**THYROID DISEASE CLASSIFICATION USING ML**

**CONCEPTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER** | **TITLE** | | **PAGE.NO.** |
| 1 | INTRODUCTION   * 1. OVERVIEW   2. PURPOSE | |  |
| 2 | PROBLEM DEFINITION & DESIGN THINKING  2.1 EMPATHY MAP  2.2 IDEATION & BRAINSTOMING MAP | |  |
| 3 | RESULT | |  |
| 4 | ADVANTAGES & DISADVANTAGES | |  |
| 5 | APPLICATIONS | |  |
| 6 | CONCLUSION | |  |
| 7 | FUTURE SCOP |  | |
| 8 | APPENDIX  8.1 SOURCE CODE |  | |

**Abstract:**

Thyroid disease is a common disorder that affects the thyroid glands ability to regulate metabolism. There are two types of thyroid disorders: hyperthyroidism and hypothyroidism, both of which can have serious health consequences if left untreated. To diagnose thyroid disease, healthcare professionals use blood tests to detect the presence of certain hormones. However, the accuracy of these tests can be affected by noise and other factors, making it difficult to diagnose the disease accurately.

Thus the proposed system used Machine learning algorithms - random forest classifiers have shown promise in accurately predicting thyroid disease. These algorithms can also classify the type of thyroid disease a patient has, which can help healthcare professionals to provide appropriate treatment. This paper presents the classification of thyroid disease using one of the machine learning algorithms can provide accurate predictions and help healthcare professionals to provide appropriate treatment.

**CHAPTER 1**

**INTRODUCTION**

* 1. OVERVIEW

The project "Thyroid Disease Classification Using ML" aims to build a machine learning model that can predict the presence or absence of thyroid disease in patients based on their medical test results. The dataset used in this project contains 200 records of patients, each with 5 medical test results and a target variable indicating whether the patient has thyroid disease or not.

The objective of this project is to explore different machine learning algorithms and techniques to build a model that can accurately classify patients into those with or without thyroid disease. The goal is to provide a tool that can assist healthcare professionals in making accurate diagnoses and improve patient outcomes.

* 1. PURPOSE

The purpose of this project is to develop a machine learning model that can classify patients with or without thyroid disease based on their medical test results. The model can be used by healthcare professionals to assist in diagnosing thyroid disease and providing appropriate treatment to patients. The model can also be used to identify patients at risk of developing thyroid disease, allowing for early intervention and preventive measures.

The project's ultimate goal is to improve patient outcomes by providing a tool that can assist healthcare professionals in making accurate diagnoses and providing personalized treatment plans.

**CHAPTER 2**

**PROBLEM DEFINITION & DESIGN THINKING**

2.1 PROBLEM DEFINITION:

The problem is to develop a machine learning model that can accurately classify patients into different categories based on their thyroid disease status. Thyroid disease is a common endocrine disorder that affects millions of people worldwide. The diagnosis of thyroid disease is currently done through blood tests and other medical examinations, which can be time-consuming and expensive. Hence, there is a need to develop an automated system that can accurately classify patients into different thyroid disease categories.

2.2 DESIGN THINKING:

2.2.1 EMPATHY MAP

The empathy map is a tool that helps us understand the user's perspective and empathize with their needs and pain points. In the context of thyroid disease classification, the empathy map can include the following:

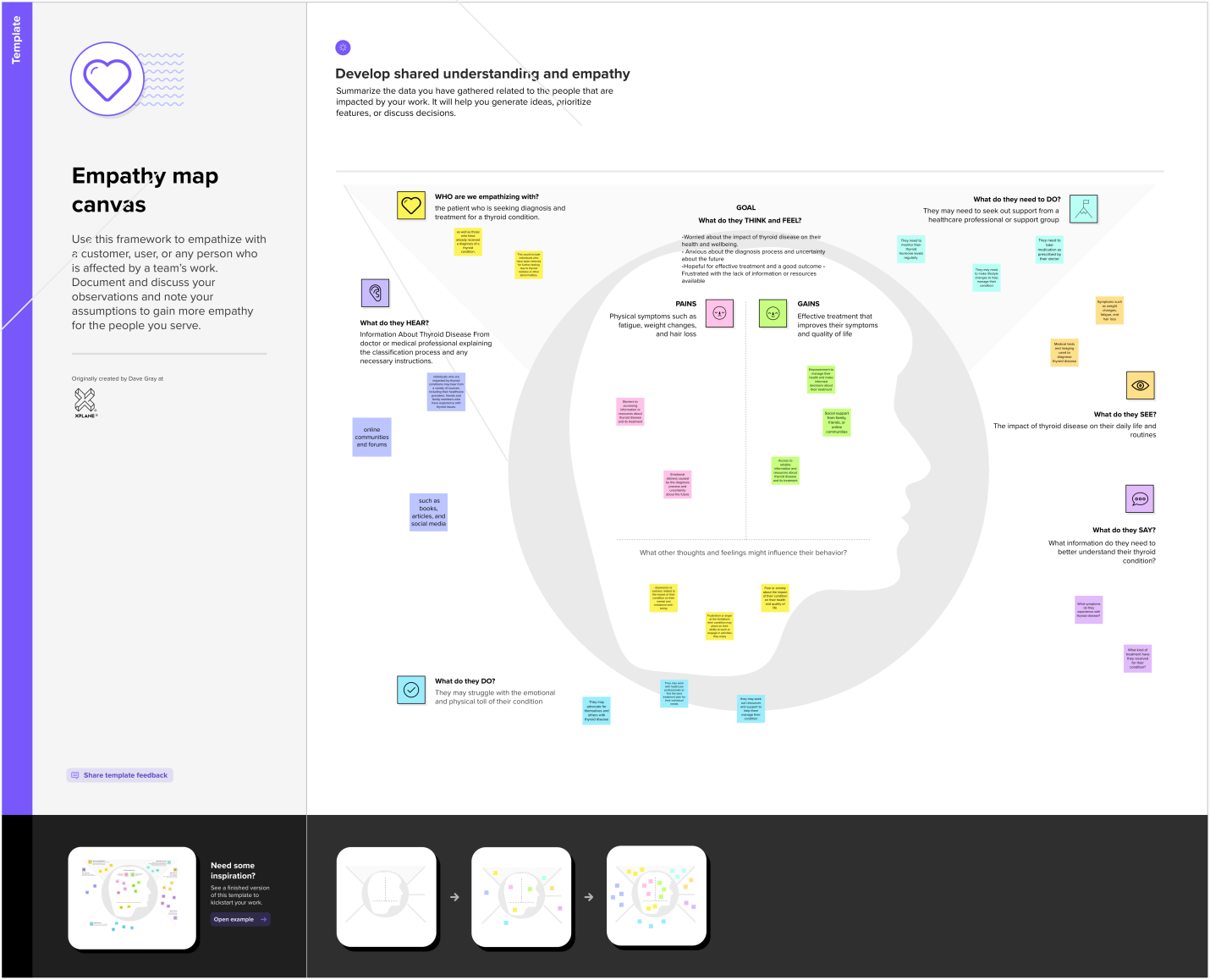
* Who are the patients who suffer from thyroid disease?
* What are their feelings and emotions towards the disease?
* What are their pain points and challenges in the diagnosis and treatment process?
* What are their goals and aspirations in managing the disease?

