Asthama Disease Prediction using Machine Learning !!!!

Importing necessary Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Reading our DATA

In [3]:
```
df = pd.read_csv('asthama.csv')
```

EDA

In [4]:
```
df.head()
```

Out[4]:

	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	Pains	Nasal-Congestion	Runny-Nose	None_
0	1	1	1	1	0	1	1	1	
1	1	1	1	1	0	1	1	1	
2	1	1	1	1	0	1	1	1	
3	1	1	1	1	0	1	1	1	

4	1	1	1	1	0	1	1	1

In [5]:
```
df.tail()
```

Out[5]:

	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	Pains	Nasal-Congestion	Runny-Nose
316795	0	0	0	0	1	0	0	0
316796	0	0	0	0	1	0	0	0
316797	0	0	0	0	1	0	0	0
316798	0	0	0	0	1	0	0	0
316799	0	0	0	0	1	0	0	0

In [6]:
```
df.shape
```

Out[6]:
```
(316800, 19)
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing', 'Sore-Throa
t',
       'None_Sympton', 'Pains', 'Nasal-Congestion', 'Runny-Nose',
       'None_Experiencing', 'Age_0-9', 'Age_10-19', 'Age_20-24', 'Age_25-
59',
       'Age_60+', 'Gender_Female', 'Gender_Male', 'Severity_Mild',
       'Severity_Moderate', 'Severity_None'],
      dtype='object')
```

In [8]:
```
df.duplicated().sum()
```

Out[8]:
```
311040
```

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
Tiredness                 0
Dry-Cough                 0
Difficulty-in-Breathing   0
Sore-Throat               0
None_Sympton              0
Pains                     0
Nasal-Congestion          0
Runny-Nose                0
None_Experiencing         0
Age_0-9                   0
Age_10-19                 0
Age_20-24                 0
Age_25-59                 0
Age_60+                   0
Gender_Female             0
Gender_Male               0
Severity_Mild             0
Severity_Moderate         0
Severity_None             0
dtype: int64
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 316800 entries, 0 to 316799
Data columns (total 19 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Tiredness                 316800 non-null  int64
 1   Dry-Cough                 316800 non-null  int64
 2   Difficulty-in-Breathing   316800 non-null  int64
 3   Sore-Throat               316800 non-null  int64
 4   None_Sympton              316800 non-null  int64
 5   Pains                     316800 non-null  int64
 6   Nasal-Congestion          316800 non-null  int64
 7   Runny-Nose                316800 non-null  int64
 8   None_Experiencing         316800 non-null  int64
 9   Age_0-9                   316800 non-null  int64
 10  Age_10-19                 316800 non-null  int64
 11  Age_20-24                 316800 non-null  int64
 12  Age_25-59                 316800 non-null  int64
 13  Age_60+                   316800 non-null  int64
 14  Gender_Female             316800 non-null  int64
 15  Gender_Male               316800 non-null  int64
 16  Severity_Mild             316800 non-null  int64
 17  Severity_Moderate         316800 non-null  int64
 18  Severity_None             316800 non-null  int64
dtypes: int64(19)
memory usage: 45.9 MB
```

In [11]:
```
df.describe()
```

Out[11]:

	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	
count	316800.000000	316800.000000	316800.000000	316800.000000	316800.000000	316800
mean	0.500000	0.562500	0.500000	0.312500	0.062500	0
std	0.500001	0.496079	0.500001	0.463513	0.242062	0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0
50%	0.500000	1.000000	0.500000	0.000000	0.000000	0
75%	1.000000	1.000000	1.000000	1.000000	0.000000	1
max	1.000000	1.000000	1.000000	1.000000	1.000000	1

We have only 1's and 0's

In [12]:
```
df.nunique()
```

Out[12]:
```
Tiredness                 2
Dry-Cough                 2
Difficulty-in-Breathing   2
Sore-Throat               2
None_Sympton              2
Pains                     2
Nasal-Congestion          2
Runny-Nose                2
None_Experiencing         2
Age_0-9                   2
Age_10-19                 2
Age_20-24                 2
Age_25-59                 2
Age_60+                   2
```

```
Gender_Female              2
Gender_Male                2
Severity_Mild              2
Severity_Moderate          2
Severity_None              2
dtype: int64
```

DATA VISUALIZATION COMES INTO THE PICTURE . .

.

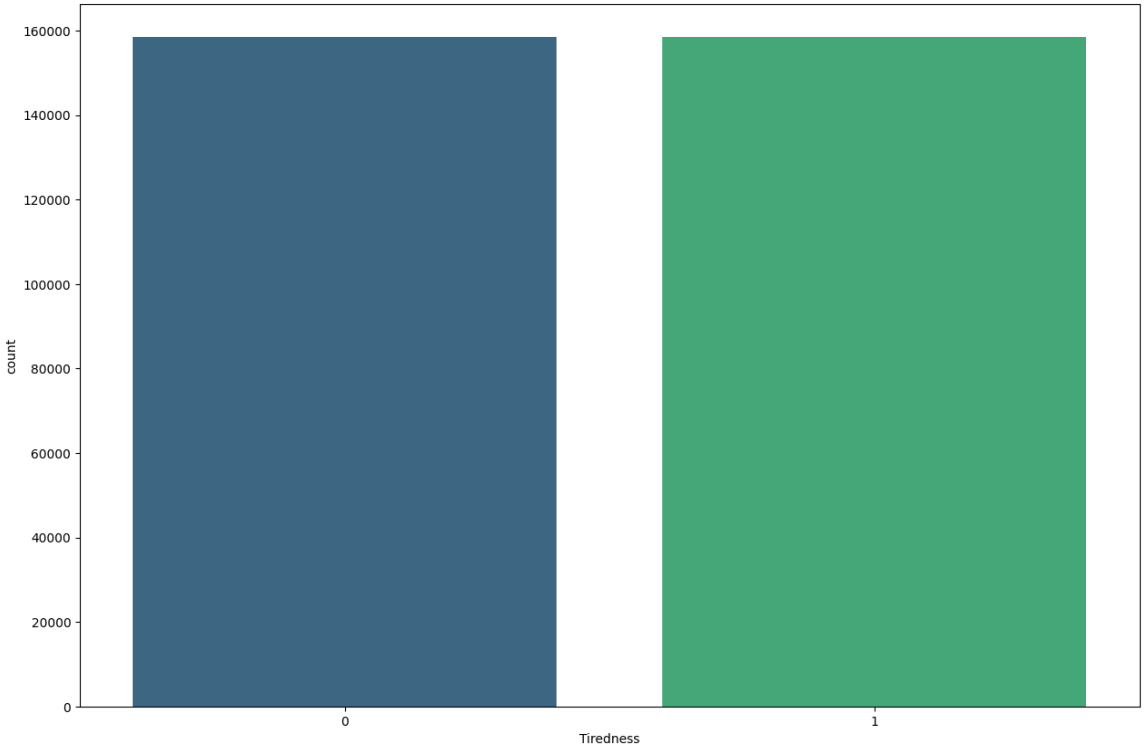
.

Count plot for individual coloums

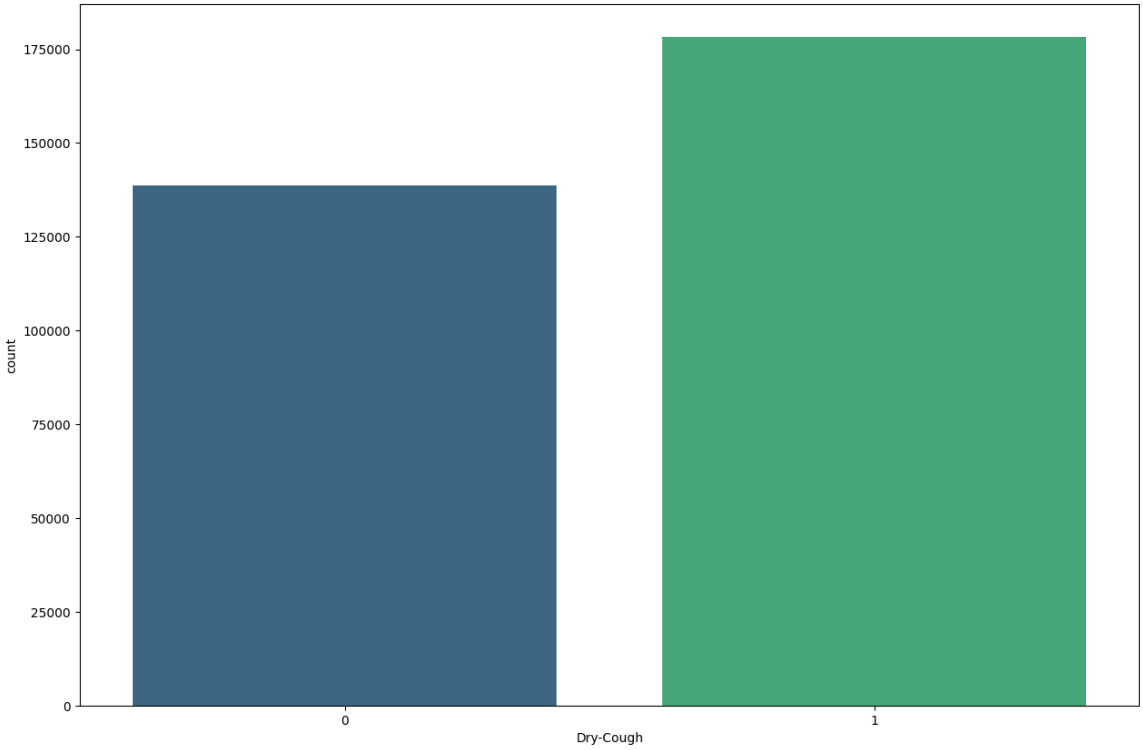
.

In [13]:
```
for i in df.columns:
    print("Count Plot for " , i)
    plt.figure(figsize=(15,10))
    sns.countplot(x = df[i] , data = df , palette = "viridis")
    plt.show()
    print('\n')
```

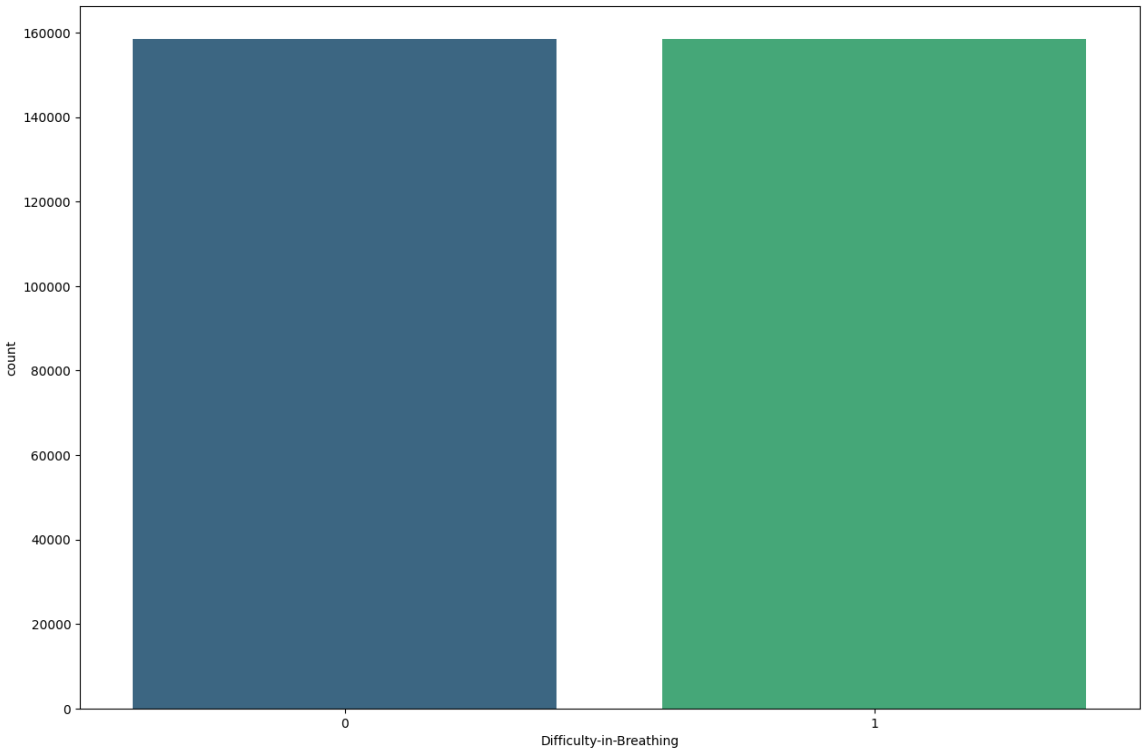
```
Count Plot for  Tiredness
```



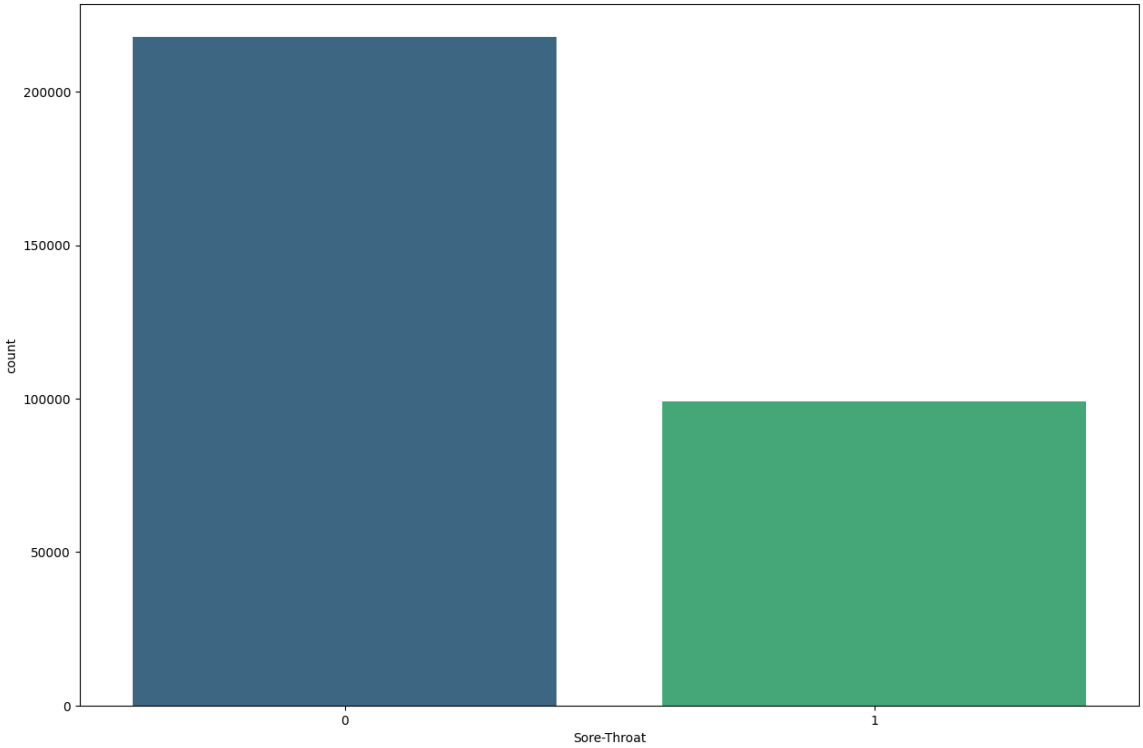

