



PEARL

By

Abdelrahman alaa 20-02024

Mostafa samir 20-00679

Mohammed sanad 20-00265

Hesham mohamed 20-01524

Ahmed elsayed 20-01733

Ahmed hesham 20-01992

Supervised by

Dr. basem elhady

Eng. salma elzayat

Department of Information Technology

Egyptian E-Learning University

June-2024

Table of content

Contents

5	Introduction to Pearl
5	What is Pearl?
5	Key Features
7	Benefits of Using Pearl
7	Getting Started
9	Motivation Behind Pearl
9	Bridging the Gap Between Discovery and Booking
9	Enhancing Trust and Transparency
10	Supporting Local Economies
10	Catering to Diverse Needs
10	Streamlining the User Experience
11	Encouraging Community Engagement
11	Successful Similar Apps
11	Yelp
12	Airbnb
13	TripAdvisor
13	Combining Strengths
16	Log In and Registering
22	Front End Development with React.js, Next.js, and Tailwind CSS
22	Overview
22	Reusable Components
24	Server-Side Rendering (SSR) with Next.js
24	Key Features of Next.js in Pearl
25	Styling with Tailwind CSS
27	Technologies Used
27	Database Schema
27	Why Prisma?
29	Integrating Prisma with MongoDB
30	User Authentication with NextAuth.js
31	Environment Variables
31	Integrating Authentication with Prisma

32	Image Upload Using Cloudinary
32	Integrating Cloudinary with Pearl
37	UI Components and Design System
38	User Flows and Navigation
39	User Feedback and Continuous Improvement
39	Conclusion
57	Step 4: Data Pre-processing
58	Evaluating sparsity
59	Step 5: Item-item Recommendations with k-Nearest Neighbors
60	Step 6: Handling the cold-start problem
61	businesses
61	Creating a business finder function
62	Step 7: Dimensionality Reduction with Matrix Factorization (advanced)

CHAPTER 1

INTRODUCTION

Introduction to Pearl

Welcome to Pearl, your ultimate platform for discovering and sharing the best local experiences, services, and accommodations. Pearl seamlessly combines the features of Yelp and Airbnb to provide users with a comprehensive, user-friendly interface for exploring everything a city has to offer, from top-rated restaurants and unique shops to cozy vacation rentals and distinctive local experiences.

What is Pearl?

Pearl is designed to be a one-stop destination for both locals and travelers looking to find the best that any city has to offer. Whether you are searching for a new dining experience, a unique activity, or a comfortable place to stay, Pearl connects you with a diverse community of users who share their reviews, ratings, and personal experiences.

Key Features

Discover Local Businesses and Services

- **User Reviews and Ratings:** Read honest reviews and ratings from fellow users to help you make informed decisions about where to eat, shop, and explore.

- **Detailed Listings:** Access comprehensive details about businesses, including menus, hours of operation, photos, and contact information.
- **Search and Filters:** Utilize powerful search and filter options to find exactly what you're looking for, whether it's a specific type of cuisine, a particular service, or a highly-rated experience.

Find Unique Accommodations

- **Vacation Rentals:** Browse a wide variety of rental properties, from cozy apartments and rustic cabins to luxurious villas and unique stays.
- **Host Profiles:** Learn about the hosts, read reviews from previous guests, and view detailed property descriptions to find the perfect match for your stay.
- **Booking and Payments:** Securely book your stay directly through Pearl, with flexible payment options and transparent pricing.

Share Your Experiences

- **Write Reviews:** Share your experiences with the Pearl community by writing reviews for businesses and accommodations you've visited.

- **Upload Photos:** Enhance your reviews with photos to give others a better sense of what to expect.
- **Create Lists:** Curate and share lists of your favorite places, from top brunch spots to hidden gems.

Benefits of Using Pearl

- **Trusted Community:** Pearl's community-driven approach ensures that all reviews and ratings are genuine and trustworthy, helping you make the best choices.
- **Convenience:** Find everything you need in one place, without having to switch between multiple apps or websites.
- **Personalized Recommendations:** Receive personalized suggestions based on your interests, previous reviews, and browsing history.

Getting Started

Getting started with Pearl is easy. Simply download the app from the App Store or Google Play, create an account, and start exploring. Whether you're planning your next trip or just looking for a new local favorite, Pearl is here to help you discover and enjoy the best experiences around you.

CHAPTER 2
MOTIVATION ,
SUCCESSFUL SIMILAR APPS

Motivation Behind Pearl

The motivation behind creating Pearl is rooted in a desire to empower local businesses and provide users with a holistic, user-friendly platform for discovering and enjoying the best local experiences. By combining the features of successful review and accommodation apps like Yelp and Airbnb, Pearl aims to address several key needs and trends in the modern consumer landscape:

Bridging the Gap Between Discovery and Booking

In today's digital age, people increasingly rely on online reviews and recommendations to make informed decisions about local businesses, from restaurants and shops to service providers. However, these services are often fragmented across different platforms. Pearl was created to bridge this gap by offering a unified platform where users can discover local businesses and book unique accommodations seamlessly.

Enhancing Trust and Transparency

With the proliferation of online information, ensuring the authenticity and reliability of reviews and listings is more important than ever. Pearl

leverages a community-driven approach to foster trust and transparency, providing users with honest, detailed reviews and ratings from real customers. This empowers users to make confident choices based on genuine feedback about local businesses.

