Covid, Allergy, Flu & Cold Prediction based on Symptoms

Name: Soh Hong Yu

Admin Number: P2100775

Class: DAAA/FT/2B/01

Module Code: ST1505 DevOps and AI Automation

References (In Harvard format):

1. Conway, W. (2021) Covid, flu, cold symptoms, Kaggle.
   Available at: https://www.kaggle.com/datasets/walterconway/covid-flu-cold-symptoms(Accessed: November 13, 2022).
2. Gov.sg (2020) I am showing respiratory symptoms (like a cough, runny nose or fever), where should I go?, MCI - Gov.SG.
   Available at: https://www.gov.sg/article/i-am-showing-respiratory-symptoms-where-should-i-go(Accessed: November 14, 2022).

Project Objective

To develop a Machine Learning (ML) web application using GitLab and Flask Framework, incorporating the 3 DevOps best practices mentioned in the Background section. This is an open-ended assessment.

For this project, we will be using the Covid, Flu, Allergy and Cold symptoms and making a machine learning model to do classification to help predict what is the user is currently experiencing.

Background Information

During the Covid 19 pandemic many Singaporeans despite having a small symptoms like cough and sneezing, they will go to their nearby hospitals and general clinics to check if they have been infected by Covid-19. This not only increase the likelihood of other Singaporeans getting affected as they travel along the common corridors in Singapore, but also increase the workload of the hospitals and general clinics. This prompted the Singaporean Government to launch campaigns to promote and educate citizens on what they can do when they experience symptoms similar to Covid-19.

Initialising Libraries and Variables

In [ ]:
```python
# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from pandas_profiling import ProfileReport
import seaborn as sns

# SKLEARN Libraries
# Preprocessing
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer, StandardScaler, OrdinalEncoder
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer, SimpleImputer

# Model Selection
from sklearn.model_selection import (
    train_test_split,
    cross_validate,
    cross_val_score,
    GridSearchCV,
    RandomizedSearchCV,
    cross_val_predict,
    learning_curve,
    StratifiedKFold,
)
from sklearn.metrics import classification_report, confusion_matrix, RocCurveDisplay
from sklearn.tree import export_graphviz
from sklearn.tree import plot_tree

# Models
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import Perceptron
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import RidgeClassifier
from sklearn.linear_model import RidgeClassifierCV
from sklearn.linear_model import SGDClassifier

# IMBLEARN
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler

# Pickle Library = Saving Models
import pickle
```

Loading Dataset

In [ ]:
```python
df = pd.read_csv("COVID, FLU, COLD Symptoms.csv", sep=",")
df.head()
```

Out[ ]:

	COUGH	MUSCLE_ACHES	TIREDNESS	SORE_THROAT	RUNNY_NOSE	STUFFY_NOSE	FEVER	NAUSEA	VOM
0	0	0	1	0	1	0	0	0	
1	0	0	1	0	0	0	0	0	
2	0	1	1	1	0	0	0	0	
3	0	0	0	1	1	0	0	0	
4	0	0	1	0	1	0	0	0	

5 rows × 21 columns

Exploratory Data Analysis

We will begin by conducting an exploratory data analysis of the data, to gain a better understanding of the characteristics of the dataset.

This data set is based on the mayo clinic site: https://www.mayoclinic.org/diseases-conditions/coronavirus/in-depth/covid-19-cold-flu-and-allergies-differences/art-20503981.

In order to tell the difference between the flu, cold, allergy, and covid this data was made.

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

Out[ ]:
```
(44453, 21)
```

This dataset contains 44453 data points with 21 columns.

Metadata:

COUGH : Boolean value that indicates if coughing is a symptom. 1 means coughing and 0 means normal.
MUSCLE_ACHES : Boolean value that indicates if muscle aches is a symptom. 1 means muscle aches and 0 means normal.
TIREDNESS : Boolean value that indicates if tiring is a symptom. 1 means tiring and 0 means normal.
SORE_THROAT : Boolean value that indicates if sore throat is a symptom. 1 means sore throat and 0 means normal.
RUNNY_NOSE : Boolean value that indicates if running nose is a symptom. 1 means running nose and 0 means normal.
STUFFY_NOSE : Boolean value that indicates if stuffy nose is a symptom. 1 means stuffy nose and 0 means normal.
FEVER : Boolean value that indicates if fever is a symptom. 1 means fever and 0 means normal.
NAUSEA : Boolean value that indicates if nausea is a symptom. 1 means nausea and 0 means normal.
VOMITING : Boolean value that indicates if vomiting is a symptom. 1 means vomiting and 0 means normal.
DIARRHEA : Boolean value that indicates if diarrhea is a symptom. 1 means diarrhea and 0 means normal.
SHORTNESS_OF_BREATH : Boolean value that indicates if shortness of breath is a symptom. 1 means shortness of breath and 0 means normal.
DIFFICULTY_BREATHING : Boolean value that indicates if difficulty breathing is a symptom. 1 means

difficulty breathing and 0 means normal.
LOSS_OF_TASTE : Boolean value that indicates if loss of taste is a symptom. 1 means loss of taste and 0 means normal.
LOSS_OF_SMELL : Boolean value that indicates if loss of smell is a symptom. 1 means loss of smell and 0 means normal.
ITCHY_NOSE : Boolean value that indicates if itchy nose is a symptom. 1 means itchy nose and 0 means normal.
ITCHY_EYES : Boolean value that indicates if itchy eyes is a symptom. 1 means itchy eyes and 0 means normal.
ITCHY_MOUTH : Boolean value that indicates if itchy mouth is a symptom. 1 means itchy mouth and 0 means normal.
ITCHY_INNER_EAR : Boolean value that indicates if itchy inner ear is a symptom. 1 means itchy inner ear and 0 means normal.
SNEEZING : Boolean value that indicates if sneezing is a symptom. 1 means sneezing and 0 means normal.
PINK_EYE : Boolean value that indicates if pink eye is a symptom. 1 means there is pink eye and 0 means normal.
TYPE : Type of illness [flu, cold, allergy & covid].

Data Exploration

To prevent mutation of our original data, we will make a copy of our data to perform eda on it.

In [ ]:
```
df_eda = df.copy()
```

Descriptive Statistics

In [ ]:
```
df_eda.shape
```

Out[ ]:
```
(44453, 21)
```

There are 44453 rows and 21 columns in the entire dataset.

In [ ]:
```
df_eda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44453 entries, 0 to 44452
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   COUGH                 44453 non-null  int64 
 1   MUSCLE_ACHES          44453 non-null  int64 
 2   TIREDNESS             44453 non-null  int64 
 3   SORE_THROAT           44453 non-null  int64 
 4   RUNNY_NOSE            44453 non-null  int64 
 5   STUFFY_NOSE           44453 non-null  int64 
 6   FEVER                 44453 non-null  int64 
 7   NAUSEA                44453 non-null  int64 
 8   VOMITING              44453 non-null  int64 
 9   DIARRHEA              44453 non-null  int64 
 10  SHORTNESS_OF_BREATH   44453 non-null  int64 
 11  DIFFICULTY_BREATHING  44453 non-null  int64 
 12  LOSS_OF_TASTE         44453 non-null  int64 
 13  LOSS_OF_SMELL         44453 non-null  int64 
 14  ITCHY_NOSE            44453 non-null  int64 
 15  ITCHY_EYES            44453 non-null  int64 
 16  ITCHY_MOUTH           44453 non-null  int64 
 17  ITCHY_INNER_EAR       44453 non-null  int64 
 18  SNEEZING              44453 non-null  int64 
 19  PINK_EYE              44453 non-null  int64 
 20  TYPE                  44453 non-null  object
dtypes: int64(20), object(1)
memory usage: 7.1+ MB
```

Observations

- There is 44453 rows and 21 columns in the dataset.
- There is no null values in the dataset.
- Majority of the dataset is int64. The only thing is TYPE is a string.

Data Information

In [ ]:
```
descriptive_stats = df_eda.describe(include="all").T
descriptive_stats["Proportion of Most Frequent Value"] = (
    descriptive_stats["freq"] / len(df_eda) * 100
)
descriptive_stats.sort_values(
    "Proportion of Most Frequent Value", ascending=False)
```

Out[ ]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max	Pro F
TYPE	44453	4	FLU	25000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	56
COUGH	44453.0	NaN	NaN	NaN	0.520662	0.499579	0.0	0.0	1.0	1.0	1.0	
MUSCLE_ACHES	44453.0	NaN	NaN	NaN	0.519762	0.499615	0.0	0.0	1.0	1.0	1.0	
TIREDNESS	44453.0	NaN	NaN	NaN	0.519897	0.49961	0.0	0.0	1.0	1.0	1.0	
SORE_THROAT	44453.0	NaN	NaN	NaN	0.519358	0.499631	0.0	0.0	1.0	1.0	1.0	
RUNNY_NOSE	44453.0	NaN	NaN	NaN	0.496232	0.499991	0.0	0.0	0.0	1.0	1.0	
STUFFY_NOSE	44453.0	NaN	NaN	NaN	0.49585	0.499988	0.0	0.0	0.0	1.0	1.0	
FEVER	44453.0	NaN	NaN	NaN	0.3254	0.468529	0.0	0.0	0.0	1.0	1.0	
NAUSEA	44453.0	NaN	NaN	NaN	0.324253	0.4681	0.0	0.0	0.0	1.0	1.0	
VOMITING	44453.0	NaN	NaN	NaN	0.324523	0.468201	0.0	0.0	0.0	1.0	1.0	
DIARRHEA	44453.0	NaN	NaN	NaN	0.323465	0.467804	0.0	0.0	0.0	1.0	1.0	
SHORTNESS_OF_BREATH	44453.0	NaN	NaN	NaN	0.323893	0.467965	0.0	0.0	0.0	1.0	1.0	
DIFFICULTY_BREATHING	44453.0	NaN	NaN	NaN	0.324455	0.468176	0.0	0.0	0.0	1.0	1.0	
LOSS_OF_TASTE	44453.0	NaN	NaN	NaN	0.422986	0.494039	0.0	0.0	0.0	1.0	1.0	
LOSS_OF_SMELL	44453.0	NaN	NaN	NaN	0.423234	0.494077	0.0	0.0	0.0	1.0	1.0	
ITCHY_NOSE	44453.0	NaN	NaN	NaN	0.184285	0.38772	0.0	0.0	0.0	0.0	1.0	
ITCHY_EYES	44453.0	NaN	NaN	NaN	0.184285	0.38772	0.0	0.0	0.0	0.0	1.0	
ITCHY_MOUTH	44453.0	NaN	NaN	NaN	0.184285	0.38772	0.0	0.0	0.0	0.0	1.0	
ITCHY_INNER_EAR	44453.0	NaN	NaN	NaN	0.184285	0.38772	0.0	0.0	0.0	0.0	1.0	
SNEEZING	44453.0	NaN	NaN	NaN	0.519313	0.499633	0.0	0.0	1.0	1.0	1.0	
PINK_EYE	44453.0	NaN	NaN	NaN	0.18424	0.387684	0.0	0.0	0.0	0.0	1.0	

<u>Observations</u>

- From looking at the most frequent value of each column, we note that there are 4 different type of labels and because of that it does not appear to be balanced.

