Course Code: CSE 3202

Course Title: Artificial Intelligence and Machine Learning Lab
Section: 01
Spring 2024

Report on
Complex Engineering Project

Submitted to:

Noman Hossain, PhD
Assistant Professor
Department of Computer Science and Engineering (CSE)
University of Liberal Arts Bangladesh

Submitted by:

Student Name	Student ID
Fahim Sadik	213014027
Samiul Islam Ramim	213014020

Date: May 4, 2024

Table of Contents

1. **Problem Statement**

The concern of this research is verifying the effects of game addiction on different university students, as well as their physical and mental health. This study analyzed data on people with a range of mentalities, behaviors, and psychological backgrounds to examine the most prevalent causes of game addiction.

2. **Objective:**
    - To understand the data preprocessing and predict real-life problems.
    - To use a classifier algorithm.
    - To predict if someone is an addict or not.
    - To find out the accuracy, precision, and recall for both classifiers and compare between them.

3. **Resources:**
    - **Anaconda**
    - **Spyder**
    - **Python 3.11**
    - **Microsoft Excel**

4. **Logic:**
    - **4.1 Logics for the program**

This study aims to find out the impacts of digital game addiction levels by using demographic variables. To conduct this research, we have taken a dataset of 458 participants and then we figured out how many people have game addiction. To determine if a person is addicted to gaming or not, a questionnaire consisting of 30 questions was circulated among various ages of people. In this fact, we have taken the answers in a multiple choice format and graded them into 5 sections, like all the questions have never, rarely, sometimes, often and very often. We set the values for them like never=1, rarely=2, sometimes=3, often=4 & very often =5. Because we used a classifier algorithm for this and a classifier needs a numerical value to work we needed to set values for those strings into numbers. We used SVC as the classifier

    - **4.2 Splitting the Data**

The dataset is split into training and test sets. Here 70% is used for training and 30% for test

#	Column	Non-Null Count	Dtype
0	thinkofPlayingGameDayLong	462 non-null	int64
1	spendFreeTimeonGame	462 non-null	int64
2	feelingofGameAddiction	462 non-null	int64
3	playingLongerThenIntended	462 non-null	int64
4	spendLargeTimeonGame	462 non-null	int64
5	unableToStopPlaying	462 non-null	int64
6	gamingToForgetRealLife	462 non-null	int64
7	gamingToReleaseStress	462 non-null	int64
8	gamingToFeelBetter	462 non-null	int64
9	unableToReduceGaming	462 non-null	int64
10	unsuccessfulInfluenceofOthers	462 non-null	int64
11	angryIssue	462 non-null	int64
12	stressIssue	462 non-null	int64
13	unsocialIssue	462 non-null	int64
14	fightIssue	462 non-null	int64
15	breakofRelationship	462 non-null	int64
16	tendToDeceive	462 non-null	int64
17	sleepIssue	462 non-null	int64
18	tendToloseHobbies	462 non-null	int64
19	tendToNeglectImportantActivities	462 non-null	int64
20	neckandBackPain	462 non-null	int64
21	orthopedicIssues	462 non-null	int64
22	eyesightIssue	462 non-null	int64
23	hearingIssue	462 non-null	int64
24	tendToCocurricularActivities	462 non-null	int64
25	tendToPresentClass	462 non-null	int64
26	tendToPeerInterraction	462 non-null	int64
27	addictionIndicator	462 non-null	int64
28	mentalDisorder	462 non-null	int64
29	physicalDisorder	462 non-null	int64

This figure shows that there is no null value in the training set.

- **4.3 Model Training:**

Using SVC, svm_model1.fit(X_train, y_train)

- **4.4 Model Prediction and Evaluation:**

```
y_pred = svm_model1.predict(X_test)
X_new = [[1,1,1,1,1,2,1,3,1,2,1,2,1,3,1,4,5,2,4,1]]
prediction = svm_model1.predict(X_new)
print("Prediction:", prediction)
```