```
Count Plot for  Dry-Cough
```



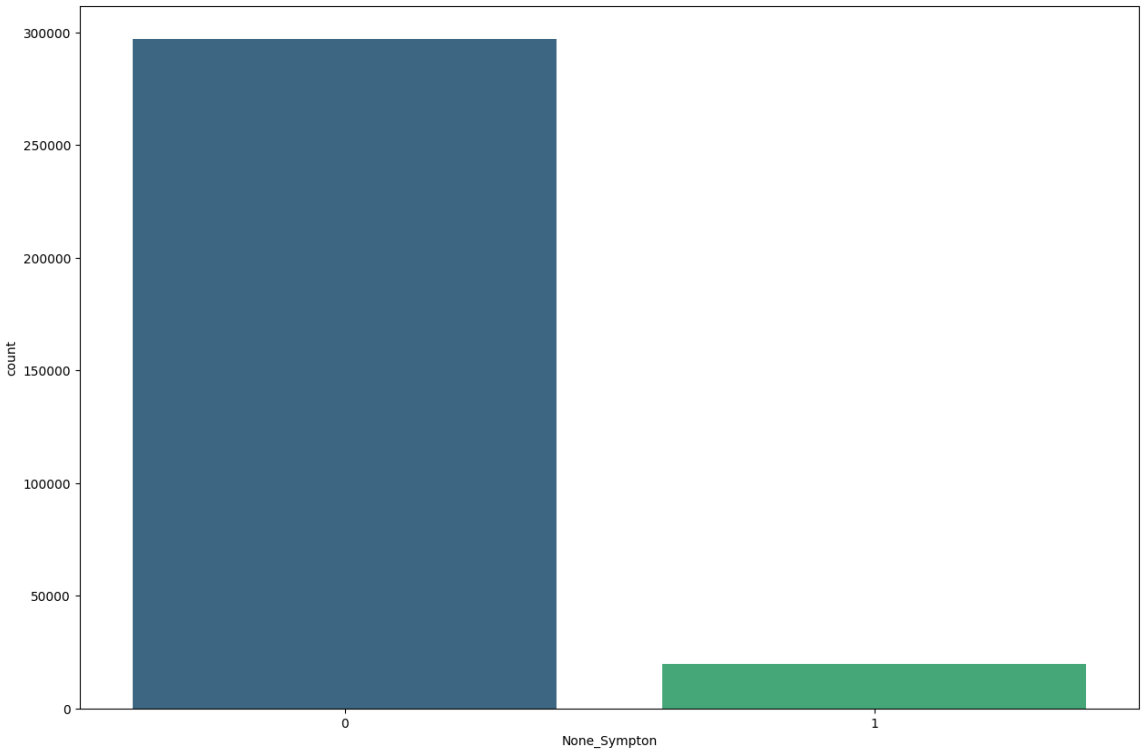
Count Plot for Difficulty-in-Breathing



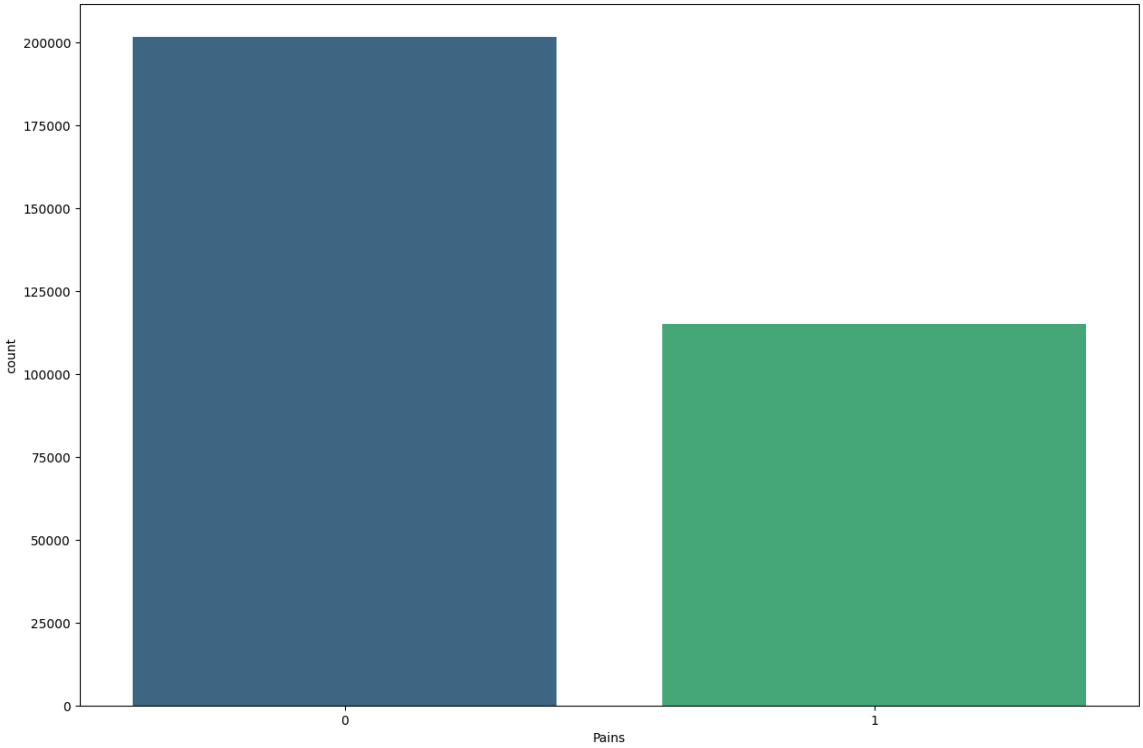
Count Plot for Sore-Throat



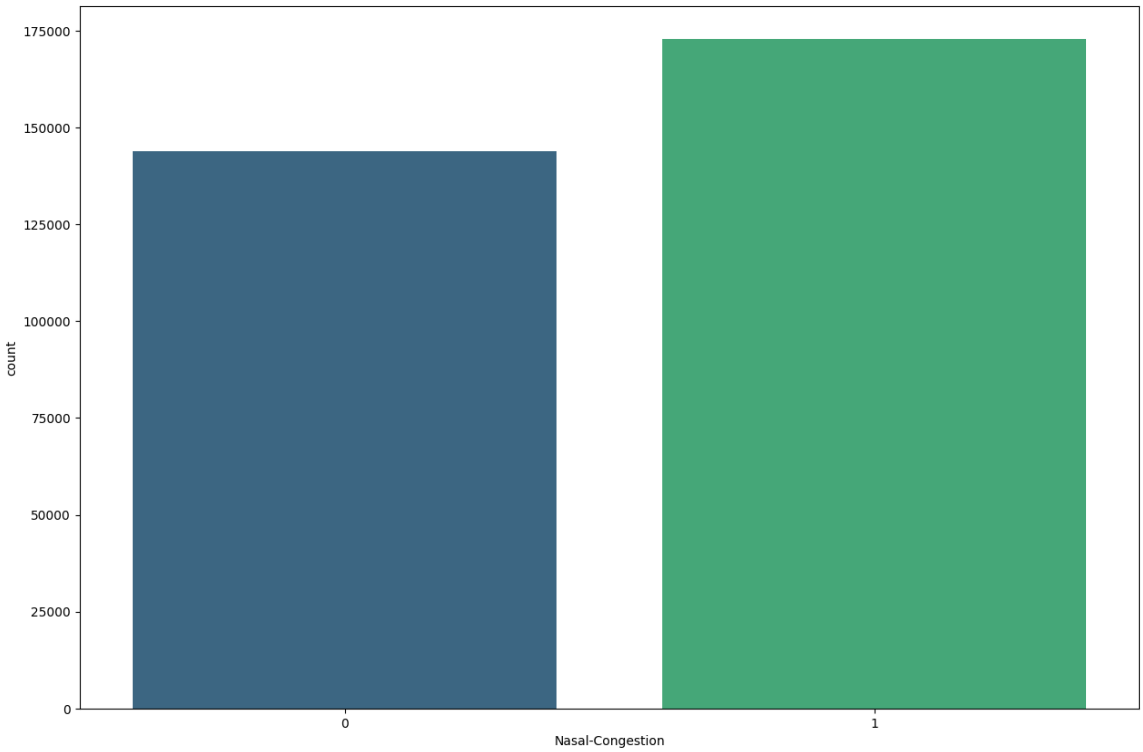
Count Plot for None_Sympton



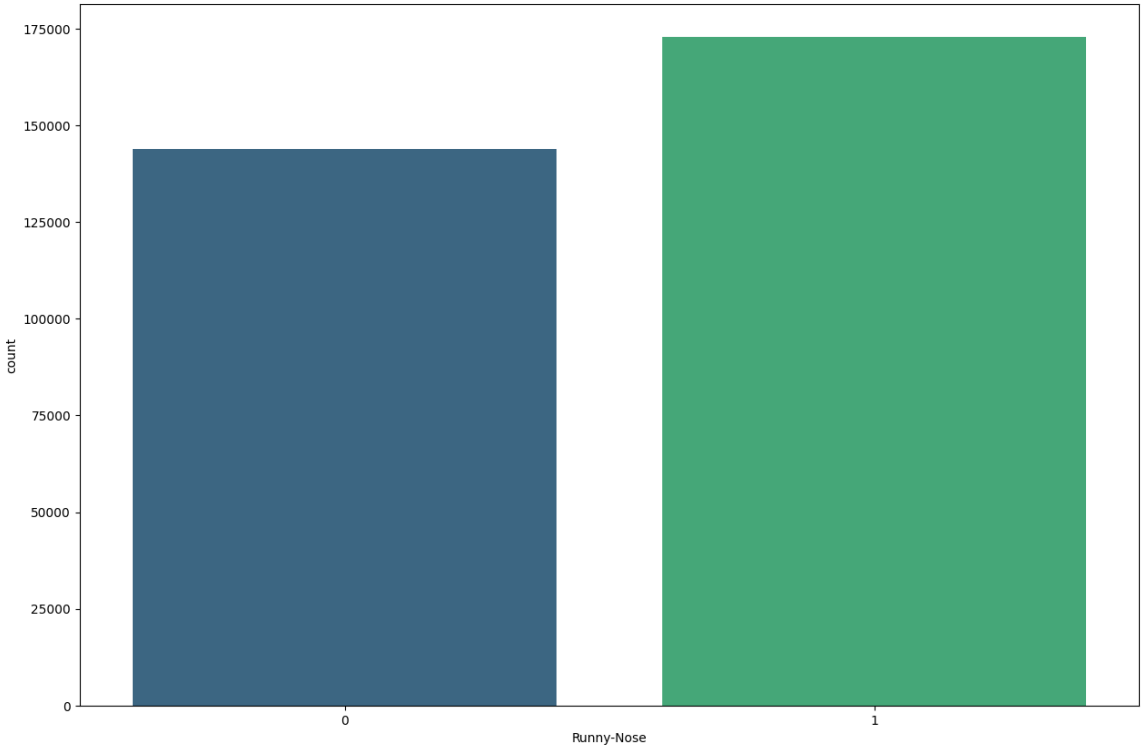
Count Plot for Pains



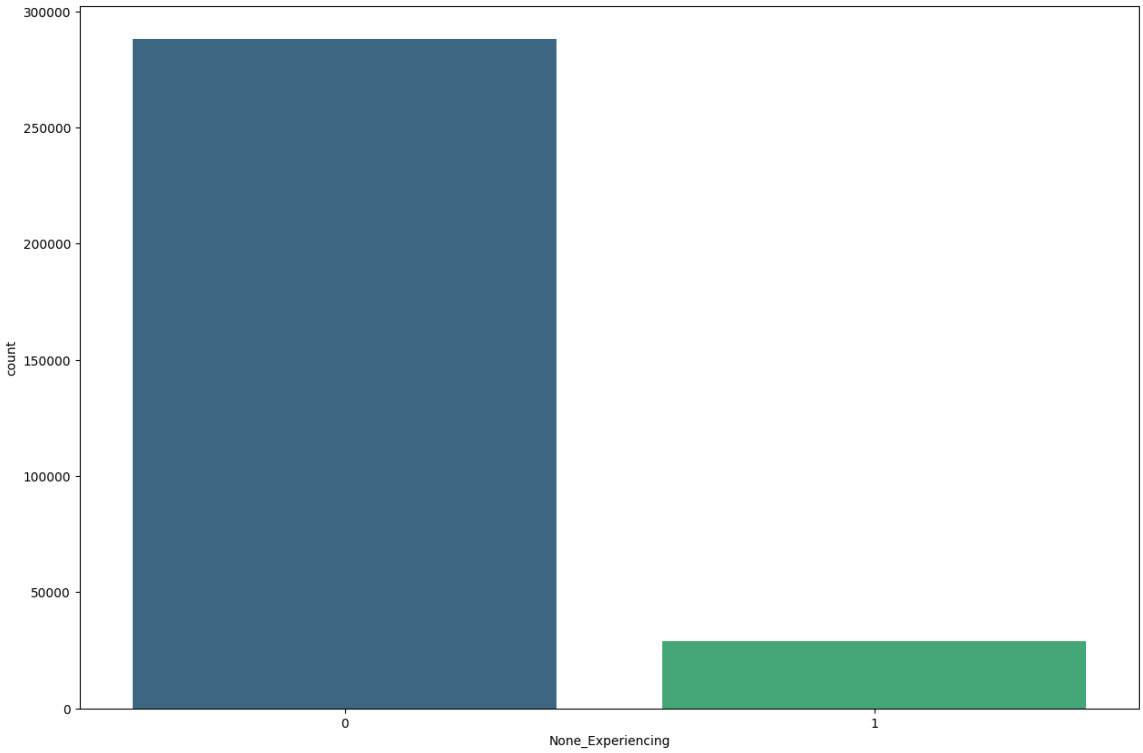
Count Plot for Nasal-Congestion



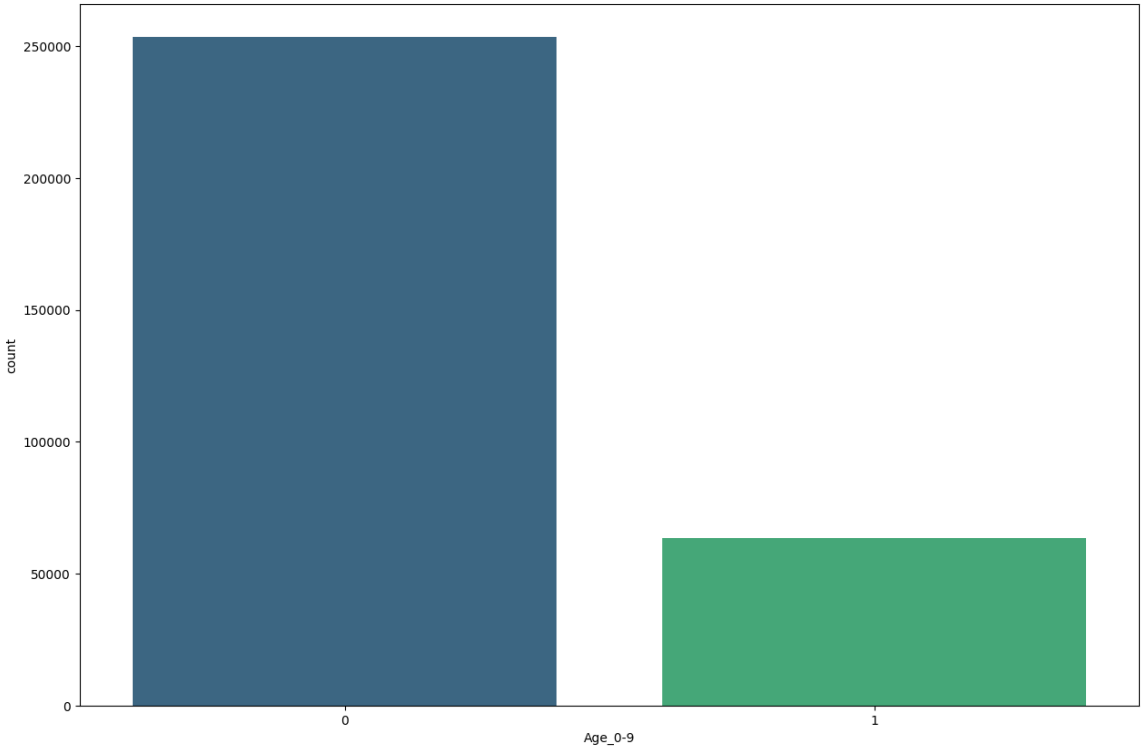
Count Plot for Runny-Nose



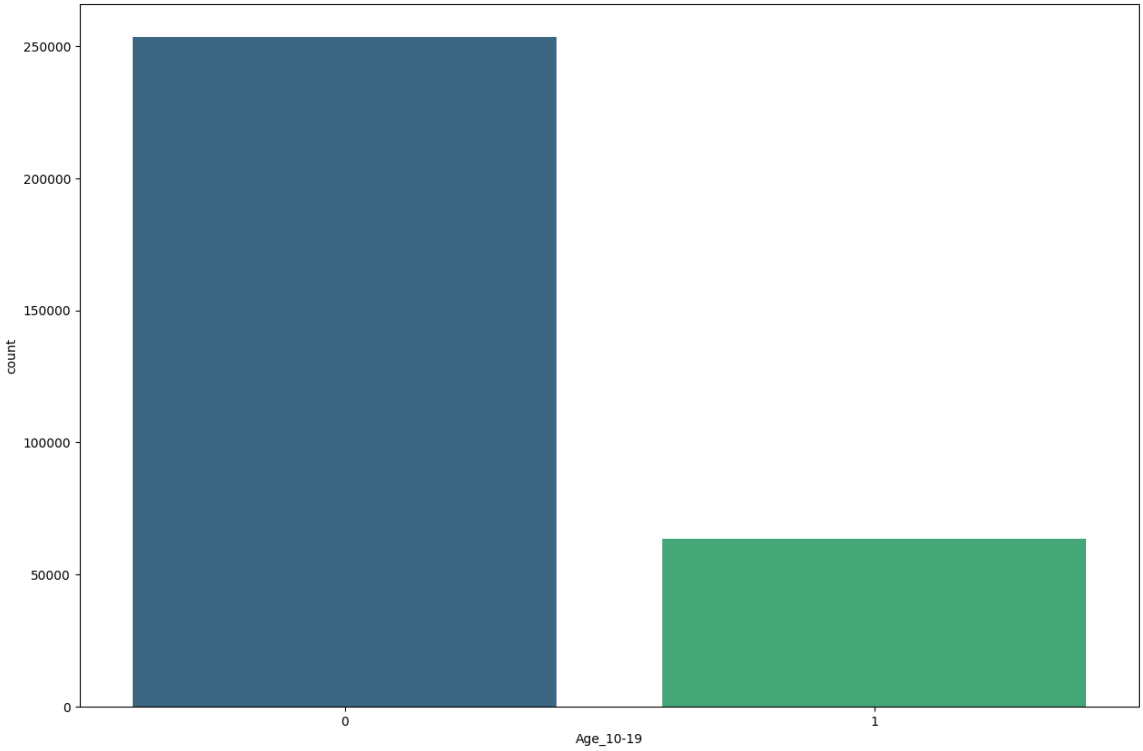
Count Plot for None_Experiencing



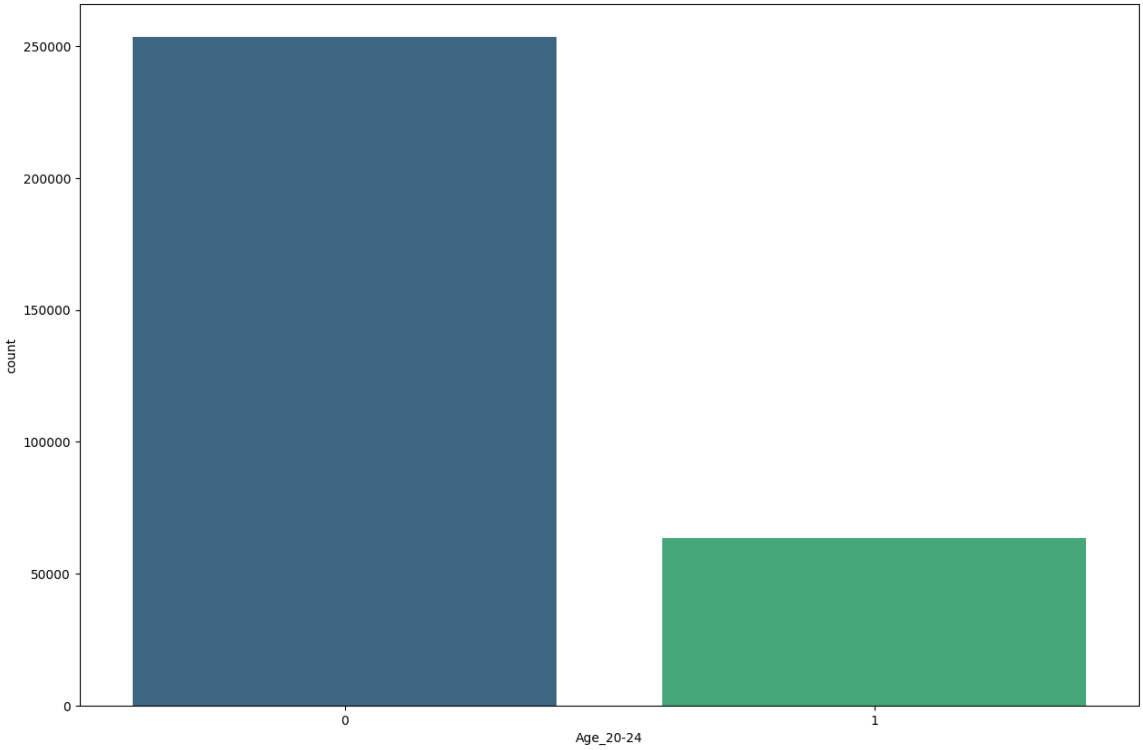
Count Plot for Age_0-9



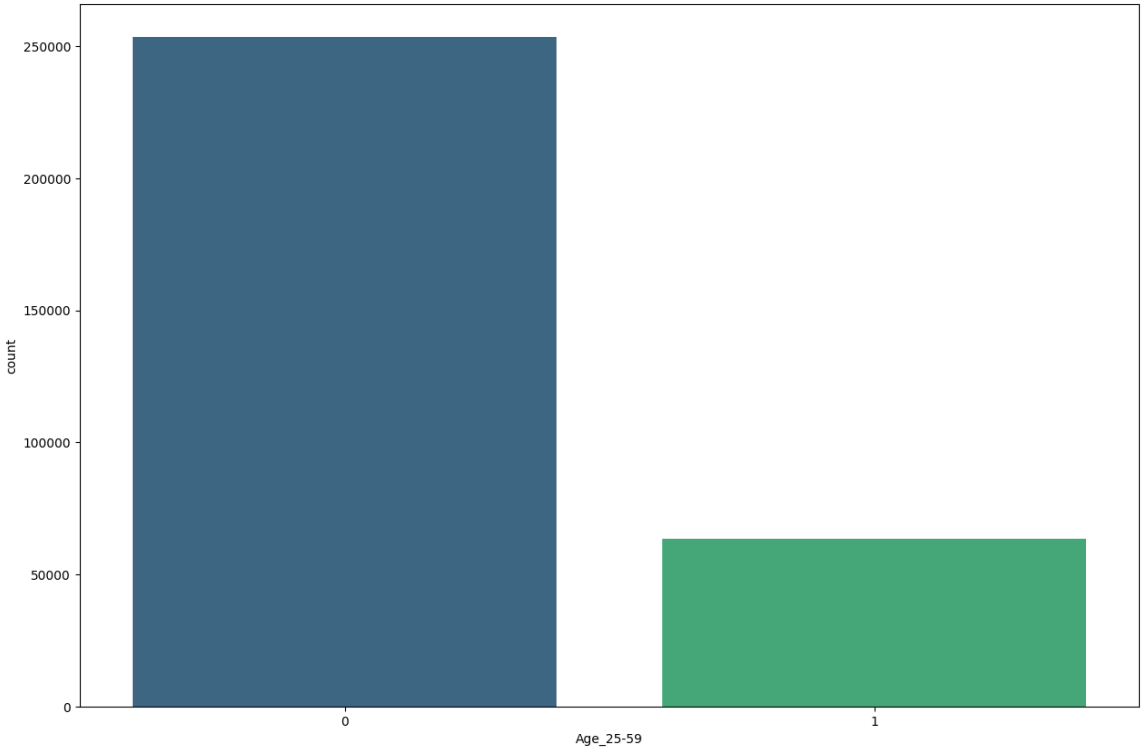
Count Plot for Age_10-19



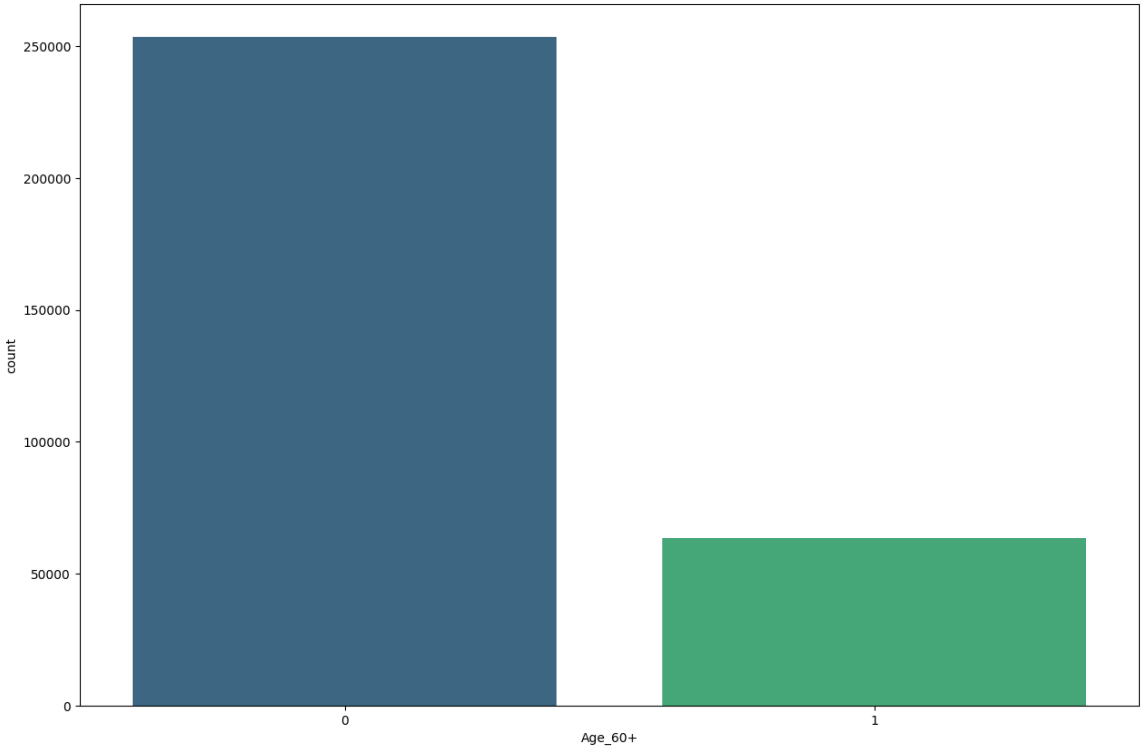
Count Plot for Age_20-24



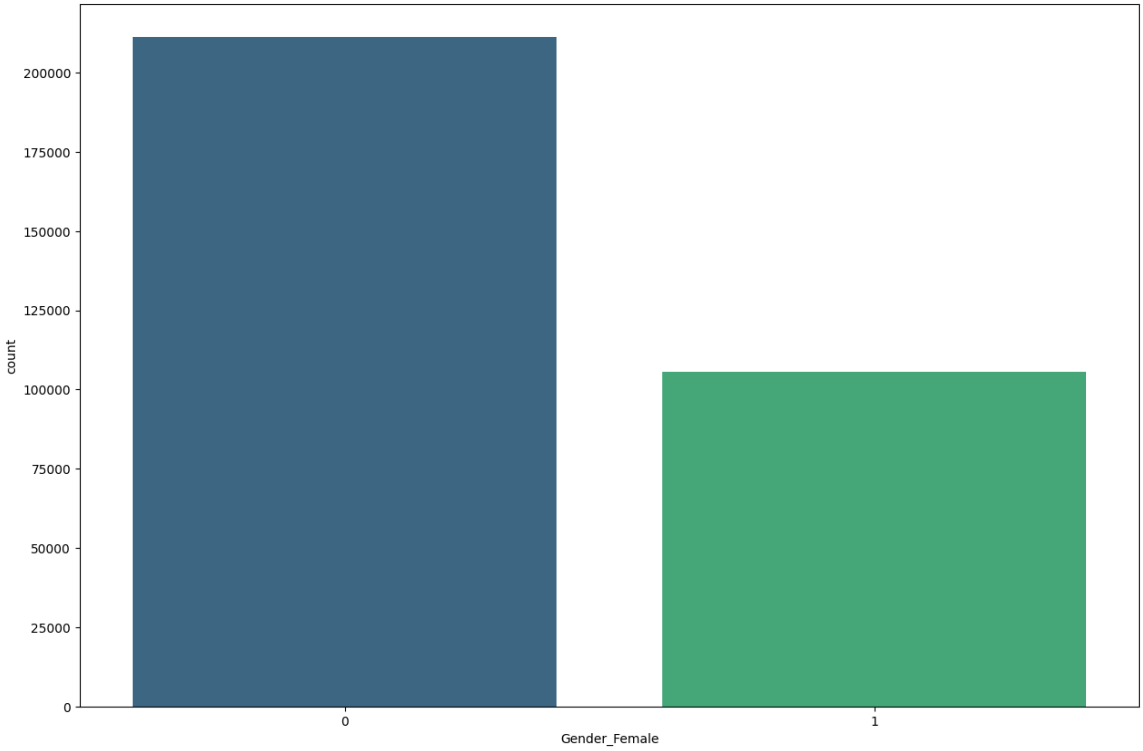
Count Plot for Age_25-59



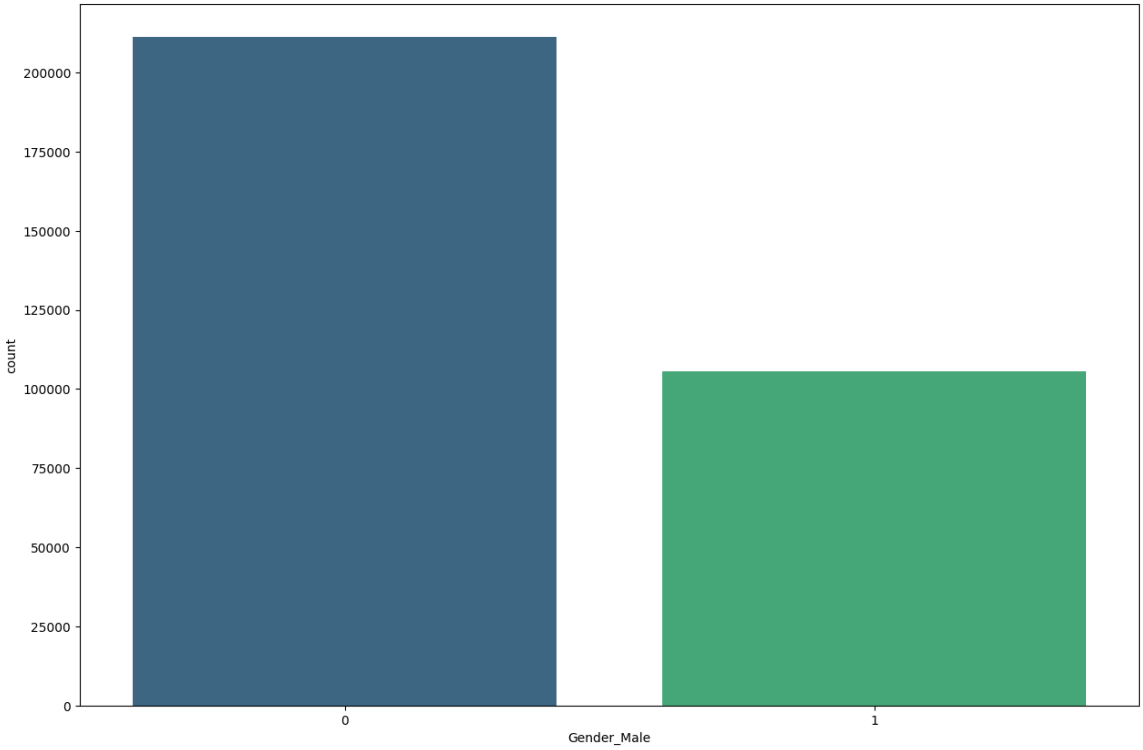
Count Plot for Age_60+



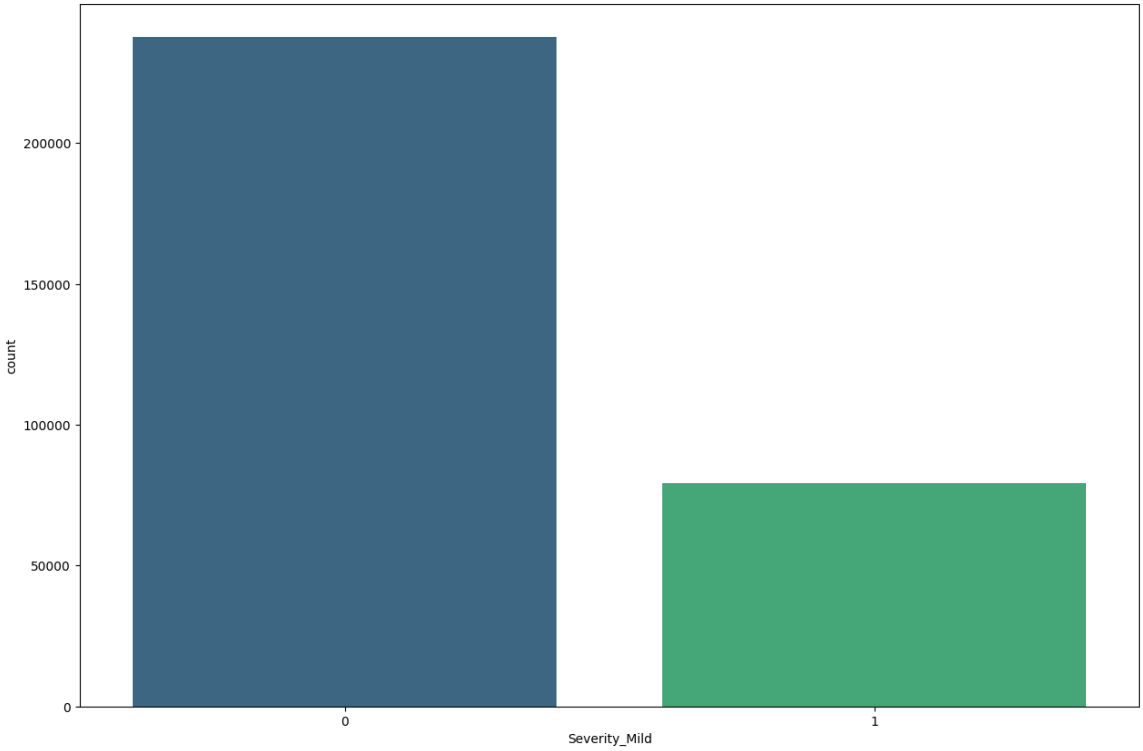
Count Plot for Gender_Female



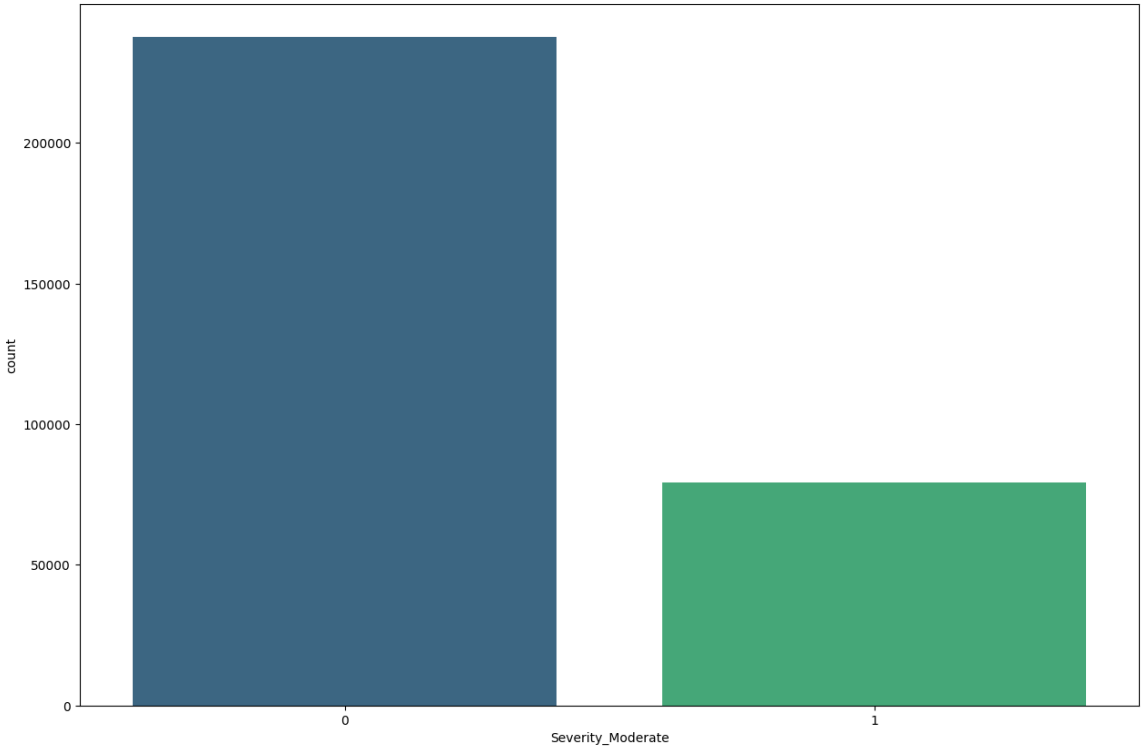
Count Plot for Gender_Male



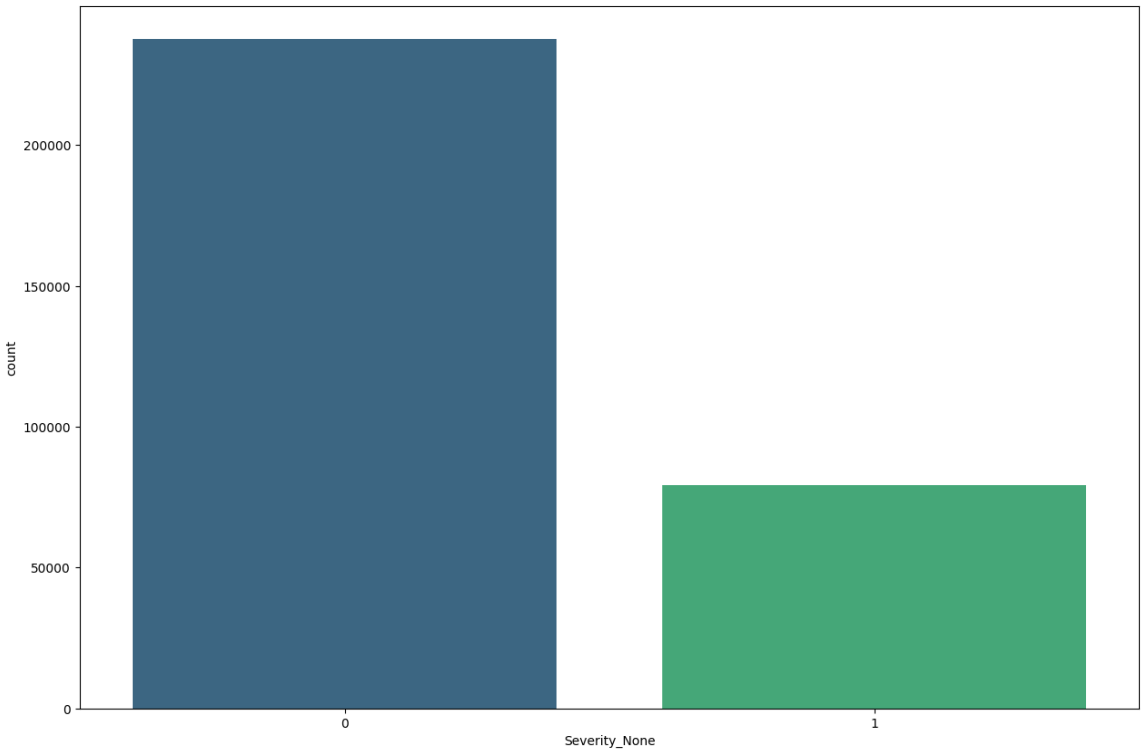
Count Plot for Severity_Mild



Count Plot for Severity_Moderate




