correlateddiseaseprediction

January 18, 2024

Greetings! We'll use common health signs to forecast illnesses in this project. Our goal is to develop a model that can **precisely detect diseases based on fundamental symptoms and health indicators** using the Disease Symptoms and Patient Profile Dataset.

This project will provide us a thorough overview of how to forecast illnesses based on fundamental medical data and also we will see how the **fundamental diseases are correlated.**

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
df=pd.read_csv("Disease_symptom_and_patient_profile_dataset.csv")
```

How Does our data look like.

[4]:
```
df.sample(10)
```

[4]:
```
                Disease Fever Cough Fatigue Difficulty Breathing  Age  \
264          Osteoporosis   Yes    No     Yes                  Yes   55
212            Depression    No   Yes     Yes                   No   50
186            Depression    No    No     Yes                   No   45
187              Diabetes   Yes    No     Yes                   No   45
6               Influenza   Yes   Yes     Yes                  Yes   25
50                 Asthma    No    No      No                  Yes   31
222           Sleep Apnea   Yes    No     Yes                  Yes   50
65           Hypertension   Yes   Yes     Yes                   No   35
99        Gastroenteritis    No    No      No                   No   38
327   Alzheimer's Disease   Yes    No     Yes                   No   70

     Gender Blood Pressure Cholesterol Level Outcome Variable
264    Male           High               Low         Positive
212  Female            Low               Low         Negative
186  Female           High              High         Positive
187  Female           High              High         Positive
6    Female         Normal            Normal         Positive
50     Male         Normal               Low         Negative
222    Male           High              High         Negative
65   Female           High            Normal         Negative
```

```
99     Male          Normal              Low          Negative
327   Female          High            Normal          Negative
```

```
[9]: #Checking Data Types
     df.info()
     print("*****************#####################*********************")
     print(f"Data size is {df.shape}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349 entries, 0 to 348
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Disease               349 non-null    object
 1   Fever                 349 non-null    object
 2   Cough                 349 non-null    object
 3   Fatigue               349 non-null    object
 4   Difficulty Breathing  349 non-null    object
 5   Age                   349 non-null    int64
 6   Gender                349 non-null    object
 7   Blood Pressure        349 non-null    object
 8   Cholesterol Level     349 non-null    object
 9   Outcome Variable      349 non-null    object
dtypes: int64(1), object(9)
memory usage: 27.4+ KB
*****************#####################*********************
Data size is (349, 10)
```

From the initial glance at our data, we can observe that most of our variables are categorical, with 'Age' being the only numerical variable.Our target variable is 'Disease', which we are trying to predict. Let's explore this variable.

```
[13]: #Let's check what all diseases are listed
      df['Disease'].unique()
```

```
[13]: array(['Influenza', 'Common Cold', 'Eczema', 'Asthma', 'Hyperthyroidism',
              'Allergic Rhinitis', 'Anxiety Disorders', 'Diabetes',
              'Gastroenteritis', 'Pancreatitis', 'Rheumatoid Arthritis',
              'Depression', 'Liver Cancer', 'Stroke', 'Urinary Tract Infection',
              'Dengue Fever', 'Hepatitis', 'Kidney Cancer', 'Migraine',
              'Muscular Dystrophy', 'Sinusitis', 'Ulcerative Colitis',
              'Bipolar Disorder', 'Bronchitis', 'Cerebral Palsy',
              'Colorectal Cancer', 'Hypertensive Heart Disease',
              'Multiple Sclerosis', 'Myocardial Infarction (Heart…',
              'Urinary Tract Infection (UTI)', 'Osteoporosis', 'Pneumonia',
              'Atherosclerosis', 'Chronic Obstructive Pulmonary…', 'Epilepsy',
              'Hypertension', 'Obsessive-Compulsive Disorde…', 'Psoriasis',
              'Rubella', 'Cirrhosis', 'Conjunctivitis (Pink Eye)',
              'Liver Disease', 'Malaria', 'Spina Bifida', 'Kidney Disease',
              'Osteoarthritis', 'Klinefelter Syndrome', 'Acne', 'Brain Tumor',
              'Cystic Fibrosis', 'Glaucoma', 'Rabies', 'Chickenpox',
              'Coronary Artery Disease', 'Eating Disorders (Anorexia,…',
              'Fibromyalgia', 'Hemophilia', 'Hypoglycemia', 'Lymphoma',
              'Tuberculosis', 'Lung Cancer', 'Hypothyroidism',
              'Autism Spectrum Disorder (ASD)', "Crohn's Disease",
              'Hyperglycemia', 'Melanoma', 'Ovarian Cancer', 'Turner Syndrome',
              'Zika Virus', 'Cataracts', 'Pneumocystis Pneumonia (PCP)',
              'Scoliosis', 'Sickle Cell Anemia', 'Tetanus', 'Anemia', 'Cholera',
              'Endometriosis', 'Sepsis', 'Sleep Apnea', 'Down Syndrome',
              'Ebola Virus', 'Lyme Disease', 'Pancreatic Cancer', 'Pneumothorax',
              'Appendicitis', 'Esophageal Cancer', 'HIV/AIDS', 'Marfan Syndrome',
              "Parkinson's Disease", 'Hemorrhoids',
              'Polycystic Ovary Syndrome (PCOS)',
              'Systemic Lupus Erythematosus…', 'Typhoid Fever',
              'Breast Cancer', 'Measles', 'Osteomyelitis', 'Polio',
              'Chronic Kidney Disease', 'Hepatitis B', 'Prader-Willi Syndrome',
              'Thyroid Cancer', 'Bladder Cancer', 'Otitis Media (Ear Infection)',
              'Tourette Syndrome', "Alzheimer's Disease",
              'Chronic Obstructive Pulmonary Disease (COPD)', 'Dementia',
              'Diverticulitis', 'Mumps', 'Cholecystitis', 'Prostate Cancer',
              'Schizophrenia', 'Gout', 'Testicular Cancer', 'Tonsillitis',
              'Williams Syndrome'], dtype=object)
```

[16]:
```
#For example let's check the possible status of any one component ( let say␣
↪Blood Pressure)  of the listed patients.
df['Blood Pressure'].unique()
```

[16]:
```
array(['Low', 'Normal', 'High'], dtype=object)
```

[19]:
```
#checking for null values
df.isnull().sum()
```

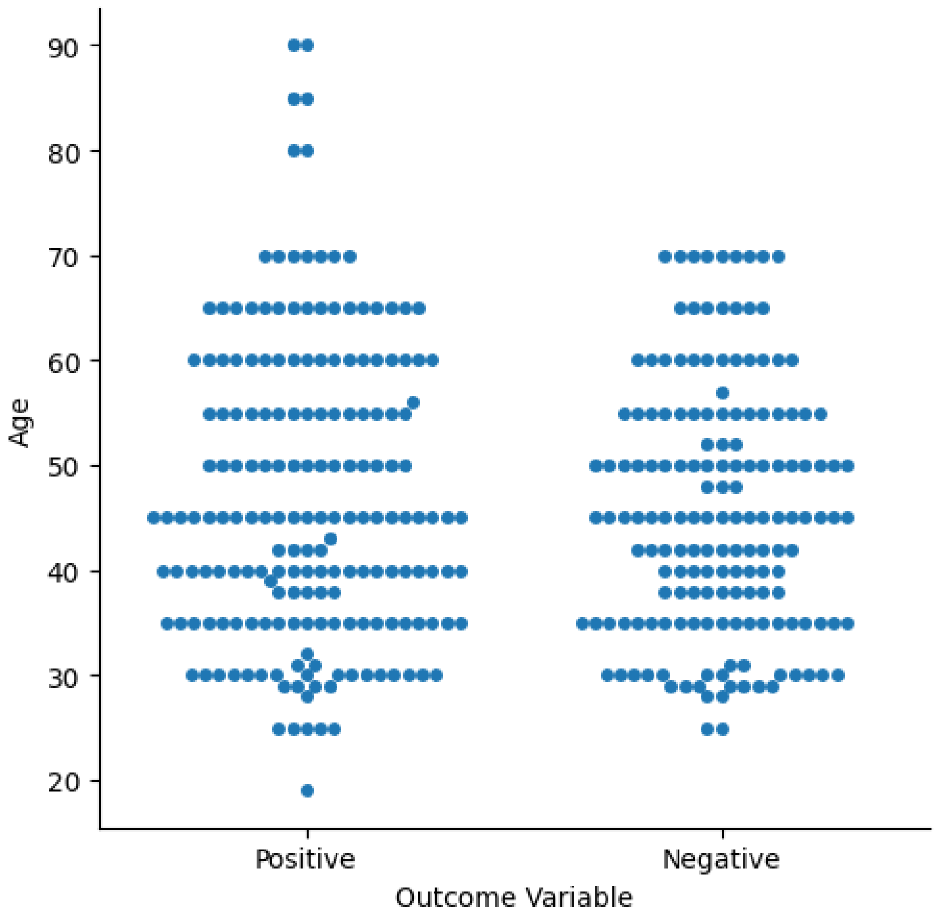
[19]:
```
Disease                 0
Fever                   0
Cough                   0
Fatigue                 0
Difficulty Breathing    0
Age                     0
Gender                  0
Blood Pressure          0
Cholesterol Level       0
Outcome Variable        0
dtype: int64
```