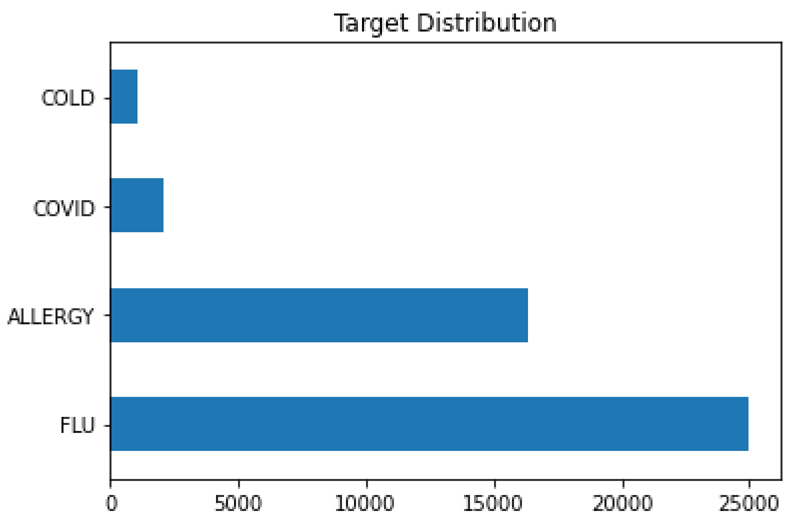
Target Variable - TYPE

Our Target Label is the TYPE column in the dataset. It is a column that have 4 different value:

1. Covid
2. Allergy
3. Cold
4. Flu

In [ ]:

```
df_eda["TYPE"].value_counts().plot(kind="barh", title="Target Distribution")
plt.show()
```



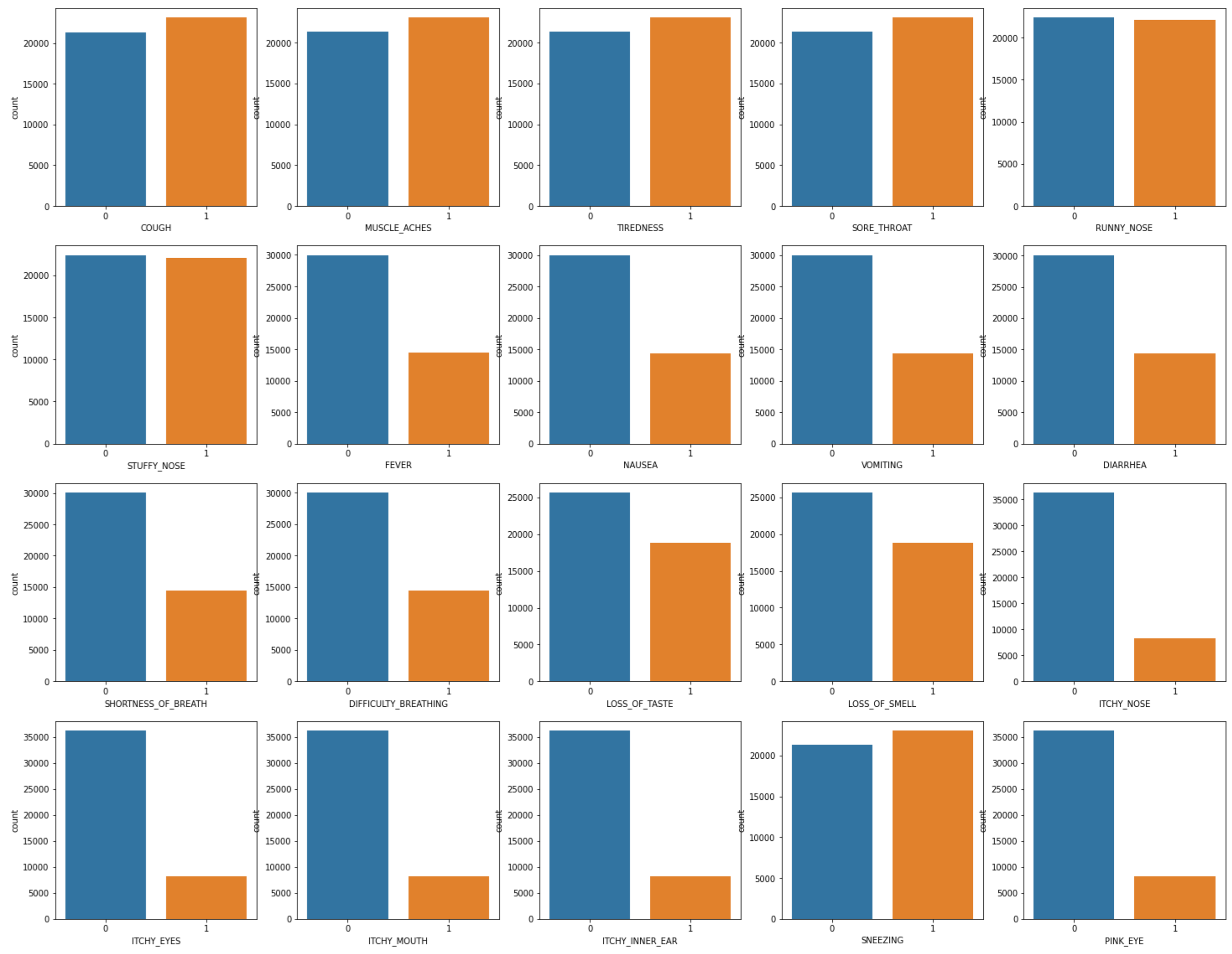
Observations

As we can see, the Target variable of TYPE is imbalanced, with almost most of the data is allergy and flu. This is a concern that we will need to solve later on.

Univariate Analysis

We will begin with a univariate analysis, analysing the distribution of each variable.

In [ ]:
```
fig, ax = plt.subplots(4, 5, figsize=(25, 20))

for idx, subplot in enumerate(ax.ravel()):
    sns.countplot(x=df.columns[idx], data=df, ax=subplot)
plt.show()
```



Observations

From the graphs we can see most of the data are balanced. There are some data that is not balanced and maybe a key symptoms[factor] why different type of illness happens. We will continue to explore and see if the hypothesis of the symptoms being a key factor.

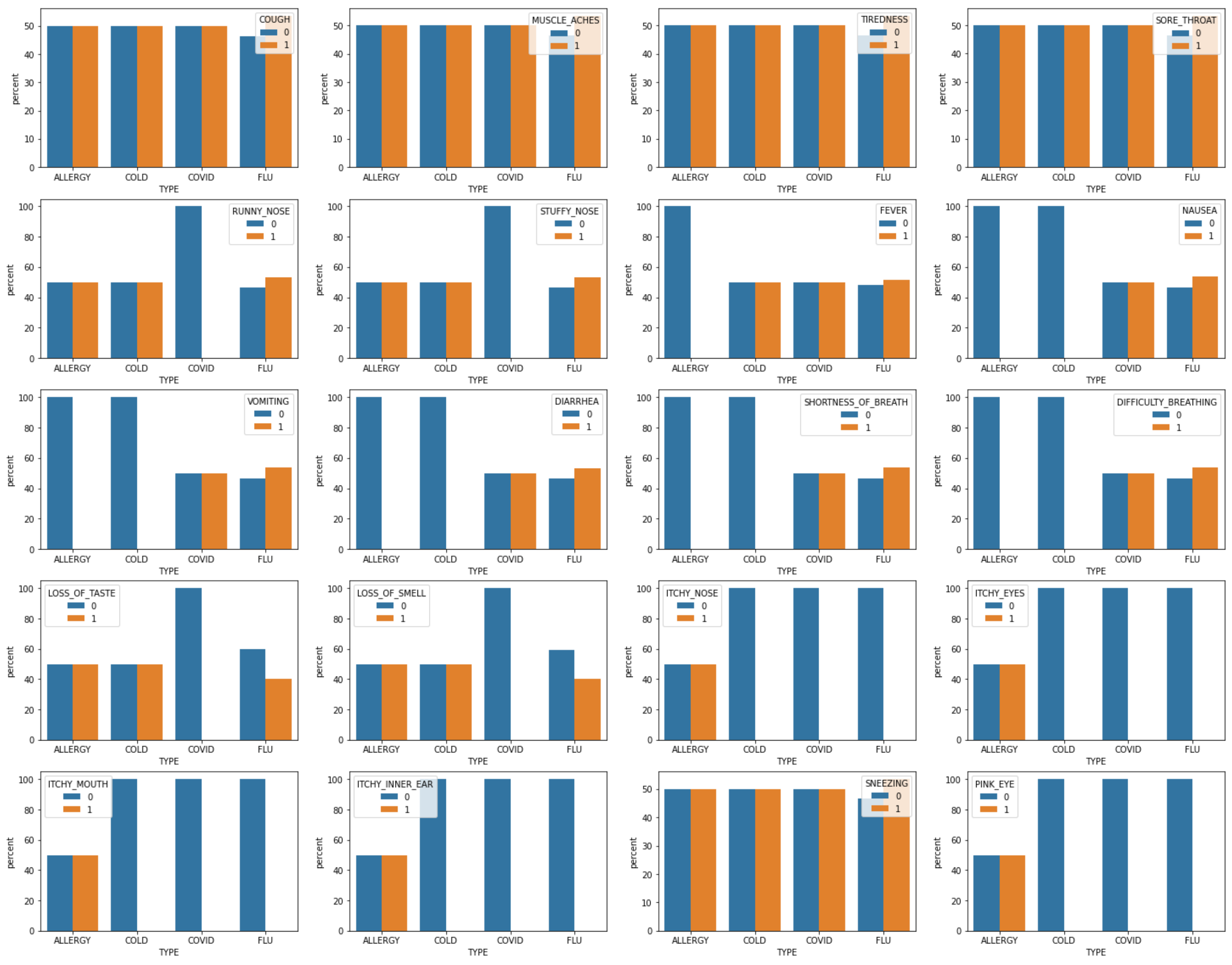
Bivariate Analysis

We will analysis the relationships between the different variables and most importantly the TYPE

Percentage of symptoms based on TYPE

In [ ]:

```
fig, ax = plt.subplots(5, 4, figsize=(25, 20))

for idx, subplot in enumerate(ax.ravel()):
    (df
     .groupby("TYPE")[df.columns[idx]]
     .value_counts(normalize=True)
     .mul(100)
     .rename('percent')
     .reset_index()
     .pipe((sns.barplot, 'data'), x="TYPE", y='percent', hue=df.columns[idx], ax=subplot))

plt.show()
```



Observations

From the different graphs, we can see that certain symptoms are not affected by the TYPE. For example the PINK_EYE is a key symptoms of allergy [though only 50%], but it is not a possible symptom for both COLD, COVID and FLU. This suggest that a tree-based model will be strong and predicting what type of illness it is.

Data Preparation

Now we move on to data preparation to prepare for model training.

Separate Target Label and Features

We will first separate our target label from the features. It is also necessary to perform a label encoding on the target label to convert it to a number.

```
In [ ]: x, y = df.drop("TYPE", axis=1), df["TYPE"]
```

Splitting Data points

To evaluate our final chosen models, we will leave a small independent test set to report on the final performance of our classifiers. When building the hold out set, we use stratify to ensure that the distribution of classes is the same in both the independent set and the training set.