```
Count Plot for  Severity_None
```



PIE plot for individual coloums

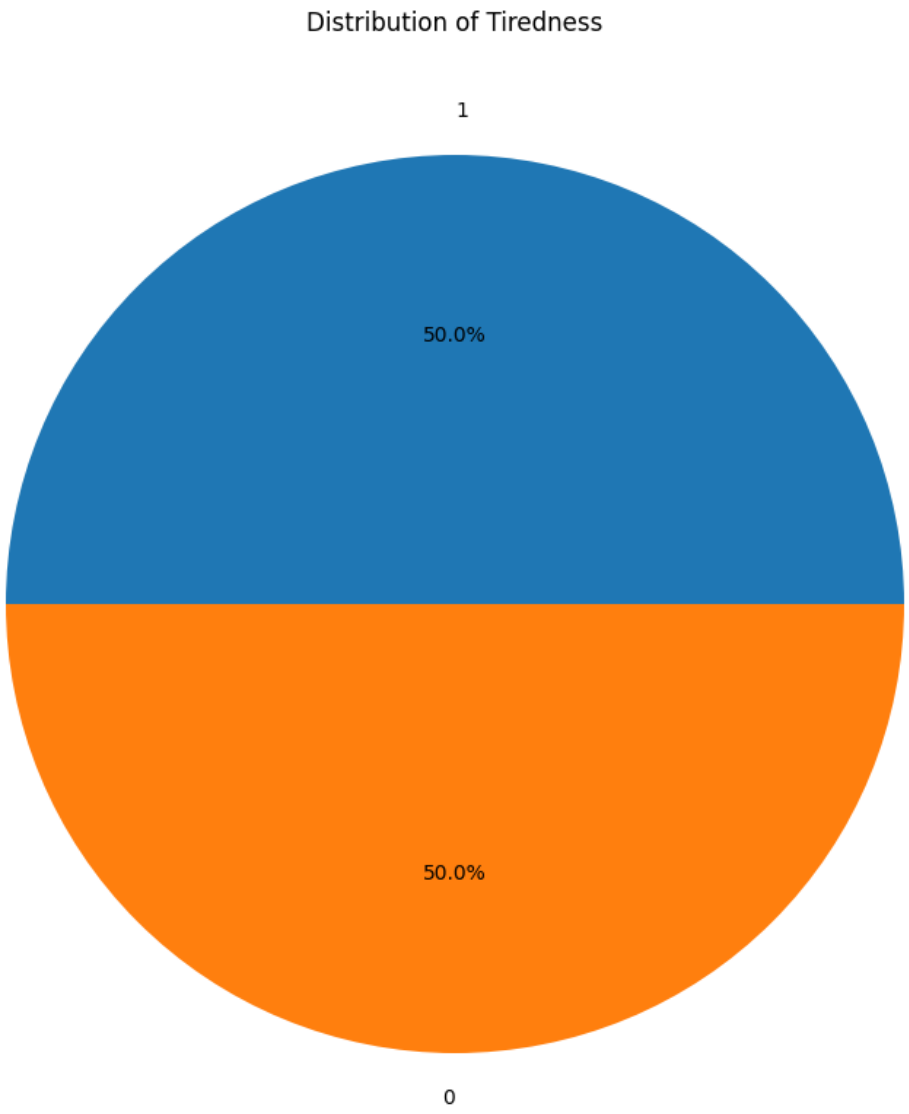
.

autopct: This parameter is a string or function used to label the wedges (to break them apart) with their numeric value.

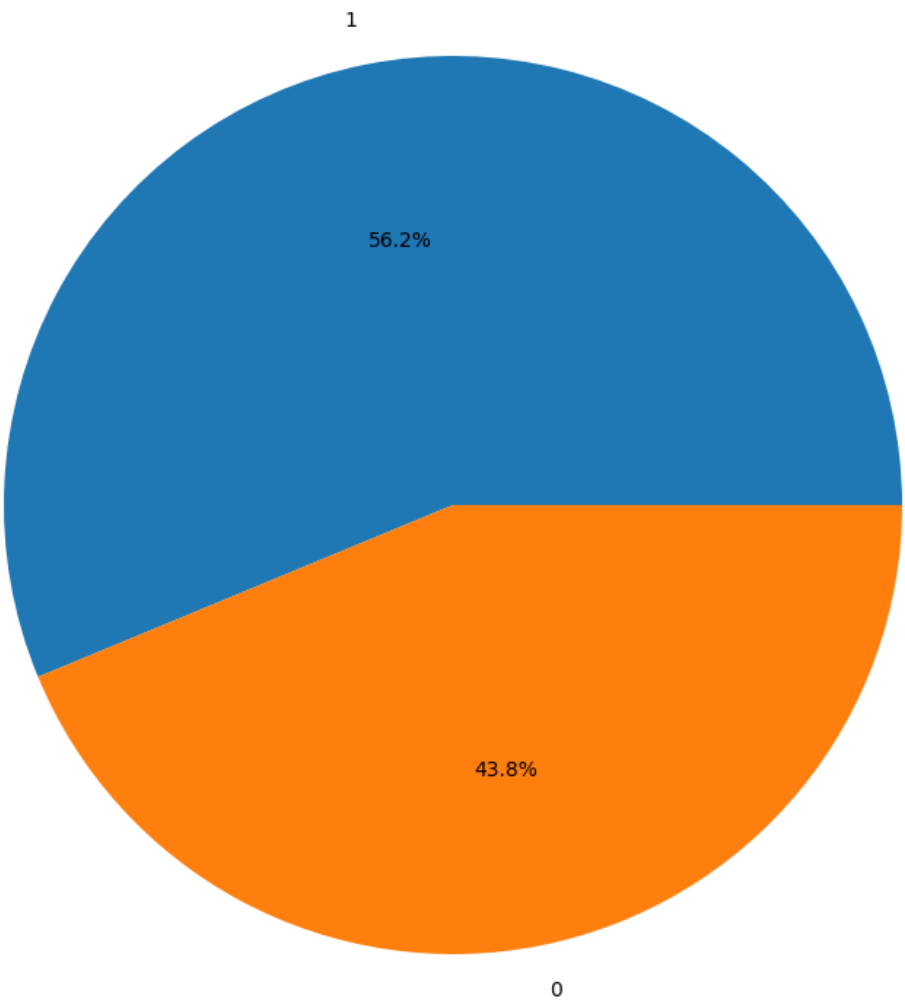
In [14]:
```
for i in df.columns:
    print('Pie plot for:', i)
    plt.figure(figsize=(20, 10))
```

```
    df[i].value_counts().plot(kind='pie', autopct='%1.1f%%')
    plt.title('Distribution of ' + i)
    plt.ylabel('')
    plt.show()
    print('\n')
```

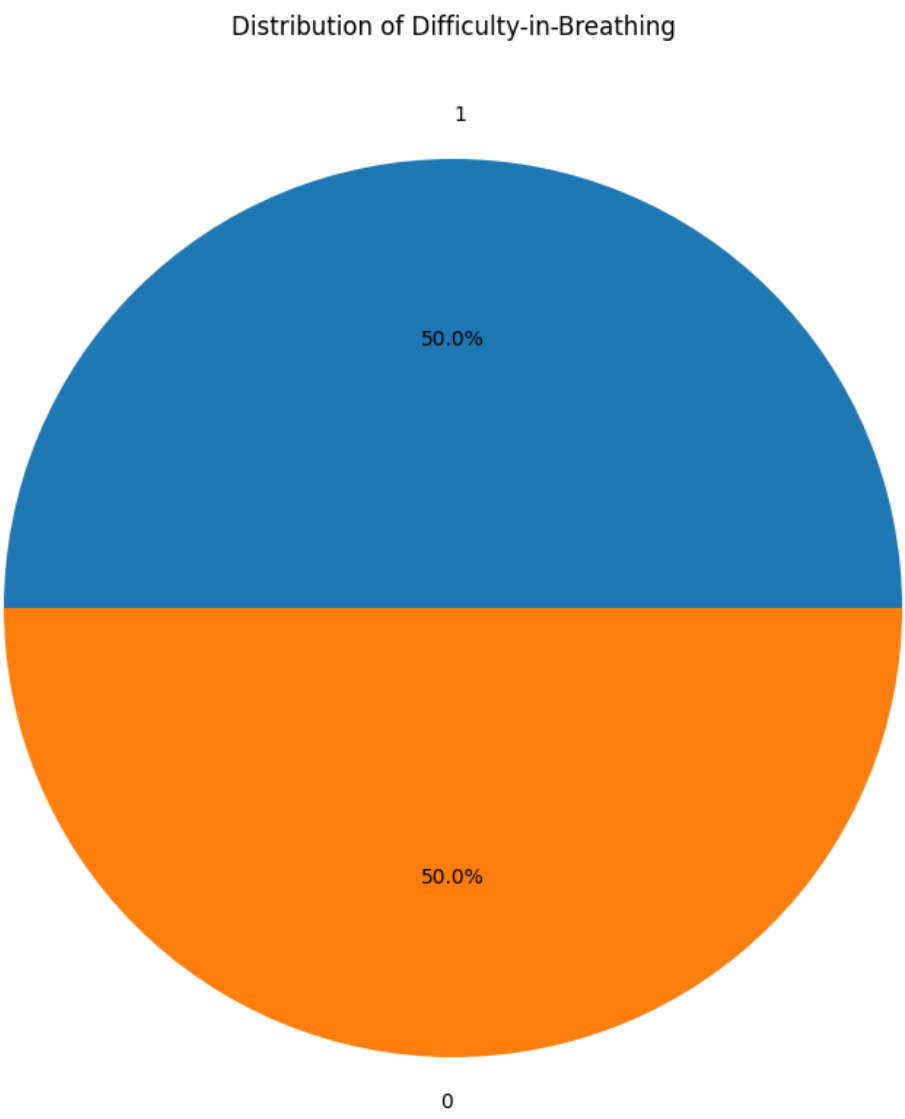
```
Pie plot for: Tiredness
```



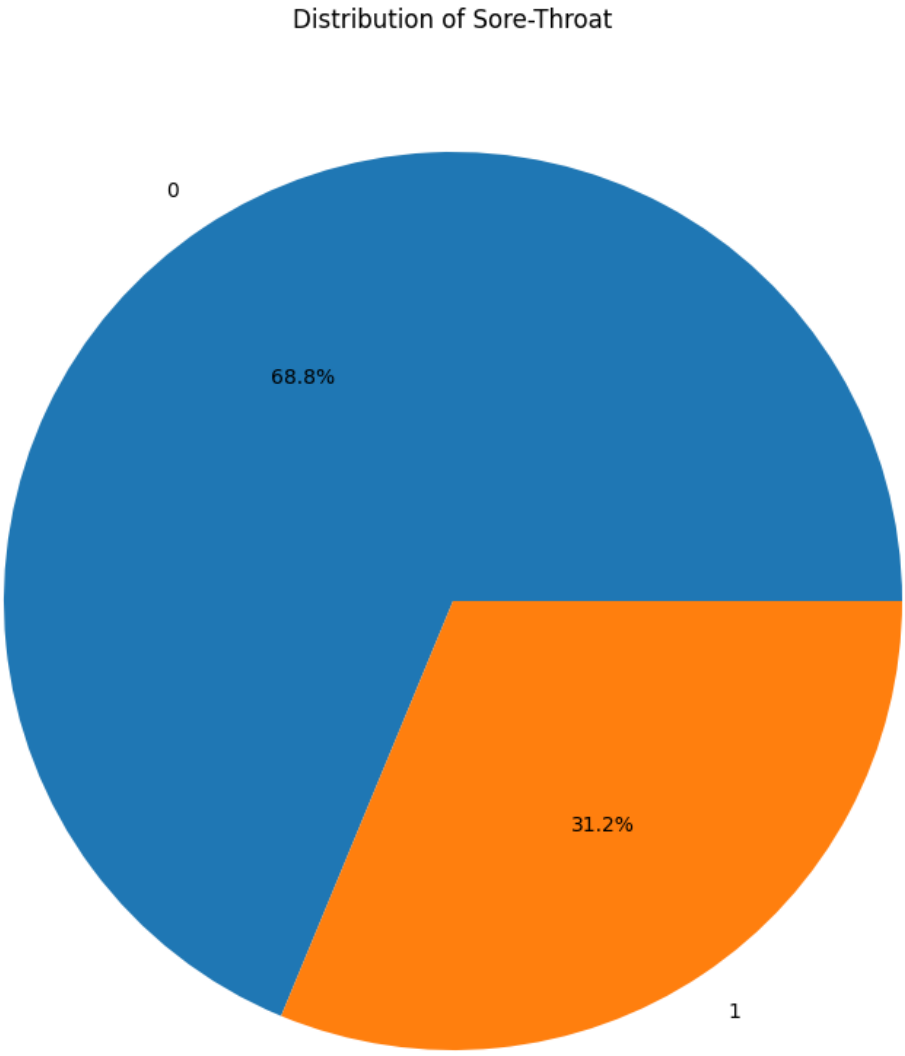

