Target Domain:

This Research focuses on Analysing the data collected from https://www.kaggle.com/datasets/omarlarasa/cov19-open-data-mexico?resource=download (https://www.kaggle.com/datasets/omarlarasa/cov19-open-data-mexico?resource=download), Originated from Mexican Government.

This Research Aims to indentify the feature ranking's that are affecting the level of covid-19.

Approach:

Building machine learning models to predict the outcomes and severity of COVID using the given dataset is a valuable endeavor. :

1) Data Preprocessing:

Handle Missing Values: Analyze the dataset for missing values in each column and apply appropriate strategies to handle them (e.g., imputation, dropping rows/columns). Data Encoding: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding. Feature Scaling: Scale numerical features to bring them to a similar scale. Common techniques include standardization or normalization.

2) Data Exploration and Visualization:

Understand the distribution of different variables and their relationships with the target variable (e.g., COVID outcome). Use visualizations to gain insights into patterns and correlations in the data, which can help inform feature selection and model choices.

3) Feature Selection:

Select relevant features for building the prediction models. one can use techniques like correlation analysis, feature importance from tree-based models, or domain knowledge to identify the most significant features.

4) Model Selection:

Given that you want to predict COVID outcomes and severity, consider using classification algorithms. Some suitable models to consider include:

Logistic Regression: A simple and interpretable model for binary classification tasks.

Random Forest: A powerful ensemble method that can handle complex interactions and provide feature importance.

Gradient Boosting: Another ensemble method that can achieve high predictive performance.

Support Vector Machines (SVM): A strong performer for binary classification tasks.

Neural Networks: Deep learning models can capture intricate patterns in data but require more data and computational resources. Model Training and Evaluation:

5) Split the dataset:

Into training and testing sets to train the models and evaluate their performance. Use appropriate evaluation metrics like accuracy, precision, recall, F1-score, ROC-AUC, etc., to assess the models' effectiveness.

Note:

Consider using cross-validation to obtain a more robust estimate of model performance. Hyperparameter Tuning:

Fine-tune the hyperparameters of the chosen models to optimize their performance. Use techniques like grid search or random search to find the best hyperparameter combinations.

6) Model Interpretation:

Depending on the chosen models (e.g., logistic regression or decision trees), interpret the model results to understand which features are most influential in predicting COVID outcomes.

Note: Ensembling (Optional):

Consider creating an ensemble model by combining predictions from multiple individual models, as it may further enhance predictive performance.

7) Reporting and Visualization:

Summarize the results of the analysis, including model performance, important features, and any insights gained from the data, in a clear and actionable manner, preferably using SHAP.

Deployment (Optional):

DATA PREPORCESSING:

let's start with data preprocessing. This step involves preparing the dataset for training machine learning models by handling missing values, encoding categorical variables, and performing feature scaling. Here's how you can do it:

Handling Missing Values:

Identify columns with missing values and decide how to handle them. You can either:

plan:

Impute missing values using strategies such as mean, median, mode, or advanced imputation techniques like K-Nearest Neighbors (KNN) or Multiple Imputation.

If a significant portion of the data is missing in a column, consider dropping the column entirely if it does not provide valuable information.

Data Encoding:

Identify categorical variables in the dataset. These are variables that have discrete categories like 'SEX', 'NATIONALITY', 'PREGNANCY', etc.

For nominal categorical variables (categories with no intrinsic ordering), use one-hot encoding to convert them into binary columns representing each category.

For ordinal categorical variables (categories with a natural order), use label encoding to map categories to integer values.

Feature Scaling:

Scale numerical features to bring them to a similar scale. This step is crucial for models that are sensitive to feature scales, such as SVM or neural networks.

Common scaling techniques include standardization (scaling to have zero mean and unit variance) and

In [1]:

```
import pandas as pd

df = pd.read_csv('C:/Users/imran/Desktop/Machine_learning_Projects/Covid19_Analysis/data

df
```

Out[1]:

	FECHA_ACTUALIZACION	ID_REGISTRO	ORIGEN	SECTOR	ENTIDAD_UM	SEXO	I
0	2021-05-03	z482b8	1	12	9	2	
1	2021-05-03	z49a69	1	12	23	1	
2	2021-05-03	z23d9d	1	12	22	2	
3	2021-05-03	z24953	1	12	9	1	
4	2021-05-03	zz8e77	1	12	9	2	
...	...	...	...	...	...	...	
6659179	2021-05-03	a1ac82	1	12	19	1	
6659180	2021-05-03	c70a75	1	6	9	2	
6659181	2021-05-03	72db0f	2	12	9	2	
6659182	2021-05-03	caf404	2	12	9	2	
6659183	2021-05-03	b7490c	2	4	5	2	

6659184 rows × 40 columns

In [2]:

```
#let us now convert the data into english and first we list the columns

# List all the columns after renaming
columns_list = df.columns.tolist()
print(columns_list)
```

```
['FECHA_ACTUALIZACION', 'ID_REGISTRO', 'ORIGEN', 'SECTOR', 'ENTIDAD_UM',
'SEXO', 'ENTIDAD_NAC', 'ENTIDAD_RES', 'MUNICIPIO_RES', 'TIPO_PACIENTE', 'F
ECHA_INGRESO', 'FECHA_SINTOMAS', 'FECHA_DEF', 'INTUBADO', 'NEUMONIA', 'EDA
D', 'NACIONALIDAD', 'EMBARAZO', 'HABLA_LENGUA_INDIG', 'INDIGENA', 'DIABETE
S', 'EPOC', 'ASMA', 'INMUSUPR', 'HIPERTENSION', 'OTRA_COM', 'CARDIOVASCULA
R', 'OBESIDAD', 'RENAL_CRONICA', 'TABAQUISMO', 'OTRO_CASO', 'TOMA_MUESTRA_
LAB', 'RESULTADO_LAB', 'TOMA_MUESTRA_ANTIGENO', 'RESULTADO_ANTIGENO', 'CLA
SIFICACION_FINAL', 'MIGRANTE', 'PAIS_NACIONALIDAD', 'PAIS_ORIGEN', 'UCI']
```