```
[20]: #let's check for number of unique values
      df.nunique()
```

```
[20]: Disease                 116
      Fever                     2
      Cough                     2
      Fatigue                   2
      Difficulty Breathing      2
      Age                      26
      Gender                    2
      Blood Pressure            3
      Cholesterol Level         3
      Outcome Variable          2
      dtype: int64
```

```
[21]: sns.catplot(x = 'Outcome Variable' , y = 'Age' , data = df , kind = "swarm")
```

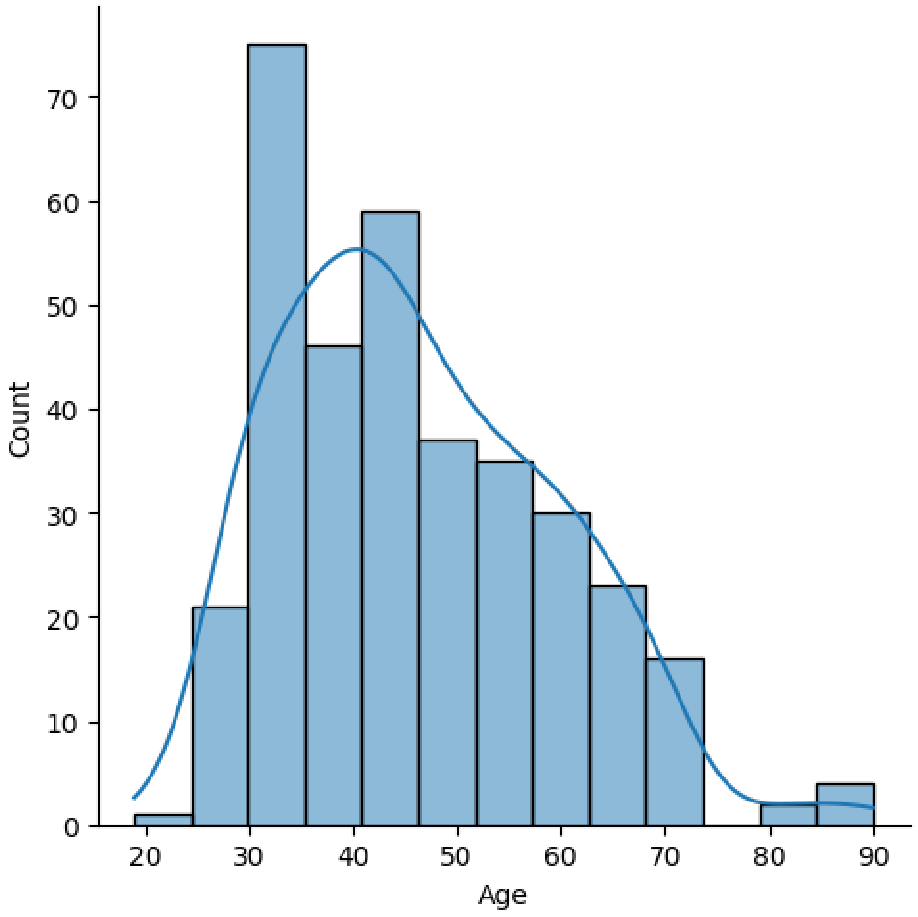
```
[21]: <seaborn.axisgrid.FacetGrid at 0x221a5d844a0>
```



People in old ages have a higher probability of being tested positive for diseases which is an outlier for our dataset

```
[22]: sns.displot(df['Age'] , kde=True)
```

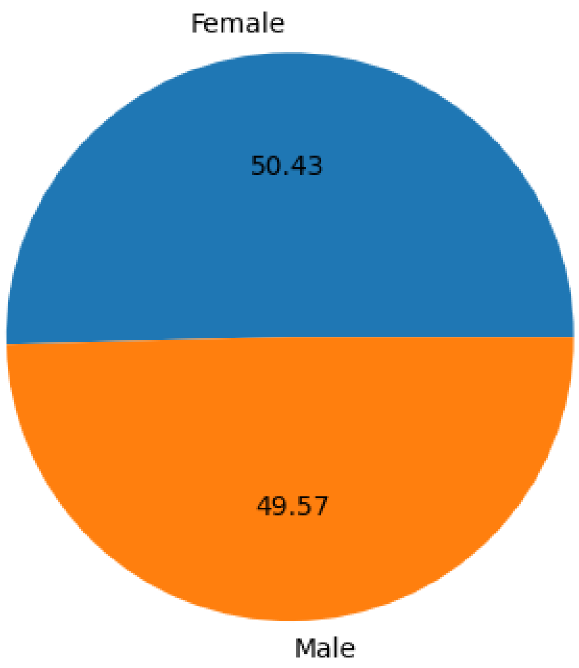
```
[22]: <seaborn.axisgrid.FacetGrid at 0x221a5480e30>
```



There is no major skewness in the dataset with a few outliers

```
[23]: df.groupby('Gender').size().plot(kind='pie', autopct='%.2f')
```

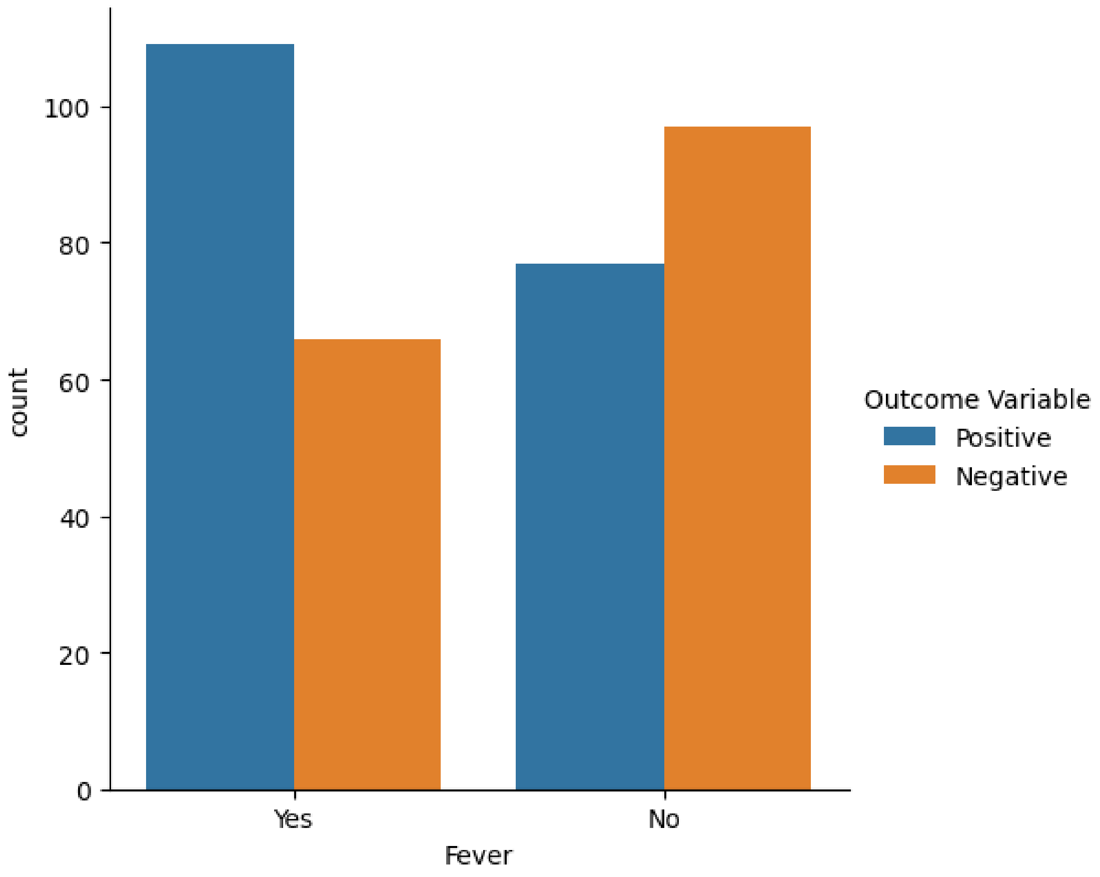
```
[23]: <Axes: >
```



The dataset is quite evenly distributed based on gender