```
Pie plot for: Dry-Cough
```



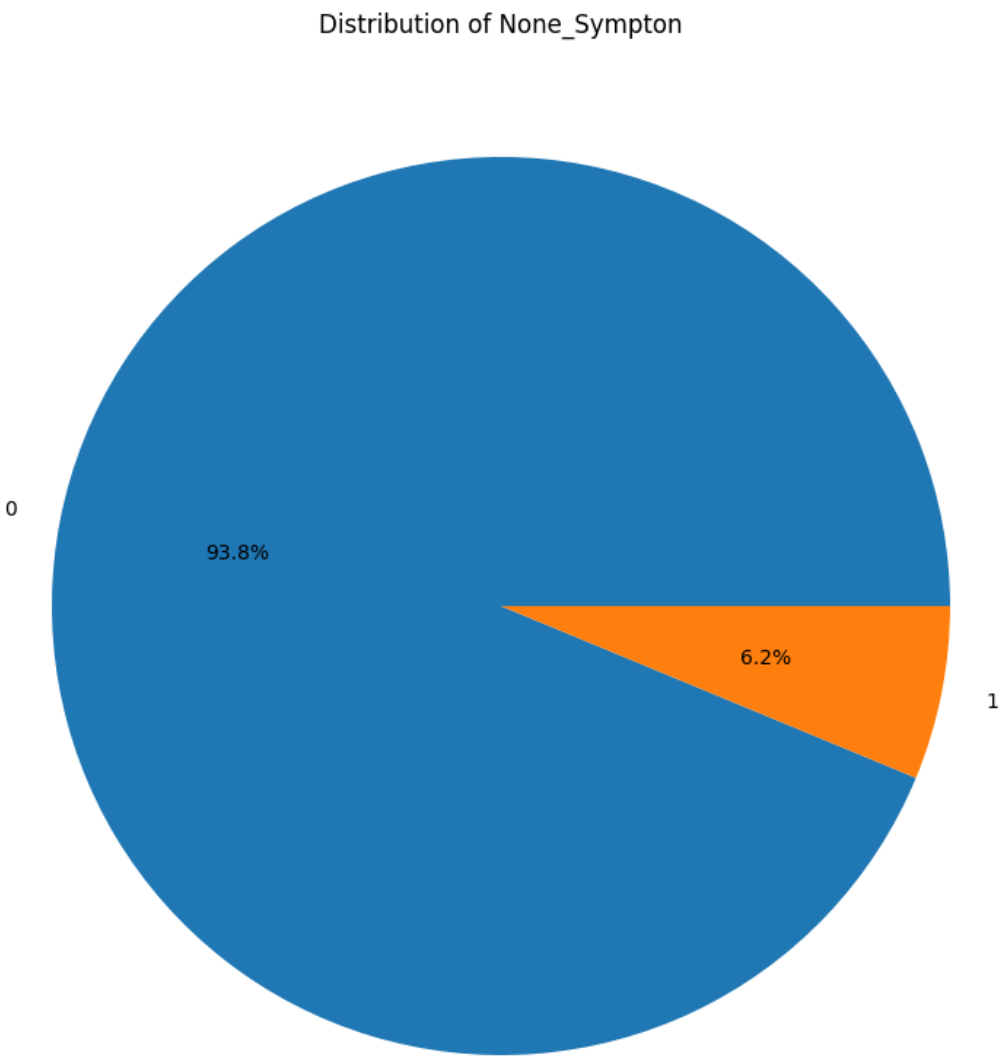
Pie plot for: Difficulty-in-Breathing



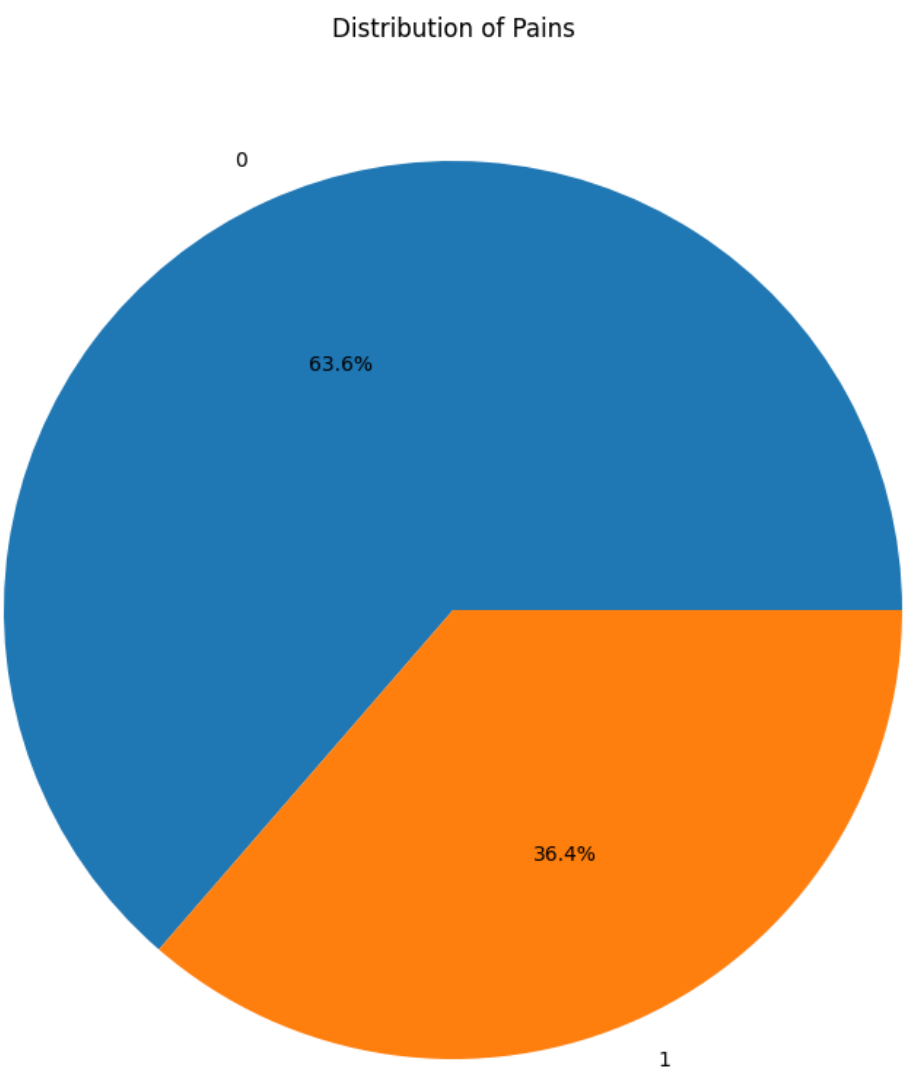
Pie plot for: Sore-Throat



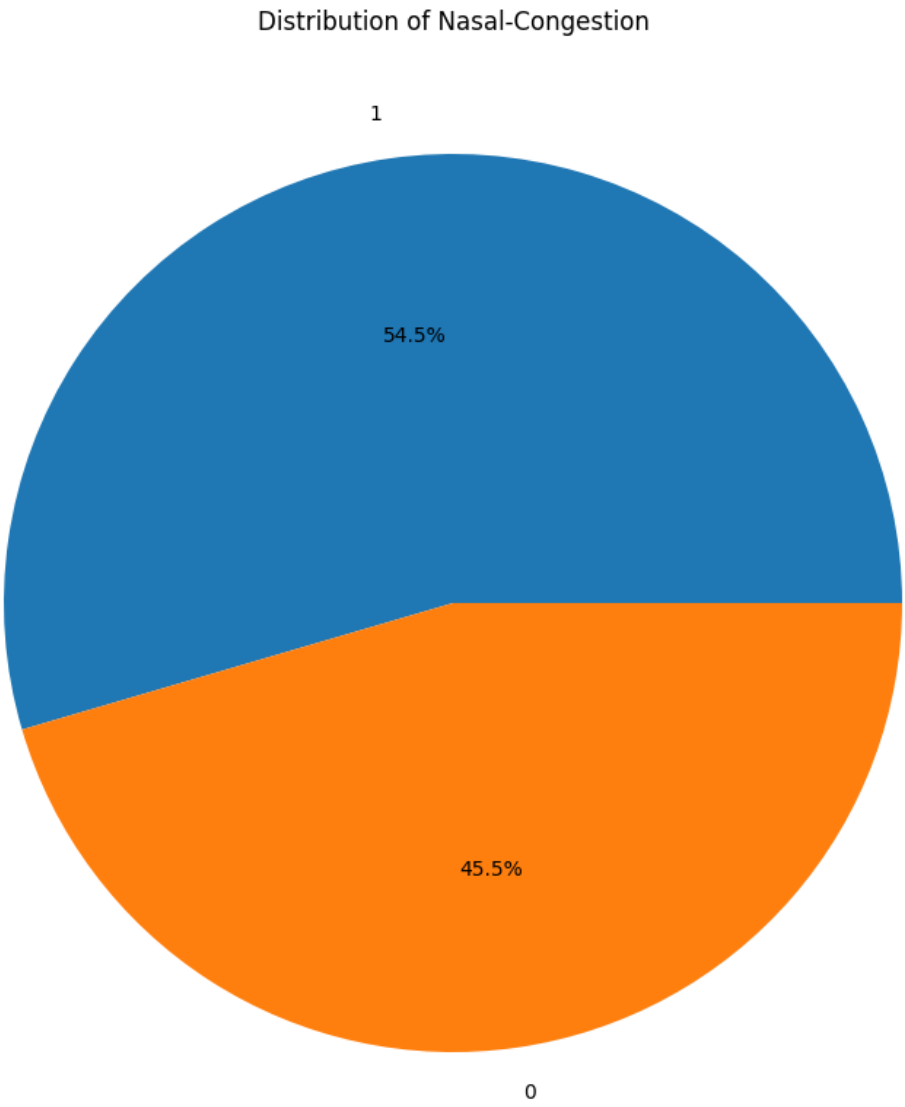
Pie plot for: None_Sympton



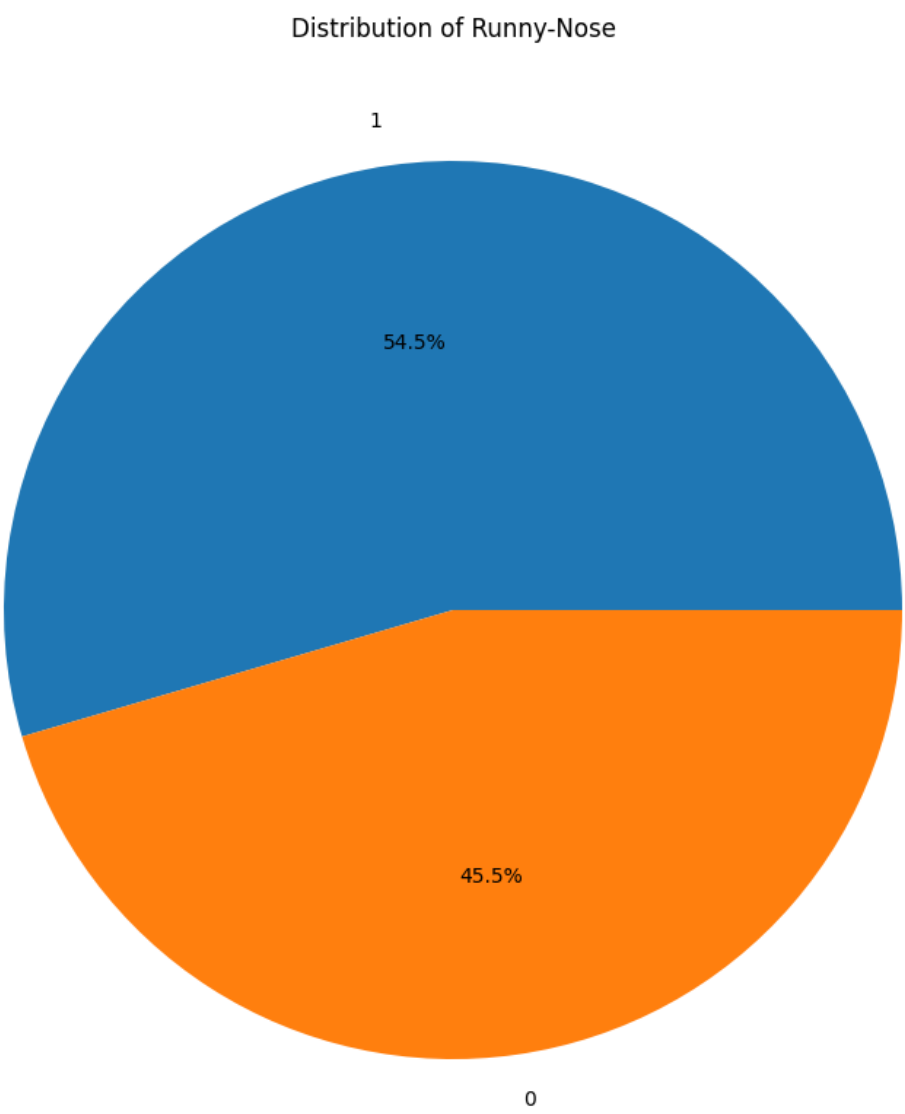
Pie plot for: Pains



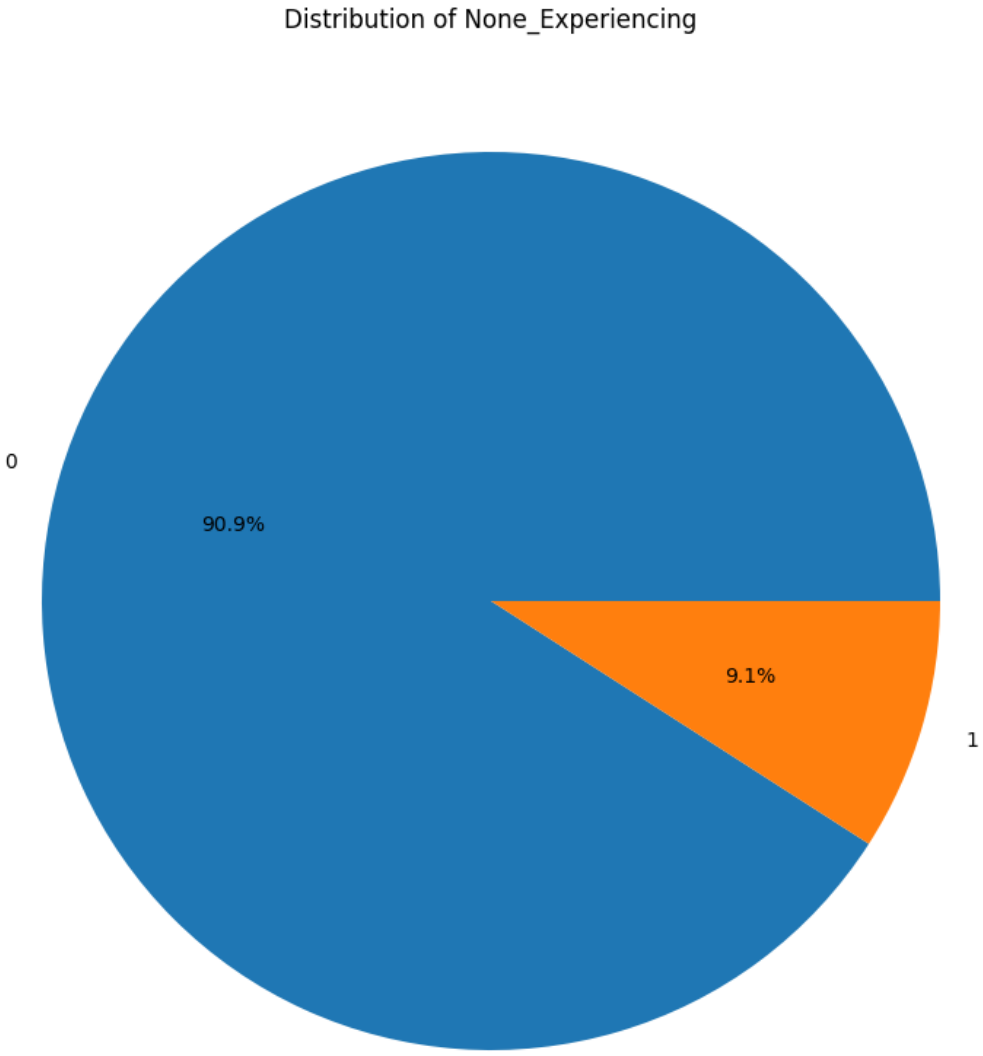
Pie plot for: Nasal-Congestion



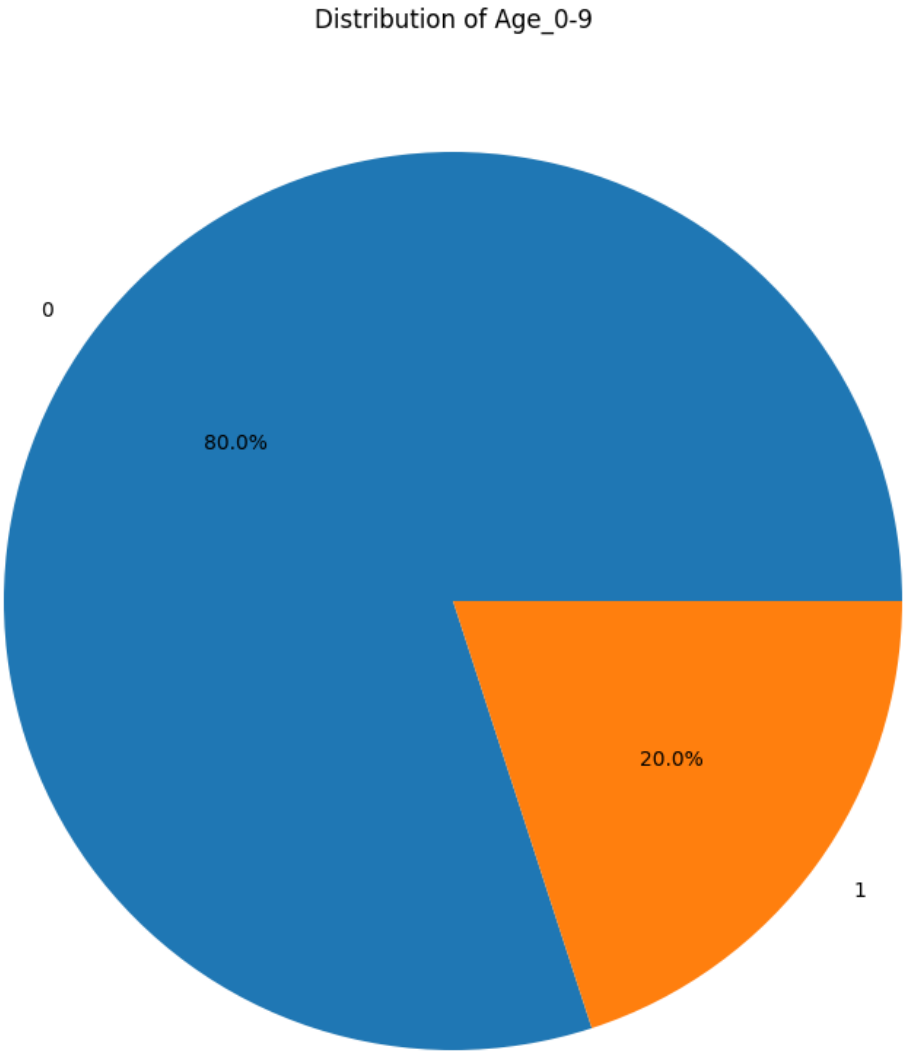
Pie plot for: Runny-Nose



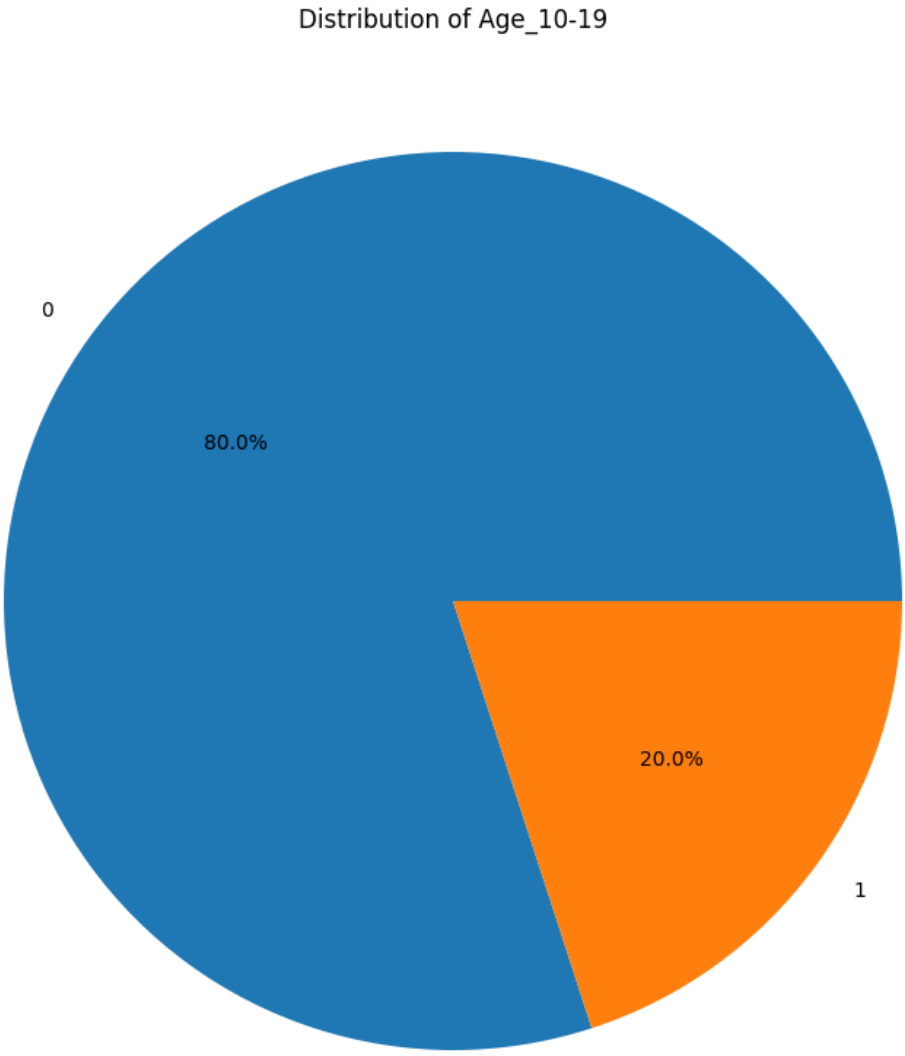
Pie plot for: None_Experiencing



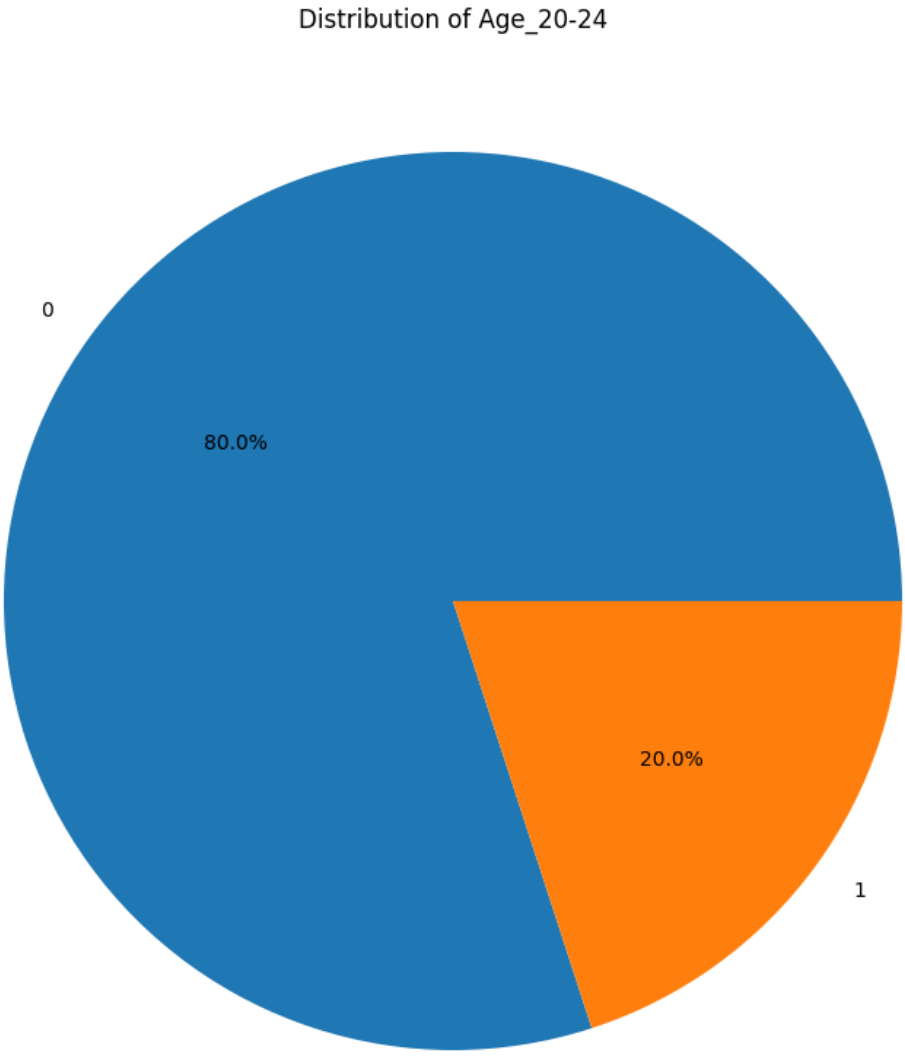
Pie plot for: Age_0-9



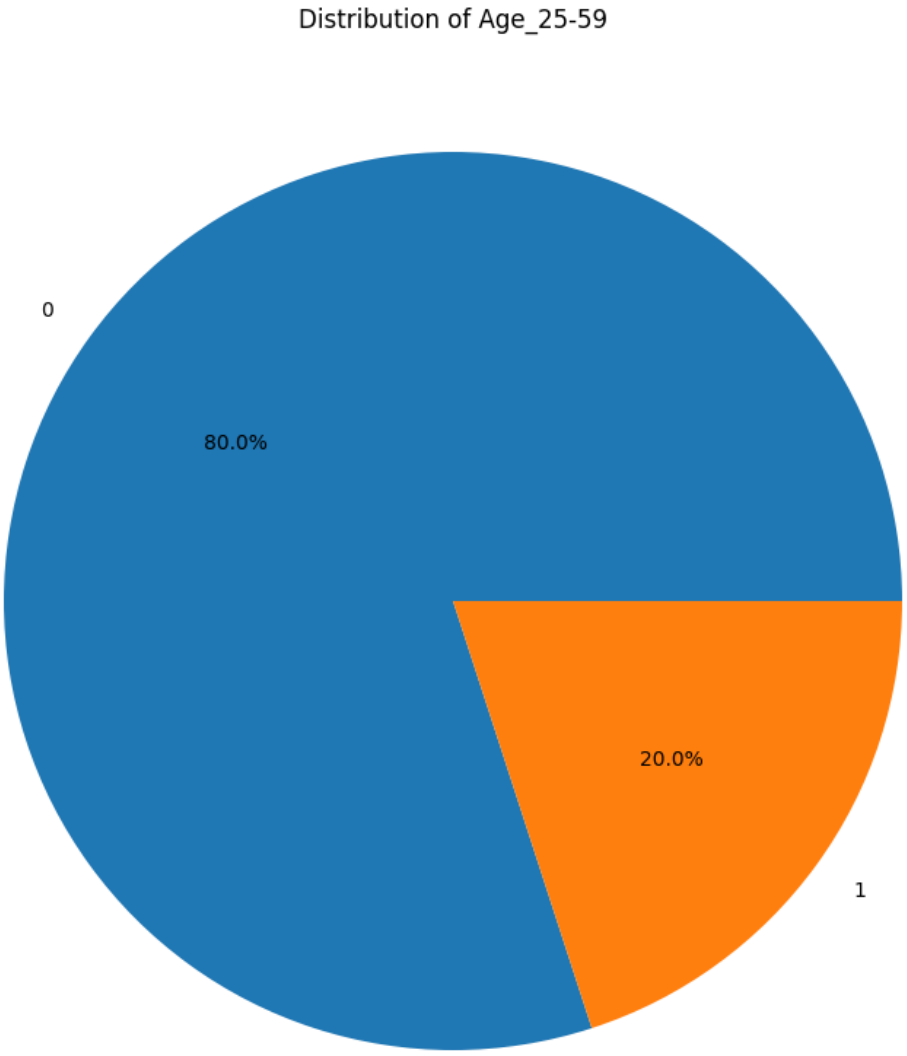
Pie plot for: Age_10-19



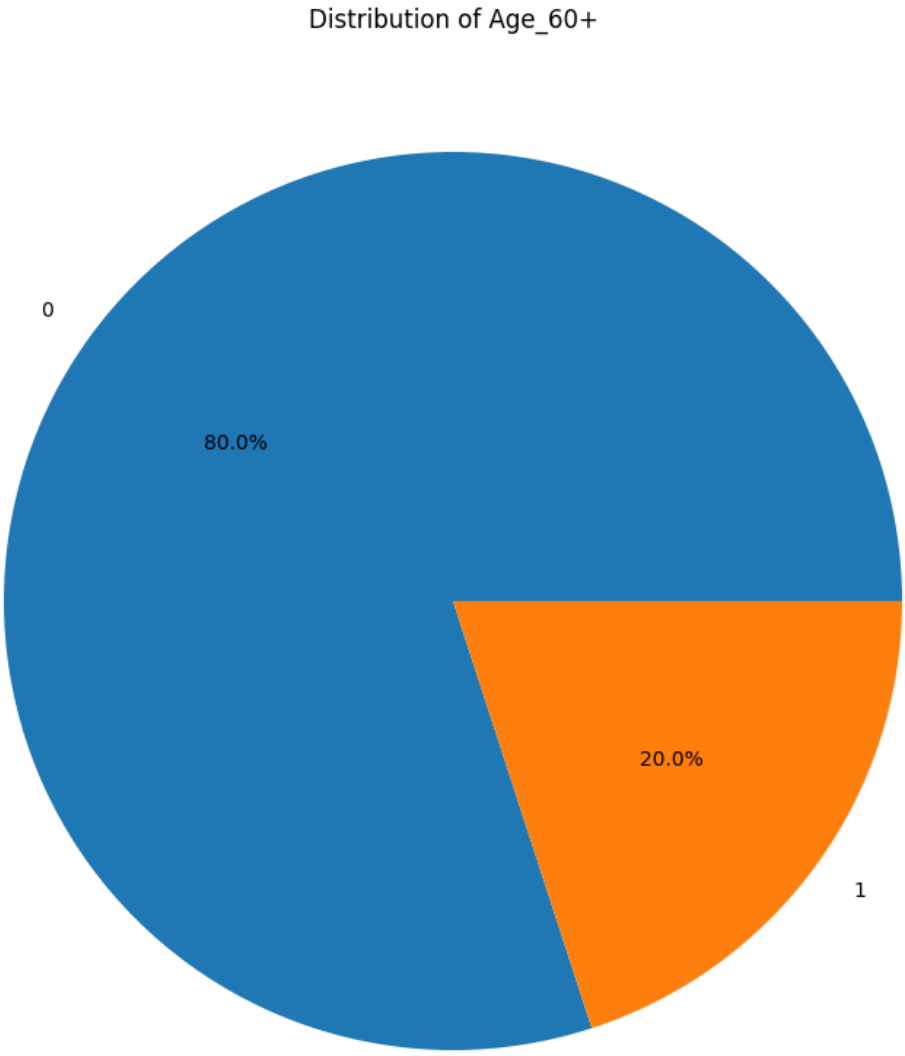
Pie plot for: Age_20-24



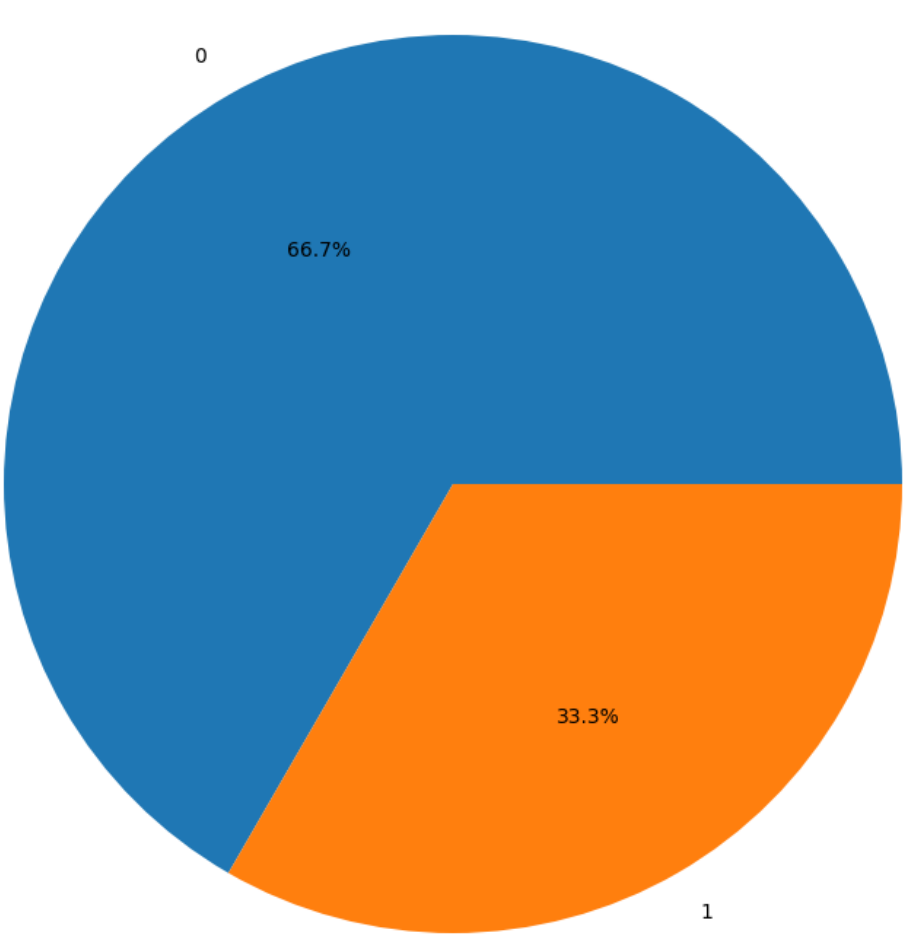
Pie plot for: Age_25-59



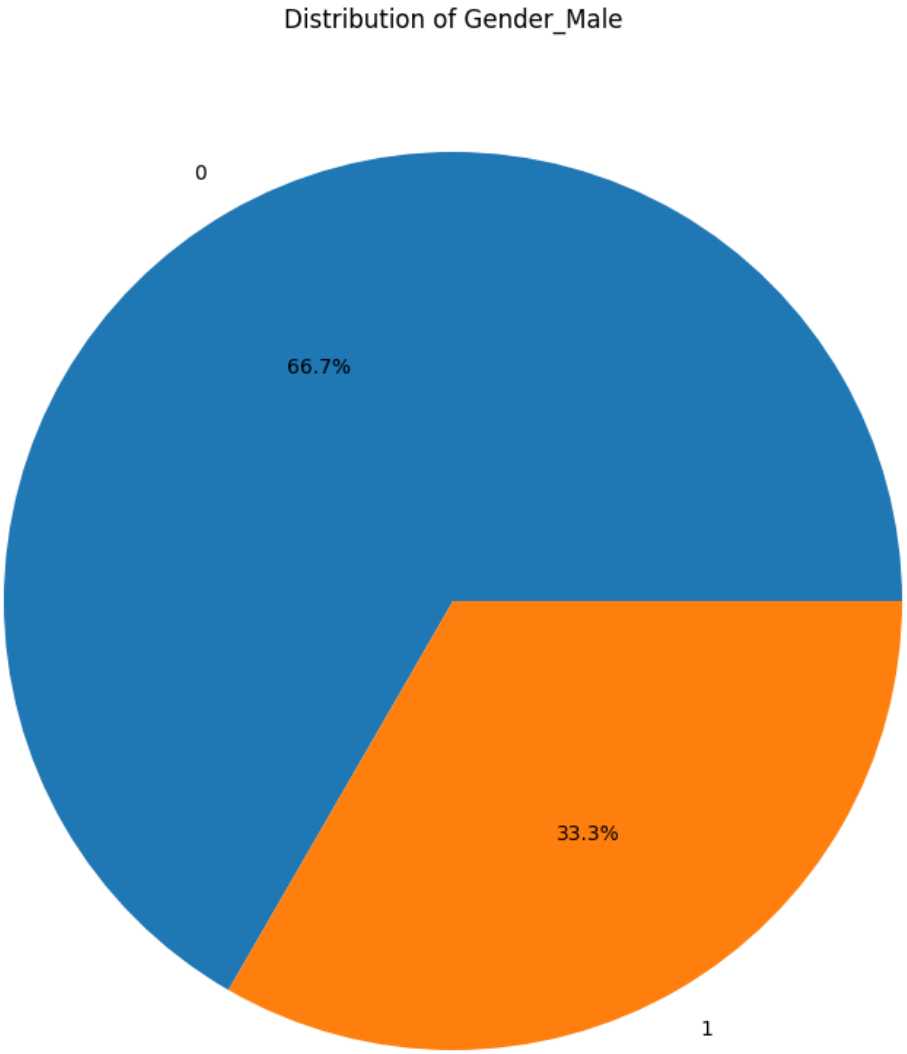
Pie plot for: Age_60+



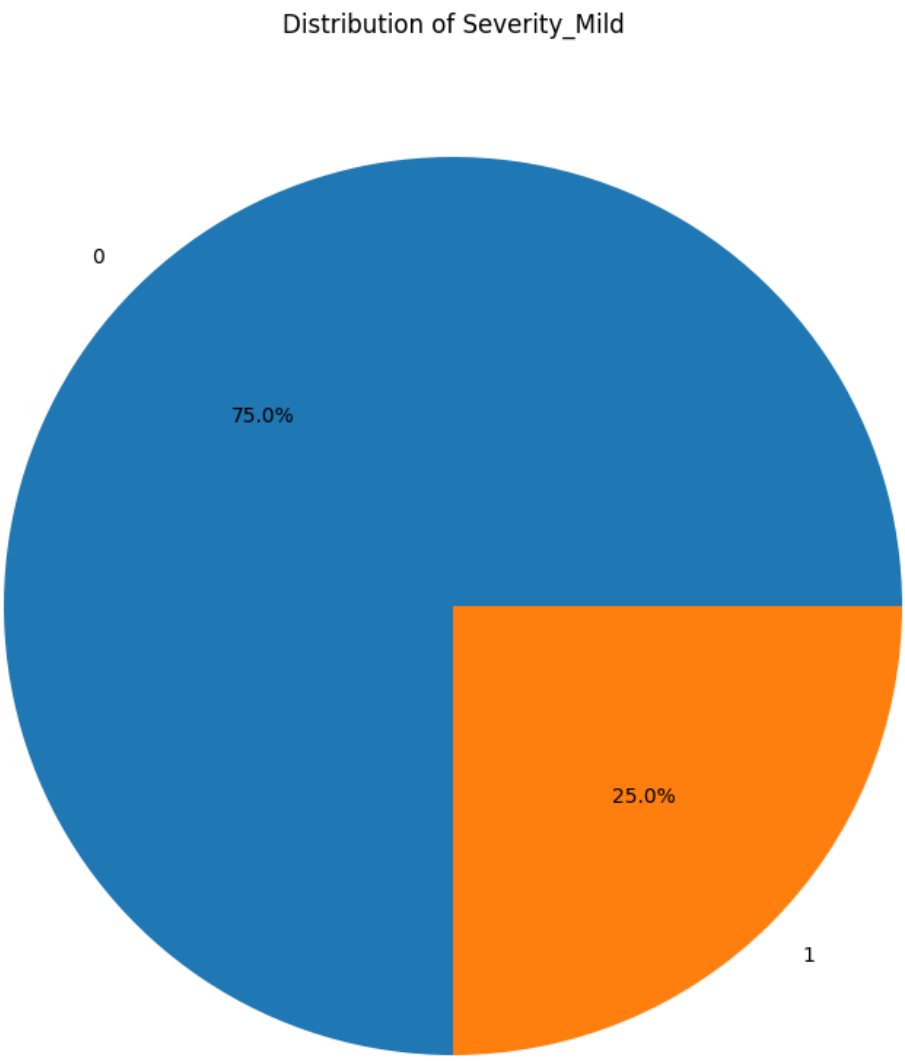
Pie plot for: Gender_Female



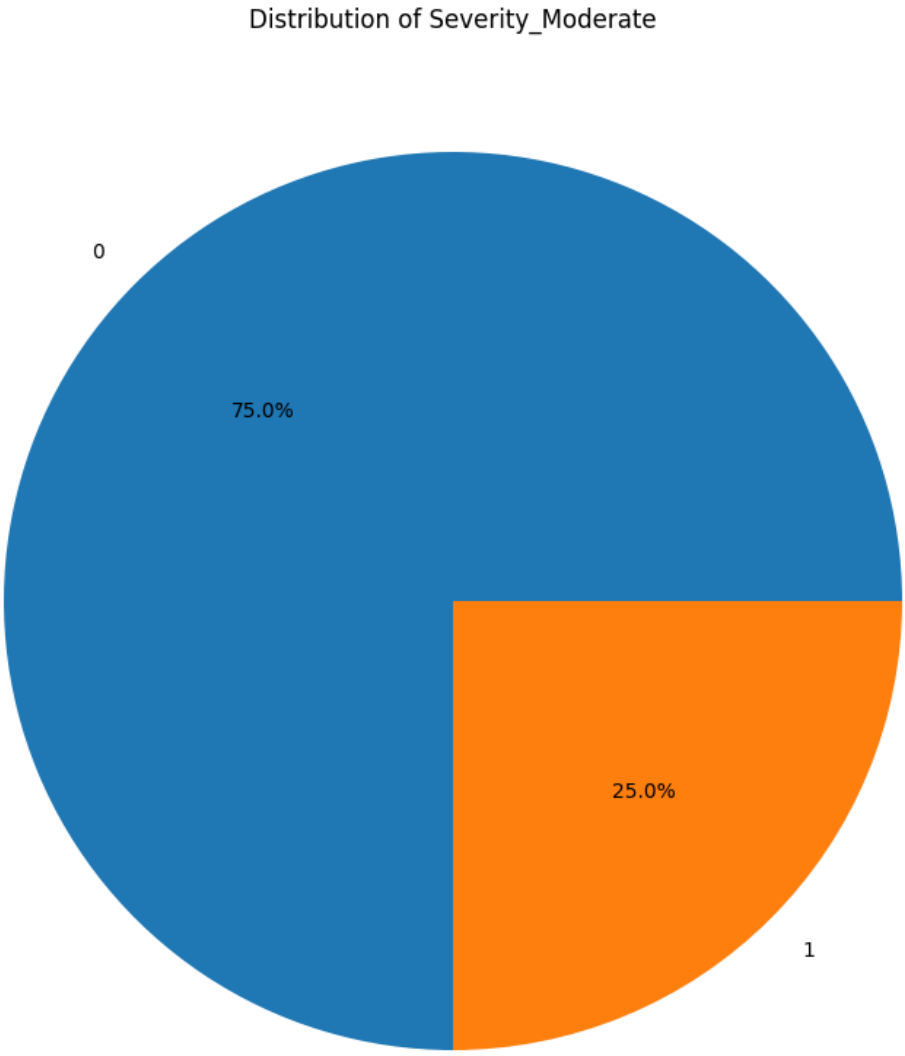

