# Progetto Basi Dati II
# Salute mentale degli studenti

Napolitano Margherita Maria

a.a 2023-2024

## 1 Introduzione

Il progetto svolto è un'applicazione web sviluppata utilizzando Node.js, Express.js e MongoDB per la gestione e l'analisi dei dati relativi alla salute mentale degli studenti. L'applicazione consente agli utenti di inserire, visualizzare, aggiornare e cancellare i dati, nonché di eseguire ricerche e visualizzare statistiche sui dati raccolti. Il dataset di riferimento è prodotto da una ricerca statistica sugli effetti della salute mentale sulla CGPA (media cumulativa dei voti) degli studenti.

## 2 Dataset

Il dataset è stato estratto dalla piattaforma *Kaggle*. I dati riportati sono stati raccolti mediante un sondaggio condotto da moduli Google compilati da studenti universitari al fine di esaminare la loro situazione accademica attuale e la salute mentale. L'analisi è stata condotta in Malesia e i dati sono stati raccolti da Iium (International Icamil University Malaysia).
Link al dataset di riferimento.
Le domande presenti nel form sono:

- Timestamp: String,
- 'Choose your gender': String,
- 'Age': Number,
- 'What is your course?': String,
- 'Your current year of Study': String,
- 'What is your CGPA?': String,
- 'Marital status': String,
- 'Do you have Depression?': String,

- 'Do you have Anxiety?': String,
- 'Do you have Panic attack?': String,
- 'Did you seek any specialist for a treatment?': String.

Nella fase di preprocessing il dataset in formato .csv è stato caricato su MongoDB e convertito in formato JSON.

```
{
  "Timestamp": "8/7/2020 12:02",
  "Choose your gender": "Female",
  "Age": 18,
  "What is your course?": "Engineering",
  "Your current year of Study": "year 1",
  "What is your CGPA?": "3.00 - 3.49",
  "Marital status": "No",
  "Do you have Depression?": "Yes",
  "Do you have Anxiety?": "No",
  "Do you have Panic attack?": "Yes",
  "Did you seek any specialist for a treatment?": "No"
}
```

# 3 Tecnologie utilizzate

Per lo sviluppo backend sono state utilizzate **Node.js, Express.js, MongoDB, Mongoose**.
*Node.js* è un ambiente di runtime JavaScript per l'esecuzione del codice server-side.
*Express.js* è un framework web per Node.js, utilizzato per creare e gestire server e routing.
*MongoDB* è un database NoSQL per la memorizzazione dei dati degli studenti.
*Mongoose* è una libreria di ODM (Object Data Modeling) per MongoDB e Node.js.
Per lo sviluppo frontend, invece, è stato utilizzato **EJS (Embedded JavaScript)**, un motore di template utilizzato per generare le pagine HTML dinamicamente.

# 4 Implemetazione

Il progetto è strutturato nei seguenti componenti principali:

- *Server e Routing:* gestione delle richieste HTTP e delle rotte dell'applicazione.
- *Database:* connessione e interazione con un database MongoDB.
- *Modello (Model):* definizione del modello dei dati e delle operazioni di database.

- *Vista (View):* interfaccia utente, renderizzata utilizzando il motore di template EJS.
- *Controllore (Controller):* logica applicativa e gestione delle operazioni sui dati.

Il motore di template EJS è impostato come motore di visualizzazione e la directory delle viste è configurata per essere views.

```
app.set('view engine', 'ejs');
app.set('views', path.join(__dirname, 'views'));
```

Listing 1: Configurazione del Motore di Template

Si effettua la connessione a un database MongoDB chiamato MentalHealth e gestisce gli eventi di connessione.

```
mongoose.connect('mongodb://localhost:27017/MentalHealth');
const db = mongoose.connection;
db.on('error', (err) => {
    console.error.bind(console, err);
});
db.once('open', () => {
    console.log('Database Connected');
});
```