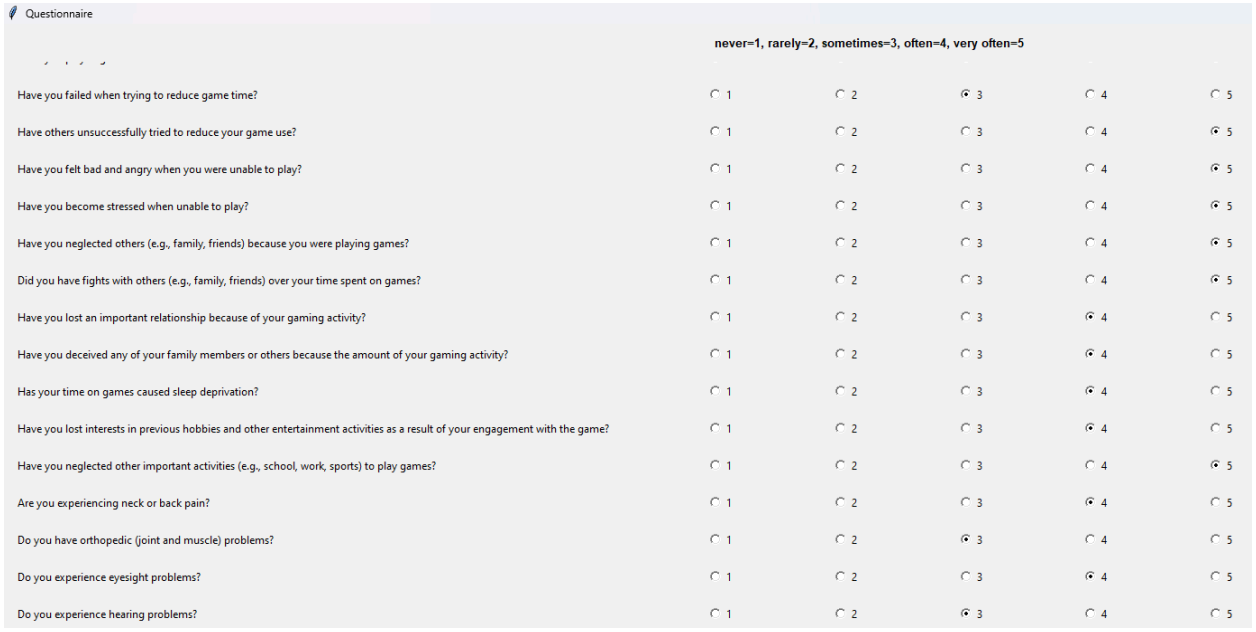
- **4.5 Performance Evaluation:**

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

- **4.6 GUI Creation:**

• The GUI was created using Tkinter.
• Input options for the prediction of an addict or not.
• Buttons are provided for triggering prediction and graphs.

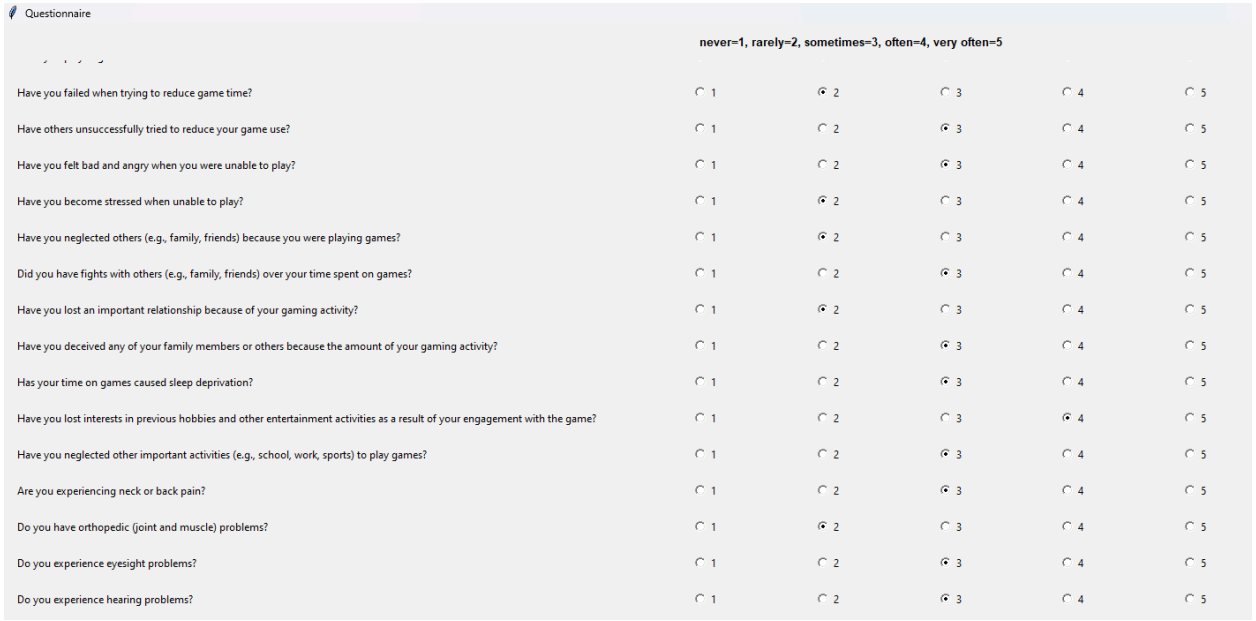
5. **Input:**



Questionnaire

never=1, rarely=2, sometimes=3, often=4, very often=5

	1	2	3	4	5
Have you failed when trying to reduce game time?	○	○	●	○	○
Have others unsuccessfully tried to reduce your game use?	○	○	○	○	●
Have you felt bad and angry when you were unable to play?	○	○	○	○	●
Have you become stressed when unable to play?	○	○	○	○	●
Have you neglected others (e.g., family, friends) because you were playing games?	○	○	○	○	●
Did you have fights with others (e.g., family, friends) over your time spent on games?	○	○	○	○	●
Have you lost an important relationship because of your gaming activity?	○	○	○	●	○
Have you deceived any of your family members or others because the amount of your gaming activity?	○	○	○	●	○
Has your time on games caused sleep deprivation?	○	○	○	●	○
Have you lost interests in previous hobbies and other entertainment activities as a result of your engagement with the game?	○	○	○	●	○
Have you neglected other important activities (e.g., school, work, sports) to play games?	○	○	○	○	●
Are you experiencing neck or back pain?	○	○	○	●	○
Do you have orthopedic (joint and muscle) problems?	○	○	●	○	○
Do you experience eyesight problems?	○	○	○	●	○
Do you experience hearing problems?	○	○	●	○	○