In [3]:

```
# New dictionary for column name translations
translation_dict = {
    'FECHA_ACTUALIZACION': 'UPDATE_DATE',
    'ID_REGISTRO': 'REGISTRATION_ID',
    'ORIGEN': 'ORIGIN',
    'SECTOR': 'SECTOR',
    'ENTIDAD_UM': 'ENTITY_UM',
    'SEXO': 'SEX',
    'ENTIDAD_NAC': 'ENTITY_BORN',
    'ENTIDAD_RES': 'ENTITY_RES',
    'MUNICIPIO_RES': 'MUNICIPALITY_RES',
    'TIPO_PACIENTE': 'PATIENT_TYPE',
    'FECHA_INGRESO': 'ADMISSION_DATE',
    'FECHA_SINTOMAS': 'SYMPTOMS_DATE',
    'FECHA_DEF': 'DEATH_DATE',
    'INTUBADO': 'INTUBATED',
    'NEUMONIA': 'PNEUMONIA',
    'EDAD': 'AGE',
    'NACIONALIDAD': 'NATIONALITY',
    'EMBARAZO': 'PREGNANCY',
    'HABLA_LENGUA_INDIG': 'SPEAKS_INDIG_LANG',
    'INDIGENA': 'INDIGENOUS',
    'DIABETES': 'DIABETES',
    'EPOC': 'COPD',
    'ASMA': 'ASTHMA',
    'INMUSUPR': 'IMMUNOSUPPRESSION',
    'HIPERTENSION': 'HYPERTENSION',
    'OTRA_COM': 'OTHER_COMORBIDITIES',
    'CARDIOVASCULAR': 'CARDIOVASCULAR',
    'OBESIDAD': 'OBESITY',
    'RENAL_CRONICA': 'CHRONIC_RENAL_FAILURE',
    'TABAQUISMO': 'SMOKING',
    'OTRO_CASO': 'ANOTHER_CASE',
    'TOMA_MUESTRA_LAB': 'LAB_SAMPLE',
    'RESULTADO_LAB': 'LAB_RESULT',
    'TOMA_MUESTRA_ANTIGENO': 'ANTIGEN_SAMPLE',
    'RESULTADO_ANTIGENO': 'ANTIGEN_RESULT',
    'CLASIFICACION_FINAL': 'FINAL_CLASSIFICATION',
    'MIGRANTE': 'MIGRANT',
    'PAIS_NACIONALIDAD': 'NATIONALITY_COUNTRY',
    'PAIS_ORIGEN': 'ORIGIN_COUNTRY',
    'UCI': 'ICU'
}
```

In [4]:

```python



# Rename the columns using the translation_dict
df.rename(columns=translation_dict, inplace=True)
df
```

Out[4]:

	UPDATE_DATE	REGISTRATION_ID	ORIGIN	SECTOR	ENTITY_UM	SEX	ENTITY_B
0	2021-05-03	z482b8	1	12	9	2	
1	2021-05-03	z49a69	1	12	23	1	
2	2021-05-03	z23d9d	1	12	22	2	
3	2021-05-03	z24953	1	12	9	1	
4	2021-05-03	zz8e77	1	12	9	2	
...	...	...	...	...	...	...	
6659179	2021-05-03	a1ac82	1	12	19	1	
6659180	2021-05-03	c70a75	1	6	9	2	
6659181	2021-05-03	72db0f	2	12	9	2	
6659182	2021-05-03	caf404	2	12	9	2	
6659183	2021-05-03	b7490c	2	4	5	2	

6659184 rows × 40 columns

In [5]:

```python
#now let us see all the colums that we have again this time in english



columns_list = df.columns.tolist()
print(columns_list)
```

```
['UPDATE_DATE', 'REGISTRATION_ID', 'ORIGIN', 'SECTOR', 'ENTITY_UM', 'SEX',
'ENTITY_BORN', 'ENTITY_RES', 'MUNICIPALITY_RES', 'PATIENT_TYPE', 'ADMISSIO
N_DATE', 'SYMPTOMS_DATE', 'DEATH_DATE', 'INTUBATED', 'PNEUMONIA', 'AGE',
'NATIONALITY', 'PREGNANCY', 'SPEAKS_INDIG_LANG', 'INDIGENOUS', 'DIABETES',
'COPD', 'ASTHMA', 'IMMUNOSUPPRESSION', 'HYPERTENSION', 'OTHER_COMORBIDITIE
S', 'CARDIOVASCULAR', 'OBESITY', 'CHRONIC_RENAL_FAILURE', 'SMOKING', 'ANOT
HER_CASE', 'LAB_SAMPLE', 'LAB_RESULT', 'ANTIGEN_SAMPLE', 'ANTIGEN_RESULT',
'FINAL_CLASSIFICATION', 'MIGRANT', 'NATIONALITY_COUNTRY', 'ORIGIN_COUNTR
Y', 'ICU']
```

In [6]:

```python
df.shape
```

Out[6]:

```
(6659184, 40)
```

In [ ]:

```
```

In [ ]:

```
```

Understanding My data:

```
UPDATE_DATE: Date of the last update for the record.
```

REGISTRATION_ID: Identifier for the patient registration.

ORIGIN: Origin of the patient (where they came from).

SECTOR: Sector or area associated with the patient.

ENTITY_UM: Entity where the patient is being treated.

SEX: Gender of the patient (Male/Female).

ENTITY_BORN: Entity where the patient was born.

ENTITY_RES: Entity where the patient currently resides.

MUNICIPALITY_RES: Municipality where the patient currently resides.

PATIENT_TYPE: Type of patient (possibly ordinal categorical, as per the previous code).

ADMISSION_DATE: Date of patient admission.

SYMPTOMS_DATE: Date when symptoms were first observed.

DEATH_DATE: Date of patient death.

INTUBATED: Indicates whether the patient was intubated or not.

PNEUMONIA: Indicates if the patient had pneumonia.

AGE: Age of the patient.

NATIONALITY: Nationality of the patient.

PREGNANCY: Indicates if the patient is pregnant or not.

SPEAKS_INDIG_LANG: Indicates if the patient speaks an indigenous language.

INDIGENOUS: Indicates if the patient is indigenous.

DIABETES: Indicates if the patient has diabetes.