Supporting Local Economies

Local businesses are the backbone of vibrant communities. Pearl is designed to support these businesses by providing them with a platform to reach a broader audience and attract new customers. By highlighting local businesses and encouraging user reviews, Pearl helps drive traffic and revenue to small, independently-owned enterprises.

Catering to Diverse Needs

Modern consumers have diverse needs and preferences, from seeking out the best local coffee shop to finding a unique vacation rental. Pearl was designed to cater to these varied interests, offering a comprehensive directory of local businesses and accommodations to suit every taste and budget. This inclusivity ensures that there is something for everyone on Pearl.

Streamlining the User Experience

Navigating multiple apps for different purposes can be cumbersome. Pearl aims to streamline the user

experience by integrating discovery, review, and booking functionalities into a single, intuitive platform. This convenience saves users time and enhances their overall experience with local businesses.

Encouraging Community Engagement

At its core, Pearl is about building a vibrant community of users who share their experiences and insights about local businesses. By encouraging users to write reviews, upload photos, and create curated lists, Pearl fosters a sense of community and collaboration, making it more than just an app, but a platform for connection and shared experiences.

Successful Similar Apps

Pearl draws inspiration from several successful apps in the market that have set benchmarks in their respective domains by supporting local businesses:

Yelp

Overview: Yelp is a widely popular platform for discovering local businesses and services, known for its extensive user reviews and ratings.

Key Features:

- Comprehensive local business listings with detailed information.
- User-generated reviews and photos.
- Search and filter options for finding specific types of local businesses.

Success Factors:

- Strong community engagement focused on local businesses.
- Reliable and detailed reviews.
- User-friendly interface.

Airbnb

Overview: Airbnb revolutionized the accommodation industry by allowing individuals to rent out their properties to travelers, offering a wide range of unique and personalized lodging options, often provided by local hosts.

Key Features:

- Diverse listings of rental properties worldwide.
- Detailed host profiles and property descriptions.
- Secure booking and payment system.

Success Factors:

- Wide variety of unique and affordable accommodations provided by locals.

- Trust-building features like reviews and host verifications.
- Seamless booking experience.

TripAdvisor

Overview: TripAdvisor is a comprehensive travel platform that offers reviews, photos, and booking options for hotels, restaurants, and attractions worldwide, with a strong emphasis on local experiences.

Key Features:

- User-generated reviews and ratings for various travel-related services, including local businesses.
- Ability to book accommodations and activities directly.
- Extensive travel forums and community interactions.

Success Factors:

- Extensive database of reviews and photos.
- Integration of discovery and booking functionalities.
- Active and engaged user community focused on local experiences.

Combining Strengths

By combining the strengths of these successful apps, Pearl aims to create a versatile and comprehensive platform that addresses the evolving needs of modern consumers while supporting local businesses. The goal is to offer a seamless experience that makes discovering, reviewing, and booking local businesses and accommodations effortless and enjoyable.

With Pearl, we aspire to enhance how people explore and experience the world around them, creating a platform that not only meets but exceeds user expectations, and significantly supports local businesses in the process.

CHAPTER 3 :

FEATURES

Log In and Registering

Easy Sign-Up and Authentication

- **Simple Registration:** Quickly create an account using your email address and a secure password.
- **Social Authentication:** Sign up or log in effortlessly using your Google or GitHub accounts. This feature ensures a smooth and hassle-free authentication process, allowing you to get started with Pearl in no time.

×

Register

Welcome to Pearl


Create an account!

Email


Name

Password

Continue



Continue with Google

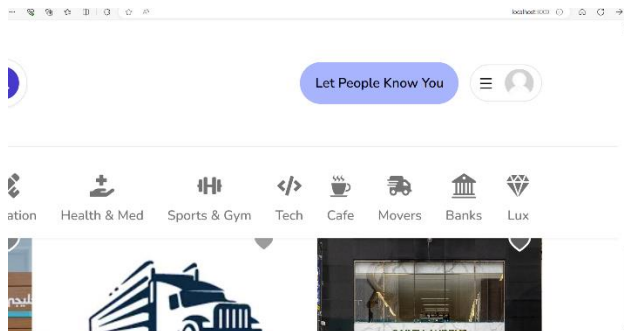


Continue with Github

[Already have an account? Log in](#)


Business and Property Listings


- **Simple Submission:** Easily add your local business or rental property to Pearl by filling out a straightforward submission form.
- **Detailed Descriptions:** Provide detailed information about your business or property, including descriptions, amenities, hours of operation, contact information, and high-quality photos.





show your work


Which of these best describes your place?
Pick a category



Beauty & Spas


Education


Health & Med


Sports & Gym


Tech


Cafe

Next

show your work

Where is your place located?
Help guests find you!


eg

BA Bosnia and Herzegovina, Europe

EG Egypt, Africa

ME Montenegro, Europe

SN Senegal, Africa



Back

Next

if you are listing a house or a property you can add some more details

×

show your work

Share some basics about your place
What amenities do you have?

Guests

How many guests do you allow?

−

4

+

Rooms

How many rooms do you have?

−

2

+

Bathrooms

How many bathrooms do you have?

−

1

+

Back


Next

you can add pictures of your local business/property

×

show your work

Add a photo of your place
Show guests what your place looks like!


Click to upload

Back

Next

add the title and description

×

show your work

How would you describe your place?
Short and sweet works best!

