EUDAIMONIA ENGINE: MACHINE LEARNING DELVING INTO HAPPINESS CLASSIFICATION

AN INDUSTRY ORIENTED MINI REPORT

Submitted to

JAWAHARLAL NEHRU TECNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

ALLA LAHARI	**21UK1A05G1**
THOTA AJAY	**21UK1A05H6**
RASURI RAKESH	**21UK1A05H9**
JAKKU NITHIN	**21UK1A05E9**

Under the guidance of

Dr. NAVEEN KUMAR RANGARAJU

(Hod of CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE

Affiliated to JNTUH, HYDERABAD

BOLLIKUNTA, WARANGAL (T.S)-506005

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

VAAGDEVI ENGINEERING COLLEGE(WARANGAL)

CERTIFICATE OF COMPLETION

INDUSTRY ORIENTED MINI PROJECT

This is to certify that the UG Project Phase-1 entitled "EUDAIMONIA ENGINE:MACHINE LEARNING DELVING INTO HAPPINESS CLASSIFICATION" is being submitted by ALLA LAHARI (21UK1A05G1),THOTA AJAY(21UK1A05H6),RASURI RAKESH(21UK1A05H9),JAKKU NITHIN(21UK1A05E9) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering to Jawaharlal Nehru Technological University Hyderabad during the academic year 2023-2024.

Project Guide	**HOD**
B.Venkanna	**Dr. Naveen Kumar**

(Assistant Professor) (Professor)

External

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Dr.P.PRASAD RAO,** Principal, Vaagdevi Engineering College for making us available all the required assistance and for his support and inspiration to carry out this UG Project Phase-1 in the institute.

We extend our heartfelt thanks to our HOD **DR. NAVEEN KUMAR RANGARAJU**, Head of department of CSE, Vaagdevi Engineering College for providing us necessary infrastructure and thereby giving us freedom to carry out the UG Project Phase-1.

We express heartfelt thanks to the guide, **B. VENKANNA,** assistant professor of CSE for her constant guidance for completion of this UG Project Phase-1.

Finally, we express our sincere thanks and gratitude to my family members and friends for their encouragement and outpouring their knowledge and experience throughout the thesis.

ALLA LAHARI	**(21UK1A05G1)**
THOTA AJAY	**(21UK1A05H6)**
RASURI RAKESH	**(21UK1A05H9)**
JAKKU NITHIN	**(21UK1A05E9)**

ABSTRACT

Understanding the factors that contribute to human happiness is a crucial endeavor in the fields of psychology, sociology, and public policy. "Happiness Classification" is a machine learning-based application designed to classify and analyze the determinants of well-being. By leveraging a comprehensive dataset encompassing various socio-economic, environmental, and personal factors, the application aims to provide insights into the key drivers of happiness.

Happiness Classification serves as a valuable tool for researchers, policymakers, and individuals interested in understanding and improving well-being. By providing a data-driven approach to happiness analysis, the application aims to support informed decision-making and the development of interventions that promote a higher quality of life.

INDEX

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW:

Happiness Classification is a pioneering application designed to explore and understand the multifaceted determinants of human well-being. Utilizing machine learning techniques, this application provides a systematic approach to classifying and analyzing factors that contribute to happiness. By examining a broad spectrum of socio-economic, environmental, and personal variables, Happiness Classification offers valuable insights into what drives happiness and how it can be enhanced.

Happiness Classification represents a significant advancement in the field of well-being analysis. By combining machine learning with comprehensive data analysis, the application offers a powerful tool for understanding and promoting happiness. Through its user-friendly interface and real-time predictive capabilities, Happiness Classification empowers users to make informed decisions that contribute to a happier and more fulfilling life.

- The core of Happiness Classification is a robust machine learning model trained on extensive datasets.
- Advanced algorithms, such as Decision Trees or Support Vector Machines, are employed to classify individuals' happiness levels based on the input data.
- The model provides a probabilistic assessment, indicating the likelihood of different happiness outcomes.

1.2 PURPOSE:

The purpose of this project is to predict the happiness classification on machine learning model classification using the decision tree model by this we can be capable to predict the happiness classification.

CHAPTER 2:
LITERATURE SURVEY

2.1 EXISTING PROBLEM

Understanding the complex and multifaceted nature of human happiness remains a significant challenge for researchers, policymakers, and individuals alike. Traditional approaches to studying well-being often rely on subjective surveys and self-reported data, which can be limited by biases and inconsistencies. Additionally, the dynamic interplay between various socio-economic, environmental, and personal factors that influence happiness is not fully captured by conventional analytical methods. This lack of comprehensive, data-driven insights hampers the ability to identify and address the root causes of unhappiness effectively. Policymakers struggle to design interventions that genuinely enhance quality of life, while individuals find it difficult to pinpoint actionable steps to improve their well-being. Consequently, there is a pressing need for innovative tools that can provide a deeper, more accurate understanding of the determinants of happiness, enabling more informed decision-making at both the individual and societal levels.

2.2 PROPOSED SOLUTION:

we will prepare the data using Google colab notebook and we use various models to predict the output. machine learning models are used very useful in predicting outcomes for large amount datasets. We use decision tree model machine learning algorithm to predict the happiness classification.

CHAPTER 3
THEORETICAL ANALYSIS

3.1 BLOCK DIAGRAM

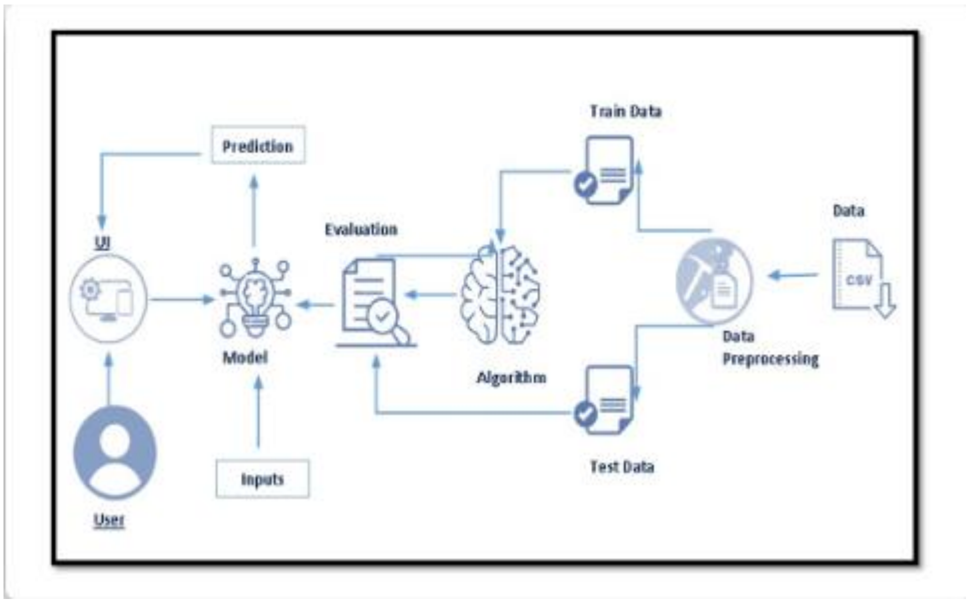


Fig 01: Flow chart

3.2 SOFTWARE & HARDWARE DESIGNING:

Software Requirements

REQUIREMENTS	**SPECIFICATIONS**
Anaconda Navigator	You must have anaconda installed in your device prior to begin.
➢ Anaconda prompt, google colab, Flask, spyder ➢ Frame work .	➢ One should have anaconda prompt and google colab. ➢ One should install flask framework through Anaconda prompt for running their web application. ➢ We need to build the mode; using JUPYTER notebook with all the imported packages.
Web browser	For all Web browsers, the following must be enabled: ● Cookies ● Java script

Hardware Requirements:

REQUIREMENTS	**SPECIFICATIONS**
Operating system	➢ Microsoft windows ➢ Unix ➢ Linux
Processing	Minimum: 4 CPU cores for one user. For each deployment, a sizing exercise is highly recommended.
RAM	Minimum 8 GB.
Operating system specifications	File descriptor limit set to 8192 on UNIX and Linux
Disk space	A minimum of 7 GB of free space is required to install the software.

CHAPTER 4
EXPERIMENTAL ANALYSIS

Analysis or the investigation made while working on the solution:

While working on the solution we investigated what Happiness Classification is, visualizing and analyzing the data, data pre-processing, Machine Learning service, model building. The key role on investigation is collection of data sets.

DATA PRE-PROCESSING:

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness, so we need to clean the dataset properly to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling duplicates data
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

DATA SET COLLECTION:

- ➔ Kaggle.com
- ➔ Machine learning repository

The data set contains thirteen classes:

1.info_avail

2.housecost

3.coolquality

4.policetrust

5.streetquality

6.events

CHAPTER 5:
FLOW CHART

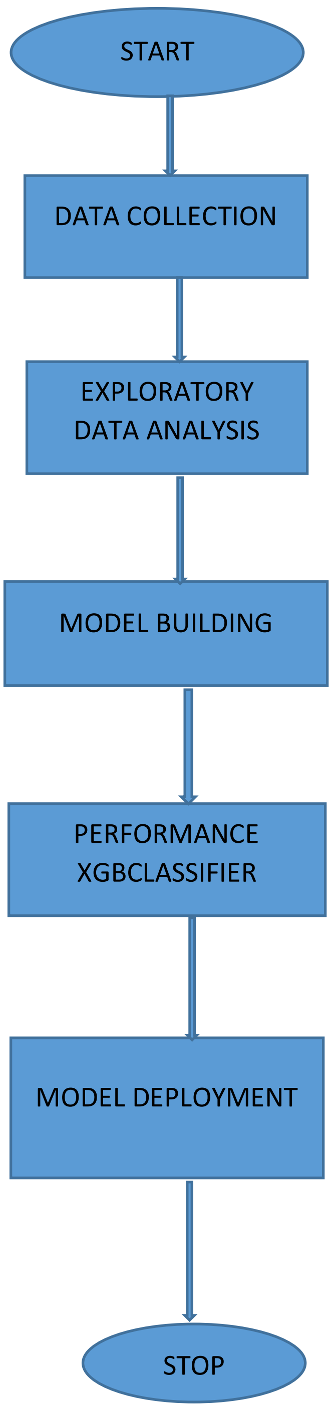
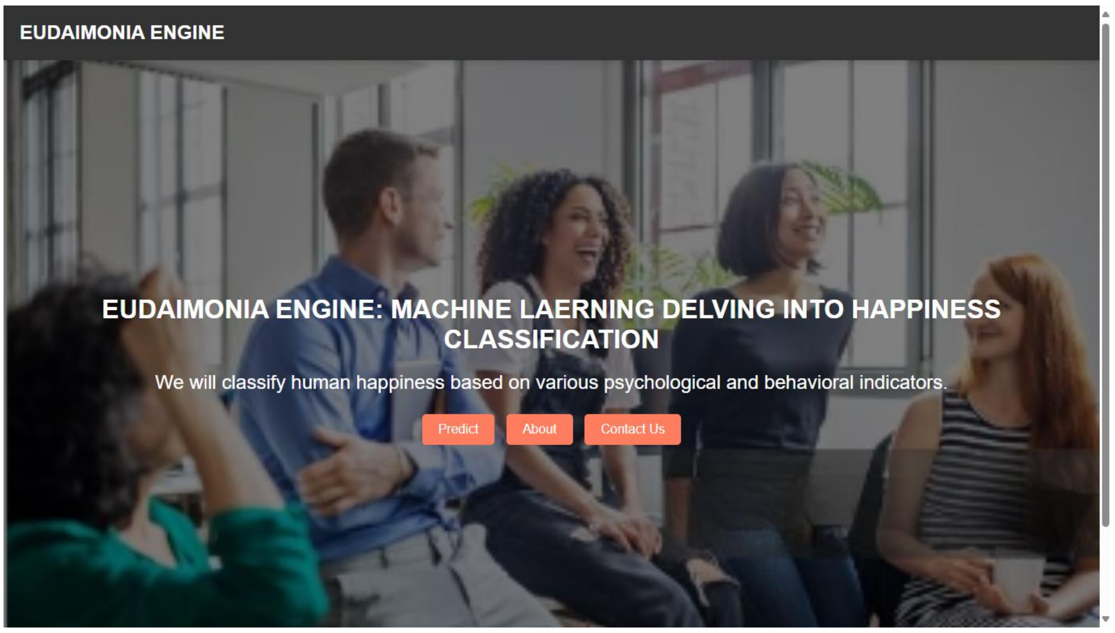


Fig 02: Flow chart of the project

CHAPTER 6

RESULT

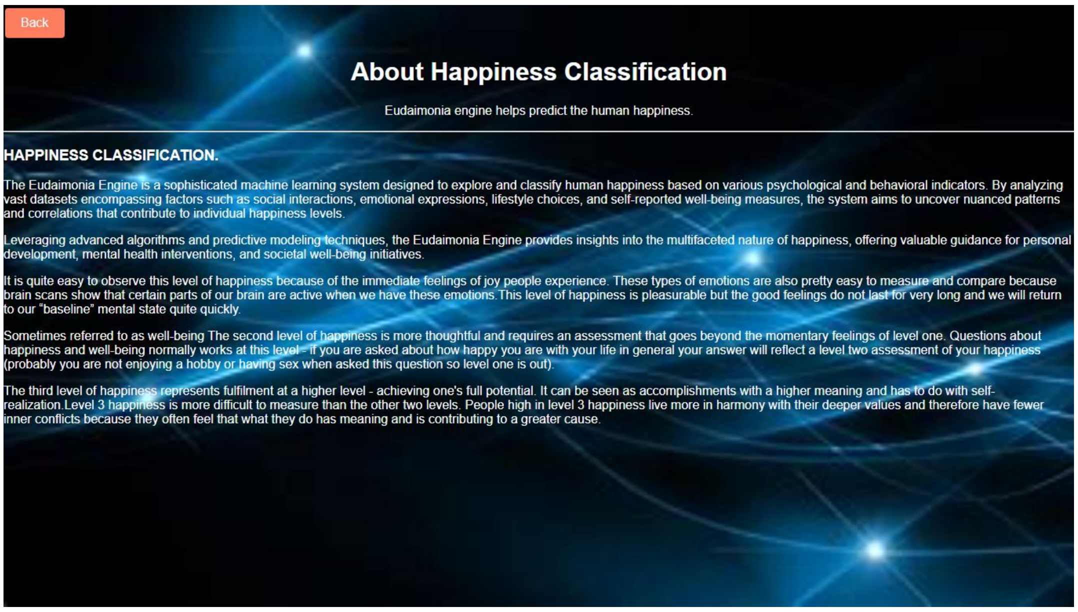
The home page will be shown as:


EUDAIMONIA ENGINE

EUDAIMONIA ENGINE: MACHINE LAERNING DELVING INTO HAPPINESS CLASSIFICATION

We will classify human happiness based on various psychological and behavioral indicators.

