Images/CXR1382_IM-0245': 2,
'Scanned Images/CXR1383_IM-0245': 2,
'Scanned Images/CXR1384_IM-0246': 2,
'Scanned Images/CXR1385_IM-0246': 2,
'Scanned Images/CXR1386_IM-0246': 2,
'Scanned Images/CXR1387_IM-0246': 1,
'Scanned Images/CXR1388_IM-0246': 2,
'Scanned Images/CXR139_IM-0248': 2,
'Scanned Images/CXR1390_IM-0249': 2,
'Scanned Images/CXR1391_IM-0250': 2,
'Scanned Images/CXR1392_IM-0251': 2,
'Scanned Images/CXR1393_IM-0251': 2,
```

```
'Scanned Images/CXR1396_IM-0252': 2,
'Scanned Images/CXR1398_IM-0254': 1,
'Scanned Images/CXR1399_IM-0255': 2,
'Scanned Images/CXR14_IM-0256': 2,
'Scanned Images/CXR1400_IM-0256': 2,
'Scanned Images/CXR1402_IM-0257': 3,
'Scanned Images/CXR1403_IM-0258': 2,
'Scanned Images/CXR1404_IM-0258': 2,
'Scanned Images/CXR1405_IM-0259': 2,
'Scanned Images/CXR1406_IM-0259': 2,
'Scanned Images/CXR1407_IM-0260': 2,
'Scanned Images/CXR1408_IM-0260': 1,
'Scanned Images/CXR1409_IM-0260': 2,
'Scanned Images/CXR141_IM-0260': 2,
'Scanned Images/CXR1410_IM-0260': 2,
'Scanned Images/CXR1411_IM-0261-0001': 2,
'Scanned Images/CXR1412_IM-0262': 2,
'Scanned Images/CXR1413_IM-0263': 2,
'Scanned Images/CXR1415_IM-0264': 2,
'Scanned Images/CXR1416_IM-0265': 2,
'Scanned Images/CXR1417_IM-0266': 2,
'Scanned Images/CXR1419_IM-0267': 2,
'Scanned Images/CXR142_IM-0267': 2,
'Scanned Images/CXR1421_IM-0269': 1,
'Scanned Images/CXR1422_IM-0269': 2,
'Scanned Images/CXR1423_IM-0270': 2,
'Scanned Images/CXR1424_IM-0271': 2,
'Scanned Images/CXR1425_IM-0272': 2,
'Scanned Images/CXR1427_IM-0273': 1,
'Scanned Images/CXR1428_IM-0274': 2,
'Scanned Images/CXR1429_IM-0275': 2,
'Scanned Images/CXR143_IM-0276': 2,
'Scanned Images/CXR1430_IM-0277': 1,
'Scanned Images/CXR1431_IM-0278': 2,
'Scanned Images/CXR1432_IM-0278': 2,
'Scanned Images/CXR1433_IM-0278': 2,
'Scanned Images/CXR1434_IM-0279': 2,
'Scanned Images/CXR1435_IM-0280': 2,
'Scanned Images/CXR1436_IM-0280': 2,
'Scanned Images/CXR1437_IM-0281': 2,
'Scanned Images/CXR1438_IM-0282': 2,
'Scanned Images/CXR1439_IM-0282': 2,
'Scanned Images/CXR144_IM-0283': 1,
'Scanned Images/CXR1440_IM-0284': 2,
'Scanned Images/CXR1441_IM-0285': 2,
'Scanned Images/CXR1442_IM-0286': 2,
'Scanned Images/CXR1443_IM-0286': 2,
'Scanned Images/CXR1444_IM-0286': 1,
'Scanned Images/CXR1445_IM-0287': 2,
```

```
'Scanned Images/CXR1447_IM-0289': 2,
'Scanned Images/CXR1448_IM-0289': 2,
'Scanned Images/CXR1449_IM-0290': 2,
'Scanned Images/CXR145_IM-0290': 2,
'Scanned Images/CXR1450_IM-0291': 2,
'Scanned Images/CXR1451_IM-0291': 2,
'Scanned Images/CXR1452_IM-0291': 2,
'Scanned Images/CXR1453_IM-0292': 2,
'Scanned Images/CXR1454_IM-0293': 2,
'Scanned Images/CXR1455_IM-0293': 2,
'Scanned Images/CXR1456_IM-0294': 2,
'Scanned Images/CXR1457_IM-0295': 1,
'Scanned Images/CXR1458_IM-0296': 2,
'Scanned Images/CXR1459_IM-0297': 3,
'Scanned Images/CXR1460_IM-0298': 2,
'Scanned Images/CXR1461_IM-0299': 2,
'Scanned Images/CXR1462_IM-0299': 2,
'Scanned Images/CXR1463_IM-0300': 1,
'Scanned Images/CXR1464_IM-0301': 2,
'Scanned Images/CXR1465_IM-0302': 2,
'Scanned Images/CXR1466_IM-0302': 2,
'Scanned Images/CXR1467_IM-0302': 2,
'Scanned Images/CXR1468_IM-0303': 2,
'Scanned Images/CXR1469_IM-0303': 2,
'Scanned Images/CXR147_IM-0303': 1,
'Scanned Images/CXR1470_IM-0303': 2,
'Scanned Images/CXR1471_IM-0304': 2,
'Scanned Images/CXR1472_IM-0305': 2,
'Scanned Images/CXR1473_IM-0306': 2,
'Scanned Images/CXR1474_IM-0307': 2,
'Scanned Images/CXR1475_IM-0307': 1,
'Scanned Images/CXR1476_IM-0308': 2,
'Scanned Images/CXR1477_IM-0309': 2,
'Scanned Images/CXR1478_IM-0310-0001': 3,
'Scanned Images/CXR1480_IM-0311': 2,
'Scanned Images/CXR1481_IM-0312': 2,
'Scanned Images/CXR1482_IM-0313': 1,
'Scanned Images/CXR1483_IM-0313': 2,
'Scanned Images/CXR1484_IM-0313': 2,
'Scanned Images/CXR1485_IM-0313': 2,
'Scanned Images/CXR1487_IM-0314': 2,
'Scanned Images/CXR1488_IM-0315': 2,
'Scanned Images/CXR1489_IM-0315': 2,
'Scanned Images/CXR1491_IM-0317': 2,
'Scanned Images/CXR1492_IM-0318': 2,
'Scanned Images/CXR1497_IM-0321': 2,
'Scanned Images/CXR1498_IM-0322': 2,
'Scanned Images/CXR1499_IM-0323': 2,
'Scanned Images/CXR15_IM-0324': 2,
```

```
'Scanned Images/CXR150_IM-0325-0001': 2,
'Scanned Images/CXR1500_IM-0326': 3,
'Scanned Images/CXR1501_IM-0327': 2,
'Scanned Images/CXR1502_IM-0328': 2,
'Scanned Images/CXR1505_IM-0330': 2,
'Scanned Images/CXR1506_IM-0330': 2,
'Scanned Images/CXR1508_IM-0330': 2,
'Scanned Images/CXR1509_IM-0331': 2,
'Scanned Images/CXR151_IM-0331': 2,
'Scanned Images/CXR1510_IM-0331': 2,
'Scanned Images/CXR1511_IM-0331': 2,
'Scanned Images/CXR1512_IM-0332': 2,
'Scanned Images/CXR1513_IM-0333': 2,
'Scanned Images/CXR1514_IM-0333': 2,
'Scanned Images/CXR1515_IM-0333': 2,
'Scanned Images/CXR1516_IM-0334': 2,
'Scanned Images/CXR1517_IM-0335': 2,
'Scanned Images/CXR1518_IM-0335': 2,
'Scanned Images/CXR1519_IM-0335': 2,
'Scanned Images/CXR152_IM-0335': 2,
'Scanned Images/CXR1520_IM-0336': 2,
'Scanned Images/CXR1521_IM-0337': 3,
'Scanned Images/CXR1522_IM-0338-0001': 1,
'Scanned Images/CXR1523_IM-0339': 3,
'Scanned Images/CXR1524_IM-0339': 2,
'Scanned Images/CXR1525_IM-0340': 4,
'Scanned Images/CXR1526_IM-0341': 2,
'Scanned Images/CXR1527_IM-0341-1001': 1,
'Scanned Images/CXR1528_IM-0341': 2,
'Scanned Images/CXR1529_IM-0342-0001': 2,
'Scanned Images/CXR153_IM-0343': 2,
'Scanned Images/CXR1530_IM-0344': 2,
'Scanned Images/CXR1531_IM-0344': 2,
'Scanned Images/CXR1532_IM-0344': 2,
'Scanned Images/CXR1533_IM-0344': 2,
'Scanned Images/CXR1534_IM-0345': 2,
'Scanned Images/CXR1535_IM-0346': 2,
'Scanned Images/CXR1537_IM-0348': 2,
'Scanned Images/CXR1538_IM-0348': 2,
'Scanned Images/CXR1539_IM-0349': 2,
'Scanned Images/CXR154_IM-0350': 2,
'Scanned Images/CXR1540_IM-0351': 2,
'Scanned Images/CXR1542_IM-0352': 2,
'Scanned Images/CXR1543_IM-0353': 2,
'Scanned Images/CXR1544_IM-0354': 2,
'Scanned Images/CXR1545_IM-0355': 2,
'Scanned Images/CXR1546_IM-0356': 2,
'Scanned Images/CXR1547_IM-0357': 2,
'Scanned Images/CXR1548_IM-0357': 2,
'Scanned Images/CXR1549_IM-0357': 1,
```

```
'Scanned Images/CXR1550_IM-0359': 2,
'Scanned Images/CXR1551_IM-0359': 2,
'Scanned Images/CXR1552_IM-0360': 2,
'Scanned Images/CXR1553_IM-0360': 2,
'Scanned Images/CXR1554_IM-0361': 2,
'Scanned Images/CXR1555_IM-0362': 2,
'Scanned Images/CXR1556_IM-0363': 2,
'Scanned Images/CXR1557_IM-0364': 1,
'Scanned Images/CXR1558_IM-0365': 2,
'Scanned Images/CXR1559_IM-0365': 2,
'Scanned Images/CXR1560_IM-0366': 2,
'Scanned Images/CXR1561_IM-0367': 2,
'Scanned Images/CXR1562_IM-0367': 2,
'Scanned Images/CXR1563_IM-0368': 2,
'Scanned Images/CXR1564_IM-0368': 2,
'Scanned Images/CXR1565_IM-0368': 2,
'Scanned Images/CXR1567_IM-0370': 2,
'Scanned Images/CXR1568_IM-0371': 2,
'Scanned Images/CXR1569_IM-0372': 2,
'Scanned Images/CXR157_IM-0372': 1,
'Scanned Images/CXR1570_IM-0372': 2,
'Scanned Images/CXR1571_IM-0373': 1,
'Scanned Images/CXR1572_IM-0373': 2,
'Scanned Images/CXR1573_IM-0374': 2,
'Scanned Images/CXR1574_IM-0374': 2,
'Scanned Images/CXR1576_IM-0375': 2,
'Scanned Images/CXR1577_IM-0375': 2,
'Scanned Images/CXR1578_IM-0376': 2,
'Scanned Images/CXR1579_IM-0376': 1,
'Scanned Images/CXR158_IM-0377': 2,
'Scanned Images/CXR1580_IM-0378': 2,
'Scanned Images/CXR1581_IM-0378': 2,
'Scanned Images/CXR1582_IM-0378': 2,
'Scanned Images/CXR1583_IM-0378': 2,
'Scanned Images/CXR1584_IM-0379': 2,
'Scanned Images/CXR1585_IM-0380': 2,
'Scanned Images/CXR1586_IM-0380': 2,
'Scanned Images/CXR1587_IM-0381': 2,
'Scanned Images/CXR1588_IM-0382': 2,
'Scanned Images/CXR1589_IM-0382': 2,
'Scanned Images/CXR159_IM-0382': 2,
'Scanned Images/CXR1590_IM-0383': 2,
'Scanned Images/CXR1591_IM-0384': 2,
'Scanned Images/CXR1592_IM-0385': 1,
'Scanned Images/CXR1593_IM-0385': 2,
'Scanned Images/CXR1594_IM-0385': 2,
'Scanned Images/CXR1595_IM-0386': 2,
'Scanned Images/CXR1596_IM-0387': 2,
'Scanned Images/CXR1597_IM-0388': 2,
```

```
'Scanned Images/CXR1598_IM-0389': 2,
'Scanned Images/CXR1599_IM-0389': 2,
'Scanned Images/CXR160_IM-0390': 2,
'Scanned Images/CXR1600_IM-0390': 2,
'Scanned Images/CXR1601_IM-0390': 2,
'Scanned Images/CXR1603_IM-0391': 2,
'Scanned Images/CXR1606_IM-0394': 1,
'Scanned Images/CXR1607_IM-0394': 1,
'Scanned Images/CXR1608_IM-0394': 2,
'Scanned Images/CXR1609_IM-0394': 2,
'Scanned Images/CXR161_IM-0394': 2,
'Scanned Images/CXR1610_IM-0395': 3,
'Scanned Images/CXR1612_IM-0397': 1,
'Scanned Images/CXR1616_IM-0399': 2,
'Scanned Images/CXR1617_IM-0399': 2,
'Scanned Images/CXR1619_IM-0400': 2,
'Scanned Images/CXR162_IM-0401': 2,
'Scanned Images/CXR1620_IM-0402': 2,
'Scanned Images/CXR1621_IM-0403': 2,
'Scanned Images/CXR1622_IM-0404': 2,
'Scanned Images/CXR1623_IM-0405': 2,
'Scanned Images/CXR1624_IM-0406': 2,
'Scanned Images/CXR1625_IM-0406': 2,
'Scanned Images/CXR1626_IM-0407': 2,
'Scanned Images/CXR1627_IM-0408': 2,
'Scanned Images/CXR1629_IM-0409': 1,
'Scanned Images/CXR163_IM-0410': 3,
'Scanned Images/CXR1630_IM-0411': 2,
'Scanned Images/CXR1631_IM-0412-0001': 2,
'Scanned Images/CXR1632_IM-0413': 2,
'Scanned Images/CXR1633_IM-0414': 3,
'Scanned Images/CXR1634_IM-0414': 2,
'Scanned Images/CXR1635_IM-0415': 2,
'Scanned Images/CXR1636_IM-0415': 2,
'Scanned Images/CXR1637_IM-0416': 1,
'Scanned Images/CXR1638_IM-0417': 3,
'Scanned Images/CXR1639_IM-0418': 2,
'Scanned Images/CXR164_IM-0419': 2,
'Scanned Images/CXR1640_IM-0420': 2,
'Scanned Images/CXR1641_IM-0420': 2,
'Scanned Images/CXR1642_IM-0421': 2,
'Scanned Images/CXR1643_IM-0421': 3,
'Scanned Images/CXR1644_IM-0422': 1,
'Scanned Images/CXR1645_IM-0422': 2,
'Scanned Images/CXR1646_IM-0423': 2,
'Scanned Images/CXR1647_IM-0424': 2,
'Scanned Images/CXR1648_IM-0425-0001': 2,
'Scanned Images/CXR1649_IM-0426': 2,
'Scanned Images/CXR165_IM-0427': 2,
```

```
'Scanned Images/CXR1652_IM-0428': 2,
'Scanned Images/CXR1653_IM-0429': 1,
'Scanned Images/CXR1654_IM-0430': 2,
'Scanned Images/CXR1655_IM-0431': 2,
'Scanned Images/CXR1656_IM-0431': 2,
'Scanned Images/CXR1657_IM-0432': 2,
'Scanned Images/CXR1658_IM-0433': 1,
'Scanned Images/CXR1659_IM-0434': 2,
'Scanned Images/CXR166_IM-0435': 2,
'Scanned Images/CXR1660_IM-0436': 2,
'Scanned Images/CXR1662_IM-0438': 1,
'Scanned Images/CXR1663_IM-0439': 2,
'Scanned Images/CXR1665_IM-0439': 2,
'Scanned Images/CXR1666_IM-0440-0001': 2,
'Scanned Images/CXR1667_IM-0441': 2,
'Scanned Images/CXR1668_IM-0441': 2,
'Scanned Images/CXR1669_IM-0441': 2,
'Scanned Images/CXR167_IM-0441': 2,
'Scanned Images/CXR1670_IM-0441': 2,
'Scanned Images/CXR1671_IM-0442': 2,
'Scanned Images/CXR1673_IM-0444': 2,
'Scanned Images/CXR1674_IM-0445': 2,
'Scanned Images/CXR1675_IM-0445': 2,
'Scanned Images/CXR1676_IM-0445': 2,
'Scanned Images/CXR1677_IM-0446': 3,
'Scanned Images/CXR1678_IM-0447-0001': 2,
'Scanned Images/CXR1679_IM-0448': 2,
'Scanned Images/CXR168_IM-0448': 2,
'Scanned Images/CXR1681_IM-0448': 2,
'Scanned Images/CXR1682_IM-0449': 2,
'Scanned Images/CXR1683_IM-0449': 2,
'Scanned Images/CXR1684_IM-0449': 2,
'Scanned Images/CXR1685_IM-0449': 2,
'Scanned Images/CXR1686_IM-0450': 2,
'Scanned Images/CXR1687_IM-0450': 2,
'Scanned Images/CXR1688_IM-0450': 2,
'Scanned Images/CXR1689_IM-0451': 2,
'Scanned Images/CXR169_IM-0452': 2,
'Scanned Images/CXR1691_IM-0453': 2,
'Scanned Images/CXR1693_IM-0454': 1,
'Scanned Images/CXR1694_IM-0455': 2,
'Scanned Images/CXR1695_IM-0456': 2,
'Scanned Images/CXR1696_IM-0457': 2,
'Scanned Images/CXR1697_IM-0458': 2,
'Scanned Images/CXR1698_IM-0458': 2,
'Scanned Images/CXR1699_IM-0459': 2,
'Scanned Images/CXR17_IM-0460': 2,
'Scanned Images/CXR1701_IM-0462': 2,
'Scanned Images/CXR1702_IM-0463': 2,
```

```
'Scanned Images/CXR1703_IM-0463': 2,
'Scanned Images/CXR1704_IM-0464': 2,
'Scanned Images/CXR1705_IM-0465': 1,
'Scanned Images/CXR1706_IM-0466': 2,
'Scanned Images/CXR1707_IM-0466': 2,
'Scanned Images/CXR1708_IM-0466': 2,
'Scanned Images/CXR1709_IM-0467': 2,
'Scanned Images/CXR1710_IM-0469': 2,
'Scanned Images/CXR1711_IM-0469': 2,
'Scanned Images/CXR1712_IM-0470': 2,
'Scanned Images/CXR1714_IM-0472': 1,
'Scanned Images/CXR1715_IM-0473': 1,
'Scanned Images/CXR1717_IM-0473': 2,
'Scanned Images/CXR1718_IM-0474': 1,
'Scanned Images/CXR1719_IM-0474': 2,
'Scanned Images/CXR172_IM-0474': 2,
'Scanned Images/CXR1720_IM-0475-0001': 2,
'Scanned Images/CXR1720_IM-0475': 1,
'Scanned Images/CXR1721_IM-0476': 2,
'Scanned Images/CXR1722_IM-0476': 1,
'Scanned Images/CXR1723_IM-0477': 2,
'Scanned Images/CXR1724_IM-0478': 1,
'Scanned Images/CXR1725_IM-0478': 2,
'Scanned Images/CXR1726_IM-0479': 2,
'Scanned Images/CXR1727_IM-0479': 2,
'Scanned Images/CXR1728_IM-0479': 2,
'Scanned Images/CXR1729_IM-0480': 2,
'Scanned Images/CXR173_IM-0481': 2,
'Scanned Images/CXR1730_IM-0481': 2,
'Scanned Images/CXR1731_IM-0481': 2,
'Scanned Images/CXR1732_IM-0482': 2,
'Scanned Images/CXR1734_IM-0484': 2,
'Scanned Images/CXR1735_IM-0484': 2,
'Scanned Images/CXR1736_IM-0484': 2,
'Scanned Images/CXR1737_IM-0485': 2,
'Scanned Images/CXR1738_IM-0486': 2,
'Scanned Images/CXR1739_IM-0487': 2,
'Scanned Images/CXR174_IM-0488': 2,
'Scanned Images/CXR1740_IM-0488': 2,
'Scanned Images/CXR1742_IM-0489': 2,
'Scanned Images/CXR1743_IM-0489': 2,
'Scanned Images/CXR1744_IM-0489': 2,
'Scanned Images/CXR1745_IM-0489': 2,
'Scanned Images/CXR1747_IM-0490': 2,
'Scanned Images/CXR1748_IM-0490': 2,
'Scanned Images/CXR175_IM-0492': 1,
'Scanned Images/CXR1750_IM-0493': 2,
'Scanned Images/CXR1751_IM-0494': 2,
'Scanned Images/CXR1752_IM-0494': 2,
```

```
'Scanned Images/CXR1753_IM-0494': 2,
'Scanned Images/CXR1757_IM-0495': 1,
'Scanned Images/CXR1758_IM-0495': 2,
'Scanned Images/CXR1759_IM-0495': 2,
'Scanned Images/CXR176_IM-0496': 2,
'Scanned Images/CXR1760_IM-0497': 2,
'Scanned Images/CXR1763_IM-0497': 2,
'Scanned Images/CXR1764_IM-0498': 1,
'Scanned Images/CXR1765_IM-0499': 2,
'Scanned Images/CXR1766_IM-0500': 2,
'Scanned Images/CXR1767_IM-0501-0001': 2,
'Scanned Images/CXR1768_IM-0502': 3,
'Scanned Images/CXR1769_IM-0503': 1,
'Scanned Images/CXR177_IM-0503': 2,
'Scanned Images/CXR1770_IM-0504': 2,
'Scanned Images/CXR1771_IM-0505': 2,
'Scanned Images/CXR1774_IM-0507': 1,
'Scanned Images/CXR1775_IM-0508': 2,
'Scanned Images/CXR1776_IM-0508': 2,
'Scanned Images/CXR1777_IM-0509': 2,
'Scanned Images/CXR1779_IM-0509': 2,
'Scanned Images/CXR178_IM-0509': 1,
'Scanned Images/CXR1780_IM-0509': 2,
'Scanned Images/CXR1781_IM-0509': 2,
'Scanned Images/CXR1782_IM-0510': 2,
'Scanned Images/CXR1785_IM-0512': 2,
'Scanned Images/CXR1786_IM-0512': 2,
'Scanned Images/CXR1787_IM-0513': 2,
'Scanned Images/CXR1788_IM-0513': 2,
'Scanned Images/CXR179_IM-0514': 2,
'Scanned Images/CXR1790_IM-0515': 1,
'Scanned Images/CXR1791_IM-0515': 2,
'Scanned Images/CXR1792_IM-0515': 2,
'Scanned Images/CXR1793_IM-0515': 1,
'Scanned Images/CXR1794_IM-0515': 2,
'Scanned Images/CXR1795_IM-0516': 2,
'Scanned Images/CXR1796_IM-0517': 2,
'Scanned Images/CXR1797_IM-0517': 2,
'Scanned Images/CXR1799_IM-0519': 2,
'Scanned Images/CXR18_IM-0520': 2,
'Scanned Images/CXR1800_IM-0520': 2,
'Scanned Images/CXR1801_IM-0520': 2,
'Scanned Images/CXR1802_IM-0521': 2,
'Scanned Images/CXR1804_IM-0522': 2,
'Scanned Images/CXR1806_IM-0524': 1,
'Scanned Images/CXR1807_IM-0524': 2,
'Scanned Images/CXR1808_IM-0524': 2,
'Scanned Images/CXR181_IM-0524': 2,
'Scanned Images/CXR1810_IM-0524': 2,
```

```
'Scanned Images/CXR1811_IM-0525': 1,
'Scanned Images/CXR1812_IM-0525': 2,
'Scanned Images/CXR1813_IM-0526': 2,
'Scanned Images/CXR1815_IM-0527': 2,
'Scanned Images/CXR1816_IM-0528': 2,
'Scanned Images/CXR1817_IM-0529': 1,
'Scanned Images/CXR182_IM-0531': 2,
'Scanned Images/CXR1820_IM-0532': 2,
'Scanned Images/CXR1821_IM-0532': 2,
'Scanned Images/CXR1822_IM-0533': 2,
'Scanned Images/CXR1823_IM-0534': 2,
'Scanned Images/CXR1824_IM-0535': 1,
'Scanned Images/CXR1826_IM-0535': 2,
'Scanned Images/CXR1827_IM-0535': 2,
'Scanned Images/CXR1829_IM-0537': 2,
'Scanned Images/CXR183_IM-0537': 2,
'Scanned Images/CXR1830_IM-0537': 1,
'Scanned Images/CXR1831_IM-0538': 2,
'Scanned Images/CXR1832_IM-0538': 2,
'Scanned Images/CXR1833_IM-0539': 2,
'Scanned Images/CXR1834_IM-0539': 2,
'Scanned Images/CXR1835_IM-0539': 2,
'Scanned Images/CXR1836_IM-0540': 2,
'Scanned Images/CXR1837_IM-0541': 1,
'Scanned Images/CXR1838_IM-0542': 1,
'Scanned Images/CXR1839_IM-0543': 2,
'Scanned Images/CXR184_IM-0544': 2,
'Scanned Images/CXR1840_IM-0545': 2,
'Scanned Images/CXR1841_IM-0545': 2,
'Scanned Images/CXR1843_IM-0546': 1,
'Scanned Images/CXR1844_IM-0547': 2,
'Scanned Images/CXR1846_IM-0549': 2,
'Scanned Images/CXR1847_IM-0550': 2,
'Scanned Images/CXR1848_IM-0550': 2,
'Scanned Images/CXR1849_IM-0550': 2,
'Scanned Images/CXR185_IM-0551': 2,
'Scanned Images/CXR1850_IM-0552': 1,
'Scanned Images/CXR1851_IM-0553': 2,
'Scanned Images/CXR1852_IM-0554': 2,
'Scanned Images/CXR1853_IM-0555': 2,
'Scanned Images/CXR1854_IM-0555': 2,
'Scanned Images/CXR1855_IM-0555': 2,
'Scanned Images/CXR1856_IM-0556': 1,
'Scanned Images/CXR1857_IM-0556': 2,
'Scanned Images/CXR186_IM-0558': 2,
'Scanned Images/CXR1860_IM-0558': 2,
'Scanned Images/CXR1861_IM-0558': 2,
'Scanned Images/CXR1863_IM-0558': 2,
'Scanned Images/CXR1864_IM-0558': 2,
```

```
'Scanned Images/CXR1865_IM-0558': 2,
'Scanned Images/CXR1866_IM-0559': 2,
'Scanned Images/CXR1867_IM-0560': 2,
'Scanned Images/CXR1868_IM-0561': 2,
'Scanned Images/CXR187_IM-0563': 2,
'Scanned Images/CXR1870_IM-0563': 2,
'Scanned Images/CXR1871_IM-0564': 2,
'Scanned Images/CXR1872_IM-0565': 1,
'Scanned Images/CXR1873_IM-0565': 2,
'Scanned Images/CXR1874_IM-0565': 2,
'Scanned Images/CXR1875_IM-0566': 2,
'Scanned Images/CXR1876_IM-0567': 2,
'Scanned Images/CXR1877_IM-0568': 2,
'Scanned Images/CXR1878_IM-0569': 2,
'Scanned Images/CXR1879_IM-0569': 2,
'Scanned Images/CXR1880_IM-0569': 1,
'Scanned Images/CXR1881_IM-0570': 2,
'Scanned Images/CXR1882_IM-0571': 2,
'Scanned Images/CXR1883_IM-0572': 2,
'Scanned Images/CXR1884_IM-0573': 3,
'Scanned Images/CXR1885_IM-0574': 2,
'Scanned Images/CXR1886_IM-0574': 2,
'Scanned Images/CXR1887_IM-0575': 1,
'Scanned Images/CXR1888_IM-0576': 2,
'Scanned Images/CXR1889_IM-0577': 2,
'Scanned Images/CXR189_IM-0578': 2,
'Scanned Images/CXR1891_IM-0580': 2,
'Scanned Images/CXR1892_IM-0580': 2,
'Scanned Images/CXR1893_IM-0580': 2,
'Scanned Images/CXR1894_IM-0581': 2,
'Scanned Images/CXR1895_IM-0581': 2,
'Scanned Images/CXR1896_IM-0581': 2,
'Scanned Images/CXR1897_IM-0581': 1,
'Scanned Images/CXR1898_IM-0581': 2,
'Scanned Images/CXR1899_IM-0582': 2,
'Scanned Images/CXR19_IM-0583': 2,
'Scanned Images/CXR190_IM-0583': 2,
'Scanned Images/CXR1900_IM-0584': 2,
'Scanned Images/CXR1901_IM-0585': 2,
'Scanned Images/CXR1902_IM-0586': 2,
'Scanned Images/CXR1903_IM-0586': 2,
'Scanned Images/CXR1905_IM-0587': 2,
'Scanned Images/CXR1908_IM-0590': 2,
'Scanned Images/CXR1909_IM-0590': 2,
'Scanned Images/CXR191_IM-0591': 2,
'Scanned Images/CXR1911_IM-0593': 2,
'Scanned Images/CXR1912_IM-0594': 2,
'Scanned Images/CXR1913_IM-0595': 2,
'Scanned Images/CXR1914_IM-0595': 2,
```

```
'Scanned Images/CXR1915_IM-0595': 2,
'Scanned Images/CXR1916_IM-0595': 2,
'Scanned Images/CXR1918_IM-0597': 2,
'Scanned Images/CXR1919_IM-0598': 3,
'Scanned Images/CXR192_IM-0598': 2,
'Scanned Images/CXR1920_IM-0598': 2,
'Scanned Images/CXR1921_IM-0598': 2,
'Scanned Images/CXR1922_IM-0598': 2,
'Scanned Images/CXR1923_IM-0598': 2,
'Scanned Images/CXR1924_IM-0598': 2,
'Scanned Images/CXR1925_IM-0599': 2,
'Scanned Images/CXR1926_IM-0600': 2,
'Scanned Images/CXR1927_IM-0600': 2,
'Scanned Images/CXR1928_IM-0600': 2,
'Scanned Images/CXR1929_IM-0600': 2,
'Scanned Images/CXR193_IM-0601': 2,
'Scanned Images/CXR1930_IM-0602-1001': 2,
'Scanned Images/CXR1931_IM-0602': 2,
'Scanned Images/CXR1932_IM-0603': 2,
'Scanned Images/CXR1933_IM-0604': 2,
'Scanned Images/CXR1934_IM-0604': 2,
'Scanned Images/CXR1935_IM-0605-0001': 2,
'Scanned Images/CXR1936_IM-0606': 2,
'Scanned Images/CXR1937_IM-0607': 2,
'Scanned Images/CXR1938_IM-0608': 2,
'Scanned Images/CXR1939_IM-0609': 1,
'Scanned Images/CXR194_IM-0609': 2,
'Scanned Images/CXR1940_IM-0610': 2,
'Scanned Images/CXR1941_IM-0610': 1,
'Scanned Images/CXR1942_IM-0611-0001': 1,
'Scanned Images/CXR1943_IM-0612': 2,
'Scanned Images/CXR1946_IM-0615': 2,
'Scanned Images/CXR1947_IM-0616': 2,
'Scanned Images/CXR1948_IM-0616': 2,
'Scanned Images/CXR195_IM-0618': 2,
'Scanned Images/CXR1950_IM-0618': 2,
'Scanned Images/CXR1951_IM-0619': 2,
'Scanned Images/CXR1952_IM-0620': 1,
'Scanned Images/CXR1953_IM-0621': 2,
'Scanned Images/CXR1954_IM-0622': 2,
'Scanned Images/CXR1956_IM-0623': 2,
'Scanned Images/CXR1957_IM-0624': 2,
'Scanned Images/CXR1958_IM-0625': 2,
'Scanned Images/CXR1959_IM-0625': 2,
'Scanned Images/CXR196_IM-0626': 1,
'Scanned Images/CXR1960_IM-0627': 2,
'Scanned Images/CXR1961_IM-0628': 2,
'Scanned Images/CXR1962_IM-0628': 2,
'Scanned Images/CXR1963_IM-0629': 2,
```

```
'Scanned Images/CXR1964_IM-0629': 2,
'Scanned Images/CXR1965_IM-0629': 2,
'Scanned Images/CXR1966_IM-0629': 2,
'Scanned Images/CXR1967_IM-0629': 2,
'Scanned Images/CXR1968_IM-0630': 2,
'Scanned Images/CXR197_IM-0631': 2,
'Scanned Images/CXR1970_IM-0632': 2,
'Scanned Images/CXR1972_IM-0633': 2,
'Scanned Images/CXR1973_IM-0633': 2,
'Scanned Images/CXR1974_IM-0633': 2,
'Scanned Images/CXR1978_IM-0636': 2,
'Scanned Images/CXR1979_IM-0637': 2,
'Scanned Images/CXR198_IM-0637': 1,
'Scanned Images/CXR1980_IM-0637': 2,
'Scanned Images/CXR1981_IM-0638': 2,
'Scanned Images/CXR1984_IM-0641-4001': 2,
'Scanned Images/CXR1985_IM-0642': 2,
'Scanned Images/CXR1986_IM-0643': 1,
'Scanned Images/CXR1987_IM-0644': 2,
'Scanned Images/CXR1990_IM-0648': 2,
'Scanned Images/CXR1991_IM-0648': 2,
'Scanned Images/CXR1992_IM-0649': 2,
'Scanned Images/CXR1993_IM-0650': 2,
'Scanned Images/CXR1994_IM-0651': 2,
'Scanned Images/CXR1995_IM-0651': 2,
'Scanned Images/CXR1997_IM-0651': 2,
'Scanned Images/CXR1999_IM-0651': 2,
'Scanned Images/CXR2_IM-0652': 2,
'Scanned Images/CXR20_IM-0653': 2,
'Scanned Images/CXR200_IM-0653': 2,
'Scanned Images/CXR2000_IM-0654': 2,
'Scanned Images/CXR2002_IM-0654': 2,
'Scanned Images/CXR2003_IM-0654': 2,
'Scanned Images/CXR2005_IM-0656': 2,
'Scanned Images/CXR2006_IM-0656': 2,
'Scanned Images/CXR2007_IM-0657-0001': 2,
'Scanned Images/CXR2008_IM-0658': 2,
'Scanned Images/CXR201_IM-0660': 2,
'Scanned Images/CXR2011_IM-0661': 2,
'Scanned Images/CXR2012_IM-0662': 2,
'Scanned Images/CXR2013_IM-0663': 2,
'Scanned Images/CXR2014_IM-0664': 2,
'Scanned Images/CXR2015_IM-0664': 2,
'Scanned Images/CXR2016_IM-0665': 2,
'Scanned Images/CXR2017_IM-0665': 2,
'Scanned Images/CXR2019_IM-0666': 2,
'Scanned Images/CXR202_IM-0667': 2,
'Scanned Images/CXR2020_IM-0668': 2,
'Scanned Images/CXR2021_IM-0668': 2,
```

```
 'Scanned Images/CXR2022_IM-0669': 2,
 'Scanned Images/CXR2023_IM-0669': 2,
 'Scanned Images/CXR2025_IM-0671': 1,
 'Scanned Images/CXR2026_IM-0671': 2,
 'Scanned Images/CXR2027_IM-0672-0001': 2,
 'Scanned Images/CXR2028_IM-0673': 2,
 'Scanned Images/CXR2029_IM-0674': 2,
 'Scanned Images/CXR203_IM-0675': 1,
 'Scanned Images/CXR2030_IM-0675': 2,
 'Scanned Images/CXR2031_IM-0676': 2,
 'Scanned Images/CXR2032_IM-0677': 2,
 'Scanned Images/CXR2033_IM-0678': 2,
 'Scanned Images/CXR2034_IM-0679': 2,
 'Scanned Images/CXR2035_IM-0680': 1,
 'Scanned Images/CXR2036_IM-0680': 2,
 'Scanned Images/CXR2038_IM-0682': 2,
 'Scanned Images/CXR2039_IM-0683': 2,
 'Scanned Images/CXR204_IM-0683': 1,
 'Scanned Images/CXR2041_IM-0685': 3,
 'Scanned Images/CXR2044_IM-0687': 2,
 'Scanned Images/CXR2045_IM-0687': 2,
 'Scanned Images/CXR2046_IM-0688': 2,
 'Scanned Images/CXR2047_IM-0688': 2,
 'Scanned Images/CXR2048_IM-0688': 2,
 'Scanned Images/CXR2052_IM-0690': 2,
 'Scanned Images/CXR2053_IM-0691': 2,
 'Scanned Images/CXR2056_IM-0694-1001': 2,
 'Scanned Images/CXR2058_IM-0696': 2,
 'Scanned Images/CXR2059_IM-0696': 2,
 'Scanned Images/CXR206_IM-0697': 1,
 'Scanned Images/CXR2060_IM-0698': 2,
 'Scanned Images/CXR2061_IM-0698': 2,
 'Scanned Images/CXR2062_IM-0699-0001': 2,
 'Scanned Images/CXR2065_IM-0701': 2,
 'Scanned Images/CXR2066_IM-0701': 2,
 'Scanned Images/CXR2067_IM-0701': 2,
 'Scanned Images/CXR2068_IM-0701': 2,
 'Scanned Images/CXR2069_IM-0702': 2,
 'Scanned Images/CXR207_IM-0703': 2,
 'Scanned Images/CXR2070_IM-0704': 2,
 ...}
```

```
len(images)
```

```
3350
```

```
def train_test_split(data):
    persons = list(data.keys())
    persons_train = persons[:2500]
    persons_cv = persons[2500:3000]
```

```
    persons_test = persons[3000:3350]
    return persons_train, persons_cv, persons_test

images_train, images_cv, images_test = train_test_split(images)

def combining_images(image_set):

    image_per_person = defaultdict(list)  # creating a list of
dictionary to store all the image paths
                                          #corresponding to a
person_id
    for pid in image_set:
        for img in dataset['Image_path'].values:
            if pid in img:
                image_per_person[pid].append(img)
            else:
                continue
    return image_per_person

img_per_person_train = combining_images(images_train)
img_per_person_cv = combining_images(images_cv)
img_per_person_test = combining_images(images_test)

len(img_per_person_train), len(images_train)
```

```
(2500, 2500)
```

```
img_per_person_train['Scanned Images/CXR1001_IM-0004']
```

```
['Scanned Images/CXR1001_IM-0004-1001.png',
 'Scanned Images/CXR1001_IM-0004-1002.png']
```

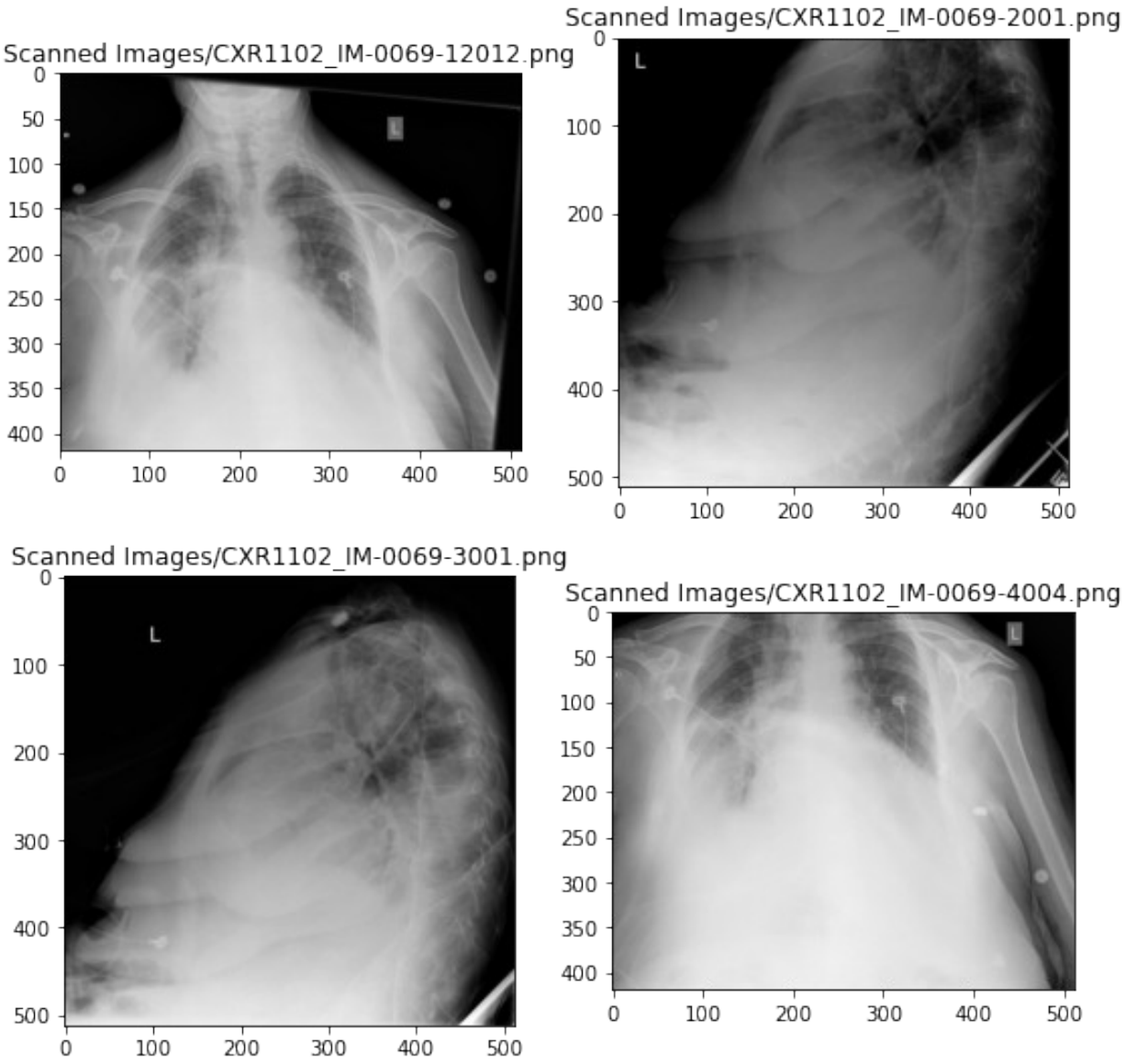
```
def load_image(file):
    img = tf.io.read_file(file)
    img = tf.image.decode_png(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return img

# just checking the ID which has 4 images
for k,v in images.items():
    if v == 4:
        print(k)
        break
```

```
Scanned Images/CXR1102_IM-0069
```

```
plt.figure(figsize=(9,9))
plt.subplot(221)
plt.imshow(load_image('Scanned Images/CXR1102_IM-0069-12012.png'))
plt.title('Scanned Images/CXR1102_IM-0069-12012.png')
plt.subplot(222)
plt.imshow(load_image('Scanned Images/CXR1102_IM-0069-2001.png'))
```

```
plt.title('Scanned Images/CXR1102_IM-0069-2001.png')
plt.subplot(223)
plt.imshow(load_image('Scanned Images/CXR1102_IM-0069-3001.png'))
plt.title('Scanned Images/CXR1102_IM-0069-3001.png')
plt.subplot(224)
plt.imshow(load_image('Scanned Images/CXR1102_IM-0069-4004.png'))
plt.title('Scanned Images/CXR1102_IM-0069-4004.png')
```

```
Text(0.5, 1.0, 'Scanned Images/CXR1102_IM-0069-4004.png')
```



2 side view and 2 front view images for the same ID

Sample chest scans of a person(4 images)

Now, we have multiple chest scans to produce a single report. Some person_ids have 1, some have 2 and the highest is 4. So we can take pairs of those images as input. If it has only one image, then it can be replicated.