Title

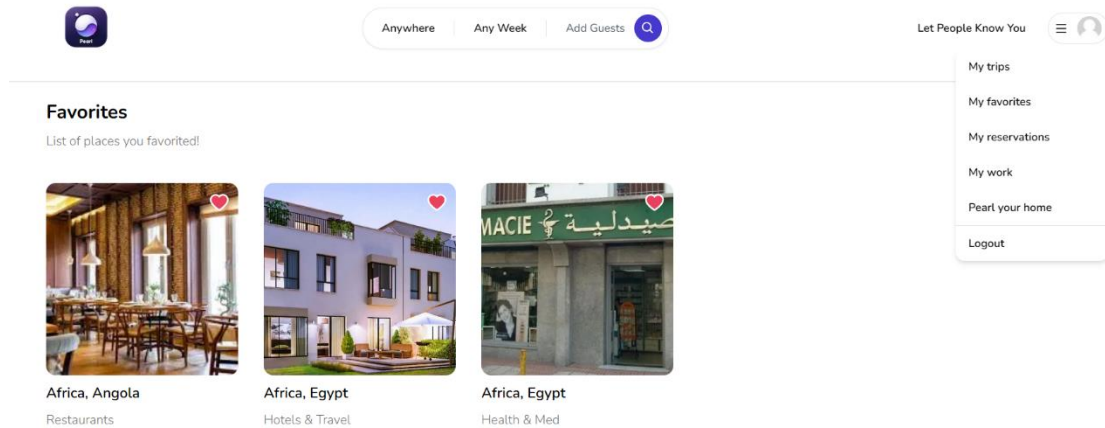
Description

Back

Next

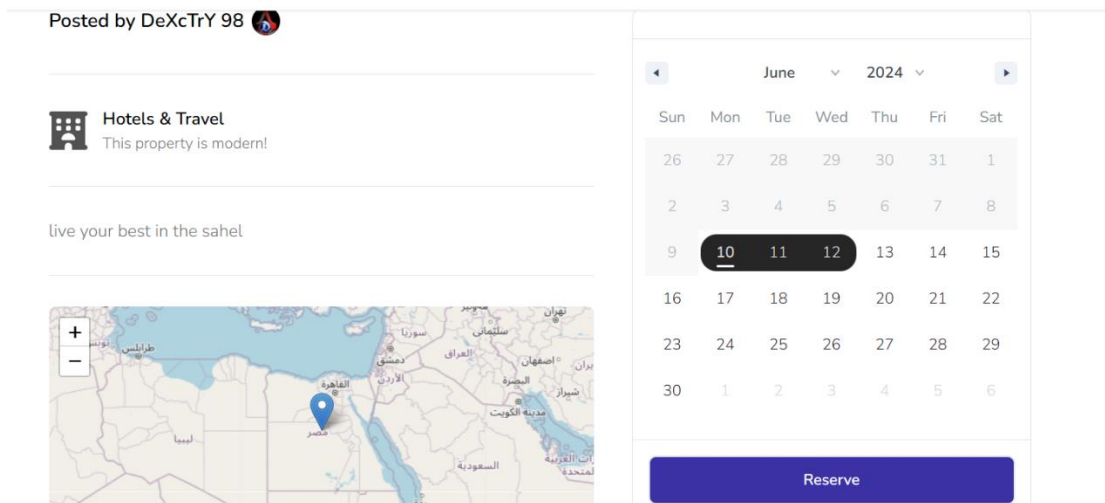
Favorites

This feature allows you to create personalized lists of your favorite local businesses, accommodations, and experiences, making it easy to keep track of the places you love



Reserve

our gateway to seamless bookings and reservations for local businesses and unique accommodations. Whether you're planning a dinner at a top-rated restaurant, booking a cozy stay for the weekend, or reserving a spot for a popular local event.



My trips

this feature allows you to effortlessly manage all your travel plans in one place, from accommodations and dining reservations to activities and local attractions.

Trips

Where you've been and where you're going



Africa, Egypt

Jun 10, 2024 - Jun 12, 2024

Cancel reservation

Properties

Whether you own a restaurant, a boutique shop, a vacation rental, or a unique stay, this section provides the tools you need to showcase your offerings and connect with a broader audience.

Properties

List of your properties



Africa, Egypt

Automotive

Delete property

CHAPTER 4 :

FRONT END

Front End Development with React.js, Next.js, and Tailwind CSS

In this chapter, we'll delve into how the front end of Pearl is crafted using React.js, Next.js, and Tailwind CSS. These technologies enable us to create a performant, scalable, and visually appealing web application. We'll focus on the importance of reusable components and the benefits of server-side rendering (SSR).

Overview

- **React.js:** A JavaScript library for building user interfaces, emphasizing the creation of reusable UI components.
- **Next.js:** A React framework that supports server-side rendering and static site generation, enhancing performance and SEO.
- **Tailwind CSS:** A utility-first CSS framework that promotes rapid and consistent UI development.