The about page about Happiness Classification:



About Happiness Classification

Eudaimonia engine helps predict the human happiness.

HAPPINESS CLASSIFICATION.

The Eudaimonia Engine is a sophisticated machine learning system designed to explore and classify human happiness based on various psychological and behavioral indicators. By analyzing vast datasets encompassing factors such as social interactions, emotional expressions, lifestyle choices, and self-reported well-being measures, the system aims to uncover nuanced patterns and correlations that contribute to individual happiness levels.

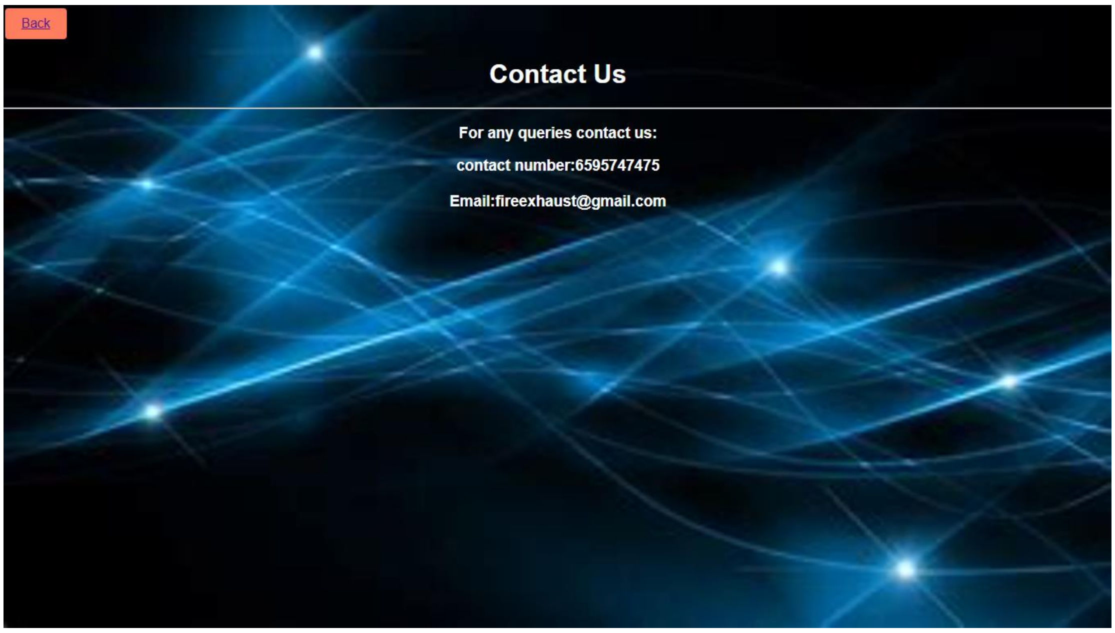
Leveraging advanced algorithms and predictive modeling techniques, the Eudaimonia Engine provides insights into the multifaceted nature of happiness, offering valuable guidance for personal development, mental health interventions, and societal well-being initiatives.

It is quite easy to observe this level of happiness because of the immediate feelings of joy people experience. These types of emotions are also pretty easy to measure and compare because brain scans show that certain parts of our brain are active when we have these emotions.This level of happiness is pleasurable but the good feelings do not last for very long and we will return to our "baseline" mental state quite quickly.

Sometimes referred to as well-being The second level of happiness is more thoughtful and requires an assessment that goes beyond the momentary feelings of level one. Questions about happiness and well-being normally works at this level - if you are asked about how happy you are with your life in general your answer will reflect a level two assessment of your happiness (probably you are not enjoying a hobby or having sex when asked this question so level one is out).

The third level of happiness represents fulfilment at a higher level - achieving one's full potential. It can be seen as accomplishments with a higher meaning and has to do with self-realization.Level 3 happiness is more difficult to measure than the other two levels. People high in level 3 happiness live more in harmony with their deeper values and therefore have fewer inner conflicts because they often feel that what they do has meaning and is contributing to a greater cause.

The contact us page for Happiness Classification:



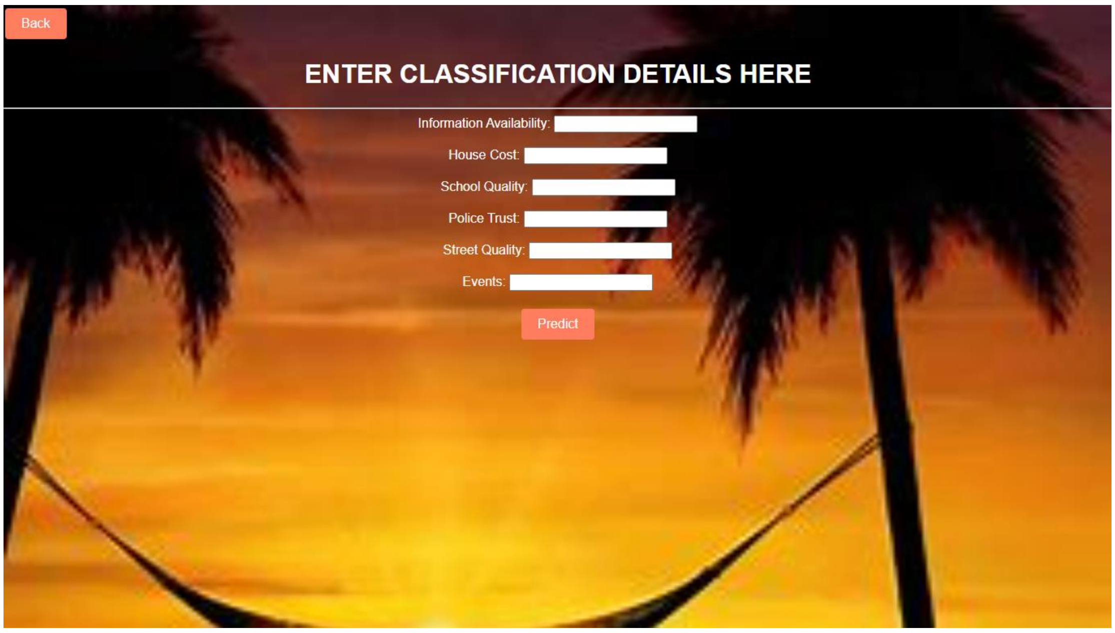
Contact Us

For any queries contact us:

contact number:6595747475

Email:fireexhaust@gmail.com

The prediction page for Happiness Classification:



ENTER CLASSIFICATION DETAILS HERE

Information Availability:

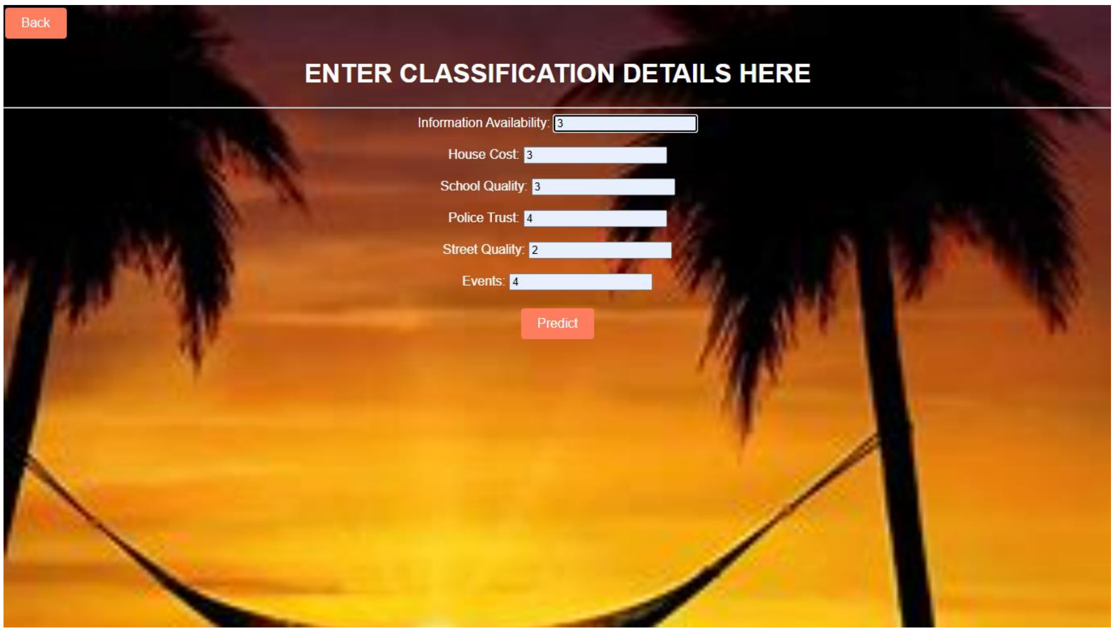
House Cost:

School Quality:

Police Trust:

Street Quality:

Events:



ENTER CLASSIFICATION DETAILS HERE

Information Availability: 3

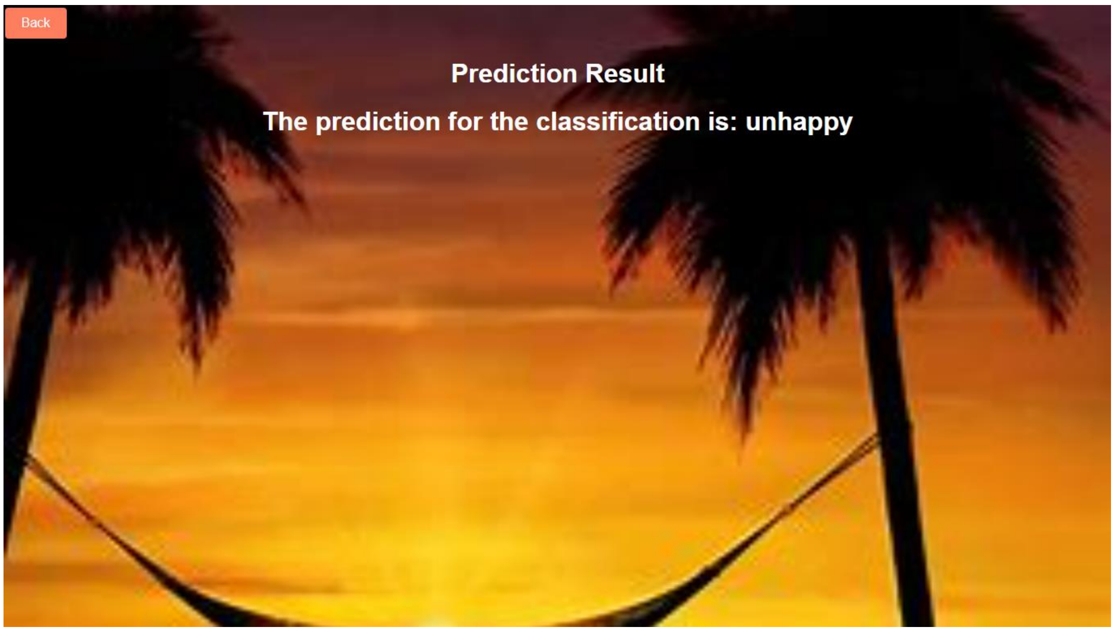
House Cost: 3

School Quality: 3

Police Trust: 4

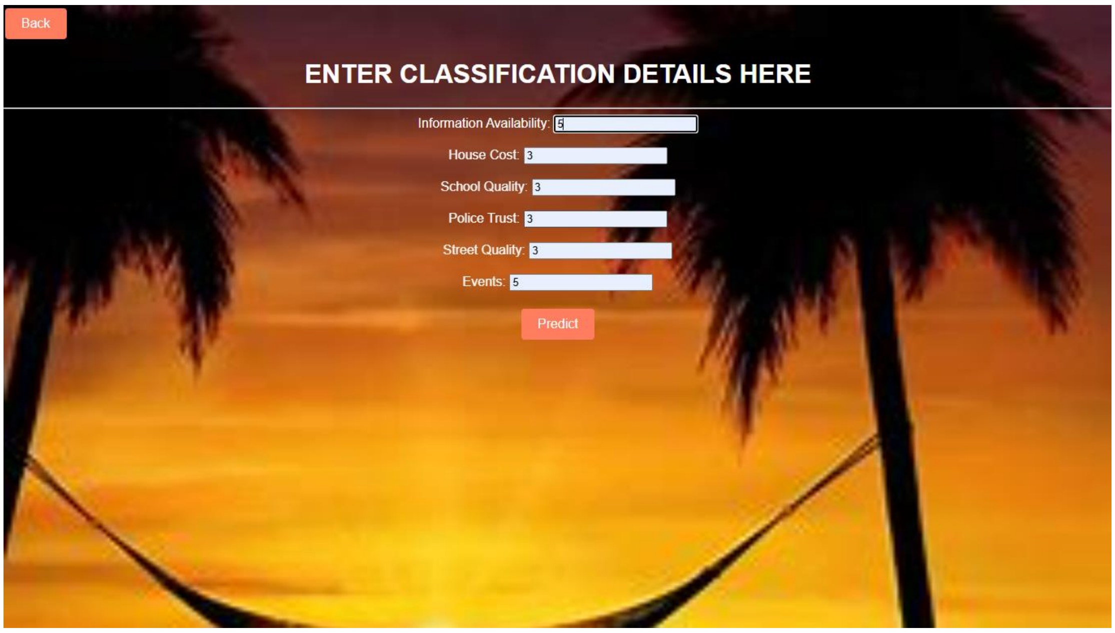
Street Quality: 2

Events: 4



Prediction Result

The prediction for the classification is: unhappy



ENTER CLASSIFICATION DETAILS HERE

Information Availability: 5

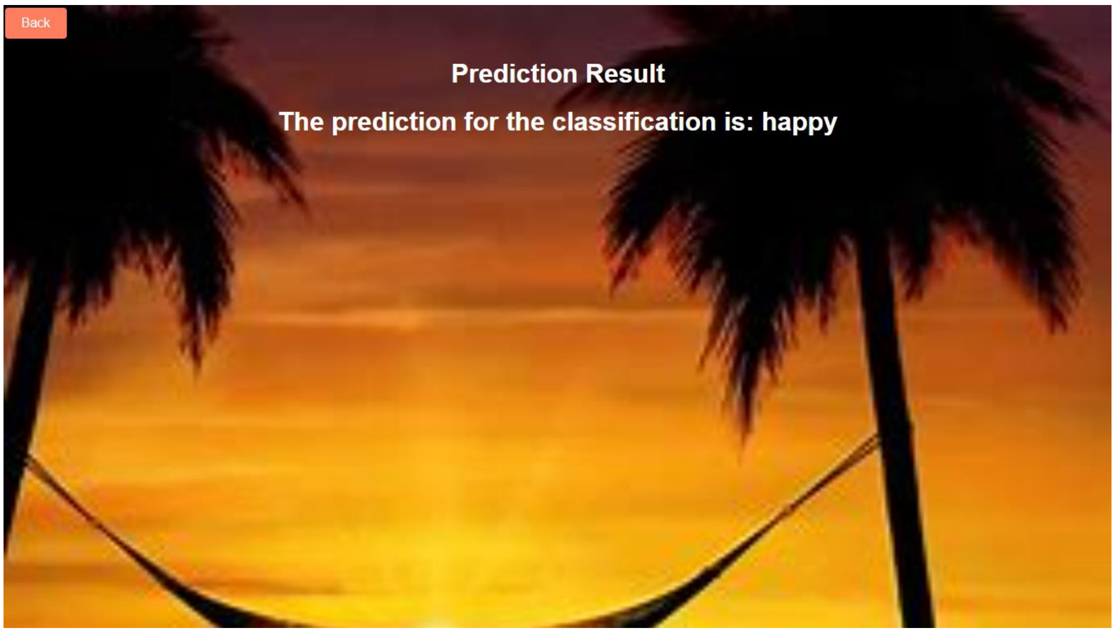
House Cost: 3

School Quality: 3

Police Trust: 3

Street Quality: 3

Events: 5



Prediction Result

The prediction for the classification is: happy

CHAPTER 7
ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- Data-Driven Insights
- Comprehensive Analysis
- Predictive Capabilities
- Real-Time Feedback
- Policy Formulation
- User-Friendly Interface

DISADVANTAGES:

- Data Privacy Concerns
- Model Limitations
- Complexity of Happiness
- Overreliance on Technology
- Implementation Challenges
- Cost and Resources

CHAPTER 8
APPLICATIONS

The areas where this solution can be used:

- ➢ Research and academic studies
- ➢ Policy making and public health

CHAPTER 9
CONCLUSION

FROM THIS PROJECT WE HAVE CONCLUDED THAT:

-→ We have Known fundamental concepts and techniques used for machine learning.

→ Gain a broad understanding about data.

→ Have knowledge on pre-processing the data/transformation techniques and some visualization techniques.

CHAPTER 10
FUTURE SCOPE

Enhancements that can be made in the future

→ Know fundamental concepts and techniques used for machine learning.

→ Gain a broad understanding about data.

→ Have knowledge on pre-processing the data/transformation techniques and some visualization concepts.

CHAPTER 11

BIBILOGRAPHY

References of previous works or websites visited/books referred for analysis about the project, previous solution findings and previous HAPPINESS CLASSIFICATION documents.

CHAPTER 12

CODE SNIPPETS

MODEL BUILDING:

1)Dataset
2)Jupyter notebook and Spyder Application Building

1. HTML file (home file, about file, predict file, submit file)
1. CSS file
2. Models in pickle format

APP.PY:

```
from flask import Flask, render_template, request, redirect, url_for
import joblib

app = Flask(__name__)

# Load the model
model = joblib.load('happy.joblib')

@app.route('/')
def home():
   return render_template('home.html')

@app.route('/about')
def about():
   return render_template('about.html')
```

```
@app.route('/contactus')
def contactus():
    return render_template('contactus.html')

@app.route('/predict')
def predict():
    return render_template('predict.html')

@app.route('/submit', methods=['POST'])
def submit():
    # Retrieve form data
    infoavail = float(request.form['infoavail'])
    housecost = float(request.form['housecost'])
    schoolquality = float(request.form['schoolquality'])
    policetrust = float(request.form['policetrust'])
    streetquality = float(request.form['streetquality'])
    events = float(request.form['events'])

    # Create a feature array for prediction
    features = [[infoavail, housecost, schoolquality, policetrust, streetquality, events]]

    # Predict happiness level
    prediction = model.predict(features)
    predict_value = prediction[0]

    # Convert prediction to "happy" or "unhappy"
    result = "happy" if predict_value == 1 else "unhappy"

    return render_template('submit.html', predict=result)

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

HOME.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <link rel="stylesheet" href="{{ url_for("static",
filename='style.css') }}">

   <title>Home Page</title>

</head>
<body class='h'>
   <div class="navbar">
      <div class="logo">EUDAIMONIA ENGINE</div>
   </div>

   <div class="hero-section">
      <div class="hero-content">
         <h1>EUDAIMONIA ENGINE: MACHINE LAERNING
DELVING INTO HAPPINESS CLASSIFICATION</h1>
         <p>We will classify human happiness based on various
psychological and behavioral indicators.</p>
         <div class="button-group">
            <button onclick="window.location.href='
http://127.0.0.1:5000/predict'">Predict</button>
            <button onclick="window.location.href='
http://127.0.0.1:5000/about'">About</button>
            <button onclick="window.location.href='
http://127.0.0.1:5000/contactus'">Contact Us</button>
         </div>
      </div>
   </div>

   <div class="footer">
      &copy; 2024 MyWebsite. All rights reserved.
   </div>
</body>
</html>
```

ABOUT.HTML:

```
<!DOCTYPE html>

<html lang="en">

<head>

   <title>About - eudaimonia engine</title>

   <meta name="viewport" content="width=device-width, initial-
scale=1.0">

   <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">

</head>

<body text="white" class="a">

<button class="ab" onclick="window.location.href='/'">Back</button>

   <h1 align="center">About Happiness Classification</h1>

   <p align="center">Eudaimonia engine helps predict the human
happiness.</p>

   <hr/>

      <h3>

         HAPPINESS CLASSIFICATION.

      </h3>

      <P>

         The Eudaimonia Engine is a sophisticated machine learning
system designed to explore and classify human happiness based on
various psychological and behavioral indicators. By analyzing vast
datasets encompassing factors such as social interactions, emotional
expressions, lifestyle choices, and self-reported well-being measures, the
system aims to uncover nuanced patterns and correlations that contribute
to individual happiness levels.

      </P>

      <p> Leveraging advanced algorithms and predictive modeling
techniques, the Eudaimonia Engine provides insights into the
```

multifaceted nature of happiness, offering valuable guidance for personal development, mental health interventions, and societal well-being initiatives.</p>

<P>It is quite easy to observe this level of happiness because of the immediate feelings of joy people experience. These types of emotions are also pretty easy to measure and compare because brain scans show that certain parts of our brain are active when we have these emotions.This level of happiness is pleasurable but the good feelings do not last for very long and we will return to our “baseline” mental state quite quickly.</P>

<p>Sometimes referred to as well-being The second level of happiness is more thoughtful and requires an assessment that goes beyond the momentary feelings of level one. Questions about happiness and well-being normally works at this level - if you are asked about how happy you are with your life in general your answer will reflect a level two assessment of your happiness (probably you are not enjoying a hobby or having sex when asked this question so level one is out).</p>

<p>The third level of happiness represents fulfilment at a higher level - achieving one's full potential. It can be seen as accomplishments with a higher meaning and has to do with self-realization.Level 3 happiness is more difficult to measure than the other two levels. People high in level 3 happiness live more in harmony with their deeper values and therefore have fewer inner conflicts because they often feel that what they do has meaning and is contributing to a greater cause.</p>

</body>

</html>

CONTACTUS.HTML:

```
<html>

<head>

<title>contact us</title>

<meta name="viewport" content="width=device-width, initial-
scale=1.0">

    <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">

</head>

<body text="white" class="abo">

<button class="co"><a href="/">Back</a></button>

<h1 align="center">Contact Us</h1>

<hr/>

<h3 align="center">For any queries contact us:</h3>

<h3 align="center">contact
number:6595747475<br><br>Email:fireexhaust@gmail.com

</h3>

</body>

</html>
```

PREDICT.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>STARTUP PROPHET</title>
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body text="#fff" class="i">
   <button class="pr" onclick="window.location.href='/'">Back</button>
   <div class="startup">
       <h1 align="center">ENTER CLASSIFICATION DETAILS
HERE</h1>
       <hr/>
       <form action="{{ url_for('submit') }}" method="POST"
align="center">
           <label for="infoavail">Information Availability:</label>
       <input type="text" id="infoavail" name="infoavail"><br><br>


       <label for="housecost">House Cost:</label>
       <input type="text" id="housecost" name="housecost"><br><br>


       <label for="schoolquality">School Quality:</label>
       <input type="text" id="schoolquality"
name="schoolquality"><br><br>
```

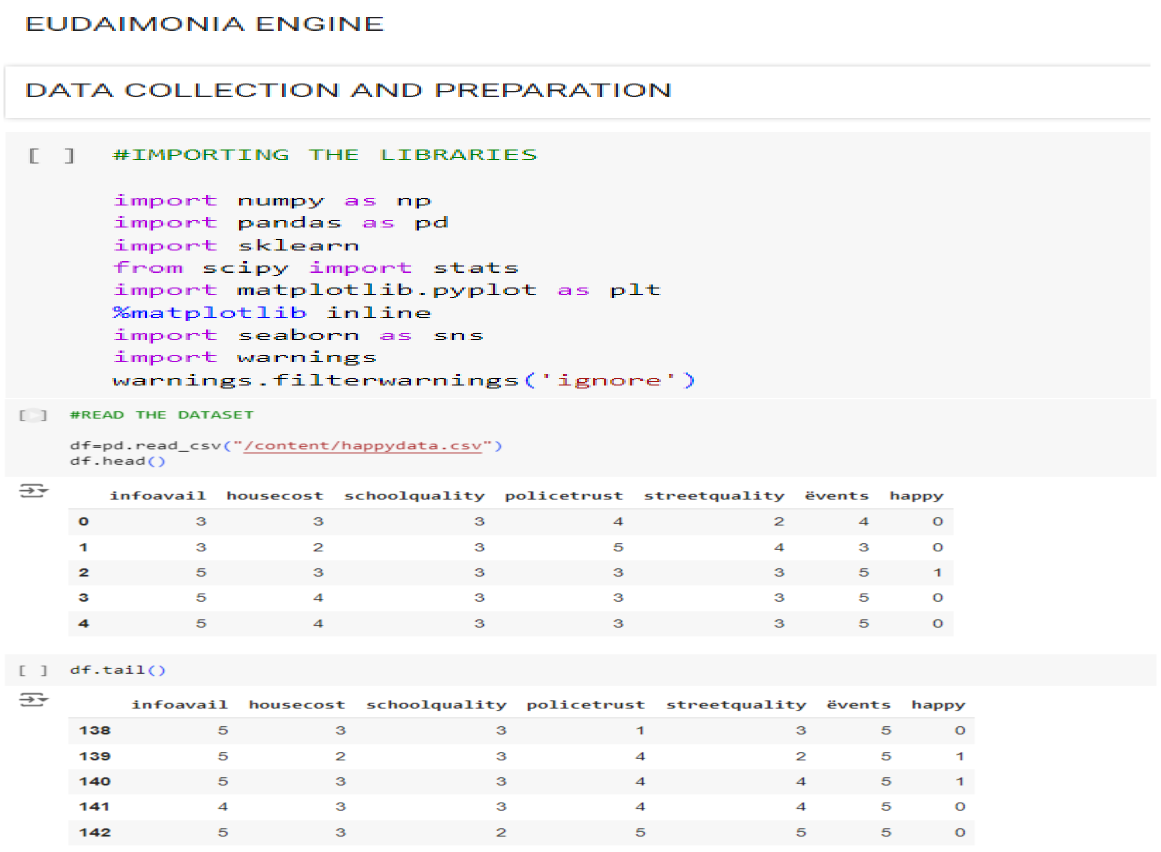
```
        <label for="policetrust">Police Trust:</label>
        <input type="text" id="policetrust" name="policetrust"><br><br>