```
def create_data(image_per_person):
    # new dataset
    person_id, image1, image2, report = [],[],[],[]
    for pid, imgs in image_per_person.items():   #contains pid and the
images associated with that pid

        if len(imgs) == 1:
            image1.append(imgs[0])
            image2.append(imgs[0])
            person_id.append(pid)
            report.append(findings[pid])
        else:
            num = 0
            a = itertools.combinations(imgs, 2)
            for i in a:
                image1.append(i[0])
                image2.append(i[1])
                person_id.append(pid + '_' + str(num))
                report.append(findings[pid])
                num += 1
    data = pd.DataFrame()
    data['Person_id'] = person_id
    data['Image1'] = image1
    data['Image2'] = image2
    data['Report'] = report

    return data

train = create_data(img_per_person_train)
test = create_data(img_per_person_test)
cv = create_data(img_per_person_cv)

train.head()
```

```
                            Person_id
Image1  \
0   Scanned Images/CXR1_1_IM-0001_0   Scanned Images/CXR1_1_IM-0001-
3001.png
1    Scanned Images/CXR10_IM-0002_0    Scanned Images/CXR10_IM-0002-
1001.png
2   Scanned Images/CXR100_IM-0002_0   Scanned Images/CXR100_IM-0002-
1001.png
3  Scanned Images/CXR1000_IM-0003_0  Scanned Images/CXR1000_IM-0003-
1001.png
4  Scanned Images/CXR1000_IM-0003_1  Scanned Images/CXR1000_IM-0003-
1001.png
```

```
                                       Image2  \
0   Scanned Images/CXR1_1_IM-0001-4001.png
1    Scanned Images/CXR10_IM-0002-2001.png
2   Scanned Images/CXR100_IM-0002-2001.png
3  Scanned Images/CXR1000_IM-0003-2001.png
4  Scanned Images/CXR1000_IM-0003-3001.png

                                                 Report
0  The cardiac silhouette and mediastinum size ar...
1  The cardiomediastinal silhouette is within nor...
2  Both lungs are clear and expanded. Heart and m...
3  There is XXXX increased opacity within the rig...
4  There is XXXX increased opacity within the rig...
```

Text Cleaning

```python
def lowercase(text):
    '''Converts to lowercase'''
    new_text = []
    for line in text:
        new_text.append(line.lower())
    return new_text

def decontractions(text):
    '''Performs decontractions in the doc'''
    new_text = []
    for phrase in text:
        phrase = re.sub(r"won't", "will not", phrase)
        phrase = re.sub(r"can\'t", "can not", phrase)
        phrase = re.sub(r"couldn\'t", "could not", phrase)
        phrase = re.sub(r"shouldn\'t", "should not", phrase)
        phrase = re.sub(r"wouldn\'t", "would not", phrase)
        # general
        phrase = re.sub(r"n\'t", " not", phrase)
        phrase = re.sub(r"\'re", " are", phrase)
        phrase = re.sub(r"\'s", " is", phrase)
        phrase = re.sub(r"\'d", " would", phrase)
        phrase = re.sub(r"\'ll", " will", phrase)
        phrase = re.sub(r"\'t", " not", phrase)
        phrase = re.sub(r"\'ve", " have", phrase)
        phrase = re.sub(r"\'m", " am", phrase)
        phrase = re.sub(r"\*+", "abuse", phrase)
        new_text.append(phrase)

    return new_text

def rem_punctuations(text):
    '''Removes punctuations'''
```

```
    punctuations = '''!()-[]{};:'"\,<>/?@#$%^&*~''' # full stop is not
removed
    new_text = []
    for line in text:
        for char in line:
            if char in punctuations:
                line = line.replace(char, "")
        new_text.append(' '.join(e for e in line.split()))
    return new_text

def rem_numbers(text):
    '''Removes numbers and irrelevant text like xxxx*'''
    new_text = []
    for line in text:
        temp = re.sub(r'x*','',line)
        new_text.append(re.sub(r'\d','',temp))
    return new_text

def words_filter(text):
    '''Removes words less than 2 characters except no and ct'''
    new_text = []
    for line in text:
        temp = line.split()
        temp2 = []
        for word in temp:
            if  len(word) <=2 and word != 'no' and word != 'ct':
                continue
            else:
                temp2.append(word)
        new_text.append(' '.join(e for e in temp2))
    return new_text

def multiple_fullstops(text):
    ''' Removes multiple full stops from the text'''
    new_text = []
    for line in text:
        new_text.append(re.sub(r'\.\.+', '.', line))
    return new_text

def fullstops(text):
    new_text = []
    for line in text:
        new_text.append(re.sub('\.', ' .', line))
    return new_text

def multiple_spaces(text):
    new_text = []
    for line in text:
        new_text.append(' '.join(e for e in line.split()))
    return new_text
```

```
def separting_startg_words(text):
    new_text = []
    for line in text:
        temp = []
        words = line.split()
        for i in words:
            if i.startswith('.') == False:
                temp.append(i)
            else:
                w = i.replace('.','. ')
                temp.append(w)
        new_text.append(' '.join(e for e in temp))
    return new_text

def rem_apostrophes(text):
    new_text = []
    for line in text:
        new_text.append(re.sub("'",'',line))
    return new_text

def text_preprocessing(text):
    '''Combines all the preprocess functions'''
    new_text = lowercase(text)
    new_text = decontractions(new_text)
    new_text = rem_punctuations(new_text)
    new_text = rem_numbers(new_text)
    new_text = words_filter(new_text)
    new_text = multiple_fullstops(new_text)
    new_text = fullstops(new_text)
    new_text = multiple_spaces(new_text)
    new_text = separting_startg_words(new_text)
    new_text = rem_apostrophes(new_text)
    return new_text

train['Report'] = text_preprocessing(train['Report'])
test['Report'] = text_preprocessing(test['Report'])
cv['Report'] = text_preprocessing(cv['Report'])

train.head()
```

```
                            Person_id
Image1  \
0   Scanned Images/CXR1_1_IM-0001_0   Scanned Images/CXR1_1_IM-0001-
3001.png
1    Scanned Images/CXR10_IM-0002_0    Scanned Images/CXR10_IM-0002-
1001.png
2   Scanned Images/CXR100_IM-0002_0   Scanned Images/CXR100_IM-0002-
1001.png
3  Scanned Images/CXR1000_IM-0003_0  Scanned Images/CXR1000_IM-0003-
```

```
1001.png
4  Scanned Images/CXR1000_IM-0003_1  Scanned Images/CXR1000_IM-0003-
1001.png
```

```
                                      Image2  \
0   Scanned Images/CXR1_1_IM-0001-4001.png
1    Scanned Images/CXR10_IM-0002-2001.png
2   Scanned Images/CXR100_IM-0002-2001.png
3  Scanned Images/CXR1000_IM-0003-2001.png
4  Scanned Images/CXR1000_IM-0003-3001.png
```

```
                                                 Report
0  the cardiac silhouette and mediastinum size ar...
1  the cardiomediastinal silhouette within normal...
2  both lungs are clear and epanded .  heart and ...
3  there increased opacity within the right upper...
4  there increased opacity within the right upper...
```

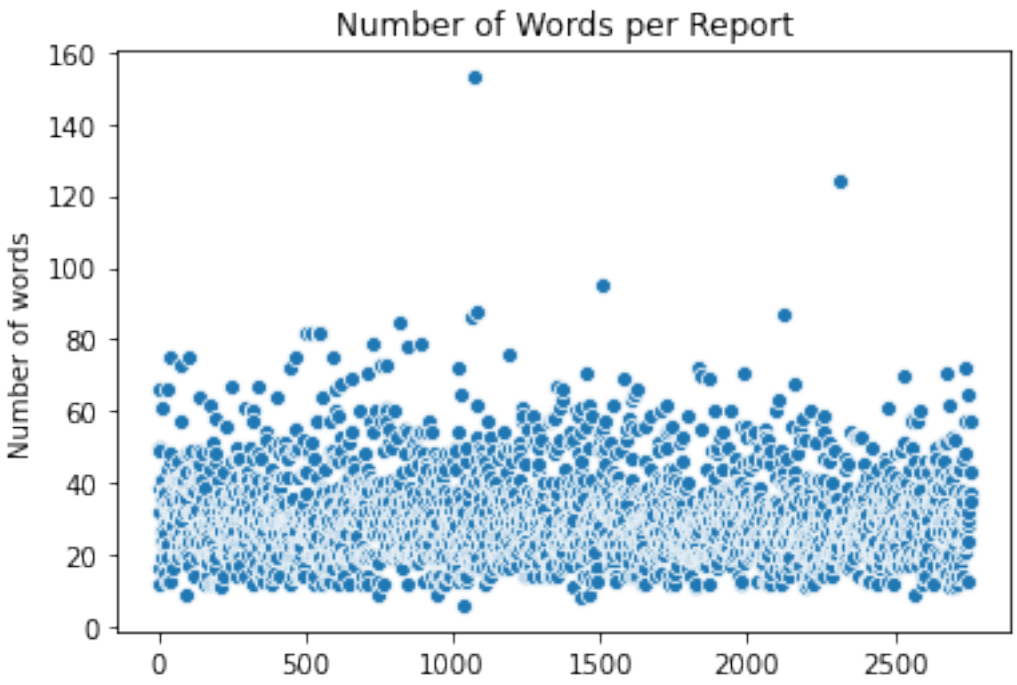
```
l = [len(e.split()) for e in train['Report'].values]  # Number of
words in each report
```

```
max(l)
```

```
153
```

```
plt.title('Number of Words per Report')
sns.scatterplot(range(train.shape[0]), l)
plt.ylabel('Number of words')
```

```
Text(0, 0.5, 'Number of words')
```



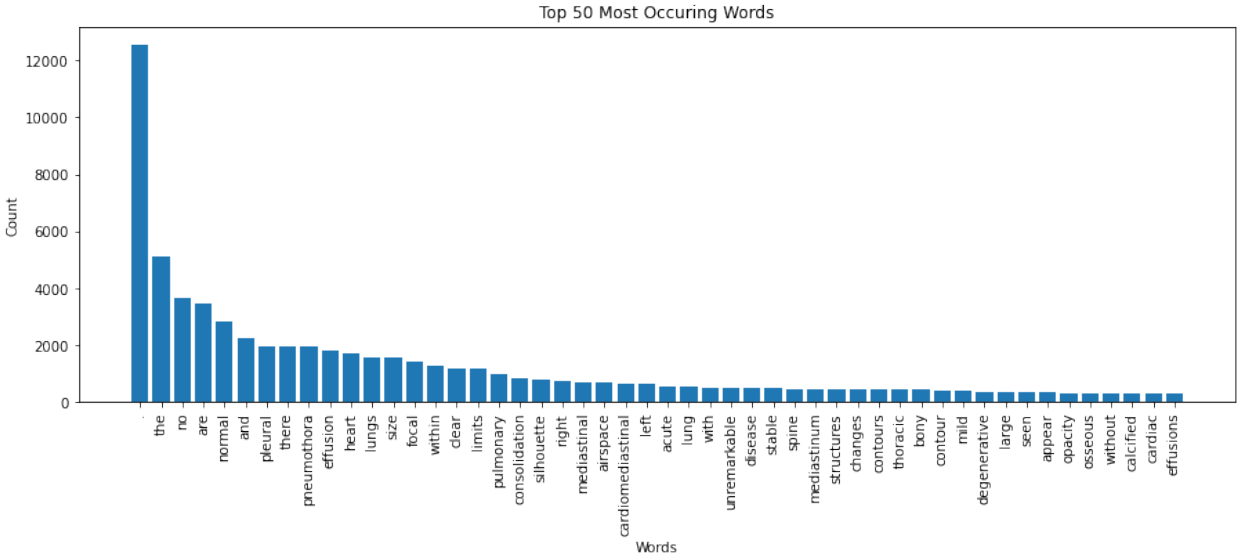
Most of the reports contain word count below 100

```
l = []
for i in train['Report'].values:
    l.extend(i.split())

c = Counter(l)

words = []
count = []
for k,v in c.items():
    words.append(k)
    count.append(v)
words_count = list(zip(count, words))

top_50_words = sorted(words_count)[::-1][:50]
bottom_50_words = sorted(words_count)[:50]

plt.figure(figsize=(15,5))
plt.bar(range(50), [c for c,w in top_50_words])
plt.title('Top 50 Most Occuring Words')
plt.xlabel('Words')
plt.ylabel('Count')
plt.xticks(ticks=range(50), labels=[w for c,w in top_50_words],
rotation=90)
```

```
([<matplotlib.axis.XTick at 0x2648e1ba788>,
  <matplotlib.axis.XTick at 0x2648e19bcc8>,
  <matplotlib.axis.XTick at 0x2648e1a23c8>,
```

```
 <matplotlib.axis.XTick at 0x2648f3bbe88>,
 <matplotlib.axis.XTick at 0x265e512d3c8>,
 <matplotlib.axis.XTick at 0x265e512d648>,
 <matplotlib.axis.XTick at 0x265e512ff88>,
 <matplotlib.axis.XTick at 0x265e51e5788>,
 <matplotlib.axis.XTick at 0x265e51ea1c8>,
 <matplotlib.axis.XTick at 0x265e51ead48>,
 <matplotlib.axis.XTick at 0x265e51ee848>,
 <matplotlib.axis.XTick at 0x265e51ea588>,
 <matplotlib.axis.XTick at 0x265e51f2248>,
 <matplotlib.axis.XTick at 0x265e51f2b48>,
 <matplotlib.axis.XTick at 0x265e51f3548>,
 <matplotlib.axis.XTick at 0x265e51f3f08>,
 <matplotlib.axis.XTick at 0x265e51f7a88>,
 <matplotlib.axis.XTick at 0x265e51fc608>,
 <matplotlib.axis.XTick at 0x265e51fa188>,
 <matplotlib.axis.XTick at 0x265e51facc8>,
 <matplotlib.axis.XTick at 0x265e5202848>,
 <matplotlib.axis.XTick at 0x265e51f2448>,
 <matplotlib.axis.XTick at 0x265e5202f08>,
 <matplotlib.axis.XTick at 0x265e5205908>,
 <matplotlib.axis.XTick at 0x265e5207308>,
 <matplotlib.axis.XTick at 0x265e5207cc8>,
 <matplotlib.axis.XTick at 0x265e520d848>,
 <matplotlib.axis.XTick at 0x265e52113c8>,
 <matplotlib.axis.XTick at 0x265e5211f08>,
 <matplotlib.axis.XTick at 0x265e5214a88>,
 <matplotlib.axis.XTick at 0x265e520d9c8>,
 <matplotlib.axis.XTick at 0x265e521a188>,
 <matplotlib.axis.XTick at 0x265e521ac88>,
 <matplotlib.axis.XTick at 0x265e521c688>,
 <matplotlib.axis.XTick at 0x265e5220088>,
 <matplotlib.axis.XTick at 0x265e5220b48>,
 <matplotlib.axis.XTick at 0x265e52236c8>,
 <matplotlib.axis.XTick at 0x265e5226248>,
 <matplotlib.axis.XTick at 0x265e5226d88>,
 <matplotlib.axis.XTick at 0x265e522c908>,
 <matplotlib.axis.XTick at 0x265e521a908>,
 <matplotlib.axis.XTick at 0x265e522cc48>,
 <matplotlib.axis.XTick at 0x265e522e648>,
 <matplotlib.axis.XTick at 0x265e5230048>,
 <matplotlib.axis.XTick at 0x265e5230a08>,
 <matplotlib.axis.XTick at 0x265e5235408>,
 <matplotlib.axis.XTick at 0x265e5235f08>,
 <matplotlib.axis.XTick at 0x265e5239a88>,
 <matplotlib.axis.XTick at 0x265e523c608>,
 <matplotlib.axis.XTick at 0x265e5230b48>],
[Text(0, 0, '.'),
 Text(1, 0, 'the'),
```

```
 Text(2, 0, 'no'),
 Text(3, 0, 'are'),
 Text(4, 0, 'normal'),
 Text(5, 0, 'and'),
 Text(6, 0, 'pleural'),
 Text(7, 0, 'there'),
 Text(8, 0, 'pneumothora'),
 Text(9, 0, 'effusion'),
 Text(10, 0, 'heart'),
 Text(11, 0, 'lungs'),
 Text(12, 0, 'size'),
 Text(13, 0, 'focal'),
 Text(14, 0, 'within'),
 Text(15, 0, 'clear'),
 Text(16, 0, 'limits'),
 Text(17, 0, 'pulmonary'),
 Text(18, 0, 'consolidation'),
 Text(19, 0, 'silhouette'),
 Text(20, 0, 'right'),
 Text(21, 0, 'mediastinal'),
 Text(22, 0, 'airspace'),
 Text(23, 0, 'cardiomediastinal'),
 Text(24, 0, 'left'),
 Text(25, 0, 'acute'),
 Text(26, 0, 'lung'),
 Text(27, 0, 'with'),
 Text(28, 0, 'unremarkable'),
 Text(29, 0, 'disease'),
 Text(30, 0, 'stable'),
 Text(31, 0, 'spine'),
 Text(32, 0, 'mediastinum'),
 Text(33, 0, 'structures'),
 Text(34, 0, 'changes'),
 Text(35, 0, 'contours'),
 Text(36, 0, 'thoracic'),
 Text(37, 0, 'bony'),
 Text(38, 0, 'contour'),
 Text(39, 0, 'mild'),
 Text(40, 0, 'degenerative'),
 Text(41, 0, 'large'),
 Text(42, 0, 'seen'),
 Text(43, 0, 'appear'),
 Text(44, 0, 'opacity'),
 Text(45, 0, 'osseous'),
 Text(46, 0, 'without'),
 Text(47, 0, 'calcified'),
 Text(48, 0, 'cardiac'),
 Text(49, 0, 'effusions')])
```

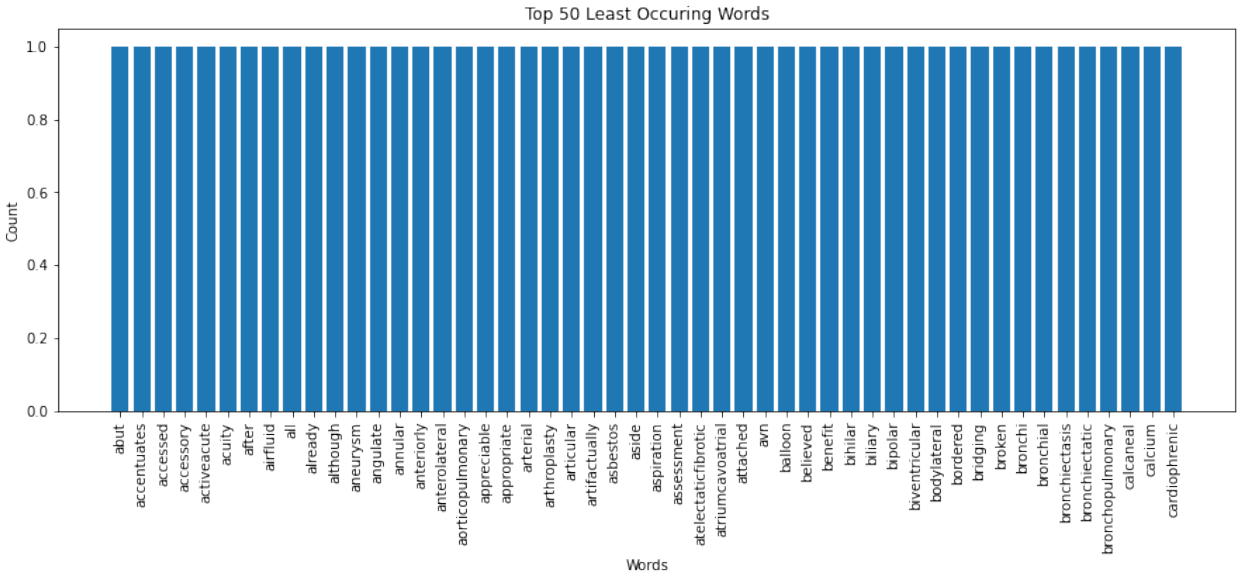



```
plt.figure(figsize=(15,5))
plt.bar(range(50), [c for c,w in bottom_50_words])
plt.title('Top 50 Least Occuring Words')
plt.xlabel('Words')
plt.ylabel('Count')
plt.xticks(ticks=range(50), labels=[w for c,w in bottom_50_words],
rotation=90)
```

```
([<matplotlib.axis.XTick at 0x265e60dd188>,
  <matplotlib.axis.XTick at 0x265e60dd8c8>,
  <matplotlib.axis.XTick at 0x265e60dd1c8>,
  <matplotlib.axis.XTick at 0x265e61907c8>,
  <matplotlib.axis.XTick at 0x265e6195088>,
  <matplotlib.axis.XTick at 0x265e61958c8>,
  <matplotlib.axis.XTick at 0x265e619a1c8>,
  <matplotlib.axis.XTick at 0x265e619a988>,
  <matplotlib.axis.XTick at 0x265e619e448>,
  <matplotlib.axis.XTick at 0x265e61a1048>,
  <matplotlib.axis.XTick at 0x265e6195988>,
  <matplotlib.axis.XTick at 0x265e61a1b88>,
  <matplotlib.axis.XTick at 0x265e61a3608>,
  <matplotlib.axis.XTick at 0x265e61a7108>,
  <matplotlib.axis.XTick at 0x265e61a7b88>,
  <matplotlib.axis.XTick at 0x265e61aa788>,
  <matplotlib.axis.XTick at 0x265e61ad388>,
  <matplotlib.axis.XTick at 0x265e61adf48>,
  <matplotlib.axis.XTick at 0x265e61b3b48>,
  <matplotlib.axis.XTick at 0x265e61aa1c8>,
  <matplotlib.axis.XTick at 0x265e61b5588>,
  <matplotlib.axis.XTick at 0x265e61b8088>,
  <matplotlib.axis.XTick at 0x265e61b8a48>,
  <matplotlib.axis.XTick at 0x265e61bb4c8>,
```

```
 <matplotlib.axis.XTick at 0x265e61bf148>,
 <matplotlib.axis.XTick at 0x265e61bfc88>,
 <matplotlib.axis.XTick at 0x265e6203888>,
 <matplotlib.axis.XTick at 0x265e6206488>,
 <matplotlib.axis.XTick at 0x265e61bf288>,
 <matplotlib.axis.XTick at 0x265e6206208>,
 <matplotlib.axis.XTick at 0x265e620b908>,
 <matplotlib.axis.XTick at 0x265e620d388>,
 <matplotlib.axis.XTick at 0x265e620ddc8>,
 <matplotlib.axis.XTick at 0x265e62109c8>,
 <matplotlib.axis.XTick at 0x265e62145c8>,
 <matplotlib.axis.XTick at 0x265e62171c8>,
 <matplotlib.axis.XTick at 0x265e6217d88>,
 <matplotlib.axis.XTick at 0x265e6210948>,
 <matplotlib.axis.XTick at 0x265e61b3d08>,
 <matplotlib.axis.XTick at 0x265e621ad88>,
 <matplotlib.axis.XTick at 0x265e621e808>,
 <matplotlib.axis.XTick at 0x265e6221288>,
 <matplotlib.axis.XTick at 0x265e6221cc8>,
 <matplotlib.axis.XTick at 0x265e6225808>,
 <matplotlib.axis.XTick at 0x265e6228408>,
 <matplotlib.axis.XTick at 0x265e622c088>,
 <matplotlib.axis.XTick at 0x265e6225448>,
 <matplotlib.axis.XTick at 0x265e622c188>,
 <matplotlib.axis.XTick at 0x265e622f488>,
 <matplotlib.axis.XTick at 0x265e622fec8>],
[Text(0, 0, 'abut'),
 Text(1, 0, 'accentuates'),
 Text(2, 0, 'accessed'),
 Text(3, 0, 'accessory'),
 Text(4, 0, 'activeacute'),
 Text(5, 0, 'acuity'),
 Text(6, 0, 'after'),
 Text(7, 0, 'airfluid'),
 Text(8, 0, 'all'),
 Text(9, 0, 'already'),
 Text(10, 0, 'although'),
 Text(11, 0, 'aneurysm'),
 Text(12, 0, 'angulate'),
 Text(13, 0, 'annular'),
 Text(14, 0, 'anteriorly'),
 Text(15, 0, 'anterolateral'),
 Text(16, 0, 'aorticopulmonary'),
 Text(17, 0, 'appreciable'),
 Text(18, 0, 'appropriate'),
 Text(19, 0, 'arterial'),
 Text(20, 0, 'arthroplasty'),
 Text(21, 0, 'articular'),
 Text(22, 0, 'artifactually'),
```

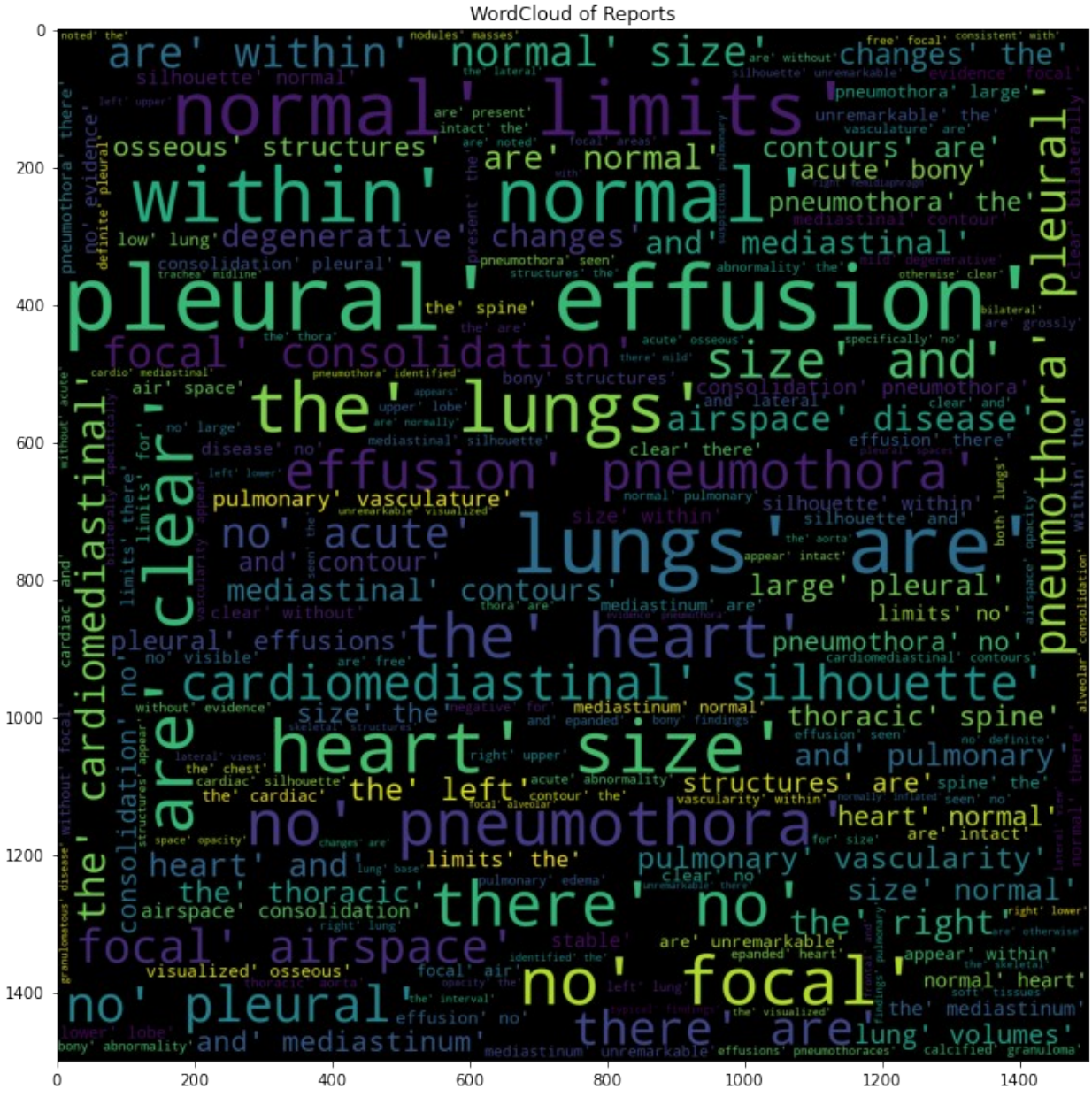
```
 Text(23, 0, 'asbestos'),
 Text(24, 0, 'aside'),
 Text(25, 0, 'aspiration'),
 Text(26, 0, 'assessment'),
 Text(27, 0, 'atelectaticfibrotic'),
 Text(28, 0, 'atriumcavoatrial'),
 Text(29, 0, 'attached'),
 Text(30, 0, 'avn'),
 Text(31, 0, 'balloon'),
 Text(32, 0, 'believed'),
 Text(33, 0, 'benefit'),
 Text(34, 0, 'bihilar'),
 Text(35, 0, 'biliary'),
 Text(36, 0, 'bipolar'),
 Text(37, 0, 'biventricular'),
 Text(38, 0, 'bodylateral'),
 Text(39, 0, 'bordered'),
 Text(40, 0, 'bridging'),
 Text(41, 0, 'broken'),
 Text(42, 0, 'bronchi'),
 Text(43, 0, 'bronchial'),
 Text(44, 0, 'bronchiectasis'),
 Text(45, 0, 'bronchiectatic'),
 Text(46, 0, 'bronchopulmonary'),
 Text(47, 0, 'calcaneal'),
 Text(48, 0, 'calcium'),
 Text(49, 0, 'cardiophrenic')])
```




```
w = WordCloud(height=1500, width=1500).generate(str(l))
```

```
plt.figure(figsize=(12,12))
plt.title('WordCloud of Reports')
plt.imshow(w)
```

```
<matplotlib.image.AxesImage at 0x265e780e648>
```




```
def remodelling(x):
    '''adds start and end tokens to a sentence '''
    return 'startseq' + ' ' + x + ' ' + 'endseq'
```

```
train['Report'] = train['Report'].apply(lambda x : remodelling(x))
test['Report'] = test['Report'].apply(lambda x : remodelling(x))
cv['Report'] = cv['Report'].apply(lambda x : remodelling(x))