Questionnaire

never=1, rarely=2, sometimes=3, often=4, very often=5

	1	2	3	4	5
Have you failed when trying to reduce game time?	○	●	○	○	○
Have others unsuccessfully tried to reduce your game use?	○	○	●	○	○
Have you felt bad and angry when you were unable to play?	○	○	●	○	○
Have you become stressed when unable to play?	○	●	○	○	○
Have you neglected others (e.g., family, friends) because you were playing games?	○	●	○	○	○
Did you have fights with others (e.g., family, friends) over your time spent on games?	○	○	●	○	○
Have you lost an important relationship because of your gaming activity?	○	●	○	○	○
Have you deceived any of your family members or others because the amount of your gaming activity?	○	○	●	○	○
Has your time on games caused sleep deprivation?	○	○	●	○	○
Have you lost interests in previous hobbies and other entertainment activities as a result of your engagement with the game?	○	○	○	●	○
Have you neglected other important activities (e.g., school, work, sports) to play games?	○	○	●	○	○
Are you experiencing neck or back pain?	○	○	●	○	○
Do you have orthopedic (joint and muscle) problems?	○	●	○	○	○
Do you experience eyesight problems?	○	○	●	○	○
Do you experience hearing problems?	○	○	●	○	○

6. Code:

Model Training Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn import metrics
from scipy.stats import ttest_1samp
from sklearn.metrics import classification_report
from mlxtend.plotting import plot_decision_regions
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.metrics import f1_score, precision_score, recall_score
import seaborn as sns
sns.set()
#import csv
import os
import tensorflow as tf
from tensorflow.keras.layers import Dense, LSTM
from tensorflow.keras import Sequential
import tkinter as tk
from tkinter import ttk
from tkinter import messagebox
import pickle
from PIL import Image, ImageTk
import warnings
warnings.filterwarnings('ignore')
df = pd.read_csv('new.csv')
# Change string into numeric data value
df['thinkofPlayingGameDayLong'] = df['thinkofPlayingGameDayLong'].apply({'Never' : 1, 'Rarely'
: 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['spendFreeTimeonGame'] = df['spendFreeTimeonGame'].apply({'Never' : 1, 'Rarely' : 2,
'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['feelingofGameAddiction'] = df['feelingofGameAddiction'].apply({'Never' : 1, 'Rarely' : 2,
'Somtimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['playingLongerThenIntended'] = df['playingLongerThenIntended'].apply({'Never' : 1, 'Rarely' :
2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['spendLargeTimeonGame'] = df['spendLargeTimeonGame'].apply({'Never' : 1, 'Rarely' : 2,
'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['unableToStopPlaying'] = df['unableToStopPlaying'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes'
: 3, 'Often' : 4, 'Very Often' : 5}.get)
df['gamingToForgetRealLife'] = df['gamingToForgetRealLife'].apply({'Never' : 1, 'Rarely' : 2,
'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['gamingToReleaseStress'] = df['gamingToReleaseStress'].apply({'Never' : 1, 'Rarely' : 2,
'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['gamingToFeelBetter'] = df['gamingToFeelBetter'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' :
3, 'Often' : 4, 'Very Often' : 5}.get)
df['unableToReduceGaming'] = df['unableToReduceGaming'].apply({'Never' : 1, 'Rarely' : 2,
'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
```

```
df['unsuccessfulInfluenceofOthers'] = df['unsuccessfulInfluenceofOthers'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['angryIssue'] = df['angryIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['stressIssue'] = df['stressIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['unsocialIssue'] = df['unsocialIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['fightIssue'] = df['fightIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['breakofRelationship'] = df['breakofRelationship'].apply({'Never' : 1, 'Rarely' : 2, 'Sometime' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToDeceive'] = df['tendToDeceive'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['sleepIssue'] = df['sleepIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToloseHobbies'] = df['tendToloseHobbies'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToNeglectImportantActivities'] = df['tendToNeglectImportantActivities'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['neckandBackPain'] = df['neckandBackPain'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['orthopedicIssues'] = df['orthopedicIssues'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['eyesightIssue'] = df['eyesightIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['hearingIssue'] = df['hearingIssue'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToCocurricularActivities'] = df['tendToCocurricularActivities'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToPresentClass'] = df['tendToPresentClass'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df['tendToPeerInterraction'] = df['tendToPeerInterraction'].apply({'Never' : 1, 'Rarely' : 2, 'Sometimes' : 3, 'Often' : 4, 'Very Often' : 5}.get)
df.iloc[:,-10:]
df.info(verbose=True)
df.describe().T
#heatmap
plt.figure(figsize=(20,20))  # on this line I just set the size of figure to 12 by 10.
p=sns.heatmap(df.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very simple solution for heatmap
#addiction Calculation
df['addictionIndicator']= df.iloc[:,:20].sum(axis=1)
df.loc[:, 'addictionIndicator'] = np.where(df.addictionIndicator>=59, 1, 0)
```