        <label for="streetquality">Street Quality:</label>
        <input type="text" id="streetquality"
name="streetquality"><br><br>


        <label for="events">Events:</label>
        <input type="text" id="events" name="events"><br><br>
           <button class="pr" onclick="window.location.href='
http://127.0.0.1:5000/submit'" type="submit">Predict</button>
        </form>
    </div>
</body>
</html>
```

<u>SUBMIT.HTML:</u>

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Prediction Result - Startup Prophet</title>
   <meta name="viewport" content="width=device-width, initial-
scale=1.0">
   <link rel="stylesheet" href="{{ url_for('static',
filename='style.css') }}">
</head>
<body text="white" class="i">
   <button class="su" onclick="window.location.href='/'">Back</button>
   <h1 align="center">Prediction Result</h1>
   <h1 align="center" size="4">The prediction for the classification is:
{{ predict }}</h1>
</body>
</html>
```

CODE SNIPPETS

MODEL BUILDING



EUDAIMONIA ENGINE

DATA COLLECTION AND PREPARATION

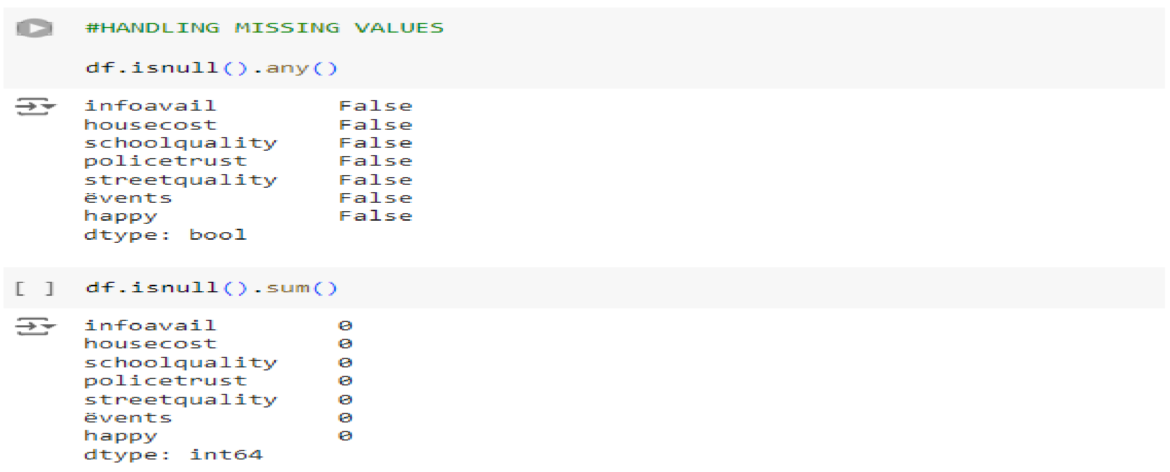
```
[ ]  #IMPORTING THE LIBRARIES

     import numpy as np
     import pandas as pd
     import sklearn
     from scipy import stats
     import matplotlib.pyplot as plt
     %matplotlib inline
     import seaborn as sns
     import warnings
     warnings.filterwarnings('ignore')
```

```
[ ]  #READ THE DATASET

     df=pd.read_csv("/content/happydata.csv")
     df.head()
```

```
     infoavail  housecost  schoolquality  policetrust  streetquality  ëvents  happy
0            3          3              3            4              2       4      0
1            3          2              3            5              4       3      0
2            5          3              3            3              3       5      1
3            5          4              3            3              3       5      0
4            5          4              3            3              3       5      0
```

```
[ ]  df.tail()
```

```
       infoavail  housecost  schoolquality  policetrust  streetquality  ëvents  happy
138            5          3              3            1              3       5      0
139            5          2              3            4              2       5      1
140            5          3              3            4              4       5      1
141            4          3              3            4              4       5      0
142            5          3              2            5              5       5      0
```

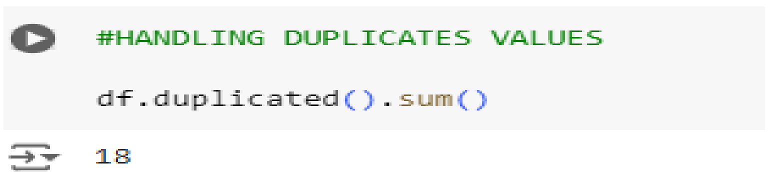


```
     #HANDLING MISSING VALUES

     df.isnull().any()
```

```
     infoavail        False
     housecost        False
     schoolquality    False
     policetrust      False
     streetquality    False
     ëvents           False
     happy            False
     dtype: bool
```

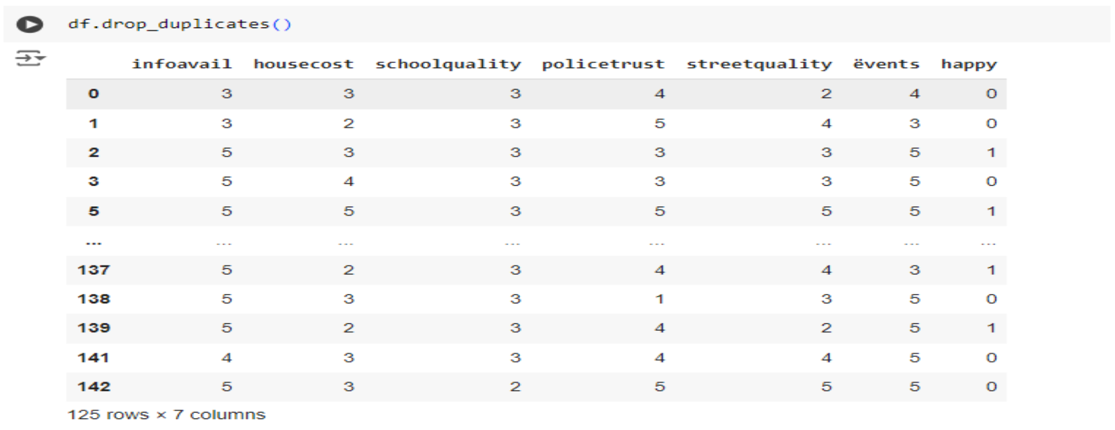
```
[ ]  df.isnull().sum()
```

```
     infoavail        0
     housecost        0
     schoolquality    0
     policetrust      0
     streetquality    0
     ëvents           0
     happy            0
     dtype: int64
```



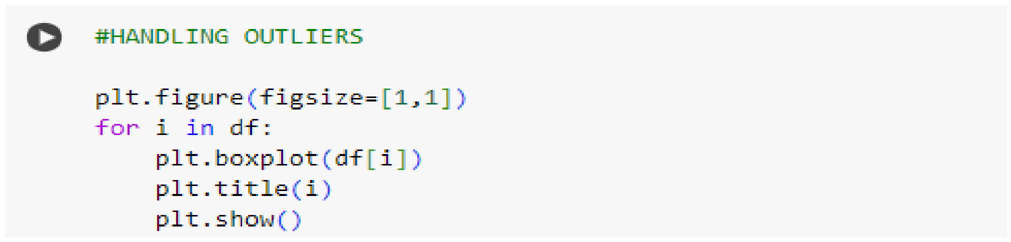
```
     #HANDLING DUPLICATES VALUES

     df.duplicated().sum()
```

```
     18
```

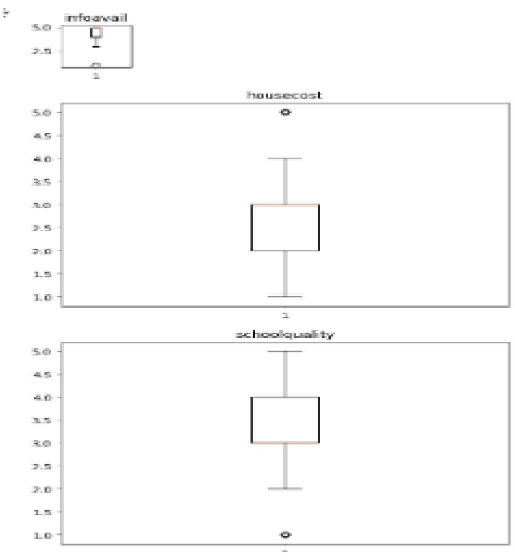


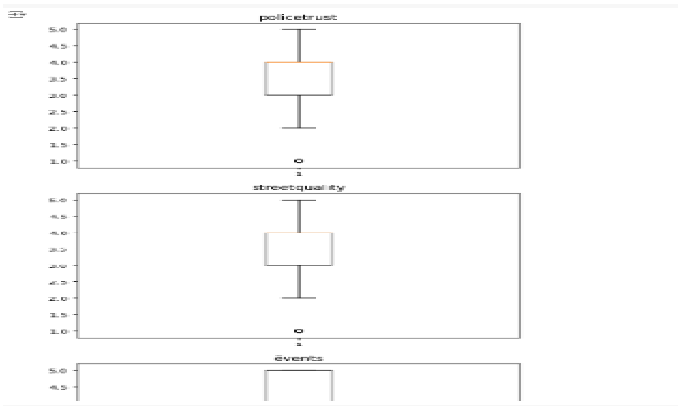
```
df.drop_duplicates()
```

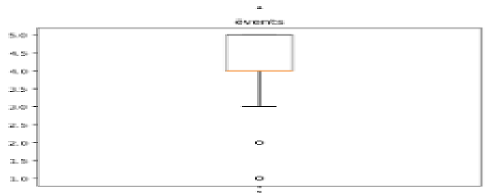
```
     infoavail  housecost  schoolquality  policetrust  streetquality  ëvents  happy
0            3          3              3            4              2       4      0
1            3          2              3            5              4       3      0
2            5          3              3            3              3       5      1
3            5          4              3            3              3       5      0
5            5          5              3            5              5       5      1
...        ...        ...            ...          ...            ...     ...    ...
137          5          2              3            4              4       3      1
138          5          3              3            1              3       5      0
139          5          2              3            4              2       5      1
141          4          3              3            4              4       5      0
142          5          3              2            5              5       5      0
125 rows × 7 columns
```

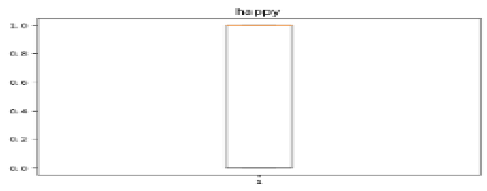


```
#HANDLING OUTLIERS

plt.figure(figsize=[1,1])
for i in df:
    plt.boxplot(df[i])
    plt.title(i)
    plt.show()
```



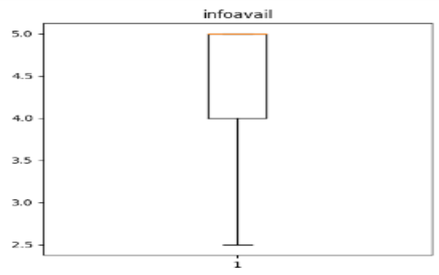


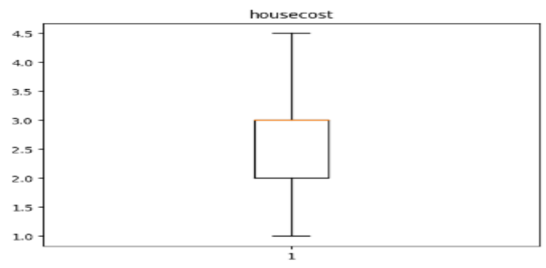


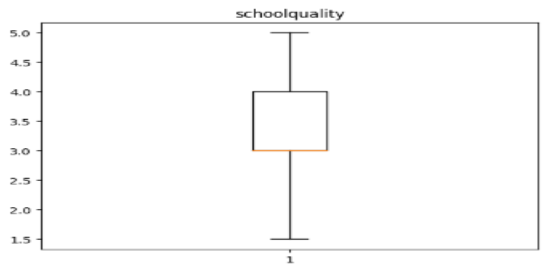


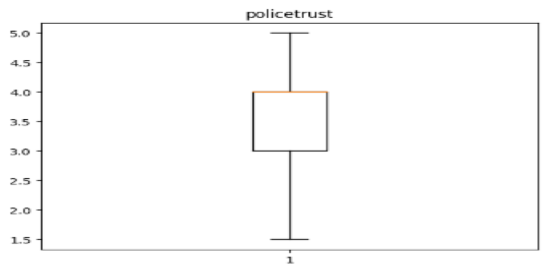
```
[ ] IQt = []
    IQt.append(df["infoavail"].quantile(0.75) - df["infoavail"].quantile(0.25))
    IQt
```
```
[1.0]
```
```
[ ] upper = []
    IQt.append(df["infoavail"].quantile(0.75) + 1.5 * IQt[0])
    IQt
```
```
[1.0, 6.5]
```
```
[ ] lower=[]
    IQt.append(df["infoavail"].quantile(0.25) - 1.5 * IQt[0])
    IQt
```
```
[1.0, 6.5, 2.5]
```
```
[ ] import numpy as np

    df["infoavail"] = np.where(df["infoavail"] == 6.5, 6.5, np.where(df["infoavail"] == 2.5, 2.5, df["infoavail"]))
```
```
[ ] IQt = []
    IQt.append(df["housecost"].quantile(0.75) - df["housecost"].quantile(0.25))
    IQt
```
```
[1.0]
```
```
[ ] upper = []
    IQt.append(df["housecost"].quantile(0.75) + 1.5 * IQt[0])
    IQt
```
```
[1.0, 4.5]
```
```
[ ] lower=[]
    IQt.append(df["housecost"].quantile(0.25) - 1.5 * IQt[0])
    IQt
```
```
[1.0, 4.5, 0.5]
```