```
[24]: sns.catplot(x='Fever' , kind='count',data=df , hue = "Outcome Variable")
```

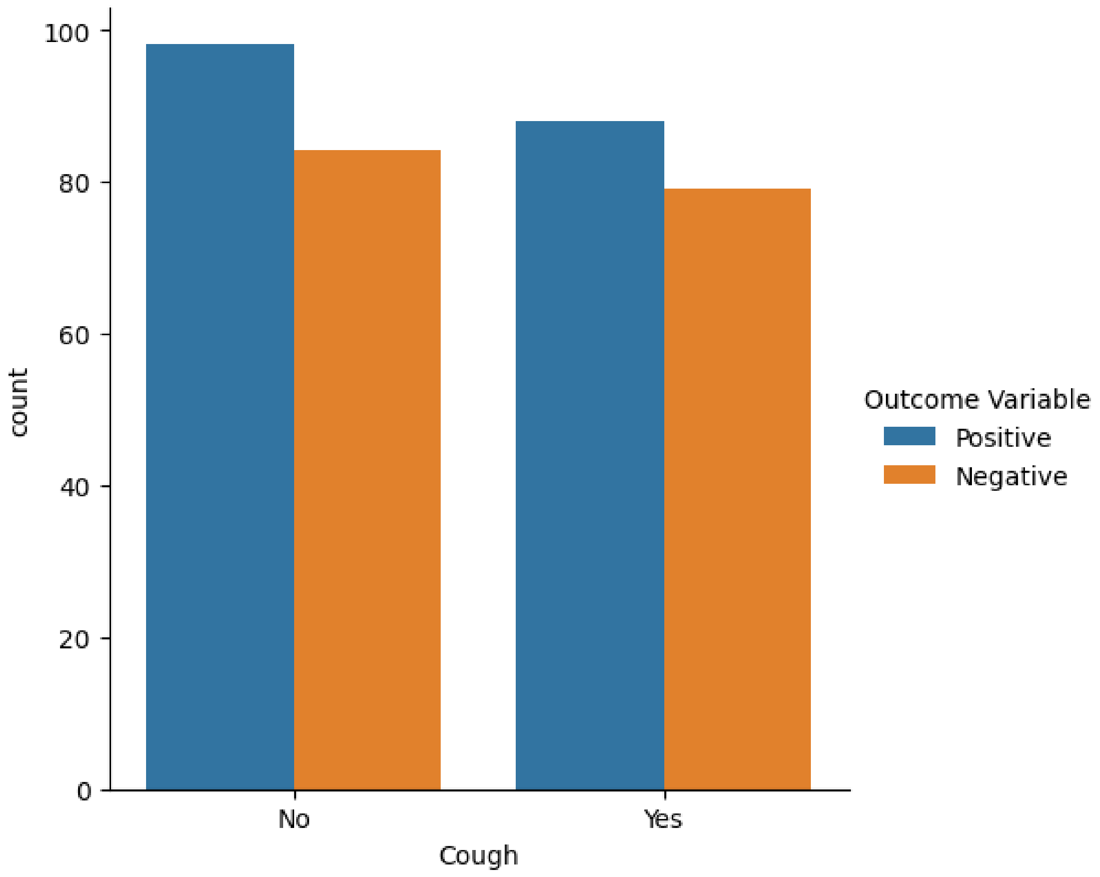
```
[24]: <seaborn.axisgrid.FacetGrid at 0x221a5eb4ef0>
```



Having Fever is a major indication of a positive diagnosis

[27]:
```
sns.catplot(x='Cough' , kind='count',data=df , hue = "Outcome Variable")
```

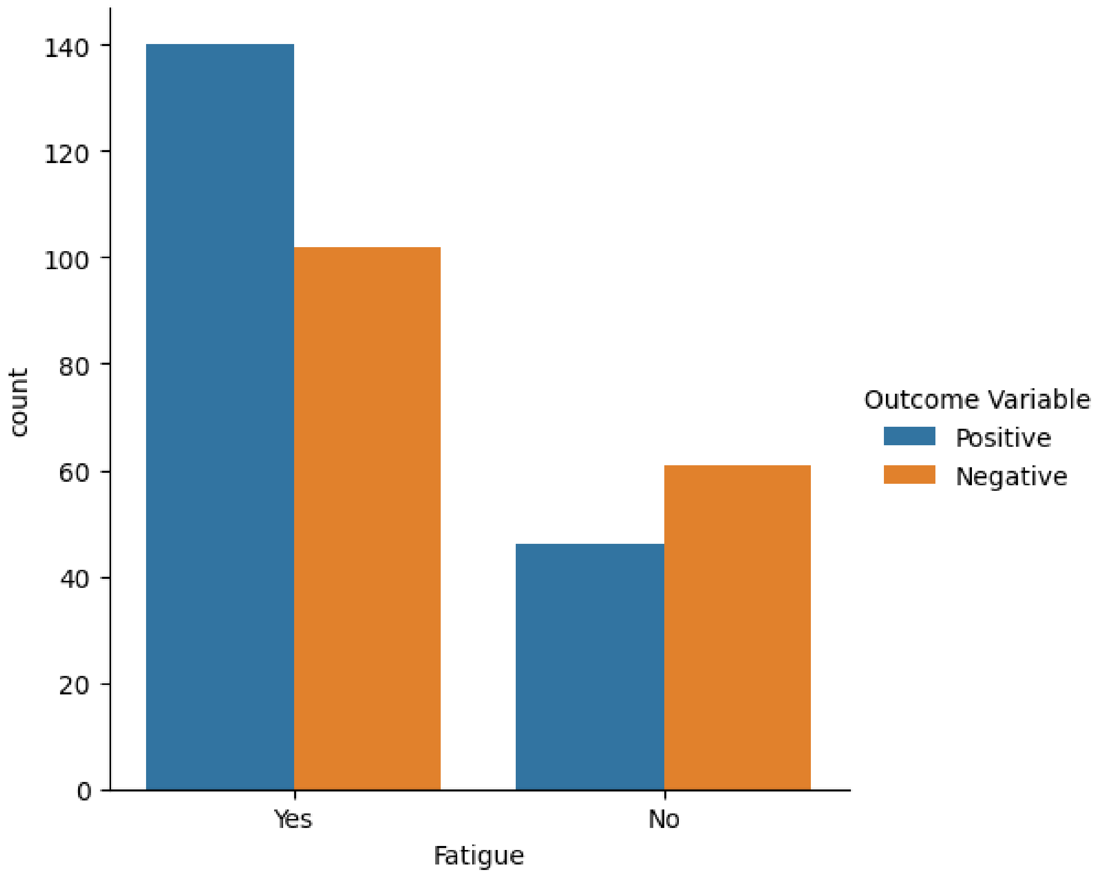
[27]:
```
<seaborn.axisgrid.FacetGrid at 0x221a5ebe330>
```



Nothing major can be inferred from cough as it is quite common

```
[28]: sns.catplot(x='Fatigue' , kind='count',data=df , hue = "Outcome Variable")
```

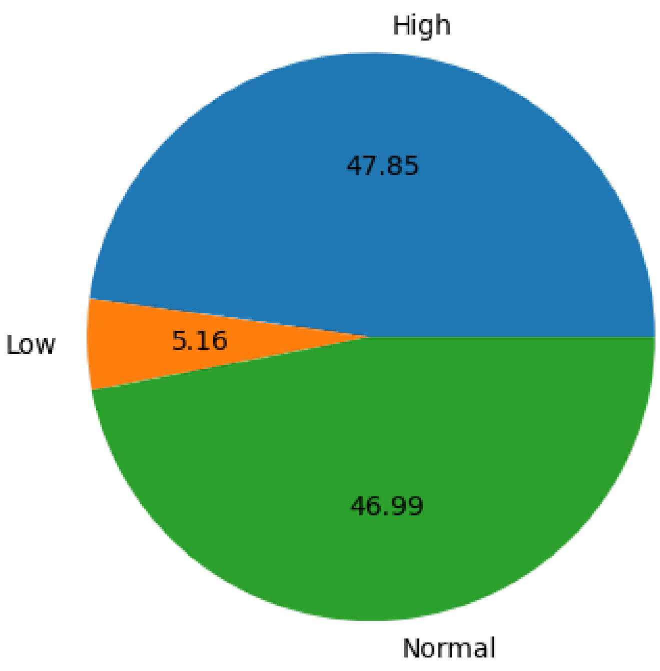
```
[28]: <seaborn.axisgrid.FacetGrid at 0x221a56596a0>
```



A lot of people irrespective of positive or negative experience a lot of fatique

```
[29]: df.groupby('Blood Pressure').size().plot(kind='pie', autopct='%.2f')
```

```
[29]: <Axes: >
```



Majority of the subjects have a normal or high blood pressure

print(sum(df.Disease.value_counts() >= 1)) print(sum(df.Disease.value_counts() == 1))

[18]:
```
print(sum(df.Disease.value_counts() > 9))
print(sum(df.Disease.value_counts() <= 9))
```