# save the cleaned data(STRUCTURED DATA)
train.to_csv('Train_Data.csv', index=False)
test.to_csv('Test_Data.csv', index=False)
cv.to_csv('CV_Data.csv', index=False)
```

Experiment 4: Building a Binary Classification Model for Disease Diagnosis

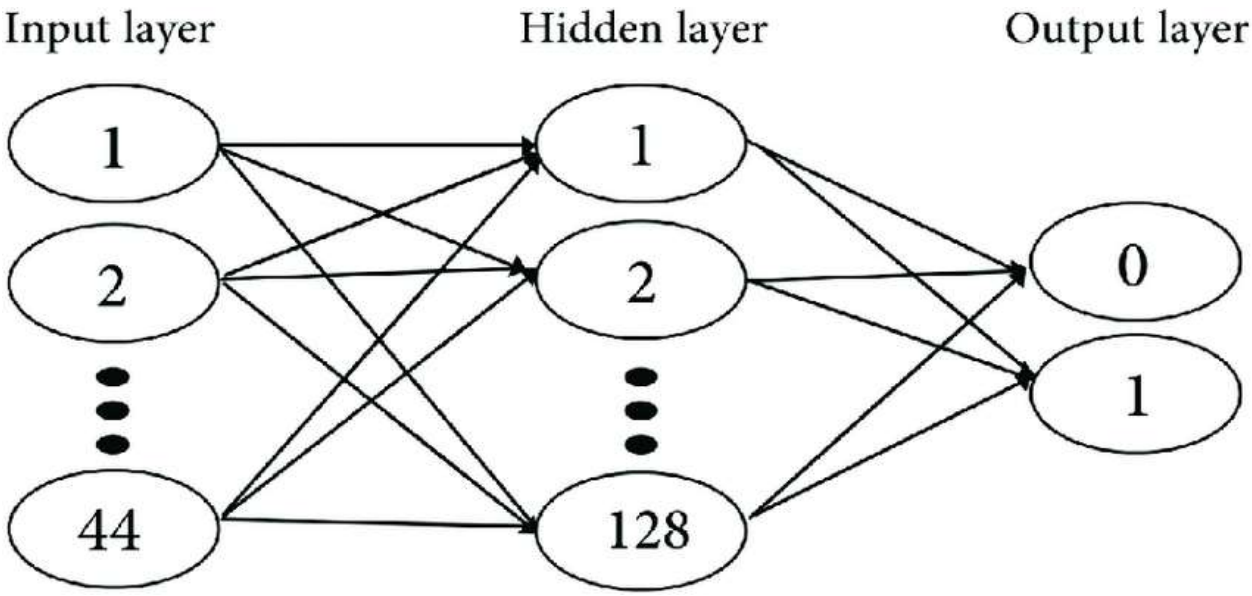
---

Aim:
To develop a binary classification model that accurately diagnoses diseases based on input medical features. This experiment will involve data preprocessing, model training, evaluation, and interpretation of results to assess the model's performance in clinical settings.

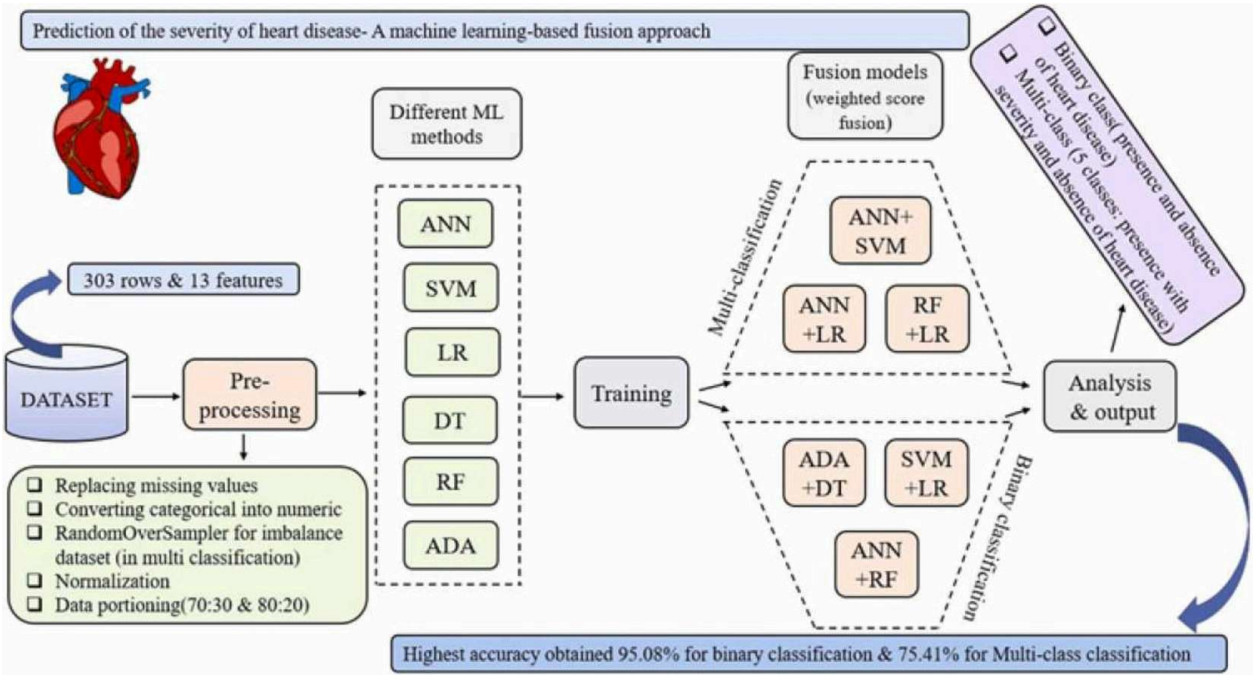
Theory:
Binary classification in the context of disease diagnosis involves predicting one of two classes—typically, "disease present" or "disease absent"—based on various input features derived from medical datasets. Machine learning algorithms, such as logistic regression, decision trees, support vector machines (SVM), or neural networks, can be employed for this task.

The key steps in building a binary classification model include:

1. **Data Collection:** Gather a suitable dataset containing relevant features (e.g., patient demographics, clinical measurements, lab results) and corresponding labels indicating disease presence or absence.
2. **Data Preprocessing:** Clean the dataset by handling missing values, encoding categorical variables, normalizing or standardizing numerical features, and splitting the data into training and testing sets.
3. **Model Selection:** Choose an appropriate machine learning algorithm based on the dataset's characteristics and the desired outcome.
4. **Model Training:** Train the model on the training dataset, allowing it to learn the patterns associated with the disease.
5. **Model Evaluation:** Evaluate the model using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to determine its performance on the testing set.
6. **Result Interpretation:** Analyze the results to understand how well the model performs and identify areas for improvement.
7. **Deployment:** Once validated, the model can be deployed in a clinical setting to assist healthcare professionals in diagnosing diseases based on new patient data.



Binary classification



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score

# loading the data from csv file to a Pandas DataFrame
parkinsons_data = pd.read_csv('parkinsons.csv')

# printing the first 5 rows of the dataframe
parkinsons_data.head()
```

```
             name  MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)
MDVP:Jitter(%)  \
0  phon_R01_S01_1      119.992       157.302        74.997
0.00784
1  phon_R01_S01_2      122.400       148.650       113.819
0.00968
2  phon_R01_S01_3      116.682       131.111       111.555
0.01050
3  phon_R01_S01_4      116.676       137.871       111.366
0.00997
4  phon_R01_S01_5      116.014       141.781       110.655
0.01284

   MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  ...
\
0           0.00007   0.00370   0.00554     0.01109       0.04374  ...

1           0.00008   0.00465   0.00696     0.01394       0.06134  ...

2           0.00009   0.00544   0.00781     0.01633       0.05233  ...

3           0.00009   0.00502   0.00698     0.01505       0.05492  ...

4           0.00011   0.00655   0.00908     0.01966       0.06425  ...


   Shimmer:DDA      NHR     HNR      RPDE       DFA   spread1
spread2  \
0      0.06545  0.02211  21.033  0.414783  0.815285 -4.813031
0.266482
1      0.09403  0.01929  19.085  0.458359  0.819521 -4.075192
0.335590
2      0.08270  0.01309  20.651  0.429895  0.825288 -4.443179
0.311173
3      0.08771  0.01353  20.644  0.434969  0.819235 -4.117501
0.334147
4      0.10470  0.01767  19.649  0.417356  0.823484 -3.747787
0.234513
```

```
          D2       PPE  status
0  2.301442  0.284654       1
1  2.486855  0.368674       1
2  2.342259  0.332634       1
3  2.405554  0.368975       1
4  2.332180  0.410335       1

[5 rows x 24 columns]
```

```
# number of rows and columns in the dataframe
parkinsons_data.shape
```

```
(195, 24)
```

```
# getting more information about the dataset
parkinsons_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   name              195 non-null    object 
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  RPDE              195 non-null    float64
 18  DFA               195 non-null    float64
 19  spread1           195 non-null    float64
 20  spread2           195 non-null    float64
 21  D2                195 non-null    float64
 22  PPE               195 non-null    float64
 23  status            195 non-null    int64  
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
# checking for missing values in each column
parkinsons_data.isnull().sum()
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
status              0
dtype: int64
```

```
# getting some statistical measures about the data
parkinsons_data.describe()
```

```
       MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
count   195.000000    195.000000    195.000000      195.000000
mean    154.228641    197.104918    116.324631        0.006220
std      41.390065     91.491548     43.521413        0.004848
min      88.333000    102.145000     65.476000        0.001680
25%     117.572000    134.862500     84.291000        0.003460
50%     148.790000    175.829000    104.315000        0.004940
75%     182.769000    224.205500    140.018500        0.007365
max     260.105000    592.030000    239.170000        0.033160

       MDVP:Jitter(Abs)    MDVP:RAP    MDVP:PPQ  Jitter:DDP
MDVP:Shimmer  \
count        195.000000  195.000000  195.000000  195.000000
195.000000
mean           0.000044    0.003306    0.003446    0.009920
0.029709
std            0.000035    0.002968    0.002759    0.008903
0.018857
```

```
min             0.000007    0.000680    0.000920    0.002040
0.009540
25%             0.000020    0.001660    0.001860    0.004985
0.016505
50%             0.000030    0.002500    0.002690    0.007490
0.022970
75%             0.000060    0.003835    0.003955    0.011505
0.037885
max             0.000260    0.021440    0.019580    0.064330
0.119080

       MDVP:Shimmer(dB)  ...  Shimmer:DDA         NHR         HNR
RPDE  \
count        195.000000  ...   195.000000  195.000000  195.000000
195.000000
mean           0.282251  ...     0.046993    0.024847   21.885974
0.498536
std            0.194877  ...     0.030459    0.040418    4.425764
0.103942
min            0.085000  ...     0.013640    0.000650    8.441000
0.256570
25%            0.148500  ...     0.024735    0.005925   19.198000
0.421306
50%            0.221000  ...     0.038360    0.011660   22.085000
0.495954
75%            0.350000  ...     0.060795    0.025640   25.075500
0.587562
max            1.302000  ...     0.169420    0.314820   33.047000
0.685151

              DFA     spread1     spread2          D2         PPE
status
count  195.000000  195.000000  195.000000  195.000000  195.000000
195.000000
mean     0.718099   -5.684397    0.226510    2.381826    0.206552
0.753846
std      0.055336    1.090208    0.083406    0.382799    0.090119
0.431878
min      0.574282   -7.964984    0.006274    1.423287    0.044539
0.000000
25%      0.674758   -6.450096    0.174351    2.099125    0.137451
1.000000
50%      0.722254   -5.720868    0.218885    2.361532    0.194052
1.000000
75%      0.761881   -5.046192    0.279234    2.636456    0.252980
1.000000
max      0.825288   -2.434031    0.450493    3.671155    0.527367
1.000000

[8 rows x 23 columns]
```

```
# distribution of target Variable
parkinsons_data['status'].value_counts()
```

```
status
1    147
0     48
Name: count, dtype: int64
```

```
print(parkinsons_data.isna().sum())  # Display the count of NaN values in each column
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
status              0
dtype: int64
```

```
print(parkinsons_data.dtypes)
```

```
name                 object
MDVP:Fo(Hz)         float64
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)      float64
MDVP:Jitter(Abs)    float64
MDVP:RAP            float64
MDVP:PPQ            float64
Jitter:DDP          float64
MDVP:Shimmer        float64
MDVP:Shimmer(dB)    float64
```

```
Shimmer:APQ3        float64
Shimmer:APQ5        float64
MDVP:APQ            float64
Shimmer:DDA         float64
NHR                 float64
HNR                 float64
RPDE                float64
DFA                 float64
spread1             float64
spread2             float64
D2                  float64
PPE                 float64
status                int64
dtype: object
```

```
# Selecting only numeric columns
numeric_columns =
parkinsons_data.select_dtypes(include=['number']).columns
mean_values = parkinsons_data.groupby('status')
[numeric_columns].mean()
print(mean_values)
```

```
        MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
status
0         181.937771    223.636750    145.207292        0.003866
1         145.180762    188.441463    106.893558        0.006989

        MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer
\
status

0               0.000023  0.001925  0.002056    0.005776      0.017615

1               0.000051  0.003757  0.003900    0.011273      0.033658


        MDVP:Shimmer(dB)  ...  Shimmer:DDA       NHR        HNR
RPDE  \
status                    ...

0               0.162958  ...     0.028511  0.011483  24.678750
0.442552
1               0.321204  ...     0.053027  0.029211  20.974048
0.516816

              DFA   spread1   spread2        D2       PPE  status
status
0        0.695716 -6.759264  0.160292  2.154491  0.123017     0.0
1        0.725408 -5.333420  0.248133  2.456058  0.233828     1.0
```

```
[2 rows x 23 columns]
```

```
X = parkinsons_data.drop(columns=['name','status'], axis=1)
Y = parkinsons_data['status']
```

```
print(X)
```

```
     MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0        119.992       157.302        74.997         0.00784
1        122.400       148.650       113.819         0.00968
2        116.682       131.111       111.555         0.01050
3        116.676       137.871       111.366         0.00997
4        116.014       141.781       110.655         0.01284
..           ...           ...           ...             ...
190      174.188       230.978        94.261         0.00459
191      209.516       253.017        89.488         0.00564
192      174.688       240.005        74.287         0.01360
193      198.764       396.961        74.904         0.00740
194      214.289       260.277        77.973         0.00567

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
0             0.00007   0.00370   0.00554     0.01109       0.04374
1             0.00008   0.00465   0.00696     0.01394       0.06134
2             0.00009   0.00544   0.00781     0.01633       0.05233
3             0.00009   0.00502   0.00698     0.01505       0.05492
4             0.00011   0.00655   0.00908     0.01966       0.06425
..                ...       ...       ...         ...           ...
190           0.00003   0.00263   0.00259     0.00790       0.04087
191           0.00003   0.00331   0.00292     0.00994       0.02751
192           0.00008   0.00624   0.00564     0.01873       0.02308
193           0.00004   0.00370   0.00390     0.01109       0.02296
194           0.00003   0.00295   0.00317     0.00885       0.01884

     MDVP:Shimmer(dB)  ...  MDVP:APQ  Shimmer:DDA      NHR     HNR
RPDE  \
0               0.426  ...   0.02971      0.06545  0.02211  21.033
0.414783
1               0.626  ...   0.04368      0.09403  0.01929  19.085
0.458359
2               0.482  ...   0.03590      0.08270  0.01309  20.651
0.429895
3               0.517  ...   0.03772      0.08771  0.01353  20.644
0.434969
4               0.584  ...   0.04465      0.10470  0.01767  19.649
0.417356
..                ...  ...       ...          ...      ...     ...
...
190             0.405  ...   0.02745      0.07008  0.02764  19.517
0.448439
```

```
191              0.263  ...   0.01879      0.04812  0.01810  19.147
0.431674
192              0.256  ...   0.01667      0.03804  0.10715  17.883
0.407567
193              0.241  ...   0.01588      0.03794  0.07223  19.020
0.451221
194              0.190  ...   0.01373      0.03078  0.04398  21.209
0.462803

          DFA   spread1   spread2        D2       PPE
0    0.815285 -4.813031  0.266482  2.301442  0.284654
1    0.819521 -4.075192  0.335590  2.486855  0.368674
2    0.825288 -4.443179  0.311173  2.342259  0.332634
3    0.819235 -4.117501  0.334147  2.405554  0.368975
4    0.823484 -3.747787  0.234513  2.332180  0.410335
..        ...       ...       ...       ...       ...
190  0.657899 -6.538586  0.121952  2.657476  0.133050
191  0.683244 -6.195325  0.129303  2.784312  0.168895
192  0.655683 -6.787197  0.158453  2.679772  0.131728
193  0.643956 -6.744577  0.207454  2.138608  0.123306
194  0.664357 -5.724056  0.190667  2.555477  0.148569

[195 rows x 22 columns]

print(Y)

0      1
1      1
2      1
3      1
4      1
      ..
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64

X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=2)

print(X.shape, X_train.shape, X_test.shape)

(195, 22) (156, 22) (39, 22)

scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
```

```
[[ 0.63239631 -0.02731081 -0.87985049 ... -0.97586547 -0.55160318
   0.07769494]
 [-1.05512719 -0.83337041 -0.9284778  ...  0.3981808  -0.61014073
   0.39291782]
 [ 0.02996187 -0.29531068 -1.12211107 ... -0.43937044 -0.62849605
  -0.50948408]
 ...
 [-0.9096785  -0.6637302  -0.160638   ...  1.22001022 -0.47404629
  -0.2159482 ]
 [-0.35977689  0.19731822 -0.79063679 ... -0.17896029 -0.47272835
   0.28181221]
 [ 1.01957066  0.19922317 -0.61914972 ... -0.716232    1.23632066
  -0.05829386]]
```

```
model = svm.SVC(kernel='linear')
```

```
# training the SVM model with training data
model.fit(X_train, Y_train)
```

```
SVC(kernel='linear')
```

```
# accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8846153846153846
```

```
# accuracy score on testing data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
```

```
print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8717948717948718
```

```
input_data =
(197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498
,0.01098,0.09700,0.00563,0.00680,0.00802,0.01689,0.00339,26.77500,0.42
2229,0.741367,-7.348300,0.177551,1.743867,0.085569)

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)

prediction = model.predict(std_data)
```

```
print(prediction)


if (prediction[0] == 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

Experiment 5: Implementing Multiclass Classification for Disease Severity Prediction

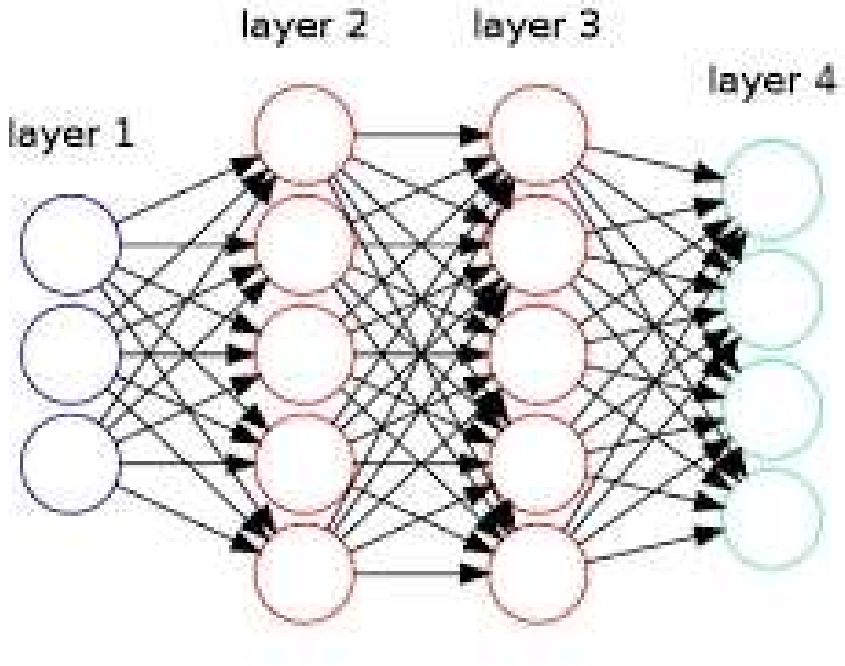
---

Aim:
To implement a multiclass classification model that predicts the severity of diseases based on various clinical and demographic features. This experiment focuses on developing a model that can classify the severity levels (e.g., mild, moderate, severe) and evaluate its performance using appropriate metrics.

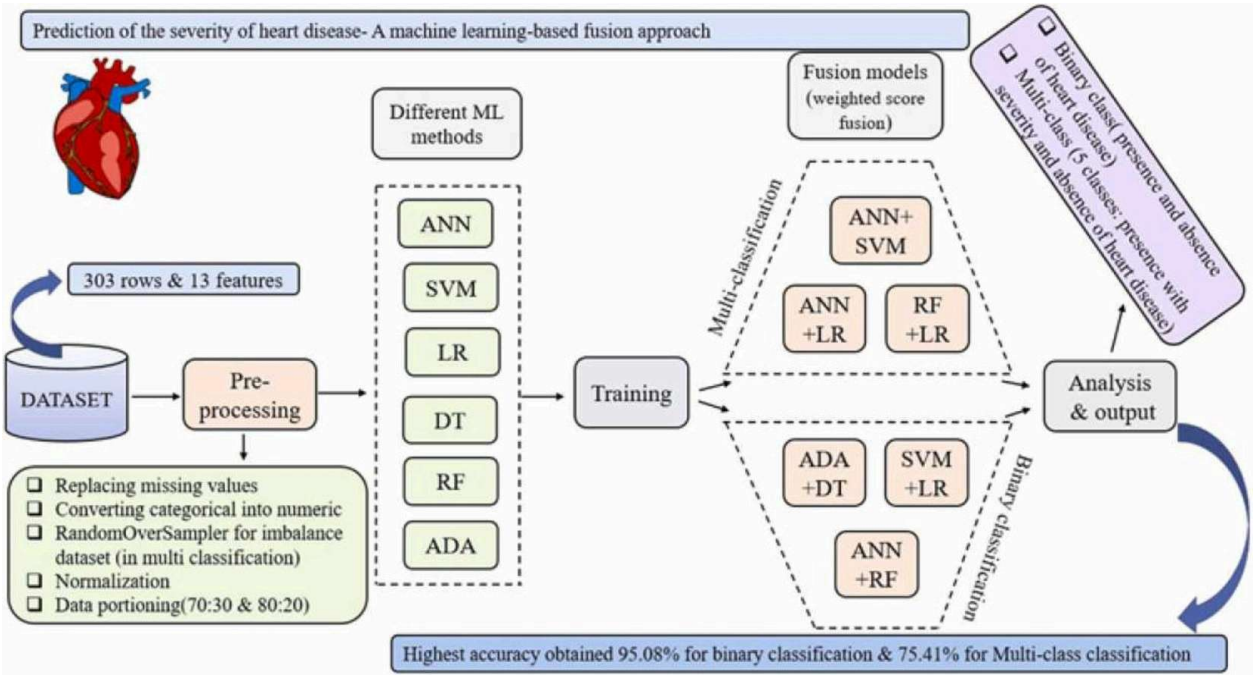
Theory:
Multiclass classification involves predicting one among three or more classes. In the context of disease severity prediction, the classes might represent different severity levels of a disease (e.g., mild, moderate, severe, or critical). Various machine learning algorithms, including decision trees, random forests, support vector machines (SVM), and neural networks, can be used for multiclass classification tasks.

The process of implementing a multiclass classification model generally includes the following steps:

1. **Data Collection:** Obtain a dataset that contains features related to patient demographics, clinical measurements, and labeled severity levels.
2. **Data Preprocessing:** Prepare the dataset by addressing missing values, encoding categorical variables, normalizing numerical features, and splitting the data into training and testing sets.
3. **Model Selection:** Choose a suitable machine learning algorithm for multiclass classification, considering the dataset's characteristics and the specific problem requirements.
4. **Model Training:** Train the model using the training dataset, allowing it to learn the relationships between features and severity levels.
5. **Model Evaluation:** Evaluate the model's performance on the testing set using metrics such as accuracy, precision, recall, F1-score, confusion matrix, and multiclass ROC-AUC.
6. **Result Interpretation:** Analyze the results to assess the model's effectiveness in predicting disease severity and identify potential areas for improvement.
7. **Deployment:** If the model performs satisfactorily, it can be deployed in clinical settings to assist healthcare professionals in predicting the severity of diseases based on new patient data.



Multiclass classification



```
from keras.models import Sequential
from keras.layers import Dense ,BatchNormalization
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler,StandardScaler
from tensorflow.keras.optimizers import Adam,SGD
from matplotlib import pyplot as plt
import numpy
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report
import sklearn.metrics as metrics
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
import numpy as np

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

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

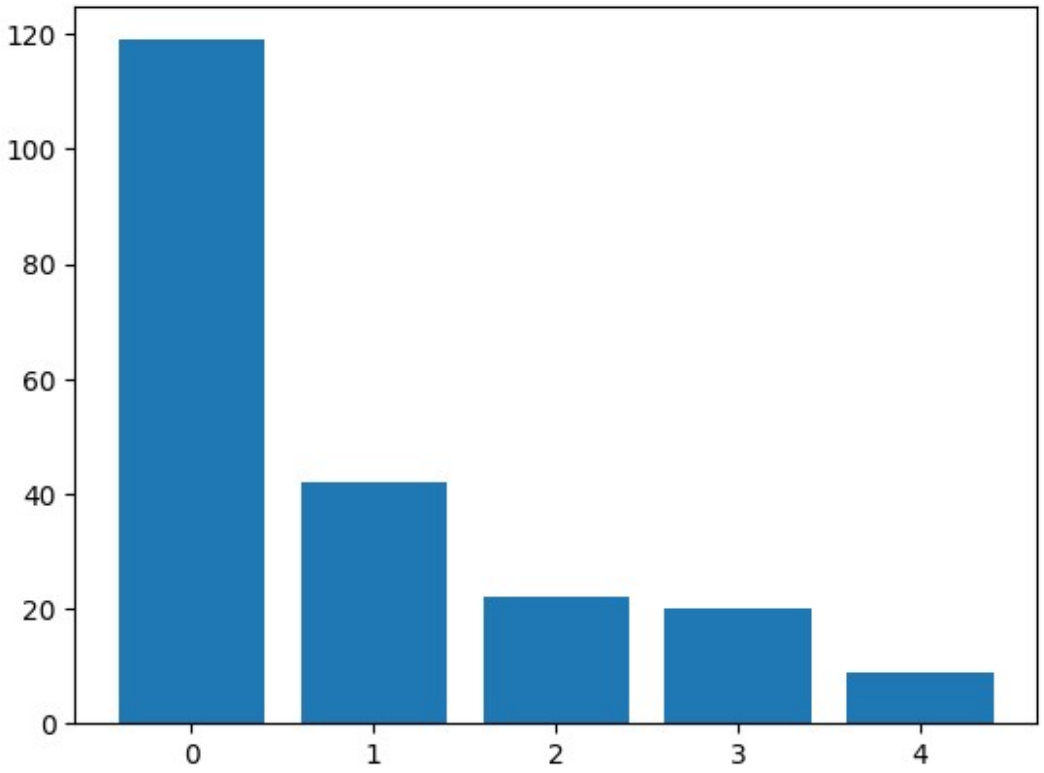
```
dataset =
numpy.loadtxt("/content/drive/MyDrive/multiclassification.csv",
delimiter=",", skiprows=1) # Skip the first row (header)

x = dataset[:,0:13]
y = dataset[:,13]

data = pd.read_csv('/content/drive/MyDrive/multiclassification.csv')

#for 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, shuffle=True)

#for 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, shuffle=True)

X_train.shape
```

```
(212, 13)
```

```
from pandas import read_csv
from collections import Counter
from matplotlib import pyplot
from sklearn.preprocessing import LabelEncoder

# summarize distribution
counter = Counter(y_train)
```

```
for k,v in counter.items():
      per = v / len(y_train) * 100
      print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
pyplot.bar(counter.keys(), counter.values())
pyplot.show()
```

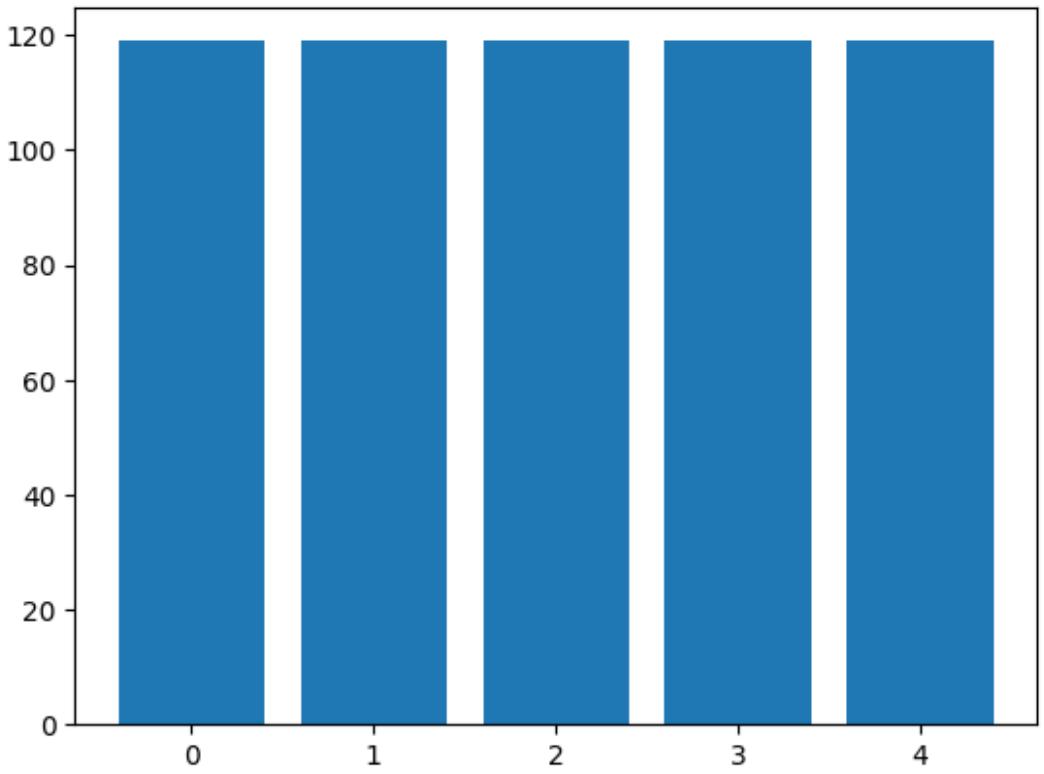
```
Class=0, n=119 (56.132%)
Class=2, n=22 (10.377%)
Class=1, n=42 (19.811%)
Class=4, n=9 (4.245%)
Class=3, n=20 (9.434%)
```




```
"""##### Over Sampling"""

from imblearn.over_sampling import RandomOverSampler

os=RandomOverSampler()
X_train_ns,y_train_ns=os.fit_resample(X_train,y_train)
print("The number of classes before fit {}".format(Counter(y_train)))
print("The number of classes after fit
{}".format(Counter(y_train_ns)))
```

```
The number of classes before fit Counter({0.0: 119, 1.0: 42, 2.0: 22,
3.0: 20, 4.0: 9})
```

```
The number of classes after fit Counter({0.0: 119, 2.0: 119, 1.0: 119,
4.0: 119, 3.0: 119})

# summarize distribution
counter = Counter(y_train_ns)
for k,v in counter.items():
      per = v / len(y_train_ns) * 100
      print('Class=%d, n=%d (%.3f%%)' % (k, v, per))
# plot the distribution
pyplot.bar(counter.keys(), counter.values())
pyplot.show()

Class=0, n=119 (20.000%)
Class=2, n=119 (20.000%)
Class=1, n=119 (20.000%)
Class=4, n=119 (20.000%)
Class=3, n=119 (20.000%)
```




```
X_train=X_train_ns
y_train=y_train_ns

X_train.shape

(595, 13)
```

```
#StandardScaler for normalization for some algortihms
sc = StandardScaler()

#min-max scaler for normalization for some algortihms
scaler = MinMaxScaler(feature_range=(0,1))

#code for svm

from sklearn.svm import SVC
model1 = SVC(probability=True,C=.01)

X_train_scaled = sc.fit_transform(X_train)
X_train_svc=X_train_scaled[:,0:13]
X_test_scaled = sc.transform(X_test)
X_test_svc=X_test_scaled[:,0:13]


model1.fit(X_train_svc,y_train)
rounded_predictions1 = model1.predict(X_test_svc)
predictions1 =model1.predict_proba(X_test_svc)
y_pred1 = rounded_predictions1

import sklearn.metrics as metrics
scores=metrics.accuracy_score(y_test,y_pred1)
print("SVM",scores)

print(metrics.confusion_matrix(y_test,y_pred1))
print(metrics.classification_report(y_test,y_pred1))
```

```
SVM 0.43956043956043955
[[34  5  6  0  1]
 [ 3  2  2  2  3]
 [ 2  1  2  4  5]
 [ 1  4  2  2  6]
 [ 0  0  3  1  0]]
              precision    recall  f1-score   support

         0.0       0.85      0.74      0.79        46
         1.0       0.17      0.17      0.17        12
         2.0       0.13      0.14      0.14        14
         3.0       0.22      0.13      0.17        15
         4.0       0.00      0.00      0.00         4

    accuracy                           0.44        91
   macro avg       0.27      0.24      0.25        91
weighted avg       0.51      0.44      0.47        91
```

```
# code for ANN

from keras.models import Sequential
```

```
from keras import regularizers
from keras.regularizers import l2
from keras.layers import Dense ,BatchNormalization
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import roc_auc_score
from tensorflow.keras.optimizers import Adam,SGD
scaler = MinMaxScaler(feature_range=(0,1))
# Define a Deep Learning Model
model4 = Sequential()
model4.add(Dense(8,
input_dim=13,use_bias=True,bias_initializer='zeros',
activation='relu', kernel_regularizer=l2(0.01) ))
model4.add(Dense(5, activation='softmax'))
# Changed 'lr' to 'learning_rate'
sgd = SGD(learning_rate=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model4.compile(loss='sparse_categorical_crossentropy',
optimizer='sgd',metrics=['accuracy'])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/base_opti
mizer.py:33: UserWarning: Argument `decay` is no longer supported and
will be ignored.
  warnings.warn(
```

```
#Normalize train & test sample
scaled_X_train_ann=scaler.fit_transform(X_train)
scaled_X_test_ann=scaler.transform(X_test)

# Train the Model

history=model4.fit(scaled_X_train_ann, y_train,validation_split=0.1,
batch_size=5,epochs=20,shuffle=True,verbose=2)


predictions4 =
model4.predict(scaled_X_test_ann,batch_size=5,verbose=0)
#rounded_predictions4 =
model4.predict_classes(scaled_X_test_ann,batch_size=5,verbose=0)
rounded_predictions4=np.argmax(predictions4,axis=1)
y_pred4 = rounded_predictions4
```

```
Epoch 1/20
107/107 - 4s - 34ms/step - accuracy: 0.1159 - loss: 1.7783 -
val_accuracy: 0.0000e+00 - val_loss: 1.7077
Epoch 2/20
```

```
107/107 - 0s - 2ms/step - accuracy: 0.2411 - loss: 1.6608 -
val_accuracy: 0.0000e+00 - val_loss: 1.9056
Epoch 3/20
107/107 - 0s - 3ms/step - accuracy: 0.3364 - loss: 1.6203 -
val_accuracy: 0.0000e+00 - val_loss: 1.9747
Epoch 4/20
107/107 - 0s - 2ms/step - accuracy: 0.3682 - loss: 1.5921 -
val_accuracy: 0.0000e+00 - val_loss: 2.0136
Epoch 5/20
107/107 - 0s - 3ms/step - accuracy: 0.3794 - loss: 1.5664 -
val_accuracy: 0.0000e+00 - val_loss: 2.0560
Epoch 6/20
107/107 - 0s - 2ms/step - accuracy: 0.4243 - loss: 1.5395 -
val_accuracy: 0.0000e+00 - val_loss: 2.0877
Epoch 7/20
107/107 - 0s - 2ms/step - accuracy: 0.4206 - loss: 1.5119 -
val_accuracy: 0.0000e+00 - val_loss: 2.1164
Epoch 8/20
107/107 - 0s - 3ms/step - accuracy: 0.4449 - loss: 1.4845 -
val_accuracy: 0.0000e+00 - val_loss: 2.1444
Epoch 9/20
107/107 - 0s - 3ms/step - accuracy: 0.4505 - loss: 1.4601 -
val_accuracy: 0.0000e+00 - val_loss: 2.1632
Epoch 10/20
107/107 - 0s - 2ms/step - accuracy: 0.4617 - loss: 1.4338 -
val_accuracy: 0.0000e+00 - val_loss: 2.1541
Epoch 11/20
107/107 - 0s - 2ms/step - accuracy: 0.4748 - loss: 1.4116 -
val_accuracy: 0.0000e+00 - val_loss: 2.1897
Epoch 12/20
107/107 - 0s - 4ms/step - accuracy: 0.4654 - loss: 1.3894 -
val_accuracy: 0.0000e+00 - val_loss: 2.1667
Epoch 13/20
107/107 - 1s - 6ms/step - accuracy: 0.4486 - loss: 1.3714 -
val_accuracy: 0.0000e+00 - val_loss: 2.1836
Epoch 14/20
107/107 - 0s - 3ms/step - accuracy: 0.4430 - loss: 1.3533 -
val_accuracy: 0.0000e+00 - val_loss: 2.1848
Epoch 15/20
107/107 - 0s - 2ms/step - accuracy: 0.5159 - loss: 1.3319 -
val_accuracy: 0.0000e+00 - val_loss: 2.1267
Epoch 16/20
107/107 - 0s - 2ms/step - accuracy: 0.4766 - loss: 1.3229 -
val_accuracy: 0.0000e+00 - val_loss: 2.1651
Epoch 17/20
107/107 - 0s - 3ms/step - accuracy: 0.5047 - loss: 1.3088 -
val_accuracy: 0.0000e+00 - val_loss: 2.1739
Epoch 18/20
107/107 - 1s - 6ms/step - accuracy: 0.4785 - loss: 1.2975 -
```

```
val_accuracy: 0.0000e+00 - val_loss: 2.1677
Epoch 19/20
107/107 - 0s - 3ms/step - accuracy: 0.4953 - loss: 1.2867 -
val_accuracy: 0.0000e+00 - val_loss: 2.1271
Epoch 20/20
107/107 - 0s - 5ms/step - accuracy: 0.5121 - loss: 1.2765 -
val_accuracy: 0.0000e+00 - val_loss: 2.1132
```

```
#print(predictions)
import sklearn.metrics as metrics
scores4=metrics.accuracy_score(y_test,y_pred4)
print("ANN",scores4)

print(metrics.confusion_matrix(y_test,y_pred4))
print(metrics.classification_report(y_test,y_pred4))
```

```
ANN 0.5274725274725275
[[36  6  3  1  0]
 [ 4  4  2  2  0]
 [ 1  2  3  8  0]
 [ 1  4  5  5  0]
 [ 0  1  1  2  0]]
              precision    recall  f1-score   support

         0.0       0.86      0.78      0.82        46
         1.0       0.24      0.33      0.28        12
         2.0       0.21      0.21      0.21        14
         3.0       0.28      0.33      0.30        15
         4.0       0.00      0.00      0.00         4

    accuracy                           0.53        91
   macro avg       0.32      0.33      0.32        91
weighted avg       0.54      0.53      0.53        91


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
```

```
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

#weighted fusion
#same procedure for every model(1,2 and 3). for every model we need to
take the prediction
# values and then the values will be merged according to the procedure
given in below.

p=1
#q=1-p
for i in range (19):
    p=p-.05
    q=1-p

    mixed=predictions1*p+predictions4*q


    out = []
    for i in range(len(y_test)):
        out.append(i)

    k=-1
    for i in range(len(y_test)):
        k=k+1
        for j in range(5):
            max=numpy.max(mixed[i])

        for j in range(5):
            if (mixed[i][j]== max):
                out[k]=j


    import numpy as np
    out = np.asarray(out)
    score3=metrics.accuracy_score(out,y_test)
    print("  mixed  p  q","%.4f" %score3, "%.2f" %p,"%.2f" %q)
```

```
  mixed  p  q 0.5385 0.95 0.05
  mixed  p  q 0.5385 0.90 0.10
  mixed  p  q 0.5275 0.85 0.15
  mixed  p  q 0.5275 0.80 0.20
  mixed  p  q 0.5275 0.75 0.25
  mixed  p  q 0.5275 0.70 0.30
  mixed  p  q 0.5275 0.65 0.35
  mixed  p  q 0.5275 0.60 0.40
  mixed  p  q 0.5275 0.55 0.45
  mixed  p  q 0.5275 0.50 0.50
  mixed  p  q 0.5275 0.45 0.55
```

```
  mixed  p  q 0.5275 0.40 0.60
  mixed  p  q 0.5275 0.35 0.65
  mixed  p  q 0.5275 0.30 0.70
  mixed  p  q 0.5275 0.25 0.75
  mixed  p  q 0.5275 0.20 0.80
  mixed  p  q 0.5275 0.15 0.85
  mixed  p  q 0.5275 0.10 0.90
  mixed  p  q 0.5275 0.05 0.95
```

```
#after selecting the value of p and q which gives the highest
accuracy.This value differs for different algorithms.
p=.55
q=1-p
mixed=predictions1*p+predictions4*q


out = []
for i in range(len(y_test)):
    out.append(i)

k=-1
#mixed
for i in range(len(y_test)):
    k=k+1
    for j in range(5):
        max=numpy.max(mixed[i])
    for j in range(5):
        if (mixed[i][j]== max):
            out[k]=j


import numpy as np
out = np.asarray(out)
score3=metrics.accuracy_score(out,y_test)
print("MIXED",score3)




print(metrics.confusion_matrix(y_test,out))
print(metrics.classification_report(y_test,out))

macro_roc_auc_ovo = roc_auc_score(y_test, mixed, multi_class="ovo",
average="macro")
weighted_roc_auc_ovo = roc_auc_score(y_test, mixed, multi_class="ovo",
average="weighted")
macro_roc_auc_ovr = roc_auc_score(y_test, mixed,
multi_class="ovr",average="macro")
weighted_roc_auc_ovr = roc_auc_score(y_test, mixed,
multi_class="ovr",average="weighted")
```

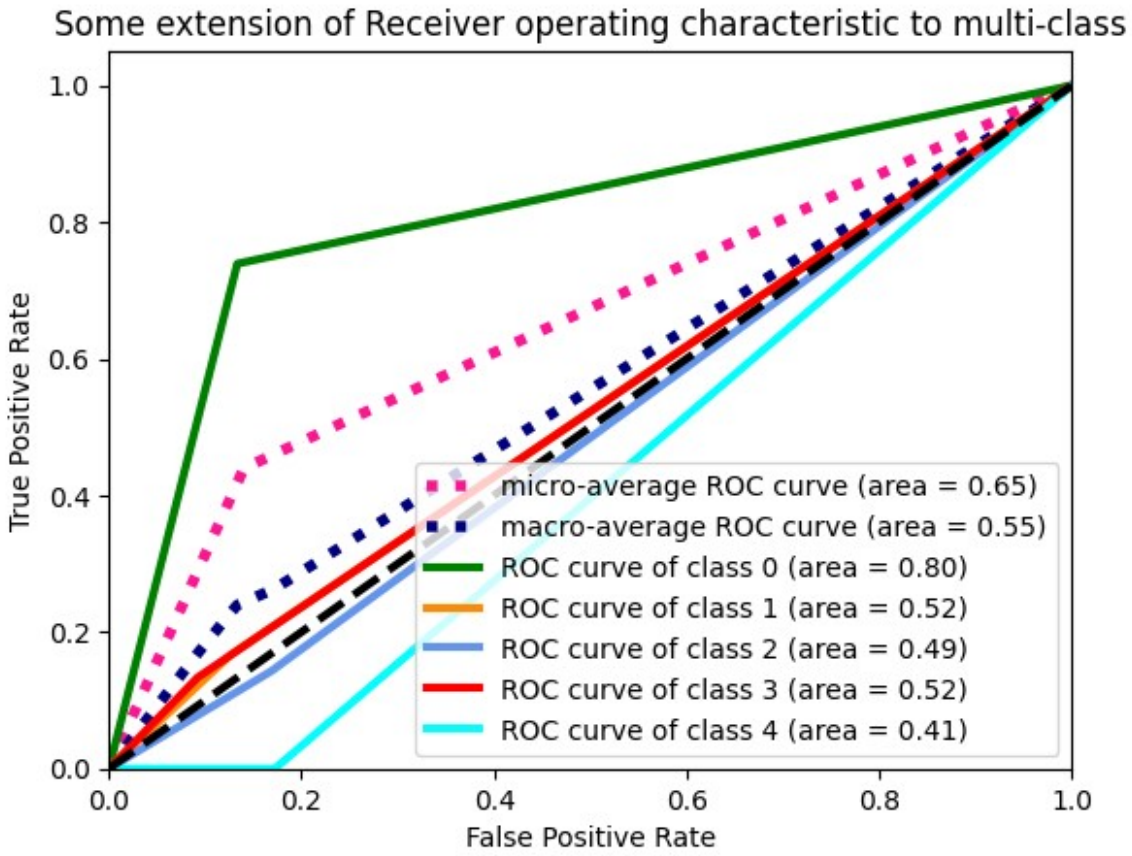
```
#print all score for roc
print("macro_ovo,weight_ovo,macro_ovr,weight_ovr",
        macro_roc_auc_ovo,
weighted_roc_auc_ovo,macro_roc_auc_ovr ,weighted_roc_auc_ovr)
```

```
MIXED 0.5274725274725275
[[36  6  3  1  0]
 [ 4  4  2  2  0]
 [ 1  2  3  8  0]
 [ 1  4  5  5  0]
 [ 0  1  1  2  0]]
              precision    recall  f1-score   support

         0.0       0.86      0.78      0.82        46
         1.0       0.24      0.33      0.28        12
         2.0       0.21      0.21      0.21        14
         3.0       0.28      0.33      0.30        15
         4.0       0.00      0.00      0.00         4

    accuracy                           0.53        91
   macro avg       0.32      0.33      0.32        91
weighted avg       0.54      0.53      0.53        91

macro_ovo,weight_ovo,macro_ovr,weight_ovr 0.68995082815735
0.7568826359975429 0.7658332578241531 0.8086500558716126

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/
_classification.py:1531: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classificatio
n.py:1531: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division`
parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
```

```
#ROC curve for SVM


from sklearn.preprocessing import label_binarize
y_test_h = label_binarize(y_test, classes=[0, 1, 2, 3, 4])
```

```
y_score1 = label_binarize(y_pred1, classes=[0, 1, 2, 3, 4])
from sklearn.metrics import roc_curve, auc
# Import interp from scipy.interpolate instead of scipy
from scipy.interpolate import interp1d as interp # interp1d is the
most likely replacement
from itertools import cycle
n_classes=5
lw=3
# 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_h[:, i], y_score1[:, 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_h.ravel(),
y_score1.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
   f = interp(fpr[i], tpr[i])
   mean_tpr += f(all_fpr)

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['green', 'darkorange', 'cornflowerblue','red','aqua'])
```

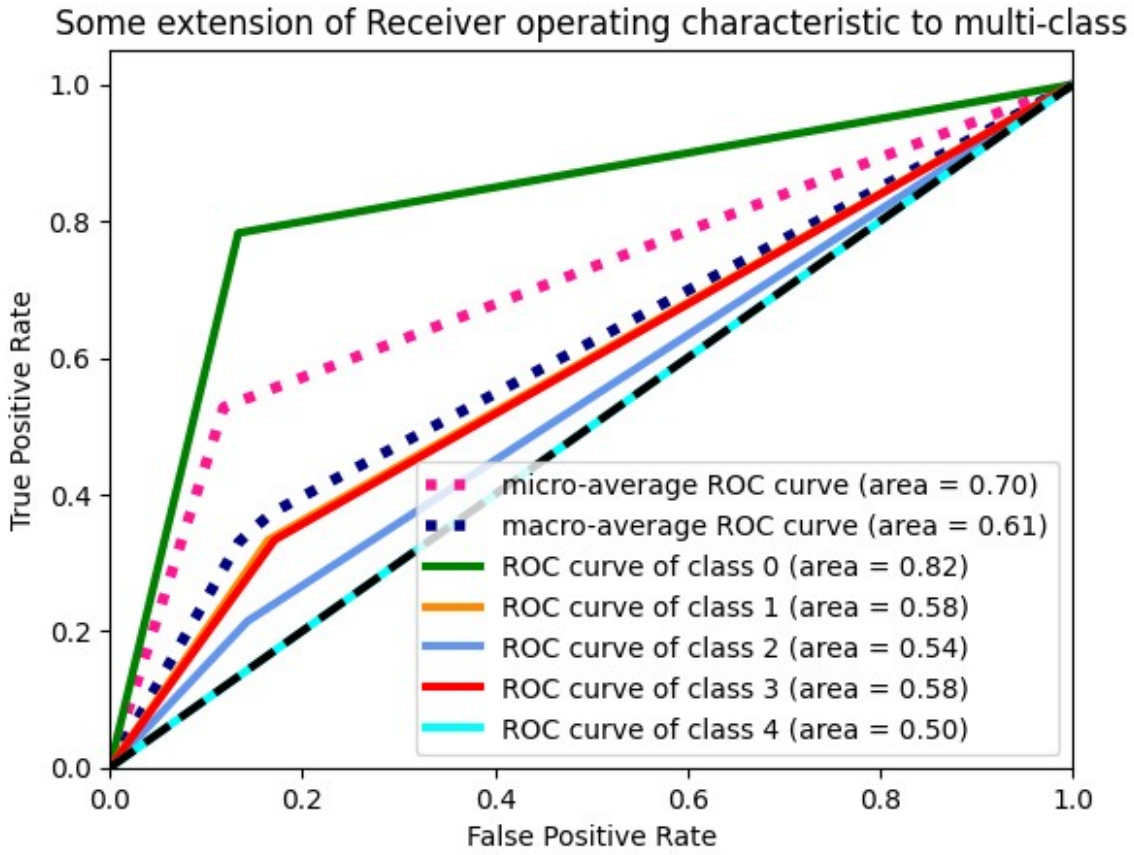
```
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to
multi-class')
plt.legend(loc="lower right")
plt.show()
```




```
#ROC curve for ANN


from sklearn.preprocessing import label_binarize
y_test_h = label_binarize(y_test, classes=[0, 1, 2, 3, 4])
```

```
y_score1 = label_binarize(y_pred4, classes=[0, 1, 2, 3, 4])
from sklearn.metrics import roc_curve, auc
from scipy.interpolate import interp1d as interp
from itertools import cycle
n_classes=5
lw=3
# 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_h[:, i], y_score1[:, 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_h.ravel(),
y_score1.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])


# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    f = interp(fpr[i], tpr[i])
    mean_tpr += f(all_fpr)

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['green', 'darkorange', 'cornflowerblue','red','aqua'])
for i, color in zip(range(n_classes), colors):
```