```
df.head()
#0=not addict, 1=Addict
#Mental Disorder Calculation
df['mentalDisorder']= df.iloc[:,[0,4,6,7,14,15,16,18]].sum(axis=1)
df.loc[:, 'mentalDisorder'] = np.where(df.mentalDisorder>=29, 1, 0)
df.head()
#0=not Mental, 1=Mental
#physical Disorder Calculation
df['physicalDisorder']= df.iloc[:,[20,21,22,23]].sum(axis=1)
df.loc[:, 'physicalDisorder'] = np.where(df.physicalDisorder>=12, 1, 0)
df.head()
#0=not physical, 1=physical
#how many are addicted
count = df['addictionIndicator'].value_counts()
#relative frequency
tot_addicted = count[1]
tot_nonaddicted = count[0]
tot_participants = tot_addicted + tot_nonaddicted
print('Total Participants:', tot_participants)
print('Total addicted:', tot_addicted)
print('Total non-addicted:', tot_nonaddicted)
print('Percentage of addicted people: {:.2f}%'.format((tot_addicted / tot_participants) * 100))
print('Percentage of non-addicted people: {:.2f}%'.format((tot_nonaddicted / tot_participants) *
100))
#0=not addict, 1=Addict
#among game addictor how many are having mental disorder
print('People having mental disorder with addiction:', len(df[(df['addictionIndicator']==1) &
(df['mentalDisorder']==1)]))
#relative frequency of mental disorder among game addictor
perc_mental_disorder = len(df[(df['addictionIndicator']==1) &
(df['mentalDisorder']==1)])/tot_addicted
print('People having mental disorder with addiction (Percentage):
{:.2f}%'.format(perc_mental_disorder * 100))
#among game addictor how many are having physical disorder
print('People having physical disorder with addiction:', len(df[(df['addictionIndicator']==1) &
(df['physicalDisorder']==1)]))
#relative frequency of physical disorder among game addictor
perc_physical_disorder = len(df[(df['addictionIndicator']==1) &
(df['physicalDisorder']==1)])/tot_addicted
print('People having physical disorder with addiction (Percentage):
{:.2f}%'.format(perc_physical_disorder * 100))
#histogram
p = df.hist(figsize = (20,20))
features = ['Addicted', 'Non-Addicted']
```

```
values = [tot_addicted, tot_nonaddicted]
plt.figure(figsize=(10, 10))
plt.bar(features, values, width=0.2)
plt.xlabel('Feature')
plt.ylabel('Frequency')
plt.title('Total number of Addicted and Non-addicted person')
plt.show()
#svm Addiction
X=df.iloc[:, 0:20]
y=df.iloc[:,27]
y
# Assuming X contains features and y contains labels
# Replace X and y with your actual data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Initialize SVC with chosen kernel
svm_model1 = SVC(kernel='linear')
# Train SVC
svm_model1.fit(X_train, y_train)
# Predictions on test set
y_pred = svm_model1.predict(X_test)
# Performance evaluation
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Precision:", precision)
print("Recall:", recall)
print("Confusion Matrix:\n", conf_matrix)
# Example prediction
# Replace X_new with the new data point(s) you want to predict on
X_new = [[1,1,1,1,1,2,1,3,1,2,1,2,1,3,1,4,5,2,4,1]]  # Replace ... with actual feature values
prediction = svm_model1.predict(X_new)
print("Prediction:", prediction)
 #  now save the model for future use
filename = 'Addiction'
pickle.dump(svm_model1, open(filename, 'wb'))
#svm MEntal
V=df.iloc[:, [0, 4, 6, 7, 14, 15, 16, 18]]
n=df.iloc[:,28]
n
# Assuming X contains features and y contains labels
```

```
# Replace X and y with your actual data
V_train, V_test, n_train, n_test = train_test_split(V, n, test_size=0.3, random_state=42)
# Initialize SVM with chosen kernel
svm_model2 = SVC(kernel='linear')
# Train SVM
svm_model2.fit(V_train, n_train)
# Predictions on test set
n_pred = svm_model2.predict(V_test)
# Performance evaluation
accuracy5 = accuracy_score(n_test, n_pred)
f15 = f1_score(n_test, n_pred)
precision5 = precision_score(n_test, n_pred)
recall5 = recall_score(n_test, n_pred)
conf_matrix5 = confusion_matrix(n_test, n_pred)
print("Accuracy:", accuracy5)
print("F1 Score:", f15)
print("Precision:", precision5)
print("Recall:", recall5)
print("Confusion Matrix:\n", conf_matrix5)
# Example prediction
# Replace X_new with the new data point(s) you want to predict on
V_new = [[1,1,1,1,1,2,3,3]]  # Replace ... with actual feature values
prediction = svm_model2.predict(V_new)
print("Prediction:", prediction)
#  now save the model for future use
filename = 'mental'
pickle.dump(svm_model2, open(filename, 'wb'))
#svm Physical
Z=df.iloc[:, 20:24]
b=df.iloc[:,29]
b
# Assuming X contains features and y contains labels
# Replace X and y with your actual data
Z_train, Z_test, b_train, b_test = train_test_split(Z, b, test_size=0.3, random_state=42)
# Initialize SVM with chosen kernel
svm_model3 = SVC(kernel='linear')
# Train SVM
svm_model3.fit(Z_train, b_train)
# Predictions on test set
b_pred = svm_model3.predict(Z_test)
# Performance evaluation
accuracy6 = accuracy_score(b_test, b_pred)
f16 = f1_score(b_test, b_pred)
precision6 = precision_score(b_test, b_pred)
```