```
Pie plot for: Gender_Male
```



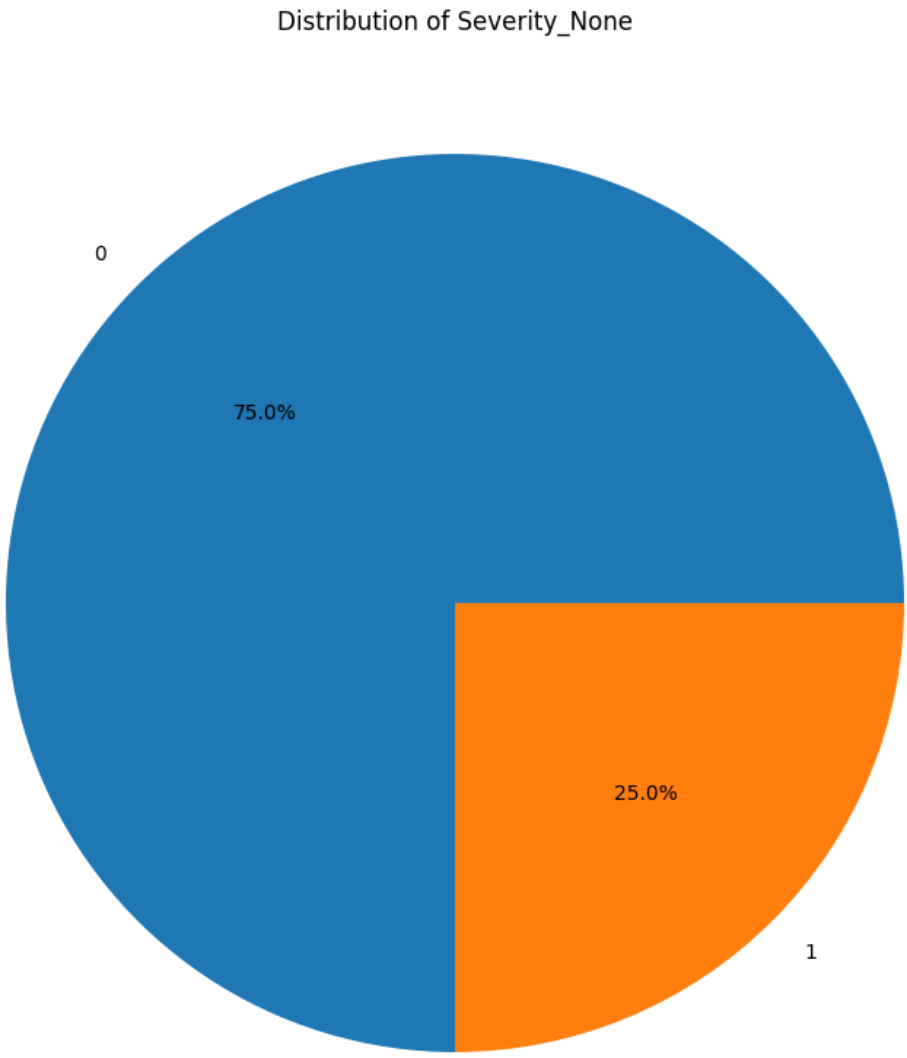
Pie plot for: Severity_Mild



Pie plot for: Severity_Moderate



Pie plot for: Severity_None



BAR plot for individual coloums

.

.index means the Name

.valuecounts means the values

In [15]:
```
for i in df.columns:
    fig = go.Figure(data = [go.Bar(x = df[i].value_counts().index ,
                                   y = df[i].value_counts())]
                    )
    fig.update_layout(