```
6
110
```

Upon examining the 'Disease' column, we notice a large number of unique diseases, many of which have only 1 to 5 samples. For a reliable disease prediction model, this sample size is insufficient.

Predicting diseases with such limited information could lead to inaccurate results and misdiagnosis, which we want to avoid. Therefore, to ensure the robustness of our model, we will focus only on the diseases that have 10 or more samples.

This decision will reduce the number of classes we are predicting down to 6, making our model more manageable and potentially more accurate.ate.

[30]:
```
df = df[df.groupby('Disease')['Disease'].transform('size') >= 10]
```

[31]:
```
df.shape
```

[31]:
```
(83, 10)
```

Before we proceed with further analysis or model building, it's crucial to ensure the quality of our data. This involves checking for and handling missing values (NaNs) and duplicate entries. set.

Missing Vlaues: Missing data can lead to misleading results and reduce the statistical power of the model. Therefore, we need to check if our dataset contains any NaN values.

Duplicate Values: Duplicate entries can bias the analysis by over-representing certain observations. Hence, it's important to identify and remove any duplicates in our dataset.

[32]:
```
df.isna().sum()
```

[32]:
```
Disease                 0
Fever                   0
Cough                   0
Fatigue                 0
Difficulty Breathing    0
Age                     0
Gender                  0
Blood Pressure          0
Cholesterol Level       0
Outcome Variable        0
dtype: int64
```

[33]:
```
df.loc[df.duplicated()]
```

[33]:
```
          Disease Fever Cough Fatigue Difficulty Breathing  Age  Gender  \
4          Asthma   Yes   Yes      No                  Yes   25    Male
35         Asthma   Yes   Yes      No                  Yes   30  Female
59         Asthma    No   Yes     Yes                  Yes   35  Female
76         Asthma   Yes   Yes      No                  Yes   35    Male
123        Asthma   Yes   Yes      No                  Yes   40  Female
126        Asthma   Yes    No     Yes                  Yes   40    Male
182        Asthma   Yes   Yes      No                  Yes   45    Male
267  Osteoporosis   Yes    No     Yes                   No   55  Female
284  Osteoporosis    No   Yes      No                   No   60    Male
308        Stroke   Yes    No     Yes                   No   65  Female
339        Stroke    No   Yes      No                   No   70    Male
344        Stroke   Yes    No     Yes                   No   80  Female
346        Stroke   Yes    No     Yes                   No   85    Male
348        Stroke   Yes    No     Yes                   No   90  Female

    Blood Pressure Cholesterol Level Outcome Variable
4           Normal            Normal         Positive
35          Normal            Normal         Positive
59            High            Normal         Negative
76          Normal            Normal         Positive
123         Normal            Normal         Positive
```

```
126         Normal             High        Positive
182         Normal           Normal        Positive
267         Normal           Normal        Positive
284           High             High        Negative
308           High              Low        Negative
339         Normal             High        Positive
344           High             High        Positive
346           High             High        Positive
348           High             High        Positive
```

[34]:
```
df = df.drop_duplicates().reset_index(drop= True)
df.shape
```

[34]:
```
(69, 10)
```

Now that our data is cleaned and we've narrowed down our focus to diseases with 10 or more samples, let's visualize the distribution of these classes. Understanding the balance of classes is important as it can influence the performance of our machine learning model.

We'll use a pie chart for this purpose. Let's plot this chart and see how balanced our classes are.

[36]:
```
#Let's first find the unique diseases we are now considering
df['Disease'].unique()
```

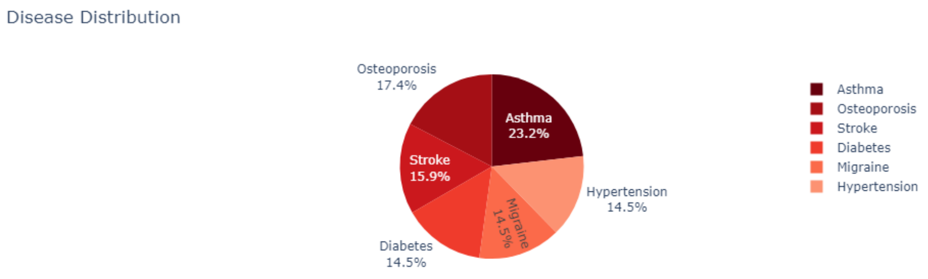
[36]:
```
array(['Asthma', 'Diabetes', 'Stroke', 'Migraine', 'Osteoporosis',
       'Hypertension'], dtype=object)
```

[38]:
```
!pip install plotly
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting plotly
  Downloading plotly-5.18.0-py3-none-any.whl.metadata (7.0 kB)
Collecting tenacity>=6.2.0 (from plotly)
  Downloading tenacity-8.2.3-py3-none-any.whl.metadata (1.0 kB)
Requirement already satisfied: packaging in
c:\users\sk731\appdata\roaming\python\python312\site-packages (from plotly)
(23.2)
Downloading plotly-5.18.0-py3-none-any.whl (15.6 MB)
   ---------------------------------------- 0.0/15.6 MB ? eta -:--:--
   ---------------------------------------- 0.0/15.6 MB 1.3 MB/s eta 0:00:13
   ---------------------------------------- 0.2/15.6 MB 2.6 MB/s eta 0:00:06
    --------------------------------------- 0.3/15.6 MB 3.2 MB/s eta 0:00:05
   - -------------------------------------- 0.5/15.6 MB 3.3 MB/s eta 0:00:05
   - -------------------------------------- 0.6/15.6 MB 3.4 MB/s eta 0:00:05
   -- ------------------------------------- 0.8/15.6 MB 3.6 MB/s eta 0:00:05
   -- ------------------------------------- 1.0/15.6 MB 3.5 MB/s eta 0:00:05
   --- ------------------------------------ 1.2/15.6 MB 3.5 MB/s eta 0:00:05
   --- ------------------------------------ 1.4/15.6 MB 3.7 MB/s eta 0:00:04
--- ------------------------------------ 1.5/15.6 MB 3.5 MB/s eta 0:00:05
---- ----------------------------------- 1.7/15.6 MB 3.6 MB/s eta 0:00:04
---- ----------------------------------- 1.9/15.6 MB 3.6 MB/s eta 0:00:04
----- ---------------------------------- 2.0/15.6 MB 3.5 MB/s eta 0:00:04
----- ---------------------------------- 2.2/15.6 MB 3.7 MB/s eta 0:00:04
----- ---------------------------------- 2.3/15.6 MB 3.7 MB/s eta 0:00:04
------ --------------------------------- 2.6/15.6 MB 3.7 MB/s eta 0:00:04
------- -------------------------------- 2.8/15.6 MB 3.7 MB/s eta 0:00:04
------- -------------------------------- 3.1/15.6 MB 3.8 MB/s eta 0:00:04
-------- ------------------------------- 3.5/15.6 MB 4.0 MB/s eta 0:00:04
-------- ------------------------------- 3.5/15.6 MB 3.9 MB/s eta 0:00:04
---------- ----------------------------- 3.9/15.6 MB 4.1 MB/s eta 0:00:03
----------- ---------------------------- 4.3/15.6 MB 4.3 MB/s eta 0:00:03
------------ --------------------------- 4.8/15.6 MB 4.6 MB/s eta 0:00:03
------------- -------------------------- 5.2/15.6 MB 4.7 MB/s eta 0:00:03
-------------- ------------------------- 5.6/15.6 MB 4.9 MB/s eta 0:00:03
--------------- ------------------------ 6.0/15.6 MB 5.0 MB/s eta 0:00:02
---------------- ----------------------- 6.5/15.6 MB 5.2 MB/s eta 0:00:02
----------------- ---------------------- 6.9/15.6 MB 5.4 MB/s eta 0:00:02
------------------ --------------------- 7.3/15.6 MB 5.5 MB/s eta 0:00:02
------------------- -------------------- 7.8/15.6 MB 5.7 MB/s eta 0:00:02
-------------------- ------------------- 8.2/15.6 MB 5.7 MB/s eta 0:00:02
--------------------- ------------------ 8.4/15.6 MB 5.7 MB/s eta 0:00:02
---------------------- ----------------- 8.8/15.6 MB 5.7 MB/s eta 0:00:02
----------------------- ---------------- 9.0/15.6 MB 5.8 MB/s eta 0:00:02
------------------------ --------------- 9.4/15.6 MB 5.8 MB/s eta 0:00:02
------------------------ --------------- 9.5/15.6 MB 5.8 MB/s eta 0:00:02
------------------------- -------------- 10.0/15.6 MB 5.8 MB/s eta 0:00:01
-------------------------- ------------- 10.3/15.6 MB 6.0 MB/s eta 0:00:01
--------------------------- ------------ 10.6/15.6 MB 6.1 MB/s eta 0:00:01
--------------------------- ------------ 10.9/15.6 MB 6.2 MB/s eta 0:00:01
---------------------------- ----------- 11.2/15.6 MB 6.3 MB/s eta 0:00:01
----------------------------- ---------- 11.4/15.6 MB 6.5 MB/s eta 0:00:01
------------------------------ --------- 11.8/15.6 MB 6.6 MB/s eta 0:00:01
------------------------------- -------- 12.1/15.6 MB 6.8 MB/s eta 0:00:01
------------------------------- -------- 12.5/15.6 MB 7.0 MB/s eta 0:00:01
-------------------------------- ------- 12.7/15.6 MB 7.2 MB/s eta 0:00:01
--------------------------------- ------ 13.0/15.6 MB 7.3 MB/s eta 0:00:01
---------------------------------- ----- 13.4/15.6 MB 7.2 MB/s eta 0:00:01
---------------------------------- ----- 13.7/15.6 MB 7.2 MB/s eta 0:00:01
----------------------------------- ---- 14.0/15.6 MB 7.4 MB/s eta 0:00:01
------------------------------------ --- 14.3/15.6 MB 7.3 MB/s eta 0:00:01
------------------------------------- -- 14.6/15.6 MB 7.3 MB/s eta 0:00:01
-------------------------------------- - 14.9/15.6 MB 7.2 MB/s eta 0:00:01
---------------------------------------  15.3/15.6 MB 7.1 MB/s eta 0:00:01
---------------------------------------  15.4/15.6 MB 7.1 MB/s eta 0:00:01
---------------------------------------  15.6/15.6 MB 6.9 MB/s eta 0:00:01
---------------------------------------- 15.6/15.6 MB 6.8 MB/s eta 0:00:00
```