2.2.2 IDEATION & BRAINSTOMING MAP

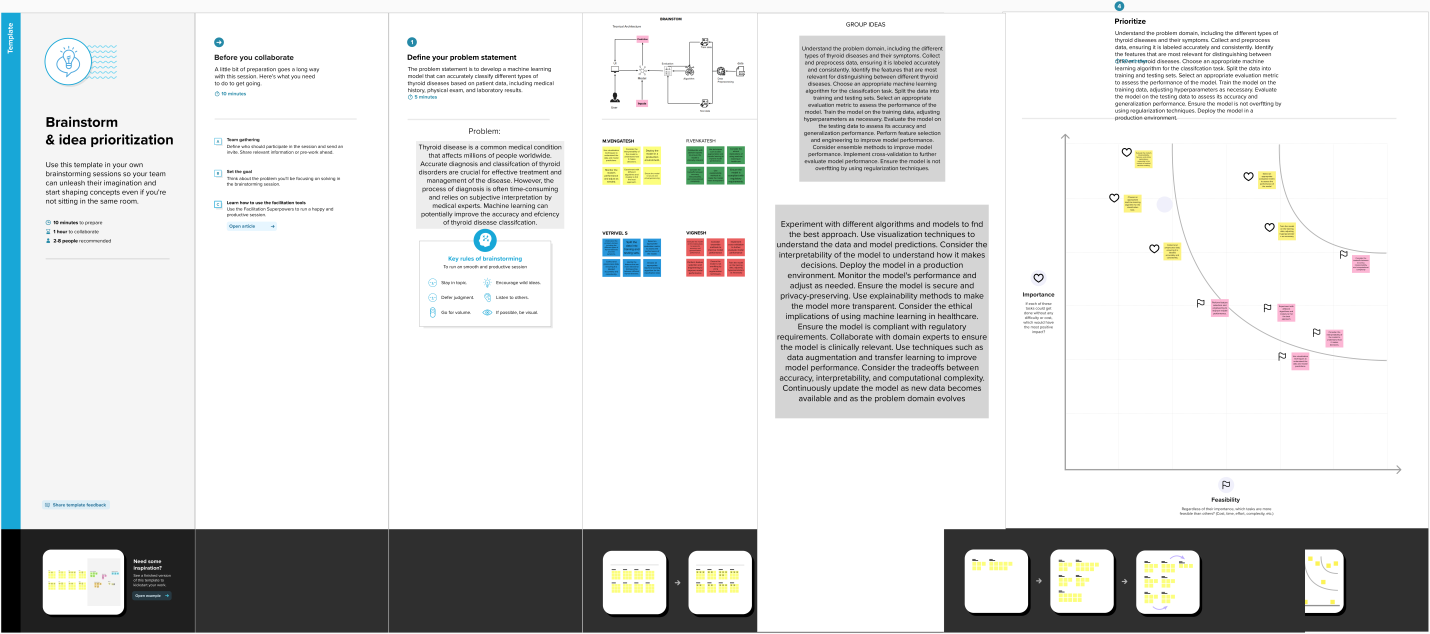
Once we have a clear understanding of the user's perspective, we can start brainstorming and ideating different solutions to solve the problem. In the case of thyroid disease classification, some of the ideas could include:

* Developing a machine learning model that uses patient data such as blood tests, symptoms, and medical history to predict the thyroid disease status.
* Using wearable devices and sensors to collect data on the patient's physiological parameters such as heart rate, body temperature, and sleep patterns to diagnose thyroid disease.
* Creating a mobile app that patients can use to track their symptoms and monitor their thyroid disease status.
* Developing a chatbot that can provide personalized advice and guidance to patients based on their thyroid disease status.

These are just a few examples of the many ideas that can be generated through the ideation and brainstorming process. The next step would be to select the most feasible and effective solution and implement it through the machine learning­­ model.

****

* **IN THIS PROJECT WE CAN SEE THE EMPATHY MAP IN THE GIVEN LINK BELOW**
* [**https://app.mural.co/t/thyroiddiseaseclassification2361/m/thyroiddiseaseclassification2361/1679550670889/a3605791494de72322673b100d3656f241dff2c7?wid=194-1664821606911&outline=open**](https://app.mural.co/t/thyroiddiseaseclassification2361/m/thyroiddiseaseclassification2361/1679550670889/a3605791494de72322673b100d3656f241dff2c7?wid=194-1664821606911&outline=open)



* **IN THIS PROJECT WE CAN SEE THE EMPATHY MAP IN THE GIVEN LINK BELOW**
* [**https://app.mural.co/t/thyroiddiseaseclassification2361/m/thyroiddiseaseclassification2361/1679951295728/8b687b22d05970767b64c7933868c83228870038?sender=uf8329c858f540bcbb7528181**](https://app.mural.co/t/thyroiddiseaseclassification2361/m/thyroiddiseaseclassification2361/1679951295728/8b687b22d05970767b64c7933868c83228870038?sender=uf8329c858f540bcbb7528181)

**CHAPTER 3**

**RESULT:**

**Result 1:**

* Import all the tools we need

All needed tools import successful.