        title = i,
        xaxis_title = i,
        yaxis_title = "Count"

        )

    fig.show()
```

PIE(Distribution) plot for individual coloums

.

In [16]:
```
for i in df.columns:
    print("Pie plot for ," , i)

    fig = px.pie(df , names=i , title = "Distribution of"  + i )
    fig.show()

    print('\n')
```

```
Pie plot for , Tiredness


Pie plot for , Dry-Cough


Pie plot for , Difficulty-in-Breathing


Pie plot for , Sore-Throat


Pie plot for , None_Sympton


Pie plot for , Pains


Pie plot for , Nasal-Congestion


Pie plot for , Runny-Nose


Pie plot for , None_Experiencing


Pie plot for , Age_0-9


Pie plot for , Age_10-19


Pie plot for , Age_20-24


Pie plot for , Age_25-59


Pie plot for , Age_60+
```

```
Pie plot for , Gender_Female

Pie plot for , Gender_Male

Pie plot for , Severity_Mild

Pie plot for , Severity_Moderate

Pie plot for , Severity_None
```

DATA ASSIMILATION

.

Structuring the data

DROP columns which wont contribute to features

In [17]:
```
df = df.drop(['Severity_None'] , axis = 1)
```

.idxmax(axis=1): This part of the code uses the idxmax function to find the column label (either 'Severity_Mild' or 'Severity_Moderate') that contains the maximum value for each row along the horizontal axis (axis=1). In other words, for each row in the DataFrame, it identifies which severity is the highest (either 'Mild' or 'Moderate').

In [19]:
```
df['Asthma_Severity'] = df[['Severity_Mild', 'Severity_Moderate']].idxmax
```

Defined dictionary called severity_mappings that maps the values from two keys, 'Severity_Mild' and 'Severity_Moderate,' to corresponding values 'Mild' and 'Moderate,' respectively.

In [20]:
```
severity_mappings = {
    'Severity_Mild' : 'Mild' ,
    'Severity_Moderate' :  'Moderate'
}
```

In [21]:
```
print(severity_mappings)
```

```
{'Severity_Mild': 'Mild', 'Severity_Moderate': 'Moderate'}
```

In [22]:
```
df['Asthma_Severity'] = df['Asthma_Severity'].map(severity_mappings)
```

In [26]:
```
df['Asthma_Severity']
```

Out[26]:
```
0             Mild
1             Mild
2             Mild
3         Moderate
4         Moderate
             ...
316795        Mild
316796        Mild
316797        Mild
```

```
316798        Mild
316799        Mild
Name: Asthma_Severity, Length: 316800, dtype: object
```

Converting to 1's and 0's like our DATA

In [29]:
```
severity_mappings =  {

    "Mild" : 0,
    "Moderate" : 1,
}
```

In [30]:
```
df['Asthma_Severity'] = df['Asthma_Severity'].map(severity_mappings)
```

In [31]:
```
df['Asthma_Severity'].value_counts()
```

Out[31]:
```
0    237600
1     79200
Name: Asthma_Severity, dtype: int64
```

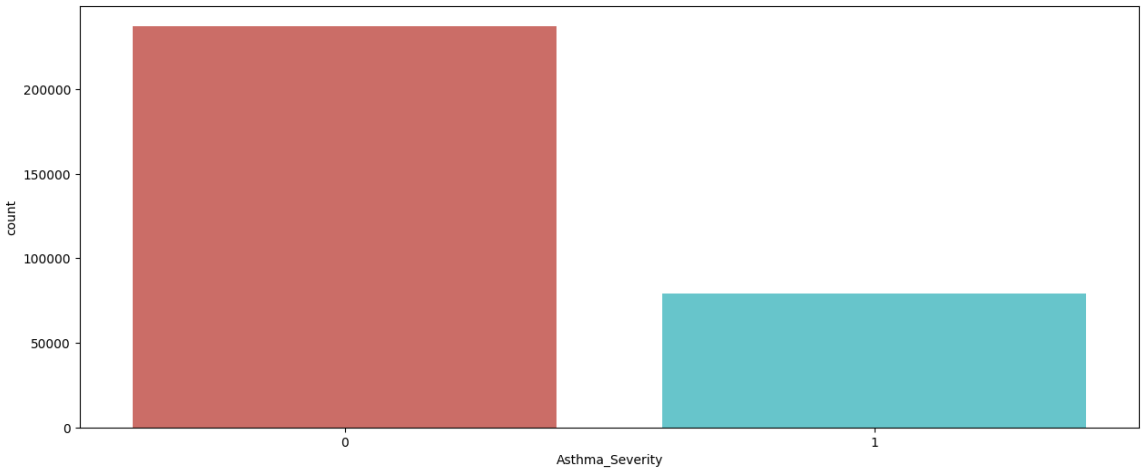
Visualize the New Df

Count plot

.

In [32]:
```
print('Countplot for :' , 'Asthma_Severity')
plt.figure(figsize=(15,6))
sns.countplot(x=df['Asthma_Severity'] , data = df , palette='hls')
```

Out[32]:
```
Countplot for : Asthma_Severity
<AxesSubplot:xlabel='Asthma_Severity', ylabel='count'>
```

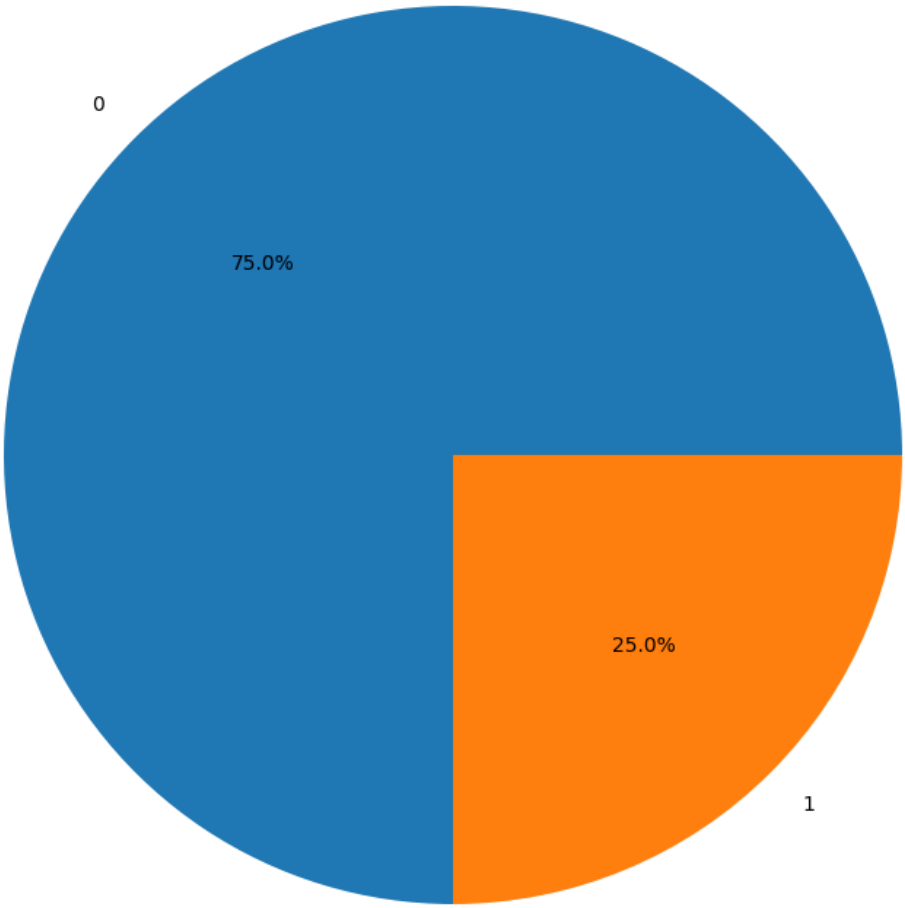


PIE PLOT

.

In [33]:
```
print('Pie plot for:', 'Asthma_Severity')
plt.figure(figsize=(20, 10))
df['Asthma_Severity'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.ylabel('')
plt.show()
```

```
Pie plot for: Asthma_Severity
```



BAR PLOT

.

In [34]:
```
fig = go.Figure(data=[go.Bar(x=df['Asthma_Severity'].value_counts().index
                             y=df['Asthma_Severity'].value_counts())])
fig.update_layout(title='Asthma_Severity', xaxis_title='Asthma_Severity',
                  yaxis_title="Count")
fig.show()
```

In [35]:
```
print('Pie plot for:', 'Asthma_Severity')
fig = px.pie(df, names='Asthma_Severity')
fig.show()
```

```
Pie plot for: Asthma_Severity
```

In [36]:
```
df.columns
```

Out[36]:
```
Index(['Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing', 'Sore-Throa
t',
       'None_Sympton', 'Pains', 'Nasal-Congestion', 'Runny-Nose',
       'None_Experiencing', 'Age_0-9', 'Age_10-19', 'Age_20-24', 'Age_25-
59',
       'Age_60+', 'Gender_Female', 'Gender_Male', 'Severity_Mild',
```

```
       'Severity_Moderate', 'Asthma_Severity'],
      dtype='object')
```

severity_levels = ['Severity_Mild', 'Severity_Moderate']:

This line creates a list called severity_levels containing the names of two columns from the DataFrame df. These columns are 'Severity_Mild' and 'Severity_Moderate.'

severity_distribution = df[severity_levels].sum():

This line calculates the sum of values in the columns specified in the severity_levels list. In this case, it calculates the sum of values in the 'Severity_Mild' and 'Severity_Moderate' columns.

In [38]:
```
severity_levels = ['Severity_Mild', 'Severity_Moderate']
severity_distribution = df[severity_levels].sum()
```

In [39]:
```
severity_distribution
```

Out[39]:
```
Severity_Mild        79200
Severity_Moderate    79200
dtype: int64
```

Drop as we dont need it anymore

In [40]:
```
df = df.drop(['Severity_Mild','Severity_Moderate'], axis = 1)
```

In [41]:
```
symptoms = ['Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing', 'Sore-Th
symptom_counts = df[symptoms].sum()
```

In [43]:
```
symptoms
```

Out[43]:
```
['Tiredness',
 'Dry-Cough',
 'Difficulty-in-Breathing',
 'Sore-Throat',
 'Pains',
 'Nasal-Congestion',
 'Runny-Nose']
```

In [42]:
```
symptom_counts
```

Out[42]:
```
Tiredness                 158400
Dry-Cough                 178200
Difficulty-in-Breathing   158400
Sore-Throat                99000
Pains                     115200
Nasal-Congestion          172800
Runny-Nose                172800
dtype: int64
```

In [44]:
```
age_groups = ['Age_0-9', 'Age_10-19', 'Age_20-24', 'Age_25-59', 'Age_60+'
age_distribution = df[age_groups].sum()
```

In [45]:
```
age_groups
```

Out[45]:
```
['Age_0-9', 'Age_10-19', 'Age_20-24', 'Age_25-59', 'Age_60+']
```

In [46]:
```
age_distribution
```

Out[46]:
```
Age_0-9       63360
Age_10-19     63360
Age_20-24     63360
Age_25-59     63360
Age_60+       63360
dtype: int64
```

In [47]:
```
gender_groups = ['Gender_Female' , 'Gender_Male']
gender_distribution = df[gender_groups].sum()
```

In [48]:
```
gender_distribution
```

Out[48]:
```
Gender_Female    105600
Gender_Male      105600
dtype: int64
```

CREATING DIFFERENT TYPES OF PLOTS for :

Symptoms

AGE

GENDER

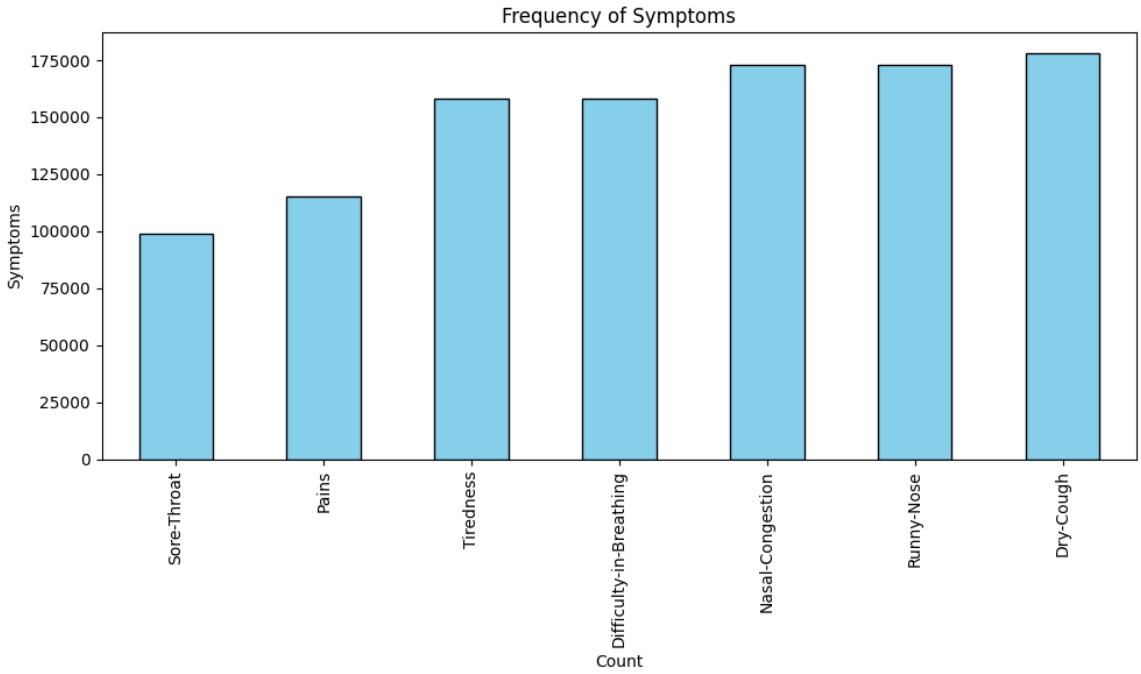
.

BAR PLOT

To see the count of specific Symptoms

.

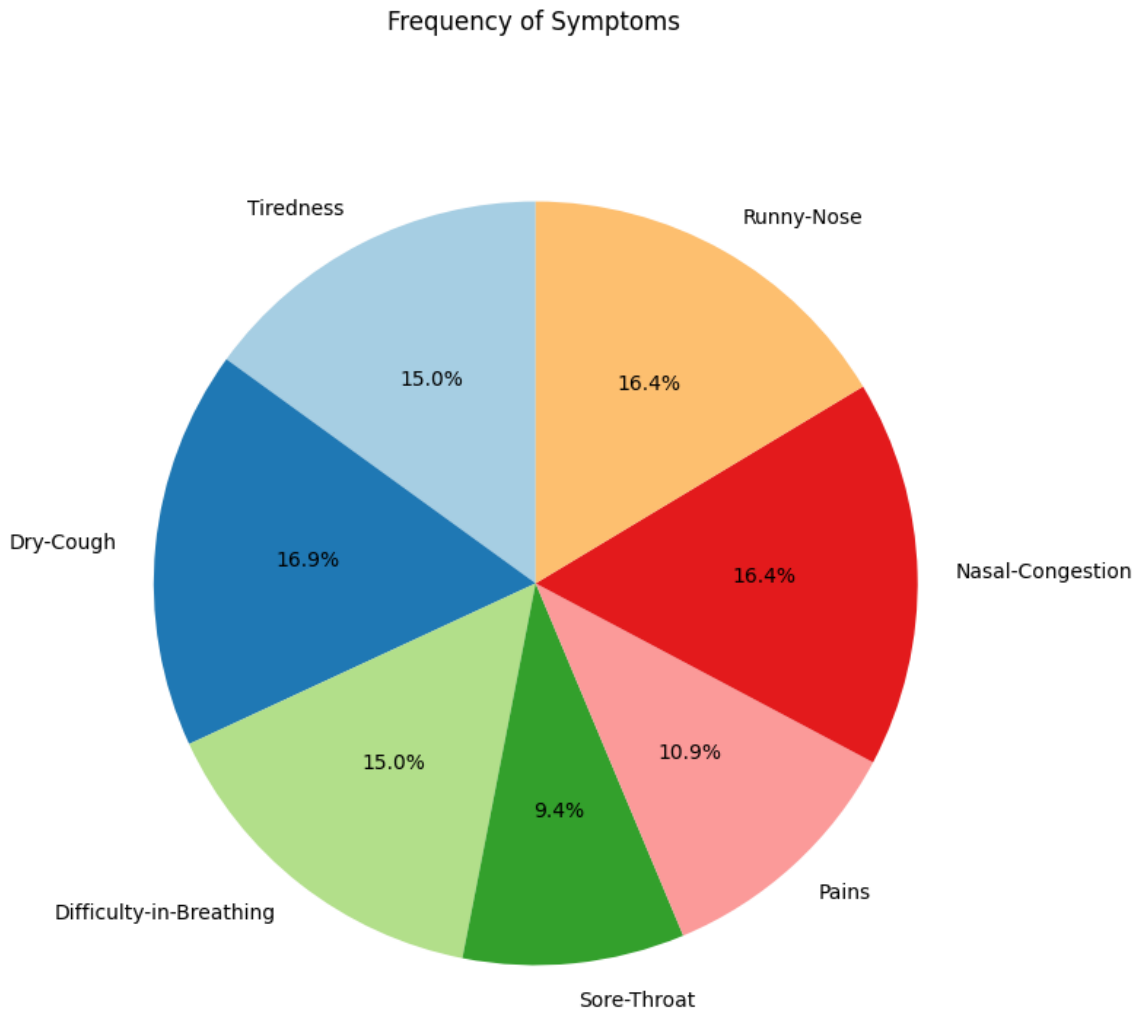
In [49]:
```
fig, ax = plt.subplots(figsize=(10, 6))

symptom_counts.sort_values().plot(kind='bar', ax=ax, color='skyblue',
                                  edgecolor='black')
ax.set_title('Frequency of Symptoms')
ax.set_xlabel('Count')
ax.set_ylabel('Symptoms')

plt.tight_layout()
plt.show()
```



PIE PLOT

.

In [50]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(symptom_counts, labels=symptom_counts.index, autopct='%1.1f%%',
       startangle=90,
       colors=plt.cm.Paired(range(len(symptom_counts))))
ax.set_title('Frequency of Symptoms')
plt.axis('equal')
plt.tight_layout()
plt.show()
```



BAR PLOT

.