```
Downloading tenacity-8.2.3-py3-none-any.whl (24 kB)
Installing collected packages: tenacity, plotly
Successfully installed plotly-5.18.0 tenacity-8.2.3
```

[39]:
```
import plotly.express as px

disease_counts = df['Disease'].value_counts().reset_index()
disease_counts.columns = ['Disease', 'Count']

fig = px.pie(disease_counts,
             values= 'Count',
             names= 'Disease',
             color_discrete_sequence= px.colors.sequential.Reds_r,
             title= 'Disease Distribution')

fig.update_traces(textinfo='percent+label')

fig.show()
```



From the pie chart, it's evident that our classes are imbalanced. Diseases like Hypertension, Diabetes, and Migraine have approximately 1.7 times fewer samples than Asthma. We have to handle class imbalance

But before we proceed with that, let's first process our categorical variables. This will allow us to perform a univariate analysis, which involves the examination of one variable at a time. This analysis can provide valuable insights into the distribution and characteristics of our variables.

dicc = {'Yes':1, 'No':0, 'Low':1, 'Normal':2, 'High':3, 'Positive':1, 'Negative':0, 'Male':0, 'Female': 1} def replace(x, dicc= dicc): if x in dicc: x = dicc[x] return x df = df.applymap(replace) df.head()

[41]:
```
df.dtypes
```

[41]:
```
Disease                    object
Fever                       int64
Cough                       int64
```

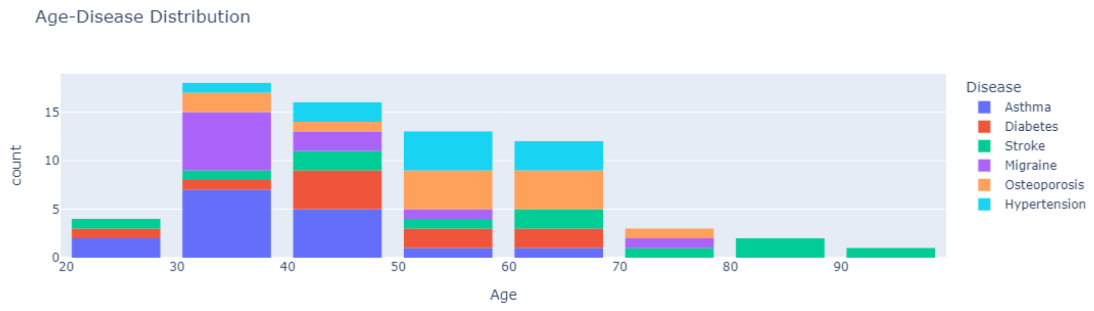
```
Fatigue                    int64
Difficulty Breathing       int64
Age                        int64
Gender                     int64
Blood Pressure             int64
Cholesterol Level          int64
Outcome Variable           int64
dtype: object
```

Having converted our categorical data into numerical format, we are now ready to perform univariate analysis. This analysis will help us understand the distribution of our variables and their individual impact on the disease prediction.

We'll start with the 'Age' variable. Age is a crucial factor in many diseases, and understanding its distribution and relationship with various diseases can provide valuable insights.

Following that, we'll examine the other variables one by one. Each of these variables - symptoms, gender, blood pressure, and cholesterol level - could potentially play a significant role in disease prediction. By analyzing them individually, we can gain a deeper understanding of their characteristics and importance.

[42]:
```python
fig = px.histogram(df,
             x = 'Age',
             title='Age-Disease Distribution',
             color= 'Disease'
                  )
fig.update_layout(bargap=0.2)

fig.show()
```



1 Univariate Analysis of 'Age' Variable

In this section, we present the findings from our univariate analysis of the 'Age' variable. The analysis reveals some intriguing patterns that could provide valuable insights into the relationship between age and certain diseases.

1.1 Age and Stroke

If the age is greater than 80, the disease is likely to be a stroke. This observation aligns with the general understanding that the risk of stroke increases with age.

1.2 Migraine and Hypertension in Age Group 20-30

Migraine and Hypertension are not present in ages between 20 and 30. This absence suggests that these conditions may be more prevalent in older age groups.

1.3 Age and Hypertension/Osteoporosis

Hypertension and Osteoporosis appear more frequently as the age increases, indicating a potential correlation between these diseases and age.

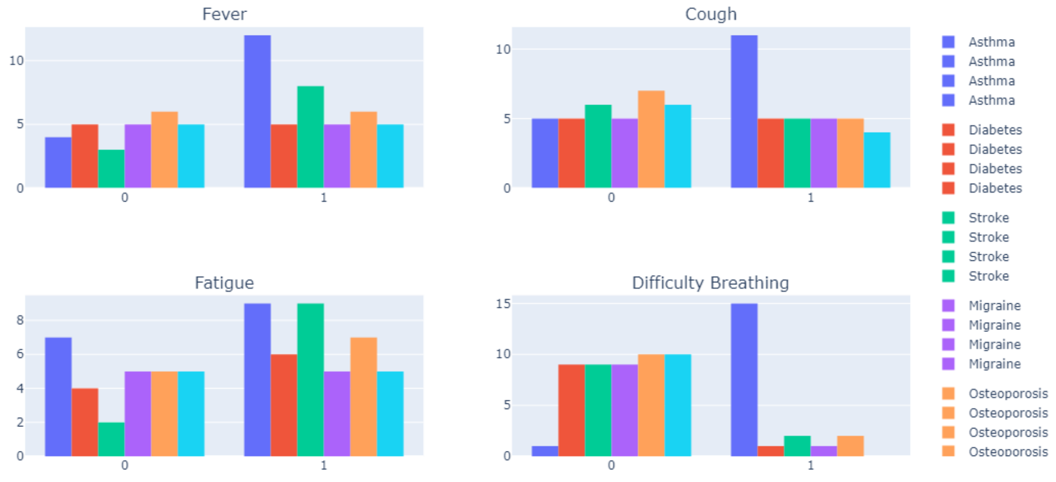
1.4 Age as a Predictor for Diseases

These observations collectively suggest that age is a valuable feature for predicting certain diseases. However, it's crucial to note that our dataset has limited samples, especially for ages greater than 80. This limitation could pose challenges when predicting new values in this age range.

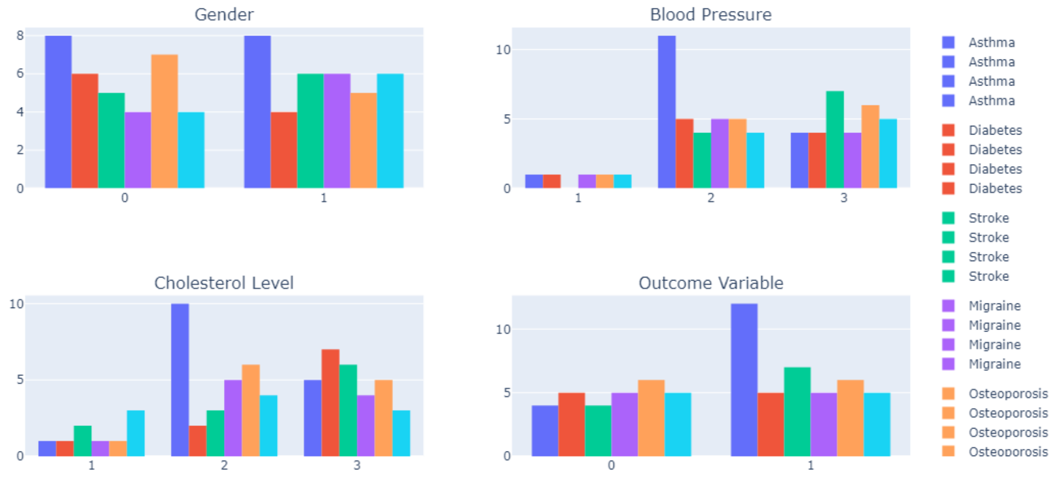
1.5 Next Steps

Next, our analysis will shift towards examining how other variables interact with different diseases. This exploration will help us understand their potential as predictors and identify any patterns or correlations that may exist. relations.relations.

[43]:
```
import plotly.subplots as sp

def subplots(df, columns):
    fig = sp.make_subplots(rows=2, cols=2, subplot_titles= columns)
    for idx, column in enumerate(columns):
        i = idx // 2 + 1
        j = idx % 2 + 1
        fig_express = px.histogram(df, x=column, title=column + '-Disease␣
 ↪Distribution', color='Disease')
        for trace in fig_express.data:
            fig.add_trace(trace, row=i, col=j)
    fig.update_layout(height=600, width=800, title_text="")
    fig.show()
subplots(df, df.columns[1:5])
```