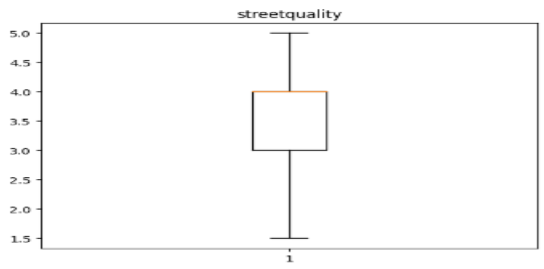
```
plt.figure(figsize=(5,5))
for i in df:
    plt.boxplot(df[i])
    plt.title(i)
    plt.show()
```

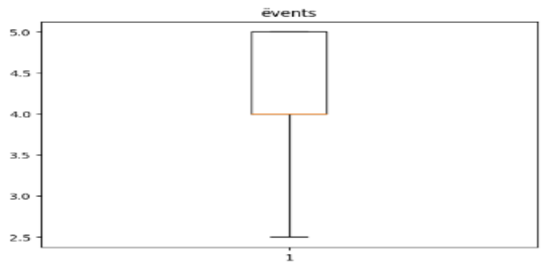


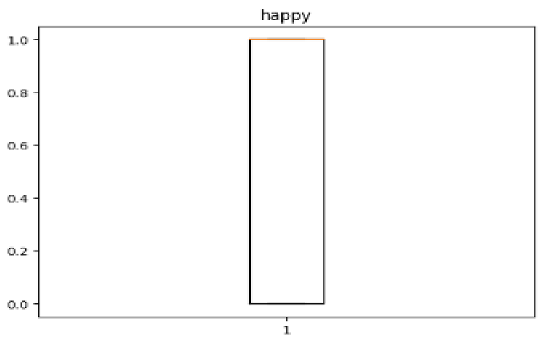



schoolquality
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1


policetrust
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1

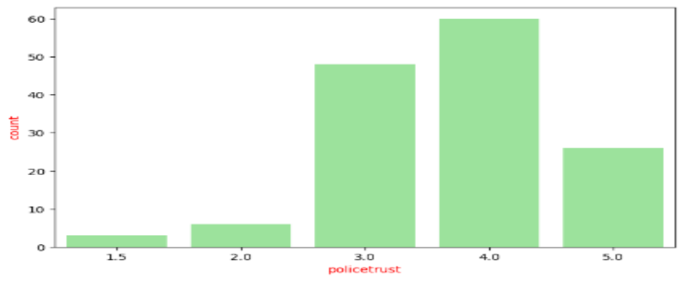

streetquality
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1


ëvents
5.0
4.5
4.0
3.5
3.0
2.5
1

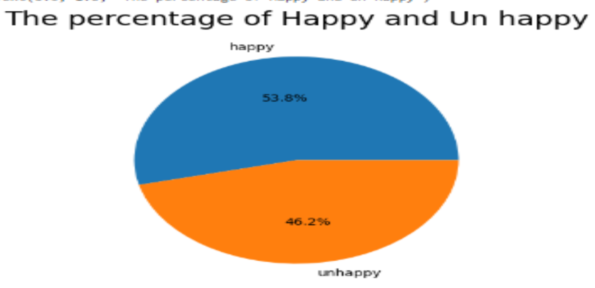

happy
1.0
0.8
0.6
0.4
0.2
0.0
1

VISUAL ANALYSIS

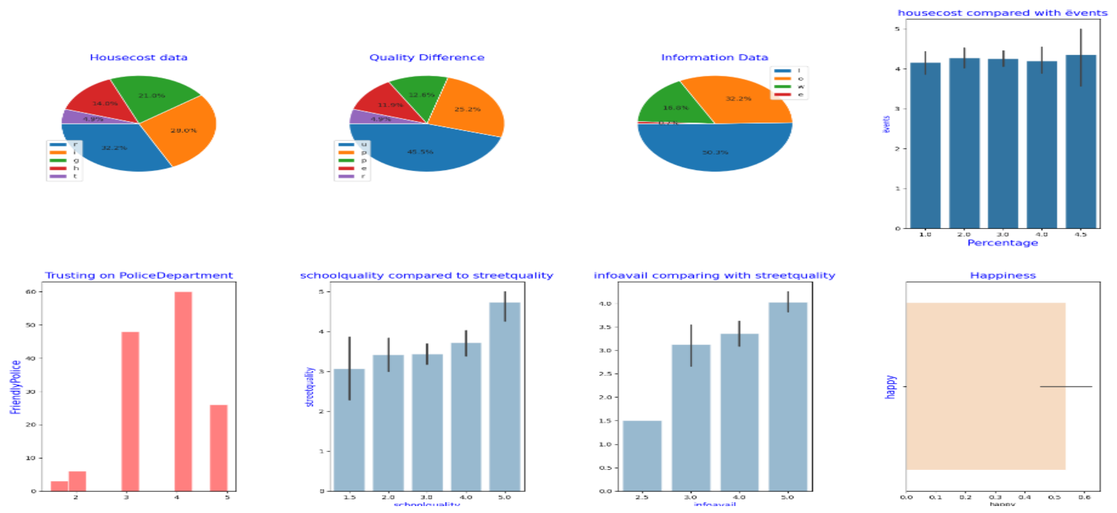
```
[ ] #UNIVARIATE ANALYSIS

    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    # Plot the countplot
    plt.figure(figsize=(8, 5))
    plt.grid(False)
    plt.title(" ", color="red", fontsize=14)
    plt.xlabel(None, color="red", fontsize=10)
    plt.ylabel(None, color="red", fontsize=10)
    sns.countplot(x=df['policetrust'], color='lightgreen')
    plt.show()
```

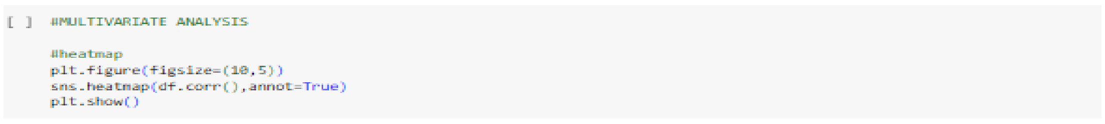


```
[ ] import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt

    #pie plot
    #checking wheather client are happy or unhappy
    plt.figure(figsize=(10, 5))
    plt.pie(df.happy.value_counts().values, labels=['happy', 'unhappy'], autopct='%1.1f%%')
    plt.title('The percentage of Happy and Un happy',color='black',size=20)

    Text(0.5, 1.0, 'The percentage of Happy and Un happy')
```

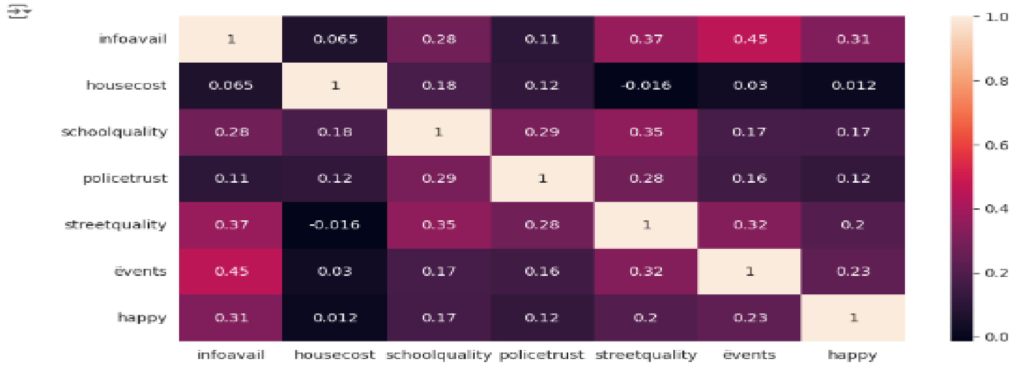


```
#BIVARIATE ANALYSIS

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 4, figsize=(20,15))
plt.tight_layout(pad=8)
plt.grid(False)

for ax in axes.flatten():
    ax.grid(False)
    sns.set_style("whitegrid")

axes[0,0].set_title('Housecost data', fontsize=15, color='blue')
axes[0,0].pie(df.housecost.value_counts(), autopct='%1.1f%%', startangle=180)
axes[0,0].legend('right')

axes[0,3].set_xlabel("housecost", fontsize=15)
axes[0,3].set_title('housecost compared with ëvents', fontsize=15, color="blue")
axes[0,3].set_xlabel("Percentage", fontsize=15, color="blue")
axes[0,3].set_ylabel(None, color="blue")
sns.barplot(x=df['housecost'], y=df['ëvents'],ax=axes[0,3])

axes[1,1].set_title('schoolquality compared to streetquality', fontsize=15, color="blue")
axes[1,1].set_xlabel('schoolquality', fontsize=12, color='blue')
axes[1,1].set_ylabel('streetquality', fontsize=12, color='blue')
sns.barplot(x=df['schoolquality'], y=df['streetquality'], ax=axes[1,1], alpha=0.5)

axes[0,1].set_title('Quality Difference', fontsize=15, color="blue")
axes[0,1].pie(df.schoolquality.value_counts(),autopct='%1.1f%%',startangle=180)
axes[0,1].legend('upper right')

axes[1,0].hist(df.policetrust, color='red', alpha=0.5)
axes[1,0].set_title('Trusting on PoliceDepartment', fontsize=15, color='blue')
axes[1,0].set_ylabel('FriendlyPolice', fontsize=15, color='blue')

# Modified the 'Infoavail' column to an existing column in the DataFrame
axes[0,2].set_title('Information Data', fontsize=15, color='blue')
axes[0,2].pie(df.infoavail.value_counts(),autopct='%1.1f%%',startangle=180)
axes[0,2].legend('lower left')

axes[1,2].set_title('infoavail comparing with streetquality', fontsize=15, color='Blue')
axes[1,2].set_xlabel('infoavail',fontsize=12,color="blue")
axes[1,2].set_ylabel(' ',fontsize=12,color='blue')
sns.barplot(x=df['infoavail'],y=df['streetquality'],ax=axes[1,2],alpha=0.5)

axes[1,3].set_title("Happiness" ,fontsize=15,color='blue')
axes[1,3].set_ylabel('happy',fontsize=15,color='blue')
sns.barplot(x=df['happy'],ax=axes[1,3],color='peachpuff')
```

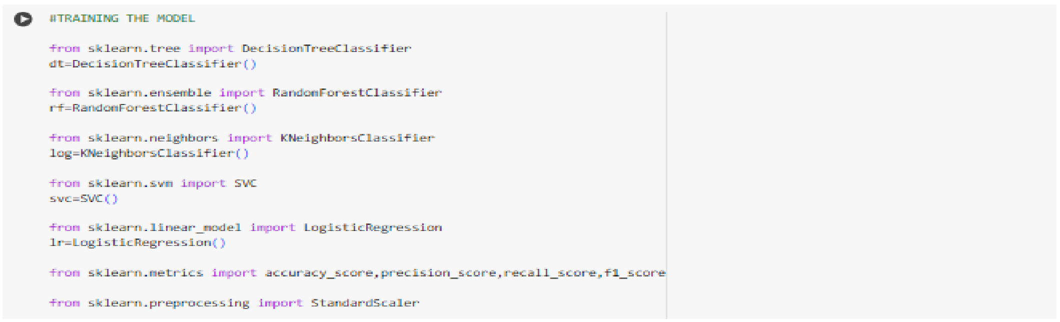




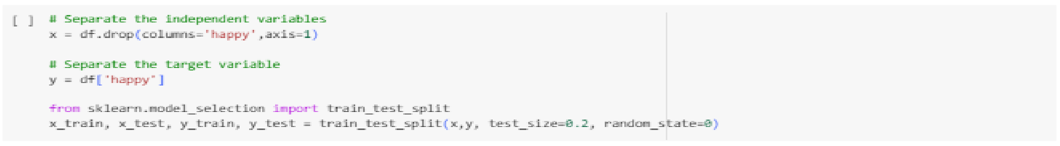
```
[ ]  #MULTIVARIATE ANALYSIS

     #heatmap
     plt.figure(figsize=(10,5))
     sns.heatmap(df.corr(),annot=True)
     plt.show()
```