COPD: Indicates if the patient has Chronic Obstructive Pulmonary Disease (COPD).

ASTHMA: Indicates if the patient has asthma.

IMMUNOSUPPRESSION: Indicates if the patient has immunosuppression.

HYPERTENSION: Indicates if the patient has hypertension (high blood pressure).

OTHER_COMORBIDITIES: Indicates if the patient has other comorbidities.

CARDIOVASCULAR: Indicates if the patient has cardiovascular issues.

OBESITY: Indicates if the patient is obese.

CHRONIC_RENAL_FAILURE: Indicates if the patient has chronic renal failure.

SMOKING: Indicates if the patient is a smoker.

ANOTHER_CASE: Indicates if the patient is related to another case.

LAB_SAMPLE: Indicates if the patient took a lab sample.

LAB_RESULT: The result of the lab sample.

ANTIGEN_SAMPLE: Indicates if the patient took an antigen sample.

ANTIGEN_RESULT: The result of the antigen sample.

FINAL_CLASSIFICATION: Final classification of the case (e.g., confirmed, suspected).

MIGRANT: Indicates if the patient is a migrant.

NATIONALITY_COUNTRY: Country of nationality for migrants.

ORIGIN_COUNTRY: Country of origin for migrants.

ICU: Indicates if the patient was admitted to the Intensive Care Unit.

In [23]:

```
# Step 1: Handling Missing Values
# Replace missing values with the mean for numerical columns
numerical_cols = ['AGE']
df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].mean())

# For categorical columns, fill missing values with a special category (e.g., 'UNKNOWN')
categorical_cols = ['SEX', 'PATIENT_TYPE', 'NATIONALITY', 'INTUBATED', 'PNEUMONIA', 'SMO
df[categorical_cols] = df[categorical_cols].fillna('UNKNOWN')

# Step 2: Data Encoding
# One-hot encode nominal categorical variables
nominal_categorical_cols = ['SEX', 'PATIENT_TYPE', 'NATIONALITY', 'INTUBATED', 'PNEUMONI
df = pd.get_dummies(df, columns=nominal_categorical_cols)

# Label encode ordinal categorical variables (if any)
# No ordinal categorical variables in the given columns.

# Step 3: Feature Scaling
# Scale numerical features using standardization
numerical_cols = ['AGE']
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

# Now, the 'df' dataset is preprocessed and ready for further steps like model selection
```

Data exploration and visualization are crucial steps that help you gain insights into your dataset and make informed decisions during the modeling process. Here's how these plots can help you:

Histograms and Distributions:

Histograms help you understand the distribution of numerical variables. By examining the shape of the distribution, you can identify potential outliers or skewed data that might need further preprocessing. For example, if the age distribution is heavily skewed, it might affect the model's performance, and you may need to transform the data. Bar Plots:

Bar plots show the distribution of categorical variables. By analyzing the frequencies of different categories, you can identify class imbalances or the prevalence of certain conditions. For example, if the dataset has an imbalanced class distribution (e.g., more Inpatients than Outpatients), it could impact model training, and you may need to consider techniques like oversampling or undersampling. Box Plots:

Box plots help you identify the spread and potential outliers in numerical variables. Outliers can be important to consider because extreme values may have a significant impact on the model's behavior. You can decide whether to remove outliers or handle them differently based on the specific context of your analysis. Correlation Heatmaps:

Correlation heatmaps provide insights into the relationships between numerical variables. Strong correlations between variables can indicate multicollinearity, where two or more features are highly correlated, which might affect model interpretability and performance. You can use this information to select a diverse set of features for modeling. Scatter Plots:

Scatter plots help visualize relationships between numerical variables. You can look for patterns, trends, or potential linear/non-linear relationships between features, which may guide you in selecting relevant predictors. Stacked Bar Plots or Heatmaps:

Stacked bar plots or heatmaps can show the relationship between a categorical variable and the target variable (COVID severity). This helps you understand if certain categories have a higher or lower likelihood of leading to severe COVID outcomes. For example, you might discover that certain comorbidities or age groups are associated with more severe outcomes. Choosing Features for Prediction: To predict the severity of COVID, you need to identify features that are likely to have a significant impact on the outcome. Here are some steps to guide your feature selection process:

Domain Knowledge: Leverage your medical expertise to identify relevant features known to be associated with COVID severity, such as age, comorbidities, vital signs, and test results.

Correlation: Examine the correlation heatmap to see which features are strongly correlated with the target variable (COVID severity). Highly correlated features are more likely to be good predictors.

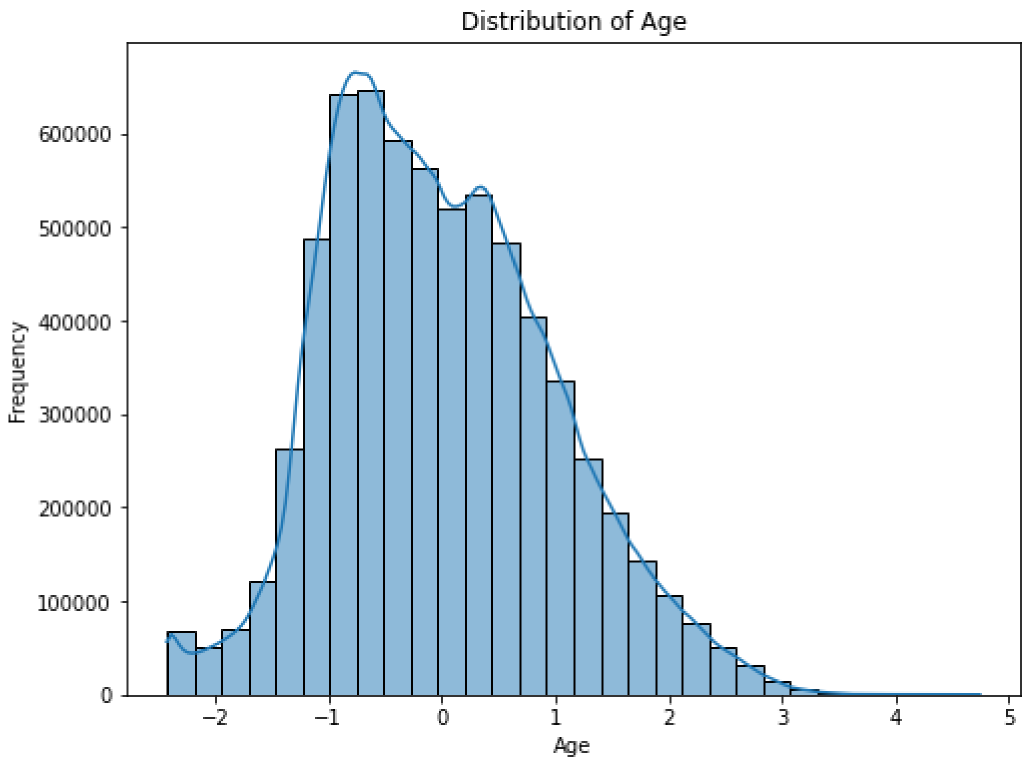
Feature Importance: If you plan to use tree-based models like Random Forest or Gradient Boosting, you can utilize their feature importance scores to rank the features based on their impact on prediction.