```
recall6 = recall_score(b_test, b_pred)
conf_matrix6 = confusion_matrix(b_test, b_pred)
print("Accuracy:", accuracy6)
print("F1 Score:", f16)
print("Precision:", precision6)
print("Recall:", recall6)
print("Confusion Matrix:\n", conf_matrix6)
# Example prediction
# Replace X_new with the new data point(s) you want to predict on
Z_new = [[1,2,1,2]]  # Replace ... with actual feature values
prediction = svm_model3.predict(Z_new)
print("Prediction:", prediction)
 #  now save the model for future use
filename = 'physic'
pickle.dump(svm_model3, open(filename, 'wb'))
null_values = df.isnull().sum()
# Check if there are any null values in the DataFrame
if null_values.any():
    print("There are null values in the DataFrame:")
    print(null_values)
else:
    print("No null values in the DataFrame.")
    df.info(verbose=True)
```

GUI code:

```
import aiproject
import tkinter as tk
from tkinter import messagebox
from tkinter import ttk
import joblib  # Import joblib directly
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import seaborn as sns
class ScrollableFormApp:
    def __init__(self, master):
        self.master = master
        self.master.title("Questionnaire")
        self.header_label = tk.Label(self.master, text="never=1, rarely=2, sometimes=3, often=4,
very often=5",
font=("Arial", 10, "bold"))
        self.header_label.pack(pady=10)
        self.questions = [
```

```
            "Did you think about playing a game all day long?",
            "Did you spend much free time on games?",
            "Have you felt addicted to a game?",
            "Did you play longer than intended?",
            "Did you spend increasing amounts of time on games?",
            "Were you unable to stop once you started playing?",
            "Did you play games to forget about real life?",
            "Have you played games to Realise Stress?",
            "Have you played games to feel better?",
            "Have you failed when trying to reduce game time?",
            "Have others unsuccessfully tried to reduce your game use?",
            "Have you felt bad and angry when you were unable to play?",
            "Have you become stressed when unable to play?",
            "Have you neglected others (e.g., family, friends) because you were playing games?",
            "Did you have fights with others (e.g., family, friends) over your time spent on games?",
            "Have you lost an important relationship because of your gaming activity?",
            "Have you deceived any of your family members or others because the amount of your
gaming activity?",
            "Has your time on games caused sleep deprivation?",
            "Have you lost interests in previous hobbies and other entertainment activities as a
result of your engagement with the game?",
            "Have you neglected other important activities (e.g., school, work, sports) to play
games?",
            "Are you experiencing neck or back pain?",
            "Do you have orthopedic (joint and muscle) problems?",
            "Do you experience eyesight problems?",
            "Do you experience hearing problems?",
            # Add more questions here
        ]
        self.answers = []
        self.create_scrollable_frame()
    def create_scrollable_frame(self):
        self.scrollable_frame = tk.Frame(self.master)
        self.scrollable_frame.pack(fill=tk.BOTH, expand=True)
        self.canvas = tk.Canvas(self.scrollable_frame)
        self.scrollbar = ttk.Scrollbar(self.scrollable_frame, orient="vertical",
command=self.canvas.yview)
        self.scrollable_frame_inner = tk.Frame(self.canvas)
        self.scrollable_frame_inner.bind("<Configure>",
lambda e: self.canvas.configure(scrollregion=self.canvas.bbox("all")))
        self.canvas.create_window((0, 0), window=self.scrollable_frame_inner, anchor="nw")
        self.canvas.configure(yscrollcommand=self.scrollbar.set)
        self.canvas.pack(side="left", fill="both", expand=True)
        self.scrollbar.pack(side="right", fill="y")
```

```
        self.create_question_widgets()
    def create_question_widgets(self):
        for i, question in enumerate(self.questions):
            question_label = tk.Label(self.scrollable_frame_inner, text=question)
            question_label.grid(row=i, column=0, padx=10, pady=10, sticky='w')
            answer_var = tk.IntVar()
            answer_var.set(0)  # Default value
            for option in range(1, 6):  # Options from 1 to 5
                radio_button = tk.Radiobutton(self.scrollable_frame_inner, text=str(option),
variable=answer_var,value=option)
                radio_button.grid(row=i, column=option, padx=10, pady=5, sticky='w')
            self.answers.append(answer_var)
        self.submit_button = tk.Button(self.master, text="Submit", command=self.submit_answer)
        self.submit_button.pack(pady=10)
        # Prediction Result Label
        self.prediction_label = tk.Label(self.scrollable_frame_inner, text="", font=("Arial", 12))
        self.prediction_label.grid(row=len(self.questions), column=0, columnspan=6, pady=10,
sticky='w')
        # Heatmap
        self.show_graph_button = tk.Button(self.master, text="Show Heatmap",
command=self.show_heatmap)
        self.show_graph_button.pack(pady=10)
    def submit_answer(self):
        unanswered = False
        for i, answer_var in enumerate(self.answers):
            answer = answer_var.get()
            if answer == 0:
                unanswered = True
                messagebox.showwarning("Warning", f"Please answer question {i + 1} before
submitting.")
                break
        if not unanswered:
            # Prepare the answers for prediction
            user_answers = [answer_var.get() for answer_var in self.answers]
            # Trim the user answers to the first 20 questions
            user_answers_first_model = user_answers[:20]
            user_answers_second_model = [user_answers[i-1] for i in [1, 5, 7, 8, 15, 16, 17, 19]]
            user_answers_third_model = user_answers[20:]
            # Load models
            model_addiction = joblib.load('Addiction')
            model_mental = joblib.load('mental')
            model_physical = joblib.load('physic')
            # Predictions
            addiction_prediction = model_addiction.predict([user_answers_first_model])
```