```
[44]: subplots(df, df.columns[6:])
```



2 Variable Analysis and Correlation

Upon visual inspection of the other variables, we can observe significant differences in disease prediction based on each feature's values. For instance, whether a person has high, normal, or low cholesterol levels can significantly influence the prediction of a disease. This is consistent with real-world observations where these variables often vary among different diseases.

2.1 Key Insights

Some valuable insights we can glean from this analysis include:

- A person with low blood pressure does not have a stroke. This could be a crucial factor in stroke prediction.
- Fatigue, cholesterol level, and blood pressure are the features that show the most variation among different values. These could potentially be strong predictors in our model.

These observations underscore the importance of these variables in predicting diseases.

2.2 Correlation Analysis

Next, let's examine how these variables correlate with each other. Understanding the relationships between different variables can help us identify patterns and potential multicollinearity, which could influence our model's performance.

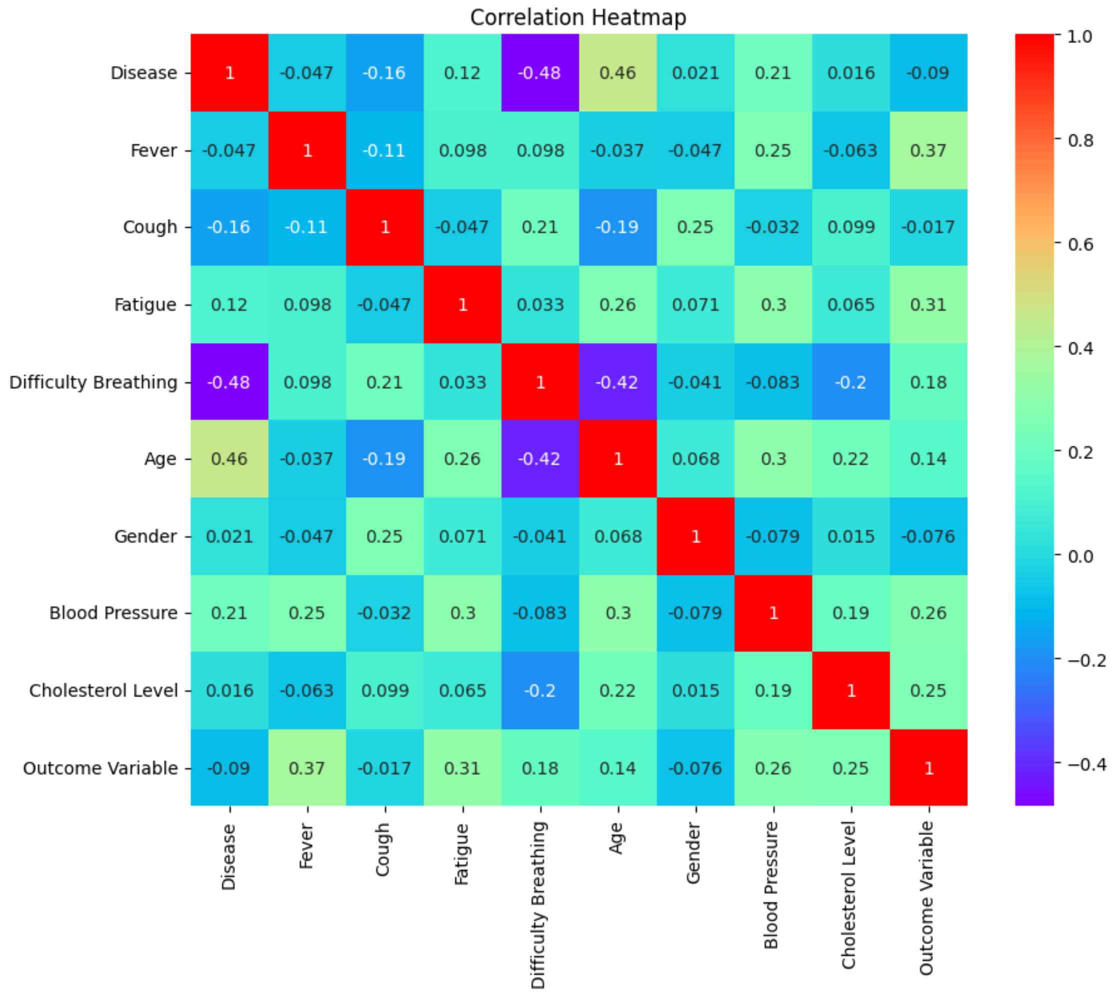
To facilitate this analysis, we'll use the LabelEncoder from sklearn to convert our categorical variables into niables for correlation analysis on analysis.

[45]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df.Disease = le.fit_transform(df.Disease)
df.head()
```

[45]:
```
   Disease  Fever  Cough  Fatigue  Difficulty Breathing  Age  Gender  \
0        0      1      1        0                     1   25       0
1        0      1      0        0                     1   28       0
2        1      0      0        0                     0   29       0
3        5      1      1        1                     1   29       1
4        3      1      0        0                     0   30       1

   Blood Pressure  Cholesterol Level  Outcome Variable
0               2                  2                 1
1               3                  2                 1
2               1                  2                 0
3               2                  2                 1
4               2                  2                 0
```

[46]:
```
# Compute the correlation matrix
corr_matrix = df.corr()

# Create a heatmap using seaborn
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='rainbow')
plt.title('Correlation Heatmap')
```

```
[46]: Text(0.5, 1.0, 'Correlation Heatmap')
```



	Disease	Fever	Cough	Fatigue	Difficulty Breathing	Age	Gender	Blood Pressure	Cholesterol Level	Outcome Variable
Disease	1	-0.047	-0.16	0.12	-0.48	0.46	0.021	0.21	0.016	-0.09
Fever	-0.047	1	-0.11	0.098	0.098	-0.037	-0.047	0.25	-0.063	0.37
Cough	-0.16	-0.11	1	-0.047	0.21	-0.19	0.25	-0.032	0.099	-0.017
Fatigue	0.12	0.098	-0.047	1	0.033	0.26	0.071	0.3	0.065	0.31
Difficulty Breathing	-0.48	0.098	0.21	0.033	1	-0.42	-0.041	-0.083	-0.2	0.18
Age	0.46	-0.037	-0.19	0.26	-0.42	1	0.068	0.3	0.22	0.14
Gender	0.021	-0.047	0.25	0.071	-0.041	0.068	1	-0.079	0.015	-0.076
Blood Pressure	0.21	0.25	-0.032	0.3	-0.083	0.3	-0.079	1	0.19	0.26
Cholesterol Level	0.016	-0.063	0.099	0.065	-0.2	0.22	0.015	0.19	1	0.25
Outcome Variable	-0.09	0.37	-0.017	0.31	0.18	0.14	-0.076	0.26	0.25	1

From the correlation graph, we can observe that none of the variables strongly correlate with the 'Disease' variable. The most correlated variables are 'Age' and 'Difficulty Breathing', but even these only score 0.4 and -0.4 respectively.

In situations where we have multiple variables with low correlation scores, machine learning can be a viable alternative for prediction tasks. However, it's important to note that machine learning algorithms, especially deep learning ones, typically require large amounts of data to perform optimally.

In our case, we have only 69 data points, which is relatively small. Furthermore, we are dealing with a multi-class problem with few examples for each disease, which adds to the complexity.

Given these constraints, we will try two machine learning algorithms that can perform well without a lot of data: K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM). We will fine-tune these models and select the one that performs best.

Let's proceed with data preprocessing and fit our data into these algorithms.

2.3 Model Selection and Analysis

[47]:
```
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline

X = df.drop(['Disease'], axis= 1).values
y = df.Disease.values
```

[48]:
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size= 0.4,␣
 ↪shuffle= True, stratify= y, random_state=30)
X_val, X_test, y_val, y_test = train_test_split(X_val, y_val, test_size= 0.5,␣
 ↪shuffle= True, stratify= y_val, random_state=30)
```

[49]:
```
svc_pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC(class_weight=␣
 ↪'balanced'))])
knn_pipe = Pipeline([('scaler', StandardScaler()), ('knn',␣
 ↪KNeighborsClassifier())])
```

[50]:
```
svc_pipe.fit(X_train, y_train)
knn_pipe.fit(X_train, y_train)
```

[50]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('knn', KNeighborsClassifier())])
```

[51]:
```
ysvc_pred = svc_pipe.predict(X_val)
yknn_pred = knn_pipe.predict(X_val)
```

Given our problem of multi-class classification with imbalanced classes, the F1 score (macro-averaged) is an appropriate choice. The F1 score is the harmonic mean of precision and recall, and it gives a better measure of the incorrectly classified cases than the accuracy metric.