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, stratify=y, shuffle=True, random_state=42
)
x_training = x_train.copy()
```

In [ ]:
```
x_training.shape
```

Out[ ]:
```
(35562, 20)
```

In [ ]:
```
x_test.shape
```

Out[ ]:
```
(8891, 20)
```

Data Preprocessing

As we have done the EDA of the data, we learn that we need to do some encoding and balancing.

Categorical Encoding

As the our dataset still contains categorical data which is hard for some of sklearn's model to train on, we have to encode the data using one of the encoders.

1. pd.get_dummies()
2. One Hot Encoder
3. Label Encoder
4. Ordinal Encoder

As the categorical data that we are going to encode is the label, we will be using the label encoder to encode the TYPE.

In [ ]:
```
from sklearn.preprocessing import FunctionTransformer, StandardScaler, LabelEncoder
```

In [ ]:
```
encoder = LabelEncoder()
y_train_enc = encoder.fit_transform(y_train)
y_train_enc
```

Out[ ]:
```
array([3, 0, 3, ..., 0, 0, 1])
```

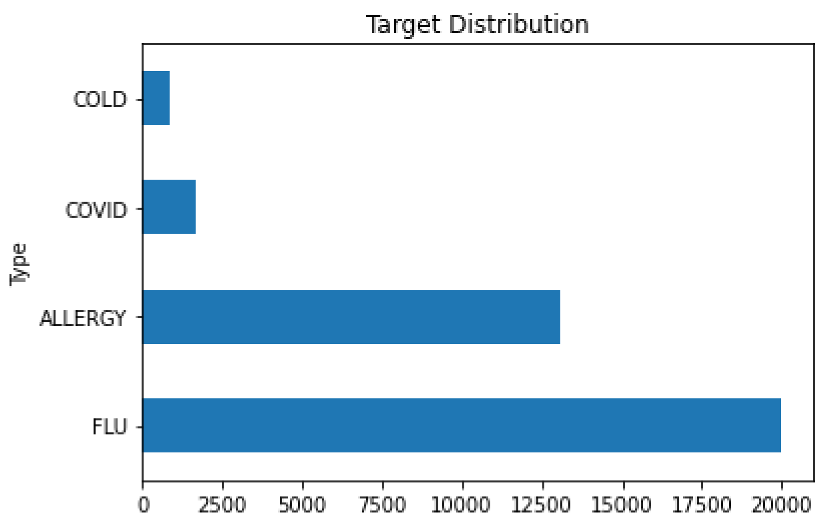
In [ ]:
```
encoder.classes_
```

Out[ ]:
```
array(['ALLERGY', 'COLD', 'COVID', 'FLU'], dtype=object)
```

Handling Imbalanced data through RESAMPLING

As mentioned before, the distribution of TYPE is imbalanced, we can visualise this through plotting of the y_train values

In [ ]:
```
ax = y_train.value_counts().plot(
    kind="barh",
    title="Target Distribution",
    xlabel="Type",
    ylabel="Number of Illness",
)
```



To solve this imbalanced, we can use the following solutions. [Using imbalanced learn]

1. Random Resampling
2. SMOTE (Synthetic Minority Oversampling Technique)

We will be using Random Resampling for this example. The reason is SMOTE is able to synthesize data which will affect all the columns like PINK_EYE which is encoded by the original dataset.

In [ ]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [ ]:
```
resample = RandomOverSampler(random_state=11)

X_resampled_train, y_resampled_train = resample.fit_resample(
    x_training, y_train_enc)
X_resampled_train
```

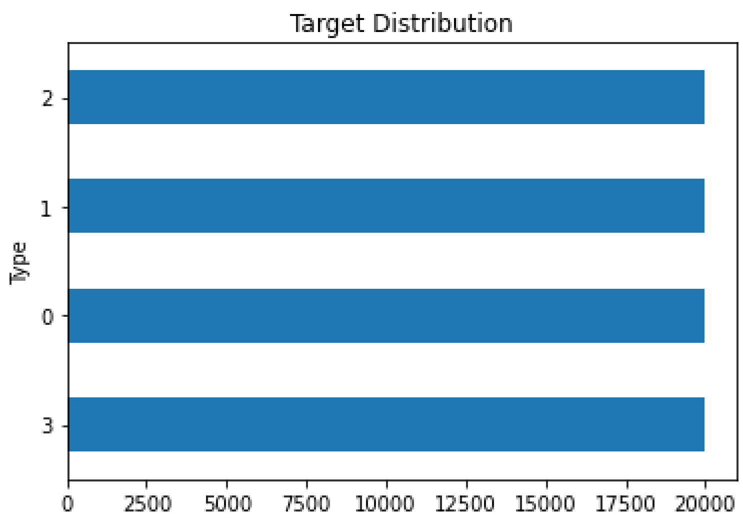
Out[ ]:

	COUGH	MUSCLE_ACHES	TIREDNESS	SORE_THROAT	RUNNY_NOSE	STUFFY_NOSE	FEVER	NAUSEA
0	1	1	0	0	0	1	1	1
1	1	0	0	1	0	1	0	0
2	1	1	0	0	0	1	0	0
3	1	0	1	0	0	0	0	0
4	0	0	1	0	0	1	1	0
...	...	...	...	...	...	...	...	...
79995	1	1	0	0	0	0	0	1
79996	0	0	1	0	0	0	1	0
79997	0	0	0	0	0	0	0	1
79998	1	1	1	1	0	0	0	0
79999	0	0	1	0	0	0	1	1

80000 rows × 20 columns

Let's see what the distribution of the labels look like after resampling

In [ ]:
```
ax = pd.Series(y_resampled_train).value_counts().plot(
    kind="barh",
    title="Target Distribution",
    xlabel="Type",
    ylabel="Number of Illness",
)
```



Model Selection

My plan for model selection is to run the data through some models and deem the best model based on metrics and explainability.

Scoring Methods

We will be using the following metrics to evaluate the model

Accuracy

- We use accuracy as a scoring metric as we have balanced the data and we want to look and what is the % of correctly predicted illness type

Recall

- We use recall as higher the recall, the lower the FN which is important in this context

f1 score

- Harmonic mean of recall and recall, it does not care about how many true negatives are being classified

ROC_AUC

- ROC_AUC also known as Area Under the Receiver Operating Characteristic Curve allow us to range of possibilities for observation(probability) in our classification

```
In [ ]: # preset scoring options
        scoring_methods = ["accuracy", "recall_weighted",
                           "f1_weighted", 'roc_auc_ovr_weighted']
```

Baseline model

A baseline model is simple and interpretable, easy to infer upon. I am going to use DummyClassifier as a baseline to benchmark against my system later on.

stratified: generates predictions by respecting the training set's class distribution.

```
In [ ]: dummy = DummyClassifier(strategy="uniform")
        dummy.fit(X_resampled_train, y_resampled_train)
        print(
            f"Baseline Accuracy Score :{dummy.score(X_resampled_train, y_resampled_train)}")
        scores = cross_validate(
            dummy,
            X_resampled_train, y_resampled_train,
            cv=10,
            scoring=scoring_methods,
            n_jobs=-1,
            return_train_score=True,
        )
        # displaying scores
        display(
            pd.DataFrame(scores)
            .append(pd.Series(pd.DataFrame(scores).mean(), name="Mean"))
            .style.apply(
                lambda x: [
                    "background-color: red; color: white" if v else "" for v in x == x.min()
                ]
            )
            .apply(
                lambda x: [
                    "background-color: green; color: white" if v else "" for v in x == x.max()
                ]
            )
        )
```

```
Baseline Accuracy Score :0.2520125
<ipython-input-24-9c1282733716>:15: FutureWarning: The frame.append method is deprecated and
will be removed from pandas in a future version. Use pandas.concat instead.
  pd.DataFrame(scores)
```

	fit_time	score_time	test_accuracy	train_accuracy	test_recall_weighted	train_recall_weighted	test_f1_w
0	0.042999	0.033001	0.252625	0.249736	0.252625	0.249736	(
1	0.046000	0.039999	0.248875	0.252389	0.248875	0.252389	(
2	0.045005	0.034999	0.257875	0.249167	0.257875	0.249167	(
3	0.043999	0.036000	0.255750	0.247472	0.255750	0.247472	(
4	0.041001	0.026999	0.247875	0.248319	0.247875	0.248319	(
5	0.046000	0.028000	0.252625	0.252264	0.252625	0.252264	(
6	0.041997	0.029004	0.252500	0.246569	0.252500	0.246569	(
7	0.043002	0.027000	0.245750	0.249403	0.245750	0.249403	(
8	0.046995	0.027000	0.248250	0.248972	0.248250	0.248972	(
9	0.032005	0.021020	0.249625	0.250722	0.249625	0.250722	(
Mean	0.042900	0.030302	0.251175	0.249501	0.251175	0.249501	(

<u>Observations</u>

- The ROC AUC score is 0.25 this means that the model is randomly selecting and predicting the points [4 different type of data]

- The model is severely under-fitted as the accuracy from the training set is very low

Checking through the different models

We import some of the important models from the sklearn model and check which model is the best using their default parameters.

We will be using accuracy, balanced_accuracy, f1, recall and roc_auc to score the models.

Learning Curves

Learning Curves is a correlation between a learner's performance on a task and the number of attempts or time required to complete the task.

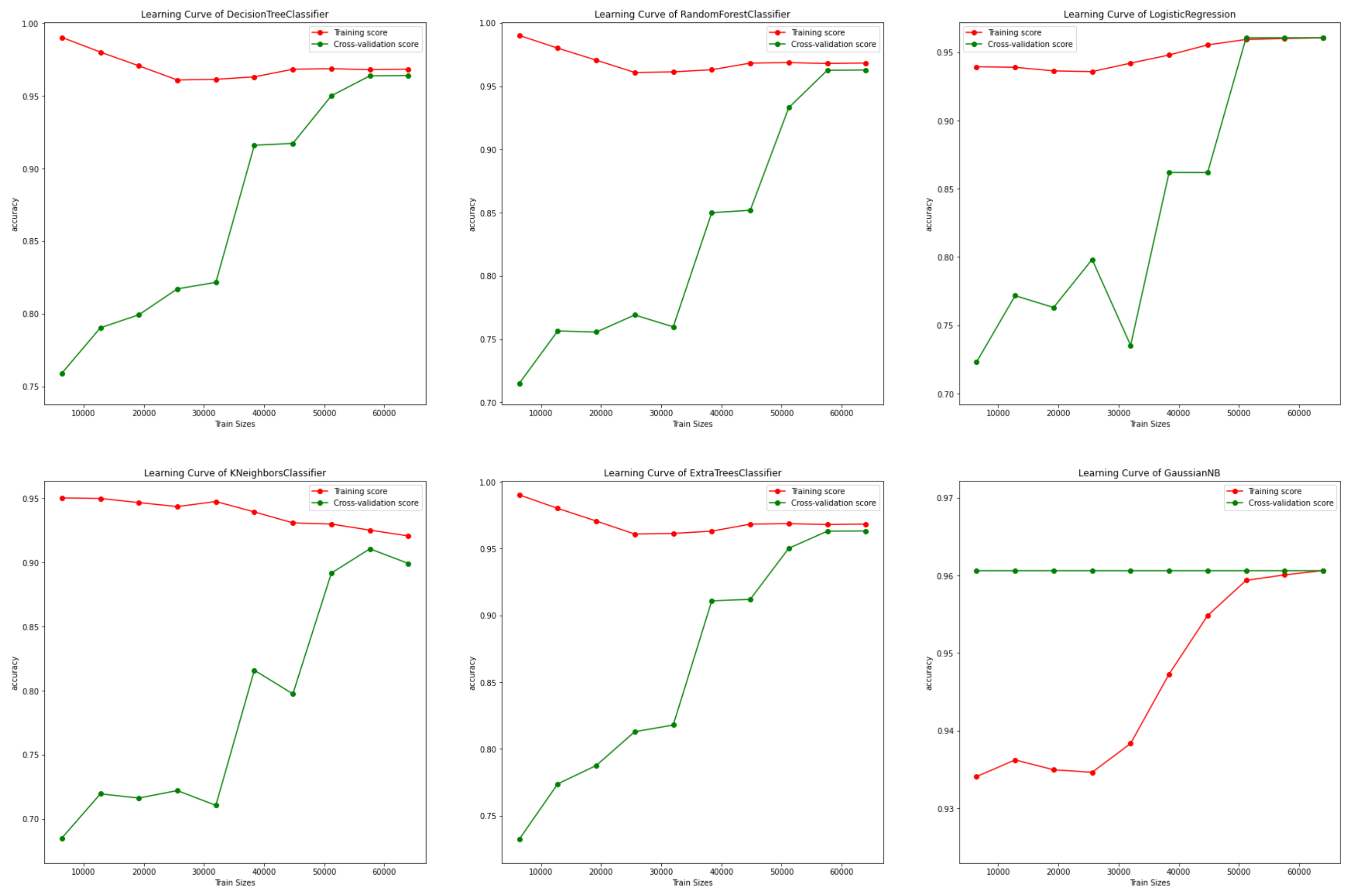
Before we begin, we need to make a function for us to plot the learning curve

In [ ]:
```python
def plot_learning_curve(
    model,
    X,
    y,
    scoring="accuracy",
    cv=StratifiedKFold(shuffle=True, random_state=42),
    train_sizes=np.linspace(0.1, 1.0, 10),
    ax=None,
):
    try:
        model_name = type(model[-1]).__name__
    except:
        model_name = type(model).__name__
    if ax is None:
        fig, ax = plt.subplots(figsize=(10, 8))
    train_sizes, train_scores, test_scores, fit_times, _ = learning_curve(
        model,
        X,
        y,
        cv=cv,
        n_jobs=-1,
        train_sizes=train_sizes,
        return_times=True,
        scoring=scoring,
    )
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)