Reusable Components

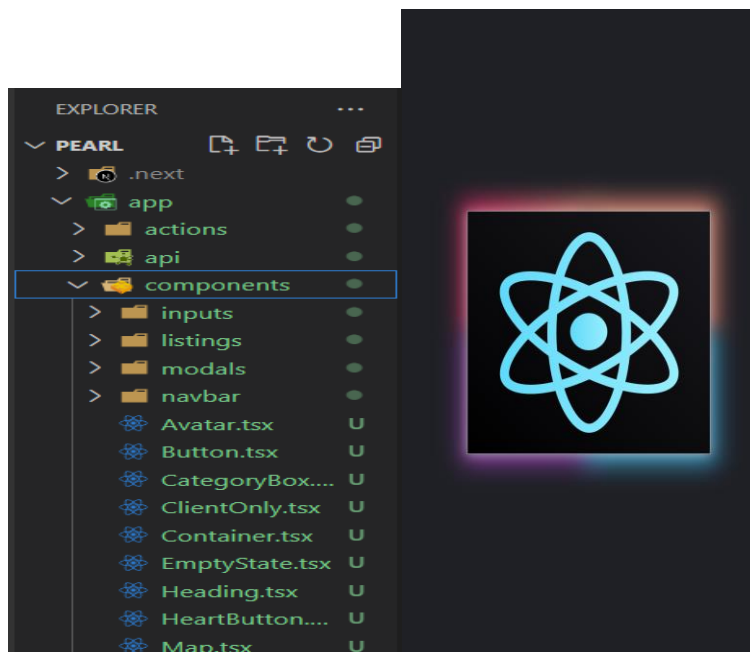
Reusable components are at the heart of React.js development, allowing us to build modular and maintainable code. In Pearl, we leverage this concept to create components that can be used across different parts of the application.

Benefits of Reusable Components

- **Consistency:** Reusable components ensure a consistent look and feel throughout the application.
- **Efficiency:** Once a component is built, it can be reused multiple times, reducing development time and effort.
- **Maintainability:** Modular components are easier to maintain and update, as changes in one component do not affect others.

Examples of Reusable Components in Pearl

- **BusinessCard:** A component to display information about a local business, such as its name, description, and image.
- **PropertyCard:** A component to showcase properties, including details like title, location, price, and image.
- **Navbar and Footer:** Common layout components that appear on every page, providing navigation and site information.



Server-Side Rendering (SSR) with Next.js

Server-side rendering is a technique where HTML is generated on the server for each request. Next.js makes SSR straightforward, enhancing performance and improving search engine optimization (SEO).

The image shows the Next.js logo, which consists of the word "NEXT" in a bold, sans-serif font, followed by ".JS" in a smaller, regular weight of the same font. The logo is centered within a light gray rectangular background.

Benefits of SSR

- **Improved Performance:** SSR delivers fully rendered pages to the client, reducing the time to first meaningful paint.
- **Better SEO:** Search engines can more easily index server-rendered content, improving the discoverability of your site.
- **Enhanced User Experience:** Users see content faster as it is rendered on the server before being sent to their browser.

Key Features of Next.js in Pearl

- **Dynamic Data Fetching:** Fetch data on the server for each request, ensuring users always see the most up-to-date information.
- **Static Site Generation (SSG):** Pre-render pages at build time for static content, combining the benefits of static and dynamic sites.
- **API Routes:** Create serverless functions directly within the Next.js app to handle backend logic without the need for a separate server.

Styling with Tailwind CSS

Tailwind CSS allows for rapid development by providing utility classes that can be combined to create complex designs directly within your HTML.



Benefits of Using Tailwind CSS

- **Consistent Design:** Utility classes ensure a consistent design language across the application.
- **Speed:** Developers can build and modify designs quickly without writing custom CSS.
- **Flexibility:** Tailwind's utility-first approach allows for extensive customization without the need for complex stylesheets.

CHAPTER 5:

DATABASE & BACKEND

In this chapter, we'll explore the backend architecture and database design for Pearl. The backend is responsible for handling data storage, user authentication, and business logic, ensuring that the application runs smoothly and securely. We will discuss the technologies used, the database schema, and how the backend integrates with the front end.

Technologies Used

- **Node.js:** A JavaScript runtime built on Chrome's V8 engine, used for building scalable network applications.
- **Next.js:** A React framework that supports server-side rendering and static site generation, which we also use for the backend APIs.
- **MongoDB:** A NoSQL database known for its flexibility and scalability.
- **Prisma:** A modern database toolkit for Node.js and TypeScript, providing an ORM (Object-Relational Mapping) layer to interact with MongoDB.
- **NextAuth.js:** A complete open-source authentication solution for Next.js applications, providing secure and flexible user authentication.

Database Schema

The database schema is designed to support the core functionalities of Pearl, including user management, business listings, property management, reviews, and bookings.

Why Prisma?

Prisma is chosen as the database toolkit for Pearl due to its modern, developer-friendly features that streamline database interactions, increase productivity, and ensure type safety. Here are several reasons why Prisma stands out:

1. Type Safety

Prisma automatically generates TypeScript types based on the database schema. This ensures that all database queries are type-safe, reducing runtime errors and improving code quality. Developers receive immediate feedback from their IDE if they make a mistake, leading to fewer bugs and more robust code.

2. Ease of Use

Prisma provides a declarative schema definition language that simplifies defining and managing database schemas. With Prisma, developers can define their data models in a human-readable format and automatically generate the corresponding database

migrations. This streamlined workflow makes it easier to maintain and evolve the database schema over time.

3. Productivity Boost

Prisma's intuitive and expressive API allows developers to write database queries in a more readable and maintainable way compared to raw SQL. It abstracts away the complexity of database operations, enabling developers to focus on business logic rather than the intricacies of database management. This leads to faster development cycles and increased productivity.

4. Advanced Query Capabilities

Prisma supports complex querying capabilities, including nested reads, writes, and filtering. These advanced features enable developers to perform powerful database operations with ease, ensuring that even the most complex use cases can be handled efficiently.