Recursive Feature Elimination (RFE): Use techniques like RFE to iteratively remove less relevant features from the dataset based on their impact on model performance.

Domain Experts: Collaborate with domain experts or medical professionals to understand the clinical relevance of certain features and how they might contribute to predicting COVID severity.

In [27]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Remove leading/trailing whitespaces from column names (to be safe)
df.columns = df.columns.str.strip()

# Example: Histogram of Age
plt.figure(figsize=(8, 6))
sns.histplot(df['AGE'], bins=30, kde=True)
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

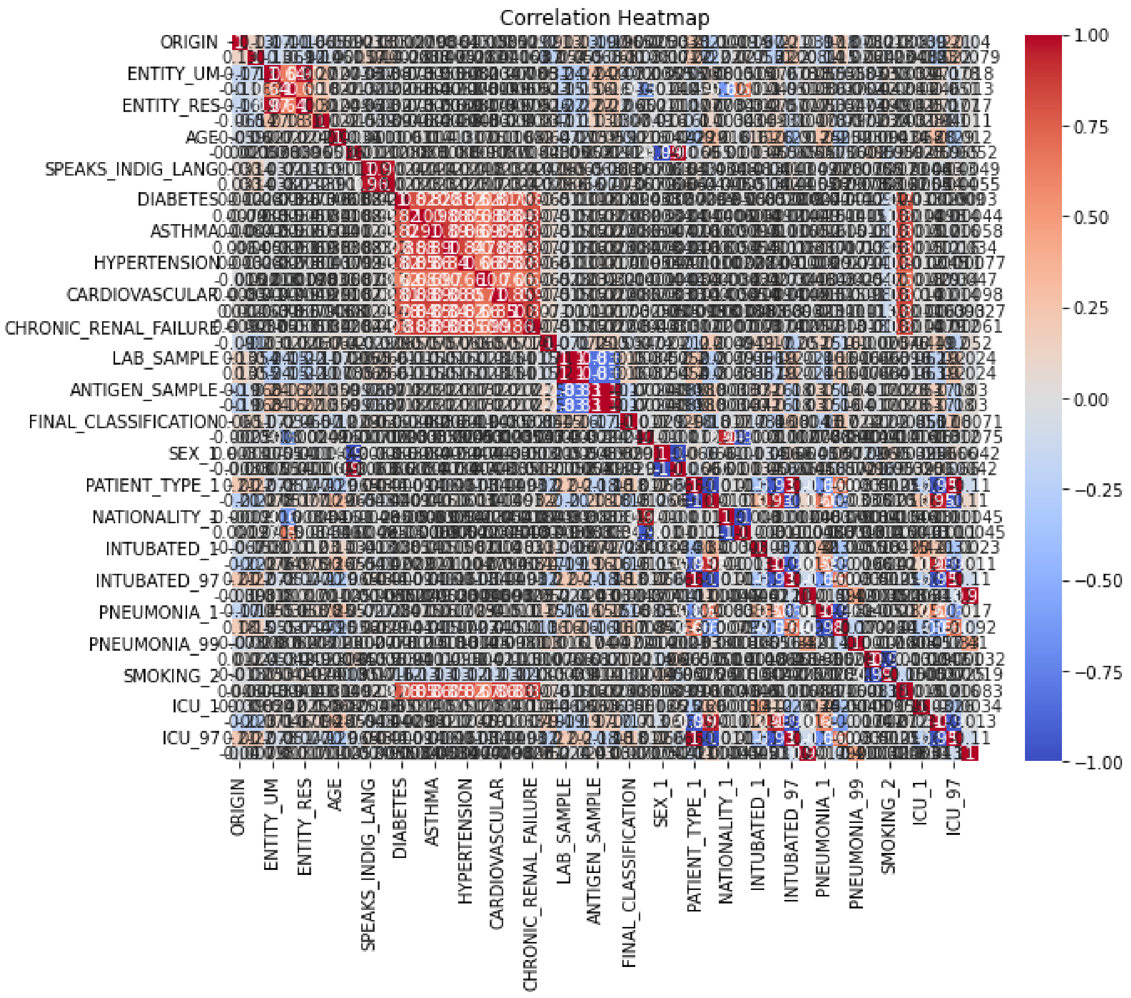


In [ ]:

```

```

In [34]:

```
# Example: Correlation heatmap
correlation_matrix = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



In [36]:

```
# Print all column names to check for any discrepancies
print(df.columns)
```