Listing 2: Connessione al Database

É poi definito il modello MentalHealth utilizzando Mongoose. Il modello specifica la struttura dei documenti nel database.

```
const Schema = new mongoose.Schema({
    Timestamp: String,
    'Choose your gender': String,
    'Age': Number,
    'What is your course?': String,
    'Your current year of Study': String,
    'What is your CGPA?': String,
    'Marital status': String,
    'Do you have Depression?': String,
    'Do you have Anxiety?': String,
    'Do you have Panic attack?': String,
    'Did you seek any specialist for a treatment?': String
}, {
    versionKey: false
});

const Model = mongoose.model('MentalHealth', Schema,
'MentalHealth');
```

Listing 3: Definizione del Modello

## 4.1 Definizione operazioni CRUD

L'obiettivo principale del progetto è effettuare operazioni CRUD (Create, Read, Update, Delete) per interagire con il database.
Definiamo la **Create** per il salvataggio di un nuovo documento.
Viene creato un nuovo oggetto item utilizzando il modello Model definito con Mongoose. Questo oggetto rappresenta un singolo documento nel database. I dati vengono passati come parametri alla funzione saveItem e sono usati per inizializzare i campi dell'oggetto item. Il metodo item.save() salva effettivamente l'oggetto nel database MongoDB.

```
async function saveItem(Timestamp, gender, Age, course,
year, cgpa, marital, depression, anxiety, panic, treatment) {
    const item = new Model({
        Timestamp: Timestamp,
        'Choose your gender': gender,
        'Age': Age,
        'What is your course?': course,
        'Your current year of Study': year,
        'What is your CGPA?': cgpa,
        'Marital status': marital,
        'Do you have Depression?': depression,
        'Do you have Anxiety?': anxiety,
        'Do you have Panic attack?': panic,
        'Did you seek any specialist for a treatment?': treatment
    });
    await item.save();
}
```

Listing 4: Create

La **Read** rende possibile la ricerca degli elementi del database secondo alcuni criteri di ricerca.
Model.find() è la query che trova tutti i documenti nel database corrispondenti al modello Model. Il metodo .exec() esegue la query in modo asincrono e restituisce una promessa. La funzione restituisce un array di tutti gli oggetti trovati nel database.

```
async function findAll() {
    const items = await Model.find({}).exec();
    return items;
}
async function findById(id) {
    const item = await Model.findById(id).exec();
```

```
    return item;
}
async function find(set) {
    console.log(set);
    const items = await Model.find(set).exec();
    return items;
}
}
```

Listing 5: Read

Mediante l'**Update** può essere effettuato l'aggiornamento dei valori di alcuni campi.
Viene costruito un oggetto item con i nuovi valori dei campi che si vogliono aggiornare. Model.findByIdAndUpdate(id, $set: item )è la query che trova e aggiorna un documento nel database basato sull'ID fornito. $set è un operatore di aggiornamento di MongoDB che imposta i nuovi valori dei campi specificati.

```
async function updateItem(id, Timestamp, gender, Age, course,
year, cgpa, marital, depression, anxiety, panic, treatment) {
    const item = {
        Timestamp: Timestamp,
        'Choose your gender': gender,
        'Age': Age,
        'What is your course?': course,
        'Your current year of Study': year,
        'What is your CGPA?': cgpa,
        'Marital status': marital,
        'Do you have Depression?': depression,
        'Do you have Anxiety?': anxiety,
        'Do you have Panic attack?': panic,
        'Did you seek any specialist for a treatment?': treatment
    };
    await Model.findByIdAndUpdate(id, { $set: item }).exec();
}
```

Listing 6: Update

Infine, con la **Delete** si elimina un intero documento tramite l'ID.
Model.findByIdAndDelete(id) è la query che trova e elimina un documento nel database basato sull'ID fornito.

```
async function deleteById(id) {
    await Model.findByIdAndDelete(id).exec();
}
```

Listing 7: Delete

Le funzioni CRUD sopra definite sono chiamate all'interno delle rotte per gestire l'interazione con il database MongoDB.