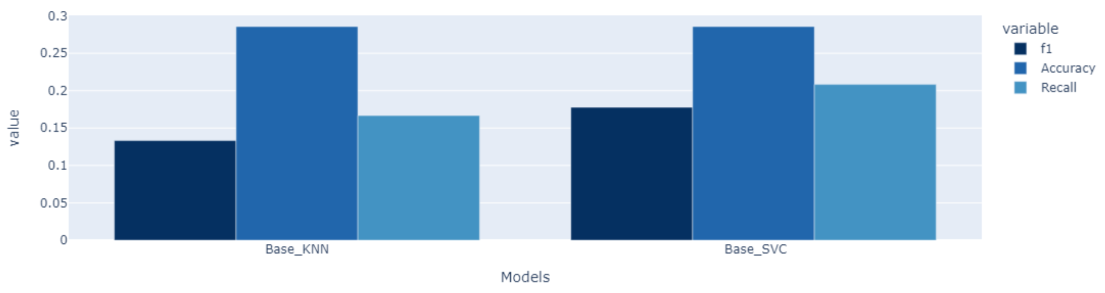
The macro-averaged F1 score calculates the F1 score for each class independently and then takes the average. This treats all classes equally, regardless of their imbalance, which is exactly what we need for our problem.

Our goal is to maximize this F1 score.

[52]:
```
from sklearn.metrics import f1_score, accuracy_score, recall_score
models = ['Base_KNN', ' Base_SVC']

f1 = [f1_score(y_val, yknn_pred, average= 'macro', zero_division= 0),␣
 ↪f1_score(y_val, ysvc_pred, average= 'macro', zero_division= 0)]
accuracy = [accuracy_score(y_val, yknn_pred), accuracy_score(y_val, ysvc_pred)]
recall = [recall_score(y_val, yknn_pred, average= 'macro'), recall_score(y_val,␣
 ↪ysvc_pred, average= 'macro')]
```

```
metrics_df = pd.DataFrame({'Models': models, 'f1': f1, 'Accuracy': accuracy,␣
↪'Recall': recall})
#metrics_df = metrics_df.melt(id_vars='Models', var_name='metric',␣
↪value_name='score')
```

[73]:
```
fig = px.bar(metrics_df, x='Models', y= ['f1', 'Accuracy', 'Recall'], barmode=␣
↪'group', color_discrete_sequence= px.colors.sequential.RdBu_r)
plt.savefig('picture4.jpg')
fig.show( )
```



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

After training our K-Nearest Neighbors (K-NN) and Support Vector Machines (SVM) models, we observe that SVM significantly outperforms K-NN in terms of the macro-averaged F1 score.

Given this performance difference, it makes sense to focus our efforts on the SVM model. We will proceed with fine-tuning this model to see if we can further improve the F1 score.

Fine Tuning

[56]:
```
from sklearn.metrics import make_scorer
f1_scorer = make_scorer(f1_score, average='macro', zero_division=0)
parameters = {
    'svc__C': [0.1, 1, 10],
    'svc__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'svc__gamma': ['scale', 'auto'],
    'svc__shrinking': [True, False],
    'svc__decision_function_shape': ['ovo', 'ovr']
}

grid_search = GridSearchCV(svc_pipe, parameters, cv=5, scoring= f1_scorer)

grid_search.fit(X_train, y_train)
```

```
print("Best Score: ", grid_search.best_score_)
print("Best Params: ", grid_search.best_params_)

best_clf = grid_search.best_estimator_
```

```
Best Score:  0.32111111111111107
Best Params:  {'svc__C': 1, 'svc__decision_function_shape': 'ovo', 'svc__gamma':
'scale', 'svc__kernel': 'rbf', 'svc__shrinking': True}
```

```
[57]: results = pd.DataFrame(grid_search.cv_results_)
results = results[['param_svc__C', 'param_svc__kernel', 'param_svc__gamma',␣
 ↪'param_svc__shrinking',
                   'param_svc__decision_function_shape', 'mean_test_score']]
results
```

```
[57]:    param_svc__C param_svc__kernel param_svc__gamma param_svc__shrinking  \
0             0.1            linear            scale                 True
1             0.1            linear            scale                False
2             0.1               rbf            scale                 True
3             0.1               rbf            scale                False
4             0.1              poly            scale                 True
..            …                 …                …                    …
91             10               rbf             auto                False
92             10              poly             auto                 True
93             10              poly             auto                False
94             10           sigmoid             auto                 True
95             10           sigmoid             auto                False

   param_svc__decision_function_shape  mean_test_score
0                                 ovo         0.164444
1                                 ovo         0.164444
2                                 ovo         0.079899
3                                 ovo         0.079899
4                                 ovo         0.074074
..                                …               …
91                                ovr         0.266667
92                                ovr         0.215556
93                                ovr         0.215556
94                                ovr         0.256667
95                                ovr         0.256667

[96 rows x 6 columns]
```

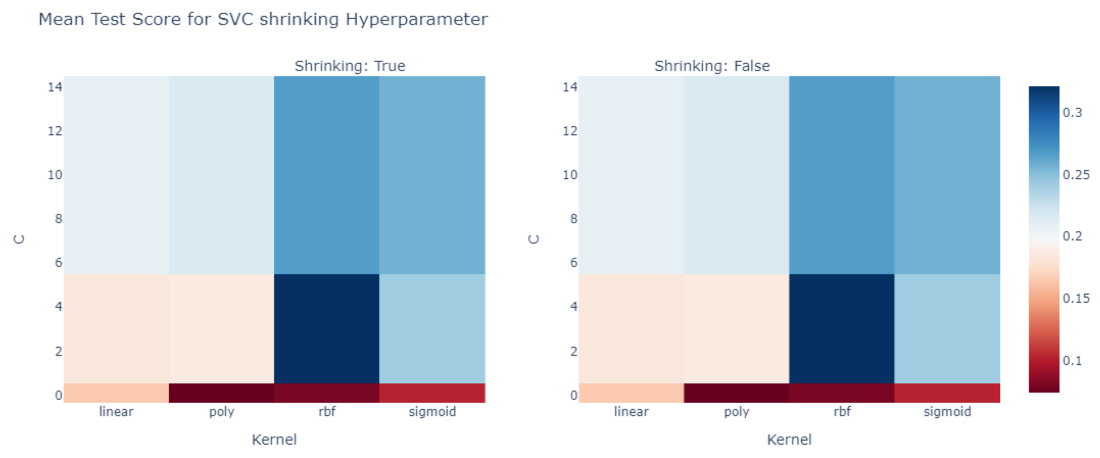
In addition to the kernel and C parameters, we will analyze the significance of other hyperparameters in achieving the highest F1 score. While the kernel and C parameters are known to have a significant impact on SVM performance, it's important to consider the influence of other hyperparameters as well.

We will specifically investigate the impact of hyperparameters such as the shrinking and the decision function shape.

Analyzing the relationship between these hyperparameters and the F1 score will provide us with a more comprehensive understanding of the model's behavioscore.

Let's proceed with the hyperparameter analysis and determine the optimal values for achieving the highest F1 score.

[70]:
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go

grouped = results.groupby(['param_svc__C', 'param_svc__kernel',␣
↪'param_svc__shrinking'])['mean_test_score'].mean().reset_index()

pivot_table_true = grouped[grouped['param_svc__shrinking'] == True].pivot_table(
    index='param_svc__C', columns='param_svc__kernel', values='mean_test_score'
)

pivot_table_false = grouped[grouped['param_svc__shrinking'] == False].
↪pivot_table(
    index='param_svc__C', columns='param_svc__kernel', values='mean_test_score'
)