5. Database Agnosticism

Prisma supports multiple databases, including PostgreSQL, MySQL, SQLite, and MongoDB. This flexibility allows developers to choose the database that best fits their application needs and potentially switch databases with minimal code changes in the future. For Pearl, Prisma's support for MongoDB is particularly valuable, enabling the use of a scalable and flexible NoSQL database.

6. Performance Optimization

Prisma's query engine is highly optimized for performance, ensuring that database operations are executed efficiently. This is crucial for applications like Pearl, where fast response times and efficient data retrieval are critical for a smooth user experience.

7. Strong Community and Ecosystem

Prisma has a vibrant and active community, which means a wealth of resources, plugins, and integrations are available. The strong ecosystem around Prisma ensures that developers can find support and tools to extend and enhance their applications.

Integrating Prisma with MongoDB

Prisma provides a type-safe ORM for MongoDB, allowing us to define and interact with our database schema using a declarative schema definition language.

```
DATABASE_URL="mongodb+srv://avdul:avdul@cluster0.n80pyao.mongodb.net/test"
```

Here is an example of how the schema for Users, Businesses, Properties, and Bookings might look in Prisma:

```
Generate
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mongodb"
  url      = env("DATABASE_URL")
}

model User {
  id          String      @id @default(auto()) @map("_id") @db.ObjectId
  name        String?
  email       String?     @unique
  emailVerified DateTime?
  image       String?
  hashedPassword String?
  createdAt   DateTime    @default(now())
  updatedAt   DateTime    @updatedAt
  favoriteIds String[]     @db.ObjectId

  accounts    Account[]
  listings    Listing[]
  reservations Reservation[]
}
```

```
schema.prisma > ...

model Account {
  id          String @id @default(auto()) @map("_id") @db.ObjectId
  userId      String @db.ObjectId
  type        String
  provider     String
  providerAccountId String
  refresh_token String? @db.String
  access_token String? @db.String
  expires_at   Int?
  token_type   String?
  scope        String?
  id_token     String? @db.String
  session_state String?

  user User @relation(fields: [userId], references: [id], onDelete: Cascade)
}
@@unique([provider, providerAccountId])

model Listing {
  id          String @id @default(auto()) @map("_id") @db.ObjectId
  title       String
  description  String
  imageSrc    String
  createdAt   DateTime @default(now())
  category    String
  numCount    Int
}
```

User Authentication with NextAuth.js

User authentication is a critical component of Pearl, ensuring secure access and personalized experiences for users. NextAuth.js is an excellent choice for handling authentication in a Next.js application due to its flexibility, ease of integration, and support for multiple authentication providers. This section covers how to implement sign-in and user authentication using NextAuth.js.

Why NextAuth.js?

1. **Multiple Authentication Providers:** NextAuth.js supports a wide range of providers, including Google, Facebook, GitHub, and Email, making it easy to offer users multiple ways to authenticate.
2. **Secure and Flexible:** NextAuth.js handles secure session management, including JWT (JSON Web Tokens) and database sessions. It also provides customizable callbacks to manage authentication workflows.
3. **Seamless Integration with Next.js:** Built specifically for Next.js, NextAuth.js integrates seamlessly with Next.js API routes and pages, simplifying the implementation process.
4. **Developer Friendly:** With a simple configuration setup, detailed documentation, and an active community, NextAuth.js makes implementing authentication straightforward and efficient.

API route for NextAuth.js in the `pages/api/auth` directory.

pages/api/auth/[...nextauth].js:

```
TS [...nextauth].ts U X
pages > api > auth > TS [...nextauth].ts > .
1 import { PrismaAdapter } from "@next-auth/prisma-adapter";
2 import { AuthOptions } from "next-auth";
3 import GithubProvider from "next-auth/providers/github"
4 import GoogleProvider from "next-auth/providers/google"
5 import CredentialsProvider from "next-auth/providers/credentials"
6 import prisma from "@app/libs/prismadb";
7 import bcrypt from "bcrypt"
8 import NextAuth from "next-auth";
9
10 export const authOptions: AuthOptions = {
11   adapter: PrismaAdapter(prisma),
12   providers: [
13     GithubProvider({
14       clientId: process.env.GITHUB_ID as string,
15       clientSecret: process.env.GITHUB_SECRET as string
16     }),
17     GoogleProvider({
18       clientId: process.env.GOOGLE_CLIENT_ID as string,
19       clientSecret: process.env.GOOGLE_CLIENT_SECRET as string
20     }),
21     CredentialsProvider({
22       name: 'credentials',
23       credentials: {
24         email: { label: 'email', type: 'text' },
25         password: { label: 'password', type: 'password' }
26       },
27       async authorize(credentials) {
28         if (!credentials?.email || !credentials?.password) {
```

Environment Variables

the required environment variables in a `.env` file:

Google authentication

Github authentication

```
DATABASE_URL="mongodb+srv://avdul:avdul@cluster0.n80pyao.mongodb.net/test"

NEXTAUTH_SECRET="NEXTAUTH_SECRET"

GITHUB_ID=ba525a01ef075897e151
GITHUB_SECRET=eeda08f83cacfaa67d5d7ef997777c8667379a7e

GOOGLE_CLIENT_ID=688658156675-
oqu14uurv126bkf4536q6o7juj0hvlslj.apps.googleusercontent.com
GOOGLE_CLIENT_SECRET=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

NEXT_PUBLIC_CLOUDINARY_CLOUD_NAME ="digurnehj"
```

Integrating Authentication with Prisma

Prisma integrates seamlessly with NextAuth.js, ensuring that user data is stored and managed efficiently in the MongoDB database.