In [51]:
```
fig = px.bar(symptom_counts, x=symptom_counts.index, y=symptom_counts.val
             labels={'x': 'Symptoms', 'y': 'Count'},
             title='Frequency of Symptoms', color_discrete_sequence=['sky
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```

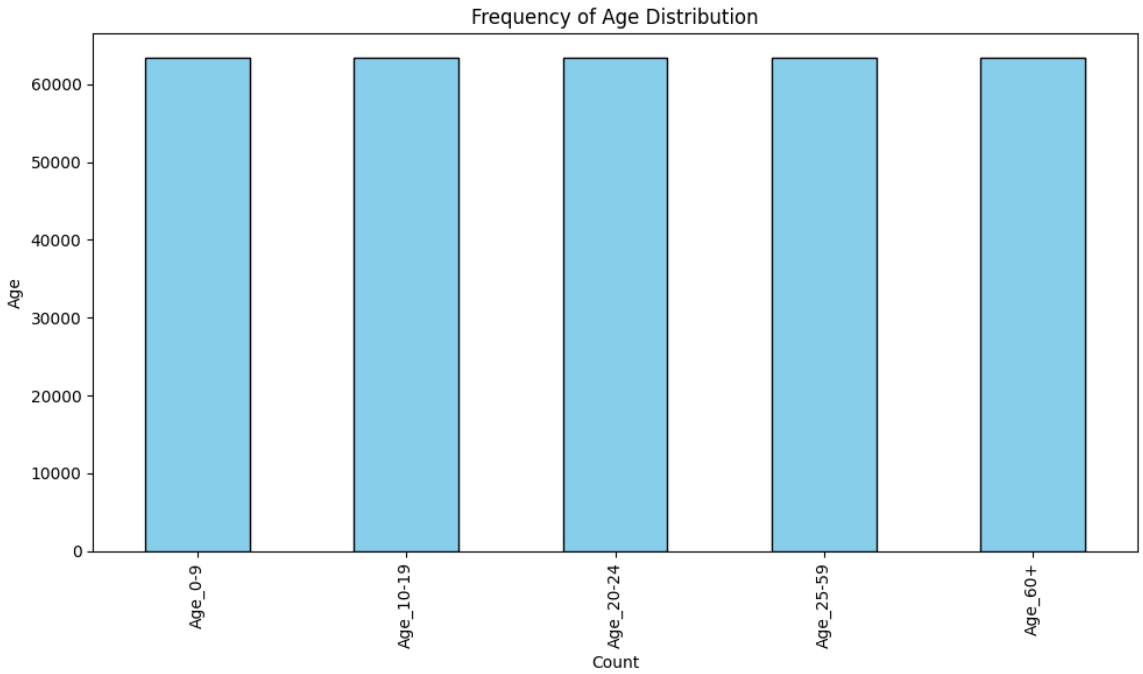
WE CAN SEE above DRY COUGH IS THE most common symptom and highest in numbers

In [52]:
```
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '
fig = px.pie(symptom_counts, values=symptom_counts.values, names=symptom_
             title='Frequency of Symptoms', color_discrete_sequence=custo
fig.show()
```

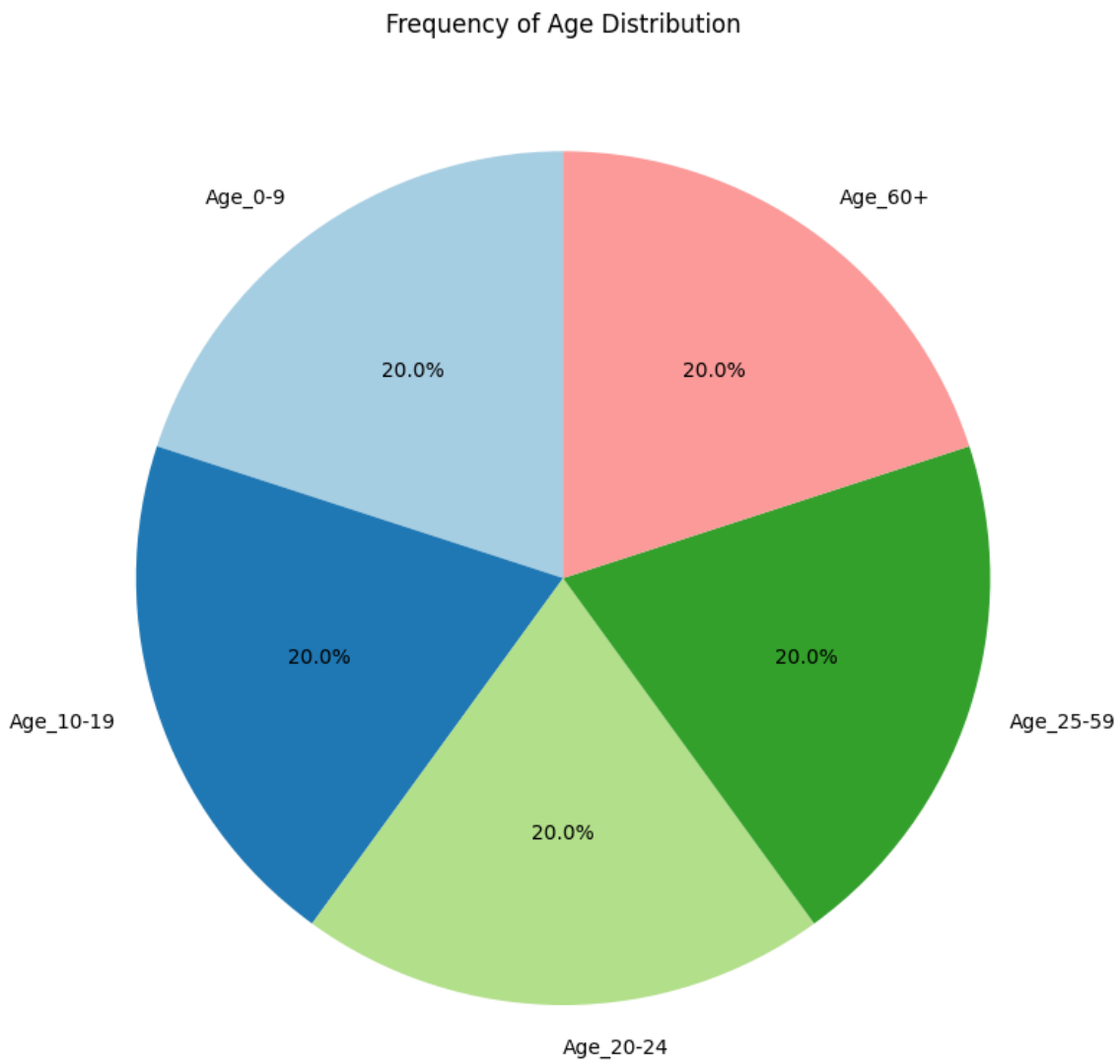
PLOT OF AGE DISTRIBUTION

.

In [53]:
```
fig, ax = plt.subplots(figsize=(10, 6))

age_distribution.sort_values().plot(kind='bar', ax=ax, color='skyblue',
                                    edgecolor='black')
ax.set_title('Frequency of Age Distribution')
ax.set_xlabel('Count')
ax.set_ylabel('Age')

plt.tight_layout()
plt.show()
```



In [54]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(age_distribution, labels=age_distribution.index, autopct='%1.1f%%'
       startangle=90,
       colors=plt.cm.Paired(range(len(age_distribution))))
ax.set_title('Frequency of Age Distribution')
plt.axis('equal')
plt.tight_layout()
plt.show()
```



In [55]:
```
fig = px.bar(age_distribution, x=age_distribution.index, y=age_distributi
             labels={'x': 'Age Distribution', 'y': 'Count'},
             title='Frequency of Age Distribution', color_discrete_sequen
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```

In [56]:
```
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '
fig = px.pie(age_distribution, values=age_distribution.values, names=age_
             title='Frequency of Age Distribution', color_discrete_sequen
fig.show()
```

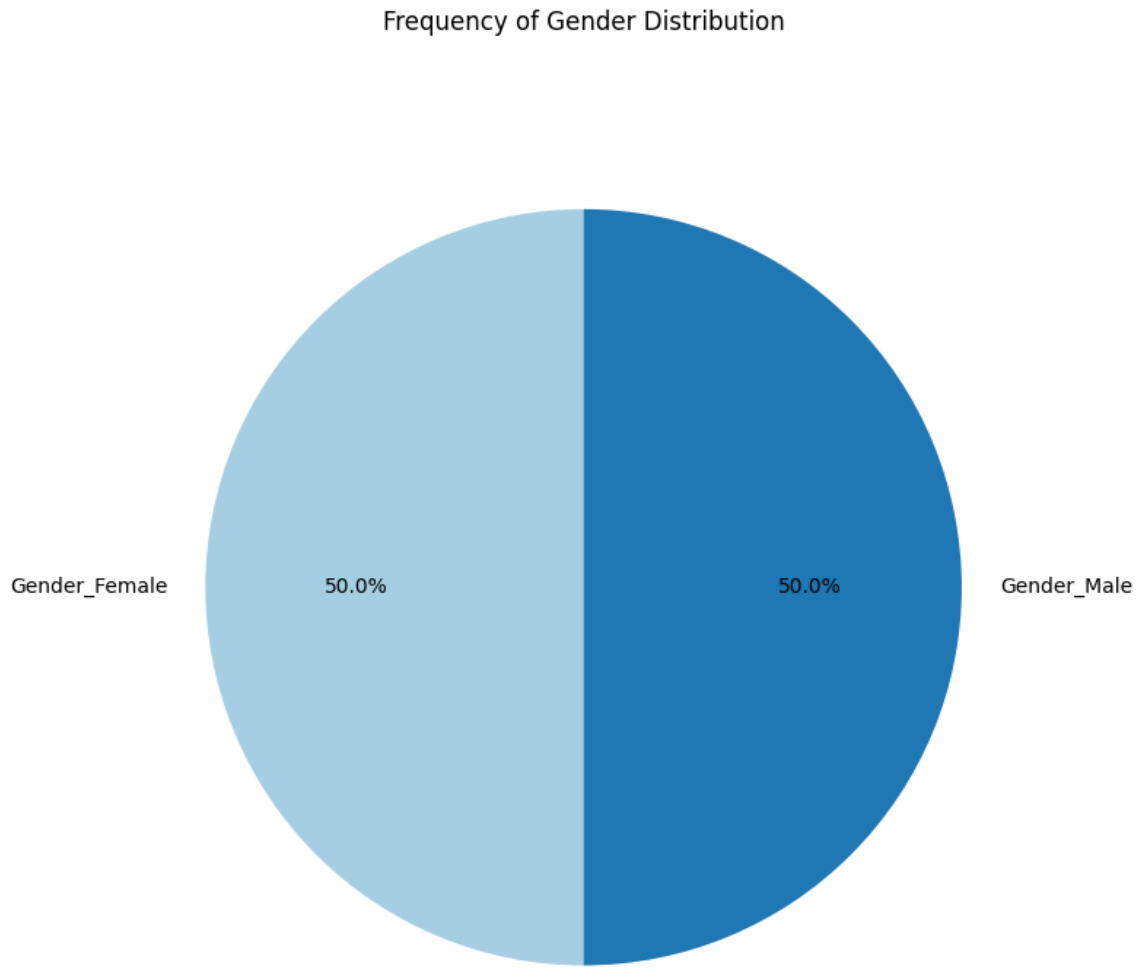
PLOT OF GENDER DISTRIBUTION

.

In [57]:
```
fig, ax = plt.subplots(figsize=(10, 6))

gender_distribution.sort_values().plot(kind='bar', ax=ax, color='skyblue'
                                       edgecolor='black')
ax.set_title('Frequency of Gender Distribution')
ax.set_xlabel('Count')
ax.set_ylabel('Gender')
```

```
plt.tight_layout()
plt.show()
```

In [58]:
```
fig, ax = plt.subplots(figsize=(8, 8))
ax.pie(gender_distribution, labels=gender_distribution.index, autopct='%1
       startangle=90,
       colors=plt.cm.Paired(range(len(age_distribution))))
ax.set_title('Frequency of Gender Distribution')
plt.axis('equal')
plt.tight_layout()
plt.show()
```



In [59]:
```
fig = px.bar(gender_distribution, x=gender_distribution.index, y=gender_d
             labels={'x': 'Gender Distribution', 'y': 'Count'},
             title='Frequency of Gender Distribution', color_discrete_seq
fig.update_layout(xaxis_tickangle=-45)
fig.show()
```

In [60]:
```
custom_colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '
fig = px.pie(gender_distribution, values=gender_distribution.values, name
             title='Frequency of Gender Distribution', color_discrete_seq
fig.show()
```

SO WE SAW AGE GROUPS AND GENDER ARE VERY EQUALLLY DISTRIBUTED

..

.corr():

This method calculates the correlation matrix for these columns. The correlation matrix is a square matrix where each entry represents the correlation coefficient between two columns. The correlation coefficient measures the degree of linear relationship between

two variables, and it can range from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no linear correlation.

The result of this code will be a DataFrame containing the correlation coefficients between the selected symptom columns in df. The matrix will show how each symptom correlates with every other symptom in the subset, providing insights into the relationships between these variables.

In [61]:
```
corr = df[symptoms].corr()
```

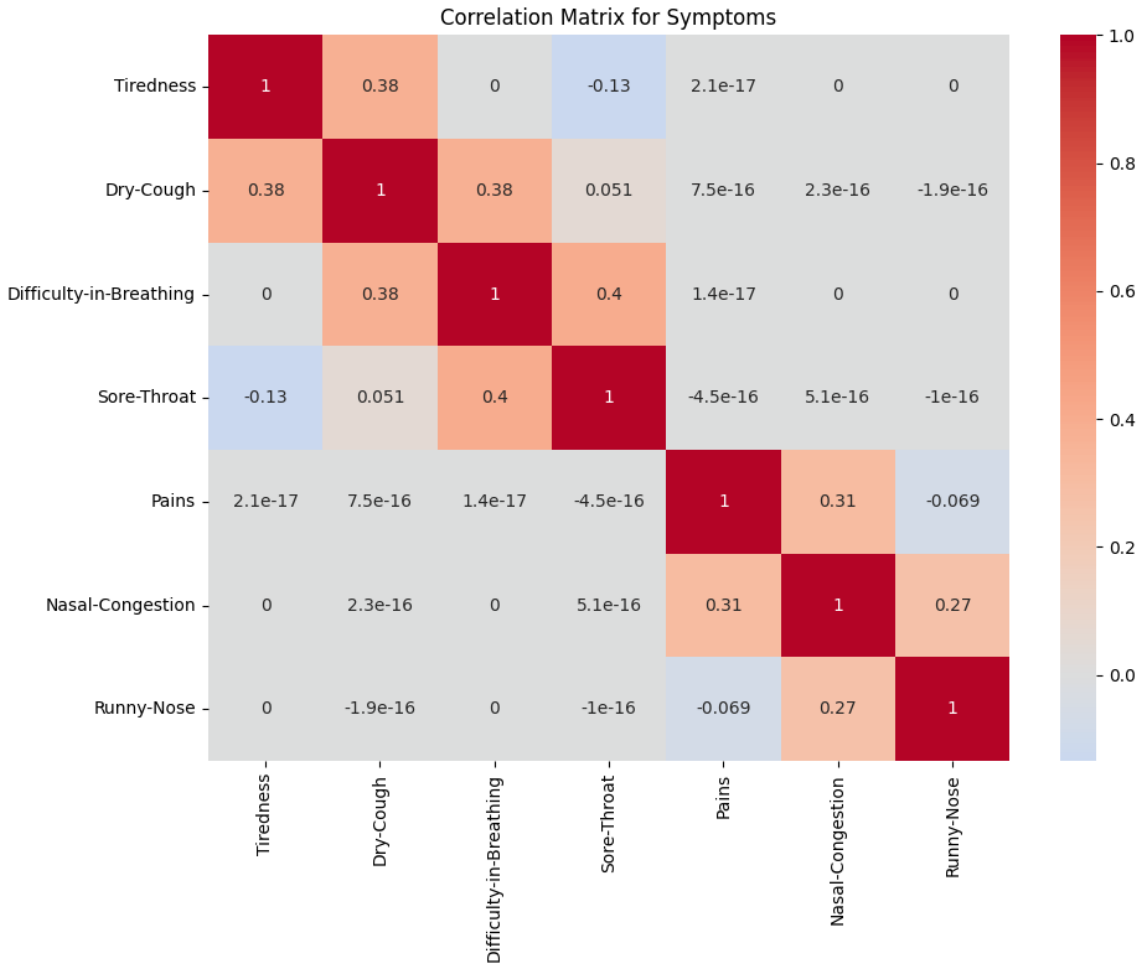
In [62]:
```
corr
```

Out[62]:

	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	Pains	Cong
Tiredness	1.000000e+00	3.779645e-01	0.000000e+00	-1.348400e-01	2.099071e-17	0.00000
Dry-Cough	3.779645e-01	1.000000e+00	3.779645e-01	5.096472e-02	7.542460e-16	2.2814
Difficulty-in-Breathing	0.000000e+00	3.779645e-01	1.000000e+00	4.045199e-01	1.399697e-17	0.00000
Sore-Throat	-1.348400e-01	5.096472e-02	4.045199e-01	1.000000e+00	-4.451275e-16	5.1020
Pains	2.099071e-17	7.542460e-16	1.399697e-17	-4.451275e-16	1.000000e+00	3.1052
Nasal-Congestion	0.000000e+00	2.281412e-16	0.000000e+00	5.102065e-16	3.105295e-01	1.00000
Runny-Nose	0.000000e+00	-1.889722e-16	0.000000e+00	-1.020501e-16	-6.900656e-02	2.6666

PLOTTINGGGG

In [63]:
```
fig, ax = plt.subplots(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0, ax=ax)
ax.set_title('Correlation Matrix for Symptoms')
plt.tight_layout()
plt.show()
```



Calculating mean for all the columns

In [64]:
```
severity_symptom_means = df.groupby(by=[col for col in df.columns if "Ast
```

In [65]:
```
severity_symptom_means
```

Out[65]:

Asthma_Severity	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	Pains	Nasal-Congestion	Runny-Nose
0	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455
1	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455

In [66]:
```
severity_symptom_means
```

Out[66]:

Asthma_Severity	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	Pains	Nasal-Congestion	Runny-Nose
0	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455
1	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455

Simple transpose

In [72]:
```
severity_symptom_means = severity_symptom_means.transpose()
```

In [73]:
```
severity_symptom_means
```

Out[73]:

Asthma_Severity	0	1
Tiredness	0.500000	0.500000
Dry-Cough	0.562500	0.562500
Difficulty-in-Breathing	0.500000	0.500000
Sore-Throat	0.312500	0.312500
Pains	0.363636	0.363636
Nasal-Congestion	0.545455	0.545455
Runny-Nose	0.545455	0.545455

CREATING NEW DATA based on this mean table

In [67]:
```
data = {
    'Asthma_Severity': [0, 1, 2],
    'Tiredness': [0.500000, 0.500000, 0.500000],
    'Dry-Cough': [0.562500, 0.562500, 0.562500],
    'Difficulty-in-Breathing': [0.500000, 0.500000, 0.500000],
    'Sore-Throat': [0.312500, 0.312500, 0.312500],
    'Pains': [0.363636, 0.363636, 0.363636],
    'Nasal-Congestion': [0.545455, 0.545455, 0.545455],
    'Runny-Nose': [0.545455, 0.545455, 0.545455]
}
```

In [68]:
```
df1 = pd.DataFrame(data)
```

In [70]:
```
df1.set_index('Asthma_Severity' , inplace=True)
```

In [71]:
```
df1
```

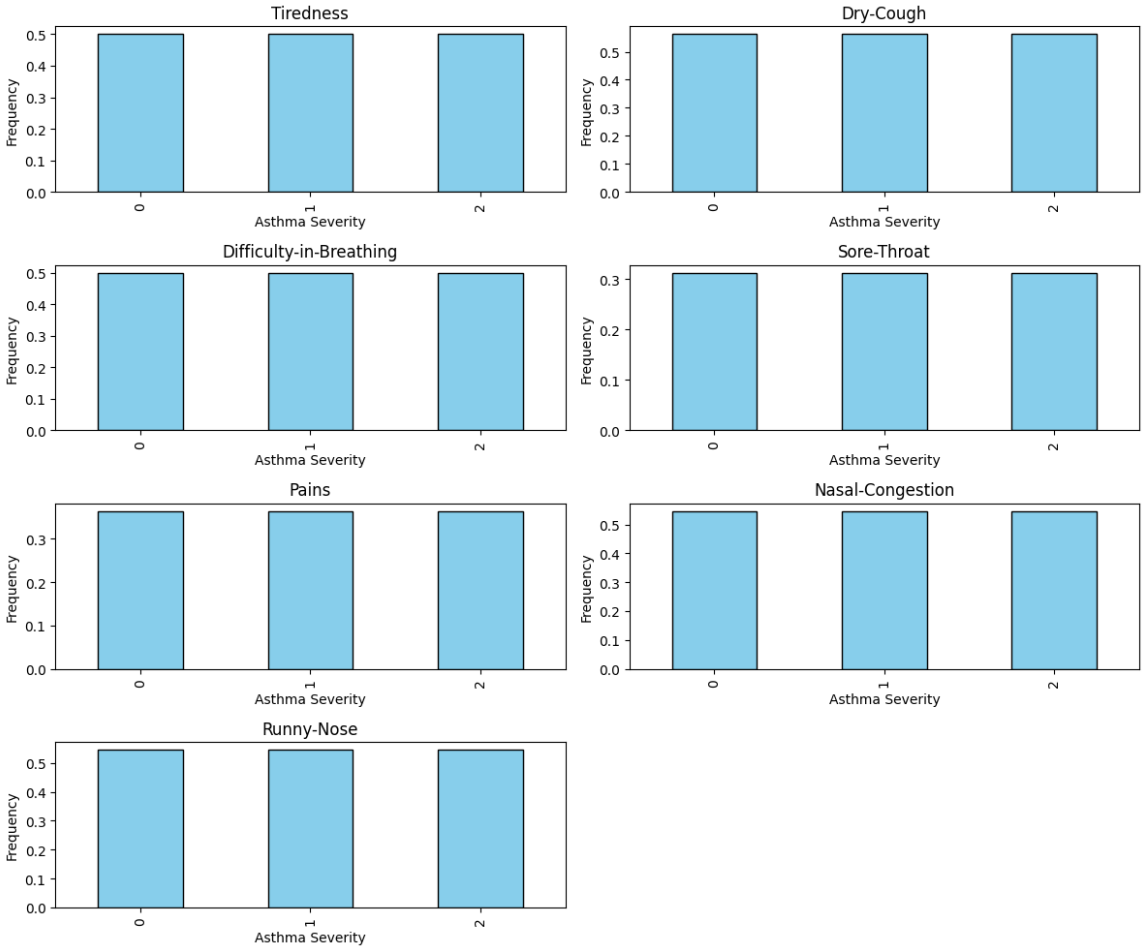
Out[71]:

Asthma_Severity	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	Pains	Nasal-Congestion	Runny-Nose
0	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455
1	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455
2	0.5	0.5625	0.5	0.3125	0.363636	0.545455	0.545455

PLOTTT

In [74]:
```
fig, axes = plt.subplots(nrows=4, ncols=2, figsize=(12, 10))
symptoms = ['Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing', 'Sore-Th