```
Index(['UPDATE_DATE', 'REGISTRATION_ID', 'ORIGIN', 'SECTOR', 'ENTITY_UM',
       'ENTITY_BORN', 'ENTITY_RES', 'MUNICIPALITY_RES', 'ADMISSION_DATE',
       'SYMPTOMS_DATE', 'DEATH_DATE', 'AGE', 'PREGNANCY', 'SPEAKS_INDIG_LA
NG',
       'INDIGENOUS', 'DIABETES', 'COPD', 'ASTHMA', 'IMMUNOSUPPRESSION',
       'HYPERTENSION', 'OTHER_COMORBIDITIES', 'CARDIOVASCULAR', 'OBESITY',
       'CHRONIC_RENAL_FAILURE', 'ANOTHER_CASE', 'LAB_SAMPLE', 'LAB_RESUL
T',
       'ANTIGEN_SAMPLE', 'ANTIGEN_RESULT', 'FINAL_CLASSIFICATION', 'MIGRAN
T',
       'NATIONALITY_COUNTRY', 'ORIGIN_COUNTRY', 'SEX_1', 'SEX_2',
       'PATIENT_TYPE_1', 'PATIENT_TYPE_2', 'NATIONALITY_1', 'NATIONALITY_
2',
       'INTUBATED_1', 'INTUBATED_2', 'INTUBATED_97', 'INTUBATED_99',
       'PNEUMONIA_1', 'PNEUMONIA_2', 'PNEUMONIA_99', 'SMOKING_1', 'SMOKING
_2',
       'SMOKING_98', 'ICU_1', 'ICU_2', 'ICU_97', 'ICU_99'],
      dtype='object')
```

In [39]:

```
# Selecting relevant features for predicting COVID severity
selected_features = ['AGE', 'DIABETES', 'HYPERTENSION', 'CARDIOVASCULAR', 'OBESITY', 'FI
                     'SEX_1', 'SEX_2', 'PATIENT_TYPE_1', 'PATIENT_TYPE_2', 'INTUBATED_1'
                     'INTUBATED_99', 'PNEUMONIA_1', 'PNEUMONIA_2', 'PNEUMONIA_99', 'SMOK
                     'ICU_1', 'ICU_2', 'ICU_97', 'ICU_99']

# Create a new DataFrame with only the selected features
df_selected = df[selected_features].copy()

# Verify the new DataFrame
print(df_selected.head())
```

```
        AGE  DIABETES  HYPERTENSION  CARDIOVASCULAR  OBESITY  \
0  0.004911         2             2               2        2
1  1.486635         1             1               2        1
2 -0.706316         2             2               2       98
3 -0.054358         2             2               2        2
4 -0.409971         2             2               2        2

   FINAL_CLASSIFICATION  SEX_1  SEX_2  PATIENT_TYPE_1  PATIENT_TYPE_2  ...
\
0                     1      0      1               1               0  ...
1                     2      1      0               0               1  ...
2                     6      0      1               1               0  ...
3                     7      1      0               1               0  ...
4                     6      0      1               1               0  ...

   PNEUMONIA_1  PNEUMONIA_2  PNEUMONIA_99  SMOKING_1  SMOKING_2  SMOKING_9
8  \
0            0            1             0          0          1
0
1            1            0             0          0          1
0
2            0            1             0          0          1
0
3            0            1             0          0          1
0
4            0            1             0          0          1
0

   ICU_1  ICU_2  ICU_97  ICU_99
0      0      0       1       0
1      1      0       0       0
2      0      0       1       0
3      0      0       1       0
4      0      0       1       0

[5 rows x 24 columns]
```

Now that we have the preprocessed data with the selected features, we can proceed to build a machine learning model to predict COVID severity based on the given dataset.

Before jumping into the model building, let's first split the data into training and testing sets. This allows us to train the model on one part of the data and evaluate its performance on another unseen part. We'll use 80% of the data for training and 20% for testing.

In [40]:

```
from sklearn.model_selection import train_test_split

# Separate features (X) and target (y)
X = df_selected.drop('FINAL_CLASSIFICATION', axis=1)
y = df_selected['FINAL_CLASSIFICATION']

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Verify the shapes of the splits
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (5327347, 23)
X_test shape: (1331837, 23)
y_train shape: (5327347,)
y_test shape: (1331837,)
```

Now, we can choose a machine learning algorithm suitable for classification tasks. Popular choices include Logistic Regression, Random Forest, Gradient Boosting, or Support Vector Machine (SVM). For simplicity, let's use the Random Forest Classifier as it can handle both numerical and categorical features well and often provides good results without much hyperparameter tuning.

In [41]:

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

# Initialize the Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the model on the training data
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Generate a confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.6161339563324941
Classification Report:
              precision    recall  f1-score   support

           1       0.14      0.00      0.00     28358
           2       0.05      0.00      0.00      1759
           3       0.61      0.19      0.28    439807
           4       0.00      0.00      0.00      2180
           5       0.01      0.00      0.00     14614
           6       0.66      0.03      0.05     69413
           7       0.62      0.95      0.75    775706

    accuracy                           0.62   1331837
   macro avg       0.30      0.17      0.16   1331837
weighted avg       0.60      0.62      0.53   1331837

Confusion Matrix:
[[    39      2   3244      0      2     20  25051]
 [     3      3   1581      0      2      8    162]
 [   121     32  81831      6     62    307 357448]
 [     0      0    231      0      0      3   1946]
 [     5      1   1927      0      2     39  12640]
 [    18      6   7989      1      8   1820  59571]
 [    87     17  38047      4     76    580 736895]]
```

To gain insights into how the features impact the model predictions, we can use SHAP (SHapley Additive exPlanations) values. SHAP values provide a way to explain the output of any machine learning model by attributing the prediction to each input feature. It helps to understand the contribution of each feature towards the final prediction.