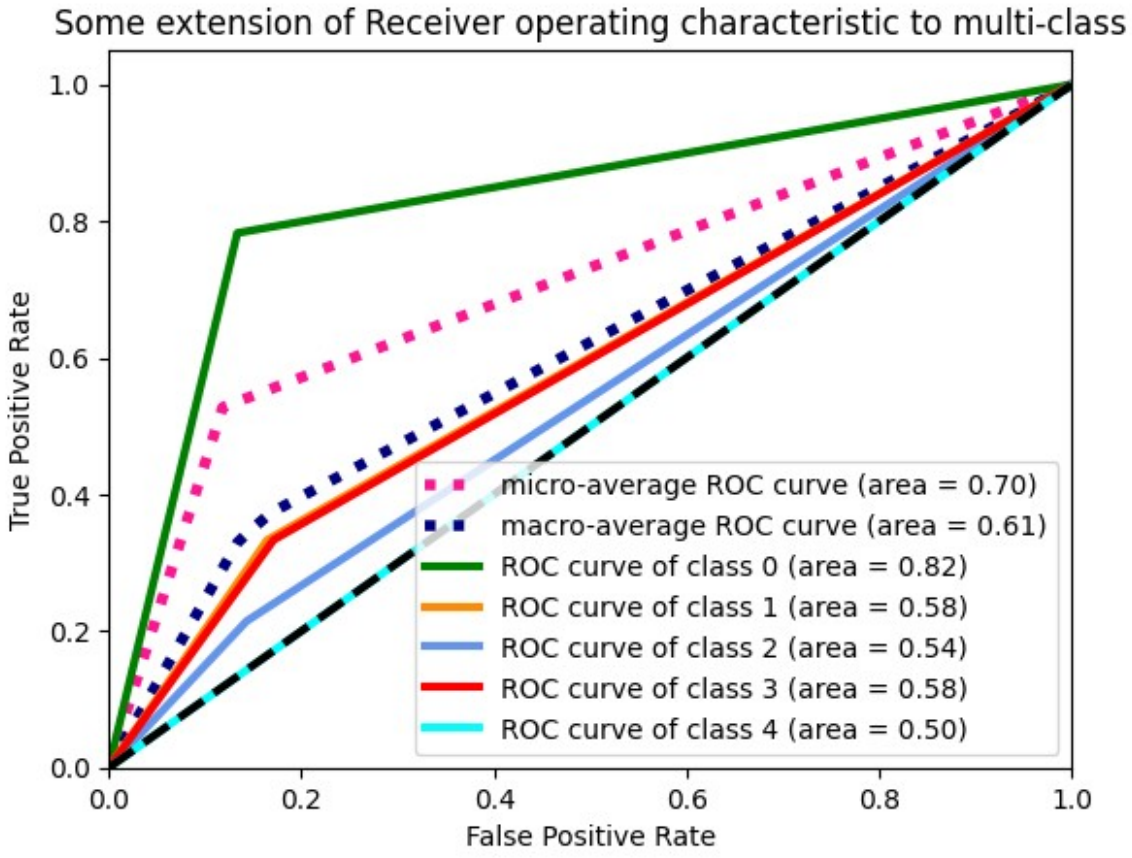
```
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to
multi-class')
plt.legend(loc="lower right")
plt.show()
```




```
#ROC curve for (ANN+SVM)


from sklearn.preprocessing import label_binarize
y_test_h = label_binarize(y_test, classes=[0, 1, 2, 3, 4])


y_score1 = label_binarize(out, classes=[0, 1, 2, 3, 4])
from sklearn.metrics import roc_curve, auc
```

```
from scipy.interpolate import interp1d as interp
from itertools import cycle
n_classes=5
lw=3
# 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_h[:, i], y_score1[:, 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_h.ravel(),
y_score1.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])


# First aggregate all false positive rates
all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

# Then interpolate all ROC curves at this points
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    f = interp(fpr[i], tpr[i])
    mean_tpr += f(all_fpr)

# Finally average it and compute AUC
mean_tpr /= n_classes

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves
plt.figure()
plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

colors = cycle(['green', 'darkorange', 'cornflowerblue','red','aqua'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=lw,
```

```
             label='ROC curve of class {0} (area = {1:0.2f})'
             ''.format(i, roc_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Some extension of Receiver operating characteristic to
multi-class')
plt.legend(loc="lower right")
plt.show()
```




```
# prompt: Based on the above trained model I want to calculate the
multiclass disease severity index score for each class and disease
also show that the disease is mild,severe and other options or levels
of severity also can you show which class is which disease?

import numpy as np
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize
```

```
# Assuming 'out' contains the final predictions from the weighted
fusion
# and 'y_test' is the true labels.

# Replace with your actual predictions and true labels
# Example (assuming 'out' and 'y_test' are already defined from
previous code)
# out = ... # your predictions
# y_test = ... # your true labels


def calculate_severity_index(predictions, true_labels):
    """Calculates a multiclass disease severity index."""

    # Placeholder for severity levels (replace with actual severity
mapping)
    severity_levels = {0: "Mild", 1: "Moderate", 2: "Severe", 3:
"Critical", 4: "Severe"}

    n_classes = len(severity_levels)  # Number of classes
    class_indices = list(severity_levels.keys()) #Class indices
    y_test_bin = label_binarize(true_labels, classes=class_indices)

    # Calculate ROC AUC scores for each class.
    roc_auc_scores = []
    for i in range(n_classes):
        roc_auc = roc_auc_score(y_test_bin[:, i], predictions[:, i])
        roc_auc_scores.append(roc_auc)


    # Calculate overall disease severity index
    severity_index = np.mean(roc_auc_scores)

    # Display disease severity for each class
    print("Multiclass Disease Severity Index:", severity_index)

    for i, class_index in enumerate(class_indices):
      print(f"Class {class_index} ({severity_levels[class_index]}):
ROC AUC = {roc_auc_scores[i]:.3f}")



# Example usage (replace with your actual predictions)
calculate_severity_index(mixed, y_test)

# Assuming 'out' is the predicted class labels and 'data' is your
DataFrame
# that contains a mapping from class labels to disease names.
# Create a dictionary for Class to Disease Name mapping.
# This mapping should be based on your actual data. Example:
```

```
class_to_disease = {0: "Disease A", 1: "Disease B", 2: "Disease C", 3:
"Disease D", 4: "Disease E"}

#Print mapping
for class_label, disease_name in class_to_disease.items():
  print(f"Class {class_label} corresponds to {disease_name}.")
```

```
Multiclass Disease Severity Index: 0.7658332578241531
Class 0 (Mild): ROC AUC = 0.900
Class 1 (Moderate): ROC AUC = 0.650
Class 2 (Severe): ROC AUC = 0.681
Class 3 (Critical): ROC AUC = 0.768
Class 4 (Severe): ROC AUC = 0.830
Class 0 corresponds to Disease A.
Class 1 corresponds to Disease B.
Class 2 corresponds to Disease C.
Class 3 corresponds to Disease D.
Class 4 corresponds to Disease E.
```

Experiment 6: Applying Time Series Analysis for Patient Vital Sign Forecasting

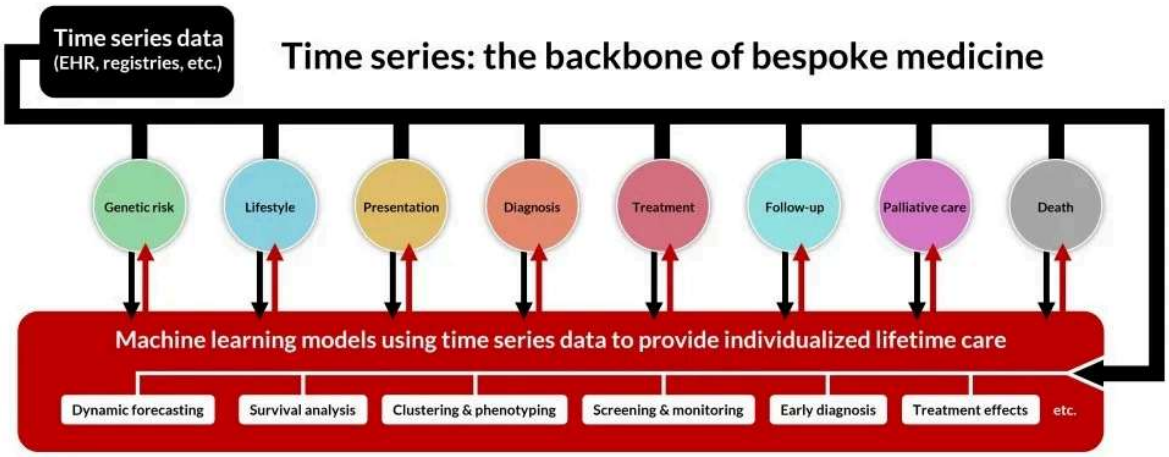
---

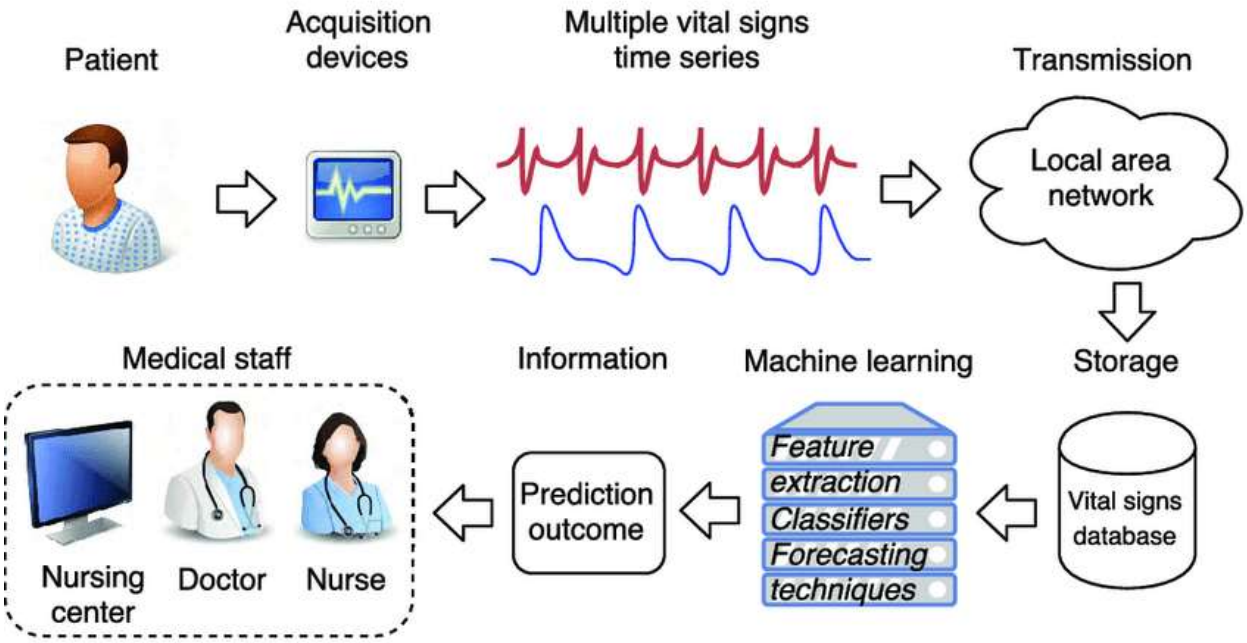
Aim:
To apply time series analysis techniques to forecast patient vital signs (such as heart rate, blood pressure, and temperature) based on historical data. This experiment focuses on developing models that can predict future vital signs, enabling proactive patient monitoring and timely interventions.

Theory:
Time series analysis involves statistical techniques used to analyze time-ordered data points to identify trends, patterns, and seasonal variations over time. In the context of patient vital sign forecasting, the objective is to predict future values based on past observations.

Key concepts and steps in time series analysis include:

1. **Data Collection:** Gather historical vital sign data from patient records, ensuring that the data is time stamped and organized chronologically.
2. **Data Preprocessing:** Clean the dataset by handling missing values, outliers, and noise. Resample or aggregate the data if necessary, and split it into training and testing datasets.
3. **Exploratory Data Analysis (EDA):** Perform EDA to visualize trends, seasonality, and autocorrelation in the vital sign data. Tools such as line plots and correlograms can help in understanding the data patterns.
4. **Model Selection:** Choose an appropriate time series forecasting model, such as ARIMA (AutoRegressive Integrated Moving Average), Exponential Smoothing, or machine learning approaches like Long Short-Term Memory (LSTM) networks.
5. **Model Training:** Train the selected model using the training dataset, allowing it to learn from historical vital sign data.
6. **Model Evaluation:** Assess the model's forecasting accuracy using metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) on the testing set.
7. **Result Interpretation:** Analyze the forecasts generated by the model, comparing them against actual vital signs to understand the model's performance and areas for potential improvement.
8. **Deployment:** If validated, deploy the forecasting model in clinical settings to assist healthcare providers in monitoring patient vital signs and predicting potential health issues.


Time series data
(EHR, registries, etc.)
Time series: the backbone of bespoke medicine
Genetic risk
Lifestyle
Presentation
Diagnosis
Treatment
Follow-up
Palliative care
Death
Machine learning models using time series data to provide individualized lifetime care
Dynamic forecasting
Survival analysis
Clustering & phenotyping
Screening & monitoring
Early diagnosis
Treatment effects
etc.


Patient
Acquisition devices
Multiple vital signs time series
Transmission
Local area network
Storage
Vital signs database
Machine learning
Feature extraction
Classifiers
Forecasting techniques
Information
Prediction outcome
Medical staff
Nursing center
Doctor
Nurse

[1]:
```
# Step 1: Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
```

[3]:
```
# Step 2: Generate Sample Time Series Data for Patient Vital Signs
np.random.seed(42)
date_rng = pd.date_range(start='2023-01-01', end='2023-12-31', freq='D')
```

[4]:
```
# Generate respiratory rate data (breaths per minute)
respiratory_rate_data = np.random.normal(loc=16, scale=3, size=len(date_rng)) +␣
 ↪np.sin(np.linspace(0, 3 * np.pi, len(date_rng))) * 2

# Generate body temperature data (degrees Celsius)
body_temperature_data = np.random.normal(loc=37, scale=0.5, size=len(date_rng))␣
 ↪+ np.sin(np.linspace(0, 3 * np.pi, len(date_rng))) * 0.2

# Generate blood pressure data (systolic and diastolic)
systolic_data = np.random.normal(loc=120, scale=10, size=len(date_rng))
diastolic_data = np.random.normal(loc=80, scale=5, size=len(date_rng))

# Generate oxygen saturation level data (SpO2)
oxygen_saturation_data = np.random.normal(loc=98, scale=1, size=len(date_rng))

# Generate heart rate variability (HRV)
hrv_data = np.random.normal(loc=50, scale=10, size=len(date_rng))

# Generate activity level (randomly selected)
activity_levels = np.random.choice(['Sedentary', 'Lightly Active', 'Active'],␣
 ↪size=len(date_rng))

# Create DataFrame
data = pd.DataFrame({
```

```
    'date': date_rng,
    'respiratory_rate': respiratory_rate_data,
    'body_temperature': body_temperature_data,
    'systolic_bp': systolic_data,
    'diastolic_bp': diastolic_data,
    'oxygen_saturation': oxygen_saturation_data,
    'hrv': hrv_data,
    'activity_level': activity_levels
})
```

[5]:
```
# Set the date as the index
data.set_index('date', inplace=True)
```

[6]:
```
# Display the generated dataset
print(data.head())
```

```
            respiratory_rate  body_temperature  systolic_bp  diastolic_bp  \
date
2023-01-01         17.490142         36.799390   121.958453     79.596417
2023-01-02         15.636986         37.117224   110.216272     80.393176
2023-01-03         18.046588         37.016648   124.082528     70.008997
2023-01-04         20.724287         37.064358   102.974164     84.581638
2023-01-05         15.504308         36.634172   130.291556     81.732442

            oxygen_saturation        hrv  activity_level
date
2023-01-01          97.861544  54.419406  Lightly Active
2023-01-02          96.775702  39.096009       Sedentary
2023-01-03          97.790977  64.109324          Active
2023-01-04          97.149480  49.014119          Active
2023-01-05          97.419477  50.188496  Lightly Active
```

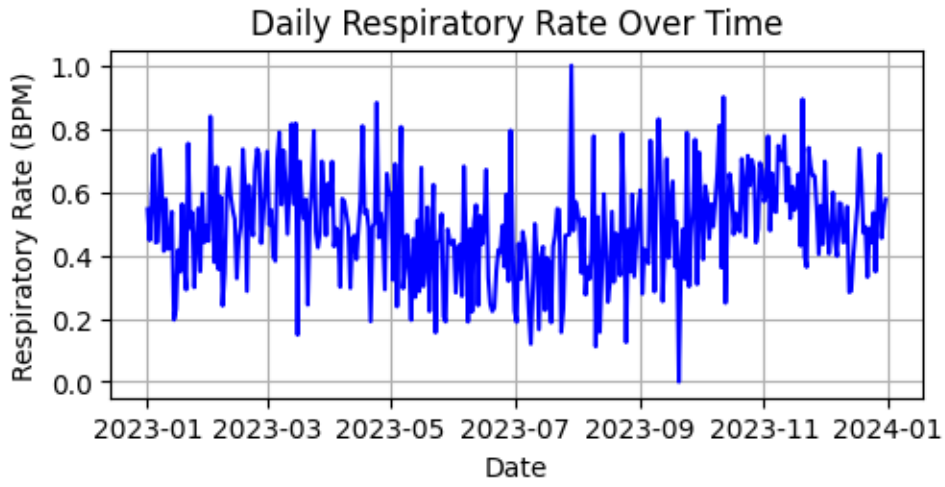
[7]:
```
# Step 3: Data Exploration and Visualization
plt.figure(figsize=(18, 10))
```

[7]:
```
<Figure size 1800x1000 with 0 Axes>
```

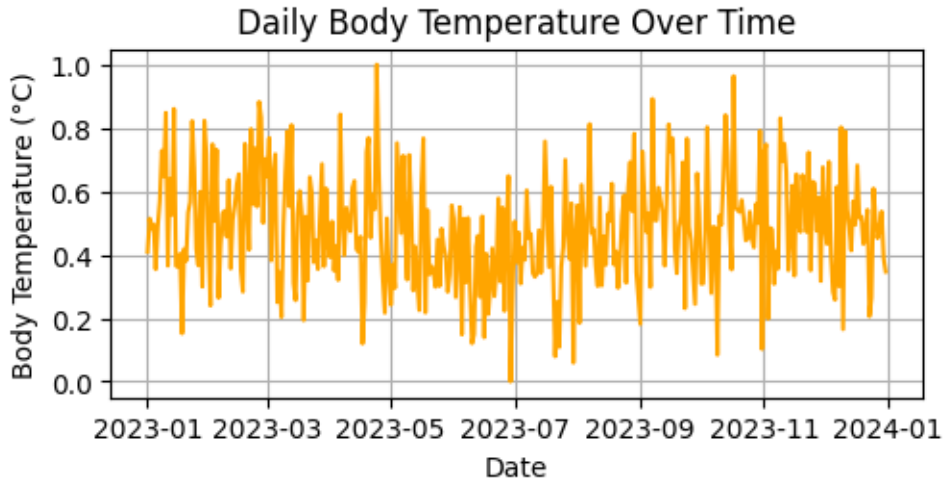
```
<Figure size 1800x1000 with 0 Axes>
```

[30]:
```
# Plot respiratory rate
# Set the figure size
plt.figure(figsize=(12, 8))
plt.subplot(3, 2, 1)
plt.plot(data.index, data['respiratory_rate'], color='blue')
plt.title('Daily Respiratory Rate Over Time')
plt.xlabel('Date')
plt.ylabel('Respiratory Rate (BPM)')
```

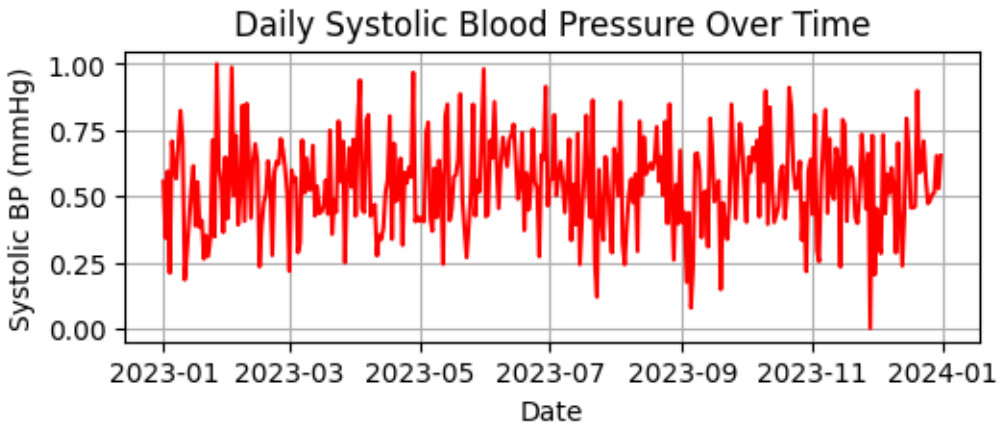
```
plt.grid()
```



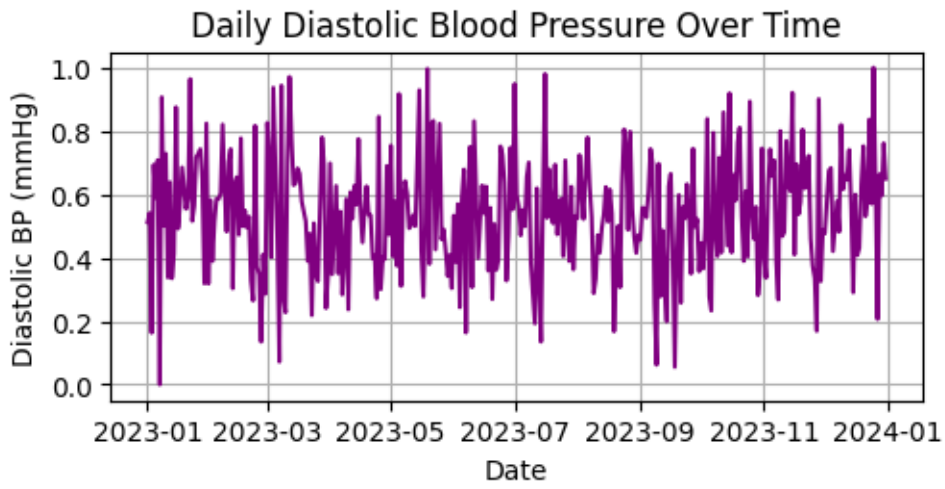
[29]:
```
# Plot body temperature
# Set the figure size
plt.figure(figsize=(12, 8))
plt.subplot(3, 2, 2)
plt.plot(data.index, data['body_temperature'], color='orange')
plt.title('Daily Body Temperature Over Time')
plt.xlabel('Date')
plt.ylabel('Body Temperature (°C)')
plt.grid()
```



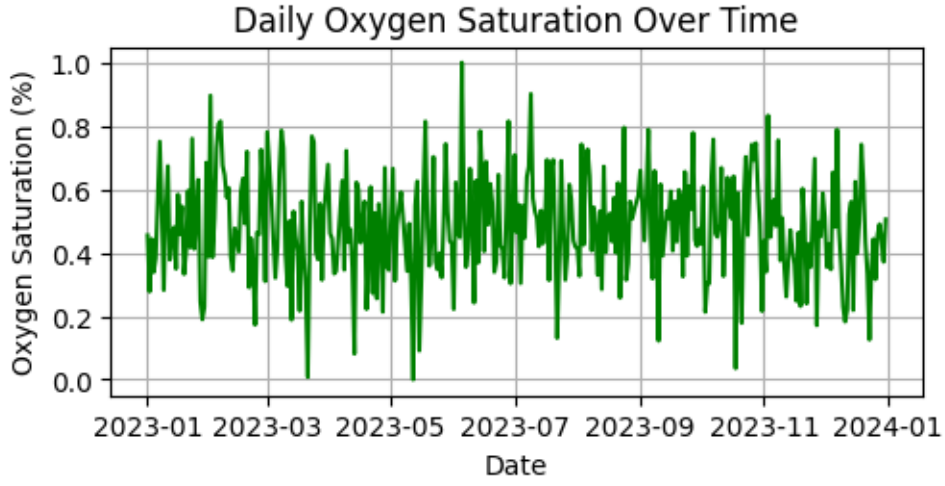
[26]:
```
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 6))  # Adjust the size as needed

# Plot systolic blood pressure
plt.subplot(3, 2, 3)
plt.plot(data.index, data['systolic_bp'], color='red')
plt.title('Daily Systolic Blood Pressure Over Time')
plt.xlabel('Date')
plt.ylabel('Systolic BP (mmHg)')
plt.grid()

# Adjust the layout to prevent overlap
plt.tight_layout()

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



[31]:
```
# Plot diastolic blood pressure
# Set the figure size
plt.figure(figsize=(12, 8))
plt.subplot(3, 2, 4)
plt.plot(data.index, data['diastolic_bp'], color='purple')
plt.title('Daily Diastolic Blood Pressure Over Time')
plt.xlabel('Date')
plt.ylabel('Diastolic BP (mmHg)')
plt.grid()
```

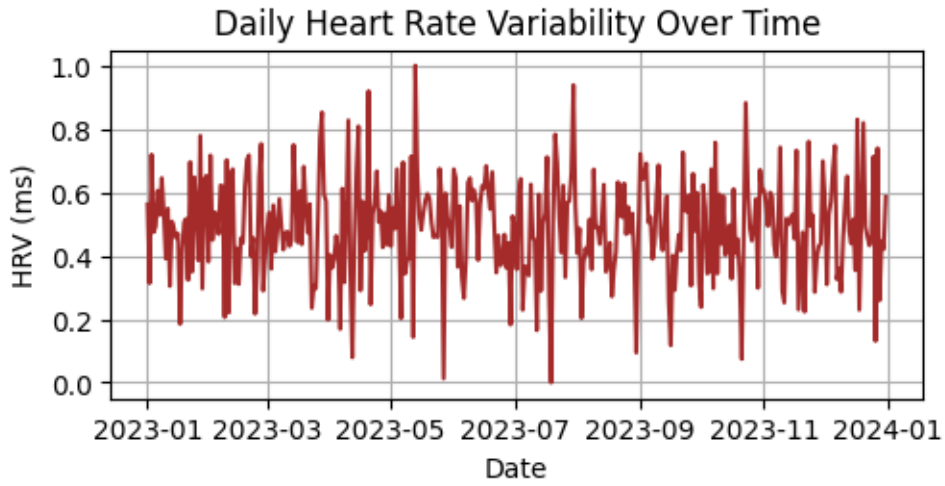


[32]:
```
# Plot oxygen saturation
# Set the figure size
plt.figure(figsize=(12, 8))
plt.subplot(3, 2, 5)
plt.plot(data.index, data['oxygen_saturation'], color='green')
plt.title('Daily Oxygen Saturation Over Time')
plt.xlabel('Date')
plt.ylabel('Oxygen Saturation (%)')
plt.grid()
```



[33]:
```
# Plot HRV
# Set the figure size
plt.figure(figsize=(12, 8))
```

```
plt.subplot(3, 2, 6)
plt.plot(data.index, data['hrv'], color='brown')
plt.title('Daily Heart Rate Variability Over Time')
plt.xlabel('Date')
plt.ylabel('HRV (ms)')
plt.grid()
```



[14]:
```
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

[15]:
```
# Step 4: Data Preprocessing
# Convert categorical activity level into numerical values
data = pd.get_dummies(data, columns=['activity_level'], drop_first=True)
```

[16]:
```
# Normalize continuous features
scaler = MinMaxScaler(feature_range=(0, 1))
data[['respiratory_rate', 'body_temperature', 'systolic_bp', 'diastolic_bp',
      'oxygen_saturation', 'hrv']] = scaler.
 ↪fit_transform(data[['respiratory_rate',
                                                          ␣
 ↪'body_temperature',
                                                           'systolic_bp',
                                                           'diastolic_bp',
                                                          ␣
 ↪'oxygen_saturation',
                                                           'hrv']])
```

[17]:
```
# Step 5: Create Sequences for LSTM Input
def create_sequences(data, seq_length=30):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data.iloc[i:i + seq_length].values)
        y.append(data.iloc[i + seq_length, 0])  # respiratory_rate as target
    return np.array(X), np.array(y)

seq_length = 30  # Sequence length of 30 days
X, y = create_sequences(data, seq_length)
```

[18]:
```
# Step 6: Split Data into Training and Testing Sets
split = int(len(X) * 0.8)
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]
```

[19]:
```
# Step 7: Build and Train LSTM Model
model = Sequential([
    LSTM(50, activation='relu', input_shape=(seq_length, X.shape[2])),
    Dropout(0.2),
    Dense(1)
])
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
```

```
C:\Users\acer\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

[20]:
```
# Convert to float32
X_train = X_train.astype(np.float32)
X_test = X_test.astype(np.float32)
y_train = y_train.astype(np.float32)
y_test = y_test.astype(np.float32)
```

[21]:
```
# Fit the model
history = model.fit(X_train, y_train, epochs=20, batch_size=16,␣
 ↪validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/20
17/17             2s 21ms/step -
loss: 0.1184 - mae: 0.2905 - val_loss: 0.0192 - val_mae: 0.1164
Epoch 2/20
17/17             0s 10ms/step -
loss: 0.0476 - mae: 0.1678 - val_loss: 0.0331 - val_mae: 0.1439
Epoch 3/20
```

```
17/17            0s 13ms/step -
loss: 0.0392 - mae: 0.1542 - val_loss: 0.0225 - val_mae: 0.1201
Epoch 4/20
17/17            0s 11ms/step -
loss: 0.0280 - mae: 0.1305 - val_loss: 0.0217 - val_mae: 0.1182
Epoch 5/20
17/17            0s 11ms/step -
loss: 0.0384 - mae: 0.1560 - val_loss: 0.0266 - val_mae: 0.1293
Epoch 6/20
17/17            0s 11ms/step -
loss: 0.0321 - mae: 0.1436 - val_loss: 0.0184 - val_mae: 0.1110
Epoch 7/20
17/17            0s 12ms/step -
loss: 0.0338 - mae: 0.1463 - val_loss: 0.0233 - val_mae: 0.1218
Epoch 8/20
17/17            0s 10ms/step -
loss: 0.0279 - mae: 0.1338 - val_loss: 0.0187 - val_mae: 0.1113
Epoch 9/20
17/17            0s 10ms/step -
loss: 0.0368 - mae: 0.1513 - val_loss: 0.0201 - val_mae: 0.1142
Epoch 10/20
17/17            0s 9ms/step - loss:
0.0334 - mae: 0.1458 - val_loss: 0.0223 - val_mae: 0.1193
Epoch 11/20
17/17            0s 9ms/step - loss:
0.0339 - mae: 0.1482 - val_loss: 0.0191 - val_mae: 0.1119
Epoch 12/20
17/17            0s 9ms/step - loss:
0.0335 - mae: 0.1410 - val_loss: 0.0233 - val_mae: 0.1216
Epoch 13/20
17/17            0s 9ms/step - loss:
0.0330 - mae: 0.1457 - val_loss: 0.0176 - val_mae: 0.1086
Epoch 14/20
17/17            0s 9ms/step - loss:
0.0299 - mae: 0.1391 - val_loss: 0.0222 - val_mae: 0.1192
Epoch 15/20
17/17            0s 10ms/step -
loss: 0.0349 - mae: 0.1466 - val_loss: 0.0238 - val_mae: 0.1227
Epoch 16/20
17/17            0s 9ms/step - loss:
0.0289 - mae: 0.1351 - val_loss: 0.0164 - val_mae: 0.1071
Epoch 17/20
17/17            0s 9ms/step - loss:
0.0353 - mae: 0.1480 - val_loss: 0.0249 - val_mae: 0.1254
Epoch 18/20
17/17            0s 9ms/step - loss:
0.0349 - mae: 0.1476 - val_loss: 0.0211 - val_mae: 0.1169
Epoch 19/20
```

```
17/17             0s 10ms/step -
loss: 0.0285 - mae: 0.1337 - val_loss: 0.0178 - val_mae: 0.1093
Epoch 20/20
17/17             0s 10ms/step -
loss: 0.0309 - mae: 0.1347 - val_loss: 0.0189 - val_mae: 0.1117
```

[22]:
```
# Step 8: Make Predictions
predictions = model.predict(X_test)
predictions_rescaled = scaler.inverse_transform(np.concatenate([predictions, np.
 ↪zeros((predictions.shape[0], 5))], axis=1))[:,0]
```

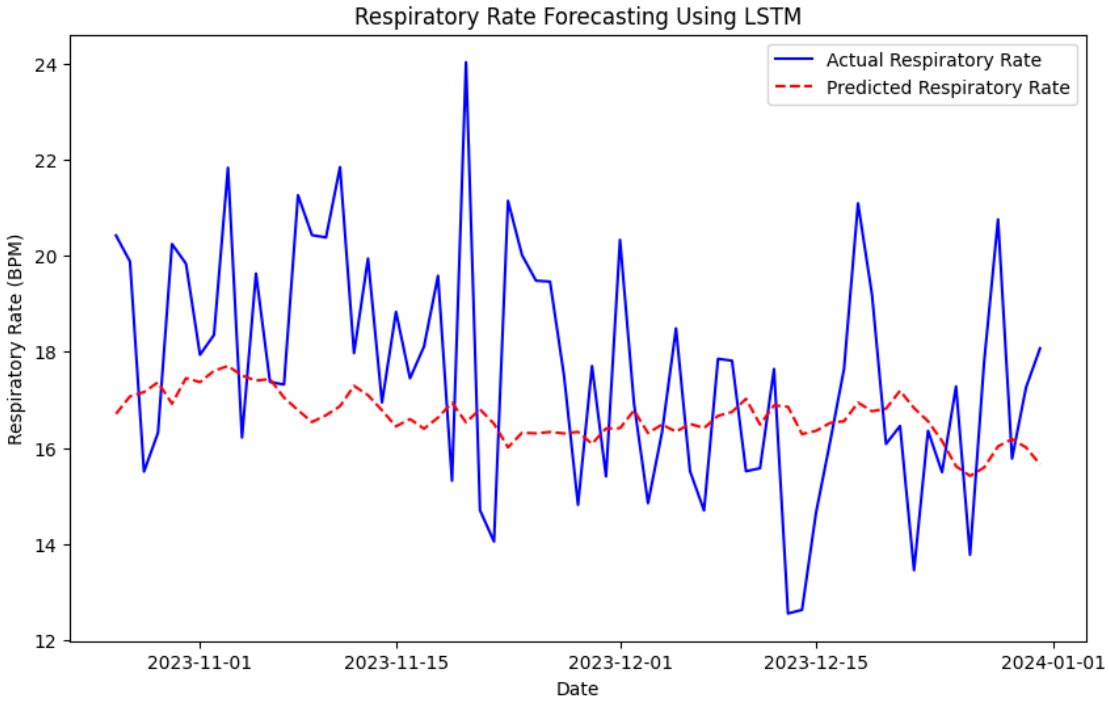
```
3/3             0s 85ms/step
```

[23]:
```
# Reshape y_test for inverse transform
y_test_rescaled = scaler.inverse_transform(np.concatenate([y_test.reshape(-1,␣
 ↪1), np.zeros((y_test.shape[0], 5))], axis=1))[:,0]
```

[24]:
```
# Step 9: Evaluate the Model
mae = np.mean(np.abs(predictions_rescaled - y_test_rescaled))
print(f"Mean Absolute Error (MAE) of the forecast: {mae:.2f} BPM")
```

```
Mean Absolute Error (MAE) of the forecast: 2.10 BPM
```

[25]:
```
# Step 10: Visualize Results
plt.figure(figsize=(10, 6))
plt.plot(data.index[-len(y_test):], y_test_rescaled, color='blue',␣
 ↪label='Actual Respiratory Rate')
plt.plot(data.index[-len(predictions):], predictions_rescaled, color='red',␣
 ↪linestyle='--', label='Predicted Respiratory Rate')
plt.title('Respiratory Rate Forecasting Using LSTM')
plt.xlabel('Date')
plt.ylabel('Respiratory Rate (BPM)')
plt.legend()
plt.show()
```



[35]:
```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assuming you have these from your LSTM model's predictions
true_values = y_test_rescaled  # Actual respiratory rates (true values)
predicted_values = predictions_rescaled  # Model's predictions

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(true_values, predicted_values)
print(f'Mean Absolute Error (MAE): {mae:.2f} BPM')

# Calculate Root Mean Square Error (RMSE)
rmse = np.sqrt(mean_squared_error(true_values, predicted_values))
print(f'Root Mean Square Error (RMSE): {rmse:.2f} BPM')

# Calculate R-squared (R²)
r_squared = r2_score(true_values, predicted_values)
print(f'R-squared (R²): {r_squared:.2f}')

# Create a DataFrame to visualize the results
results = pd.DataFrame({
    'True Values': true_values,
    'Predicted Values': predicted_values,
```

```
    'Error': predicted_values - true_values
})

print("\nEvaluation Results:")
print(results)
```

```
Mean Absolute Error (MAE): 2.10 BPM
Root Mean Square Error (RMSE): 2.59 BPM
R-squared (R²): -0.15

Evaluation Results:
    True Values  Predicted Values     Error
0     20.419547         16.707484 -3.712063
1     19.876311         17.069886 -2.806425
2     15.505571         17.161654  1.656083
3     16.315808         17.363318  1.047510
4     20.240689         16.919837 -3.320852
..          …               …       …
62    17.764807         15.589082 -2.175725
63    20.753414         16.031177 -4.722237
64    15.777243         16.181816  0.404573
65    17.256914         16.009695 -1.247219
66    18.070432         15.656148 -2.414284

[67 rows x 3 columns]
```

[ ]:

Experiment 7: Developing a Convolutional Neural Network (CNN) for Medical Image Classification

---

Aim:
To develop a Convolutional Neural Network (CNN) model for classifying medical images into different categories, such as identifying the presence of specific diseases or conditions in imaging data (e.g., X-rays, MRIs, or CT scans). This experiment focuses on leveraging deep learning techniques to enhance diagnostic accuracy and efficiency in medical imaging.

Theory:
Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for processing grid-like data, such as images. They excel in image classification tasks due to their ability to automatically learn hierarchical feature representations from raw pixel data. The key components of a CNN include:

1. **Convolutional Layers:** These layers apply convolution operations to the input images, utilizing filters to extract features such as edges, textures, and shapes. Each filter creates a feature map that highlights specific patterns.
2. **Activation Functions:** Non-linear activation functions (commonly ReLU) are applied after convolution operations to introduce non-linearity, allowing the model to learn complex patterns.
3. **Pooling Layers:** Pooling layers (e.g., max pooling) downsample the feature maps, reducing their spatial dimensions while retaining important features. This helps in minimizing computational complexity and preventing overfitting.
4. **Fully Connected Layers:** After several convolutional and pooling layers, the output is flattened and passed through one or more fully connected layers, leading to the final classification output.
5. **Loss Function:** The loss function (e.g., categorical cross-entropy) measures the model's performance by comparing the predicted class probabilities with the actual labels.
6. **Optimization Algorithm:** An optimization algorithm (e.g., Adam or SGD) is used to minimize the loss function during training by updating the network's weights.

Dataset Description

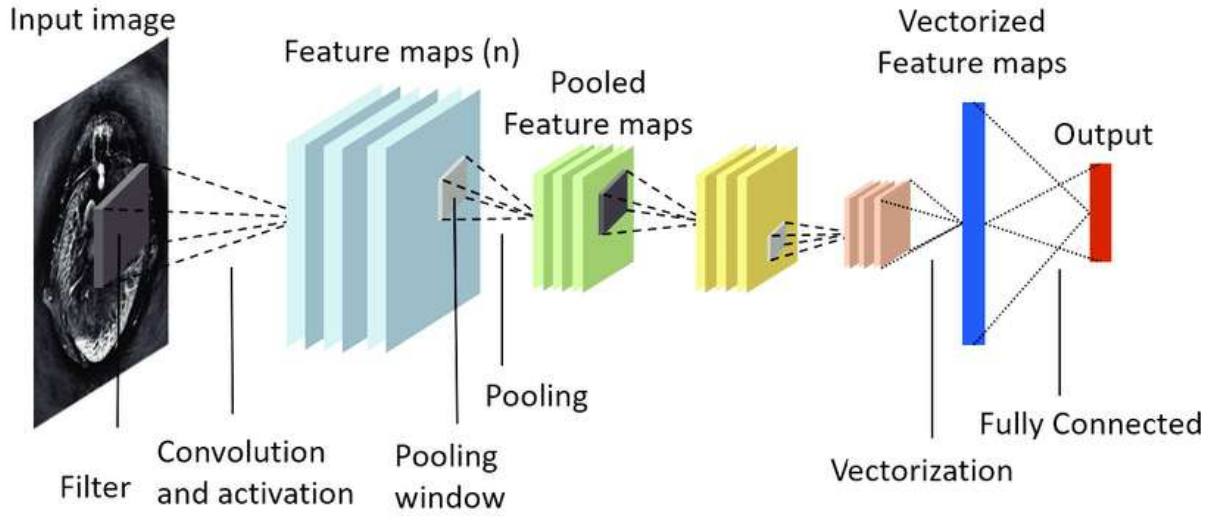
This dataset contains 5,856 validated Chest X-Ray images. The images are split into a training set and a testing set of independent patients. Images are labeled as (disease:NORMAL/BACTERIA/VIRUS)-(randomized patient ID)-(image number of a patient).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

Steps to Implement CNN for Medical Image Classification:

1. **Data Collection:** Gather a dataset of medical images with corresponding labels indicating the conditions or diseases present.
2. **Data Preprocessing:** Preprocess the images by resizing, normalizing, and augmenting them (e.g., flipping, rotating) to increase dataset variability and reduce overfitting.
3. **Model Architecture Design:** Design a CNN architecture suitable for the image classification task, specifying the number of convolutional layers, filter sizes, pooling operations, and fully connected layers.
4. **Model Compilation:** Compile the CNN model by selecting the loss function, optimizer, and evaluation metrics.
5. **Model Training:** Train the model using the training dataset, monitoring performance on a validation set to adjust hyperparameters as needed.
6. **Model Evaluation:** Evaluate the trained model on a separate testing dataset, calculating performance metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
7. **Result Interpretation:** Analyze the results to understand the model's performance in classifying medical images and identify areas for improvement or further research.
8. **Deployment:** If the model performs satisfactorily, deploy it in clinical settings to assist healthcare professionals in making diagnostic decisions based on medical images.



Importing Packages and Dataset

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

pd.options.display.max_colwidth = 100

import random
import os

from numpy.random import seed
seed(42)

random.seed(42)
os.environ['PYTHONHASHSEED'] = str(42)
os.environ['TF_DETERMINISTIC_OPS'] = '1'

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import glob
import cv2

from tensorflow.random import set_seed
set_seed(42)

import warnings
warnings.filterwarnings('ignore')

IMG_SIZE = 224
BATCH = 32
SEED = 42

#main_path = "../input/chest-xray-pneumonia/chest_xray/"
main_path = "Downloads/archive/chest_xray"
```

```
train_path = os.path.join(main_path,"train")
test_path=os.path.join(main_path,"test")

train_normal = glob.glob(train_path+"/NORMAL/*.jpeg")
train_pneumonia = glob.glob(train_path+"/PNEUMONIA/*.jpeg")

test_normal = glob.glob(test_path+"/NORMAL/*.jpeg")
test_pneumonia = glob.glob(test_path+"/PNEUMONIA/*.jpeg")

train_list = [x for x in train_normal]
train_list.extend([x for x in train_pneumonia])

df_train = pd.DataFrame(np.concatenate([['Normal']*len(train_normal) ,
['Pneumonia']*len(train_pneumonia)]), columns = ['class'])
df_train['image'] = [x for x in train_list]

test_list = [x for x in test_normal]
test_list.extend([x for x in test_pneumonia])

df_test = pd.DataFrame(np.concatenate([['Normal']*len(test_normal) ,
['Pneumonia']*len(test_pneumonia)]), columns = ['class'])
df_test['image'] = [x for x in test_list]

df_train
```

```
          class  \
0        Normal
1        Normal
2        Normal
3        Normal
4        Normal
...         ...
5211  Pneumonia
5212  Pneumonia
5213  Pneumonia
5214  Pneumonia
5215  Pneumonia


image
0               Downloads/archive/chest_xray\train/NORMAL\IM-0115-
0001.jpeg
1               Downloads/archive/chest_xray\train/NORMAL\IM-0117-
0001.jpeg
2               Downloads/archive/chest_xray\train/NORMAL\IM-0119-
0001.jpeg
3               Downloads/archive/chest_xray\train/NORMAL\IM-0122-
0001.jpeg
4               Downloads/archive/chest_xray\train/NORMAL\IM-0125-
0001.jpeg
...
```