## 4.2 Routes

Le rotte (routes) in un'applicazione web servono a definire come il server deve rispondere alle diverse richieste HTTP (GET, POST, PUT, DELETE) fatte dai client. Le rotte mappano le richieste degli utenti a specifiche funzioni o logiche di business che risiedono sul server. In questo progetto, le rotte gestiscono diverse operazioni sui dati degli studenti, permettendo agli utenti di inserire, visualizzare, aggiornare, cancellare e cercare dati, nonché di visualizzare statistiche.

La rotta "/" gestisce la richiesta per la home page chiamando la funzione findAll per ottenere tutti i dati dal database e successivamente li passa alla vista homepage per essere visualizzati. Il metodo utilizzato è *GET*.

```
app.get('/', async (req, res) => {
    const items = await findAll();
    const data = {
        items: items,
    };
    res.render('homepage', { data });
});
```

Listing 8: Homepage

La rotta "/form" visualizza il form per l'inserimento dei dati con *GET* mentre con la *POST* gestisce l'invio dei dati dal form, salva un nuovo elemento nel database e reindirizza alla home page.

```
app.get('/form', (req, res) => {
    res.render('form', {});
});

app.post('/form', async (req, res) => {
    const body = req.body;
    await saveItem(Date(), body.gender, body.age, body.course,
    body.year, body.cpga, body.marital, body.depression,
    body.anxiety, body.panic, body.treatment);
    res.redirect('/');
});
```

Listing 9: Form inserimento

La rotta "/item/:id" gestisce la visualizzazione dei dettagli di un singolo elemento identificato dall'ID.

```
app.get('/item/:id', async (req, res) => {
    const id = req.params.id;
    const item = await findById(id);
    const itemFormatted = {
        id: item._id,
```

```
        gender: item['Choose your gender'],
        age: item['Age'],
        course: item['What is your course?'],
        year: item['Your current year of Study'],
        cpga: item['What is your CGPA?'],
        marital: item['Marital status'],
        depression: item['Do you have Depression?'],
        anxiety: item['Do you have Anxiety?'],
        panic: item['Do you have Panic attack?'],
        treatment: item['Did you seek any specialist for a
        treatment?']
    };
    const data = {
        item: itemFormatted
    };
    res.render('item', { data });
});
```

Listing 10: Visualizzazione

La rotta "/update/:id" definisce un endpoint che recupera un elemento dal database usando un id fornito nell'URL, formatta i dati dell'elemento in un formato più leggibile e poi rende un template update passando i dati formattati al template.

```
app.get('/update/:id', async (req, res) => {
    const id = req.params.id;
    const item = await findById(id);
    const itemFormatted = {
        id: item._id,
        gender: item['Choose your gender'],
        age: item['Age'],
        course: item['What is your course?'],
        year: item['Your current year of Study'],
        cpga: item['What is your CGPA?'],
        marital: item['Marital status'],
        depression: item['Do you have Depression?'],
        anxiety: item['Do you have Anxiety?'],
        panic: item['Do you have Panic attack?'],
        treatment: item['Did you seek any specialist for
        a treatment?']
    };
    const data ={
        item: itemFormatted
    };
    res.render('update', { data });
});
```

Listing 11: Aggiornamento

La route "/update/exec/:id" gestisce l'invio dei dati aggiornati, aggiorna l'elemento nel database e reindirizza alla home page.

```
app.post('/update/exec/:id', async (req, res) => {
    let id = req.params.id;
    const body = req.body;
    await updateItem(id, Date(), body.gender, body.age,
    body.course, body.year, body.cpga, body.marital,
    body.depression, body.anxiety, body.panic, body.treatment);
    res.redirect('/');
});
```

Listing 12: Aggiornamento exec

La rotta "/delete/:id" gestisce la cancellazione di un elemento specifico identificato dall'ID e reindirizza alla home page.

```
app.get('/delete/:id', async (req, res) => {
    const id = req.params.id;
    await deleteById(id);
    res.redirect('/');
});
```