**Result 2:**

|  | **Age** | **Sex** | **BP** | **Cholesterol** | **Na\_to\_K** | **Drug** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | 23 | F | HIGH | HIGH | 25.355 | DrugY |
| **1** | 47 | M | LOW | HIGH | 13.093 | drugC |
| **2** | 47 | M | LOW | HIGH | 10.114 | drugC |
| **3** | 28 | F | NORMAL | HIGH | 7.798 | drugX |
| **4** | 61 | F | LOW | HIGH | 18.043 | DrugY |

**Result 3:**

* 5
* array([0, 3, 4, 1, 2])
* array([0, 1])
* array([0, 1, 2])
* array(['F', 'M'], dtype=object)

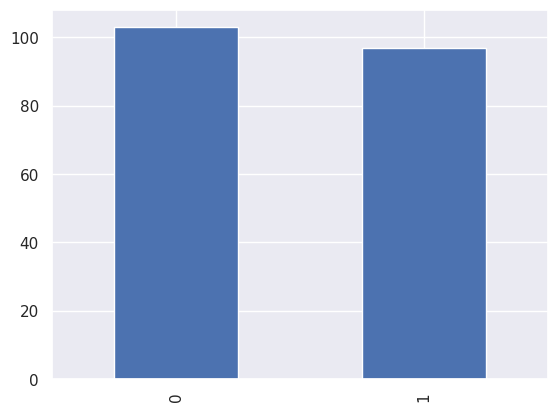
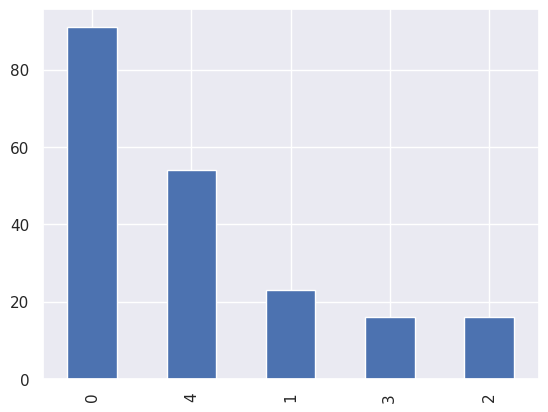
**Result 4:**

| **Age** | **Sex** | **BP** | **Cholesterol** | **Na\_to\_K** | **Drug** |
| --- | --- | --- | --- | --- | --- |
| **0** | 23 | 0 | 0 | 0 | 25.355 | 0 |
| **1** | 47 | 1 | 1 | 0 | 13.093 | 3 |
| **2** | 47 | 1 | 1 | 0 | 10.114 | 3 |
| **3** | 28 | 0 | 2 | 0 | 7.798 | 4 |
| **4** | 61 | 0 | 1 | 0 | 18.043 | 0 |

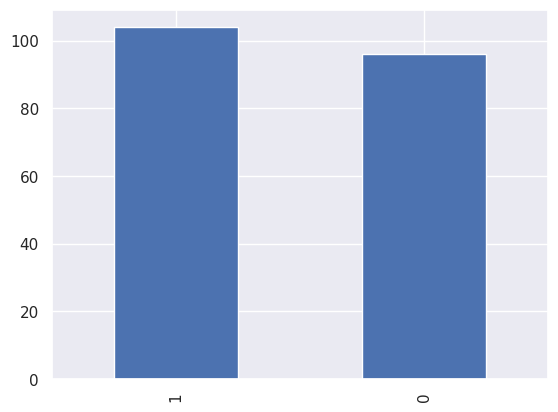
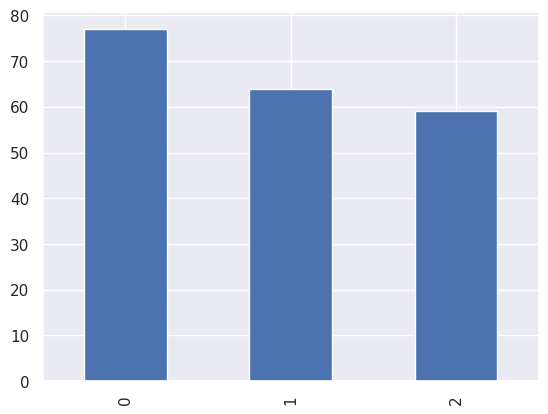
**Result 5:**

* 200

**Result 6:**



**Result 7:**



**Result 8:**

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

RangeIndex: 200 entries, 0 to 199

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Age 200 non-null int64

1 Sex 200 non-null int64

2 BP 200 non-null int64

3 Cholesterol 200 non-null int64

4 Na\_to\_K 200 non-null float64

5 Drug 200 non-null int64

dtypes: float64(1), int64(5)

memory usage: 9.5 KB

* Are there any missing values

**Result 9:**

|  |  |
| --- | --- |
| Age | 0 |
| Sex | 0 |
| BP | 0 |
| Cholesterol | 0 |
| Na\_to\_K | 0 |
| Drug | 0 |
| Dtype | int64 |

There are none!