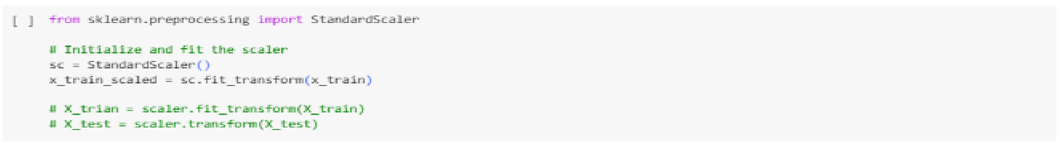
MODEL BUILDING



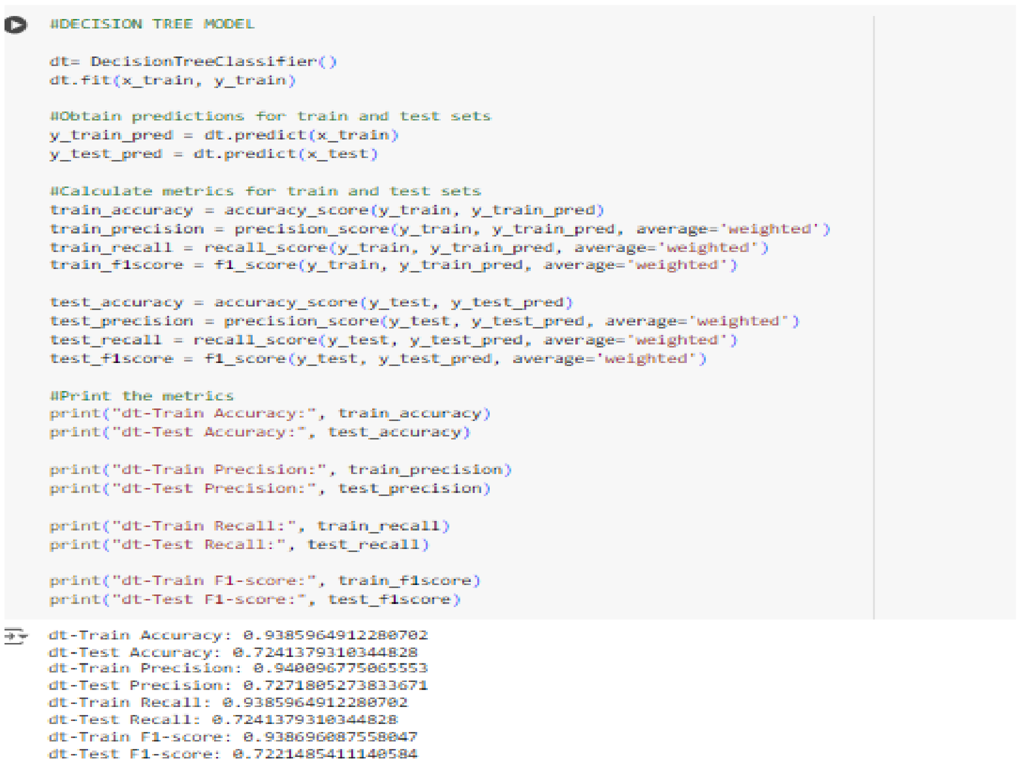
```
#TRAINING THE MODEL

from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()

from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()

from sklearn.neighbors import KNeighborsClassifier
log=KNeighborsClassifier()

from sklearn.svm import SVC
svc=SVC()

from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score

from sklearn.preprocessing import StandardScaler
```



```
[ ]  # Separate the independent variables
     x = df.drop(columns='happy',axis=1)

     # Separate the target variable
     y = df['happy']

     from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=0)
```

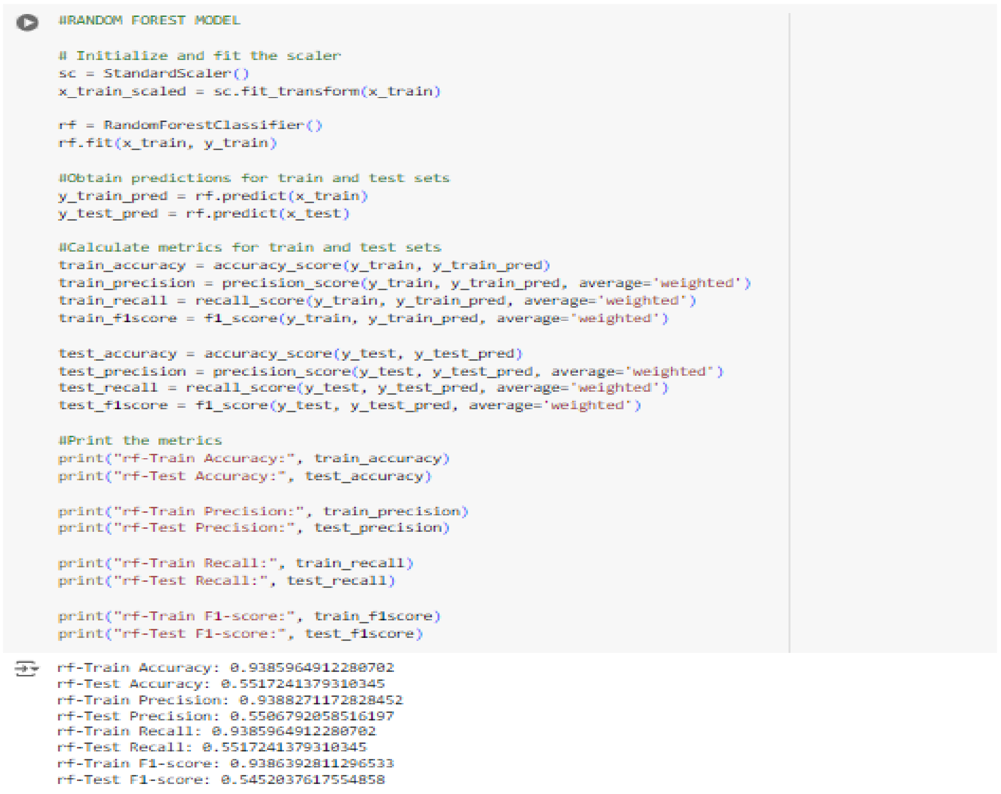


```
[ ]  from sklearn.preprocessing import StandardScaler

     # Initialize and fit the scaler
     sc = StandardScaler()
     x_train_scaled = sc.fit_transform(x_train)

     # X_trian = scaler.fit_transform(X_train)
     # X_test = scaler.transform(X_test)
```



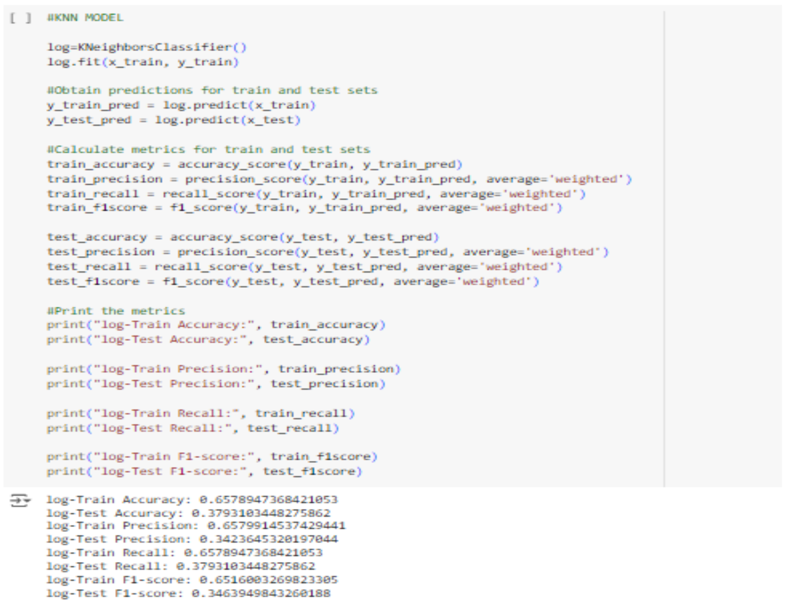
```
#DECISION TREE MODEL

dt= DecisionTreeClassifier()
dt.fit(x_train, y_train)

#Obtain predictions for train and test sets
y_train_pred = dt.predict(x_train)
y_test_pred = dt.predict(x_test)

#Calculate metrics for train and test sets
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred, average='weighted')
train_recall = recall_score(y_train, y_train_pred, average='weighted')
train_f1score = f1_score(y_train, y_train_pred, average='weighted')

test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred, average='weighted')
test_recall = recall_score(y_test, y_test_pred, average='weighted')
test_f1score = f1_score(y_test, y_test_pred, average='weighted')

#Print the metrics
print("dt-Train Accuracy:", train_accuracy)
print("dt-Test Accuracy:", test_accuracy)

print("dt-Train Precision:", train_precision)
print("dt-Test Precision:", test_precision)

print("dt-Train Recall:", train_recall)
print("dt-Test Recall:", test_recall)

print("dt-Train F1-score:", train_f1score)
print("dt-Test F1-score:", test_f1score)
```

```
dt-Train Accuracy: 0.9385964912280702
dt-Test Accuracy: 0.7241379310344828
dt-Train Precision: 0.940096775065553
dt-Test Precision: 0.7271805273833671
dt-Train Recall: 0.9385964912280702
dt-Test Recall: 0.7241379310344828
dt-Train F1-score: 0.9386960087558047
dt-Test F1-score: 0.7221485411140584
```



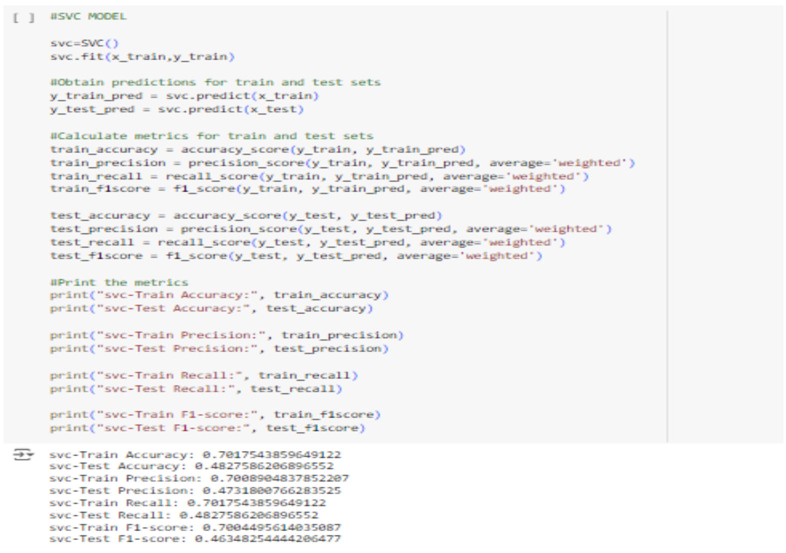
```
#RANDOM FOREST MODEL

# Initialize and fit the scaler
sc = StandardScaler()
x_train_scaled = sc.fit_transform(x_train)

rf = RandomForestClassifier()
rf.fit(x_train, y_train)

#Obtain predictions for train and test sets
y_train_pred = rf.predict(x_train)
y_test_pred = rf.predict(x_test)

#Calculate metrics for train and test sets
train_accuracy = accuracy_score(y_train, y_train_pred)
train_precision = precision_score(y_train, y_train_pred, average='weighted')
train_recall = recall_score(y_train, y_train_pred, average='weighted')
train_f1score = f1_score(y_train, y_train_pred, average='weighted')

test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred, average='weighted')
test_recall = recall_score(y_test, y_test_pred, average='weighted')
test_f1score = f1_score(y_test, y_test_pred, average='weighted')

#Print the metrics
print("rf-Train Accuracy:", train_accuracy)
print("rf-Test Accuracy:", test_accuracy)

print("rf-Train Precision:", train_precision)
print("rf-Test Precision:", test_precision)

print("rf-Train Recall:", train_recall)
print("rf-Test Recall:", test_recall)

print("rf-Train F1-score:", train_f1score)
print("rf-Test F1-score:", test_f1score)
```

```
rf-Train Accuracy: 0.9385964912280702
rf-Test Accuracy: 0.5517241379310345
rf-Train Precision: 0.9388271172828452
rf-Test Precision: 0.5506792058516197
rf-Train Recall: 0.9385964912280702
rf-Test Recall: 0.5517241379310345
rf-Train F1-score: 0.9386392811296533
rf-Test F1-score: 0.5452037617554858
```



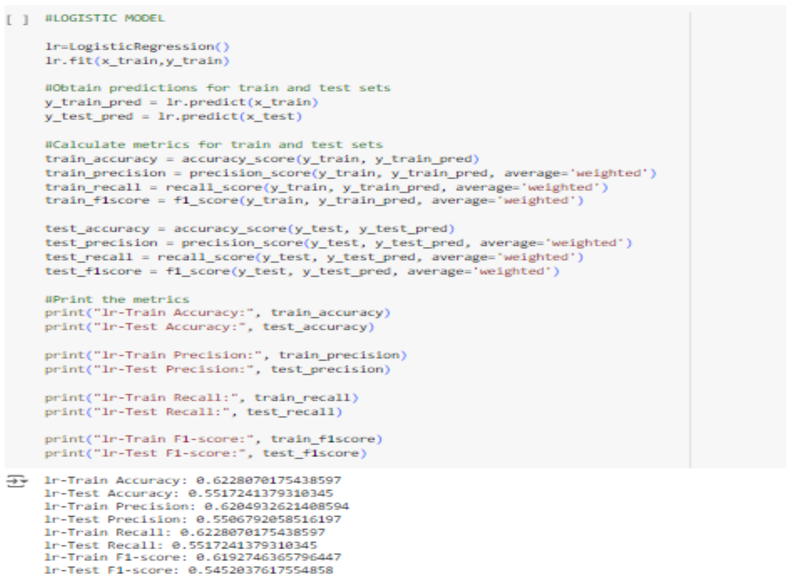
```
[ ] #KNN MODEL

    log=KNeighborsClassifier()
    log.fit(x_train, y_train)

    #Obtain predictions for train and test sets
    y_train_pred = log.predict(x_train)
    y_test_pred = log.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("log-Train Accuracy:", train_accuracy)
    print("log-Test Accuracy:", test_accuracy)

    print("log-Train Precision:", train_precision)
    print("log-Test Precision:", test_precision)

    print("log-Train Recall:", train_recall)
    print("log-Test Recall:", test_recall)

    print("log-Train F1-score:", train_f1score)
    print("log-Test F1-score:", test_f1score)
```

```
log-Train Accuracy: 0.6578947368421053
log-Test Accuracy: 0.3793103448275862
log-Train Precision: 0.6579914537429441
log-Test Precision: 0.3423645320197044
log-Train Recall: 0.6578947368421053
log-Test Recall: 0.3793103448275862
log-Train F1-score: 0.6516003269823305
log-Test F1-score: 0.3463949843260188
```