    ax.plot(train_sizes, train_scores_mean, "o-",
            color="r", label="Training score")
    ax.plot(
        train_sizes, test_scores_mean, "o-", color="g", label="Cross-validation score"
    )
    ax.legend(loc="best")
    ax.set(
        ylim=(
            min(np.nanmin(train_scores), np.nanmin(test_scores)) - 0.01,
            max(np.nanmax(train_scores), np.nanmax(test_scores)) + 0.01,
        )
    )
    ax.set_title(f"Learning Curve of {model_name}")
    ax.set_ylabel(f"{scoring}")
    ax.set_xlabel("Train Sizes")
    return ax
```

In [ ]:
```python
# Initiate Classifiers with default hyper parameters
models = [
    ("DecisionTreeClassifier", DecisionTreeClassifier()),
    ("RandomForestClassifier", RandomForestClassifier()),
    ("LogisticRegression", LogisticRegression()),
    ("KNeighborsClassifier", KNeighborsClassifier()),
    ("ExtraTreesClassifier", ExtraTreesClassifier()),
    ("GaussianNB", GaussianNB()),
]
```

In [ ]:
```python
fig, ax = plt.subplots(2, 3, figsize=(30, 20))

out = []

for idx, subplot in enumerate(ax.ravel()):
    plot_learning_curve(
        models[idx][1],
        X_resampled_train,
        y_resampled_train,
        scoring="accuracy",
        ax=subplot
    )
    # cross validate
    score = cross_validate(
        models[idx][1],
        X_resampled_train,
        y_resampled_train,
        scoring=scoring_methods,
        n_jobs=-1,
        verbose=1,
        cv=10,
        return_train_score=True,
    )
    # get the average score and then store in a Series
    out.append(pd.Series(score, name=models[idx][0]).apply(np.mean))
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.3s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   19.9s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   10.0s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed: 24.6min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:   21.6s finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  10 out of  10 | elapsed:    1.7s finished
```



```
In [ ]: pd.DataFrame(out).sort_values(
            by=[
                "test_accuracy",
                "test_recall_weighted",
                "test_f1_weighted",
                "test_roc_auc_ovr_weighted",
            ],
            ascending=False,
        ).style.apply(
            lambda x: [
                "background-color: red; color: white" if v else "" for v in x == x.min()]
        ).apply(
            lambda x: [
                "background-color: green; color: white" if v else "" for v in x == x.max()
            ]
        )
```

Out[ ]:

	fit_time	score_time	test_accuracy	train_accuracy	test_recall_weighted	train_recall_w
DecisionTreeClassifier	0.420508	0.042110	0.963887	0.968292	0.963887	(
ExtraTreesClassifier	13.467522	0.770004	0.963125	0.968292	0.963125	(
RandomForestClassifier	12.506521	0.709096	0.962662	0.968292	0.962662	(
LogisticRegression	8.955542	0.047871	0.960613	0.960613	0.960613	(
GaussianNB	0.160938	0.096258	0.960613	0.960613	0.960613	(
KNeighborsClassifier	0.175395	110.792156	0.900350	0.921371	0.900350	(

Observations

- Some models performs equally well with 96.3% mean cross-validation accuracy score of 5 folds, highlighted in green.

- KNeighborsClassifier is the worst performing model because the data is boolean values and forming clusters using KNN is hard which explains the performance.

Selection of model

Therefore, we will use the DecisionTreeClassifier. Not only does it have one of the best accuracy fro both train and validation accuracy, it takes the shortest time to score. However, to have a better comparison of the different models, we will be also including the ExtraTreesClassifier as DecisionTree is very likely to overfit.

Initializing DecisionTreeClassifier

In [ ]:
```python
d_tree_clf = DecisionTreeClassifier()
d_tree_clf.fit(X_resampled_train, y_resampled_train)
print(
    f"Model Accuracy Score :{d_tree_clf.score(X_resampled_train, y_resampled_train)}")
scores = cross_validate(
    d_tree_clf,
    X_resampled_train, y_resampled_train,
    cv=10,
    scoring=scoring_methods,
    n_jobs=-1,
    return_train_score=True,
)
# displaying scores
display(
    pd.DataFrame(scores)
    .append(pd.Series(pd.DataFrame(scores).mean(), name="Mean"))
    .style.apply(
        lambda x: [
            "background-color: red; color: white" if v else "" for v in x == x.min()
        ]
    )
    .apply(
        lambda x: [
            "background-color: green; color: white" if v else "" for v in x == x.max()
        ]
    )
)
```

```
Model Accuracy Score :0.96825
```

```
<ipython-input-29-d0b0b0aa66b7>:15: FutureWarning: The frame.append method is deprecated and
will be removed from pandas in a future version. Use pandas.concat instead.
  pd.DataFrame(scores)
```

	fit_time	score_time	test_accuracy	train_accuracy	test_recall_weighted	train_recall_weighted	test_f1_w
0	0.425525	0.048001	0.961750	0.968458	0.961750	0.968458	(
1	0.435526	0.040165	0.966875	0.968000	0.966875	0.968000	(
2	0.412525	0.039000	0.963250	0.968444	0.963250	0.968444	(
3	0.422523	0.041002	0.965000	0.968125	0.965000	0.968125	(
4	0.421528	0.042000	0.963750	0.968319	0.963750	0.968319	(
5	0.419528	0.039007	0.962000	0.968556	0.962000	0.968556	(
6	0.408518	0.038008	0.963750	0.968333	0.963750	0.968333	(
7	0.414526	0.038002	0.964875	0.968236	0.964875	0.968236	(
8	0.419528	0.042999	0.962625	0.968417	0.962625	0.968417	(
9	0.407526	0.041003	0.964875	0.968028	0.964875	0.968028	(
Mean	0.418725	0.040919	0.963875	0.968292	0.963875	0.968292	(

Observation

- 10 fold cross validation shows that the model is strong at predicting based on high average test accuracy of 96.3% and 96.8% for training accuracy. This suggest that the model is not overfitted.

Initializing ExtraTreesClassifier

In [ ]:
```python
e_tree_clf = ExtraTreesClassifier()
e_tree_clf.fit(X_resampled_train, y_resampled_train)
print(
    f"Model Accuracy Score :{e_tree_clf.score(X_resampled_train, y_resampled_train)}")
scores = cross_validate(
    e_tree_clf,
    X_resampled_train, y_resampled_train,
    cv=10,
    scoring=scoring_methods,
    n_jobs=-1,
    return_train_score=True,
)
# displaying scores
display(
    pd.DataFrame(scores)
    .append(pd.Series(pd.DataFrame(scores).mean(), name="Mean"))
    .style.apply(
        lambda x: [
            "background-color: red; color: white" if v else "" for v in x == x.min()
        ]
    )
    .apply(
        lambda x: [
            "background-color: green; color: white" if v else "" for v in x == x.max()
        ]
    )
)
```