```
            mental_prediction = model_mental.predict([user_answers_second_model])
            physical_prediction = model_physical.predict([user_answers_third_model])
            # Convert predictions to "Yes" or "No"
            addiction_result = "Yes" if addiction_prediction[0] == 1 else "No"
            mental_result = "Yes" if mental_prediction[0] == 1 else "No"
            physical_result = "Yes" if physical_prediction[0] == 1 else "No"
            # Display predictions in the root window
            result_message = "RESULT\n\n"
            prediction_message = (
                f"Addiction prediction: {addiction_result}\n"
                f"Addiction accuracy: {aiproject.accuracy}, "
                f"Addiction F1 score: {aiproject.f1}, "
                f"Addiction precision: {aiproject.precision}, "
                f"Addiction Recall: {aiproject.recall}, "
                f"Addiction Confusion Matrix: {aiproject.conf_matrix}\n\n"
                f"Mental health prediction: {mental_result}\n"
                f"Mental accuracy: {aiproject.accuracy5}, "
                f"Mental F1 score: {aiproject.accuracy5}, "
                f"Mental precision: {aiproject.accuracy5}, "
                f"Mental Recall: {aiproject.accuracy5}, "
                f"Mental Confusion Matrix: {aiproject.accuracy5}\n\n"
                f"Physical problems prediction: {physical_result}\n"
                f"Physical accuracy: {aiproject.accuracy6}, "
                f"Physical F1 score: {aiproject.f16}, "
                f"Physical precision: {aiproject.precision6}, "
                f"Physical Recall: {aiproject.recall6}, "
                f"Physical Confusion Matrix: {aiproject.conf_matrix6}"            )
            self.prediction_label.config(text=result_message + prediction_message)
    def show_heatmap(self):
        fig, ax = plt.subplots(figsize=(20,20))
        sns.heatmap(aiproject.df.corr(), annot=True, cmap='RdYlGn', ax=ax)
        ax.set_title('Correlation Heatmap')
        plt.tight_layout()
 # Convert the Matplotlib figure to a tkinter-compatible format and display it in the application
        self.canvas_heatmap = FigureCanvasTkAgg(fig, master=self.master)
        self.canvas_heatmap.draw()
        self.canvas_heatmap.get_tk_widget().pack(side=tk.BOTTOM, fill=tk.BOTH, expand=True)
    def on_configure(self, event):
        self.canvas_hist.configure(scrollregion=self.canvas_hist.bbox("all"))
def main():
    root = tk.Tk()
    root.geometry("1000x800")
    app = ScrollableFormApp(root)
    root.mainloop()
```