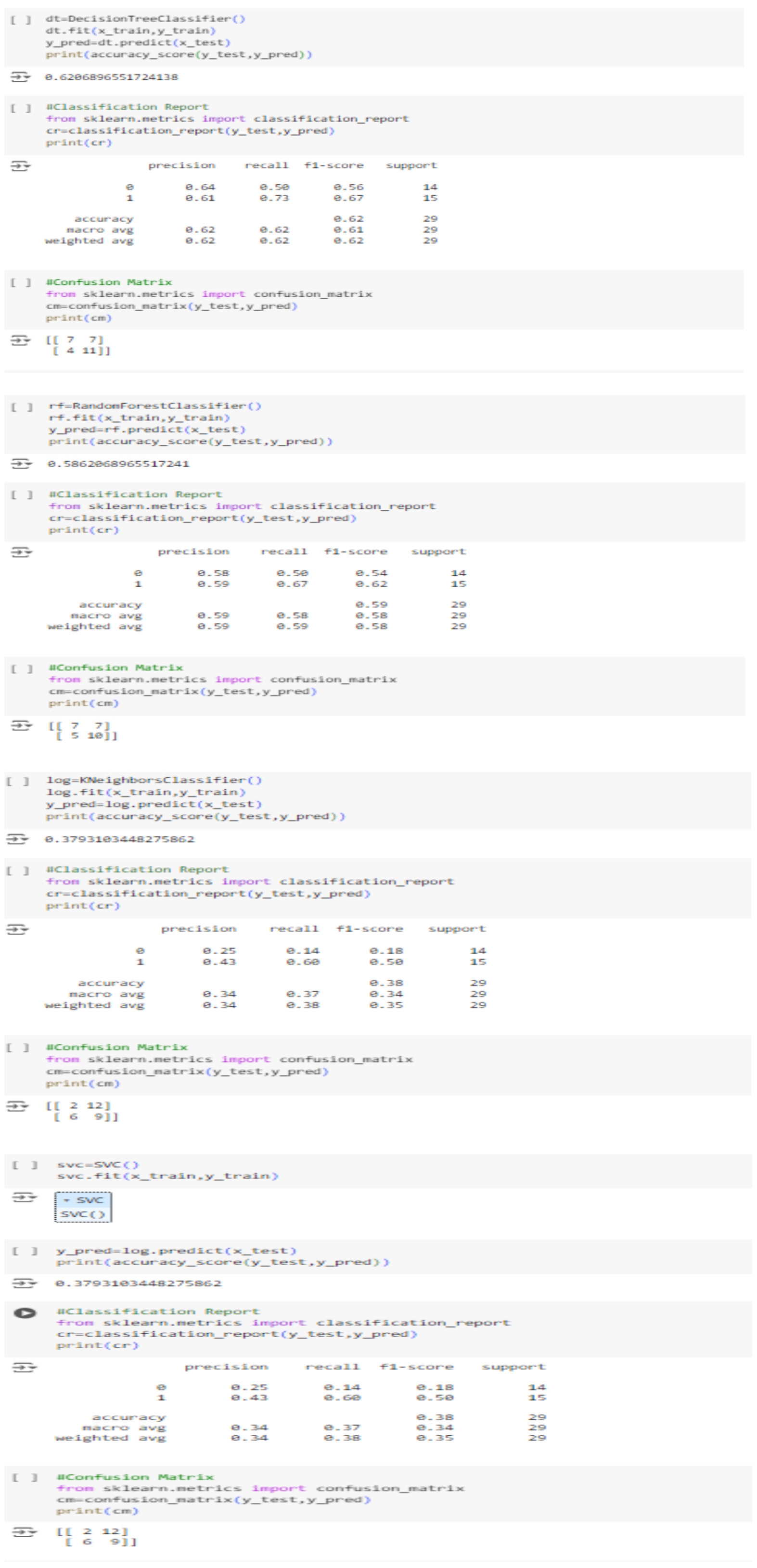
```
[ ] #SVC MODEL

    svc=SVC()
    svc.fit(x_train,y_train)

    #Obtain predictions for train and test sets
    y_train_pred = svc.predict(x_train)
    y_test_pred = svc.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("svc-Train Accuracy:", train_accuracy)
    print("svc-Test Accuracy:", test_accuracy)

    print("svc-Train Precision:", train_precision)
    print("svc-Test Precision:", test_precision)

    print("svc-Train Recall:", train_recall)
    print("svc-Test Recall:", test_recall)

    print("svc-Train F1-score:", train_f1score)
    print("svc-Test F1-score:", test_f1score)
```

```
svc-Train Accuracy: 0.7017543859649122
svc-Test Accuracy: 0.4827586206896552
svc-Train Precision: 0.7008904837852207
svc-Test Precision: 0.4731800766283525
svc-Train Recall: 0.7017543859649122
svc-Test Recall: 0.4827586206896552
svc-Train F1-score: 0.7004495614035087
svc-Test F1-score: 0.46348254444206477
```



```
[ ] #LOGISTIC MODEL

    lr=LogisticRegression()
    lr.fit(x_train,y_train)

    #Obtain predictions for train and test sets
    y_train_pred = lr.predict(x_train)
    y_test_pred = lr.predict(x_test)

    #Calculate metrics for train and test sets
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='weighted')
    train_recall = recall_score(y_train, y_train_pred, average='weighted')
    train_f1score = f1_score(y_train, y_train_pred, average='weighted')

    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='weighted')
    test_recall = recall_score(y_test, y_test_pred, average='weighted')
    test_f1score = f1_score(y_test, y_test_pred, average='weighted')

    #Print the metrics
    print("lr-Train Accuracy:", train_accuracy)
    print("lr-Test Accuracy:", test_accuracy)

    print("lr-Train Precision:", train_precision)
    print("lr-Test Precision:", test_precision)

    print("lr-Train Recall:", train_recall)
    print("lr-Test Recall:", test_recall)

    print("lr-Train F1-score:", train_f1score)
    print("lr-Test F1-score:", test_f1score)
```

```
lr-Train Accuracy: 0.6228070175438597
lr-Test Accuracy: 0.5517241379310345
lr-Train Precision: 0.6204932621408594
lr-Test Precision: 0.5506792058516197
lr-Train Recall: 0.6228070175438597
lr-Test Recall: 0.5517241379310345
lr-Train F1-score: 0.6192746365796447
lr-Test F1-score: 0.5452037617554858
```



```
dt=DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred=dt.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.6206896551724138
```

```
#Classification Report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.64      0.50      0.56        14
           1       0.61      0.73      0.67        15

    accuracy                           0.62        29
   macro avg       0.62      0.62      0.61        29
weighted avg       0.62      0.62      0.62        29
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[ 7  7]
 [ 4 11]]
```

```
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred=rf.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.5862068965517241
```

```
#Classification Report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.58      0.50      0.54        14
           1       0.59      0.67      0.62        15

    accuracy                           0.59        29
   macro avg       0.59      0.58      0.58        29
weighted avg       0.59      0.59      0.58        29
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[ 7  7]
 [ 5 10]]
```

```
log=KNeighborsClassifier()
log.fit(x_train,y_train)
y_pred=log.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.3793103448275862
```

```
#Classification Report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.25      0.14      0.18        14
           1       0.43      0.60      0.50        15

    accuracy                           0.38        29
   macro avg       0.34      0.37      0.34        29
weighted avg       0.34      0.38      0.35        29
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[ 2 12]
 [ 6  9]]
```

```
svc=SVC()
svc.fit(x_train,y_train)
```

```
▾ SVC
SVC()
```

```
y_pred=log.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

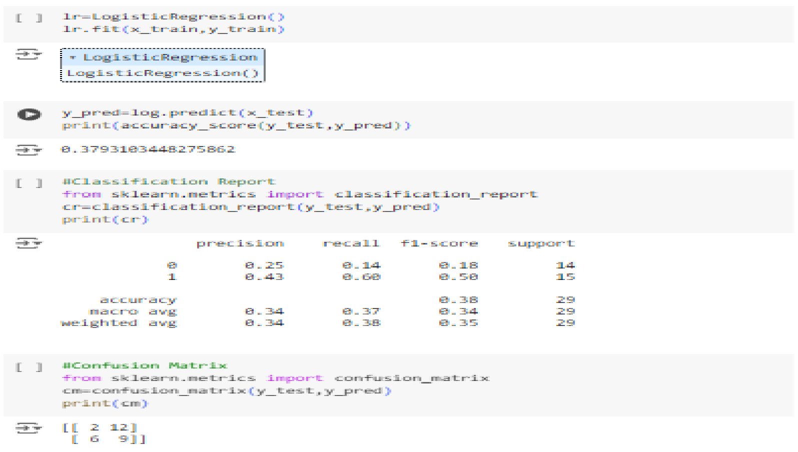
```
0.3793103448275862
```

```
#Classification Report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.25      0.14      0.18        14
           1       0.43      0.60      0.50        15

    accuracy                           0.38        29
   macro avg       0.34      0.37      0.34        29
weighted avg       0.34      0.38      0.35        29
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[ 2 12]
 [ 6  9]]
```



```
lr=LogisticRegression()
lr.fit(x_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred=log.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

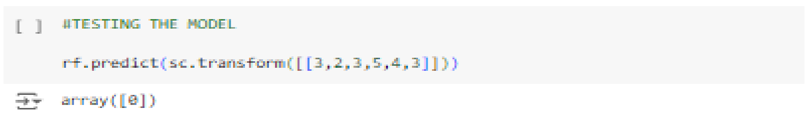
```
0.3793103448275862
```

```
#Classification Report
from sklearn.metrics import classification_report
cr=classification_report(y_test,y_pred)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.25      0.14      0.18        14
           1       0.43      0.60      0.50        15

    accuracy                           0.38        29
   macro avg       0.34      0.37      0.34        29
weighted avg       0.34      0.38      0.35        29
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

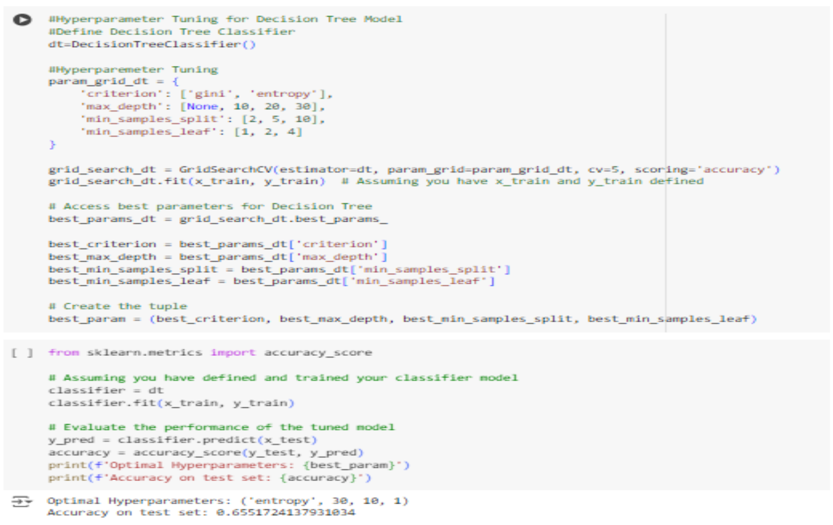
```
[[ 2 12]
 [ 6  9]]
```



```
#TESTING THE MODEL

rf.predict(sc.transform([[3,2,3,5,4,3]]))
```

```
array([0])
```

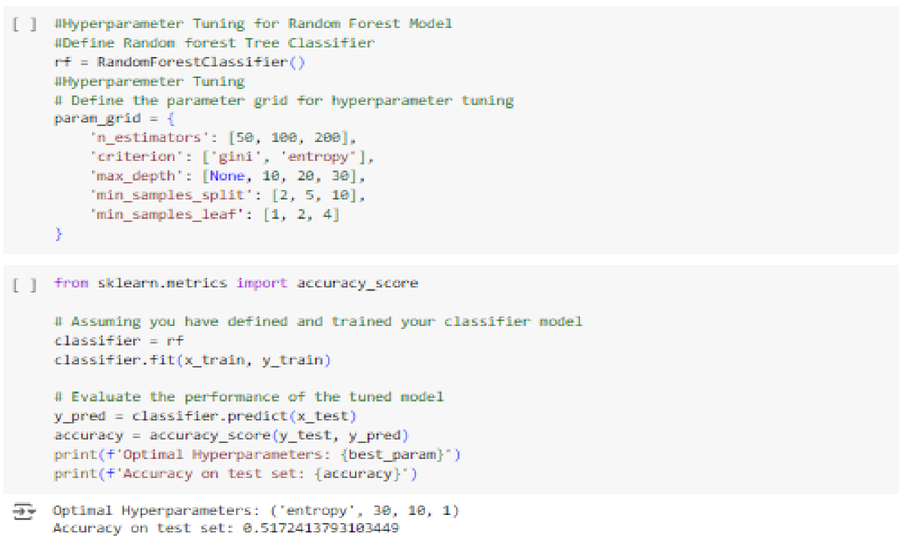
HYPERPARAMETER TUNING