Listing 13: Cancellazione

La rotta "/find" gestisce le ricerche all'interno del database basate su criteri specifici forniti dall'utente. É utilizzata un'espressione Regex per permettere una ricerca case insensitive.

```
app.post('/find', async (req, res) => {
    let itemFormatted;
    const isNumeric = (string) => /^[+-]?\d+$/.test(string);
    if(req.body.searchbar===""){
        itemFormatted = {};
    } else if (!isNumeric(req.body.searchbar)) {
        itemFormatted = { [req.body.column]: { $regex: new
        RegExp(`^${req.body.searchbar}$`, 'iy') } };
    } else {
        itemFormatted = { [req.body.column]:
        parseInt(req.body.searchbar) };
    }
    try {
        const items = await find(itemFormatted);
        const data = {
            items: items,
        };
```

```
        res.render('homepage', { data });
    } catch (err) {
        console.error('Error finding items:', err);
        res.status(500).send('Internal Server Error');
    }
});
```

Listing 14: Ricerca

Infine, la rotta "/statistiche" visualizza le statistiche basate sui dati raccolti.

```
app.get('/statistiche', async (req, res) => {
    const items = await findAll();
    const data = {
        items: items,
    };
    res.render('statistiche', { data });
});
```

Listing 15: Statistiche

## 4.3 Interfaccia utente

L'interfaccia implementata è user-friendly in quanto è semplice da utilizzare. Troveremo una pagina iniziale in cui è presente la lista dei form presente nel database (Fig.1), la possibilità di ricercare in base ad alcuni criteri come il sesso, l'età o il corso frequentato (Fig.2) e i pulsante per compilare un nuovo form (Fig.3) e per visualizzare le statistiche dei dati.