```
if __name__ == "__main__":
    main()
```

7. Output:

RESULT

Addiction prediction: Yes
Addiction accuracy: 0.9928057553956835, Addiction F1 score: 0.9919999999999999, Addiction precision: 0.9841269841269841, Addiction Recall: 1.0, Addiction Confusion Matrix: [[76 1]
[0 62]]

Mental health prediction: No
Mental accuracy: 0.9928057553956835, Mental F1 score: 0.9928057553956835, Mental precision: 0.9928057553956835, Mental Recall: 0.9928057553956835, Mental Confusion Matrix: 0.9928057553956835

Physical problems prediction: Yes
Physical accuracy: 1.0, Physical F1 score: 1.0, Physical precision: 1.0, Physical Recall: 1.0, Physical Confusion Matrix: [[70 0]
[0 69]]

FIG- Test-1

RESULT

Addiction prediction: No
Addiction accuracy: 0.9928057553956835, Addiction F1 score: 0.9919999999999999, Addiction precision: 0.9841269841269841, Addiction Recall: 1.0, Addiction Confusion Matrix: [[76 1]
[0 62]]

Mental health prediction: No
Mental accuracy: 0.9928057553956835, Mental F1 score: 0.9928057553956835, Mental precision: 0.9928057553956835, Mental Recall: 0.9928057553956835, Mental Confusion Matrix: 0.9928057553956835

Physical problems prediction: No
Physical accuracy: 1.0, Physical F1 score: 1.0, Physical precision: 1.0, Physical Recall: 1.0, Physical Confusion Matrix: [[70 0]
[0 69]]

FIG- Test-2

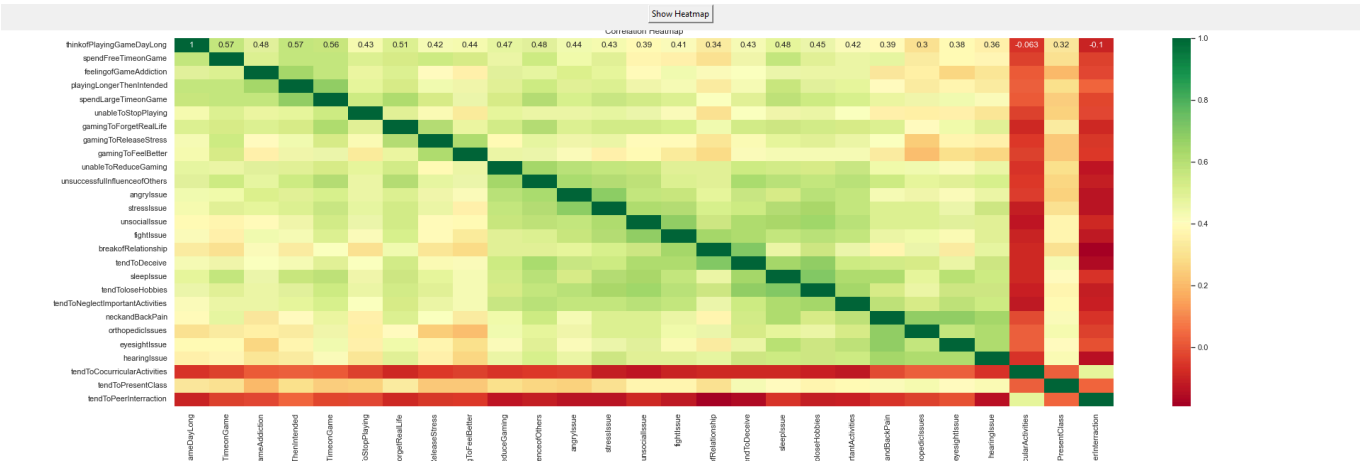


FIG- Heat map