```
Model Accuracy Score :0.96825
```

```
<ipython-input-30-8ee10f3dd74b>:15: FutureWarning: The frame.append method is deprecated and
will be removed from pandas in a future version. Use pandas.concat instead.
  pd.DataFrame(scores)
```

	fit_time	score_time	test_accuracy	train_accuracy	test_recall_weighted	train_recall_weighted	test_f1_
0	11.762709	0.710999	0.961125	0.968458	0.961125	0.968458	
1	11.829713	0.669080	0.966250	0.968000	0.966250	0.968000	
2	12.012710	0.670998	0.962625	0.968444	0.962625	0.968444	
3	11.927710	0.661999	0.964125	0.968125	0.964125	0.968125	
4	11.816685	0.672000	0.962625	0.968319	0.962625	0.968319	
5	11.919709	0.675999	0.961500	0.968556	0.961500	0.968556	
6	11.980709	0.700002	0.962875	0.968333	0.962875	0.968333	
7	11.953001	0.684998	0.963750	0.968236	0.963750	0.968236	
8	11.848997	0.662002	0.962375	0.968417	0.962375	0.968417	
9	11.824004	0.665995	0.964375	0.968028	0.964375	0.968028	
Mean	11.887595	0.677407	0.963163	0.968292	0.963163	0.968292	

Observation

- 10 fold cross validation shows that the model is strong at predicting based on high test accuracy of 96.3% and 96.8% for training accuracy. This suggest that the model is not overfitted.

Comparing models with baseline

To compare the different attributes of the models, we will be using a few tools.

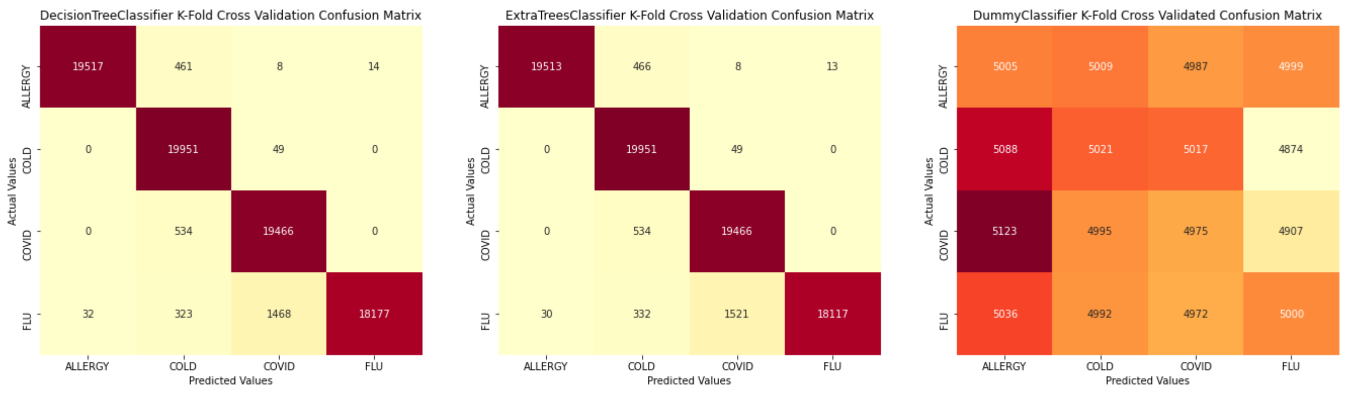
1. Confusion Matrix
2. Learning Curve

Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classifier on a set of test data for which the true values are known.

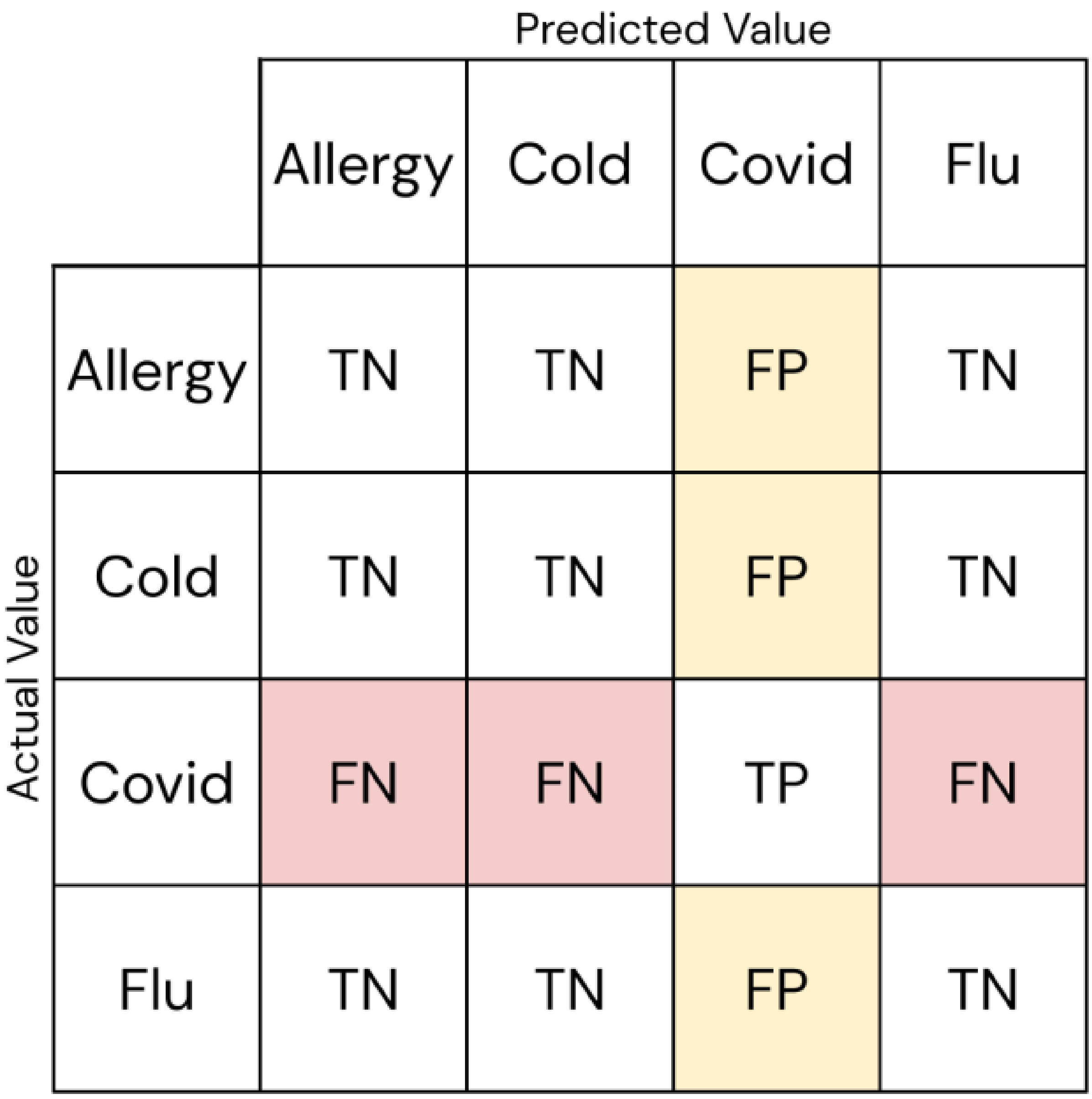
In [ ]:

```python
# Cross Validating Confusion Matrix
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(23, 6))

# DecisionTreeClassifier
d_tree_cm = confusion_matrix(
    y_resampled_train, cross_val_predict(
        d_tree_clf, X_resampled_train, y_resampled_train, cv=10)
)
sns.heatmap(d_tree_cm, annot=True, fmt="", ax=ax1, cbar=False, cmap="YlOrRd")
ax1.set_title(
    "DecisionTreeClassifier K-Fold Cross Validation Confusion Matrix")
ax1.set_ylabel("Actual Values")
ax1.set_xlabel("Predicted Values")
ax1.set_yticklabels(encoder.inverse_transform([0, 1, 2, 3]))
ax1.set_xticklabels(encoder.inverse_transform([0, 1, 2, 3]))

# ExtraTreesClassifier
e_tree_cm = confusion_matrix(
    y_resampled_train, cross_val_predict(
        e_tree_clf, X_resampled_train, y_resampled_train, cv=10)
)
sns.heatmap(e_tree_cm, annot=True, fmt="", ax=ax2, cbar=False, cmap="YlOrRd")
ax2.set_title("ExtraTreesClassifier K-Fold Cross Validation Confusion Matrix")
ax2.set_ylabel("Actual Values")
ax2.set_xlabel("Predicted Values")
ax2.set_yticklabels(encoder.inverse_transform([0, 1, 2, 3]))
ax2.set_xticklabels(encoder.inverse_transform([0, 1, 2, 3]))

# Baseline
dummy_cm = confusion_matrix(y_resampled_train, cross_val_predict(
    dummy, X_resampled_train, y_resampled_train, cv=10))
sns.heatmap(dummy_cm, annot=True, fmt="", ax=ax3, cbar=False, cmap="YlOrRd")
ax3.set_title("DummyClassifier K-Fold Cross Validated Confusion Matrix")
ax3.set_ylabel("Actual Values")
ax3.set_xlabel("Predicted Values")
ax3.set_yticklabels(encoder.inverse_transform([0, 1, 2, 3]))
ax3.set_xticklabels(encoder.inverse_transform([0, 1, 2, 3]))

plt.show()
```



How Confusion Matrix Works in this context?

The following image is created using Google Slides:



Predicted Value

Actual Value	Allergy	Cold	Covid	Flu
Allergy	TN	TN	FP	TN
Cold	TN	TN	FP	TN
Covid	FN	FN	TP	FN
Flu	TN	TN	FP	TN

Legend:
FN => False Negative
TN => True Negative
FP => False Positive
TP => True Positive

In this case, we want to be able to predict which type of illness the patient is having so that we can target specific treatment plans for the patient. Both FP and FN must be lowered so that the model can be more accurate.

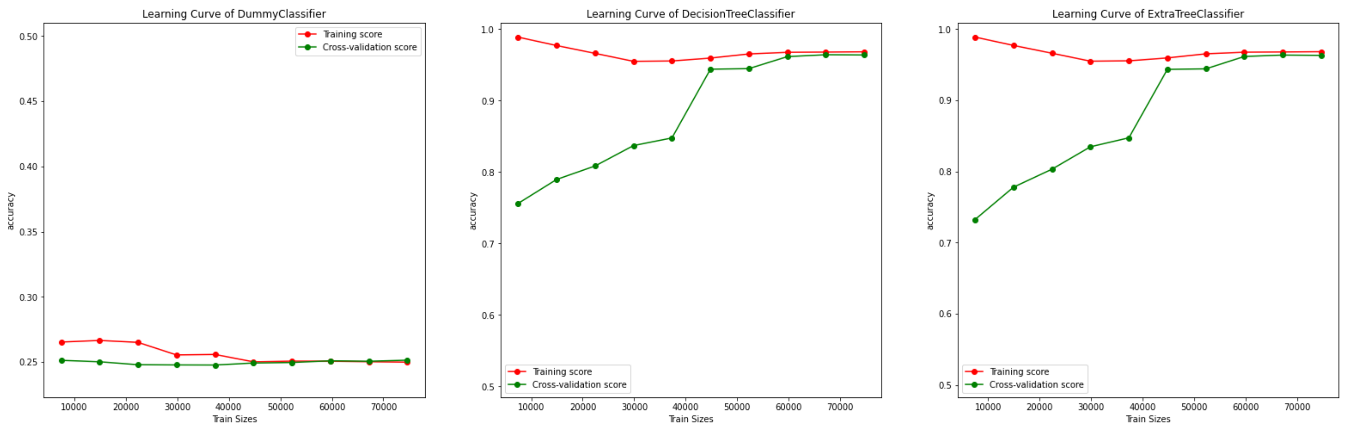
As such, we will be using accuracy as our comparing factor.

Observations

- Baseline model's predictions are all over the place
- DecisionTreeClassifier model predictions for FN is 534 (Close To 0 based %) and FP is 1525. This means that the model is able to help us predict with minor errors.
- ExtraTreesClassifier model predictions for FN is 534 (Close To 0 based %) and FP is 1574. This means that the model is able to help us predict with minor errors.

In [ ]:

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(27, 8))
plot_learning_curve(dummy, X_resampled_train, y_resampled_train,
                    scoring="accuracy", cv=15, ax=ax1)
plot_learning_curve(d_tree_clf, X_resampled_train,
                    y_resampled_train, scoring="accuracy", cv=15, ax=ax2)
plot_learning_curve(e_tree_clf, X_resampled_train,
                    y_resampled_train, scoring="accuracy", cv=15, ax=ax3)
plt.show()
```



Observations

- Learning Curve for DummyClassifier is very random and is unrepresentative. This shows that the model is severely under-fitted
- Learning Curve of ExtraTreesClassifier and DecisionTreeClassifier is increases and becomes more generalised as more data is added
- The DecisionTreeClassifier performs slightly better overall than the ExtraTreesClassifier but we will see if there is any improvement for both models

Model Improvement

Although our models perform very well, there are still some improvements to be made to our models by simplifying them to make them more generalizable. Let's see what are the possible params that can be tuned.