In [42]:

```
pip install shap
```

```
Collecting shap
  Downloading shap-0.42.1-cp310-cp310-win_amd64.whl (462 kB)
     ------------------------------------- 462.3/462.3 kB 14.1 MB/s eta 0:
00:00
Requirement already satisfied: numpy in c:\users\imran\appdata\local\progr
ams\python\python310\lib\site-packages (from shap) (1.23.0)
Requirement already satisfied: scipy in c:\users\imran\appdata\local\progr
ams\python\python310\lib\site-packages (from shap) (1.8.1)
Requirement already satisfied: scikit-learn in c:\users\imran\appdata\loca
l\programs\python\python310\lib\site-packages (from shap) (1.1.1)
Requirement already satisfied: pandas in c:\users\imran\appdata\local\prog
rams\python\python310\lib\site-packages (from shap) (1.4.2)
Requirement already satisfied: tqdm>=4.27.0 in c:\users\imran\appdata\loca
l\programs\python\python310\lib\site-packages (from shap) (4.64.0)
Requirement already satisfied: packaging>20.9 in c:\users\imran\appdata\lo
cal\programs\python\python310\lib\site-packages (from shap) (21.3)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in c:\users\imran\appdata\local\progr
ams\python\python310\lib\site-packages (from shap) (0.56.4)
Collecting cloudpickle (from shap)
  Downloading cloudpickle-2.2.1-py3-none-any.whl (25 kB)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\imran
\appdata\local\programs\python\python310\lib\site-packages (from packaging
>20.9->shap) (3.0.9)
Requirement already satisfied: colorama in c:\users\imran\appdata\local\pr
ograms\python\python310\lib\site-packages (from tqdm>=4.27.0->shap) (0.4.
4)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in c:\users\imra
n\appdata\local\programs\python\python310\lib\site-packages (from numba->s
hap) (0.39.1)
Requirement already satisfied: setuptools in c:\users\imran\appdata\local
\programs\python\python310\lib\site-packages (from numba->shap) (63.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\imran\ap
pdata\local\programs\python\python310\lib\site-packages (from pandas->sha
p) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\imran\appdata\loca
l\programs\python\python310\lib\site-packages (from pandas->shap) (2022.1)
Requirement already satisfied: joblib>=1.0.0 in c:\users\imran\appdata\loc
al\programs\python\python310\lib\site-packages (from scikit-learn->shap)
(1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\imran\appd
ata\local\programs\python\python310\lib\site-packages (from scikit-learn->
shap) (3.1.0)
Requirement already satisfied: six>=1.5 in c:\users\imran\appdata\local\pr
ograms\python\python310\lib\site-packages (from python-dateutil>=2.8.1->pa
ndas->shap) (1.16.0)
Installing collected packages: slicer, cloudpickle, shap
Successfully installed cloudpickle-2.2.1 shap-0.42.1 slicer-0.0.7
Note: you may need to restart the kernel to use updated packages.
```

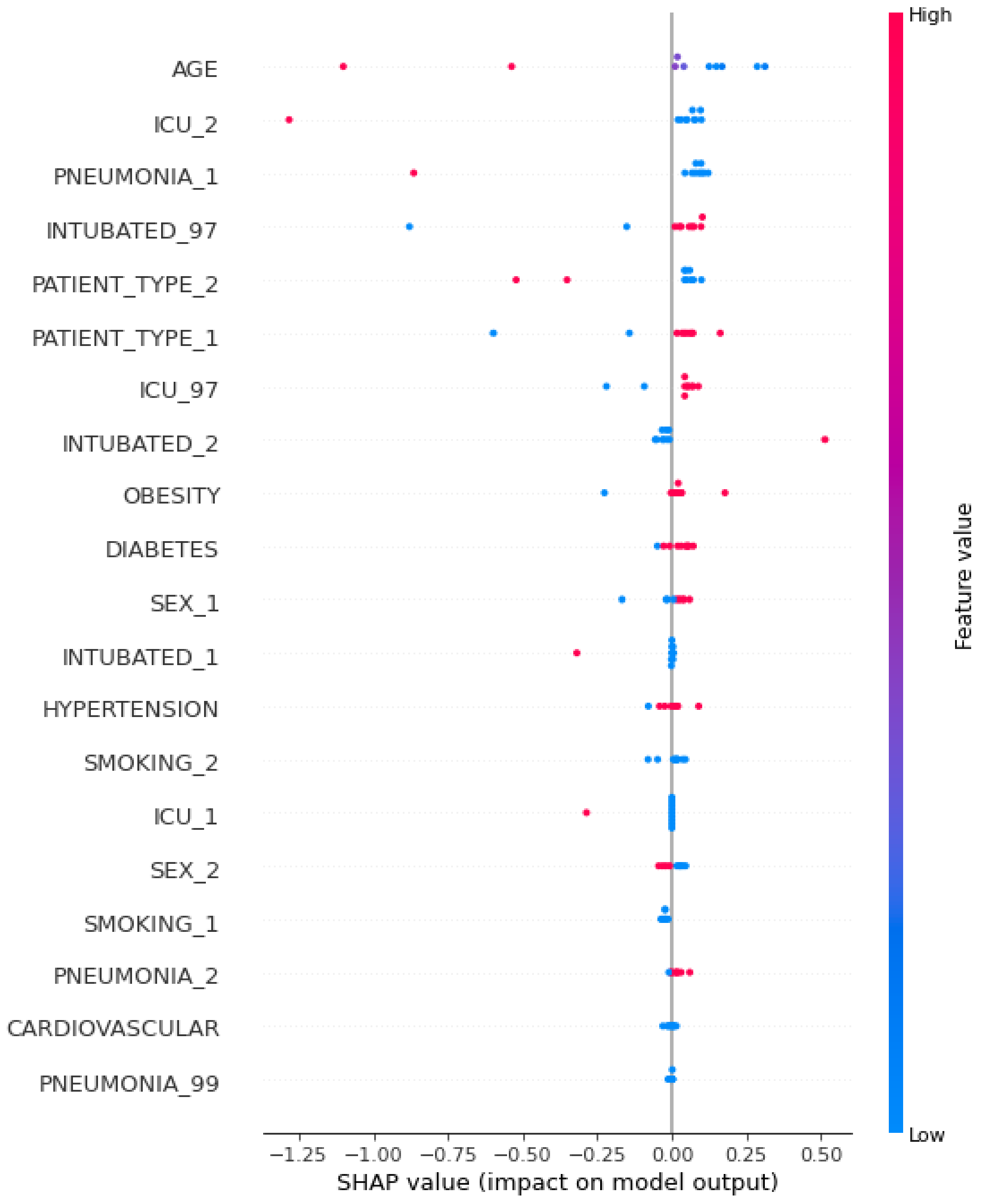
```
WARNING: Ignoring invalid distribution -treamlit (c:\users\imran\appdata\l
ocal\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\imran\appdata\l
ocal\programs\python\python310\lib\site-packages)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

The summary plot will show a horizontal bar chart for each feature, indicating how much that feature contributed to increasing or decreasing the prediction for the first ten samples. Positive SHAP values indicate an increase in the prediction, while negative SHAP values indicate a decrease.

In [43]:

```
import shap

# Initialize the SHAP explainer with the trained model's predictions
explainer = shap.Explainer(rf_classifier.predict, X_train)