1. Prisma Schema for Users

Update the Prisma schema to include user models

2. Running Migrations

Generate and apply migrations to update the database schema:

```
npx prisma migrate dev --name init
```

Image Upload Using Cloudinary

Image uploads are a crucial feature for Pearl, allowing users to add photos to their business listings, properties, and reviews. Cloudinary is a powerful cloud-based service that provides comprehensive image and video management capabilities, making it an excellent choice for handling image uploads and storage. This section covers how to integrate Cloudinary into Pearl for seamless image upload functionality.

Why Cloudinary?

1. **Scalability:** Cloudinary handles image storage, transformation, and delivery at scale, ensuring efficient management of large volumes of images.
2. **Performance:** With built-in CDN (Content Delivery Network) integration, Cloudinary ensures fast image delivery, improving the user experience.
3. **Transformation and Optimization:** Cloudinary provides on-the-fly image transformations, optimizing images for different devices and resolutions, which helps reduce load times.
4. **Ease of Use:** Cloudinary offers a straightforward API for uploading and managing images, making it easy to integrate into any web application.

Integrating Cloudinary with Pearl

Step 1: Setup Cloudinary Account

1. **Create an Account:** Sign up for a Cloudinary account at cloudinary.com.
2. **Get API Credentials:** After creating an account, obtain your Cloudinary cloud name, API key, and API secret from the Cloudinary dashboard.

Step 2: Install Cloudinary Package

Install the Cloudinary package to handle image uploads in your Next.js application.

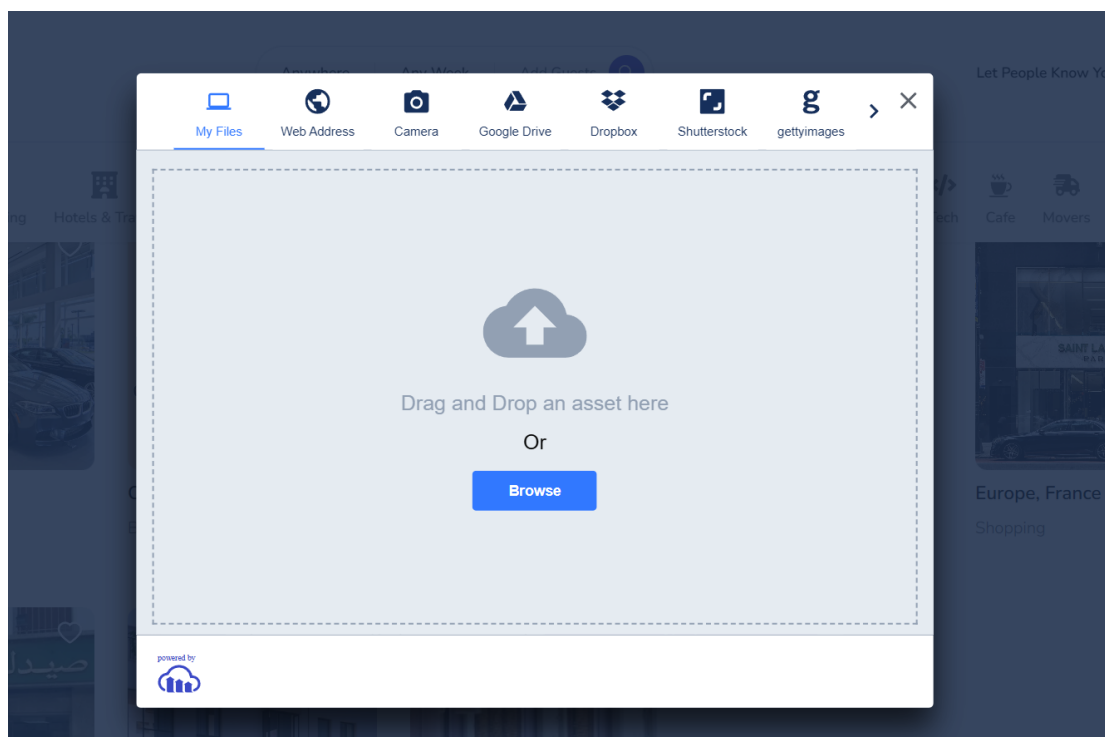
npm install cloudinary

Step 3: add it to .env file

```
NEXT_PUBLIC_CLOUDINARY_CLOUD_NAME ="digurnehj"
```

Step 4: configure cloudinary

```
ts getListingById.ts U TS route.ts U ImageUpload.tsx U x
app > components > inputs > ImageUpload.tsx > ImageUpload
2
3 import { CldUploadWidget } from "next-cloudinary";
4 import Image from "next/image";
5 import { useCallback } from "react";
6 import { TbPhotoPlus } from 'react-icons/tb'
7
8 declare global {
9   var cloudinary: any
10 }
11
12 const uploadPreset = "zuf6trzx";
13
14 interface ImageUploadProps {
15   onChange: (value: string) => void;
16   value: string;
17 }
18
19 const ImageUpload: React.FC<ImageUploadProps> = ({
20   onChange,
21   value
22 }) => {
23   const handleUpload = useCallback((result: any) => {
24     onChange(result.info.secure_url);
25   }, [onChange]);
26
27   return (
28     <CldUploadWidget
29       onUpload={handleUpload}
```



CHAPTER 6 :

UI\UX

The user interface (UI) and user experience (UX) design of Pearl are central to ensuring an intuitive, engaging, and efficient interaction for users. By employing modern design principles, leveraging Tailwind CSS for styling, and integrating best practices in usability, Pearl aims to deliver a seamless and enjoyable experience for its users.