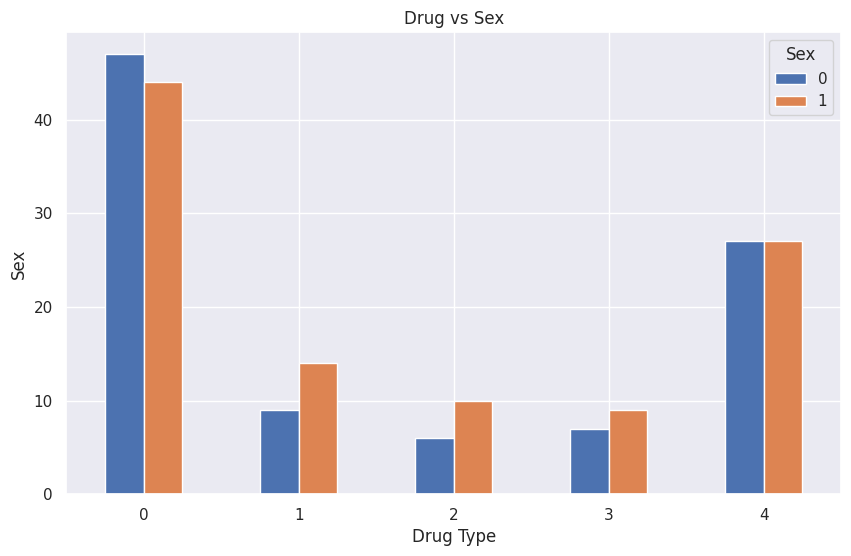
**Result 10:**

|  | **Age** | **Sex** | **BP** | **Cholesterol** | **Na\_to\_K** | **Drug** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 200.000000 | 200.000000 | 200.000000 | 200.000000 | 200.000000 | 200.000000 |
| **mean** | 44.315000 | 0.520000 | 0.910000 | 0.485000 | 16.084485 | 1.595000 |
| **std** | 16.544315 | 0.500854 | 0.821752 | 0.501029 | 7.223956 | 1.716305 |
| **min** | 15.000000 | 0.000000 | 0.000000 | 0.000000 | 6.269000 | 0.000000 |
| **25%** | 31.000000 | 0.000000 | 0.000000 | 0.000000 | 10.445500 | 0.000000 |
| **50%** | 45.000000 | 1.000000 | 1.000000 | 0.000000 | 13.936500 | 1.000000 |
| **75%** | 58.000000 | 1.000000 | 2.000000 | 1.000000 | 19.380000 | 4.000000 |
| **max** | 74.000000 | 1.000000 | 2.000000 | 1.000000 | 38.247000 | 4.000000 |

**Result 11:**

| **Sex** | **0** | **1** |
| --- | --- | --- |
| **Drug** |  |  |
| **0** | 47 | 44 |
| **1** | 9 | 14 |
| **2** | 6 | 10 |
| **3** | 7 | 9 |
| **4** | 27 | 27 |

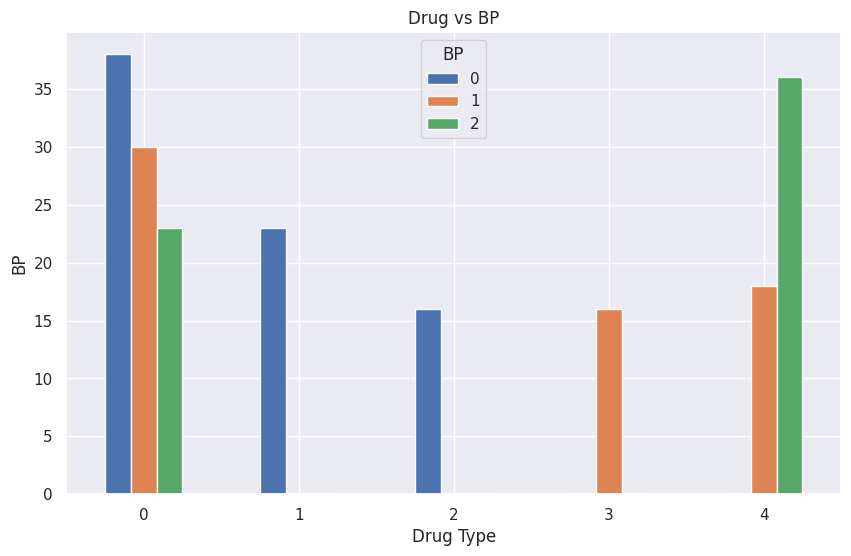
**Result 12:**



**Result 13:**

| **BP** | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **Drug** |  |  |  |
| **0** | 38 | 30 | 23 |
| **1** | 23 | 0 | 0 |
| **2** | 16 | 0 | 0 |
| **3** | 0 | 16 | 0 |
| **4** | 0 | 18 | 36 |

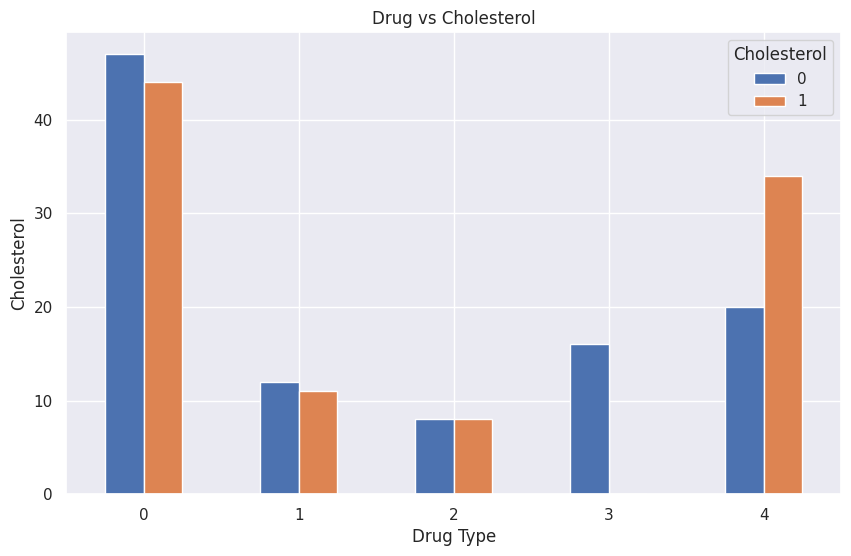
**Result 14:**



**Result 15:**

| **Cholesterol** | **0** | **1** |
| --- | --- | --- |
| **Drug** |  |  |
| **0** | 47 | 44 |
| **1** | 12 | 11 |
| **2** | 8 | 8 |
| **3** | 16 | 0 |
| **4** | 20 | 34 |

**Result 16:**



**Result 17:**

| **Cholesterol** | **0** | **1** |
| --- | --- | --- |
| **Sex** |  |  |
| **0** | 49 | 47 |
| **1** | 54 | 50 |

**Result 18:**

| **BP** | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **Sex** |  |  |  |
| **0** | 38 | 28 | 30 |
| **1** | 39 | 36 | 29 |

**Result 19:**

| **BP** | **0** | **1** | **2** |
| --- | --- | --- | --- |
| **Cholesterol** |  |  |  |
| **0** | 35 | 31 | 37 |
| **1** | 42 | 33 | 22 |

**Result 20:**

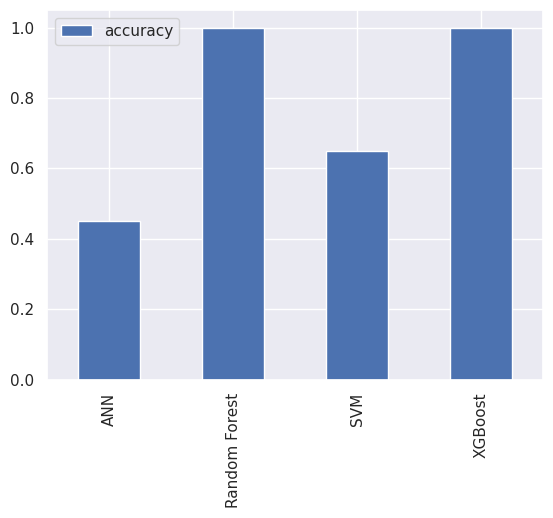
* Split data into train and test sets
* Split data into train and test sets successful.
* Split into train & test set

Split into train & test set successful.