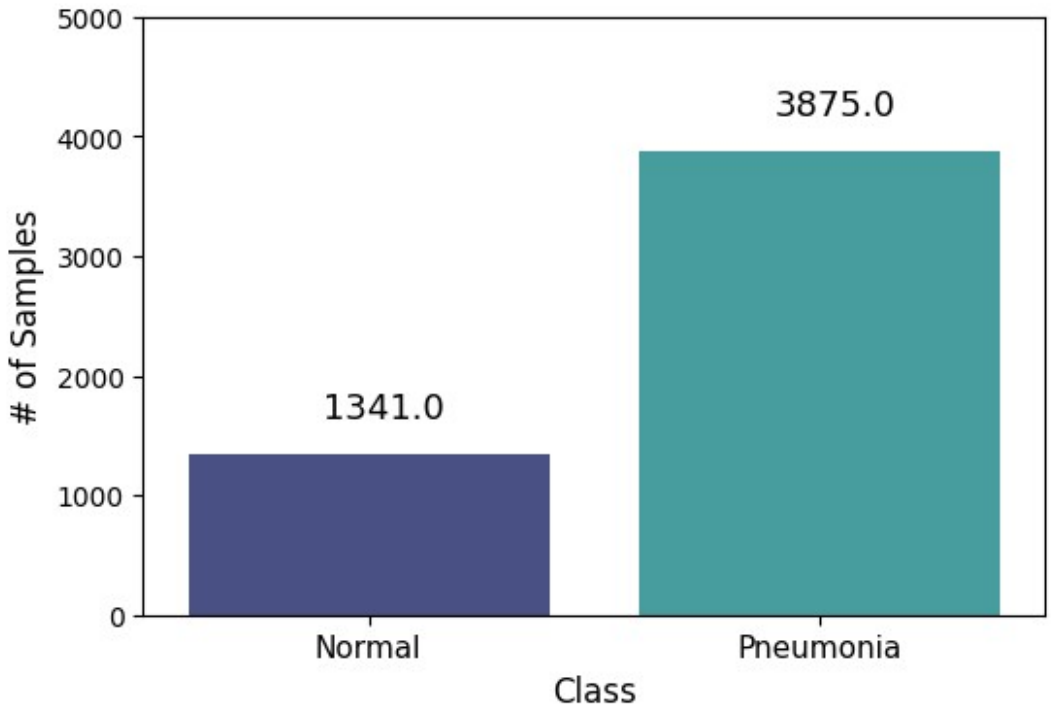
```
...
5211   Downloads/archive/chest_xray\train/PNEUMONIA\
person99_virus_183.jpeg
5212  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_38.jpeg
5213  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_39.jpeg
5214  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_40.jpeg
5215  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_41.jpeg

[5216 rows x 2 columns]

df_test

          class  \
0       Normal
1       Normal
2       Normal
3       Normal
4       Normal
..         ...
619  Pneumonia
620  Pneumonia
621  Pneumonia
622  Pneumonia
623  Pneumonia


image
0                Downloads/archive/chest_xray\test/NORMAL\IM-0001-
0001.jpeg
1                Downloads/archive/chest_xray\test/NORMAL\IM-0003-
0001.jpeg
2                Downloads/archive/chest_xray\test/NORMAL\IM-0005-
0001.jpeg
3                Downloads/archive/chest_xray\test/NORMAL\IM-0006-
0001.jpeg
4                Downloads/archive/chest_xray\test/NORMAL\IM-0007-
0001.jpeg
..
...
619  Downloads/archive/chest_xray\test/PNEUMONIA\
person96_bacteria_465.jpeg
620  Downloads/archive/chest_xray\test/PNEUMONIA\
person96_bacteria_466.jpeg
621  Downloads/archive/chest_xray\test/PNEUMONIA\
person97_bacteria_468.jpeg
622  Downloads/archive/chest_xray\test/PNEUMONIA\
```

```
person99_bacteria_473.jpeg
623  Downloads/archive/chest_xray\test/PNEUMONIA\
person99_bacteria_474.jpeg

[624 rows x 2 columns]
```

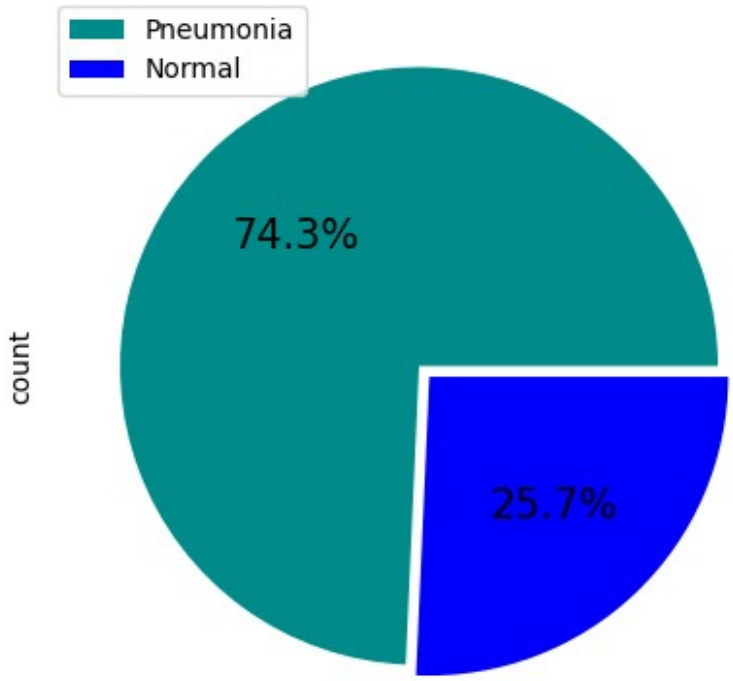
Exploring the Data

Let's check the target distribution on each set

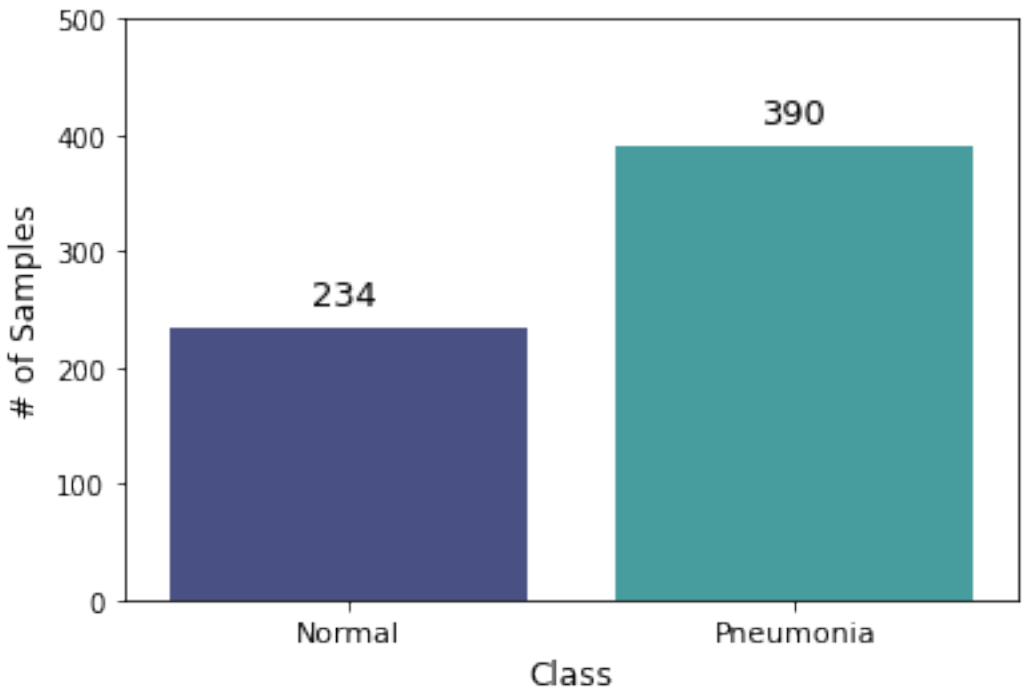
```
plt.figure(figsize=(6,4))

ax = sns.countplot(x='class', data=df_train, palette="mako")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,5000)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.30, p.get_height()
+300), fontsize = 13)

plt.show()
```



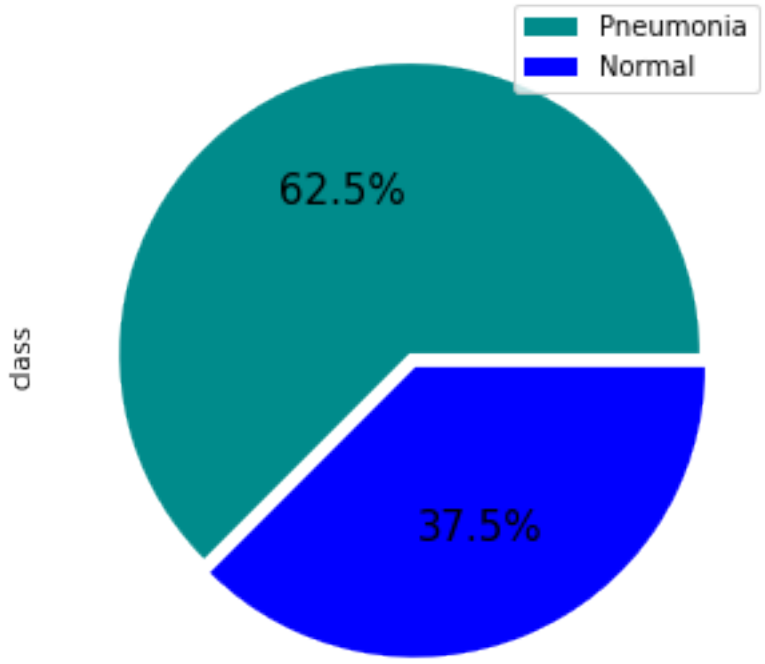
```
plt.figure(figsize=(7,5))

df_train['class'].value_counts().plot(kind='pie',labels = ['',''],
autopct='%1.1f%%', colors = ['darkcyan','blue'], explode = [0,0.05],
textprops = {"fontsize":15})

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```




```
plt.figure(figsize=(6,4))

ax = sns.countplot(x='class', data=df_test, palette="mako")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,500)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.32, p.get_height()+20),
fontsize = 13)

plt.show()
```




```
plt.figure(figsize=(7,5))

df_test['class'].value_counts().plot(kind='pie',labels = ['',''],
autopct='%1.1f%%', colors = ['darkcyan','blue'], explode = [0,0.05],
textprops = {"fontsize":15})

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```

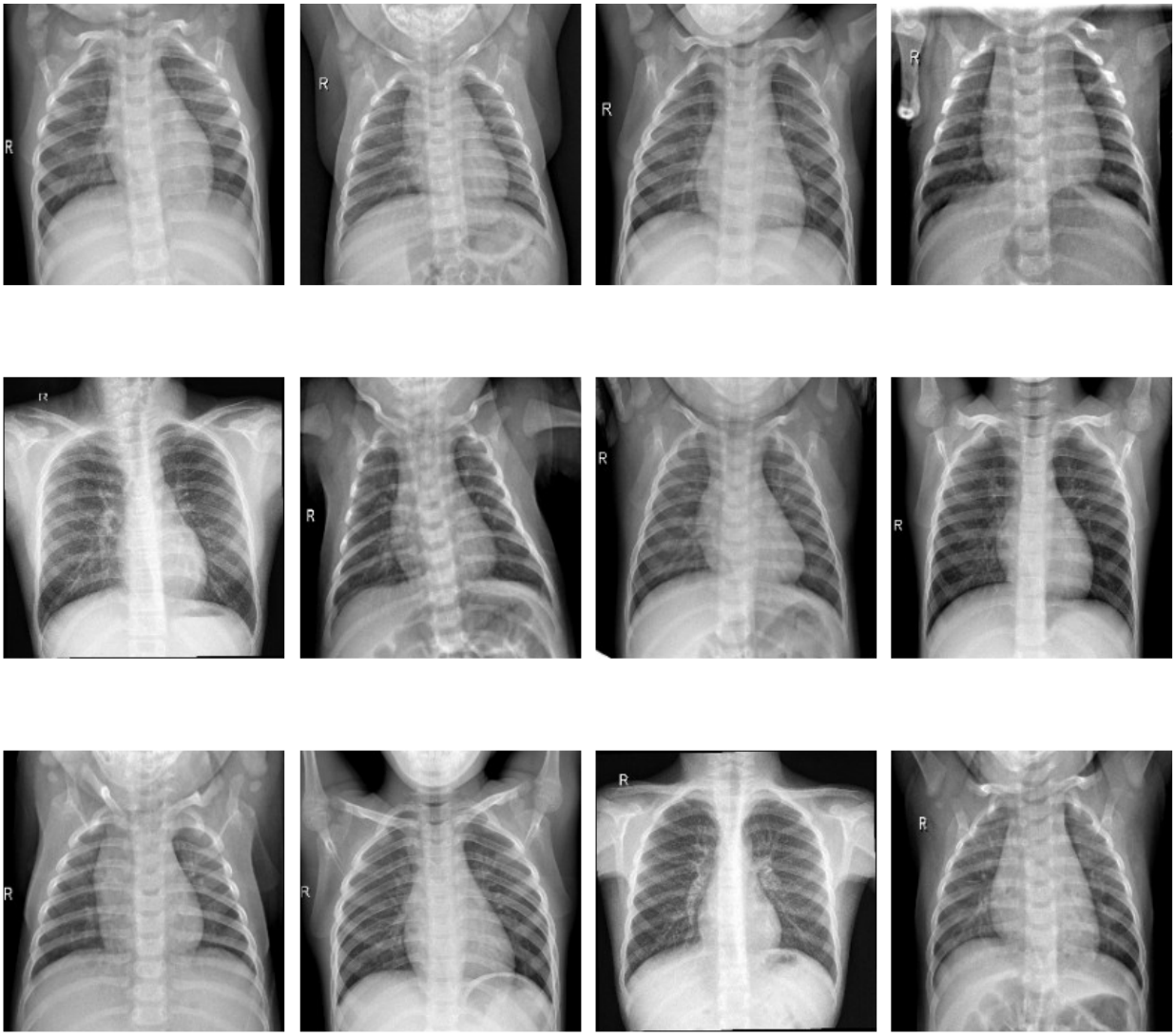


The distributions from these datasets are a little different from each other. Both are slightly imbalanced, having more samples from the positive class (Pneumonia), with the training set being a little more imbalanced.

Before we move on to the next section, we will take a look at a few examples from each dataset.

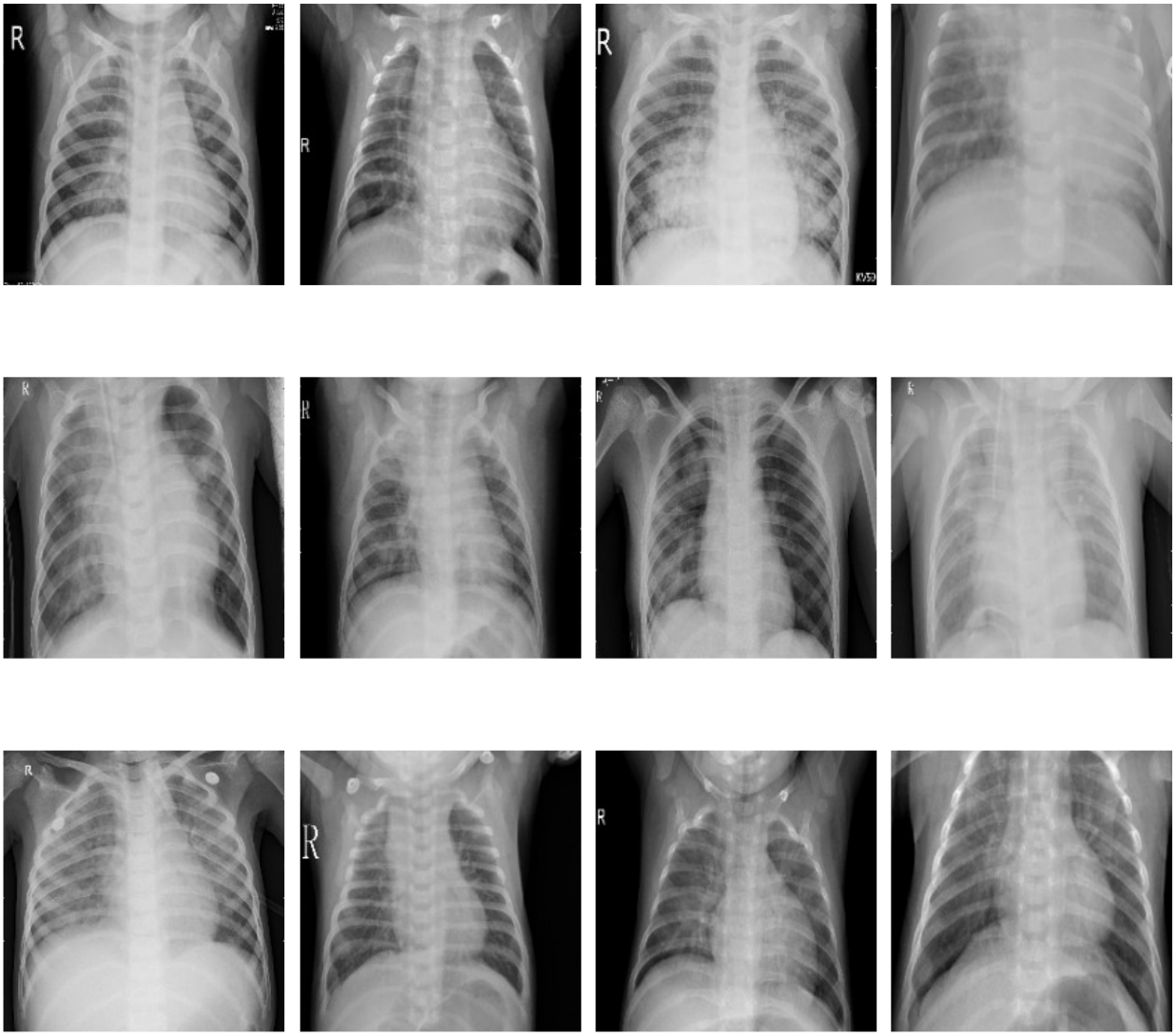
```
print('Train Set - Normal')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set - Normal
```



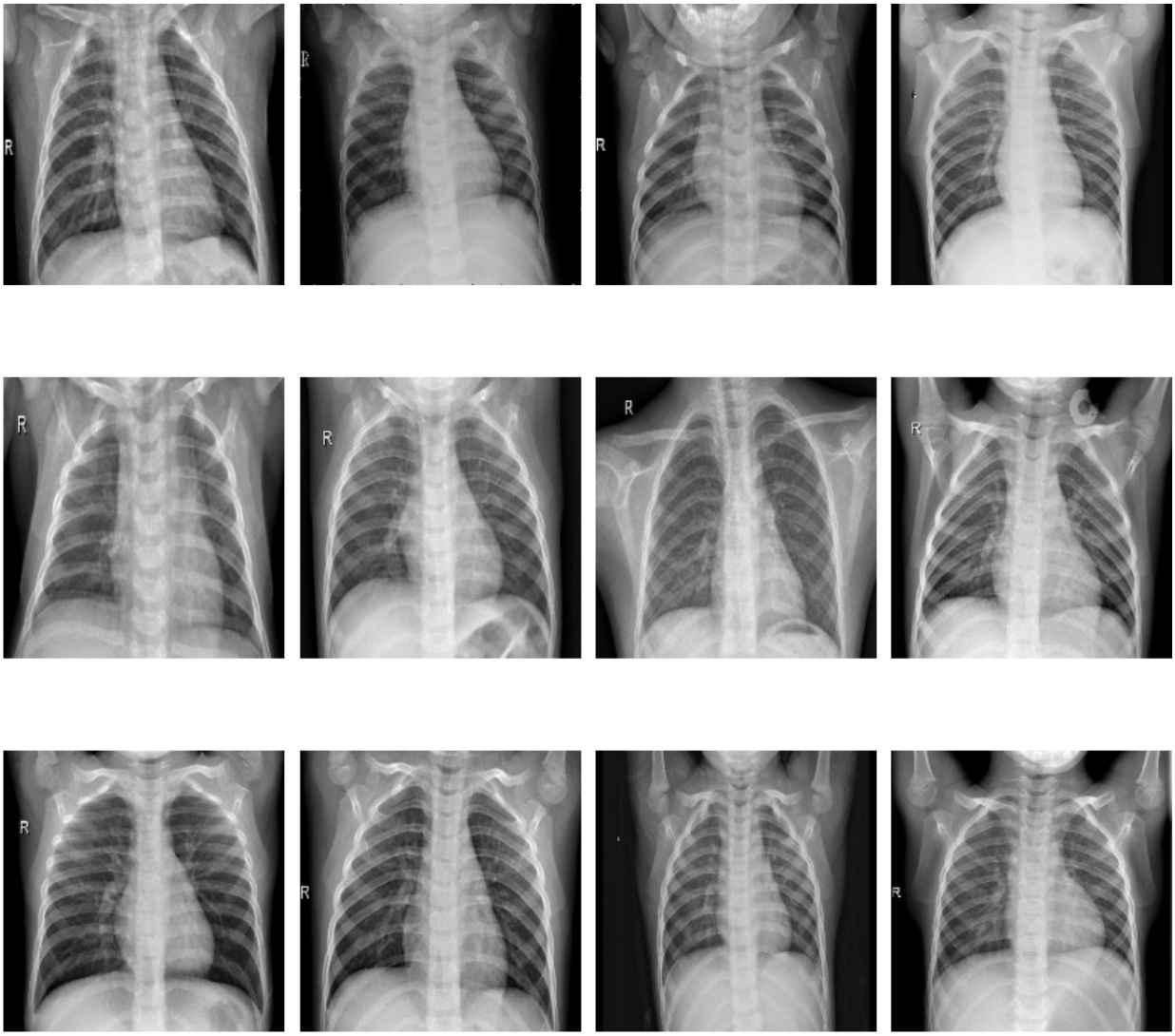
```
print('Train Set - Pneumonia')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set - Pneumonia
```



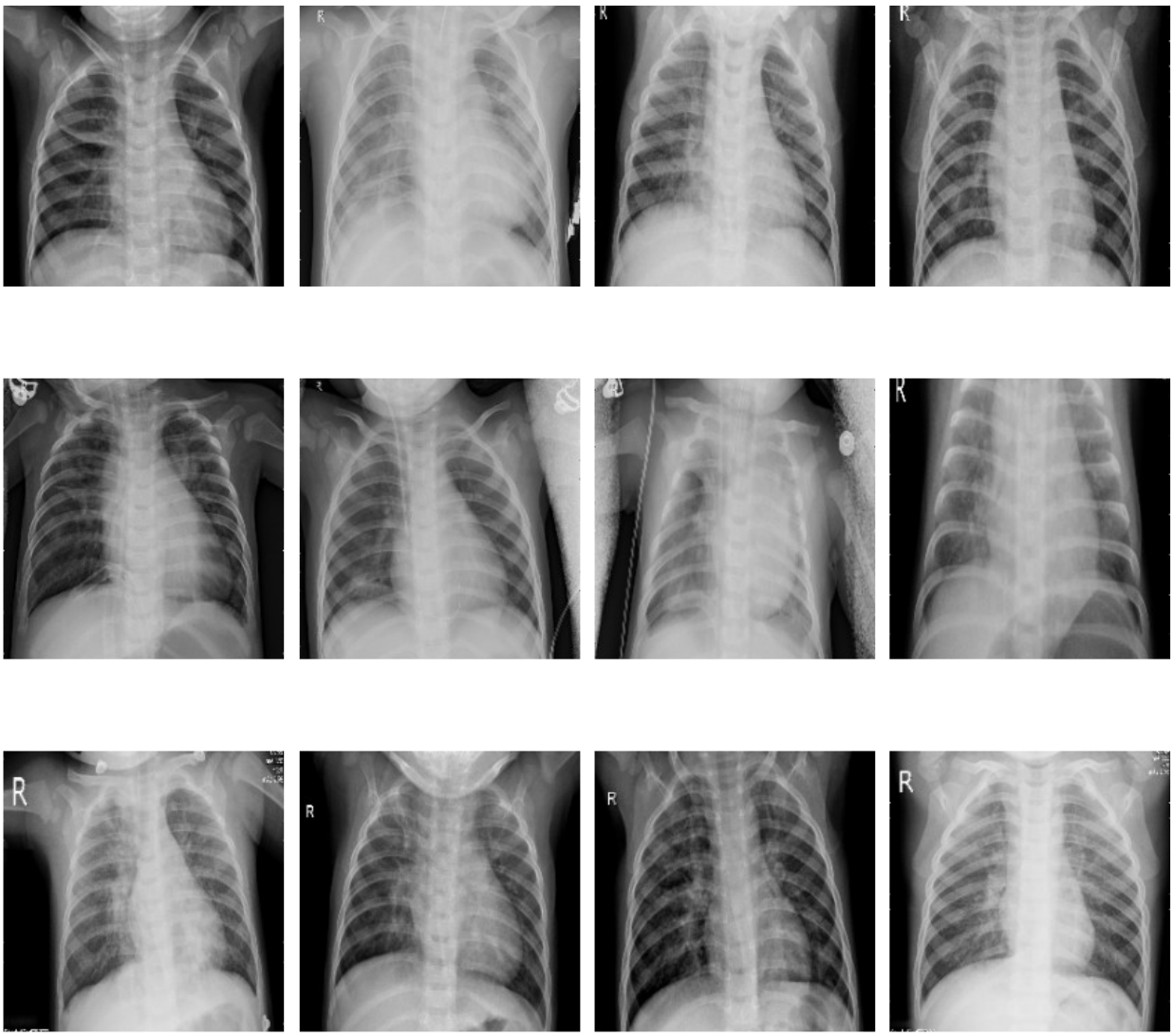
```
print('Test Set - Normal')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set - Normal
```



```
print('Test Set - Pneumonia')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set - Pneumonia
```



Preparing the Data

First, we need to create a validation set. To do that, we apply a simple stratified split on the original train dataset, using 80% for actual training and 20% for validation purposes.

```
train_df, val_df = train_test_split(df_train, test_size = 0.20,
random_state = SEED, stratify = df_train['class'])

train_df
```

```
          class  \
3566  Pneumonia
2866  Pneumonia
2681  Pneumonia
1199     Normal
```

```
4619  Pneumonia
...         ...
3476  Pneumonia
678      Normal
1560  Pneumonia
2769  Pneumonia
4881  Pneumonia


image
3566
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
6559742-0002.jpeg
2866
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4609274-0001.jpeg
2681
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
1149864-0002.jpeg
1199
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
6315622-0002.jpeg
4619
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1753514-0002.jpeg
...
...
3476
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
3408906-0008.jpeg
678
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
4688588-0001.jpeg
1560
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-8467199-0001.jpeg
2769
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-6909155-0001.jpeg
4881
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1820776-0002.jpeg

[4185 rows x 2 columns]

val_df

          class  \
2945  Pneumonia
4878  Pneumonia
```

```
3177  Pneumonia
972      Normal
3059  Pneumonia
...          ...
253      Normal
4315  Pneumonia
687      Normal
3417  Pneumonia
3745  Pneumonia


image
2945
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-2474652-0008.jpeg
4878
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-2739133-0001.jpeg
3177
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4133607-0005.jpeg
972
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
3476904-0001.jpeg
3059
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
8446214-0003.jpeg
...
...
253
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
9862823-0002.jpeg
4315
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1083680-0014.jpeg
687
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
8991560-0001.jpeg
3417
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-7942103-0006.jpeg
3745
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4638514-0001.jpeg

[1047 rows x 2 columns]
```

Now, we're going to load the images from the folders and prepare them to feed our models.

We begin by defining the data generators. With Keras Image Data Generator, we can rescale the pixel values and apply random transformation techniques for data augmentation on the fly. We define two different generators. The val_datagen is used to simply rescale the validation and test sets. The train_datagen includes some transformations to augment the train set.

We apply those generators on each dataset using the flow_from_dataframe method. Apart from the transformations defined in each generator, the images are also resized based on the target_size set.

```
# https://vijayabhaskar96.medium.com/tutorial-on-keras-flow-from-dataframe-1fd4493d237c

train_datagen = ImageDataGenerator(rescale=1/255.,
                                   zoom_range = 0.1,
                                   #rotation_range = 0.1,
                                   width_shift_range = 0.1,
                                   height_shift_range = 0.1)

val_datagen = ImageDataGenerator(rescale=1/255.)

ds_train = train_datagen.flow_from_dataframe(train_df,
                                             #directory=train_path,
#dataframe contains the full paths
                                             x_col = 'image',
                                             y_col = 'class',
                                             target_size = (IMG_SIZE,
IMG_SIZE),
                                             class_mode = 'binary',
                                             batch_size = BATCH,
                                             seed = SEED)

ds_val = val_datagen.flow_from_dataframe(val_df,
                                         #directory=train_path,
                                         x_col = 'image',
                                         y_col = 'class',
                                         target_size = (IMG_SIZE,
IMG_SIZE),
                                         class_mode = 'binary',
                                         batch_size = BATCH,
                                         seed = SEED)

ds_test = val_datagen.flow_from_dataframe(df_test,
                                         #directory=test_path,
                                         x_col = 'image',
                                         y_col = 'class',
                                         target_size = (IMG_SIZE,
IMG_SIZE),
                                         class_mode = 'binary',
                                         batch_size = 1,
                                         shuffle = False)
```

```
Found 4185 validated image filenames belonging to 2 classes.
Found 1047 validated image filenames belonging to 2 classes.
Found 624 validated image filenames belonging to 2 classes.
```

Now, we are ready for the next stage: creating and training the image classification models.

Using CNN Model

```
#Setting callbakcs

early_stopping = callbacks.EarlyStopping(
    monitor='val_loss',
    patience=5,
    min_delta=1e-7,
    restore_best_weights=True,
)

plateau = callbacks.ReduceLROnPlateau(
    monitor='val_loss',
    factor = 0.2,
    patience = 2,
    min_delt = 1e-7,
    cooldown = 0,
    verbose = 1
)
```

Let's define our first model 'from scratch' and see how it performs.

```
def get_model():

    #Input shape = [width, height, color channels]
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

    # Block One
    x = layers.Conv2D(filters=16, kernel_size=3, padding='valid')
(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)

    # Block Two
    x = layers.Conv2D(filters=32, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.2)(x)
```

```
    # Block Three
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.Conv2D(filters=64, kernel_size=3, padding='valid')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPool2D()(x)
    x = layers.Dropout(0.4)(x)

    # Head
    #x = layers.BatchNormalization()(x)
    x = layers.Flatten()(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.5)(x)

    #Final Layer (Output)
    output = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs=[inputs], outputs=output)

    return model

keras.backend.clear_session()

model = get_model()
model.compile(loss='binary_crossentropy'
              , optimizer = keras.optimizers.Adam(learning_rate=3e-5),
metrics='binary_accuracy')

model.summary()
```

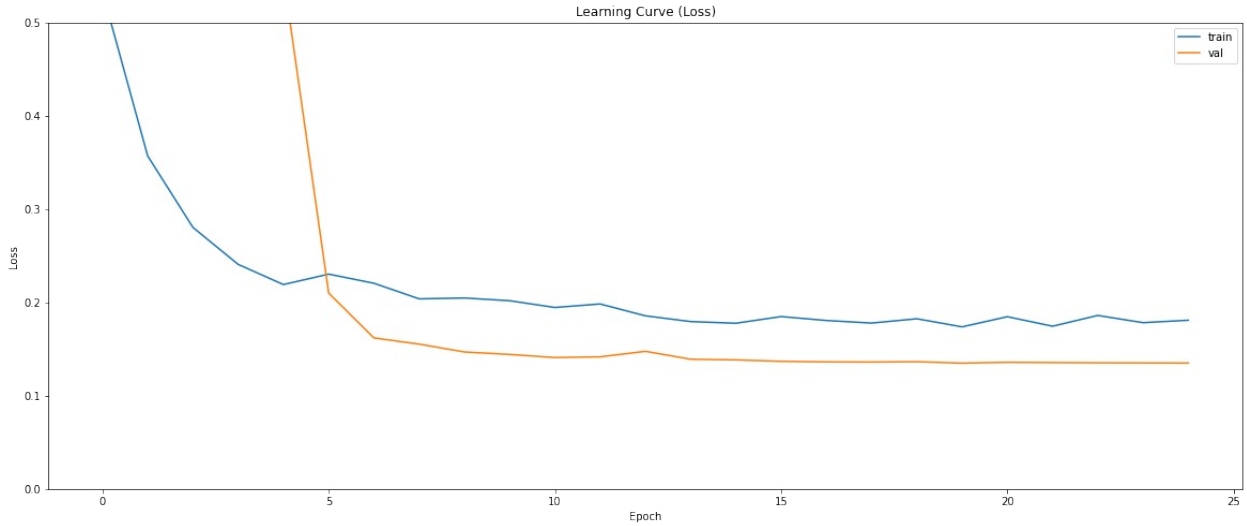
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 222, 222, 16)      448
_________________________________________________________________
batch_normalization (BatchNo (None, 222, 222, 16)      64
_________________________________________________________________
activation (Activation)      (None, 222, 222, 16)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 16)      0
_________________________________________________________________
dropout (Dropout)            (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 109, 109, 32)      4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 109, 109, 32)      128
_________________________________________________________________
```

```
activation_1 (Activation)    (None, 109, 109, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 54, 54, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 52, 52, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 64)        36928
_________________________________________________________________
batch_normalization_2 (Batch (None, 50, 50, 64)        256
_________________________________________________________________
activation_2 (Activation)    (None, 50, 50, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 25, 25, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 40000)             0
_________________________________________________________________
dense (Dense)                (None, 64)                2560064
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 2,621,089
Trainable params: 2,620,865
Non-trainable params: 224
_________________________________________________________________

history = model.fit(ds_train,
          batch_size = BATCH, epochs = 50,
          validation_data=ds_val,
          callbacks=[early_stopping, plateau],
          steps_per_epoch=(len(train_df)/BATCH),
          validation_steps=(len(val_df)/BATCH));

Epoch 1/50
130/130 [==============================] - 101s 767ms/step - loss:
0.6391 - binary_accuracy: 0.7145 - val_loss: 0.7722 -
val_binary_accuracy: 0.7421
Epoch 2/50
130/130 [==============================] - 100s 766ms/step - loss:
0.3928 - binary_accuracy: 0.8159 - val_loss: 1.2521 -
val_binary_accuracy: 0.7421
Epoch 3/50
130/130 [==============================] - 100s 768ms/step - loss:
0.3035 - binary_accuracy: 0.8698 - val_loss: 1.4061 -
val_binary_accuracy: 0.7421
```

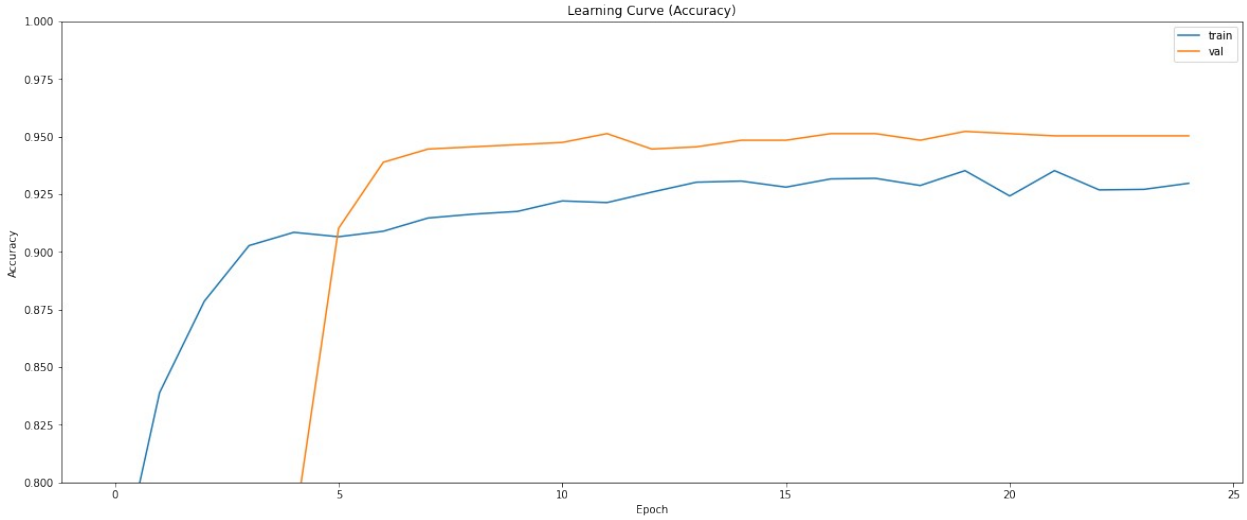
```
Epoch 00003: ReduceLROnPlateau reducing learning rate to
5.9999998484272515e-06.
Epoch 4/50
130/130 [==============================] - 100s 765ms/step - loss:
0.2445 - binary_accuracy: 0.8957 - val_loss: 1.1610 -
val_binary_accuracy: 0.7421
Epoch 5/50
130/130 [==============================] - 100s 764ms/step - loss:
0.2176 - binary_accuracy: 0.9116 - val_loss: 0.5502 -
val_binary_accuracy: 0.7784
Epoch 6/50
130/130 [==============================] - 99s 759ms/step - loss:
0.2325 - binary_accuracy: 0.9110 - val_loss: 0.2100 -
val_binary_accuracy: 0.9102
Epoch 7/50
130/130 [==============================] - 100s 764ms/step - loss:
0.2114 - binary_accuracy: 0.9126 - val_loss: 0.1620 -
val_binary_accuracy: 0.9389
Epoch 8/50
130/130 [==============================] - 100s 765ms/step - loss:
0.1958 - binary_accuracy: 0.9197 - val_loss: 0.1553 -
val_binary_accuracy: 0.9446
Epoch 9/50
130/130 [==============================] - 100s 766ms/step - loss:
0.2095 - binary_accuracy: 0.9100 - val_loss: 0.1469 -
val_binary_accuracy: 0.9456
Epoch 10/50
130/130 [==============================] - 100s 761ms/step - loss:
0.1932 - binary_accuracy: 0.9221 - val_loss: 0.1443 -
val_binary_accuracy: 0.9465
Epoch 11/50
130/130 [==============================] - 100s 762ms/step - loss:
0.1942 - binary_accuracy: 0.9264 - val_loss: 0.1410 -
val_binary_accuracy: 0.9475
Epoch 12/50
130/130 [==============================] - 100s 761ms/step - loss:
0.1806 - binary_accuracy: 0.9248 - val_loss: 0.1417 -
val_binary_accuracy: 0.9513
Epoch 13/50
130/130 [==============================] - 99s 760ms/step - loss:
0.1887 - binary_accuracy: 0.9236 - val_loss: 0.1476 -
val_binary_accuracy: 0.9446

Epoch 00013: ReduceLROnPlateau reducing learning rate to
1.1999999514955563e-06.
Epoch 14/50
130/130 [==============================] - 100s 763ms/step - loss:
0.1752 - binary_accuracy: 0.9334 - val_loss: 0.1391 -
```

```
val_binary_accuracy: 0.9456
Epoch 15/50
130/130 [==============================] - 100s 766ms/step - loss:
0.1896 - binary_accuracy: 0.9262 - val_loss: 0.1384 -
val_binary_accuracy: 0.9484
Epoch 16/50
130/130 [==============================] - 101s 769ms/step - loss:
0.2044 - binary_accuracy: 0.9254 - val_loss: 0.1369 -
val_binary_accuracy: 0.9484
Epoch 17/50
130/130 [==============================] - 101s 770ms/step - loss:
0.1856 - binary_accuracy: 0.9278 - val_loss: 0.1363 -
val_binary_accuracy: 0.9513
Epoch 18/50
130/130 [==============================] - 101s 770ms/step - loss:
0.1729 - binary_accuracy: 0.9312 - val_loss: 0.1361 -
val_binary_accuracy: 0.9513
Epoch 19/50
130/130 [==============================] - 100s 765ms/step - loss:
0.1774 - binary_accuracy: 0.9335 - val_loss: 0.1364 -
val_binary_accuracy: 0.9484
Epoch 20/50
130/130 [==============================] - 100s 761ms/step - loss:
0.1673 - binary_accuracy: 0.9365 - val_loss: 0.1349 -
val_binary_accuracy: 0.9522
Epoch 21/50
130/130 [==============================] - 99s 759ms/step - loss:
0.1823 - binary_accuracy: 0.9253 - val_loss: 0.1358 -
val_binary_accuracy: 0.9513
Epoch 22/50
130/130 [==============================] - 100s 761ms/step - loss:
0.1670 - binary_accuracy: 0.9395 - val_loss: 0.1355 -
val_binary_accuracy: 0.9503