Design Principles

1. **User-Centered Design:** Pearl prioritizes the needs and preferences of its users, ensuring that every design decision is aimed at enhancing the user experience. This involves understanding user behaviors, motivations, and pain points through user research and feedback.
2. **Consistency:** A consistent design language is maintained across the application to provide a cohesive look and feel. This includes consistent use of colors, typography, and component styles.
3. **Accessibility:** Pearl is designed to be accessible to all users, including those with disabilities. This involves following accessibility guidelines, ensuring proper contrast ratios, keyboard navigability, and screen reader compatibility.
4. **Simplicity and Clarity:** The design emphasizes simplicity and clarity, avoiding unnecessary complexity and ensuring that information is presented in a clear and understandable manner. This helps users complete their tasks efficiently.
5. **Responsive Design:** Pearl is designed to be fully responsive, providing an optimal viewing experience across a wide range of devices, from mobile phones to desktop computers. This ensures usability regardless of screen size or device.

UI Components and Design System

Reusable Components

Pearl employs a component-based architecture, where UI elements are encapsulated into reusable components. This approach enhances maintainability, consistency, and scalability.

- **Buttons:** Standardized button components with different variants (primary, secondary, disabled) to ensure consistency and reusability across the application.
- **Forms:** Modular form components (input fields, text areas, dropdowns, checkboxes) that can be easily reused and customized for different forms throughout the app.
- **Cards:** Card components for displaying business listings, property details, and reviews in a visually appealing and consistent manner.
- **Modals:** Reusable modal components for displaying overlays and dialog boxes, ensuring a consistent experience for pop-up interactions.

Tailwind CSS for Styling

Tailwind CSS is used for styling the UI components, offering a utility-first approach that simplifies the process of building custom designs.

1. **Utility Classes:** Tailwind's utility classes allow for rapid styling directly in the HTML, reducing the need for custom CSS and promoting consistency.
2. **Customization:** Tailwind's configuration file enables extensive customization of the design system, including colors, spacing, typography, and breakpoints.
3. **Responsive Design:** Tailwind provides responsive utility classes that make it easy to apply styles for different screen sizes, ensuring a seamless responsive design.

User Flows and Navigation

Onboarding

The onboarding process is designed to be quick and straightforward, helping new users get started with Pearl effortlessly.

1. **Welcome Screen:** A friendly welcome screen introducing users to Pearl and its key features.
2. **Sign-Up/Sign-In:** Simple and secure authentication forms, with options for social logins (e.g., Google) and email/password.
3. **Guided Tour:** An optional guided tour that highlights the main features and functionalities of the app, helping users understand how to navigate and use Pearl effectively.

Search and Discovery

The search and discovery features are central to Pearl, enabling users to find local businesses and properties with ease.

1. **Search Bar:** A prominent search bar on the homepage, allowing users to quickly search for businesses or properties by name, category, or location.
2. **Filter and Sort Options:** Advanced filtering and sorting options to refine search results based on criteria such as ratings, distance, price, and more.
3. **Interactive Map:** An interactive map view that allows users to explore businesses and properties in their geographical context, enhancing the discovery experience.

User Profiles and Dashboards

User profiles and dashboards provide a personalized experience, allowing users to manage their information, reviews, and reservations.

1. **Profile Management:** Users can view and edit their profile information, including contact details, profile picture, and preferences.
2. **My Trips:** A dedicated section for users to view and manage their upcoming and past trips, including reservation details and status.
3. **Favorites:** A section where users can save and manage their favorite businesses and properties for quick access.
4. **Owned Local Businesses/Properties:** For users who own businesses or properties listed on Pearl, this section allows them to manage their listings, including updating information, adding images, and viewing reviews.

User Feedback and Continuous Improvement

Pearl incorporates mechanisms for gathering user feedback to continually improve the UI/UX.

1. **Feedback Forms:** Easy-to-access feedback forms throughout the application, allowing users to provide suggestions, report issues, and share their experiences.
2. **User Testing:** Regular user testing sessions to gather insights into how users interact with the app and identify areas for improvement.
3. **Analytics:** Integration of analytics tools to monitor user behavior, track key metrics, and identify trends that can inform design decisions.

Conclusion

The UI/UX design of Pearl is crafted to provide an intuitive, engaging, and efficient user experience. By focusing on user-centered design principles, employing reusable components with Tailwind CSS, and continually gathering user feedback, Pearl aims to deliver a seamless and enjoyable platform for discovering and interacting with local businesses and properties.

CHAPTER 7 :

AI

Machine learning and AI

Chapter 1

In this notebook, we analyze a dataset from Yelp containing information about various businesses. We aim to preprocess this data, particularly focusing on categories and cleaning unnecessary columns.

1. Importing Libraries

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

We start by importing the necessary libraries for data manipulation and visualization.

- `pandas`: Used for data manipulation and analysis.
- `numpy`: Provides support for large, multi-dimensional arrays and matrices.
- `matplotlib`: A plotting library for creating static, animated, and interactive visualizations.
- `seaborn`: Built on top of `matplotlib`, provides a high-level interface for drawing attractive and informative statistical graphics.