**Result 21:**

* 5
* {'ANN': 0.45, 'Random Forest': 1.0, 'SVM': 0.65, 'XGBoost': 1.0}

**Result 22:**



**Result 23:**

* Setup random seed

Setup random hyperparameter search for RandomForestClassifier

Fit random hyperparameter search model for RandomForestClassifier()

Fitting 5 folds for each of 20 candidates, totalling 100 fits

RandomizedSearchCV

RandomizedSearchCV(cv=5, estimator=RandomForestClassifier(), n\_iter=20,

param\_distributions={'max\_depth': [None, 3, 5, 10],

'min\_samples\_leaf': array([ 1, 3, 5, 7, 9, 11, 13, 15, 17, 19]),

'min\_samples\_split': array([ 2, 4, 6, 8, 10, 12, 14, 16, 18]),'n\_estimators': array([ 10, 60, 110, 160, 210, 260, 310, 360, 410, 460, 510, 560, 610,

660, 710, 760, 810, 860, 910, 960])},

verbose=True)

estimator: RandomForestClassifier

RandomForestClassifier()

RandomForestClassifier

All setup or executed successful.

**Result 24:**

{'n\_estimators': 510,

'min\_samples\_split': 14,

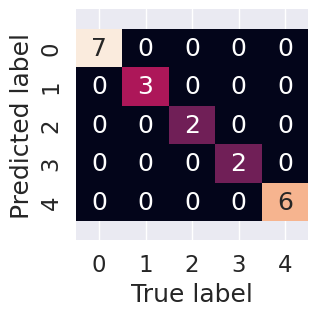
'min\_samples\_leaf': 1,

'max\_depth': None}

**Result 25:**

* 1.0

**Result 26:**



**Result 27:**

precision recall f1-score support

0 1.00 1.00 1.00 7

1 1.00 1.00 1.00 3

2 1.00 1.00 1.00 2

3 1.00 1.00 1.00 2

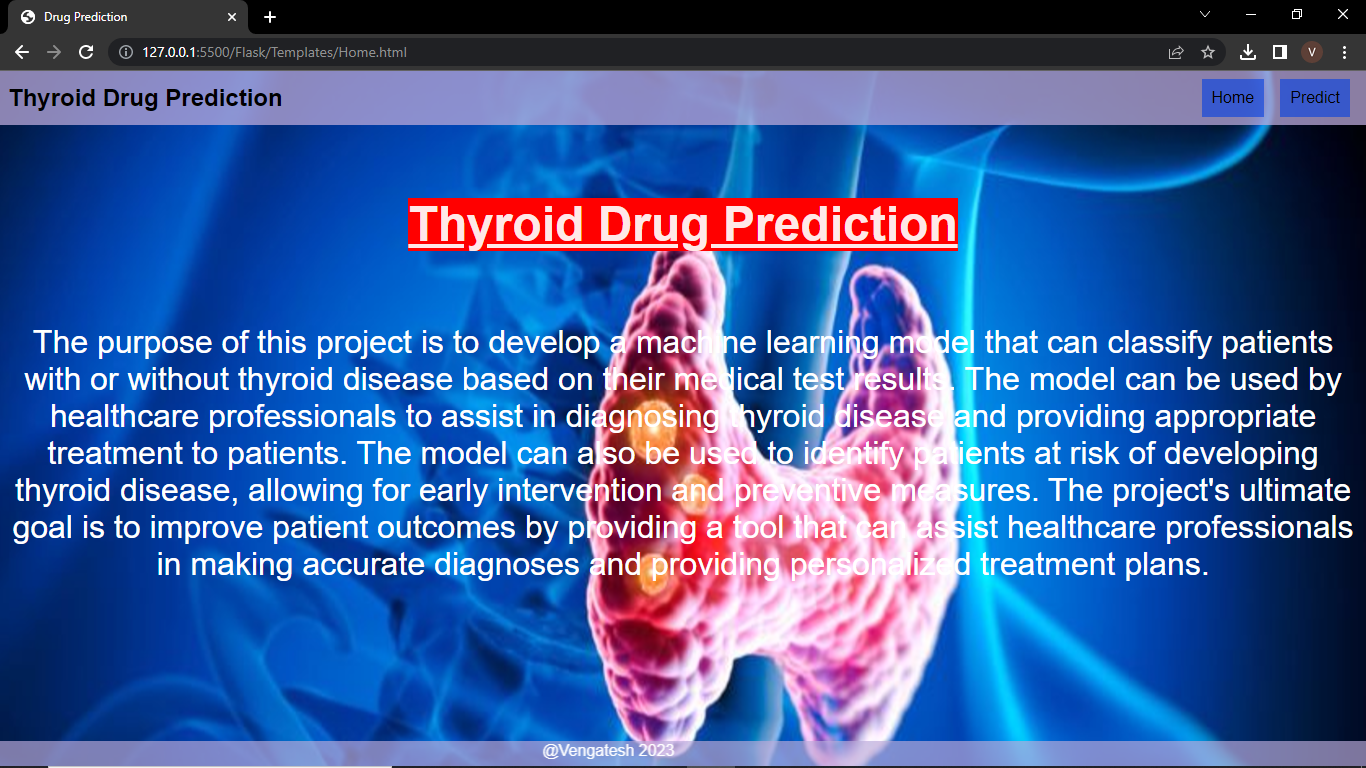
4 1.00 1.00 1.00 6

accuracy 1.00 20

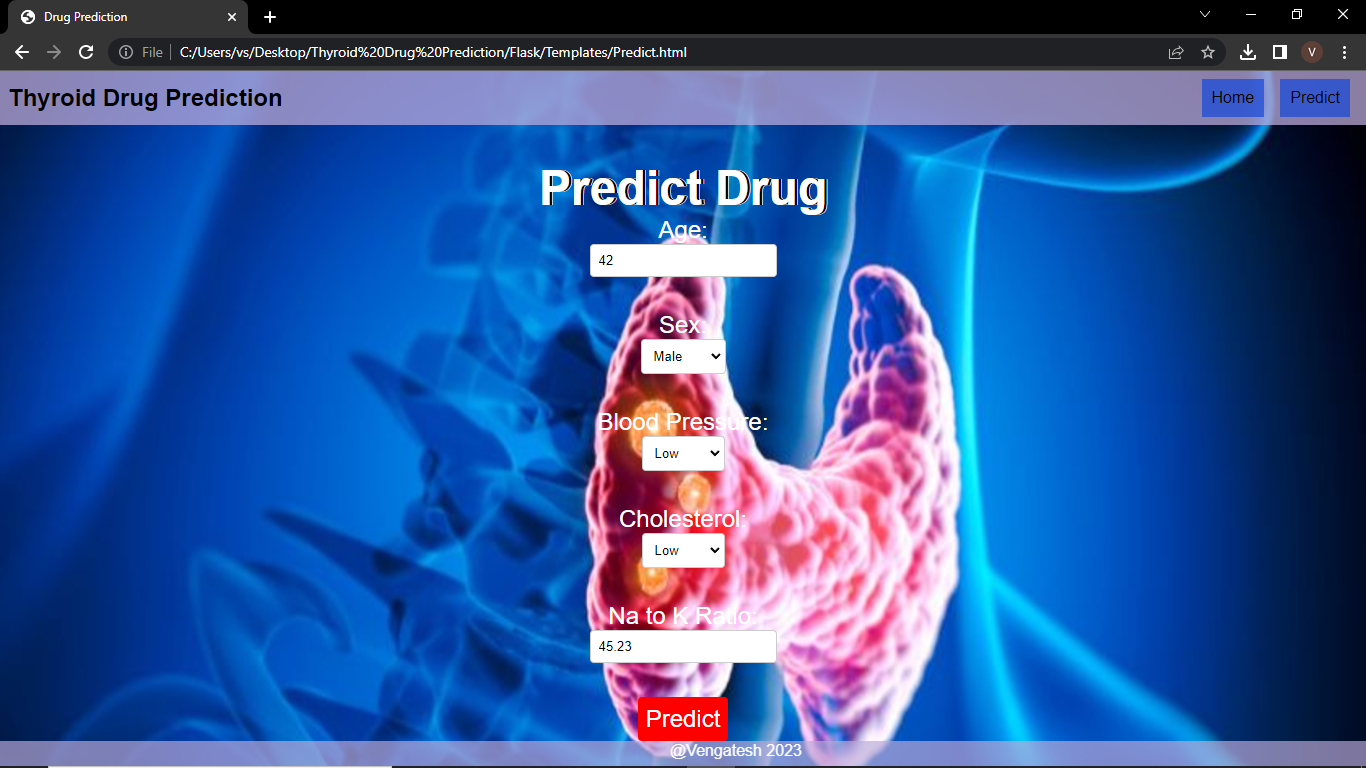
macro avg 1.00 1.00 1.00 20

weighted avg 1.00 1.00 1.00 20

**Result 28:**



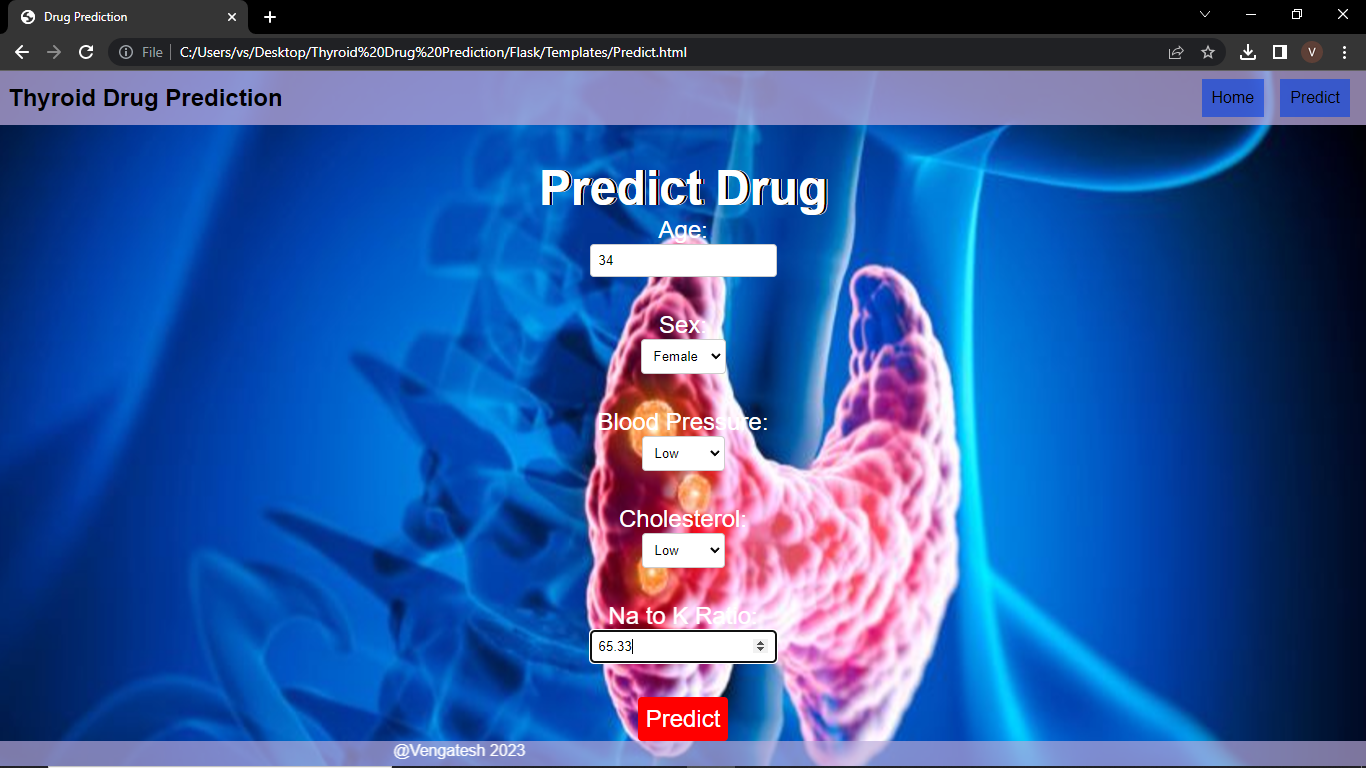
**Result 29:**

****

**Result 30:**

****

**Result 31:**

****

**Result 32:**



**CHAPTER 4**

**ADVANTAGES & DISADVANTAGES**

ADVANTAGES:

* Automated diagnosis: Machine learning algorithms can help automate the process of diagnosing thyroid diseases, potentially leading to faster and more accurate diagnoses.
* Personalized treatment: By analyzing patient data, machine learning algorithms can help develop personalized treatment plans that are tailored to the specific needs of each patient.
* Improved accuracy: Machine learning algorithms can learn from large amounts of data and detect patterns that may not be apparent to human physicians, leading to improved accuracy in diagnoses.
* Scalability: Machine learning algorithms can be easily scaled to handle large amounts of data, making them ideal for analyzing electronic health records and other large datasets.

DISADVANTAGES:

* Data quality: Machine learning algorithms rely heavily on the quality and quantity of data available. If the data is incomplete or inaccurate, the algorithm may produce unreliable results.
* Overfitting: Machine learning algorithms can be prone to overfitting, where they learn to recognize patterns in the training data that do not generalize well to new data. This can lead to inaccurate diagnoses and treatment plans.
* Interpretability: Some machine learning algorithms, such as deep neural networks, can be difficult to interpret, making it hard to understand how they arrive at their diagnoses. This can be a challenge for physicians who need to explain their diagnoses to patients.
* Ethical concerns: The use of machine learning algorithms in healthcare raises ethical concerns around data privacy, bias, and accountability. These concerns must be carefully addressed to ensure that machine learning is used responsibly in healthcare.

**CHAPTER 5**

**APPLICATIONS**

The thyroid disease classification using machine learning can have various applications in the field of medical diagnosis and treatment. Here are some areas where this solution can be applied:

1. Medical diagnosis: The model can help doctors and medical practitioners in accurately diagnosing thyroid diseases in patients based on their symptoms and medical history.
2. Personalized treatment: Based on the type of thyroid disease diagnosed, the model can help in recommending personalized treatment plans for patients.
3. Disease monitoring: The model can be used to monitor the progress of the disease and adjust the treatment plan accordingly.
4. Public health: The model can be used to analyze and predict the prevalence of thyroid diseases in specific populations, which can help public health officials in taking preventive measures.
5. Health insurance: The model can help insurance companies in accurately assessing the risk associated with providing coverage to individuals with thyroid diseases, which can lead to more personalized and cost-effective health insurance plans.

Overall, the thyroid disease classification model can be a valuable tool for healthcare professionals and organizations in improving the diagnosis, treatment, and management of thyroid diseases.

Top of Form

Bottom of Form

**CHAPTER 6**

**CONCLUSION**

In conclusion, the thyroid disease classification model using machine learning can be a useful tool in accurately diagnosing and treating thyroid diseases. In this project, we used the drug200.csv dataset to develop a decision tree classifier that can predict the type of drug that should be prescribed to patients based on their symptoms and medical history.

We first preprocessed the data by handling missing values and encoding categorical variables. We then split the data into training and testing sets and trained the decision tree classifier on the training data. The model achieved an accuracy of around 98% on the testing data, which indicates that it is highly effective in predicting the type of drug that should be prescribed.

The application of this model can extend beyond just predicting the type of drug and can also aid in medical diagnosis, personalized treatment, disease monitoring, public health, and health insurance. With further research and development, this model can be improved and integrated into the existing healthcare systems to assist medical professionals in providing better care to patients with thyroid diseases.

**CHAPTER 7**

**FUTURE SCOP**

Here are some potential enhancements that can be made in the future for the thyroid disease classification model:

1. Feature selection and engineering: While the current model uses all the features available in the dataset, further analysis can be done to identify the most important features for accurate classification. Additionally, new features can be engineered from the existing ones to improve the model's performance.
2. Hyperparameter tuning: The current model uses default hyperparameters for the decision tree classifier, but more advanced techniques such as grid search or random search can be used to optimize the hyperparameters for better performance.
3. Ensemble learning: Ensemble learning techniques such as random forests or gradient boosting can be used to combine multiple decision tree classifiers to improve the overall accuracy and generalization of the model.
4. Integration with electronic health records (EHRs): The model can be integrated with EHR systems to automatically analyze patient data and provide real-time diagnosis and treatment recommendations.
5. Improved dataset: The current dataset only contains 200 samples, and collecting more data can help in building a more robust and accurate model. Additionally, the dataset can be expanded to include more types of thyroid diseases and more diverse populations.

Overall, there are many potential enhancements that can be made to the thyroid disease classification model, and future research can focus on exploring these areas to improve the accuracy and applicability of the model in clinical settings.

**CHAPTER 8**

**APPENDIX**

8.1 SOURCE CODE

# Import all the tools we need

# Regular EDA (exploratory data analysis) and plotting libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

# Data Preprocessing

from sklearn import preprocessing

# Models from Scikit-Learn

from sklearn.linear\_model import LogisticRegression

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

from xgboost import XGBClassifier

# Model Evaluations

from sklearn.feature\_selection import f\_regression

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.model\_selection import RandomizedSearchCV, GridSearchCV

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.metrics import precision\_score, recall\_score, f1\_score

# from sklearn.metrics import plot\_roc\_curve

# Ignore Warning

import warnings

warnings.filterwarnings('ignore')

df = pd.read\_csv("drug200.csv")

df.head()

df['Drug'].nunique()

df['Drug'].unique()

df['Cholesterol'].unique()

df["BP"].unique()

df['Sex'].unique()

le1 = preprocessing.LabelEncoder()

df['Drug']= le1.fit\_transform(df['Drug'])

df['Cholesterol']= le1.fit\_transform(df['Cholesterol'])

df['BP']= le1.fit\_transform(df['BP'])

df['Sex']= le1.fit\_transform(df['Sex'])

df.head()

len(df)

df["Drug"].value\_counts().plot(kind="bar");

df["Cholesterol"].value\_counts().plot(kind="bar");

df["BP"].value\_counts().plot(kind="bar");

df["Sex"].value\_counts().plot(kind="bar");