# Calculate SHAP values for a set of samples from the test data
# We choose a small number (e.g., 10) for demonstration purposes, but you can increase i
shap_values = explainer(X_test.iloc[:10])

# Summary plot to visualize the impact of features on predictions for the first sample
shap.summary_plot(shap_values, X_test.iloc[:10])
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead t
o force console mode (e.g. in jupyter console)
```



This individual SHAP value plot will show how each feature contributed to the prediction for the first sample.

In [46]:

```
import shap

# Initialize the SHAP explainer with the trained model's predictions
explainer = shap.TreeExplainer(rf_classifier)

# Calculate SHAP values for the first sample from the test data
shap_values = explainer.shap_values(X_test.iloc[0])

# Individual SHAP value plot for the first sample
shap.initjs()
shap.force_plot(explainer.expected_value[1], shap_values[1], X_test.iloc[0])
```

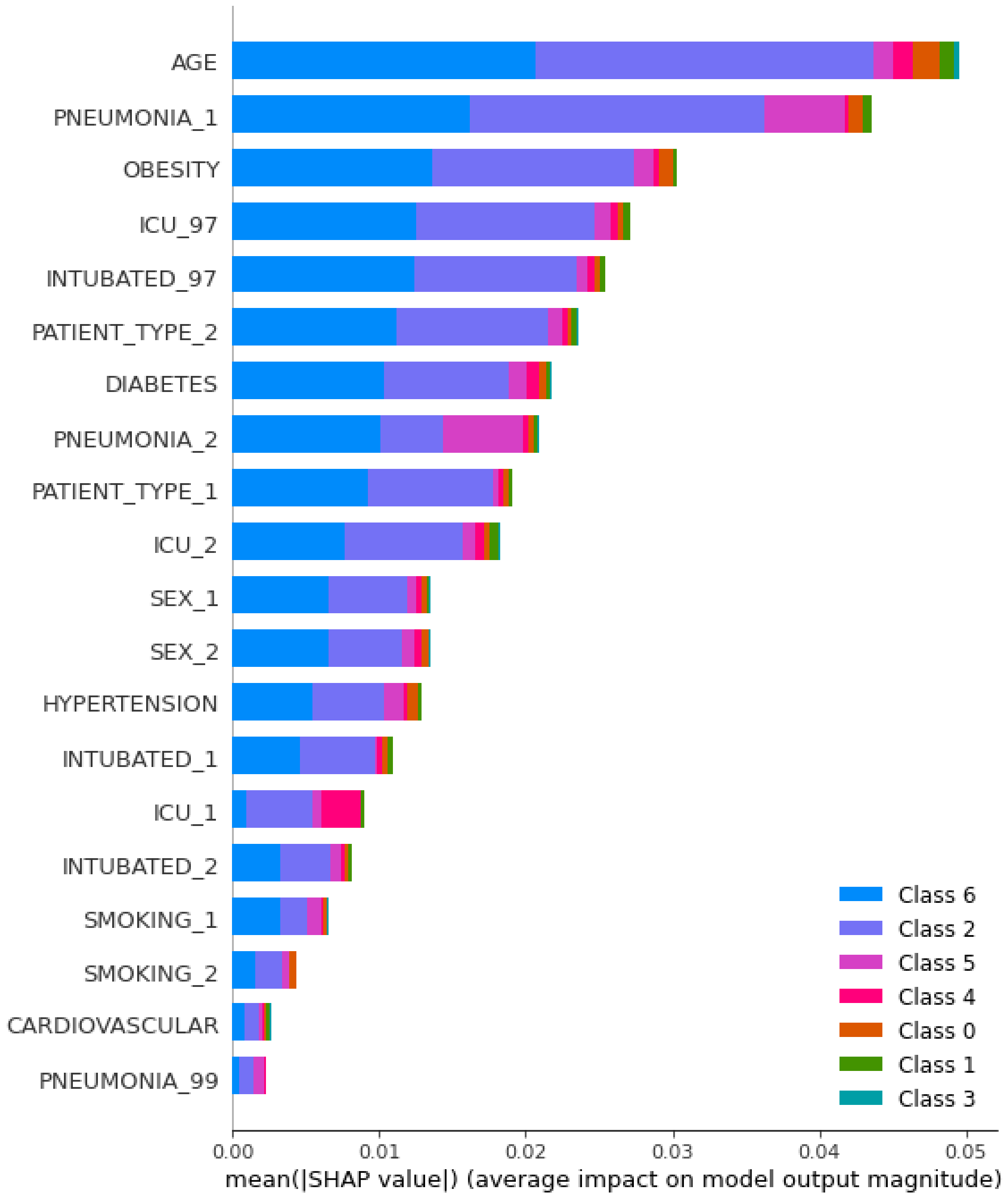
Out[46]:

Visualization omitted, Javascript library not loaded!

Have you run `initjs()` in this notebook? If this notebook was from another user you must also trust this notebook (File -> Trust notebook). If you are viewing this notebook on github the Javascript has been stripped for security. If you are using JupyterLab this error is because a JupyterLab extension has not yet been written.

In [47]:

```
import shap

# Initialize the SHAP explainer with the trained model's predictions
explainer = shap.TreeExplainer(rf_classifier)

# Calculate SHAP values for a set of samples from the test data
# We choose a small number (e.g., 10) for demonstration purposes, but you can increase i
shap_values = explainer.shap_values(X_test.iloc[:10])