In [ ]:

```
list(DecisionTreeClassifier().get_params().keys())
```

Out[ ]:

```
['ccp_alpha',
 'class_weight',
 'criterion',
 'max_depth',
 'max_features',
 'max_leaf_nodes',
 'min_impurity_decrease',
 'min_samples_leaf',
 'min_samples_split',
 'min_weight_fraction_leaf',
 'random_state',
 'splitter']
```

In [ ]:

```
list(ExtraTreesClassifier().get_params().keys())
```

Out[ ]:
```
['bootstrap',
 'ccp_alpha',
 'class_weight',
 'criterion',
 'max_depth',
 'max_features',
 'max_leaf_nodes',
 'max_samples',
 'min_impurity_decrease',
 'min_samples_leaf',
 'min_samples_split',
 'min_weight_fraction_leaf',
 'n_estimators',
 'n_jobs',
 'oob_score',
 'random_state',
 'verbose',
 'warm_start']
```

Hyperparameter Tuning and Evaluation

We will be using RandomizedSearchCV to run through the parameters to see which model's parameter will give the best f1 score.

Parameters to be tuned (DecisionTreeClassifier):

1. max_depth - maximum depth of the tree, basically how big can the tree grow
2. max_leaf_nodes - stopping criteria for number of leaves

Parameters to be tuned (ExtraTreesClassifier):

1. max_depth - maximum depth of the tree, basically how big can the tree grow
2. max_leaf_nodes - stopping criteria for number of leaves
3. n_estimators - The number of trees in the forest

DecisionTreeClassifier

In [ ]:
```python
# Create the parameter grid
params_grid = {
    "max_depth": [5, 10, 20, 30, 40, 50, 60,
                                70, 80, 90, 100],
    "max_leaf_nodes": np.arange(10, 16)
    }

d_tree_grid_search = GridSearchCV(
    DecisionTreeClassifier(
        min_samples_split=2, min_samples_leaf=1, criterion="entropy", random_state=42
    ),
    params_grid,
    cv=15,
    verbose=1,
    n_jobs=-1,
    scoring="accuracy",
)

# Fitting Model
d_tree_grid_search.fit(X_resampled_train, y_resampled_train)
print(d_tree_grid_search.best_estimator_)
print(d_tree_grid_search.best_params_)
print(d_tree_grid_search.best_score_)
```

```
Fitting 15 folds for each of 66 candidates, totalling 990 fits
DecisionTreeClassifier(criterion='entropy', max_depth=10, max_leaf_nodes=15,
                       random_state=42)
{'max_depth': 10, 'max_leaf_nodes': 15}
0.8874499725599179
```

ExtraTreesClassifier

In [ ]:
```python
# Create the parameter grid
params_grid = {
    "max_depth": [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
    "max_leaf_nodes": np.arange(10, 16),
    "n_estimators": [100, 500, 1000],
}
e_tree_random_search = RandomizedSearchCV(
    ExtraTreesClassifier(
        min_samples_split=2, min_samples_leaf=1, criterion="entropy", random_state=42
    ),
    params_grid,
    cv=5,
    verbose=1,
    n_jobs=-1,
    scoring="accuracy",
    n_iter=50,
    random_state=22,
)

# Fitting Model
e_tree_random_search.fit(X_resampled_train, y_resampled_train)
print(e_tree_random_search.best_estimator_)
print(e_tree_random_search.best_params_)
print(e_tree_random_search.best_score_)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
ExtraTreesClassifier(criterion='entropy', max_depth=90, max_leaf_nodes=15,
                     n_estimators=1000, random_state=42)
{'n_estimators': 1000, 'max_leaf_nodes': 15, 'max_depth': 90}
0.9571250000000001
```

Model Evaluation

We will evaluate the model on an independent test set to see if the model is able to generalize to unseen examples

In [ ]:
```python
y_test.shape
```

Out[ ]:
```
(8891,)
```

There are 8891 unseen test examples

Initiate model after hyperparameter training

After training, we will have the model which will perform the best. Let's see if there is a boost in results after training

DecisionTreeClassifier

In [ ]:
```python
tuned_d_tree_clf = d_tree_grid_search.best_estimator_

# Fitting model
```

```
tuned_d_tree_clf.fit(X_resampled_train, y_resampled_train)
# Creating predictions
d_tree_y_pred = tuned_d_tree_clf.predict(x_test)
```

ExtraTreesClassifier

In [ ]:
```
tuned_e_tree_clf = e_tree_random_search.best_estimator_

# Fitting model
tuned_e_tree_clf.fit(X_resampled_train, y_resampled_train)
# Creating predictions
e_tree_y_pred = tuned_e_tree_clf.predict(x_test)
```

Comparing ExtraTreesClassifier and DecisionTreeClassifier

Classification Report

In [ ]:
```
# Classification Report
print(
    f"""
DecisionTreeClassifier:
{classification_report(encoder.transform(y_test), d_tree_y_pred, digits=3)}

ExtraTreesClassifier:
{classification_report(encoder.transform(y_test), e_tree_y_pred, digits=3)}
"""
)
```

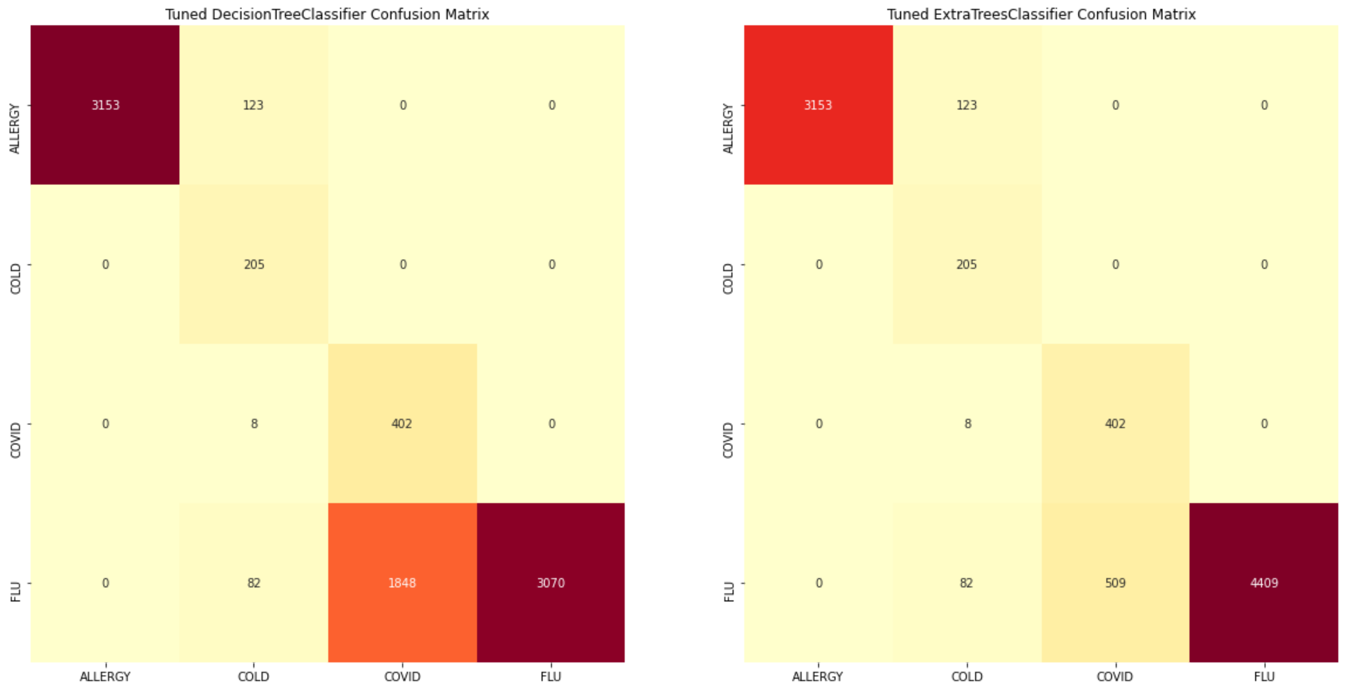
```
DecisionTreeClassifier:
              precision    recall  f1-score   support

           0      1.000     0.962     0.981      3276
           1      0.490     1.000     0.658       205
           2      0.179     0.980     0.302       410
           3      1.000     0.614     0.761      5000

    accuracy                          0.768      8891
   macro avg      0.667     0.889     0.676      8891
weighted avg      0.950     0.768     0.818      8891


ExtraTreesClassifier:
              precision    recall  f1-score   support

           0      1.000     0.962     0.981      3276
           1      0.490     1.000     0.658       205
           2      0.441     0.980     0.609       410
           3      1.000     0.882     0.937      5000

    accuracy                          0.919      8891
   macro avg      0.733     0.956     0.796      8891
weighted avg      0.962     0.919     0.932      8891
```

Observations

From our quick classification report, we can see that ExtraTreesClassifier models are more accurate in predicting whether what type of illness the patient has. If you are wondering why the untuned model has higher overall score as during our hyperparameter tuning, we limited the tree to be able to have a max depth of [5,10,20,30,40,50,60,70,80,90,100] as we do not want to make models overfit as DecisionTrees are known for overfitting. Therefore, there is a decline in overall score.

Confusion Matrix

```
In [ ]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
        # Confusion Matrix
        d_tree_tuned_cm = confusion_matrix(encoder.transform(y_test), d_tree_y_pred)
        e_tree_tuned_cm = confusion_matrix(encoder.transform(y_test), e_tree_y_pred)
        plt.figure(figsize=(10, 10))
        sns.heatmap(d_tree_tuned_cm, annot=True, fmt="",
                    cbar=False, cmap="YlOrRd", ax=ax1)
        ax1.set_title("Tuned DecisionTreeClassifier Confusion Matrix")
        ax1.set_yticklabels(encoder.inverse_transform([0, 1, 2, 3]))
        ax1.set_xticklabels(encoder.inverse_transform([0, 1, 2, 3]))
        sns.heatmap(e_tree_tuned_cm, annot=True, fmt="",
                    cbar=False, cmap="YlOrRd", ax=ax2)
        ax2.set_title("Tuned ExtraTreesClassifier Confusion Matrix")
        ax2.set_yticklabels(encoder.inverse_transform([0, 1, 2, 3]))
        ax2.set_xticklabels(encoder.inverse_transform([0, 1, 2, 3]))
        plt.show()
```