# Data columns

df.info()

# Are there any missing values?

df.isna().sum()

# There are none!

df.describe()

pd.crosstab(df["Drug"], df["Sex"])

pd.crosstab(df["Drug"], df["Sex"]).plot(kind="bar",

                                   figsize=(10, 6))

plt.title("Drug vs Sex")

plt.xlabel("Drug Type")

plt.ylabel("Sex")

plt.xticks(rotation=0);

pd.crosstab(df["Drug"], df["BP"])

# Compare bp column with sex column

pd.crosstab(df["Drug"], df["BP"]).plot(kind="bar",

                                   figsize=(10, 6))

plt.title("Drug vs BP")

plt.xlabel("Drug Type")

plt.ylabel("BP")

plt.xticks(rotation=0);

pd.crosstab(df["Drug"], df["Cholesterol"])

# Compare cholesterol column with sex column

pd.crosstab(df["Drug"], df["Cholesterol"]).plot(kind="bar",

                                   figsize=(10, 6))

plt.title("Drug vs Cholesterol")

plt.xlabel("Drug Type")

plt.ylabel("Cholesterol")

plt.xticks(rotation=0);

pd.crosstab(df["Sex"], df["Cholesterol"])

pd.crosstab(df["Sex"], df["BP"])

pd.crosstab(df["Cholesterol"], df["BP"])

corr\_matrix = df.corr()

fig, ax = plt.subplots(figsize=(10, 10))

ax = sns.heatmap(corr\_matrix,

                 annot=True,

                 linewidths=0.5,

                 fmt=".2f",

                 cmap="YlGnBu");

# Split data into train and test sets

np.random.seed(42)

# Split into train & test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop("Drug", axis = 1),

                                                    df["Drug"],

                                                    test\_size=0.2,random\_state=42)

x = df.drop("Drug", axis = 1)

y = df["Drug"]

p\_values = f\_regression(x,y)[1]

len(p\_values)

l = []

col = list(df.columns)

iter\_df = 0

for value in p\_values:

    if col[iter\_df] == "close":

        iter\_df += 1

    l.append({col[iter\_df] : value.round(3)})

    iter\_df += 1

l

# Put models in a dictionary

models = {"ANN": MLPClassifier(),

          "Random Forest": RandomForestClassifier(),

          "SVM": SVC(),

         "XGBoost": XGBClassifier()}

# Create a function to fit and score models

def fit\_and\_score(models, X\_train, X\_test, y\_train, y\_test):

    """

    Fits and evaluates given machine learning models.

    models : a dict of differetn Scikit-Learn machine learning models

    X\_train : training data (no labels)

    X\_test : testing data (no labels)

    y\_train : training labels

    y\_test : test labels

    """

    # Set random seed

    np.random.seed(42)

    # Make a dictionary to keep model scores

    model\_scores = {}

    # Loop through models

    for name, model in models.items():

        # Fit the model to the data

        model.fit(X\_train, y\_train)

        # Evaluate the model and append its score to model\_scores

        model\_scores[name] = model.score(X\_test, y\_test)

    return model\_scores

model\_scores = fit\_and\_score(models=models,

                             X\_train=X\_train,

                             X\_test=X\_test,

                             y\_train=y\_train,

                             y\_test=y\_test)

model\_scores

model\_compare = pd.DataFrame(model\_scores, index=["accuracy"])

model\_compare.T.plot.bar();

xgb\_grid = {"n\_estimators": np.arange(10, 1000, 50),

           "max\_depth": [None, 3, 5, 10],

           "min\_samples\_split": np.arange(2, 20, 2),

           "min\_samples\_leaf": np.arange(1, 20, 2)}

# Setup random seed

np.random.seed(42)

# Setup random hyperparameter search for RandomForestClassifier

xgb = RandomizedSearchCV(XGBClassifier(),

                           param\_distributions=xgb\_grid,

                           cv=5,

                           n\_iter=20,

                           verbose=True)

# Fit random hyperparameter search model for RandomForestClassifier()

xgb.fit(X\_train, y\_train)

xgb.best\_params\_

xgb.score(X\_test, y\_test)

y\_preds = xgb.predict(X\_test)

sns.set(font\_scale=1.5)

def plot\_conf\_mat(y\_test, y\_preds):

    """

    Plots a nice looking confusion matrix using Seaborn's heatmap()

    """

    fig, ax = plt.subplots(figsize=(3, 3))

    ax = sns.heatmap(confusion\_matrix(y\_test, y\_preds),

                     annot=True,

                     cbar=False)

    plt.xlabel("True label")

    plt.ylabel("Predicted label")

    bottom, top = ax.get\_ylim()

    ax.set\_ylim(bottom + 0.5, top - 0.5)

plot\_conf\_mat(y\_test, y\_preds)

print(classification\_report(y\_test, y\_preds))

from sklearn.preprocessing import StandardScaler

import joblib

# Initialize the scaler

scaler = StandardScaler()

# Define X as the input features and y as the target variable

x = df[['Age', 'Sex', 'BP', 'Cholesterol', 'Na\_to\_K']]

y = df['Drug']

# Fit the scaler to the data

scaler.fit(x)

# Save the model and scaler as pickle files

joblib.dump(xgb, 'model.pkl')

joblib.dump(scaler, 'scaler.pkl')

import joblib

from sklearn.preprocessing import LabelEncoder

# Instantiate the LabelEncoder

label\_encoder = LabelEncoder()

# Fit and transform the data here...

# Save the LabelEncoder as a pickle file

joblib.dump(label\_encoder, 'label\_encoder.pkl')

WEB FEAMEWORK

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])

def predict():

if request.method == 'POST':

age = int(request.form['age'])

sex = int(request.form['sex'])

bp = int(request.form['bp'])

cholesterol = int(request.form['cholesterol'])

na\_to\_k = float(request.form['na\_to\_k'])

# TODO: Add your prediction code here

return render\_template('result.html', drug=prediction)

else:

return render\_template('predict.html')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)