Epoch 00022: ReduceLROnPlateau reducing learning rate to
2.3999998575163774e-07.
Epoch 23/50
130/130 [==============================] - 99s 760ms/step - loss:
0.1986 - binary_accuracy: 0.9197 - val_loss: 0.1353 -
val_binary_accuracy: 0.9503
Epoch 24/50
130/130 [==============================] - 99s 761ms/step - loss:
0.1758 - binary_accuracy: 0.9293 - val_loss: 0.1352 -
val_binary_accuracy: 0.9503

Epoch 00024: ReduceLROnPlateau reducing learning rate to
4.799999828719592e-08.
Epoch 25/50
130/130 [==============================] - 99s 757ms/step - loss:
```

```
0.1900 - binary_accuracy: 0.9283 - val_loss: 0.1351 -
val_binary_accuracy: 0.9503

fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y = history.history['loss'])
sns.lineplot(x = history.epoch, y = history.history['val_loss'])
ax.set_title('Learning Curve (Loss)')
ax.set_ylabel('Loss')
ax.set_xlabel('Epoch')
ax.set_ylim(0, 0.5)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y =
history.history['binary_accuracy'])
sns.lineplot(x = history.epoch, y =
history.history['val_binary_accuracy'])
ax.set_title('Learning Curve (Accuracy)')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylim(0.80, 1.0)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
score = model.evaluate(ds_val, steps = len(val_df)/BATCH, verbose = 0)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
Val loss: 0.1349286437034607
Val accuracy: 0.9522445201873779
```

```
score = model.evaluate(ds_test, steps = len(df_test), verbose = 0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.4296790361404419
Test accuracy: 0.8493589758872986
```

Experiment 8: Building a Recurrent Neural Network (RNN) for Predicting Patient Readmission

---

Aim:
To develop a Recurrent Neural Network (RNN) model for predicting patient readmission rates based on historical patient data. This experiment aims to leverage sequential data to identify patterns that contribute to readmission, enabling healthcare providers to implement preventive measures and improve patient outcomes.

Theory:
Recurrent Neural Networks (RNNs) are a class of neural networks designed to process sequential data by maintaining a hidden state that captures information about previous inputs. This characteristic makes RNNs particularly suitable for tasks where temporal dependencies are significant, such as time series forecasting or natural language processing.

Key components of RNNs include:

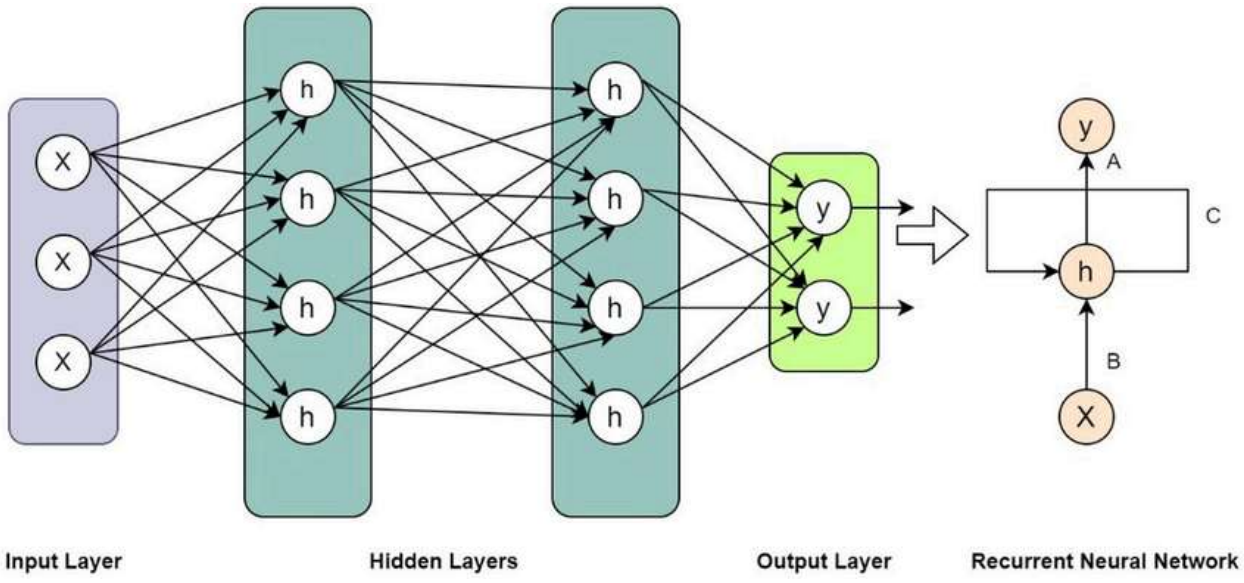
1. **Hidden State:** The hidden state in RNNs acts as a memory that holds information about previous time steps. It is updated at each time step based on the current input and the previous hidden state.
2. **Input Sequence:** The model takes a sequence of inputs, which can represent time series data (e.g., patient vitals over time) or ordered categorical data (e.g., patient diagnoses and treatments).
3. **Activation Functions:** Non-linear activation functions (e.g., tanh or ReLU) are applied to the hidden state to introduce non-linearity in the model.
4. **Loss Function:** The loss function (e.g., binary cross-entropy for binary classification) measures how well the model's predictions match the actual readmission outcomes.
5. **Optimization Algorithm:** An optimization algorithm (e.g., Adam or RMSprop) is used to minimize the loss function by updating the model's parameters during training.

Steps to Implement RNN for Patient Readmission Prediction:

1. **Data Collection:** Gather a dataset containing historical patient records, including features such as demographics, clinical measurements, diagnosis codes, treatment history, and readmission status.
2. **Data Preprocessing:** Clean and preprocess the data by handling missing values, normalizing features, and encoding categorical variables. Structure the data as sequences, with each patient's record represented over time.
3. **Model Architecture Design:** Design an RNN architecture suitable for predicting readmissions. This may involve using Long Short-Term Memory (LSTM) or Gated

Recurrent Unit (GRU) cells, which are advanced types of RNNs designed to handle vanishing gradient issues.

4. **Model Compilation:** Compile the RNN model by selecting an appropriate loss function, optimizer, and evaluation metrics. This step is crucial for ensuring the model is trained correctly.
5. **Model Training:** Train the model using the training dataset, monitoring its performance on a validation set to prevent overfitting. Use techniques like dropout regularization to improve generalization.
6. **Model Evaluation:** Evaluate the trained model on a separate test dataset, calculating performance metrics such as accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC-ROC).
7. **Result Interpretation:** Analyze the model's predictions to understand factors influencing patient readmission. Identify key features that contribute to readmission risk.
8. **Deployment:** If the model demonstrates satisfactory performance, deploy it in clinical settings to assist healthcare providers in making informed decisions regarding patient management and discharge planning.



[1]:
```
import pandas as pd
import numpy as np
from datetime import timedelta, datetime
```

[6]:
```
import numpy as np
import pandas as pd
from datetime import datetime, timedelta

np.random.seed(42)

num_patients = 1000

patient_ids = [f'Patient_{i+1}' for i in range(num_patients)]

ages = np.random.randint(20, 80, size=num_patients)

genders = np.random.choice(['Male', 'Female'], size=num_patients)

start_date = datetime(2023, 1, 1)
end_date = datetime(2023, 12, 31)
admission_dates = [start_date + timedelta(days=np.random.randint(0, 365)) for _␣
 ↪in range(num_patients)]

length_of_stays = np.random.randint(1, 15, size=num_patients)

num_medications = np.random.randint(1, 6, size=num_patients)

diagnosis_options = ['Circulatory', 'Respiratory', 'Other', 'Diabetes']
diag1 = np.random.choice(diagnosis_options, size=num_patients)
diag2 = np.random.choice(diagnosis_options, size=num_patients)
diag3 = np.random.choice(diagnosis_options, size=num_patients)

diabetes_medication = np.random.choice(['Yes', 'No'], size=num_patients)

readmissions = []
for length in length_of_stays:
    if length > 14:
```

```
        readmissions.append(0)
    else:
        readmissions.append(np.random.choice([0, 1], p=[0.7, 0.3]))

data = pd.DataFrame({
    'Patient_ID': patient_ids,
    'Age': ages,
    'Gender': genders,
    'Admission_Date': admission_dates,
    'Length_of_Stay': length_of_stays,
    'Num_Medications': num_medications,
    'diag1': diag1,
    'diag2': diag2,
    'diag3': diag3,
    'Diabetes_Medication': diabetes_medication,
    'Readmission': readmissions
})
```

[7]:
```
data
```

[7]:
```
       Patient_ID  Age  Gender Admission_Date  Length_of_Stay  \
0       Patient_1   58  Female     2023-04-18               4
1       Patient_2   71  Female     2023-11-07               1
2       Patient_3   48    Male     2023-08-15               3
3       Patient_4   34  Female     2023-06-19               1
4       Patient_5   62    Male     2023-02-03              12
..            ...  ...     ...            ...             ...
995   Patient_996   23    Male     2023-08-30               8
996   Patient_997   20    Male     2023-08-01               8
997   Patient_998   68  Female     2023-06-30              13
998   Patient_999   59    Male     2023-12-04               5
999  Patient_1000   51  Female     2023-06-22              13

     Num_Medications        diag1        diag2        diag3  \
0                  4  Circulatory  Circulatory  Circulatory
1                  5  Circulatory  Circulatory  Respiratory
2                  1  Respiratory     Diabetes     Diabetes
3                  5  Respiratory  Respiratory     Diabetes
4                  2     Diabetes  Respiratory  Circulatory
..               ...          ...          ...          ...
995                4  Circulatory  Circulatory        Other
996                3  Circulatory     Diabetes     Diabetes
997                1  Circulatory  Respiratory  Respiratory
998                2  Respiratory     Diabetes        Other
999                4        Other  Respiratory  Respiratory

    Diabetes_Medication  Readmission
```

```
0                   Yes            0
1                    No            1
2                   Yes            0
3                    No            1
4                   Yes            0
..                  …          …
995                  No            0
996                 Yes            0
997                  No            0
998                  No            0
999                  No            0

[1000 rows x 11 columns]
```

[8]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
from tensorflow.keras.utils import to_categorical
```

[9]:
```
data['Gender'] = LabelEncoder().fit_transform(data['Gender'])  # Male=1,␣
 ↪Female=0
data['Diabetes_Medication'] = LabelEncoder().
 ↪fit_transform(data['Diabetes_Medication'])  # Yes=1, No=0
```

[10]:
```
diag_columns = ['diag1', 'diag2', 'diag3']
for col in diag_columns:
    data[col] = LabelEncoder().fit_transform(data[col])
```

[11]:
```
data.drop(columns=['Patient_ID', 'Admission_Date'], inplace=True)
```

[64]:
```
scaler = MinMaxScaler()
data[['Age', 'Length_of_Stay', 'Num_Medications']] = scaler.
 ↪fit_transform(data[['Age', 'Length_of_Stay', 'Num_Medications']])

X = data.drop(columns=['Readmission']).values
y = data['Readmission'].values

X = X.reshape(X.shape[0], 1, X.shape[1])
```

[13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[50]:
```
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.layers import Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import BatchNormalization


scaler = StandardScaler()
X_scaled = X.copy()
X_scaled[:, :, :4] = scaler.fit_transform(X[:, :, :4].reshape(-1, 4)).reshape(X.
 ↪shape[0], X.shape[1], 4)

# Step 2: Build the improved RNN model
model = Sequential()
model.add(SimpleRNN(128, activation='relu', input_shape=(X_scaled.shape[1],␣
 ↪X_scaled.shape[2]), return_sequences=True))
model.add(BatchNormalization())  # Adding Batch Normalization
model.add(Dropout(0.3))
model.add(SimpleRNN(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

[51]:
```
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])
```

[52]:
```
history = model.fit(X_train, y_train, epochs=30, batch_size=16,␣
 ↪validation_data=(X_test, y_test))
```

```
Epoch 1/30
50/50             4s 12ms/step -
accuracy: 0.4774 - loss: 0.8103 - val_accuracy: 0.7550 - val_loss: 0.6093
Epoch 2/30
50/50             1s 4ms/step -
accuracy: 0.6770 - loss: 0.6445 - val_accuracy: 0.7550 - val_loss: 0.6175
Epoch 3/30
50/50             0s 4ms/step -
accuracy: 0.7047 - loss: 0.6042 - val_accuracy: 0.7550 - val_loss: 0.6171
Epoch 4/30
50/50             0s 4ms/step -
accuracy: 0.7202 - loss: 0.5907 - val_accuracy: 0.7550 - val_loss: 0.5945
```

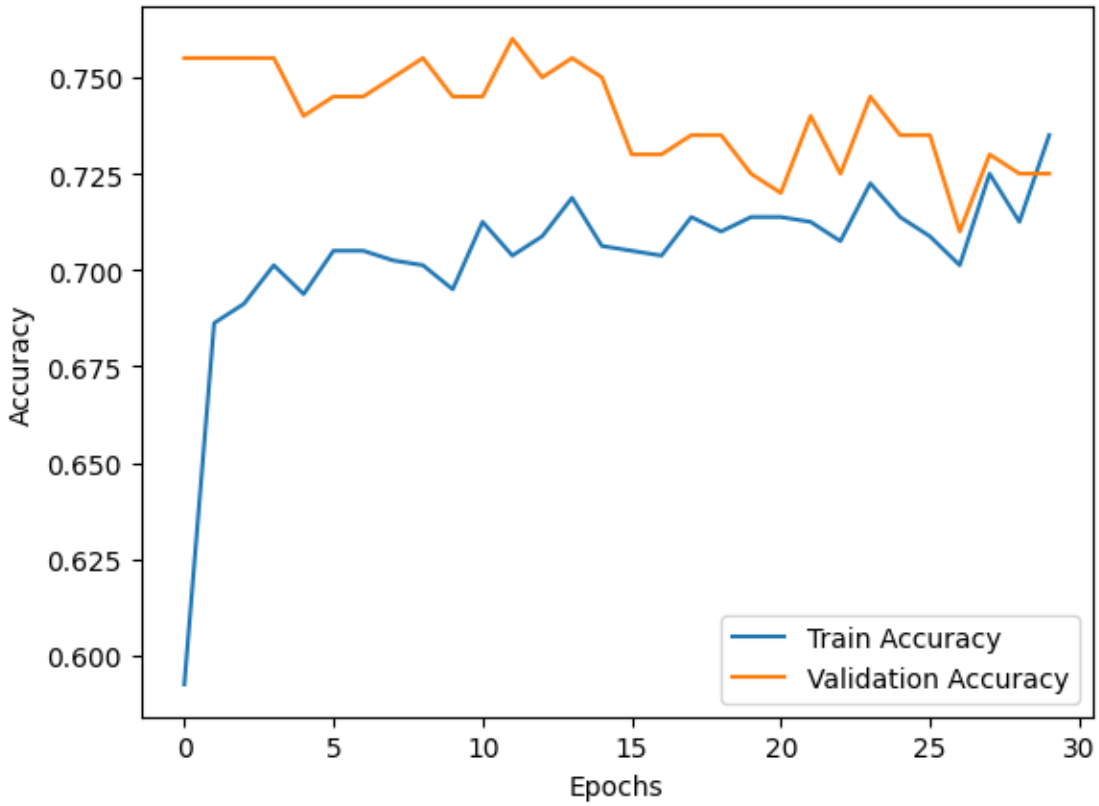
```
Epoch 5/30
50/50             0s 4ms/step -
accuracy: 0.7043 - loss: 0.5992 - val_accuracy: 0.7400 - val_loss: 0.6054
Epoch 6/30
50/50             0s 5ms/step -
accuracy: 0.6806 - loss: 0.6293 - val_accuracy: 0.7450 - val_loss: 0.5860
Epoch 7/30
50/50             0s 4ms/step -
accuracy: 0.7104 - loss: 0.5986 - val_accuracy: 0.7450 - val_loss: 0.5794
Epoch 8/30
50/50             0s 4ms/step -
accuracy: 0.7127 - loss: 0.5888 - val_accuracy: 0.7500 - val_loss: 0.5695
Epoch 9/30
50/50             0s 4ms/step -
accuracy: 0.7078 - loss: 0.5889 - val_accuracy: 0.7550 - val_loss: 0.5739
Epoch 10/30
50/50             0s 5ms/step -
accuracy: 0.6888 - loss: 0.5972 - val_accuracy: 0.7450 - val_loss: 0.5751
Epoch 11/30
50/50             0s 4ms/step -
accuracy: 0.7006 - loss: 0.5872 - val_accuracy: 0.7450 - val_loss: 0.5686
Epoch 12/30
50/50             0s 4ms/step -
accuracy: 0.7181 - loss: 0.5767 - val_accuracy: 0.7600 - val_loss: 0.5695
Epoch 13/30
50/50             0s 4ms/step -
accuracy: 0.7078 - loss: 0.5989 - val_accuracy: 0.7500 - val_loss: 0.5704
Epoch 14/30
50/50             0s 4ms/step -
accuracy: 0.6924 - loss: 0.6227 - val_accuracy: 0.7550 - val_loss: 0.5633
Epoch 15/30
50/50             0s 4ms/step -
accuracy: 0.7039 - loss: 0.5676 - val_accuracy: 0.7500 - val_loss: 0.5710
Epoch 16/30
50/50             0s 7ms/step -
accuracy: 0.7342 - loss: 0.5562 - val_accuracy: 0.7300 - val_loss: 0.5806
Epoch 17/30
50/50             1s 6ms/step -
accuracy: 0.7005 - loss: 0.6054 - val_accuracy: 0.7300 - val_loss: 0.5811
Epoch 18/30
50/50             1s 6ms/step -
accuracy: 0.7064 - loss: 0.5780 - val_accuracy: 0.7350 - val_loss: 0.5759
Epoch 19/30
50/50             0s 6ms/step -
accuracy: 0.6913 - loss: 0.5970 - val_accuracy: 0.7350 - val_loss: 0.5789
Epoch 20/30
50/50             0s 7ms/step -
accuracy: 0.7182 - loss: 0.5825 - val_accuracy: 0.7250 - val_loss: 0.6006
```

```
Epoch 21/30
50/50             0s 4ms/step -
accuracy: 0.7158 - loss: 0.5682 - val_accuracy: 0.7200 - val_loss: 0.5941
Epoch 22/30
50/50             0s 4ms/step -
accuracy: 0.7638 - loss: 0.5135 - val_accuracy: 0.7400 - val_loss: 0.6018
Epoch 23/30
50/50             0s 4ms/step -
accuracy: 0.7070 - loss: 0.5714 - val_accuracy: 0.7250 - val_loss: 0.5945
Epoch 24/30
50/50             0s 4ms/step -
accuracy: 0.7232 - loss: 0.5724 - val_accuracy: 0.7450 - val_loss: 0.5862
Epoch 25/30
50/50             0s 4ms/step -
accuracy: 0.7004 - loss: 0.5770 - val_accuracy: 0.7350 - val_loss: 0.6118
Epoch 26/30
50/50             0s 4ms/step -
accuracy: 0.6733 - loss: 0.5805 - val_accuracy: 0.7350 - val_loss: 0.5930
Epoch 27/30
50/50             0s 4ms/step -
accuracy: 0.7166 - loss: 0.5691 - val_accuracy: 0.7100 - val_loss: 0.6027
Epoch 28/30
50/50             0s 4ms/step -
accuracy: 0.7010 - loss: 0.5731 - val_accuracy: 0.7300 - val_loss: 0.6011
Epoch 29/30
50/50             0s 4ms/step -
accuracy: 0.6869 - loss: 0.6018 - val_accuracy: 0.7250 - val_loss: 0.6054
Epoch 30/30
50/50             0s 4ms/step -
accuracy: 0.7302 - loss: 0.5669 - val_accuracy: 0.7250 - val_loss: 0.6149
```

[53]:
```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f" Accuracy with SimpleRNN: {accuracy:.4f}")
```

```
7/7             0s 2ms/step -
accuracy: 0.7224 - loss: 0.6162
 Accuracy with SimpleRNN: 0.7250
```

[54]:
```python
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[62]:
```
gender_mapping = {'Male': 1, 'Female': 0}
diabetes_mapping = {'Yes': 1, 'No': 0}
diagnosis_mapping = {diag: idx for idx, diag in enumerate(set(diag1).
 ↪union(diag2, diag3))}

data['Gender'] = data['Gender'].map(gender_mapping)
data['Diabetes_Medication'] = data['Diabetes_Medication'].map(diabetes_mapping)
data['diag1'] = data['diag1'].map(diagnosis_mapping)
data['diag2'] = data['diag2'].map(diagnosis_mapping)
data['diag3'] = data['diag3'].map(diagnosis_mapping)
```

[67]:
```
#Predicting Readmission for a New Patient

def predict_readmission(new_patient_data):
    # Create DataFrame for new patient data
    new_patient_df = pd.DataFrame([new_patient_data])

    new_patient_df['Gender'] = new_patient_df['Gender'].map(gender_mapping)
    new_patient_df['Diabetes_Medication'] =␣
 ↪new_patient_df['Diabetes_Medication'].map(diabetes_mapping)
    new_patient_df['diag1'] = new_patient_df['diag1'].map(diagnosis_mapping)
```

```
    new_patient_df['diag2'] = new_patient_df['diag2'].map(diagnosis_mapping)
    new_patient_df['diag3'] = new_patient_df['diag3'].map(diagnosis_mapping)

    new_patient_df[['Age', 'Length_of_Stay', 'Num_Medications']] = scaler.
 ↪transform(new_patient_df[['Age', 'Length_of_Stay', 'Num_Medications']])

    new_patient_scaled = new_patient_df.values.reshape(1, 1, new_patient_df.
 ↪shape[1])
    predicted_prob = model.predict(new_patient_scaled)
    predicted_readmission = int(predicted_prob > 0.5)

    return predicted_prob[0][0], predicted_readmission

new_patient_data = {
    'Age': 71,
    'Gender': 'Female',
    'Length_of_Stay': 1,
    'Num_Medications': 5,
    'diag1': 'Circulatory',
    'diag2': 'Circulatory',
    'diag3': 'Respiratory',
    'Diabetes_Medication': 'No'
}

# Predict readmission
probability, readmission = predict_readmission(new_patient_data)
if probability is not None:
    print(f"Predicted Readmission Probability: {probability:.4f}")
    print(f"Predicted Readmission: {'Yes' if readmission == 1 else 'No'}")
```

```
1/1             0s 30ms/step
Predicted Readmission Probability: 0.0082
Predicted Readmission: No
```

```
<ipython-input-67-55bc8f2fc103>:20: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  predicted_readmission = int(predicted_prob > 0.5)  # Convert probability to
binary outcome
```

[ ]:

Experiment 9: Implementing Transfer Learning for Medical Image Feature Extraction

---

Aim:
To leverage transfer learning techniques to extract meaningful features from medical images, improving the efficiency and accuracy of medical diagnostics. This experiment aims to utilize pre-trained deep learning models on large datasets to facilitate the feature extraction process for specific medical imaging tasks, such as identifying tumors or abnormalities in radiological scans.

Theory:
Transfer learning is a machine learning technique where a model developed for a specific task is reused as the starting point for a model on a second task. This approach is particularly useful in domains like medical imaging, where labeled data can be scarce and expensive to obtain.

Key concepts include:

1. **Pre-trained Models:** These are deep learning models trained on large datasets (e.g., ImageNet) that can generalize well to various tasks. Common pre-trained models include VGG16, ResNet, Inception, and MobileNet.
2. **Feature Extraction:** In transfer learning, the earlier layers of a pre-trained model capture general features (edges, textures, etc.), while later layers capture more task-specific features. By using the earlier layers of a pre-trained model, we can extract features from medical images without training a model from scratch.
3. **Fine-tuning:** After feature extraction, the model can be fine-tuned on a smaller dataset specific to the medical imaging task. This involves training the model on the new dataset while adjusting the weights slightly to adapt to the new features.
4. **Data Augmentation:** Techniques such as rotation, flipping, scaling, and normalization can enhance the dataset by creating variations of the existing images, improving the model's robustness and ability to generalize.
5. **Evaluation Metrics:** Common metrics for evaluating feature extraction in medical imaging include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC).

Dataset Description

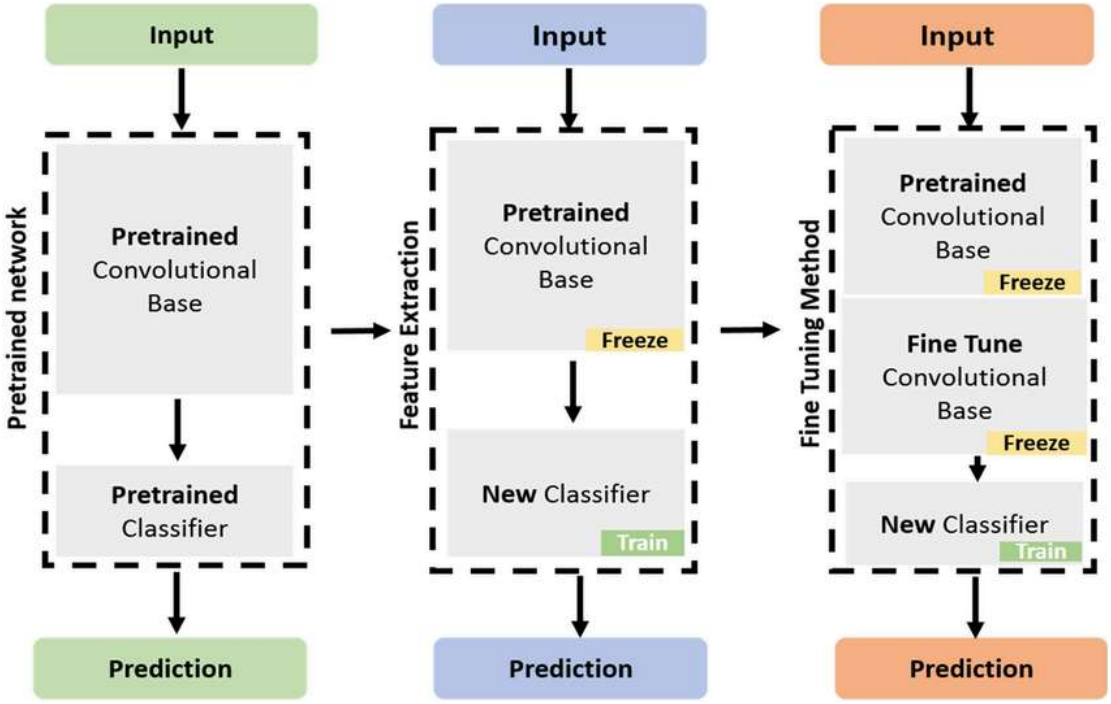
This dataset contains 5,856 validated Chest X-Ray images. The images are split into a training set and a testing set of independent patients. Images are labeled as (disease:NORMAL/BACTERIA/VIRUS)-(randomized patient ID)-(image number of a patient).

Chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

For the analysis of chest x-ray images, all chest radiographs were initially screened for quality control by removing all low quality or unreadable scans. The diagnoses for the images were then graded by two expert physicians before being cleared for training the AI system. In order to account for any grading errors, the evaluation set was also checked by a third expert.

Steps to Implement Transfer Learning for Medical Image Feature Extraction:

1. **Data Collection:** Acquire a medical imaging dataset that is relevant to the diagnostic task. This could involve images of tumors, lesions, or other abnormalities across various modalities such as X-rays, MRIs, or CT scans.
2. **Data Preprocessing:** Preprocess the images to ensure they are suitable for input into the pre-trained model. This includes resizing images, normalizing pixel values, and applying any necessary data augmentation techniques.
3. **Model Selection:** Choose a suitable pre-trained model based on the specific task and computational resources. Popular choices include VGG16, ResNet50, and InceptionV3.
4. **Feature Extraction:** Load the pre-trained model, excluding the final classification layers. Use the model to extract features from the medical images, generating a feature vector for each image.
5. **Feature Analysis:** Analyze the extracted features to identify patterns and relationships relevant to the medical diagnostic task. This may involve visualization techniques such as t-SNE or PCA to explore the feature space.
6. **Fine-tuning (optional):** If a labeled dataset is available, fine-tune the model by training a classifier on top of the extracted features. This step can enhance the model's performance on the specific task by adjusting the weights of the later layers.
7. **Model Evaluation:** Evaluate the effectiveness of the feature extraction process by assessing the model's performance on a validation or test dataset. Utilize appropriate metrics to measure success.
8. **Result Interpretation:** Interpret the results to draw insights into the predictive power of the extracted features. Analyze the significance of different features in relation to diagnostic outcomes.
9. **Deployment:** If the feature extraction model demonstrates utility, consider deploying it as part of a larger diagnostic tool to assist healthcare professionals in making data-driven decisions based on medical images.


Input
Pretrained network
Pretrained Convolutional Base
Pretrained Classifier
Prediction
Input
Feature Extraction
Pretrained Convolutional Base
Freeze
New Classifier
Train
Prediction
Input
Fine Tuning Method
Pretrained Convolutional Base
Freeze
Fine Tune Convolutional Base
Freeze
New Classifier
Train
Prediction

Importing Packages and Dataset

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

pd.options.display.max_colwidth = 100

import random
import os

from numpy.random import seed
seed(42)

random.seed(42)
os.environ['PYTHONHASHSEED'] = str(42)
os.environ['TF_DETERMINISTIC_OPS'] = '1'

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import callbacks
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

import glob
import cv2

from tensorflow.random import set_seed
set_seed(42)

import warnings
warnings.filterwarnings('ignore')

IMG_SIZE = 224
BATCH = 32
SEED = 42

#main_path = "../input/chest-xray-pneumonia/chest_xray/"
main_path = "Downloads/archive/chest_xray"
```

```
train_path = os.path.join(main_path,"train")
test_path=os.path.join(main_path,"test")

train_normal = glob.glob(train_path+"/NORMAL/*.jpeg")
train_pneumonia = glob.glob(train_path+"/PNEUMONIA/*.jpeg")

test_normal = glob.glob(test_path+"/NORMAL/*.jpeg")
test_pneumonia = glob.glob(test_path+"/PNEUMONIA/*.jpeg")

train_list = [x for x in train_normal]
train_list.extend([x for x in train_pneumonia])

df_train = pd.DataFrame(np.concatenate([['Normal']*len(train_normal) ,
['Pneumonia']*len(train_pneumonia)]), columns = ['class'])
df_train['image'] = [x for x in train_list]

test_list = [x for x in test_normal]
test_list.extend([x for x in test_pneumonia])

df_test = pd.DataFrame(np.concatenate([['Normal']*len(test_normal) ,
['Pneumonia']*len(test_pneumonia)]), columns = ['class'])
df_test['image'] = [x for x in test_list]

df_train
```

```
          class  \
0        Normal
1        Normal
2        Normal
3        Normal
4        Normal
...         ...
5211  Pneumonia
5212  Pneumonia
5213  Pneumonia
5214  Pneumonia
5215  Pneumonia


image
0               Downloads/archive/chest_xray\train/NORMAL\IM-0115-
0001.jpeg
1               Downloads/archive/chest_xray\train/NORMAL\IM-0117-
0001.jpeg
2               Downloads/archive/chest_xray\train/NORMAL\IM-0119-
0001.jpeg
3               Downloads/archive/chest_xray\train/NORMAL\IM-0122-
0001.jpeg
4               Downloads/archive/chest_xray\train/NORMAL\IM-0125-
0001.jpeg
...
```

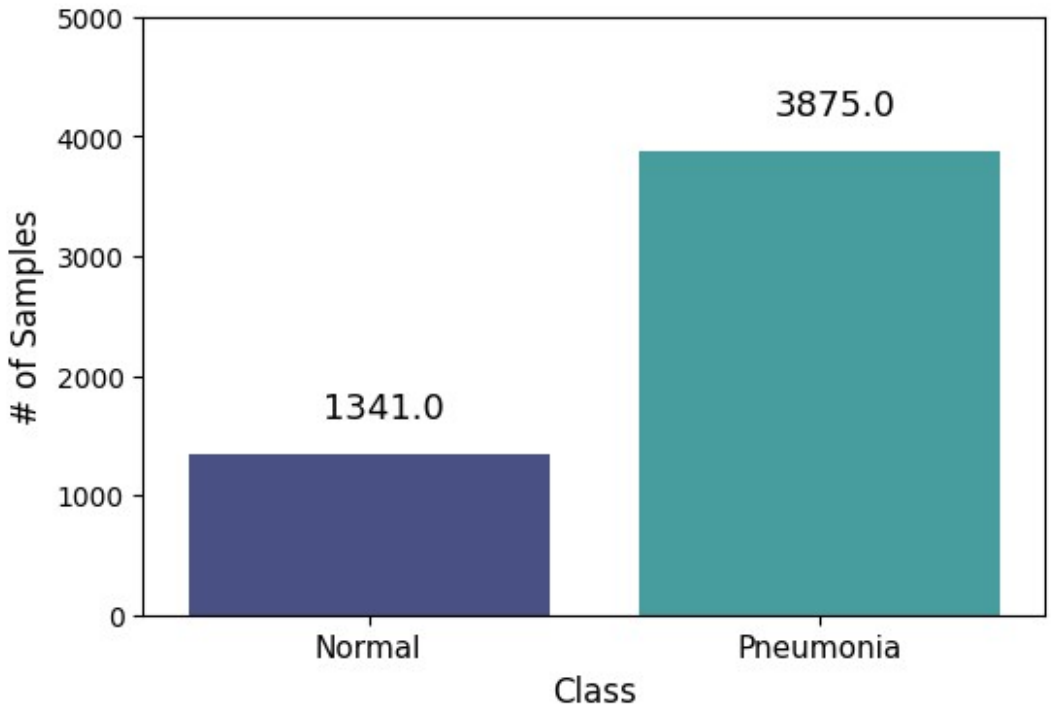
```
...
5211   Downloads/archive/chest_xray\train/PNEUMONIA\
person99_virus_183.jpeg
5212  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_38.jpeg
5213  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_39.jpeg
5214  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_40.jpeg
5215  Downloads/archive/chest_xray\train/PNEUMONIA\
person9_bacteria_41.jpeg

[5216 rows x 2 columns]

df_test

          class  \
0       Normal
1       Normal
2       Normal
3       Normal
4       Normal
..         ...
619  Pneumonia
620  Pneumonia
621  Pneumonia
622  Pneumonia
623  Pneumonia


image
0                Downloads/archive/chest_xray\test/NORMAL\IM-0001-
0001.jpeg
1                Downloads/archive/chest_xray\test/NORMAL\IM-0003-
0001.jpeg
2                Downloads/archive/chest_xray\test/NORMAL\IM-0005-
0001.jpeg
3                Downloads/archive/chest_xray\test/NORMAL\IM-0006-
0001.jpeg
4                Downloads/archive/chest_xray\test/NORMAL\IM-0007-
0001.jpeg
..
...
619  Downloads/archive/chest_xray\test/PNEUMONIA\
person96_bacteria_465.jpeg
620  Downloads/archive/chest_xray\test/PNEUMONIA\
person96_bacteria_466.jpeg
621  Downloads/archive/chest_xray\test/PNEUMONIA\
person97_bacteria_468.jpeg
622  Downloads/archive/chest_xray\test/PNEUMONIA\
```

```
person99_bacteria_473.jpeg
623  Downloads/archive/chest_xray\test/PNEUMONIA\
person99_bacteria_474.jpeg

[624 rows x 2 columns]
```

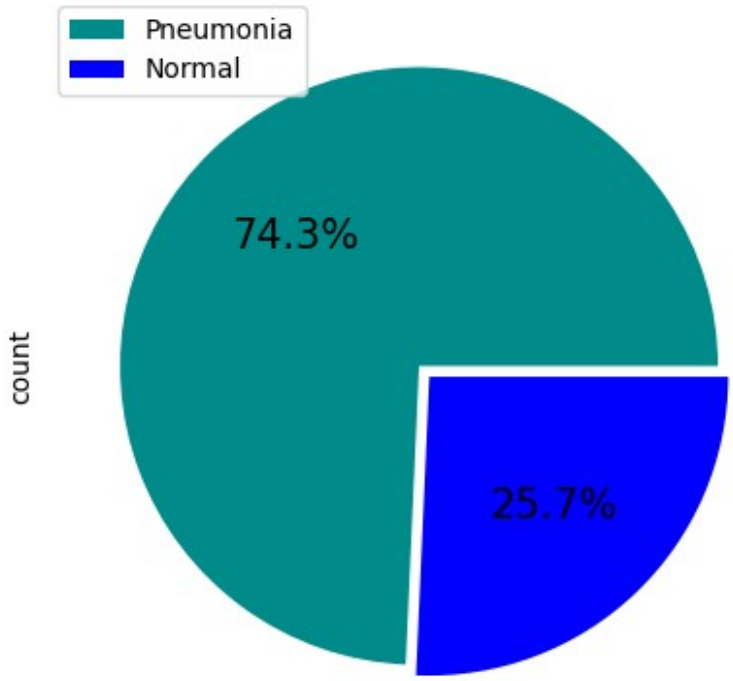
Exploring the Data

Let's check the target distribution on each set

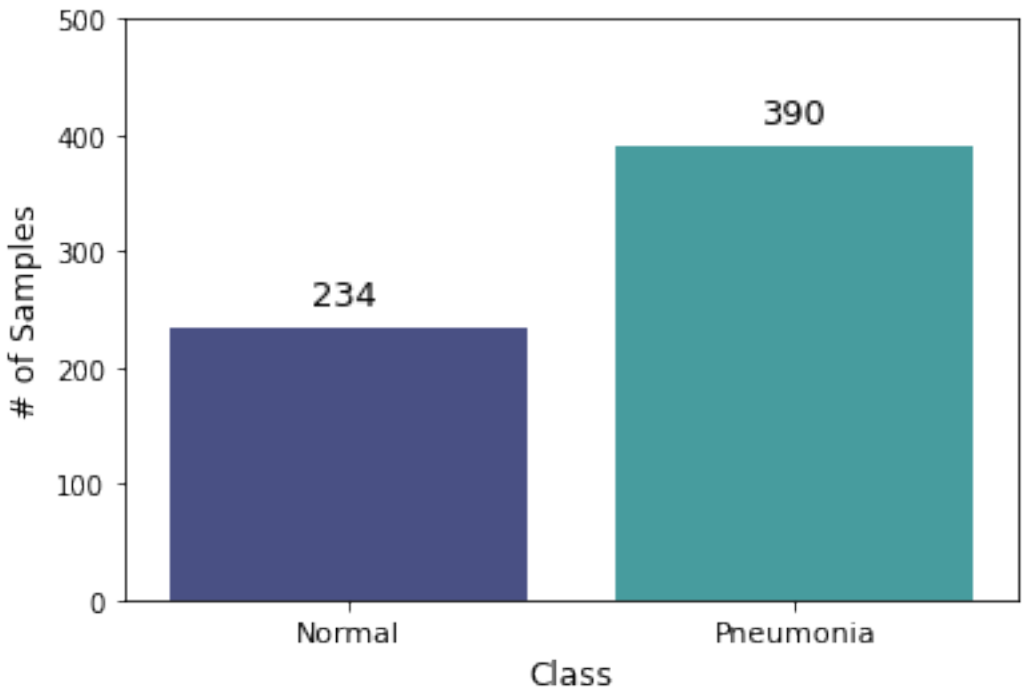
```
plt.figure(figsize=(6,4))

ax = sns.countplot(x='class', data=df_train, palette="mako")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,5000)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.30, p.get_height()
+300), fontsize = 13)

plt.show()
```



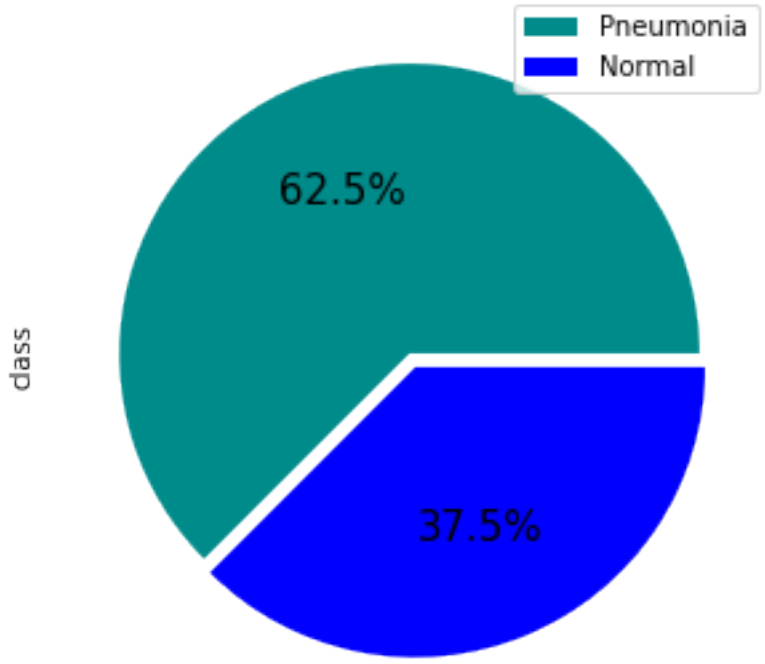
```
plt.figure(figsize=(7,5))

df_train['class'].value_counts().plot(kind='pie',labels = ['',''],
autopct='%1.1f%%', colors = ['darkcyan','blue'], explode = [0,0.05],
textprops = {"fontsize":15})

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```




```
plt.figure(figsize=(6,4))

ax = sns.countplot(x='class', data=df_test, palette="mako")

plt.xlabel("Class", fontsize= 12)
plt.ylabel("# of Samples", fontsize= 12)
plt.ylim(0,500)
plt.xticks([0,1], ['Normal', 'Pneumonia'], fontsize = 11)

for p in ax.patches:
    ax.annotate((p.get_height()), (p.get_x()+0.32, p.get_height()+20),
fontsize = 13)

plt.show()
```




```
plt.figure(figsize=(7,5))

df_test['class'].value_counts().plot(kind='pie',labels = ['',''],
autopct='%1.1f%%', colors = ['darkcyan','blue'], explode = [0,0.05],
textprops = {"fontsize":15})

plt.legend(labels=['Pneumonia', 'Normal'])
plt.show()
```

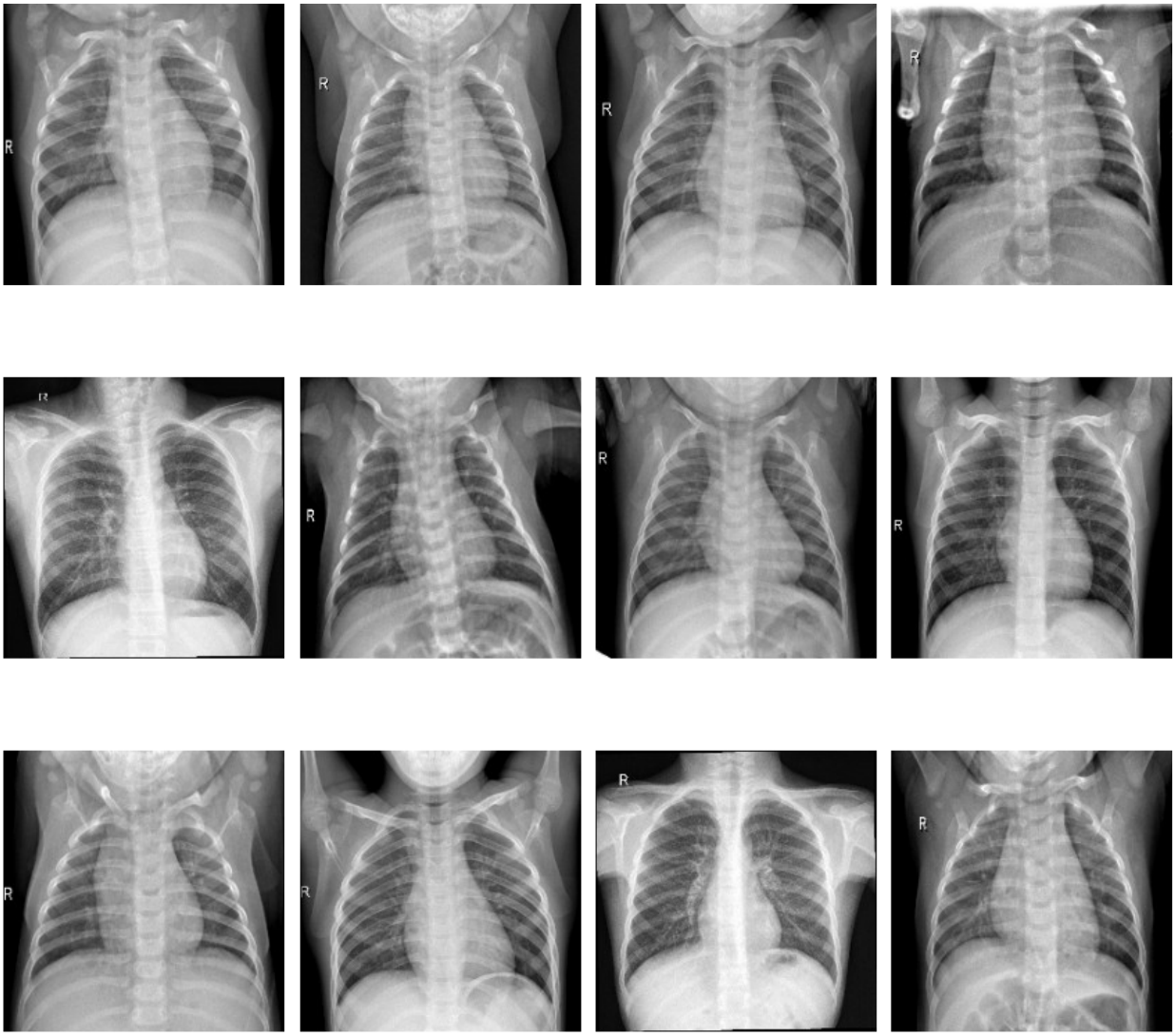


The distributions from these datasets are a little different from each other. Both are slightly imbalanced, having more samples from the positive class (Pneumonia), with the training set being a little more imbalanced.