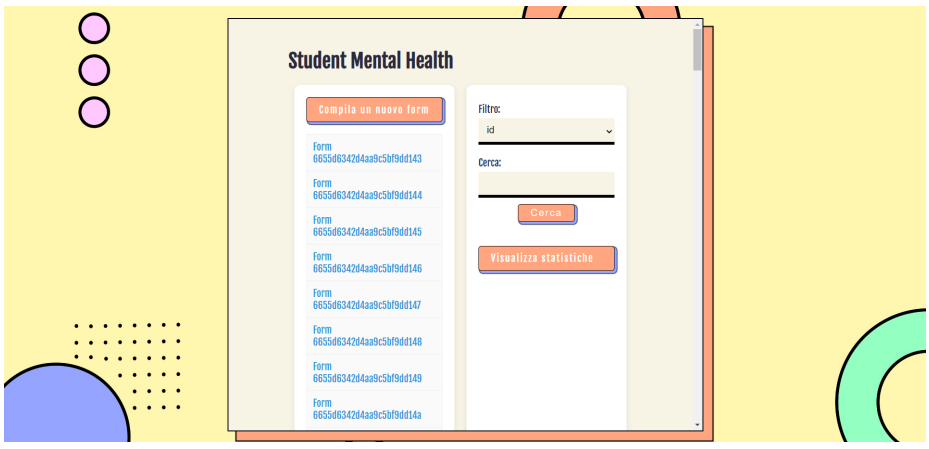


Figure 1: Homepage

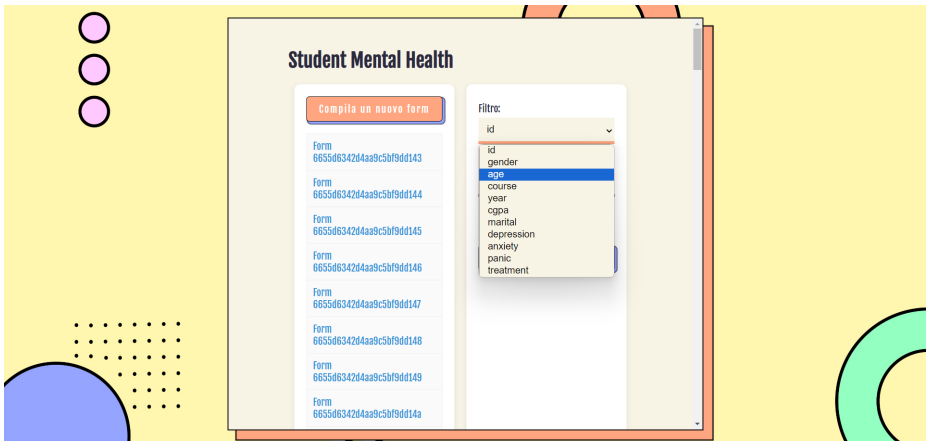


Figure 2: Ricerca


Figure 3: Compilazione form

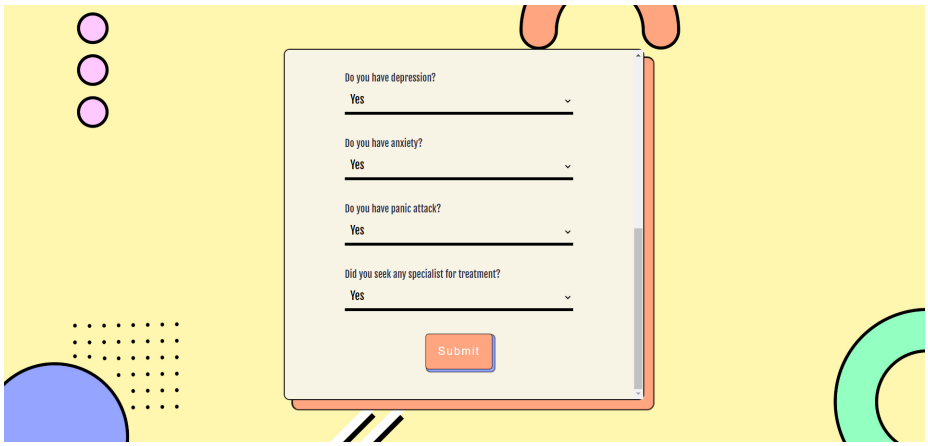


Figure 4: Compilazione form

Cliccando sul nome del form di nostro interesse, sarà possibile visualizzare i dati relativi ad esso in una pagina statica (Fig.5), modificare (Fig. 6) i valori o eliminare l'intero documento.


Figure 5: Visualizzazione form


Figure 6: Update

# 5 Analisi statistica

Tramite l'applicativo è possibile accedere alla visualizzazione dell'analisi statistica dei dati in esame. Il range di età dei partecipanti allo studio è tra i 18 e i 24 anni e si evince che il maggior numero di essi ha 18 anni e di conseguenza frequenta il primo anno di università (Fig. 8).

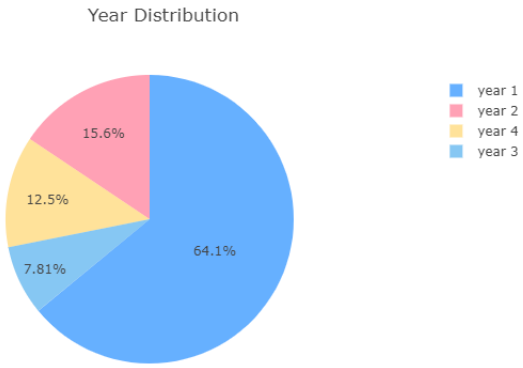


Figure 7: Anno frequentato

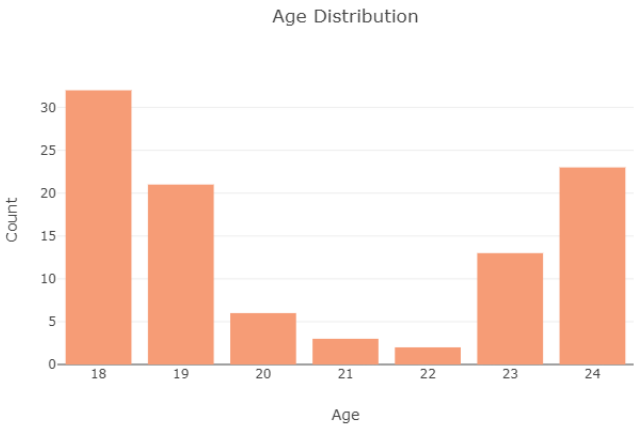


Figure 8: Età

I corsi più frequentati sono Ingegneria e BCS (Bachelor of Computer Science) (Fig.9).