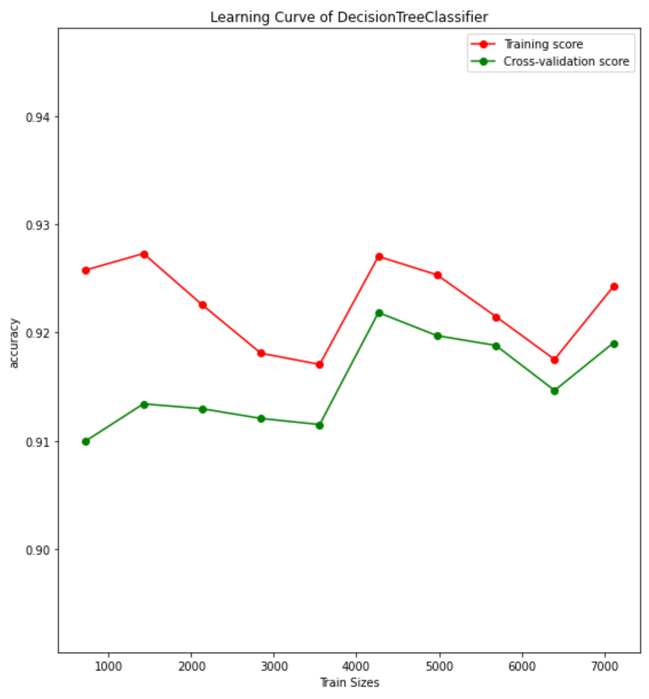


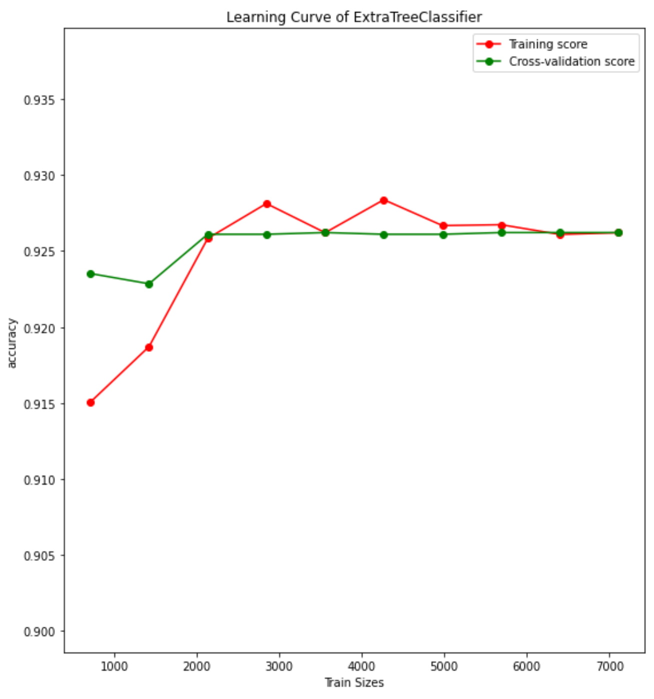
```
<Figure size 720x720 with 0 Axes>
```

Observations

We can see that our model is able to predict the TYPE with high recall as the FN value is 8 and 8 for DecisionTreeClassifier and ExtraTreesClassifier. However, we can see that for ExtraTreesClassifier, it is able to reduce value of FP at 509 compared to DecisionTreeClassifier at 1848.

```
In [ ]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 10))
        plot_learning_curve(tuned_d_tree_clf, x_test, encoder.transform(y_test), scoring="accuracy",
        plot_learning_curve(tuned_e_tree_clf, x_test, encoder.transform(y_test), scoring="accuracy",
        plt.show()
```





Observations

Based on the learning curve, we can see that the DecisionTreeClassifier and ExtraTreesClassifier has a good accuracy. But ExtraTreesClassifier have smaller variance change in accuracy score which makes it a better model.

Saving model

In [ ]:
```
pickle.dump(tuned_d_tree_clf, open("decision_tree_classifier.p", "wb"))
tuned_d_tree_clf = pickle.load(open("decision_tree_classifier.p", "rb"))
```

In [ ]:
```
pickle.dump(tuned_e_tree_clf, open("extra_trees_classifier.p", "wb"))
tuned_e_tree_clf = pickle.load(open("extra_trees_classifier.p", "rb"))
```

We will be using ExtraTreesClassifier as it has the highest accuracy after training.