Before we move on to the next section, we will take a look at a few examples from each dataset.

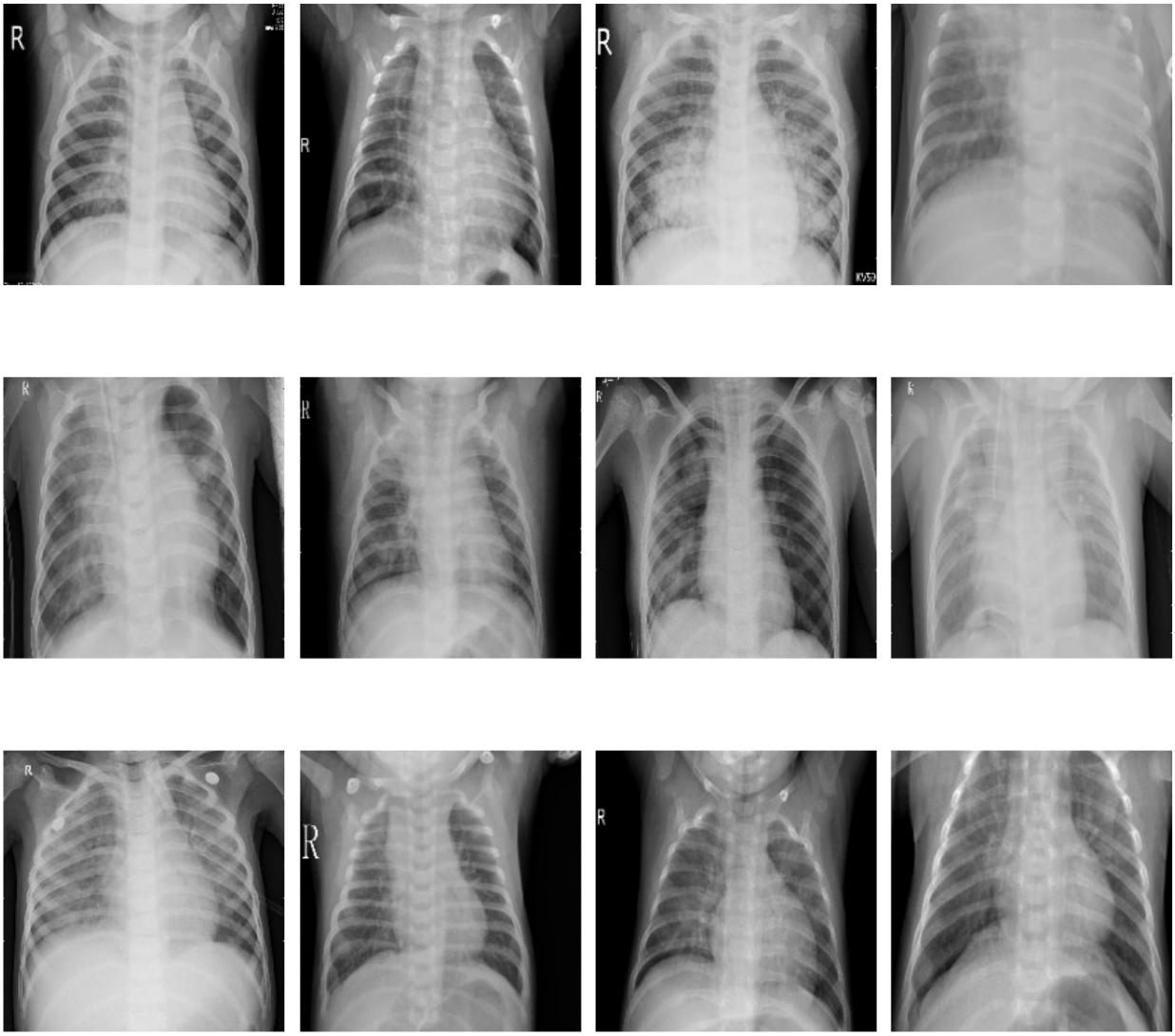
```
print('Train Set - Normal')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set - Normal
```



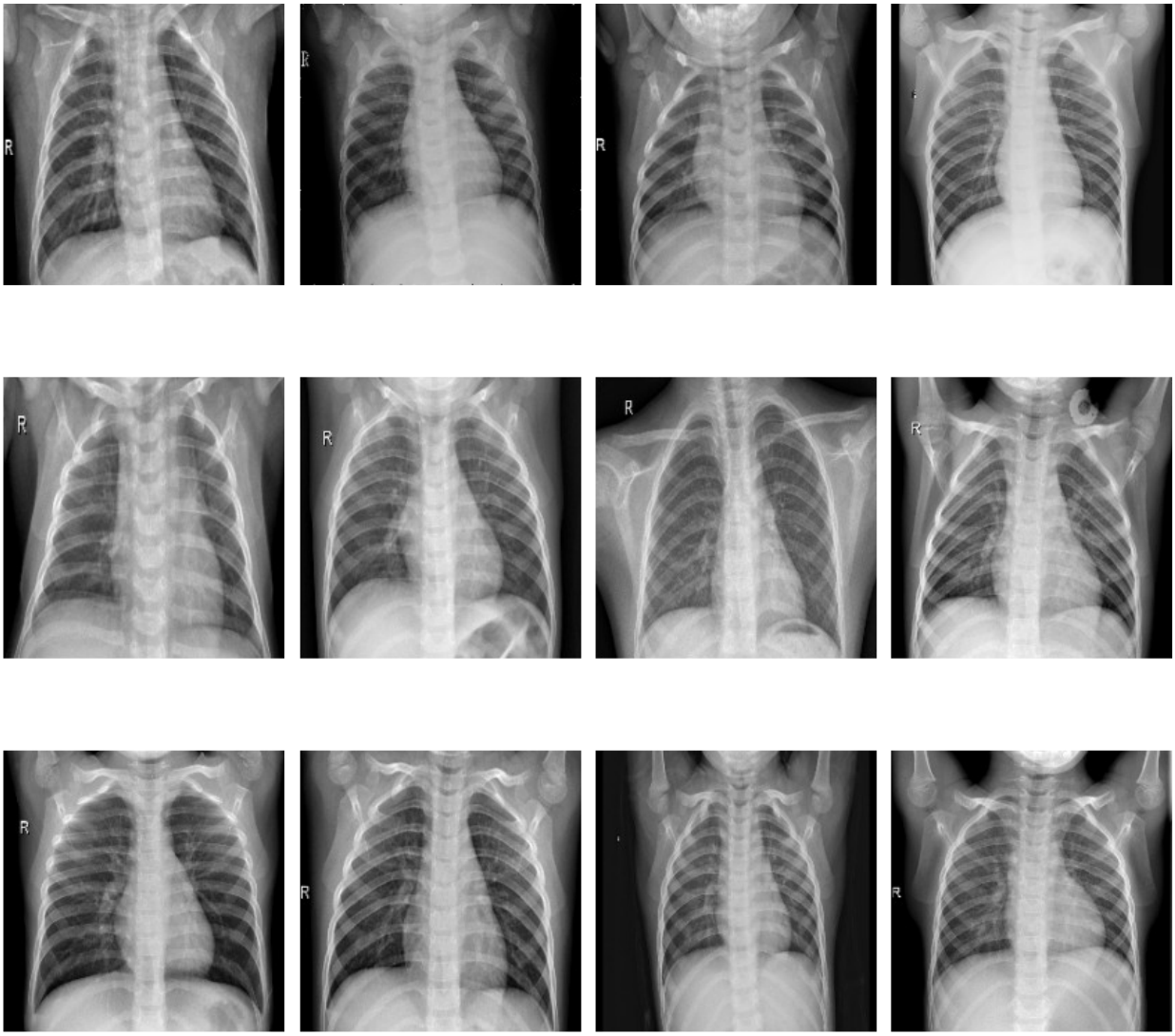
```
print('Train Set - Pneumonia')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(train_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Train Set - Pneumonia
```



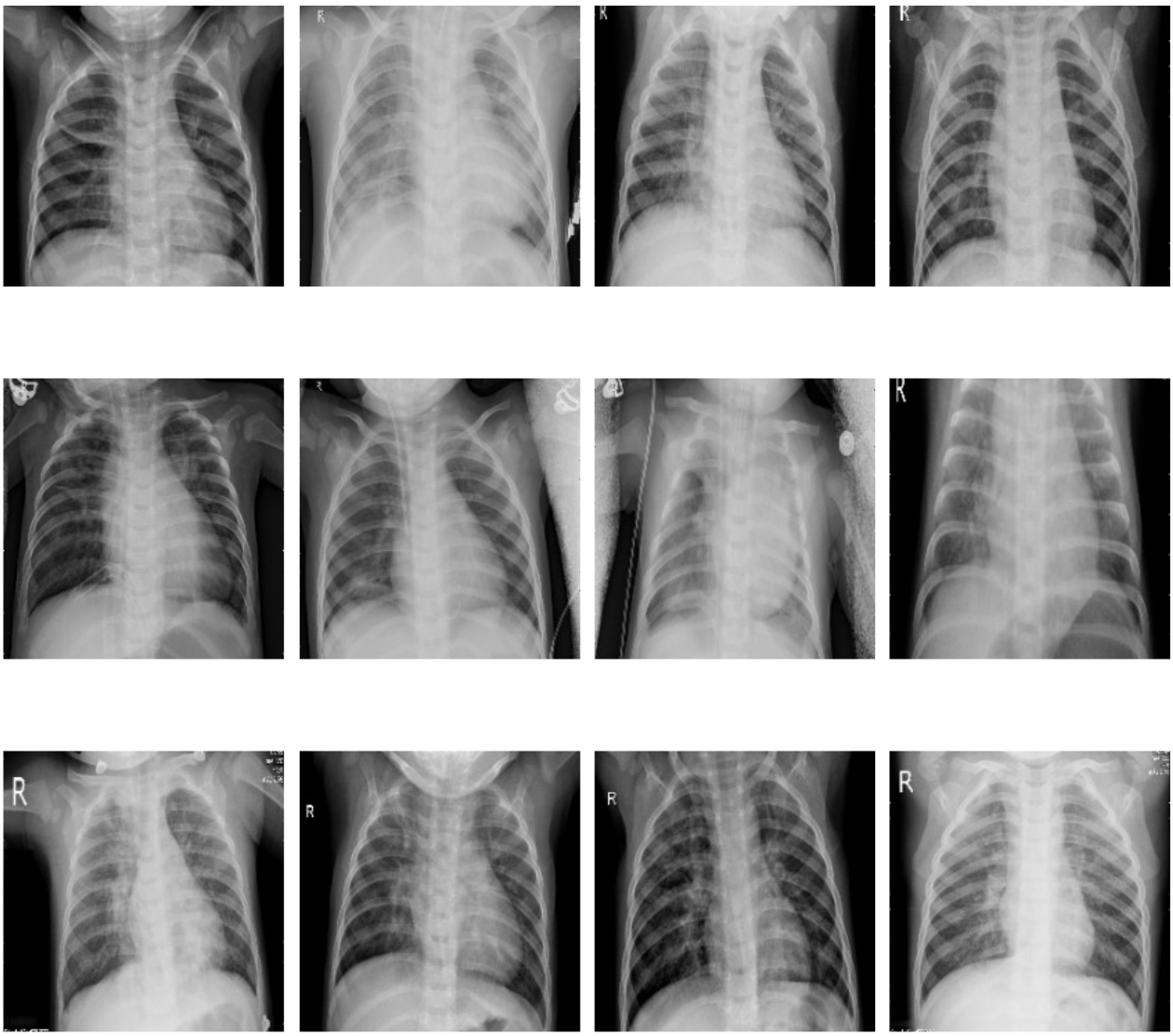
```
print('Test Set - Normal')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_normal[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set - Normal
```



```
print('Test Set - Pneumonia')

plt.figure(figsize=(12,12))

for i in range(0, 12):
    plt.subplot(3,4,i + 1)
    img = cv2.imread(test_pneumonia[i])
    img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
    plt.imshow(img)
    plt.axis("off")

plt.tight_layout()

plt.show()
```

```
Test Set - Pneumonia
```



Preparing the Data

First, we need to create a validation set. To do that, we apply a simple stratified split on the original train dataset, using 80% for actual training and 20% for validation purposes.

```
train_df, val_df = train_test_split(df_train, test_size = 0.20,
random_state = SEED, stratify = df_train['class'])

train_df
```

```
          class  \
3566  Pneumonia
2866  Pneumonia
2681  Pneumonia
1199     Normal
```

```
4619  Pneumonia
...         ...
3476  Pneumonia
678      Normal
1560  Pneumonia
2769  Pneumonia
4881  Pneumonia


image
3566
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
6559742-0002.jpeg
2866
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4609274-0001.jpeg
2681
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
1149864-0002.jpeg
1199
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
6315622-0002.jpeg
4619
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1753514-0002.jpeg
...
...
3476
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
3408906-0008.jpeg
678
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
4688588-0001.jpeg
1560
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-8467199-0001.jpeg
2769
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-6909155-0001.jpeg
4881
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1820776-0002.jpeg

[4185 rows x 2 columns]

val_df

          class  \
2945  Pneumonia
4878  Pneumonia
```

```
3177  Pneumonia
972      Normal
3059  Pneumonia
...          ...
253      Normal
4315  Pneumonia
687      Normal
3417  Pneumonia
3745  Pneumonia


image
2945
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-2474652-0008.jpeg
4878
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-2739133-0001.jpeg
3177
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4133607-0005.jpeg
972
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
3476904-0001.jpeg
3059
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/VIRUS-
8446214-0003.jpeg
...
...
253
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
9862823-0002.jpeg
4315
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-1083680-0014.jpeg
687
../input/labeled-chest-xray-images/chest_xray/train/NORMAL/NORMAL-
8991560-0001.jpeg
3417
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-7942103-0006.jpeg
3745
../input/labeled-chest-xray-images/chest_xray/train/PNEUMONIA/BACTERIA
-4638514-0001.jpeg

[1047 rows x 2 columns]
```

Now, we're going to load the images from the folders and prepare them to feed our models.

We begin by defining the data generators. With Keras Image Data Generator, we can rescale the pixel values and apply random transformation techniques for data augmentation on the fly. We define two different generators. The val_datagen is used to simply rescale the validation and test sets. The train_datagen includes some transformations to augment the train set.

We apply those generators on each dataset using the flow_from_dataframe method. Apart from the transformations defined in each generator, the images are also resized based on the target_size set.

```
# https://vijayabhaskar96.medium.com/tutorial-on-keras-flow-from-dataframe-1fd4493d237c

train_datagen = ImageDataGenerator(rescale=1/255.,
                                   zoom_range = 0.1,
                                   #rotation_range = 0.1,
                                   width_shift_range = 0.1,
                                   height_shift_range = 0.1)

val_datagen = ImageDataGenerator(rescale=1/255.)

ds_train = train_datagen.flow_from_dataframe(train_df,
                                             #directory=train_path,
#dataframe contains the full paths
                                             x_col = 'image',
                                             y_col = 'class',
                                             target_size = (IMG_SIZE,
IMG_SIZE),
                                             class_mode = 'binary',
                                             batch_size = BATCH,
                                             seed = SEED)

ds_val = val_datagen.flow_from_dataframe(val_df,
                                         #directory=train_path,
                                         x_col = 'image',
                                         y_col = 'class',
                                         target_size = (IMG_SIZE,
IMG_SIZE),
                                         class_mode = 'binary',
                                         batch_size = BATCH,
                                         seed = SEED)

ds_test = val_datagen.flow_from_dataframe(df_test,
                                         #directory=test_path,
                                         x_col = 'image',
                                         y_col = 'class',
                                         target_size = (IMG_SIZE,
IMG_SIZE),
                                         class_mode = 'binary',
                                         batch_size = 1,
                                         shuffle = False)
```

```
Found 4185 validated image filenames belonging to 2 classes.
Found 1047 validated image filenames belonging to 2 classes.
Found 624 validated image filenames belonging to 2 classes.
```

Now, we are ready for the next stage: creating and training the image classification models.

Transfer Learning

```
base_model = tf.keras.applications.ResNet152V2(
    weights='imagenet',
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False)

base_model.trainable = False

def get_pretrained():

    #Input shape = [width, height, color channels]
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

    x = base_model(inputs)

    # Head
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.1)(x)

    #Final Layer (Output)
    output = layers.Dense(1, activation='sigmoid')(x)

    model = keras.Model(inputs=[inputs], outputs=output)

    return model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/
resnet152v2_weights_tf_dim_ordering_tf_kernels_notop.h5
234553344/234545216 [==============================] - 1s 0us/step
```

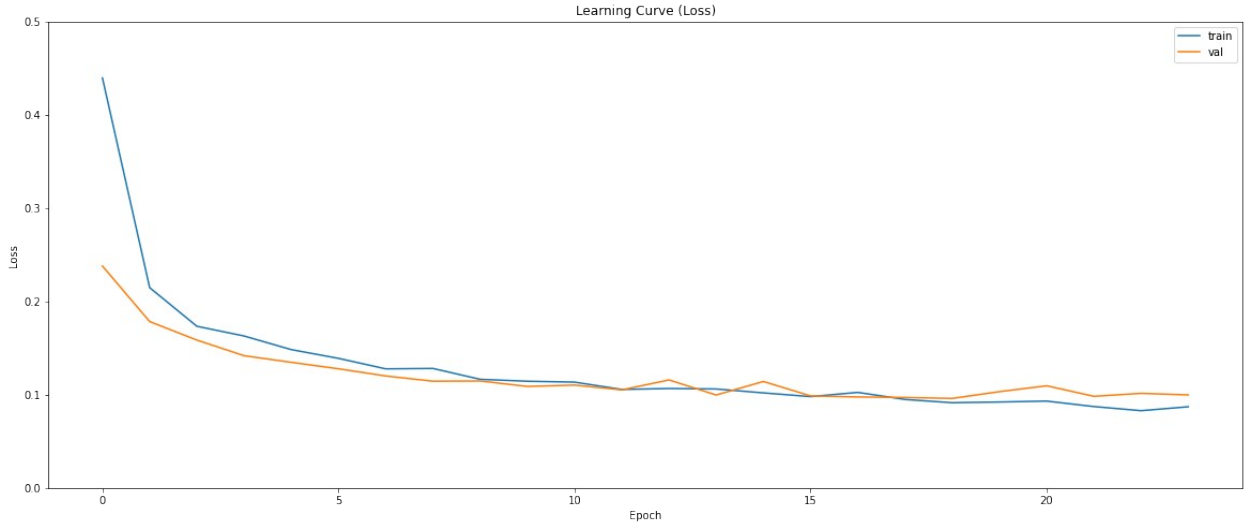
```
keras.backend.clear_session()

model_pretrained = get_pretrained()
model_pretrained.compile(loss='binary_crossentropy'
              , optimizer = keras.optimizers.Adam(learning_rate=5e-5),
metrics='binary_accuracy')

model_pretrained.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
resnet152v2 (Functional)     (None, 7, 7, 2048)        58331648
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 58,594,049
Trainable params: 262,401
Non-trainable params: 58,331,648
_________________________________________________________________

history = model_pretrained.fit(ds_train,
          batch_size = BATCH, epochs = 50,
          validation_data=ds_val,
          callbacks=[early_stopping, plateau],
          steps_per_epoch=(len(train_df)/BATCH),
          validation_steps=(len(val_df)/BATCH));

Epoch 1/50
130/130 [==============================] - 116s 823ms/step - loss:
0.6397 - binary_accuracy: 0.6697 - val_loss: 0.2379 -
val_binary_accuracy: 0.9179
Epoch 2/50
130/130 [==============================] - 103s 791ms/step - loss:
0.2347 - binary_accuracy: 0.9156 - val_loss: 0.1784 -
val_binary_accuracy: 0.9379
Epoch 3/50
130/130 [==============================] - 103s 791ms/step - loss:
0.1833 - binary_accuracy: 0.9363 - val_loss: 0.1586 -
val_binary_accuracy: 0.9475
Epoch 4/50
130/130 [==============================] - 105s 803ms/step - loss:
0.1761 - binary_accuracy: 0.9292 - val_loss: 0.1418 -
val_binary_accuracy: 0.9522
Epoch 5/50
130/130 [==============================] - 105s 803ms/step - loss:
0.1471 - binary_accuracy: 0.9466 - val_loss: 0.1347 -
val_binary_accuracy: 0.9561
Epoch 6/50
130/130 [==============================] - 103s 790ms/step - loss:
```

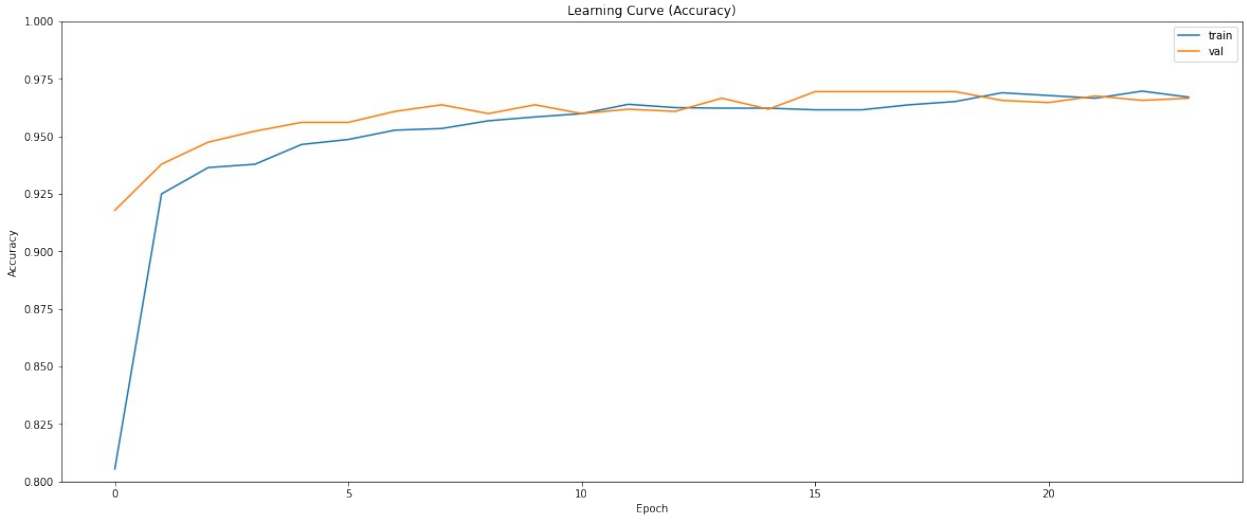
```
0.1403 - binary_accuracy: 0.9473 - val_loss: 0.1278 -
val_binary_accuracy: 0.9561
Epoch 7/50
130/130 [==============================] - 104s 797ms/step - loss:
0.1185 - binary_accuracy: 0.9540 - val_loss: 0.1201 -
val_binary_accuracy: 0.9608
Epoch 8/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1305 - binary_accuracy: 0.9502 - val_loss: 0.1145 -
val_binary_accuracy: 0.9637
Epoch 9/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1094 - binary_accuracy: 0.9569 - val_loss: 0.1148 -
val_binary_accuracy: 0.9599
Epoch 10/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1166 - binary_accuracy: 0.9581 - val_loss: 0.1089 -
val_binary_accuracy: 0.9637
Epoch 11/50
130/130 [==============================] - 103s 791ms/step - loss:
0.1203 - binary_accuracy: 0.9584 - val_loss: 0.1103 -
val_binary_accuracy: 0.9599
Epoch 12/50
130/130 [==============================] - 103s 791ms/step - loss:
0.1107 - binary_accuracy: 0.9628 - val_loss: 0.1052 -
val_binary_accuracy: 0.9618
Epoch 13/50
130/130 [==============================] - 104s 793ms/step - loss:
0.1032 - binary_accuracy: 0.9651 - val_loss: 0.1158 -
val_binary_accuracy: 0.9608
Epoch 14/50
130/130 [==============================] - 104s 793ms/step - loss:
0.1211 - binary_accuracy: 0.9558 - val_loss: 0.0997 -
val_binary_accuracy: 0.9666
Epoch 15/50
130/130 [==============================] - 102s 780ms/step - loss:
0.1006 - binary_accuracy: 0.9642 - val_loss: 0.1142 -
val_binary_accuracy: 0.9618
Epoch 16/50
130/130 [==============================] - 102s 782ms/step - loss:
0.0930 - binary_accuracy: 0.9648 - val_loss: 0.0987 -
val_binary_accuracy: 0.9694
Epoch 17/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1035 - binary_accuracy: 0.9608 - val_loss: 0.0976 -
val_binary_accuracy: 0.9694
Epoch 18/50
130/130 [==============================] - 102s 782ms/step - loss:
0.0897 - binary_accuracy: 0.9640 - val_loss: 0.0971 -
```

```
val_binary_accuracy: 0.9694
Epoch 19/50
130/130 [==============================] - 104s 792ms/step - loss:
0.0979 - binary_accuracy: 0.9607 - val_loss: 0.0960 -
val_binary_accuracy: 0.9694
Epoch 20/50
130/130 [==============================] - 105s 802ms/step - loss:
0.1003 - binary_accuracy: 0.9697 - val_loss: 0.1033 -
val_binary_accuracy: 0.9656
Epoch 21/50
130/130 [==============================] - 105s 802ms/step - loss:
0.0948 - binary_accuracy: 0.9664 - val_loss: 0.1096 -
val_binary_accuracy: 0.9647

Epoch 00021: ReduceLROnPlateau reducing learning rate to
9.999999747378752e-06.
Epoch 22/50
130/130 [==============================] - 102s 778ms/step - loss:
0.0757 - binary_accuracy: 0.9710 - val_loss: 0.0983 -
val_binary_accuracy: 0.9675
Epoch 23/50
130/130 [==============================] - 102s 779ms/step - loss:
0.0776 - binary_accuracy: 0.9732 - val_loss: 0.1015 -
val_binary_accuracy: 0.9656

Epoch 00023: ReduceLROnPlateau reducing learning rate to
1.9999999494757505e-06.
Epoch 24/50
130/130 [==============================] - 103s 788ms/step - loss:
0.0920 - binary_accuracy: 0.9631 - val_loss: 0.0998 -
val_binary_accuracy: 0.9666
```

```
fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y = history.history['loss'])
sns.lineplot(x = history.epoch, y = history.history['val_loss'])
ax.set_title('Learning Curve (Loss)')
ax.set_ylabel('Loss')
ax.set_xlabel('Epoch')
ax.set_ylim(0, 0.5)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y =
history.history['binary_accuracy'])
sns.lineplot(x = history.epoch, y =
history.history['val_binary_accuracy'])
ax.set_title('Learning Curve (Accuracy)')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylim(0.80, 1.0)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
score = model_pretrained.evaluate(ds_val, steps = len(val_df)/BATCH,
verbose = 0)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
Val loss: 0.09604005515575409
Val accuracy: 0.9694364666938782
```

```
score = model_pretrained.evaluate(ds_test, steps = len(df_test),
verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.28737810254096985
Test accuracy: 0.8814102411270142
```

Fine Tuning

Our last approach is called Fine Tuning. In the last section, all the layers from the pretrained model were 'frozen', preserving the weights calculated during its training on the ImageNet dataset. Now, we are going to unfreeze a few of its last layers and continue the training, tuning the weights from these layers according to our dataset.

```
base_model.trainable = True

# Freeze all layers except for the
for layer in base_model.layers[:-13]:
    layer.trainable = False

# Check which layers are tuneable (trainable)
for layer_number, layer in enumerate(base_model.layers):
    print(layer_number, layer.name, layer.trainable)
```

```
0 input_2 False
1 conv1_pad False
2 conv1_conv False
3 pool1_pad False
4 pool1_pool False
5 conv2_block1_preact_bn False
6 conv2_block1_preact_relu False
7 conv2_block1_1_conv False
8 conv2_block1_1_bn False
9 conv2_block1_1_relu False
10 conv2_block1_2_pad False
11 conv2_block1_2_conv False
12 conv2_block1_2_bn False
13 conv2_block1_2_relu False
14 conv2_block1_0_conv False
15 conv2_block1_3_conv False
16 conv2_block1_out False
17 conv2_block2_preact_bn False
18 conv2_block2_preact_relu False
19 conv2_block2_1_conv False
```

```
20 conv2_block2_1_bn False
21 conv2_block2_1_relu False
22 conv2_block2_2_pad False
23 conv2_block2_2_conv False
24 conv2_block2_2_bn False
25 conv2_block2_2_relu False
26 conv2_block2_3_conv False
27 conv2_block2_out False
28 conv2_block3_preact_bn False
29 conv2_block3_preact_relu False
30 conv2_block3_1_conv False
31 conv2_block3_1_bn False
32 conv2_block3_1_relu False
33 conv2_block3_2_pad False
34 conv2_block3_2_conv False
35 conv2_block3_2_bn False
36 conv2_block3_2_relu False
37 max_pooling2d_3 False
38 conv2_block3_3_conv False
39 conv2_block3_out False
40 conv3_block1_preact_bn False
41 conv3_block1_preact_relu False
42 conv3_block1_1_conv False
43 conv3_block1_1_bn False
44 conv3_block1_1_relu False
45 conv3_block1_2_pad False
46 conv3_block1_2_conv False
47 conv3_block1_2_bn False
48 conv3_block1_2_relu False
49 conv3_block1_0_conv False
50 conv3_block1_3_conv False
51 conv3_block1_out False
52 conv3_block2_preact_bn False
53 conv3_block2_preact_relu False
54 conv3_block2_1_conv False
55 conv3_block2_1_bn False
56 conv3_block2_1_relu False
57 conv3_block2_2_pad False
58 conv3_block2_2_conv False
59 conv3_block2_2_bn False
60 conv3_block2_2_relu False
61 conv3_block2_3_conv False
62 conv3_block2_out False
63 conv3_block3_preact_bn False
64 conv3_block3_preact_relu False
65 conv3_block3_1_conv False
66 conv3_block3_1_bn False
67 conv3_block3_1_relu False
68 conv3_block3_2_pad False
```

```
69 conv3_block3_2_conv False
70 conv3_block3_2_bn False
71 conv3_block3_2_relu False
72 conv3_block3_3_conv False
73 conv3_block3_out False
74 conv3_block4_preact_bn False
75 conv3_block4_preact_relu False
76 conv3_block4_1_conv False
77 conv3_block4_1_bn False
78 conv3_block4_1_relu False
79 conv3_block4_2_pad False
80 conv3_block4_2_conv False
81 conv3_block4_2_bn False
82 conv3_block4_2_relu False
83 conv3_block4_3_conv False
84 conv3_block4_out False
85 conv3_block5_preact_bn False
86 conv3_block5_preact_relu False
87 conv3_block5_1_conv False
88 conv3_block5_1_bn False
89 conv3_block5_1_relu False
90 conv3_block5_2_pad False
91 conv3_block5_2_conv False
92 conv3_block5_2_bn False
93 conv3_block5_2_relu False
94 conv3_block5_3_conv False
95 conv3_block5_out False
96 conv3_block6_preact_bn False
97 conv3_block6_preact_relu False
98 conv3_block6_1_conv False
99 conv3_block6_1_bn False
100 conv3_block6_1_relu False
101 conv3_block6_2_pad False
102 conv3_block6_2_conv False
103 conv3_block6_2_bn False
104 conv3_block6_2_relu False
105 conv3_block6_3_conv False
106 conv3_block6_out False
107 conv3_block7_preact_bn False
108 conv3_block7_preact_relu False
109 conv3_block7_1_conv False
110 conv3_block7_1_bn False
111 conv3_block7_1_relu False
112 conv3_block7_2_pad False
113 conv3_block7_2_conv False
114 conv3_block7_2_bn False
115 conv3_block7_2_relu False
116 conv3_block7_3_conv False
117 conv3_block7_out False
```

```
118 conv3_block8_preact_bn False
119 conv3_block8_preact_relu False
120 conv3_block8_1_conv False
121 conv3_block8_1_bn False
122 conv3_block8_1_relu False
123 conv3_block8_2_pad False
124 conv3_block8_2_conv False
125 conv3_block8_2_bn False
126 conv3_block8_2_relu False
127 max_pooling2d_4 False
128 conv3_block8_3_conv False
129 conv3_block8_out False
130 conv4_block1_preact_bn False
131 conv4_block1_preact_relu False
132 conv4_block1_1_conv False
133 conv4_block1_1_bn False
134 conv4_block1_1_relu False
135 conv4_block1_2_pad False
136 conv4_block1_2_conv False
137 conv4_block1_2_bn False
138 conv4_block1_2_relu False
139 conv4_block1_0_conv False
140 conv4_block1_3_conv False
141 conv4_block1_out False
142 conv4_block2_preact_bn False
143 conv4_block2_preact_relu False
144 conv4_block2_1_conv False
145 conv4_block2_1_bn False
146 conv4_block2_1_relu False
147 conv4_block2_2_pad False
148 conv4_block2_2_conv False
149 conv4_block2_2_bn False
150 conv4_block2_2_relu False
151 conv4_block2_3_conv False
152 conv4_block2_out False
153 conv4_block3_preact_bn False
154 conv4_block3_preact_relu False
155 conv4_block3_1_conv False
156 conv4_block3_1_bn False
157 conv4_block3_1_relu False
158 conv4_block3_2_pad False
159 conv4_block3_2_conv False
160 conv4_block3_2_bn False
161 conv4_block3_2_relu False
162 conv4_block3_3_conv False
163 conv4_block3_out False
164 conv4_block4_preact_bn False
165 conv4_block4_preact_relu False
166 conv4_block4_1_conv False
```

```
167 conv4_block4_1_bn False
168 conv4_block4_1_relu False
169 conv4_block4_2_pad False
170 conv4_block4_2_conv False
171 conv4_block4_2_bn False
172 conv4_block4_2_relu False
173 conv4_block4_3_conv False
174 conv4_block4_out False
175 conv4_block5_preact_bn False
176 conv4_block5_preact_relu False
177 conv4_block5_1_conv False
178 conv4_block5_1_bn False
179 conv4_block5_1_relu False
180 conv4_block5_2_pad False
181 conv4_block5_2_conv False
182 conv4_block5_2_bn False
183 conv4_block5_2_relu False
184 conv4_block5_3_conv False
185 conv4_block5_out False
186 conv4_block6_preact_bn False
187 conv4_block6_preact_relu False
188 conv4_block6_1_conv False
189 conv4_block6_1_bn False
190 conv4_block6_1_relu False
191 conv4_block6_2_pad False
192 conv4_block6_2_conv False
193 conv4_block6_2_bn False
194 conv4_block6_2_relu False
195 conv4_block6_3_conv False
196 conv4_block6_out False
197 conv4_block7_preact_bn False
198 conv4_block7_preact_relu False
199 conv4_block7_1_conv False
200 conv4_block7_1_bn False
201 conv4_block7_1_relu False
202 conv4_block7_2_pad False
203 conv4_block7_2_conv False
204 conv4_block7_2_bn False
205 conv4_block7_2_relu False
206 conv4_block7_3_conv False
207 conv4_block7_out False
208 conv4_block8_preact_bn False
209 conv4_block8_preact_relu False
210 conv4_block8_1_conv False
211 conv4_block8_1_bn False
212 conv4_block8_1_relu False
213 conv4_block8_2_pad False
214 conv4_block8_2_conv False
215 conv4_block8_2_bn False
```

```
216 conv4_block8_2_relu False
217 conv4_block8_3_conv False
218 conv4_block8_out False
219 conv4_block9_preact_bn False
220 conv4_block9_preact_relu False
221 conv4_block9_1_conv False
222 conv4_block9_1_bn False
223 conv4_block9_1_relu False
224 conv4_block9_2_pad False
225 conv4_block9_2_conv False
226 conv4_block9_2_bn False
227 conv4_block9_2_relu False
228 conv4_block9_3_conv False
229 conv4_block9_out False
230 conv4_block10_preact_bn False
231 conv4_block10_preact_relu False
232 conv4_block10_1_conv False
233 conv4_block10_1_bn False
234 conv4_block10_1_relu False
235 conv4_block10_2_pad False
236 conv4_block10_2_conv False
237 conv4_block10_2_bn False
238 conv4_block10_2_relu False
239 conv4_block10_3_conv False
240 conv4_block10_out False
241 conv4_block11_preact_bn False
242 conv4_block11_preact_relu False
243 conv4_block11_1_conv False
244 conv4_block11_1_bn False
245 conv4_block11_1_relu False
246 conv4_block11_2_pad False
247 conv4_block11_2_conv False
248 conv4_block11_2_bn False
249 conv4_block11_2_relu False
250 conv4_block11_3_conv False
251 conv4_block11_out False
252 conv4_block12_preact_bn False
253 conv4_block12_preact_relu False
254 conv4_block12_1_conv False
255 conv4_block12_1_bn False
256 conv4_block12_1_relu False
257 conv4_block12_2_pad False
258 conv4_block12_2_conv False
259 conv4_block12_2_bn False
260 conv4_block12_2_relu False
261 conv4_block12_3_conv False
262 conv4_block12_out False
263 conv4_block13_preact_bn False
264 conv4_block13_preact_relu False
```

```
265 conv4_block13_1_conv False
266 conv4_block13_1_bn False
267 conv4_block13_1_relu False
268 conv4_block13_2_pad False
269 conv4_block13_2_conv False
270 conv4_block13_2_bn False
271 conv4_block13_2_relu False
272 conv4_block13_3_conv False
273 conv4_block13_out False
274 conv4_block14_preact_bn False
275 conv4_block14_preact_relu False
276 conv4_block14_1_conv False
277 conv4_block14_1_bn False
278 conv4_block14_1_relu False
279 conv4_block14_2_pad False
280 conv4_block14_2_conv False
281 conv4_block14_2_bn False
282 conv4_block14_2_relu False
283 conv4_block14_3_conv False
284 conv4_block14_out False
285 conv4_block15_preact_bn False
286 conv4_block15_preact_relu False
287 conv4_block15_1_conv False
288 conv4_block15_1_bn False
289 conv4_block15_1_relu False
290 conv4_block15_2_pad False
291 conv4_block15_2_conv False
292 conv4_block15_2_bn False
293 conv4_block15_2_relu False
294 conv4_block15_3_conv False
295 conv4_block15_out False
296 conv4_block16_preact_bn False
297 conv4_block16_preact_relu False
298 conv4_block16_1_conv False
299 conv4_block16_1_bn False
300 conv4_block16_1_relu False
301 conv4_block16_2_pad False
302 conv4_block16_2_conv False
303 conv4_block16_2_bn False
304 conv4_block16_2_relu False
305 conv4_block16_3_conv False
306 conv4_block16_out False
307 conv4_block17_preact_bn False
308 conv4_block17_preact_relu False
309 conv4_block17_1_conv False
310 conv4_block17_1_bn False
311 conv4_block17_1_relu False
312 conv4_block17_2_pad False
313 conv4_block17_2_conv False
```

```
314 conv4_block17_2_bn False
315 conv4_block17_2_relu False
316 conv4_block17_3_conv False
317 conv4_block17_out False
318 conv4_block18_preact_bn False
319 conv4_block18_preact_relu False
320 conv4_block18_1_conv False
321 conv4_block18_1_bn False
322 conv4_block18_1_relu False
323 conv4_block18_2_pad False
324 conv4_block18_2_conv False
325 conv4_block18_2_bn False
326 conv4_block18_2_relu False
327 conv4_block18_3_conv False
328 conv4_block18_out False
329 conv4_block19_preact_bn False
330 conv4_block19_preact_relu False
331 conv4_block19_1_conv False
332 conv4_block19_1_bn False
333 conv4_block19_1_relu False
334 conv4_block19_2_pad False
335 conv4_block19_2_conv False
336 conv4_block19_2_bn False
337 conv4_block19_2_relu False
338 conv4_block19_3_conv False
339 conv4_block19_out False
340 conv4_block20_preact_bn False
341 conv4_block20_preact_relu False
342 conv4_block20_1_conv False
343 conv4_block20_1_bn False
344 conv4_block20_1_relu False
345 conv4_block20_2_pad False
346 conv4_block20_2_conv False
347 conv4_block20_2_bn False
348 conv4_block20_2_relu False
349 conv4_block20_3_conv False
350 conv4_block20_out False
351 conv4_block21_preact_bn False
352 conv4_block21_preact_relu False
353 conv4_block21_1_conv False
354 conv4_block21_1_bn False
355 conv4_block21_1_relu False
356 conv4_block21_2_pad False
357 conv4_block21_2_conv False
358 conv4_block21_2_bn False
359 conv4_block21_2_relu False
360 conv4_block21_3_conv False
361 conv4_block21_out False
362 conv4_block22_preact_bn False
```

```
363 conv4_block22_preact_relu False
364 conv4_block22_1_conv False
365 conv4_block22_1_bn False
366 conv4_block22_1_relu False
367 conv4_block22_2_pad False
368 conv4_block22_2_conv False
369 conv4_block22_2_bn False
370 conv4_block22_2_relu False
371 conv4_block22_3_conv False
372 conv4_block22_out False
373 conv4_block23_preact_bn False
374 conv4_block23_preact_relu False
375 conv4_block23_1_conv False
376 conv4_block23_1_bn False
377 conv4_block23_1_relu False
378 conv4_block23_2_pad False
379 conv4_block23_2_conv False
380 conv4_block23_2_bn False
381 conv4_block23_2_relu False
382 conv4_block23_3_conv False
383 conv4_block23_out False
384 conv4_block24_preact_bn False
385 conv4_block24_preact_relu False
386 conv4_block24_1_conv False
387 conv4_block24_1_bn False
388 conv4_block24_1_relu False
389 conv4_block24_2_pad False
390 conv4_block24_2_conv False
391 conv4_block24_2_bn False
392 conv4_block24_2_relu False
393 conv4_block24_3_conv False
394 conv4_block24_out False
395 conv4_block25_preact_bn False
396 conv4_block25_preact_relu False
397 conv4_block25_1_conv False
398 conv4_block25_1_bn False
399 conv4_block25_1_relu False
400 conv4_block25_2_pad False
401 conv4_block25_2_conv False
402 conv4_block25_2_bn False
403 conv4_block25_2_relu False
404 conv4_block25_3_conv False
405 conv4_block25_out False
406 conv4_block26_preact_bn False
407 conv4_block26_preact_relu False
408 conv4_block26_1_conv False
409 conv4_block26_1_bn False
410 conv4_block26_1_relu False
411 conv4_block26_2_pad False
```

```
412 conv4_block26_2_conv False
413 conv4_block26_2_bn False
414 conv4_block26_2_relu False
415 conv4_block26_3_conv False
416 conv4_block26_out False
417 conv4_block27_preact_bn False
418 conv4_block27_preact_relu False
419 conv4_block27_1_conv False
420 conv4_block27_1_bn False
421 conv4_block27_1_relu False
422 conv4_block27_2_pad False
423 conv4_block27_2_conv False
424 conv4_block27_2_bn False
425 conv4_block27_2_relu False
426 conv4_block27_3_conv False
427 conv4_block27_out False
428 conv4_block28_preact_bn False
429 conv4_block28_preact_relu False
430 conv4_block28_1_conv False
431 conv4_block28_1_bn False
432 conv4_block28_1_relu False
433 conv4_block28_2_pad False
434 conv4_block28_2_conv False
435 conv4_block28_2_bn False
436 conv4_block28_2_relu False
437 conv4_block28_3_conv False
438 conv4_block28_out False
439 conv4_block29_preact_bn False
440 conv4_block29_preact_relu False
441 conv4_block29_1_conv False
442 conv4_block29_1_bn False
443 conv4_block29_1_relu False
444 conv4_block29_2_pad False
445 conv4_block29_2_conv False
446 conv4_block29_2_bn False
447 conv4_block29_2_relu False
448 conv4_block29_3_conv False
449 conv4_block29_out False
450 conv4_block30_preact_bn False
451 conv4_block30_preact_relu False
452 conv4_block30_1_conv False
453 conv4_block30_1_bn False
454 conv4_block30_1_relu False
455 conv4_block30_2_pad False
456 conv4_block30_2_conv False
457 conv4_block30_2_bn False
458 conv4_block30_2_relu False
459 conv4_block30_3_conv False
460 conv4_block30_out False
```

```
461 conv4_block31_preact_bn False
462 conv4_block31_preact_relu False
463 conv4_block31_1_conv False
464 conv4_block31_1_bn False
465 conv4_block31_1_relu False
466 conv4_block31_2_pad False
467 conv4_block31_2_conv False
468 conv4_block31_2_bn False
469 conv4_block31_2_relu False
470 conv4_block31_3_conv False
471 conv4_block31_out False
472 conv4_block32_preact_bn False
473 conv4_block32_preact_relu False
474 conv4_block32_1_conv False
475 conv4_block32_1_bn False
476 conv4_block32_1_relu False
477 conv4_block32_2_pad False
478 conv4_block32_2_conv False
479 conv4_block32_2_bn False
480 conv4_block32_2_relu False
481 conv4_block32_3_conv False
482 conv4_block32_out False
483 conv4_block33_preact_bn False
484 conv4_block33_preact_relu False
485 conv4_block33_1_conv False
486 conv4_block33_1_bn False
487 conv4_block33_1_relu False
488 conv4_block33_2_pad False
489 conv4_block33_2_conv False
490 conv4_block33_2_bn False
491 conv4_block33_2_relu False
492 conv4_block33_3_conv False
493 conv4_block33_out False
494 conv4_block34_preact_bn False
495 conv4_block34_preact_relu False
496 conv4_block34_1_conv False
497 conv4_block34_1_bn False
498 conv4_block34_1_relu False
499 conv4_block34_2_pad False
500 conv4_block34_2_conv False
501 conv4_block34_2_bn False
502 conv4_block34_2_relu False
503 conv4_block34_3_conv False
504 conv4_block34_out False
505 conv4_block35_preact_bn False
506 conv4_block35_preact_relu False
507 conv4_block35_1_conv False
508 conv4_block35_1_bn False
509 conv4_block35_1_relu False
510 conv4_block35_2_pad False
```