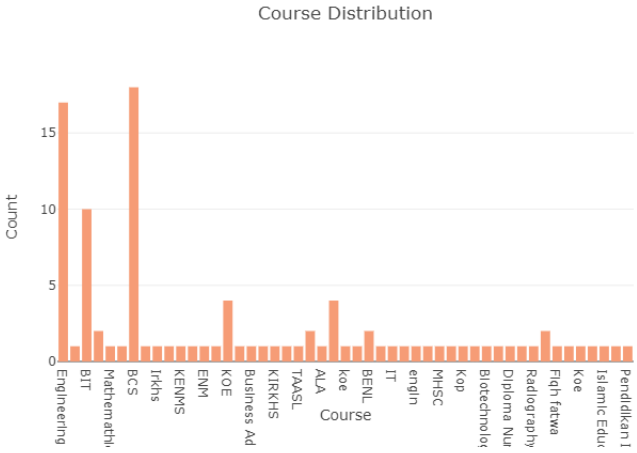


Figure 9: Corso frequentato

Nel nostro dataset abbiamo più studenti donne 74,3% rispetto agli studenti maschi 25,7% (Fig. 11).

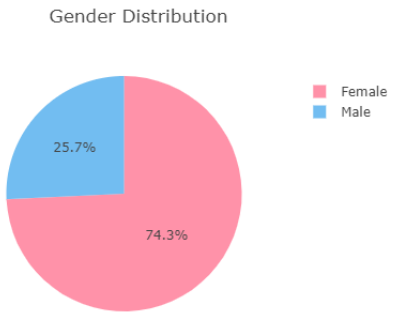


Figure 10: Genere

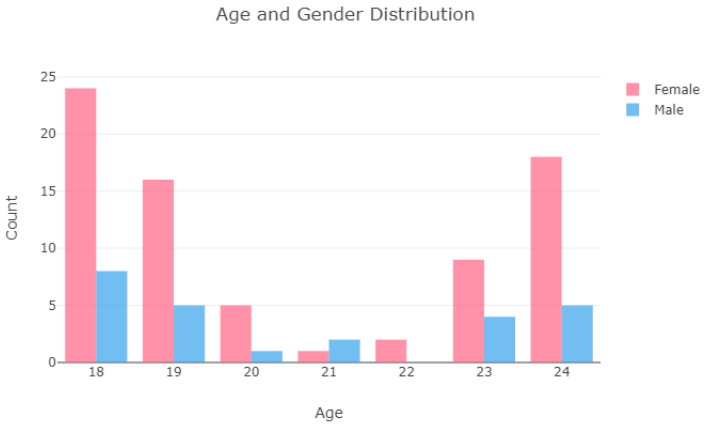


Figure 11: Genere e età

I dati relativi alle percentuali di depressione, ansia, attacchi di panico e aiuto di uno specialista sono: (Fig. 12)

- **depressione:** 35 studenti su 101;
- **ansia:** 34 su 101;
- **attacco di panico:** 33 su 101;
- **trattamento:** 6 studenti su 101 hanno un trattamento con uno specialista.

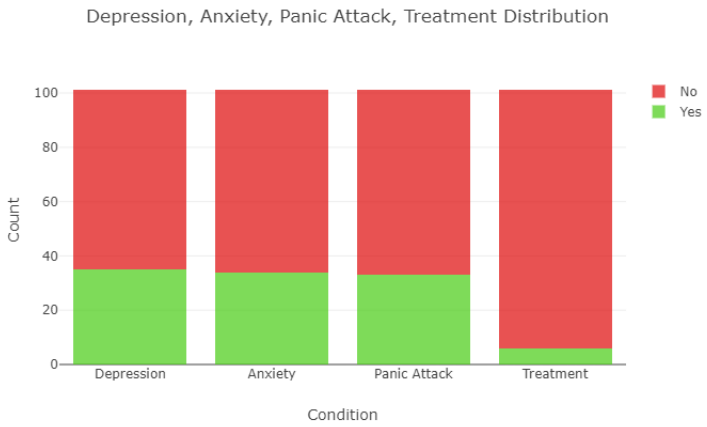


Figure 12: Statistiche depressione, ansia, attacchi di panico e trattamento

L'obiettivo principale dell'analisi è studiare l'effetto che la CGPA ha sulla salute mentale degli studenti. (Fig.15)