```
#Hyperparameter Tuning for Decision Tree Model
#Define Decision Tree Classifier
dt=DecisionTreeClassifier()

#Hyperparemeter Tuning
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_dt = GridSearchCV(estimator=dt, param_grid=param_grid_dt, cv=5, scoring='accuracy')
grid_search_dt.fit(x_train, y_train)  # Assuming you have x_train and y_train defined

# Access best parameters for Decision Tree
best_params_dt = grid_search_dt.best_params_

best_criterion = best_params_dt['criterion']
best_max_depth = best_params_dt['max_depth']
best_min_samples_split = best_params_dt['min_samples_split']
best_min_samples_leaf = best_params_dt['min_samples_leaf']

# Create the tuple
best_param = (best_criterion, best_max_depth, best_min_samples_split, best_min_samples_leaf)
```

```
from sklearn.metrics import accuracy_score

# Assuming you have defined and trained your classifier model
classifier = dt
classifier.fit(x_train, y_train)

# Evaluate the performance of the tuned model
y_pred = classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Optimal Hyperparameters: {best_param}')
print(f'Accuracy on test set: {accuracy}')
```

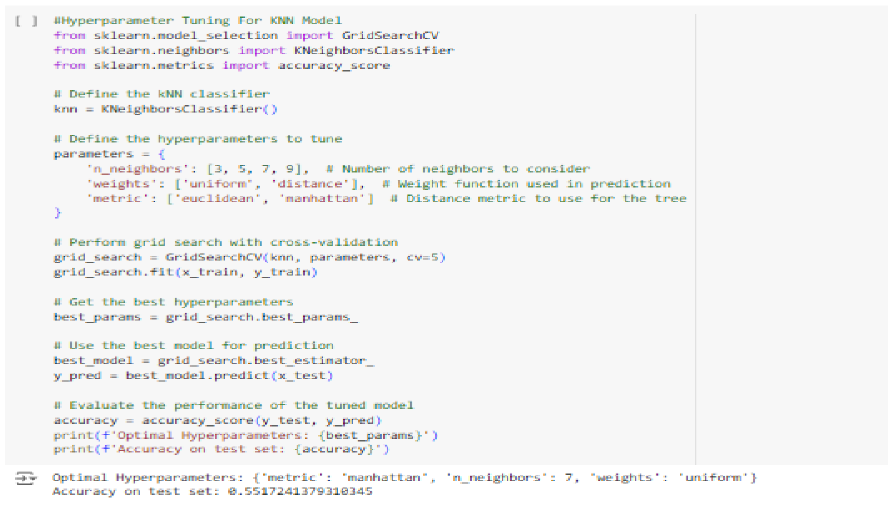
```
Optimal Hyperparameters: ('entropy', 30, 10, 1)
Accuracy on test set: 0.6551724137931034
```



```
#Hyperparameter Tuning for Random Forest Model
#Define Random forest Tree Classifier
rf = RandomForestClassifier()
#Hyperparemeter Tuning
# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

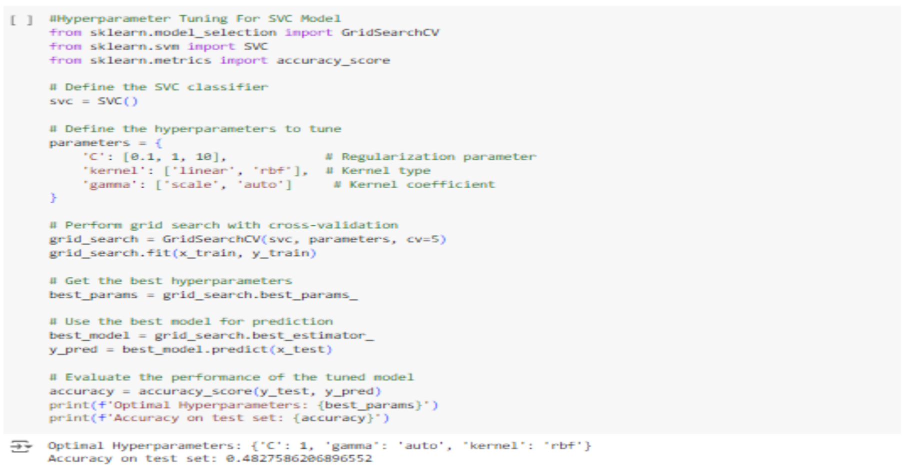
```
from sklearn.metrics import accuracy_score

# Assuming you have defined and trained your classifier model
classifier = rf
classifier.fit(x_train, y_train)

# Evaluate the performance of the tuned model
y_pred = classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Optimal Hyperparameters: {best_param}')
print(f'Accuracy on test set: {accuracy}')
```

```
Optimal Hyperparameters: ('entropy', 30, 10, 1)
Accuracy on test set: 0.5172413793103449
```



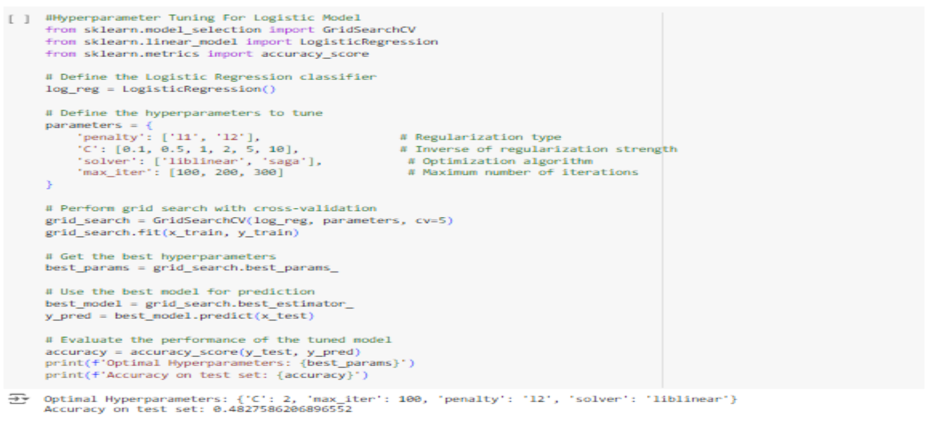
```
[ ]  #Hyperparameter Tuning For KNN Model
     from sklearn.model_selection import GridSearchCV
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score

     # Define the kNN classifier
     knn = KNeighborsClassifier()

     # Define the hyperparameters to tune
     parameters = {
         'n_neighbors': [3, 5, 7, 9],  # Number of neighbors to consider
         'weights': ['uniform', 'distance'],  # Weight function used in prediction
         'metric': ['euclidean', 'manhattan']  # Distance metric to use for the tree
     }

     # Perform grid search with cross-validation
     grid_search = GridSearchCV(knn, parameters, cv=5)
     grid_search.fit(x_train, y_train)

     # Get the best hyperparameters
     best_params = grid_search.best_params_

     # Use the best model for prediction
     best_model = grid_search.best_estimator_
     y_pred = best_model.predict(x_test)

     # Evaluate the performance of the tuned model
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Optimal Hyperparameters: {best_params}')
     print(f'Accuracy on test set: {accuracy}')
```

```
Optimal Hyperparameters: {'metric': 'manhattan', 'n_neighbors': 7, 'weights': 'uniform'}
Accuracy on test set: 0.5517241379310345
```



```
[ ]  #Hyperparameter Tuning For SVC Model
     from sklearn.model_selection import GridSearchCV
     from sklearn.svm import SVC
     from sklearn.metrics import accuracy_score

     # Define the SVC classifier
     svc = SVC()

     # Define the hyperparameters to tune
     parameters = {
         'C': [0.1, 1, 10],                # Regularization parameter
         'kernel': ['linear', 'rbf'],      # Kernel type
         'gamma': ['scale', 'auto']         # Kernel coefficient
     }

     # Perform grid search with cross-validation
     grid_search = GridSearchCV(svc, parameters, cv=5)
     grid_search.fit(x_train, y_train)

     # Get the best hyperparameters
     best_params = grid_search.best_params_

     # Use the best model for prediction
     best_model = grid_search.best_estimator_
     y_pred = best_model.predict(x_test)

     # Evaluate the performance of the tuned model
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Optimal Hyperparameters: {best_params}')
     print(f'Accuracy on test set: {accuracy}')
```

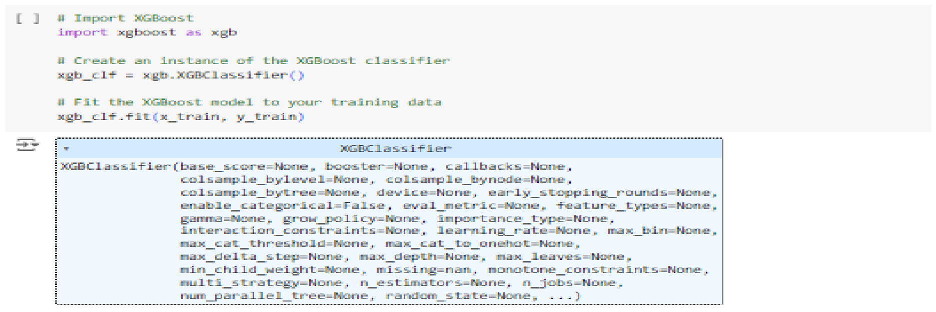
```
Optimal Hyperparameters: {'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}
Accuracy on test set: 0.4827586206896552
```



```
[ ]  #Hyperparameter Tuning For Logistic Model
     from sklearn.model_selection import GridSearchCV
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import accuracy_score

     # Define the Logistic Regression classifier
     log_reg = LogisticRegression()

     # Define the hyperparameters to tune
     parameters = {
         'penalty': ['l1', 'l2'],                      # Regularization type
         'C': [0.1, 0.5, 1, 2, 5, 10],                 # Inverse of regularization strength
         'solver': ['liblinear', 'saga'],               # Optimization algorithm
         'max_iter': [100, 200, 300]                    # Maximum number of iterations
     }

     # Perform grid search with cross-validation
     grid_search = GridSearchCV(log_reg, parameters, cv=5)
     grid_search.fit(x_train, y_train)

     # Get the best hyperparameters
     best_params = grid_search.best_params_

     # Use the best model for prediction
     best_model = grid_search.best_estimator_
     y_pred = best_model.predict(x_test)

     # Evaluate the performance of the tuned model
     accuracy = accuracy_score(y_test, y_pred)
     print(f'Optimal Hyperparameters: {best_params}')
     print(f'Accuracy on test set: {accuracy}')
```

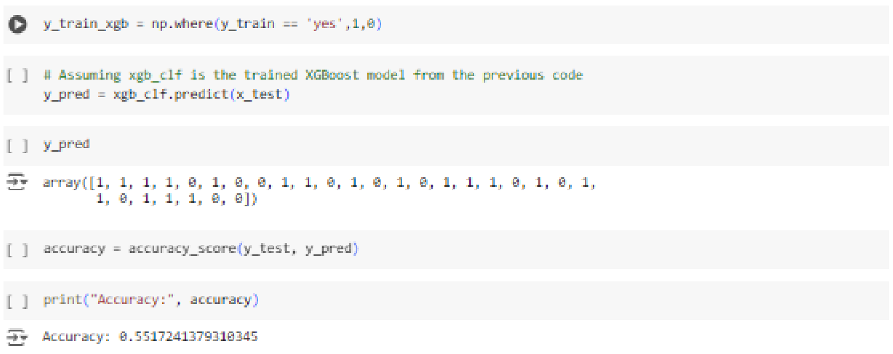
```
Optimal Hyperparameters: {'C': 2, 'max_iter': 100, 'penalty': 'l2', 'solver': 'liblinear'}
Accuracy on test set: 0.4827586206896552
```

PERFORMANCE XGBCLASSIFIER MODEL



```
[ ]  # Import XGBoost
     import xgboost as xgb

     # Create an instance of the XGBoost classifier
     xgb_clf = xgb.XGBClassifier()

     # Fit the XGBoost model to your training data
     xgb_clf.fit(x_train, y_train)
```

```
                              XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```



```
y_train_xgb = np.where(y_train == 'yes',1,0)
```

```
# Assuming xgb_clf is the trained XGBoost model from the previous code
y_pred = xgb_clf.predict(x_test)
```

```
y_pred
```

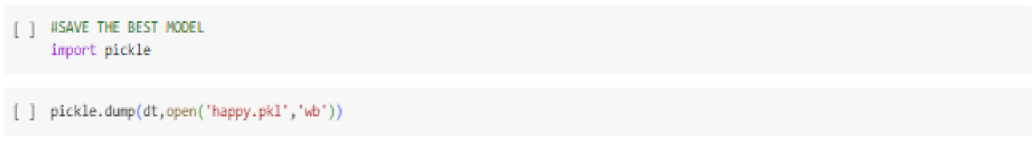
```
array([1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 1, 0, 0])
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

```
Accuracy: 0.5517241379310345
```

MODEL DEPLOYMENT



```
#SAVE THE BEST MODEL
import pickle
```

```
pickle.dump(dt,open('happy.pkl','wb'))
```