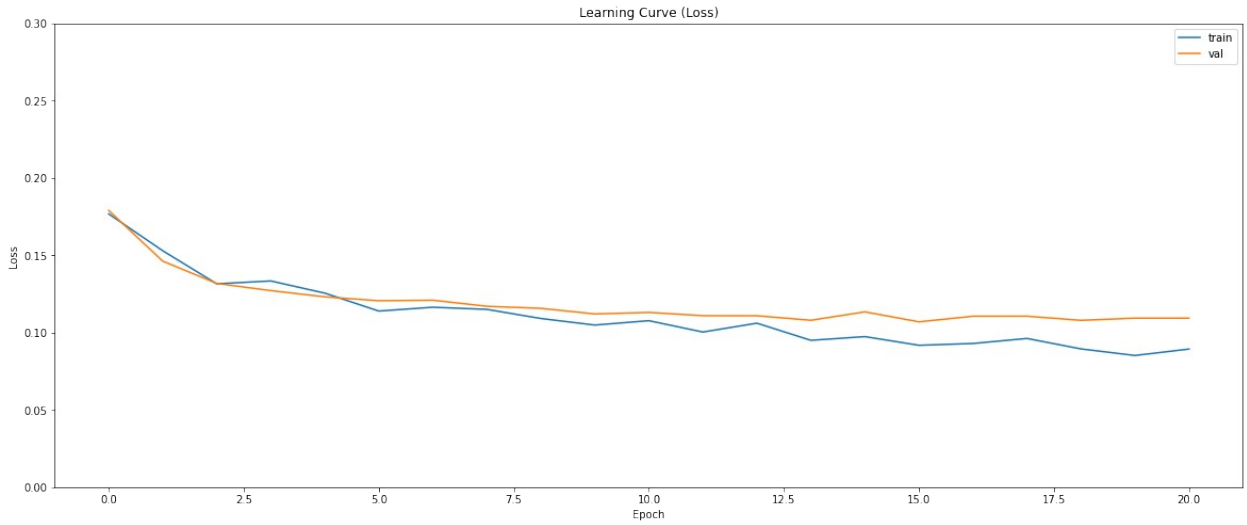
```
511 conv4_block35_2_conv False
512 conv4_block35_2_bn False
513 conv4_block35_2_relu False
514 conv4_block35_3_conv False
515 conv4_block35_out False
516 conv4_block36_preact_bn False
517 conv4_block36_preact_relu False
518 conv4_block36_1_conv False
519 conv4_block36_1_bn False
520 conv4_block36_1_relu False
521 conv4_block36_2_pad False
522 conv4_block36_2_conv False
523 conv4_block36_2_bn False
524 conv4_block36_2_relu False
525 max_pooling2d_5 False
526 conv4_block36_3_conv False
527 conv4_block36_out False
528 conv5_block1_preact_bn False
529 conv5_block1_preact_relu False
530 conv5_block1_1_conv False
531 conv5_block1_1_bn False
532 conv5_block1_1_relu False
533 conv5_block1_2_pad False
534 conv5_block1_2_conv False
535 conv5_block1_2_bn False
536 conv5_block1_2_relu False
537 conv5_block1_0_conv False
538 conv5_block1_3_conv False
539 conv5_block1_out False
540 conv5_block2_preact_bn False
541 conv5_block2_preact_relu False
542 conv5_block2_1_conv False
543 conv5_block2_1_bn False
544 conv5_block2_1_relu False
545 conv5_block2_2_pad False
546 conv5_block2_2_conv False
547 conv5_block2_2_bn False
548 conv5_block2_2_relu False
549 conv5_block2_3_conv False
550 conv5_block2_out False
551 conv5_block3_preact_bn True
552 conv5_block3_preact_relu True
553 conv5_block3_1_conv True
554 conv5_block3_1_bn True
555 conv5_block3_1_relu True
556 conv5_block3_2_pad True
557 conv5_block3_2_conv True
558 conv5_block3_2_bn True
559 conv5_block3_2_relu True
```

```
560 conv5_block3_3_conv True
561 conv5_block3_out True
562 post_bn True
563 post_relu True

model_pretrained.compile(loss='binary_crossentropy'
              , optimizer = keras.optimizers.Adam(learning_rate=2e-6),
metrics='binary_accuracy')

model_pretrained.summary()

Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
resnet152v2 (Functional)     (None, 7, 7, 2048)        58331648
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 58,594,049
Trainable params: 4,731,137
Non-trainable params: 53,862,912
_________________________________________________________________

history = model_pretrained.fit(ds_train,
          batch_size = BATCH, epochs = 50,
          validation_data=ds_val,
          callbacks=[early_stopping, plateau],
          steps_per_epoch=(len(train_df)/BATCH),
          validation_steps=(len(val_df)/BATCH));

Epoch 1/50
130/130 [==============================] - 104s 797ms/step - loss:
0.1608 - binary_accuracy: 0.9419 - val_loss: 0.1463 -
val_binary_accuracy: 0.9570
Epoch 3/50
130/130 [==============================] - 103s 784ms/step - loss:
0.1316 - binary_accuracy: 0.9564 - val_loss: 0.1318 -
val_binary_accuracy: 0.9608
Epoch 4/50
130/130 [==============================] - 103s 785ms/step - loss:
0.1370 - binary_accuracy: 0.9554 - val_loss: 0.1272 -
```

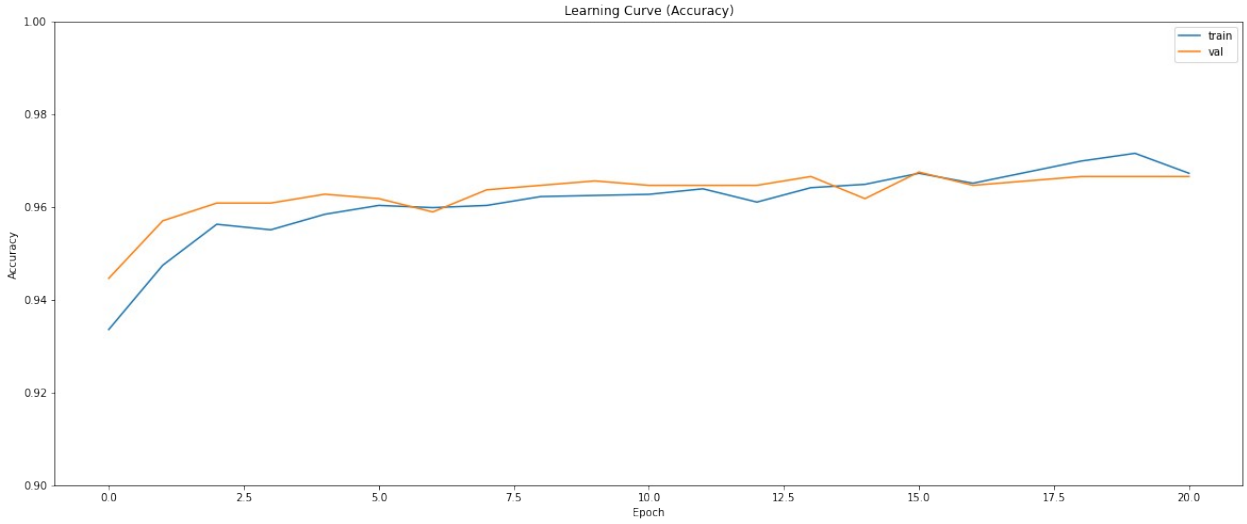
```
val_binary_accuracy: 0.9608
Epoch 5/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1315 - binary_accuracy: 0.9561 - val_loss: 0.1232 -
val_binary_accuracy: 0.9628
Epoch 6/50
130/130 [==============================] - 103s 790ms/step - loss:
0.1176 - binary_accuracy: 0.9576 - val_loss: 0.1206 -
val_binary_accuracy: 0.9618
Epoch 7/50
130/130 [==============================] - 103s 788ms/step - loss:
0.1156 - binary_accuracy: 0.9597 - val_loss: 0.1210 -
val_binary_accuracy: 0.9589
Epoch 8/50
130/130 [==============================] - 103s 791ms/step - loss:
0.1166 - binary_accuracy: 0.9607 - val_loss: 0.1171 -
val_binary_accuracy: 0.9637
Epoch 9/50
130/130 [==============================] - 103s 790ms/step - loss:
0.1038 - binary_accuracy: 0.9642 - val_loss: 0.1157 -
val_binary_accuracy: 0.9647
Epoch 10/50
130/130 [==============================] - 105s 804ms/step - loss:
0.1038 - binary_accuracy: 0.9660 - val_loss: 0.1121 -
val_binary_accuracy: 0.9656
Epoch 11/50
130/130 [==============================] - 104s 796ms/step - loss:
0.1129 - binary_accuracy: 0.9596 - val_loss: 0.1131 -
val_binary_accuracy: 0.9647
Epoch 12/50
130/130 [==============================] - 103s 785ms/step - loss:
0.1005 - binary_accuracy: 0.9612 - val_loss: 0.1110 -
val_binary_accuracy: 0.9647
Epoch 13/50
130/130 [==============================] - 103s 789ms/step - loss:
0.1088 - binary_accuracy: 0.9600 - val_loss: 0.1109 -
val_binary_accuracy: 0.9647
Epoch 14/50
130/130 [==============================] - 103s 788ms/step - loss:
0.0948 - binary_accuracy: 0.9636 - val_loss: 0.1080 -
val_binary_accuracy: 0.9666
Epoch 15/50
130/130 [==============================] - 104s 796ms/step - loss:
0.0966 - binary_accuracy: 0.9676 - val_loss: 0.1134 -
val_binary_accuracy: 0.9618
Epoch 16/50
130/130 [==============================] - 103s 790ms/step - loss:
0.0954 - binary_accuracy: 0.9653 - val_loss: 0.1070 -
val_binary_accuracy: 0.9675
```

```
Epoch 17/50
130/130 [==============================] - 103s 791ms/step - loss:
0.0999 - binary_accuracy: 0.9596 - val_loss: 0.1106 -
val_binary_accuracy: 0.9647
Epoch 18/50
130/130 [==============================] - 103s 789ms/step - loss:
0.0924 - binary_accuracy: 0.9685 - val_loss: 0.1106 -
val_binary_accuracy: 0.9656

Epoch 00018: ReduceLROnPlateau reducing learning rate to
3.999999989900971e-07.
Epoch 19/50
130/130 [==============================] - 103s 787ms/step - loss:
0.0888 - binary_accuracy: 0.9708 - val_loss: 0.1080 -
val_binary_accuracy: 0.9666
Epoch 20/50
130/130 [==============================] - 103s 788ms/step - loss:
0.0815 - binary_accuracy: 0.9738 - val_loss: 0.1094 -
val_binary_accuracy: 0.9666

Epoch 00020: ReduceLROnPlateau reducing learning rate to
8.00000009348878e-08.
Epoch 21/50
130/130 [==============================] - 103s 788ms/step - loss:
0.0870 - binary_accuracy: 0.9677 - val_loss: 0.1094 -
val_binary_accuracy: 0.9666
```

```
fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y = history.history['loss'])
sns.lineplot(x = history.epoch, y = history.history['val_loss'])
ax.set_title('Learning Curve (Loss)')
ax.set_ylabel('Loss')
ax.set_xlabel('Epoch')
ax.set_ylim(0, 0.3)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
fig, ax = plt.subplots(figsize=(20,8))
sns.lineplot(x = history.epoch, y =
history.history['binary_accuracy'])
sns.lineplot(x = history.epoch, y =
history.history['val_binary_accuracy'])
ax.set_title('Learning Curve (Accuracy)')
ax.set_ylabel('Accuracy')
ax.set_xlabel('Epoch')
ax.set_ylim(0.90, 1.0)
ax.legend(['train', 'val'], loc='best')
plt.show()
```




```
score = model_pretrained.evaluate(ds_val, steps = len(val_df)/BATCH,
verbose = 0)
print('Val loss:', score[0])
print('Val accuracy:', score[1])
```

```
Val loss: 0.10701081901788712
Val accuracy: 0.9675262570381165
```

```
score = model_pretrained.evaluate(ds_test, steps = len(df_test),
verbose = 0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.21019524335861206
Test accuracy: 0.9198718070983887
```

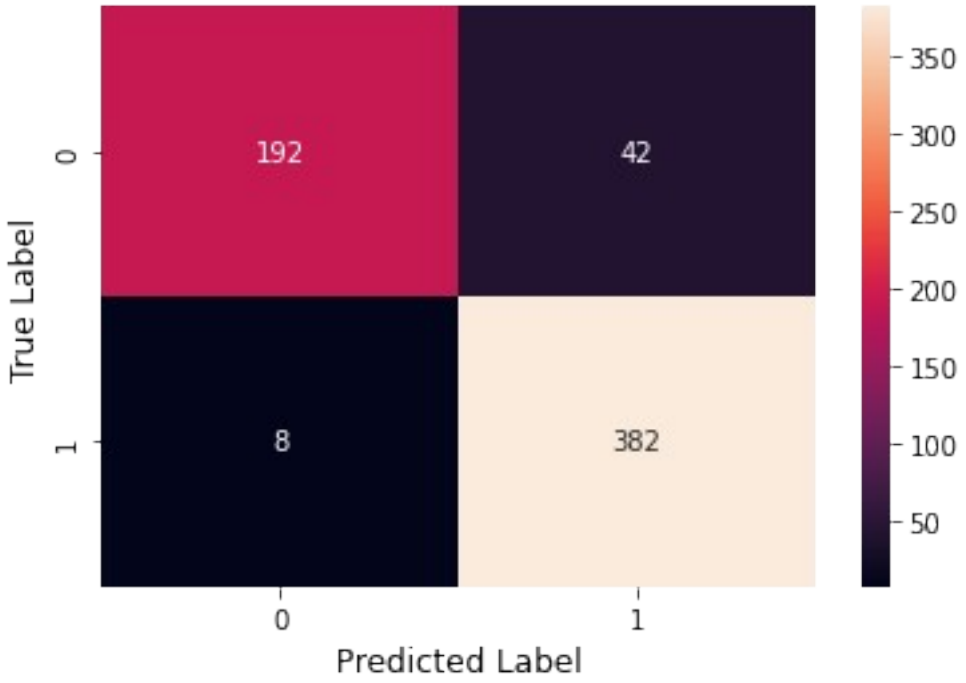
As expected, the fine-tuning approach has reached the best score. We end this notebook by showing a few performance metrics.

Performance Metrics

```
num_label = {'Normal': 0, 'Pneumonia' : 1}
Y_test = df_test['class'].copy().map(num_label).astype('int')

ds_test.reset()
predictions = model_pretrained.predict(ds_test, steps=len(ds_test),
verbose=0)
pred_labels= np.where(predictions>0.5, 1, 0)

print("Test Accuracy: ", accuracy_score(Y_test, pred_labels))
```

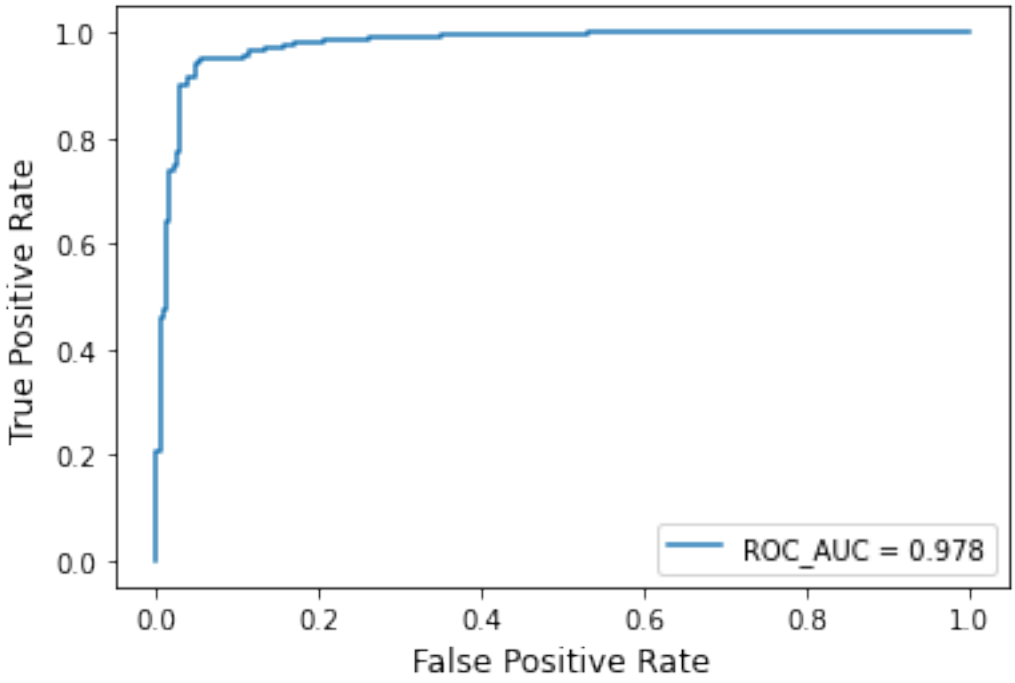
```
Test Accuracy:  0.9198717948717948
```

```
confusion_matrix = metrics.confusion_matrix(Y_test, pred_labels)
sns.heatmap(confusion_matrix, annot=True, fmt="d")

plt.xlabel("Predicted Label", fontsize= 12)
plt.ylabel("True Label", fontsize= 12)

plt.show()
```




```
print(metrics.classification_report(Y_test, pred_labels, labels = [0,
1]))
```

```
              precision    recall  f1-score   support

           0       0.96      0.82      0.88       234
           1       0.90      0.98      0.94       390

    accuracy                           0.92       624
   macro avg       0.93      0.90      0.91       624
weighted avg       0.92      0.92      0.92       624


roc_auc = metrics.roc_auc_score(Y_test, predictions)
print('ROC_AUC: ', roc_auc)

fpr, tpr, thresholds = metrics.roc_curve(Y_test, predictions)

plt.plot(fpr, tpr, label = 'ROC_AUC = %0.3f' % roc_auc)

plt.xlabel("False Positive Rate", fontsize= 12)
plt.ylabel("True Positive Rate", fontsize= 12)
plt.legend(loc="lower right")

plt.show()

ROC_AUC:  0.9779092702169625
```


True Positive Rate
False Positive Rate
ROC_AUC = 0.978
0.0
0.2
0.4
0.6
0.8
1.0

Experiment 10: Overview of Different Evaluation Metrics for Healthcare Datasets

---

Aim:
To provide a comprehensive overview of various evaluation metrics used for assessing the performance of predictive models in healthcare datasets. Understanding these metrics is crucial for determining the accuracy and reliability of models used for disease diagnosis, patient outcomes prediction, and medical image classification.

Theory:
In healthcare, evaluating the performance of predictive models is essential to ensure they provide accurate and meaningful results. Various evaluation metrics are used depending on the specific context of the task, such as binary classification, multiclass classification, or regression. Here's a breakdown of the most commonly used evaluation metrics in healthcare settings:

1. **Accuracy:**
   Accuracy measures the proportion of correctly classified instances out of the total instances. It is calculated as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

   Where:

   - TP = True Positives
   - TN = True Negatives
   - FP = False Positives
   - FN = False Negatives

   Use Case: While accuracy is a useful metric, it may be misleading in imbalanced datasets, where one class significantly outnumbers another.

2. **Precision:**
   Precision measures the accuracy of the positive predictions made by the model. It is calculated as:

$$Precision = \frac{TP}{TP + FP}$$

Use Case: Precision is particularly important in scenarios where false positives carry significant risks, such as misdiagnosing a disease.

3. **Recall (Sensitivity or True Positive Rate):**
   Recall measures the ability of a model to identify all relevant instances. It is calculated as:

$$Recall = \frac{TP}{TP + FN}$$

Use Case: Recall is critical in healthcare when missing a positive case (e.g., a patient with a disease) can have serious consequences.

4. **F1-Score:**
   The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is calculated as:

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Use Case: The F1-score is particularly useful in imbalanced datasets, as it considers both false positives and false negatives.

5. **Area Under the Receiver Operating Characteristic Curve (AUC-ROC):**
   The AUC-ROC measures the model's ability to distinguish between classes at various thresholds. AUC represents the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance.
   Use Case: AUC-ROC is valuable in medical diagnostics, helping clinicians understand the trade-off between sensitivity and specificity.

6. **Area Under the Precision-Recall Curve (AUC-PR):**
   AUC-PR focuses on the performance of the model across different thresholds, particularly in cases of imbalanced datasets. It plots precision against recall for different thresholds.
   Use Case: AUC-PR is beneficial when dealing with highly imbalanced datasets, where positive cases are rare.
7. **Mean Absolute Error (MAE) and Mean Squared Error (MSE):**
   These metrics are primarily used for regression tasks. MAE measures the average magnitude of errors in a set of predictions, while MSE measures the average of the squares of the errors.

Mean squared error	$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$

   Use Case: MAE and MSE are used for evaluating continuous outcomes, such as predicting patient vital signs.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import svm
from sklearn.metrics import accuracy_score

# loading the data from csv file to a Pandas DataFrame
parkinsons_data = pd.read_csv('/content/parkinsons (1).csv')

# printing the first 5 rows of the dataframe
parkinsons_data.head()

{"type":"dataframe","variable_name":"parkinsons_data"}

# number of rows and columns in the dataframe
parkinsons_data.shape

(195, 24)

# getting more information about the dataset
parkinsons_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 195 entries, 0 to 194
Data columns (total 24 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   name              195 non-null    object
 1   MDVP:Fo(Hz)       195 non-null    float64
 2   MDVP:Fhi(Hz)      195 non-null    float64
 3   MDVP:Flo(Hz)      195 non-null    float64
 4   MDVP:Jitter(%)    195 non-null    float64
 5   MDVP:Jitter(Abs)  195 non-null    float64
 6   MDVP:RAP          195 non-null    float64
 7   MDVP:PPQ          195 non-null    float64
 8   Jitter:DDP        195 non-null    float64
 9   MDVP:Shimmer      195 non-null    float64
 10  MDVP:Shimmer(dB)  195 non-null    float64
 11  Shimmer:APQ3      195 non-null    float64
 12  Shimmer:APQ5      195 non-null    float64
 13  MDVP:APQ          195 non-null    float64
 14  Shimmer:DDA       195 non-null    float64
 15  NHR               195 non-null    float64
 16  HNR               195 non-null    float64
 17  RPDE              195 non-null    float64
 18  DFA               195 non-null    float64
 19  spread1           195 non-null    float64
 20  spread2           195 non-null    float64
 21  D2                195 non-null    float64
 22  PPE               195 non-null    float64
```

```
 23  status            195 non-null    int64
dtypes: float64(22), int64(1), object(1)
memory usage: 36.7+ KB
```

```
# checking for missing values in each column
parkinsons_data.isnull().sum()
```

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
status              0
dtype: int64
```

```
# getting some statistical measures about the data
parkinsons_data.describe()
```

```
{"type":"dataframe"}
```

```
# distribution of target Variable
parkinsons_data['status'].value_counts()
```

```
status
1    147
0     48
Name: count, dtype: int64
```

```
print(parkinsons_data.isna().sum())  # Display the count of NaN values
in each column
```

```
name                0
MDVP:Fo(Hz)         0
```

```
MDVP:Fhi(Hz)         0
MDVP:Flo(Hz)         0
MDVP:Jitter(%)       0
MDVP:Jitter(Abs)     0
MDVP:RAP             0
MDVP:PPQ             0
Jitter:DDP           0
MDVP:Shimmer         0
MDVP:Shimmer(dB)     0
Shimmer:APQ3         0
Shimmer:APQ5         0
MDVP:APQ             0
Shimmer:DDA          0
NHR                  0
HNR                  0
RPDE                 0
DFA                  0
spread1              0
spread2              0
D2                   0
PPE                  0
status               0
dtype: int64
```

```
print(parkinsons_data.dtypes)
```

```
name                 object
MDVP:Fo(Hz)         float64
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)      float64
MDVP:Jitter(Abs)    float64
MDVP:RAP            float64
MDVP:PPQ            float64
Jitter:DDP          float64
MDVP:Shimmer        float64
MDVP:Shimmer(dB)    float64
Shimmer:APQ3        float64
Shimmer:APQ5        float64
MDVP:APQ            float64
Shimmer:DDA         float64
NHR                 float64
HNR                 float64
RPDE                float64
DFA                 float64
spread1             float64
spread2             float64
D2                  float64
PPE                 float64
```

```
status                int64
dtype: object

# Selecting only numeric columns
numeric_columns =
parkinsons_data.select_dtypes(include=['number']).columns
mean_values = parkinsons_data.groupby('status')
[numeric_columns].mean()
print(mean_values)

        MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
status
0        181.937771    223.636750    145.207292        0.003866
1        145.180762    188.441463    106.893558        0.006989

        MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer
\
status

0               0.000023  0.001925  0.002056    0.005776      0.017615

1               0.000051  0.003757  0.003900    0.011273      0.033658


        MDVP:Shimmer(dB)  ...  Shimmer:DDA       NHR        HNR
RPDE  \
status                    ...

0               0.162958  ...     0.028511  0.011483  24.678750
0.442552
1               0.321204  ...     0.053027  0.029211  20.974048
0.516816

             DFA   spread1   spread2        D2       PPE  status
status
0       0.695716 -6.759264  0.160292  2.154491  0.123017     0.0
1       0.725408 -5.333420  0.248133  2.456058  0.233828     1.0

[2 rows x 23 columns]

X = parkinsons_data.drop(columns=['name','status'], axis=1)
Y = parkinsons_data['status']

print(X)

     MDVP:Fo(Hz)  MDVP:Fhi(Hz)  MDVP:Flo(Hz)  MDVP:Jitter(%)  \
0        119.992       157.302        74.997         0.00784
1        122.400       148.650       113.819         0.00968
2        116.682       131.111       111.555         0.01050
3        116.676       137.871       111.366         0.00997
4        116.014       141.781       110.655         0.01284
```

```
..           ...          ...          ...          ...
190      174.188      230.978       94.261      0.00459
191      209.516      253.017       89.488      0.00564
192      174.688      240.005       74.287      0.01360
193      198.764      396.961       74.904      0.00740
194      214.289      260.277       77.973      0.00567

     MDVP:Jitter(Abs)  MDVP:RAP  MDVP:PPQ  Jitter:DDP  MDVP:Shimmer  \
0             0.00007   0.00370   0.00554     0.01109       0.04374
1             0.00008   0.00465   0.00696     0.01394       0.06134
2             0.00009   0.00544   0.00781     0.01633       0.05233
3             0.00009   0.00502   0.00698     0.01505       0.05492
4             0.00011   0.00655   0.00908     0.01966       0.06425
..                ...       ...       ...         ...           ...
190           0.00003   0.00263   0.00259     0.00790       0.04087
191           0.00003   0.00331   0.00292     0.00994       0.02751
192           0.00008   0.00624   0.00564     0.01873       0.02308
193           0.00004   0.00370   0.00390     0.01109       0.02296
194           0.00003   0.00295   0.00317     0.00885       0.01884

     MDVP:Shimmer(dB)  ...  MDVP:APQ  Shimmer:DDA      NHR     HNR
RPDE  \
0               0.426  ...   0.02971      0.06545  0.02211  21.033
0.414783
1               0.626  ...   0.04368      0.09403  0.01929  19.085
0.458359
2               0.482  ...   0.03590      0.08270  0.01309  20.651
0.429895
3               0.517  ...   0.03772      0.08771  0.01353  20.644
0.434969
4               0.584  ...   0.04465      0.10470  0.01767  19.649
0.417356
..                ...  ...       ...          ...      ...     ...
...
190             0.405  ...   0.02745      0.07008  0.02764  19.517
0.448439
191             0.263  ...   0.01879      0.04812  0.01810  19.147
0.431674
192             0.256  ...   0.01667      0.03804  0.10715  17.883
0.407567
193             0.241  ...   0.01588      0.03794  0.07223  19.020
0.451221
194             0.190  ...   0.01373      0.03078  0.04398  21.209
0.462803

          DFA   spread1   spread2        D2       PPE
0    0.815285 -4.813031  0.266482  2.301442  0.284654
1    0.819521 -4.075192  0.335590  2.486855  0.368674
2    0.825288 -4.443179  0.311173  2.342259  0.332634
3    0.819235 -4.117501  0.334147  2.405554  0.368975
```

```
4    0.823484 -3.747787  0.234513  2.332180  0.410335
..        ...       ...       ...       ...       ...
190  0.657899 -6.538586  0.121952  2.657476  0.133050
191  0.683244 -6.195325  0.129303  2.784312  0.168895
192  0.655683 -6.787197  0.158453  2.679772  0.131728
193  0.643956 -6.744577  0.207454  2.138608  0.123306
194  0.664357 -5.724056  0.190667  2.555477  0.148569

[195 rows x 22 columns]
```

```
print(Y)
```

```
0      1
1      1
2      1
3      1
4      1
      ..
190    0
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
test_size=0.2, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(195, 22) (156, 22) (39, 22)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
print(X_train)
```

```
[[ 0.63239631 -0.02731081 -0.87985049 ... -0.97586547 -0.55160318
   0.07769494]
 [-1.05512719 -0.83337041 -0.9284778  ...  0.3981808  -0.61014073
   0.39291782]
 [ 0.02996187 -0.29531068 -1.12211107 ... -0.43937044 -0.62849605
  -0.50948408]
 ...
 [-0.9096785  -0.6637302  -0.160638   ...  1.22001022 -0.47404629
  -0.2159482 ]
 [-0.35977689  0.19731822 -0.79063679 ... -0.17896029 -0.47272835
   0.28181221]
 [ 1.01957066  0.19922317 -0.61914972 ... -0.716232    1.23632066
  -0.05829386]]
```

```
model = svm.SVC(kernel='linear')

# training the SVM model with training data
model.fit(X_train, Y_train)
```

```
SVC(kernel='linear')
```

```
# accuracy score on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(Y_train, X_train_prediction)

print('Accuracy score of training data : ', training_data_accuracy)
```

```
Accuracy score of training data :  0.8846153846153846
```

```
# accuracy score on testing data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(Y_test, X_test_prediction)

print('Accuracy score of test data : ', test_data_accuracy)
```

```
Accuracy score of test data :  0.8717948717948718
```

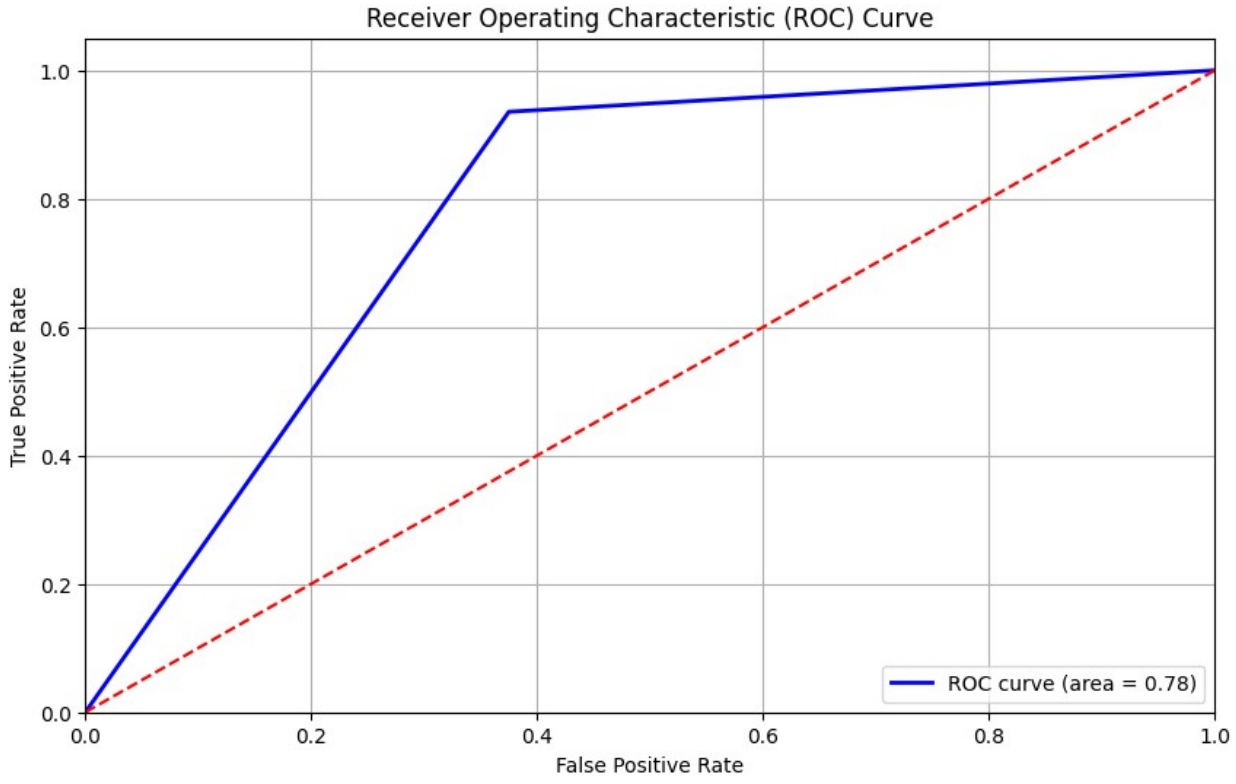
```
input_data =
(197.07600,206.89600,192.05500,0.00289,0.00001,0.00166,0.00168,0.00498
,0.01098,0.09700,0.00563,0.00680,0.00802,0.01689,0.00339,26.77500,0.42
2229,0.741367,-7.348300,0.177551,1.743867,0.085569)

# changing input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the numpy array
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

# standardize the data
std_data = scaler.transform(input_data_reshaped)

prediction = model.predict(std_data)
print(prediction)


if (prediction[0] == 0):
  print("The Person does not have Parkinsons Disease")

else:
  print("The Person has Parkinsons")
```

```
[0]
The Person does not have Parkinsons Disease
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493:
UserWarning: X does not have valid feature names, but StandardScaler
```

```
was fitted with feature names
  warnings.warn(
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(Y_test, X_test_prediction)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area =
{:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], color='red', linestyle='--')
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')
plt.legend(loc='lower right')
plt.grid()
plt.show()
```



```python
from sklearn.metrics import classification_report

print(classification_report(Y_test, X_test_prediction))
```

```
              precision    recall  f1-score   support

           0       0.71      0.62      0.67         8
           1       0.91      0.94      0.92        31

    accuracy                           0.87        39
   macro avg       0.81      0.78      0.79        39
weighted avg       0.87      0.87      0.87        39
```

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(Y_test, X_test_prediction)
mse = mean_squared_error(Y_test, X_test_prediction)

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
```

```
Mean Absolute Error (MAE): 0.1282051282051282
Mean Squared Error (MSE): 0.1282051282051282
```

```python
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(Y_test, X_test_prediction).ravel()
specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

print(f'Specificity: {specificity:.2f}')
```

```
Specificity: 0.62
```