for i, symptom in enumerate(symptoms):
    row, col = divmod(i, 2)
    df1[symptom].plot(kind='bar', ax=axes[row, col], color='skyblue', edg
    axes[row, col].set_title(symptom)
    axes[row, col].set_xlabel('Asthma Severity')
    axes[row, col].set_ylabel('Frequency')

if len(symptoms) < len(axes.flat):
```

```
    for i in range(len(symptoms), len(axes.flat)):
        fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```



old df

In [75]:
```
df
```

Out[75]:

	Tiredness	Dry-Cough	Difficulty-in-Breathing	Sore-Throat	None_Sympton	Pains	Nasal-Congestion	Runny-Nose
0	1	1	1	1	0	1	1	1
1	1	1	1	1	0	1	1	1
2	1	1	1	1	0	1	1	1
3	1	1	1	1	0	1	1	1
4	1	1	1	1	0	1	1	1
...	...	...	...	...	...	...	...	...
316795	0	0	0	0	1	0	0	0
316796	0	0	0	0	1	0	0	0
316797	0	0	0	0	1	0	0	0
316798	0	0	0	0	1	0	0	0
316799	0	0	0	0	1	0	0	0

316800 rows × 17 columns

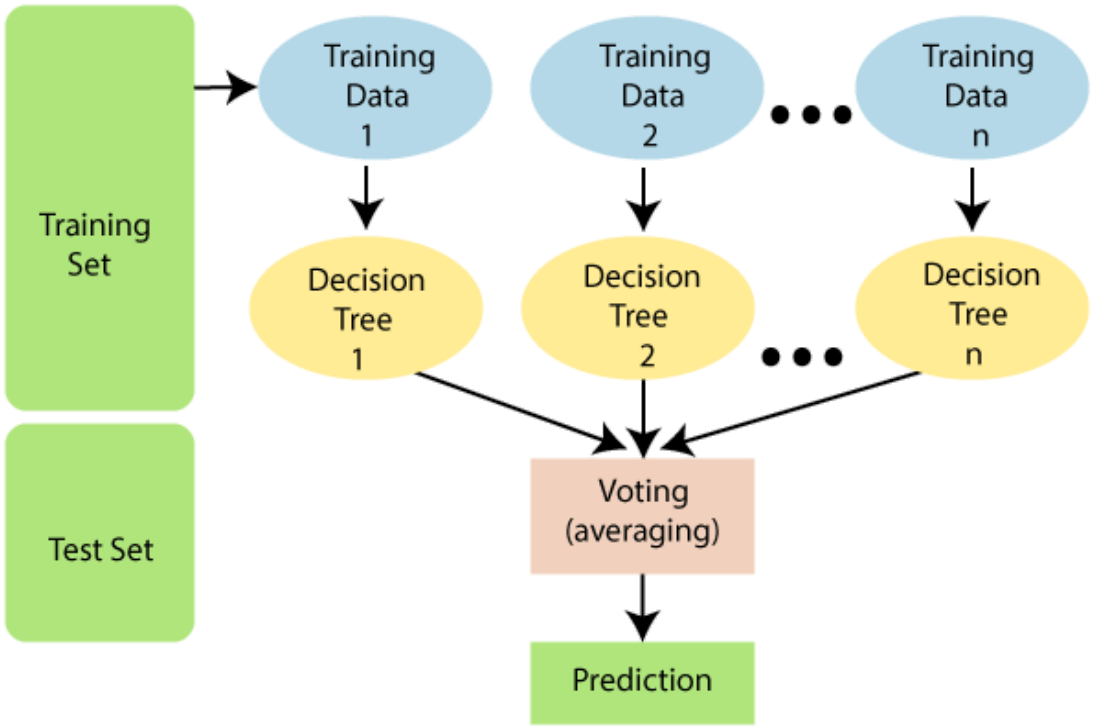
Feature Selection

Where Asthma Severity will be our Target Variable

In [79]:
```
x = df.drop(['Asthma_Severity'] , axis = 1)

y = df['Asthma_Severity']
```

MODEL TRAINING

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



RANDOM FOREST IS A DECISION MADE FROM MULTIPLE DECISION TREEES

. .

In [80]:
```
from sklearn.ensemble import RandomForestClassifier
```

n_estimators=100: This parameter specifies the number of decision trees (estimators) to include in the random forest.

random_state=42: This parameter sets the random seed for the random number generator used by the random forest algorithm.

In [81]:
```
clf = RandomForestClassifier(n_estimators=100 , random_state=42)
```

In [82]:
```
clf.fit(x , y)
```

Out[82]:
```
▼ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

feature_importances = clf.feature*importances*

is used to retrieve the feature importances from a trained Random Forest classifier (clf) after it has been fit to a dataset. Feature importances provide information about the contribution of each feature (or variable) in the dataset to the model's predictions. These importances are typically expressed as values between 0 and 1, and they sum up to 1 across all features.

In [83]:
```
feature_importances = clf.feature_importances_
```

In [84]:
```
feature_importance_df = pd.DataFrame({'Feature' : x.columns , "Importance
```

In [85]:
```
feature_importance_df = feature_importance_df.sort_values(by='Importance'
```

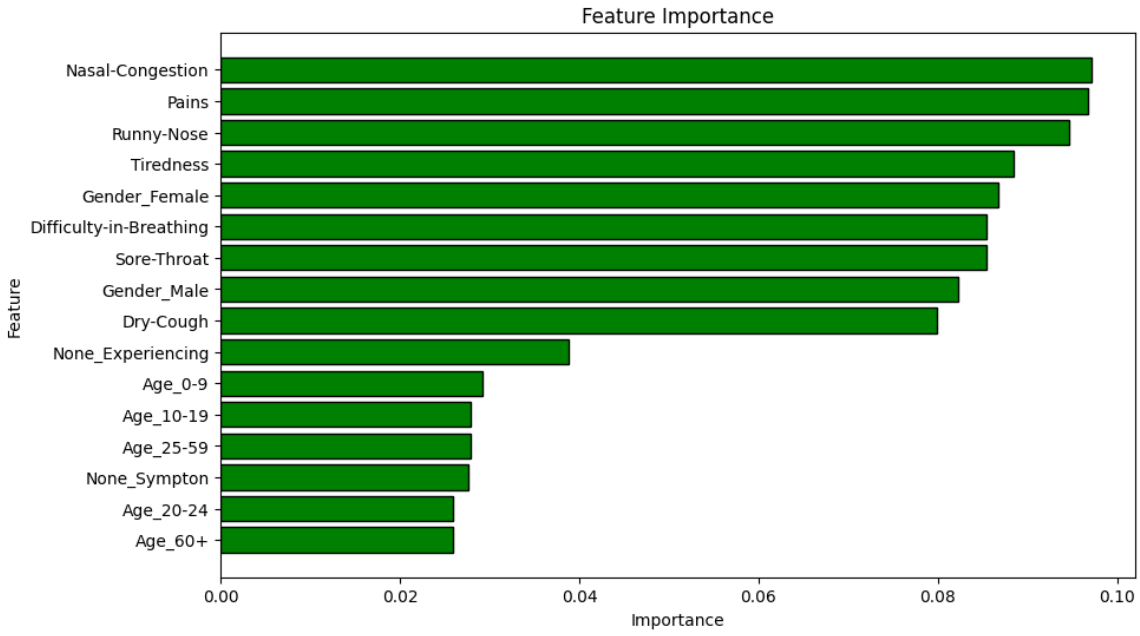
In [86]:
```
feature_importance_df
```

Out[86]:

	Feature	Importance
6	Nasal-Congestion	0.097225
5	Pains	0.096774
7	Runny-Nose	0.094631
0	Tiredness	0.088506
14	Gender_Female	0.086758
2	Difficulty-in-Breathing	0.085456
3	Sore-Throat	0.085436
15	Gender_Male	0.082218
1	Dry-Cough	0.079887
8	None_Experiencing	0.038831
9	Age_0-9	0.029175
10	Age_10-19	0.027898
12	Age_25-59	0.027824
4	None_Sympton	0.027545
11	Age_20-24	0.025962
13	Age_60+	0.025874

PLOTT THE SAME

In [87]:
```
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importa
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.gca().invert_yaxis()
```



In [88]:
```
X = df[symptoms + gender_groups]
```

In [89]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

Splitting Data into Train and Test Sets

.

.

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

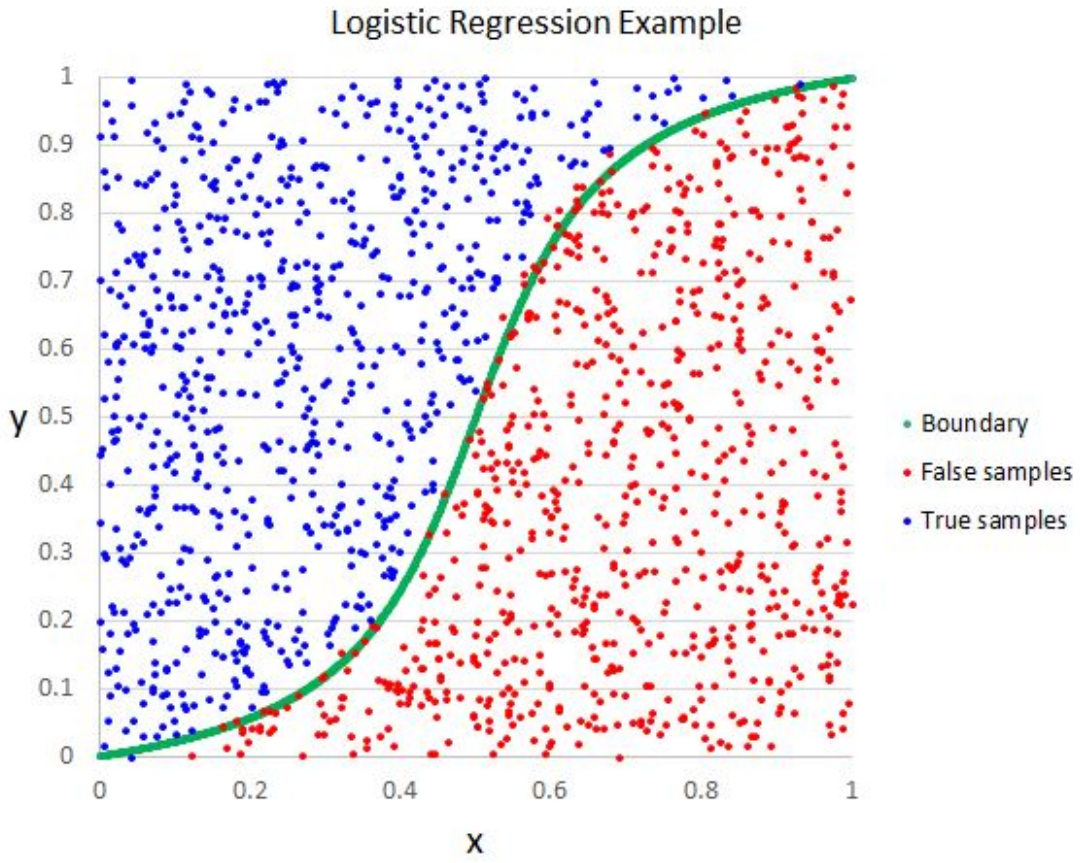
In [93]:
```
X_train,X_test,y_train,y_test=train_test_split(X,
                                               y,
                                               test_size=0.2,
                                               stratify = y,
                                               random_state=42)
```

In [94]:
```
from sklearn.metrics import accuracy_score , confusion_matrix , classific
```

LOGISTIC REGRESSION

.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html



Logistic Regression is one of the basic and popular algorithms to solve a classification problem

In [95]:
```
lr = LogisticRegression(random_state=42)
```

In [96]:
```
lr.fit(X_train , y_train)
```

Out[96]:
```
▼ LogisticRegression
LogisticRegression(random_state=42)
```

In [97]:
```
y_pred_lr = lr.predict(X_test)
```

In [98]:
```
accuracy = accuracy_score(y_test , y_pred_lr)
print(f'Accuracy : {accuracy :.2f}')
```

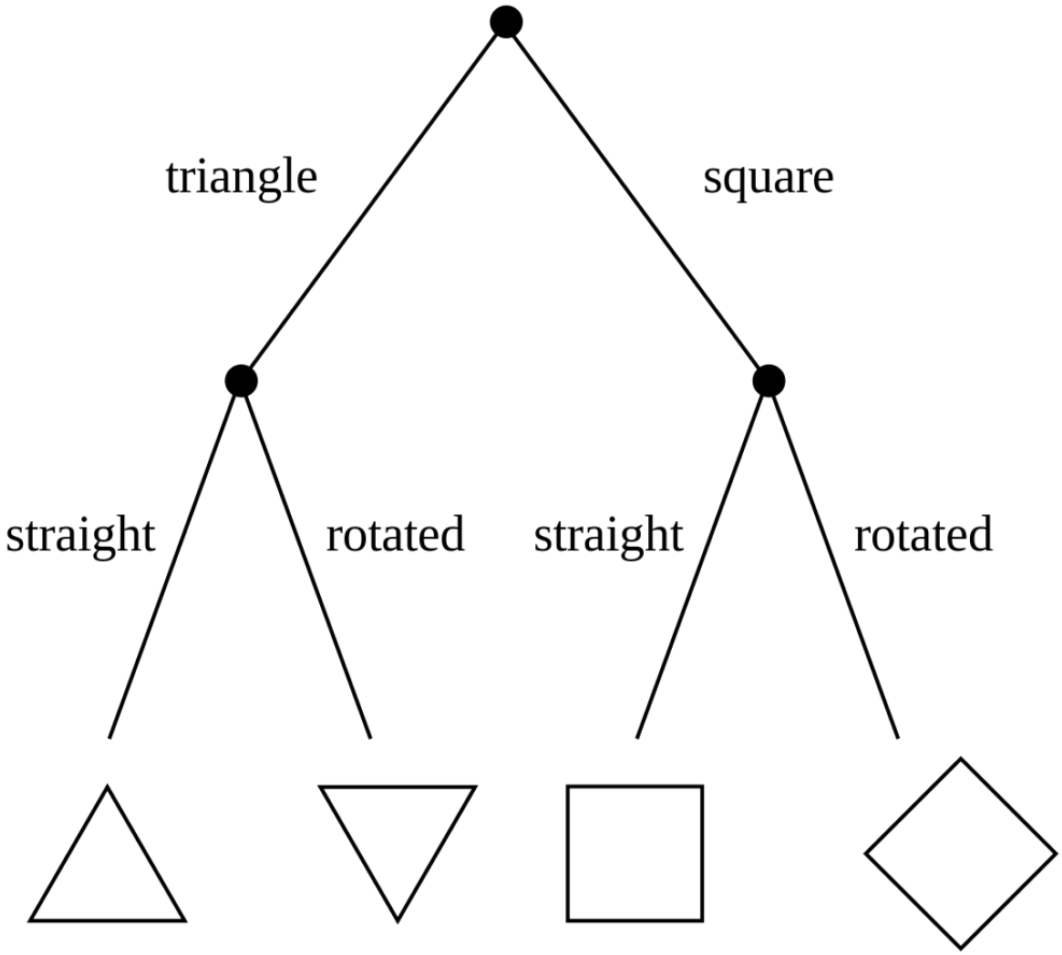
```
Accuracy : 0.75
```

DECISION TREE

.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.

https://scikit-learn.org/stable/modules/tree.html



In [99]:
```
dt = DecisionTreeClassifier()
```

In [100…
```
model_dt = dt.fit(X_train , y_train)
```

In [101…
```
y_pred_dt  = model_dt.predict(X_test)
```

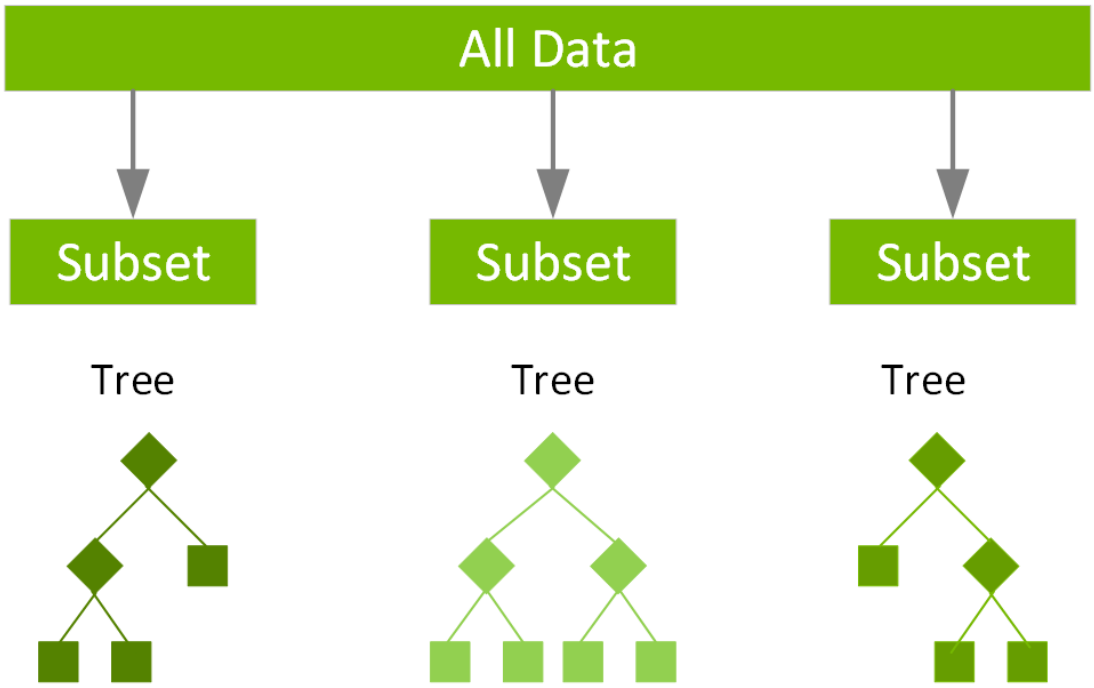
In [102…
```
accuracy = accuracy_score(y_test, y_pred_dt)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.75
```

XGBoost

is an optimized distributed gradient boosting library designed for efficient and scalable training of machine learning models.

https://www.geeksforgeeks.org/xgboost/



In [106…
```
import xgboost as xgb
```

In [108…
```
clf = xgb.XGBClassifier(
    objective='multi:softmax',
    num_class=len(pd.unique(y)),
    eval_metric='mlogloss',
    use_label_encoder=False
)
```

In [109…
```
clf.fit(X_train, y_train)
```

Out[109]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping
_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=Non
e,
              learning_rate=None, max_bin=None, max_cat_threshol
d=None,
              max_cat_to_onehot=None, max_delta_step=None, max_d
```

In [110…
```
y_pred_xgb = clf.predict(X_test)
```

In [111…
```
accuracy = accuracy_score(y_test, y_pred_xgb)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.75
```

So , in summary we were able to create a machine learning model for "Asthama Prediciton using supervised Data" where we tried multiple machine learning techniques like LOGISTIC REGRESSION , DECISION TREES , XGBOOST and we could achieve an accuracy of

0.75

THANKYOU !!!!!