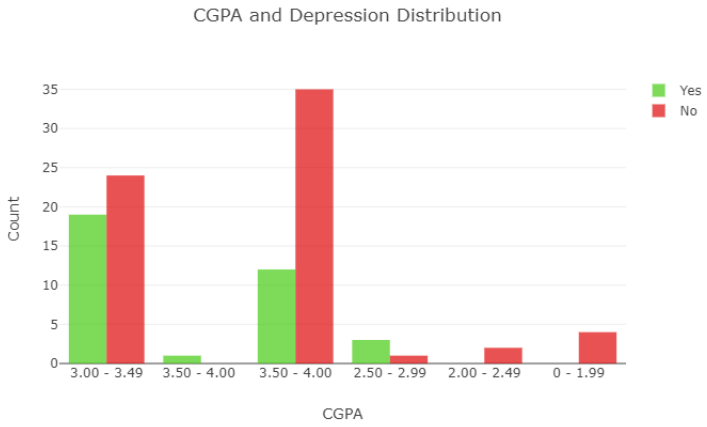


Figure 13: CGPA e depressione

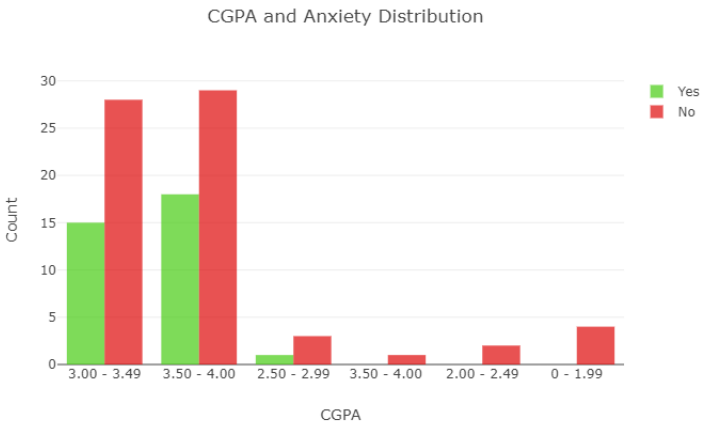


Figure 14: CGPA e ansia

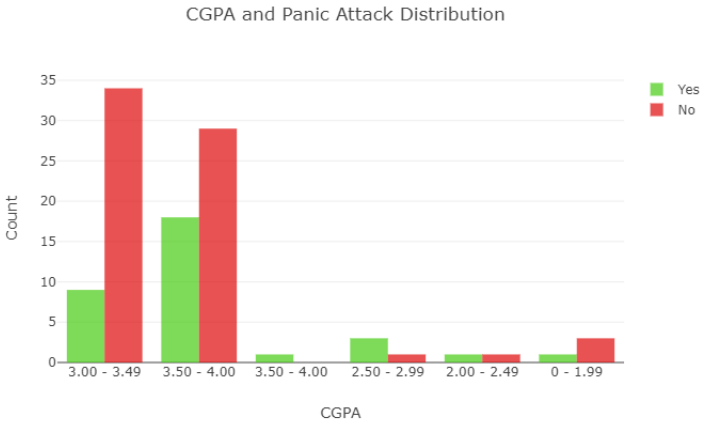


Figure 15: CGPA e attacchi di panico

Si evince che il numero di studenti che non soffre di depressione, ansia o attacchi di panico è più basso rispetto agli studenti che ne soffrono.

In conclusione, possiamo dire che la CGPA non ha un impatto significativo sulla salute mentale degli studenti.