# Summary plot to visualize the impact of features on predictions for the first sample
shap.summary_plot(shap_values, X_test.iloc[:10], plot_type='bar')
```



In the bar graph generated by the shap.summary_plot function, each bar represents the impact of a specific feature on the model's prediction for a particular sample (in this case, the first sample from the test data). The graph shows the magnitude and direction of the effect of each feature on the model's output.

The features are listed on the y-axis of the bar graph, and the x-axis represents the SHAP (SHapley Additive exPlanations) values. SHAP values provide a unified measure of feature importance in model predictions and are based on cooperative game theory.

The height of each bar indicates the absolute impact of the corresponding feature on the model's prediction for the given sample. Positive SHAP values (bars pointing to the right) indicate that the feature's contribution pushed the prediction towards a higher severity class, while negative SHAP values (bars pointing to the left) suggest that the feature pushed the prediction towards a lower severity class.

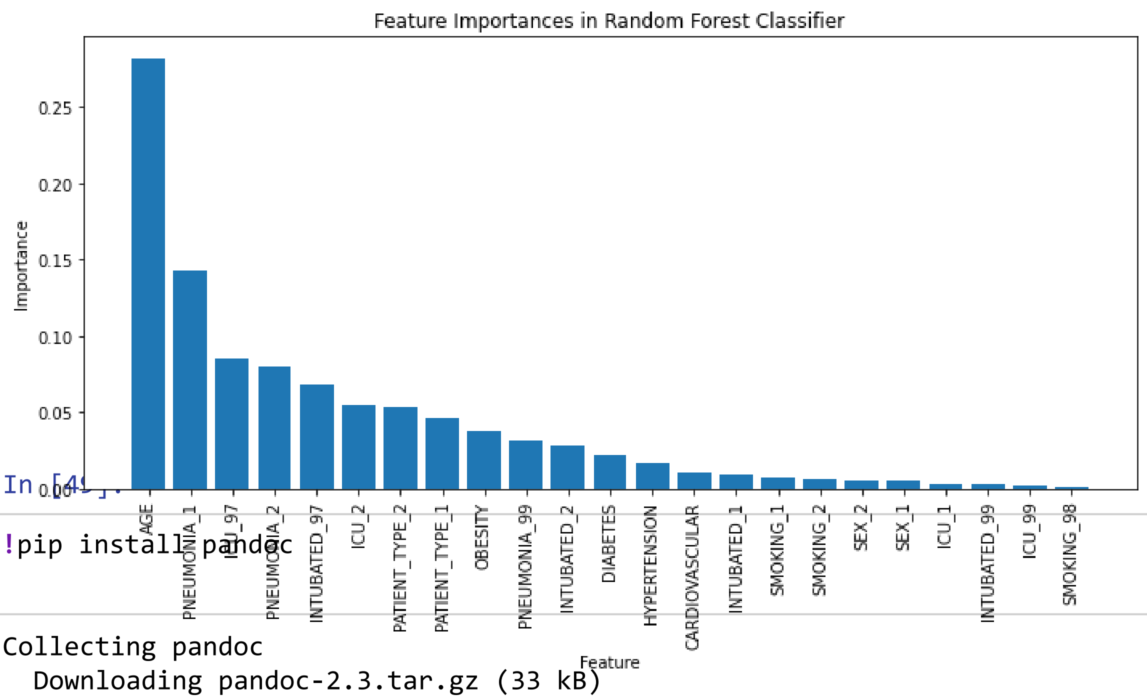
The color of the bar is used to distinguish between positive and negative values. For example, blue bars represent negative SHAP values, and red bars represent positive SHAP values.

By interpreting the bar graph, you can gain insights into how each feature influenced the model's decision for the specific sample and understand the factors contributing to the predicted severity of COVID.

In [48]:

```
# Get the feature importances from the trained model
feature_importances = rf_classifier.feature_importances_

# Create a DataFrame to store the feature importances along with their corresponding fea
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': feature_

# Sort the DataFrame in descending order of feature importances
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=Fal

# Print the sorted feature importances
print("Feature Importances:")
print(feature_importance_df)

# Plot the feature importances (optional, requires matplotlib)
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xticks(rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.title('Feature Importances in Random Forest Classifier')
plt.tight_layout()
plt.show()
```

```
Feature Importances:
           Feature  Importance
0              AGE    0.282220
13     PNEUMONIA_1    0.143158
21          ICU_97    0.084862
14     PNEUMONIA_2    0.079687
11    INTUBATED_97    0.068657
20           ICU_2    0.054328
8   PATIENT_TYPE_2    0.053732
7   PATIENT_TYPE_1    0.045924
4          OBESITY    0.038220
15    PNEUMONIA_99    0.030858
10     INTUBATED_2    0.028065
1         DIABETES    0.022211
2     HYPERTENSION    0.017067
3   CARDIOVASCULAR    0.009965
9      INTUBATED_1    0.008872
16       SMOKING_1    0.007490
17       SMOKING_2    0.006546
6            SEX_2    0.004962
5            SEX_1    0.004943
19           ICU_1    0.003172
12    INTUBATED_99    0.002520
22          ICU_99    0.001676
18      SMOKING_98    0.000867
```



In [49]:

```
!pip install pandoc
```

```
Collecting pandoc
  Downloading pandoc-2.3.tar.gz (33 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting plumbum (from pandoc)
  Downloading plumbum-1.8.2-py3-none-any.whl (127 kB)
     ------------------------------------ 127.0/127.0 kB 934.5 kB/s eta 0:
00:00
Collecting ply (from pandoc)
  Downloading ply-3.11-py2.py3-none-any.whl (49 kB)
     ---------------------------------------- 49.6/49.6 kB 1.3 MB/s eta 0:
00:00
Requirement already satisfied: pywin32 in c:\users\imran\appdata\local\pro
grams\python\python310\lib\site-packages (from plumbum->pandoc) (304)
Building wheels for collected packages: pandoc
  Building wheel for pandoc (setup.py): started
  Building wheel for pandoc (setup.py): finished with status 'done'
  Created wheel for pandoc: filename=pandoc-2.3-py3-none-any.whl size=3326
3 sha256=b8e3f8a4f6e32481895c34d972bf0610f6e72c04598cea4d576b57cb944e37a9
  Stored in directory: c:\users\imran\appdata\local\pip\cache\wheels\76\27
\c2\c26175310aadcb8741b77657a1bb49c50cc7d4cdbf9eee0005
Successfully built pandoc
Installing collected packages: ply, plumbum, pandoc
Successfully installed pandoc-2.3 plumbum-1.8.2 ply-3.11

WARNING: Ignoring invalid distribution -treamlit (c:\users\imran\appdata\l
ocal\programs\python\python310\lib\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\imran\appdata\l
ocal\programs\python\python310\lib\site-packages)

[notice] A new release of pip is available: 23.1.2 -> 23.2.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [ ]:

```
```