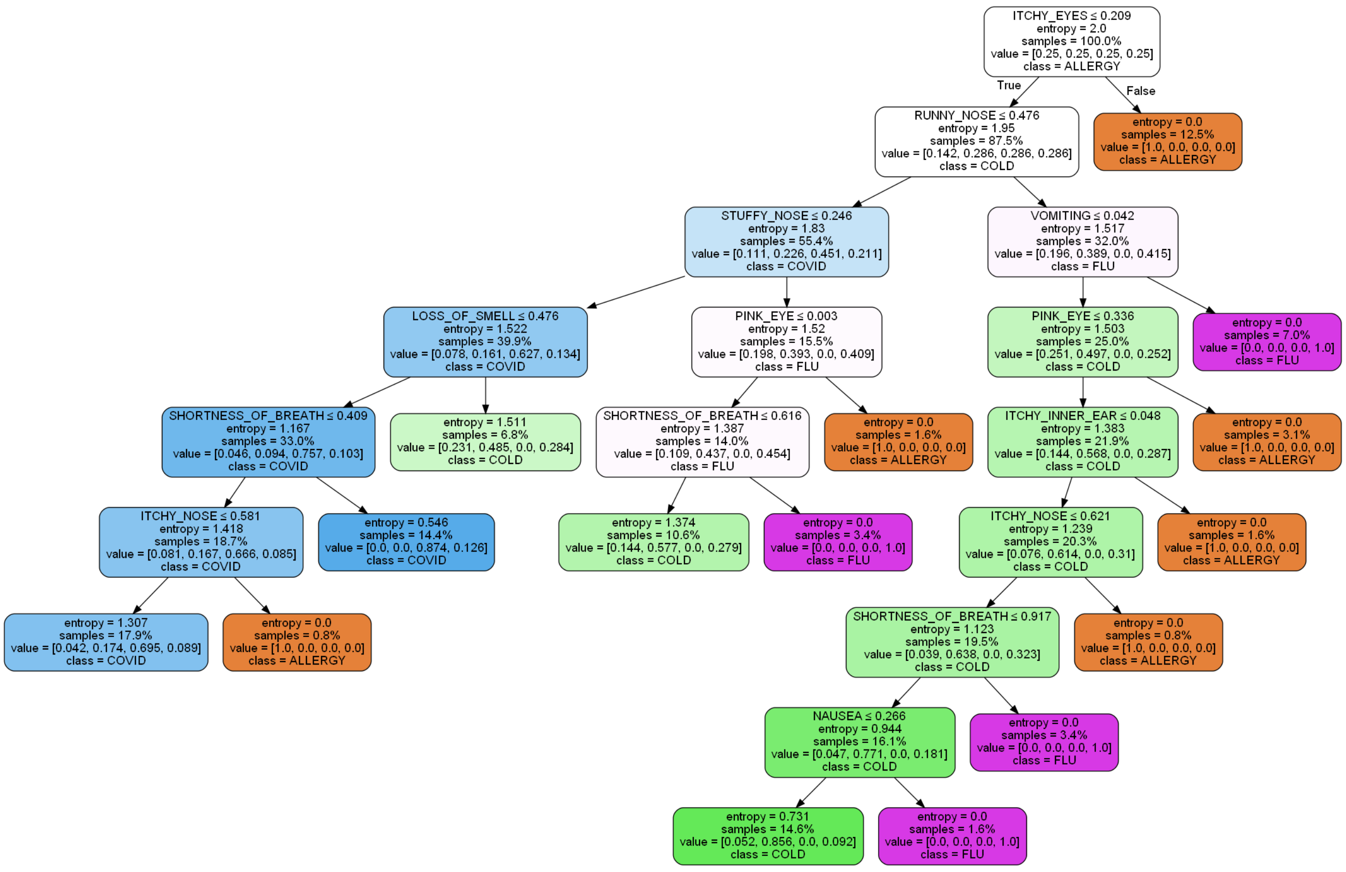
Model Visualisation

In [ ]:
```
sub_tree_42 = tuned_e_tree_clf[42]
dot_data = export_graphviz(
    sub_tree_42,
    out_file=None,
    filled=True,
    rounded=True,
    special_characters=True,
    proportion=True,
    impurity=True,  # enable them if you want
    feature_names=x_train.columns,
    class_names=encoder.inverse_transform([0, 1, 2, 3]),
)
print(dot_data)
```

```
digraph Tree {
node [shape=box, style="filled, rounded", color="black", fontname="helvetica"] ;
edge [fontname="helvetica"] ;
0 [label=<ITCHY_EYES &le; 0.209<br/>entropy = 2.0<br/>samples = 100.0%<br/>value = [0.25, 0.2
5, 0.25, 0.25]<br/>class = ALLERGY>, fillcolor="#ffffff"] ;
1 [label=<RUNNY_NOSE &le; 0.476<br/>entropy = 1.95<br/>samples = 87.5%<br/>value = [0.142, 0.
286, 0.286, 0.286]<br/>class = COLD>, fillcolor="#ffffff"] ;
0 -> 1 [labeldistance=2.5, labelangle=45, headlabel="True"] ;
3 [label=<STUFFY_NOSE &le; 0.246<br/>entropy = 1.83<br/>samples = 55.4%<br/>value = [0.111,
0.226, 0.451, 0.211]<br/>class = COVID>, fillcolor="#c5e3f7"] ;
1 -> 3 ;
5 [label=<LOSS_OF_SMELL &le; 0.476<br/>entropy = 1.522<br/>samples = 39.9%<br/>value = [0.07
8, 0.161, 0.627, 0.134]<br/>class = COVID>, fillcolor="#91c9f1"] ;
3 -> 5 ;
7 [label=<SHORTNESS_OF_BREATH &le; 0.409<br/>entropy = 1.167<br/>samples = 33.0%<br/>value =
[0.046, 0.094, 0.757, 0.103]<br/>class = COVID>, fillcolor="#6fb8ec"] ;
5 -> 7 ;
15 [label=<ITCHY_NOSE &le; 0.581<br/>entropy = 1.418<br/>samples = 18.7%<br/>value = [0.081,
0.167, 0.666, 0.085]<br/>class = COVID>, fillcolor="#88c4ef"] ;
7 -> 15 ;
27 [label=<entropy = 1.307<br/>samples = 17.9%<br/>value = [0.042, 0.174, 0.695, 0.089]<br/>c
lass = COVID>, fillcolor="#82c1ef"] ;
15 -> 27 ;
28 [label=<entropy = 0.0<br/>samples = 0.8%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
15 -> 28 ;
16 [label=<entropy = 0.546<br/>samples = 14.4%<br/>value = [0.0, 0.0, 0.874, 0.126]<br/>class
= COVID>, fillcolor="#56abe9"] ;
7 -> 16 ;
8 [label=<entropy = 1.511<br/>samples = 6.8%<br/>value = [0.231, 0.485, 0.0, 0.284]<br/>class
= COLD>, fillcolor="#cbf8c7"] ;
5 -> 8 ;
6 [label=<PINK_EYE &le; 0.003<br/>entropy = 1.52<br/>samples = 15.5%<br/>value = [0.198, 0.39
3, 0.0, 0.409]<br/>class = FLU>, fillcolor="#fefafe"] ;
3 -> 6 ;
17 [label=<SHORTNESS_OF_BREATH &le; 0.616<br/>entropy = 1.387<br/>samples = 14.0%<br/>value =
[0.109, 0.437, 0.0, 0.454]<br/>class = FLU>, fillcolor="#fef9fe"] ;
6 -> 17 ;
19 [label=<entropy = 1.374<br/>samples = 10.6%<br/>value = [0.144, 0.577, 0.0, 0.279]<br/>cla
ss = COLD>, fillcolor="#b3f4ad"] ;
17 -> 19 ;
20 [label=<entropy = 0.0<br/>samples = 3.4%<br/>value = [0.0, 0.0, 0.0, 1.0]<br/>class = FLU
>, fillcolor="#d739e5"] ;
17 -> 20 ;
18 [label=<entropy = 0.0<br/>samples = 1.6%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
6 -> 18 ;
4 [label=<VOMITING &le; 0.042<br/>entropy = 1.517<br/>samples = 32.0%<br/>value = [0.196, 0.3
89, 0.0, 0.415]<br/>class = FLU>, fillcolor="#fdf6fe"] ;
1 -> 4 ;
9 [label=<PINK_EYE &le; 0.336<br/>entropy = 1.503<br/>samples = 25.0%<br/>value = [0.251, 0.4
97, 0.0, 0.252]<br/>class = COLD>, fillcolor="#c3f6be"] ;
4 -> 9 ;
11 [label=<ITCHY_INNER_EAR &le; 0.048<br/>entropy = 1.383<br/>samples = 21.9%<br/>value = [0.
144, 0.568, 0.0, 0.287]<br/>class = COLD>, fillcolor="#b6f5b1"] ;
9 -> 11 ;
13 [label=<ITCHY_NOSE &le; 0.621<br/>entropy = 1.239<br/>samples = 20.3%<br/>value = [0.076,
0.614, 0.0, 0.31]<br/>class = COLD>, fillcolor="#aef4a8"] ;
11 -> 13 ;
21 [label=<SHORTNESS_OF_BREATH &le; 0.917<br/>entropy = 1.123<br/>samples = 19.5%<br/>value =
[0.039, 0.638, 0.0, 0.323]<br/>class = COLD>, fillcolor="#a9f3a3"] ;
13 -> 21 ;
23 [label=<NAUSEA &le; 0.266<br/>entropy = 0.944<br/>samples = 16.1%<br/>value = [0.047, 0.77
1, 0.0, 0.181]<br/>class = COLD>, fillcolor="#7aec70"] ;
21 -> 23 ;
25 [label=<entropy = 0.731<br/>samples = 14.6%<br/>value = [0.052, 0.856, 0.0, 0.092]<br/>cla
ss = COLD>, fillcolor="#64e958"] ;
23 -> 25 ;
26 [label=<entropy = 0.0<br/>samples = 1.6%<br/>value = [0.0, 0.0, 0.0, 1.0]<br/>class = FLU
>, fillcolor="#d739e5"] ;
23 -> 26 ;
24 [label=<entropy = 0.0<br/>samples = 3.4%<br/>value = [0.0, 0.0, 0.0, 1.0]<br/>class = FLU
>, fillcolor="#d739e5"] ;
21 -> 24 ;
22 [label=<entropy = 0.0<br/>samples = 0.8%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
13 -> 22 ;
14 [label=<entropy = 0.0<br/>samples = 1.6%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
11 -> 14 ;
12 [label=<entropy = 0.0<br/>samples = 3.1%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
9 -> 12 ;
10 [label=<entropy = 0.0<br/>samples = 7.0%<br/>value = [0.0, 0.0, 0.0, 1.0]<br/>class = FLU
>, fillcolor="#d739e5"] ;
4 -> 10 ;
2 [label=<entropy = 0.0<br/>samples = 12.5%<br/>value = [1.0, 0.0, 0.0, 0.0]<br/>class = ALLE
RGY>, fillcolor="#e58139"] ;
0 -> 2 [labeldistance=2.5, labelangle=-45, headlabel="False"] ;
}
```

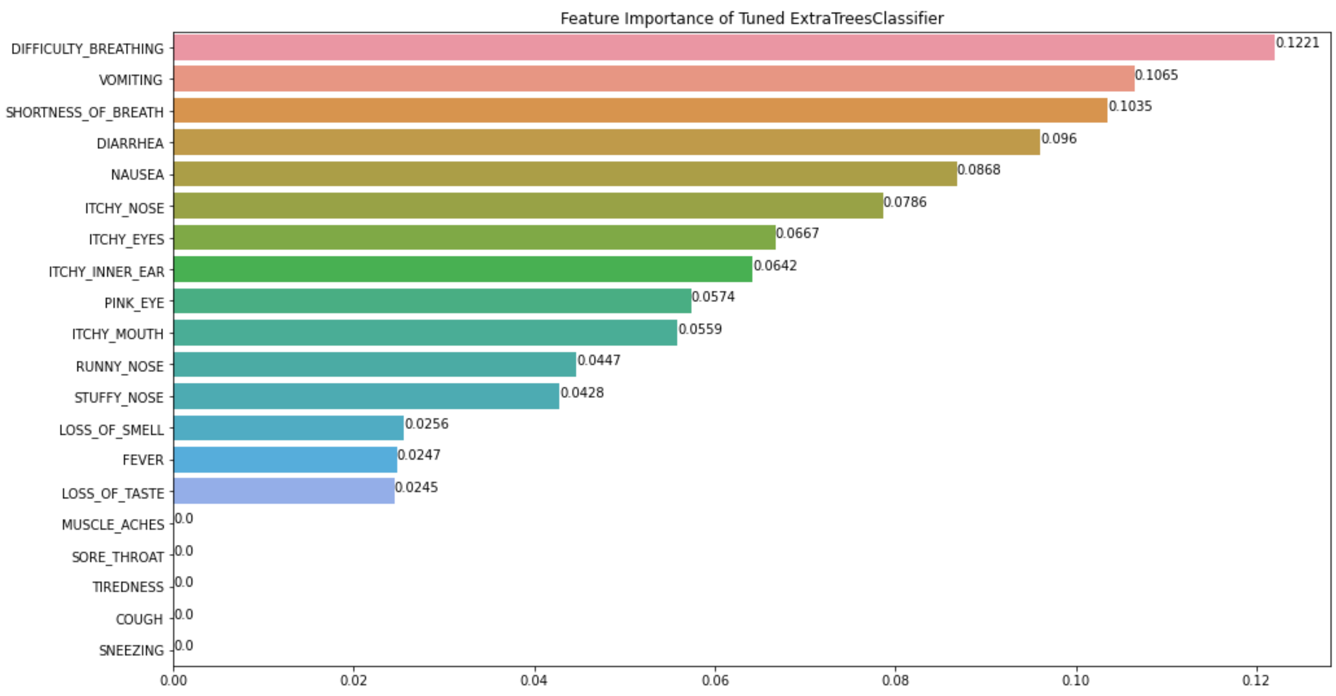
In [ ]:
```python
from graphviz import Source
from PIL import Image

src = Source(dot_data, filename="extraTrees", format="png")
Image.open('./extraTrees.png')
```

Out[ ]:


ITCHY_EYES ≤ 0.209 entropy = 2.0 samples = 100.0% value = [0.25, 0.25, 0.25, 0.25] class = ALLERGY
True
False
RUNNY_NOSE ≤ 0.476 entropy = 1.95 samples = 87.5% value = [0.142, 0.286, 0.286, 0.286] class = COLD
entropy = 0.0 samples = 12.5% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
STUFFY_NOSE ≤ 0.246 entropy = 1.83 samples = 55.4% value = [0.111, 0.226, 0.451, 0.211] class = COVID
VOMITING ≤ 0.042 entropy = 1.517 samples = 32.0% value = [0.196, 0.389, 0.0, 0.415] class = FLU
LOSS_OF_SMELL ≤ 0.476 entropy = 1.522 samples = 39.9% value = [0.078, 0.161, 0.627, 0.134] class = COVID
PINK_EYE ≤ 0.003 entropy = 1.52 samples = 15.5% value = [0.198, 0.393, 0.0, 0.409] class = FLU
PINK_EYE ≤ 0.336 entropy = 1.503 samples = 25.0% value = [0.251, 0.497, 0.0, 0.252] class = COLD
entropy = 0.0 samples = 7.0% value = [0.0, 0.0, 0.0, 1.0] class = FLU
SHORTNESS_OF_BREATH ≤ 0.409 entropy = 1.167 samples = 33.0% value = [0.046, 0.094, 0.757, 0.103] class = COVID
entropy = 1.511 samples = 6.8% value = [0.231, 0.485, 0.0, 0.284] class = COLD
SHORTNESS_OF_BREATH ≤ 0.616 entropy = 1.387 samples = 14.0% value = [0.109, 0.437, 0.0, 0.454] class = FLU
entropy = 0.0 samples = 1.6% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
ITCHY_INNER_EAR ≤ 0.048 entropy = 1.383 samples = 21.9% value = [0.144, 0.568, 0.0, 0.287] class = COLD
entropy = 0.0 samples = 3.1% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
ITCHY_NOSE ≤ 0.581 entropy = 1.418 samples = 18.7% value = [0.081, 0.167, 0.666, 0.085] class = COVID
entropy = 0.546 samples = 14.4% value = [0.0, 0.0, 0.874, 0.126] class = COVID
entropy = 1.374 samples = 10.6% value = [0.144, 0.577, 0.0, 0.279] class = COLD
entropy = 0.0 samples = 3.4% value = [0.0, 0.0, 0.0, 1.0] class = FLU
ITCHY_NOSE ≤ 0.621 entropy = 1.239 samples = 20.3% value = [0.076, 0.614, 0.0, 0.31] class = COLD
entropy = 0.0 samples = 1.6% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
entropy = 1.307 samples = 17.9% value = [0.042, 0.174, 0.695, 0.089] class = COVID
entropy = 0.0 samples = 0.8% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
SHORTNESS_OF_BREATH ≤ 0.917 entropy = 1.123 samples = 19.5% value = [0.039, 0.638, 0.0, 0.323] class = COLD
entropy = 0.0 samples = 0.8% value = [1.0, 0.0, 0.0, 0.0] class = ALLERGY
NAUSEA ≤ 0.266 entropy = 0.944 samples = 16.1% value = [0.047, 0.771, 0.0, 0.181] class = COLD
entropy = 0.0 samples = 3.4% value = [0.0, 0.0, 0.0, 1.0] class = FLU
entropy = 0.731 samples = 14.6% value = [0.052, 0.856, 0.0, 0.092] class = COLD
entropy = 0.0 samples = 1.6% value = [0.0, 0.0, 0.0, 1.0] class = FLU

Features Importance

In [ ]:
```python
importance = pd.Series(
    tuned_e_tree_clf.feature_importances_, index=x_train.columns
).sort_values(ascending=False)
fig, ax = plt.subplots(figsize=(16, 9))
sns.barplot(
    x=importance[importance >
                 0].values, y=importance[importance > 0].index, ax=ax
)
ax.set_title("Feature Importance of Tuned {}".format("ExtraTreesClassifier"))
[
    ax.text(x=v, y=i, s=round(v, 4))
    for i, v in zip(np.arange(0, len(importance)), importance[importance > 0].values)
]
plt.show()
```



Summary

We have successfully created a model that is able to predict what type of illness the patient has based on the different symptoms. Through extensive feature engineering, we have been able to develop a simple model that is able to predict what the patients have.