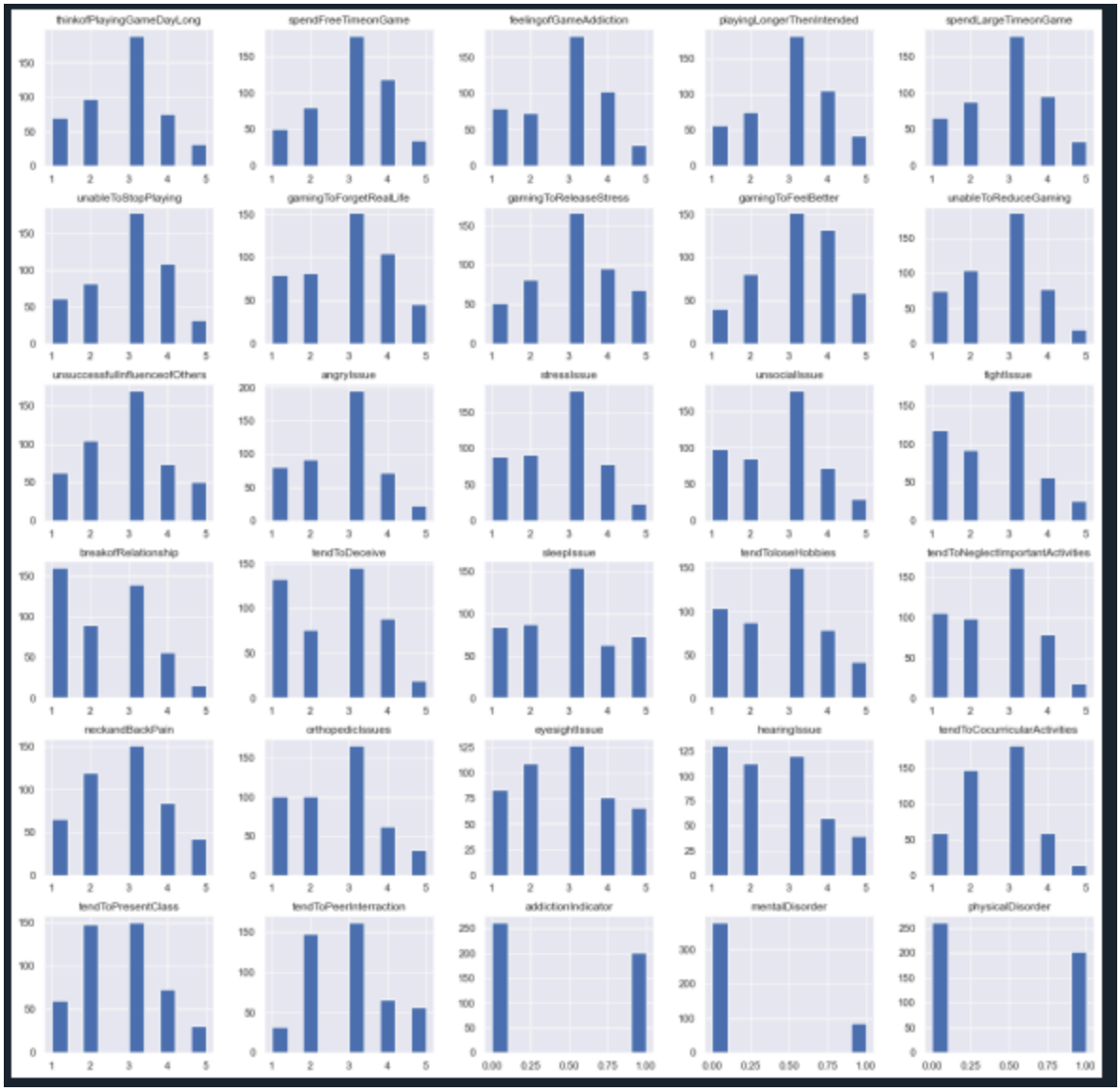


FIG- Histogram of dataset

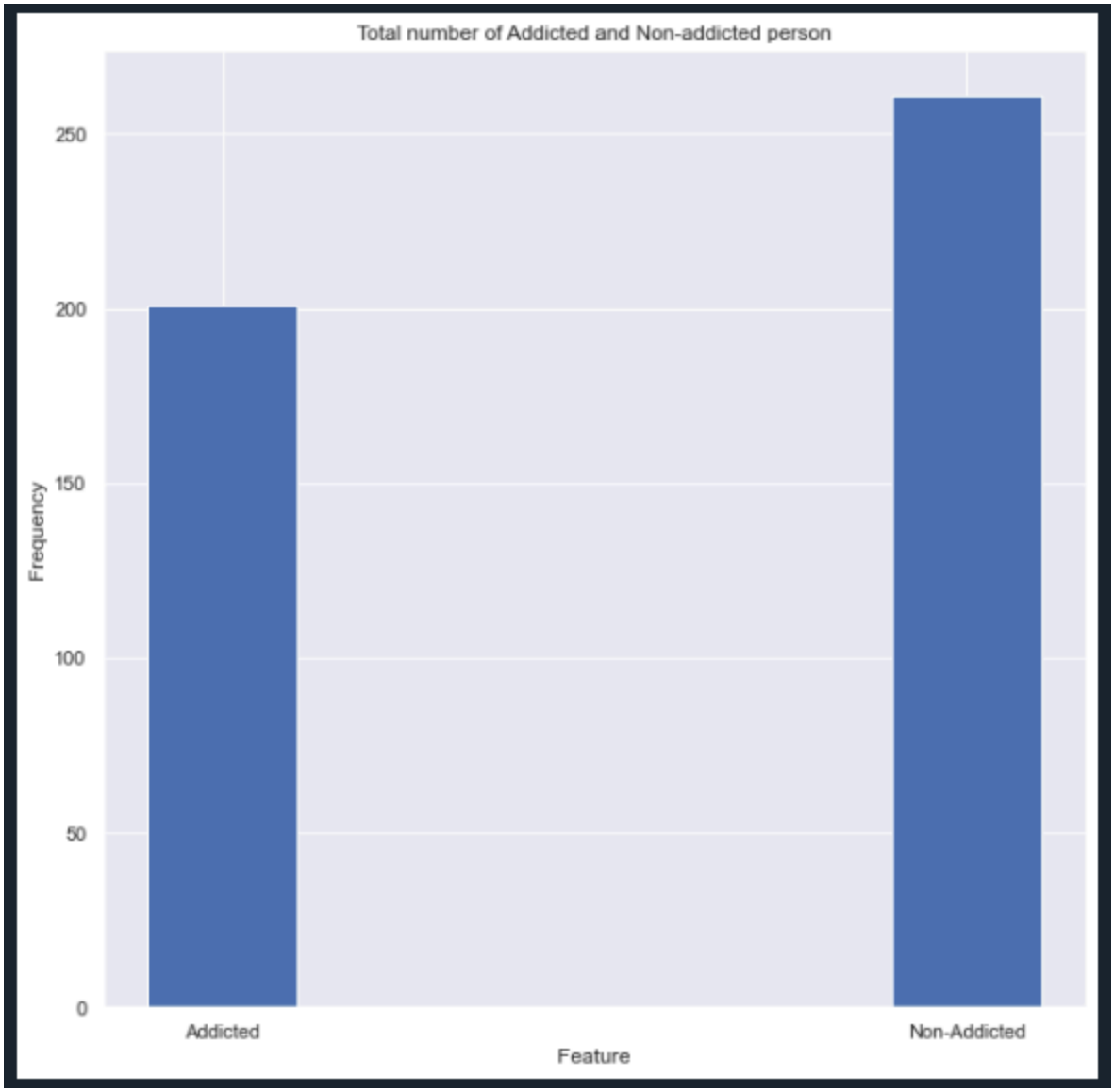


FIG- Barchart

8. **Error Handling:**

Error handling: using Tkinter there is no required function so using this feature in the form for getting the input value from the user. We need to set two variables with 2 values. One is answer_value another one er required_value. In the Initial stage, the answer_value is 0, if the user selects any option it becomes 1. Then, when the user presses the submit button if the answer value is less than required value(value=1) then it will show a message box to fill in the answer. If the answer if answer_value is not less than required_value then show successful.

9. **Conclusion:**

This research shows the accuracy, prediction and graphical views of a dataset that we gathered. This indicates and predicts if a person is an addict or non-addict to games and has mental or physical problems.

	Addiction	Mental	Physical
Accuracy	0.9928057553956835	.09928057553956835	1.0
F1 Score	0 .9919999999999999	0.9928057553956835	1.0
Precision	0.9841269841269841	0.9928057553956835	1.0
Recall	1.0	0.9928057553956835	1.0

For physical health, the value of the predictions is 1.0. It is a false report cause no prediction can be 100% accurate. Other than that the accuracy, F1, Precision, and recall are shown by using SVC.