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Heatmap(z=pivot_table_true.values, x=pivot_table_true.columns,
               y=pivot_table_true.index, colorscale='RdBu', showscale=False),
    row=1, col=1
)

fig.add_trace(
    go.Heatmap(z=pivot_table_false.values, x=pivot_table_false.columns,
               y=pivot_table_false.index, colorscale='RdBu'),
    row=1, col=2
)

fig.update_layout(
    height=500, width=1000, title_text="Mean Test Score for SVC shrinking␣
↪Hyperparameter",
    annotations=[
        go.layout.Annotation(
            text="Shrinking: True",
            xref="paper", yref="paper",
            x=0.25, y=1.07, showarrow=False,
            font=dict(size=14,)
        ),
        go.layout.Annotation(
            text="Shrinking: False",
            xref="paper", yref="paper",
            x=0.75, y=1.07, showarrow=False,
            font=dict(size=14,)
        )
    ]
)
fig.update_xaxes(title_text="Kernel", row=1, col=1)
fig.update_xaxes(title_text="Kernel", row=1, col=2)
fig.update_yaxes(title_text="C", row=1, col=1)
fig.update_yaxes(title_text="C", row=1, col=2)
plt.savefig('picture1.jpg')

fig.show()
```

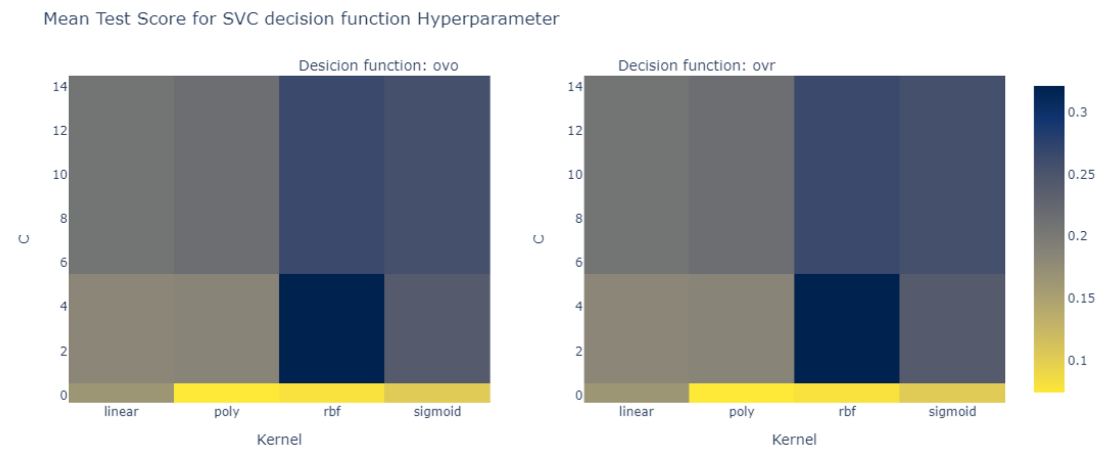


```
<Figure size 640x480 with 0 Axes>
```

Here we can see that there is no meaningful difference between using the shrinking hyperparameter or not. Here, the kernel and C variants determine the mean test score with no differences, whether the parameter is true or false. Now let's see if the decision function shape makes a difference or not.

```
[71]: grouped = results.groupby(['param_svc__C', 'param_svc__kernel',␣
        ↪'param_svc__decision_function_shape'])['mean_test_score'].mean().
        ↪reset_index()

      pivot_table_true = grouped[grouped['param_svc__decision_function_shape'] ==␣
        ↪'ovo'].pivot(
          index='param_svc__C', columns='param_svc__kernel', values='mean_test_score')
```

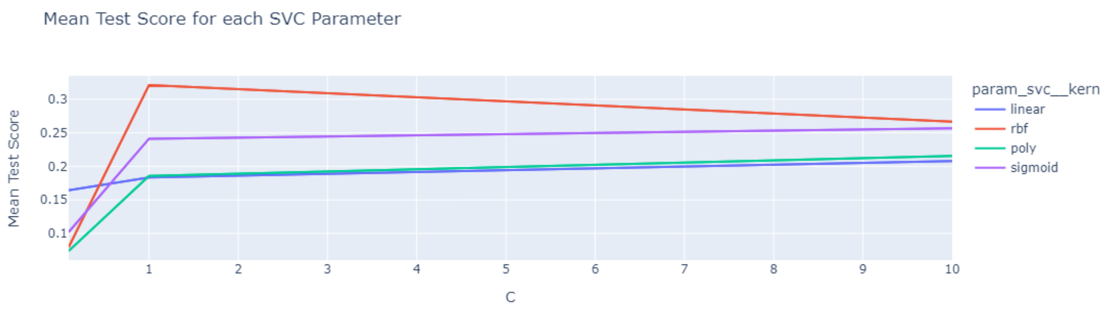
```
pivot_table_false = grouped[grouped['param_svc__decision_function_shape'] ==␣
 ↪'ovr'].pivot(
    index='param_svc__C', columns='param_svc__kernel', values='mean_test_score')

fig = make_subplots(rows=1, cols=2)

fig.add_trace(
    go.Heatmap(z=pivot_table_true.values, x=pivot_table_true.columns,
               y=pivot_table_true.index, colorscale= px.colors.sequential.
 ↪Cividis_r, showscale=False),
    row=1, col=1
)

fig.add_trace(
    go.Heatmap(z=pivot_table_false.values, x=pivot_table_false.columns,
               y=pivot_table_false.index, colorscale= px.colors.sequential.
 ↪Cividis_r),
    row=1, col=2
)

fig.update_layout(
    height=500, width=1000, title_text="Mean Test Score for SVC decision␣
 ↪function Hyperparameter",
    annotations=[
        go.layout.Annotation(
            text="Desicion function: ovo",
            xref="paper", yref="paper",
            x=0.25, y=1.07, showarrow=False,
            font=dict(size=14,)
        ),
        go.layout.Annotation(
            text="Decision function: ovr",
            xref="paper", yref="paper",
            x=0.75, y=1.07, showarrow=False,
            font=dict(size=14,)
        )
    ]
)
fig.update_xaxes(title_text="Kernel", row=1, col=1)
fig.update_xaxes(title_text="Kernel", row=1, col=2)
fig.update_yaxes(title_text="C", row=1, col=1)
fig.update_yaxes(title_text="C", row=1, col=2)
plt.savefig('picture2.jpg')

fig.show()
```



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

Here we can observe a similar result as with the shrinking hyperparameter - there is no effect on the score when we use OVO (One-vs-One) or OVR (One-vs-Rest) for the decision function. Now, let's plot how the score changes when we modify the kernel and the C parameter.

```
[72]: fig = px.line(results, x="param_svc__C", y="mean_test_score",
                   color="param_svc__kernel",
                   line_group="param_svc__shrinking",
                   hover_name="param_svc__decision_function_shape",
                   labels={"mean_test_score": "Mean Test Score", "param_svc__C":␣
       ↪"C"},
                   title="Mean Test Score for each SVC Parameter")
      plt.savefig('picture3.jpg')
      fig.show()
```



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

We can observe a significant effect of the kernel and C parameter on the mean test scores. The range of scores varies widely, ranging from 0.06 to 0.255. This emphasizes the importance of these parameters in the SVC (Support Vector Classifier).

Lastly, we will use the best model to predict our test data and report the final results. Additionally, we will examine the samples that were classified incorrectly. Let's proceed with these tasks.

[68]:
```
y_pred_test = best_clf.predict(X_test)
print('Test score with best model: ', f1_score(y_test, y_pred_test, average=␣
 ↪'macro', zero_division= 0))
```

```
Test score with best model:  0.3611111111111111
```

[69]:
```
import pandas as pd

X_test_df = pd.DataFrame(X_test)
df = pd.DataFrame({'actual': le.inverse_transform(y_test), 'predicted': le.
 ↪inverse_transform(y_pred_test)})
df = df.set_index(X_test_df.index)
df = pd.concat([X_test_df, df], axis=1)
misclassified = df[df['actual'] != df['predicted']]

print(misclassified)
```

```
    0  1  2  3   4  5  6  7  8        actual     predicted
0   0  0  1  0  65  0  2  3  0      Diabetes  Osteoporosis
1   1  1  1  0  35  1  3  2  0  Hypertension       Migraine
2   1  0  1  0  55  1  2  2  1  Osteoporosis  Hypertension
4   1  0  1  0  45  1  3  3  1      Diabetes  Hypertension
6   0  0  1  0  45  0  2  2  0        Stroke  Osteoporosis
8   0  1  1  0  35  0  3  3  1      Migraine  Osteoporosis
10  1  1  0  0  52  0  2  1  0  Hypertension  Osteoporosis
11  1  0  1  1  55  0  3  1  1  Osteoporosis      Diabetes
13  0  0  0  1  31  0  1  2  0  Osteoporosis      Diabetes
```

Here, we can observe that this model performs well in predicting asthma cases but performs poorly in predicting other conditions in general. This suggests that we can use this model with a one-vs-all approach, where one class represents asthma, and the model can be used as a second opinion to determine if a person has asthma or not.

It's important to note that asthma is the most frequent class in the training data used for this model. However, even with this imbalance, we have a limited number of samples. Therefore, we can consider implementing data augmentation techniques to see if the model can improve its accuracy in predicting the other diseases.

By augmenting the data, we can generate additional samples using techniques such as rotation, scaling, or adding noise. This can potentially help the model generalize better and improve its performance in predicting the less frequent diseases.

Overall, it is crucial to explore different approaches, such as data augmentation, to enhance the model's accuracy and make more accurate predictions for a wider range of conditions.

2.4 Comment

In conclusion, this project explored the prediction of diseases using basic medical information. The model achieved an F1 macro average score of 0.3611 for the six classes, indicating room for improvement in accurately predicting diseases with basic medical information alone.

While the current model's performance may be limited, there are opportunities for further exploration and enhancement. Collecting diverse data, employing feature engineering techniques, exploring alternative algorithms, fine-tuning hyperparameters, and seeking domain expertise can contribute to improving the model's accuracy and reliability.