2. Loading the Data

```
In [2]: business = pd.read_json(r"yelp_academic_dataset_business.json" , lines = True)
```

We load the dataset using `pandas`' `read_json` method.

This reads the JSON file line by line into a DataFrame.

****3. Initial Data Exploration****

We explore the dataset's structure using the `info` method.

```
In [3]: business.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42153 entries, 0 to 42152
Data columns (total 15 columns):
#   Column          Non-Null Count  Dtype
---  -
0   business_id     42153 non-null  object
1   full_address    42153 non-null  object
2   hours           42153 non-null  object
3   open            42153 non-null  bool
4   categories      42153 non-null  object
5   city            42153 non-null  object
6   review_count    42153 non-null  int64
7   name            42153 non-null  object
8   neighborhoods   42153 non-null  object
9   longitude       42153 non-null  float64
10  state           42153 non-null  object
11  stars           42153 non-null  float64
12  latitude        42153 non-null  float64
13  attributes      42153 non-null  object
14  type            42153 non-null  object
dtypes: bool(1), float64(3), int64(1), object(10)
memory usage: 4.5+ MB
```

This provides a concise summary of the DataFrame, including the data types of each column and the number of non-null values.

****4. Extracting Categories****

We aim to extract and process the categories from the dataset.

Extracting the categories

- we will divide our data according to the categories

```
In [5]: categories = []
error_index_zero = dict()
error_index_not_list = dict()

for i in range(len(business)):
    category = business.loc[i, "categories"]
    if type(category) == list:
        if len(category) >= 1:
            categories.append(category[0])
        else:
            error_index_zero[i] = business.loc[i, ["business_id", "name"]]
    else:
        error_index_not_list[i] = business.loc[i, ["business_id", "name"]]
```

****Explanation: ****

- We initialize empty lists and dictionaries to store categories and errors.
- We iterate through the dataset, checking the type of the `category` field.
- If it's a list and has elements, we append the first category to the `categories` list.
- If it's empty or not a list, we log the business ID and name into the respective dictionaries.
- we found that the Column category contains ZERO values , So we need to remove them

****5. Handling Zero Categories****

We identify and handle businesses with no categories.

now we have zero list categories , lets see the count

```
In [7]: type(error_index_zero)
```

```
Out[7]: dict
```

```
In [8]: len(error_index_zero.keys())
```

```
Out[8]: 339
```

find the percentage of the index that have no category

```
In [9]: (len(error_index_zero.keys()) / len(business))*100
```

```
Out[9]: 0.804213223258131
```

the percentage is 0.8% of the total data

we can remove them easily since they are few compared to the data

- Calculate the percentage of businesses with no categories.
- Since it's a small percentage, we decide to remove them.

****6. Removing Unnecessary Rows****

We drop rows with no categories.

```
In [10]: # we create a list of index we want to remove and drop them
index_rm = list(error_index_zero.keys())
print(type(index_rm))
index_rm
```

```
394,
449,
530,
548,
569,
597,
637,
670,
713,
813,
827,
846,
1104,
1122,
1312,
1605,
1678,
1803,
2022,
2460,
```

```
In [11]: %store index_rm
Stored 'index_rm' (list)
```

```
In [12]: # we drop the index
business = business.drop(index_rm)
```

```
In [13]: business.shape
```

```
Out[13]: (41814, 15)
```

- Convert the keys of `error_index_zero` to a list.
- Drop these indices from the DataFrame.
- Check the new shape of the DataFrame to ensure rows were removed.

****7. Resetting Index****

We reset the DataFrame index.

now we reset the index to start working on the data

```
In [15]: # resetting the index  
business = business.reset_index(drop=True)
```

```
In [16]: business.iloc[119:124,:]
```

```
Out[16]:
```

	business_id	full_address	hours	open	categories	city	review_count	name	neighborhood
119	qQvpS262VPEHKZvM6j5uhQ	1902 Bartillon Dr Westchester Gardens Madison, WI 53704	{}	True	[Bars, Sports Bars, Nightlife]	New York	14	Sports Pub	Westchester Gardens
120	XWDk-pLSbVE2Vw9Z-zusQ	2655 E Washington Ave Starkweather - Yahara Madison, WI 53704	{}	True	[Fast Food, Restaurants]	New York	10	Burger King	Starkweather - Yahara
121	T9I9Z3FKMSCIYsP2g4efw	2632 Milwaukee St Starkweather - Yahara Madison, WI 53704	{'Monday': {'close': '20:00', 'open': '10:00'}, ...}	True	[Hair Salons, Beauty & Spas]	New York	7	Rebecca Lynn Studio	Starkweather - Yahara
122	fBSd-fI9-9LkitwrFRcBng	2021 Zeier Rd Madison, WI 53704	{}	True	[Arts & Crafts, Shopping, Fabric Stores]	New York	8	Jo-Ann Fabrics & Crafts	
123	1hBOR97oX5Xph3J8Tdob7w	1601 Aspen Commons Middleton, WI 53562	{}	True	[Burgers, American (Traditional), Restaurants]	New York	46	Cheeseburger in Paradise	

- Resetting the index to ensure it starts from 0 and increments by 1.

****8. Creating Categories DataFrame****

We create a DataFrame to store categories and business IDs.

```
In [20]: categories_df = categories_df.T
```

```
In [21]: categories_df.columns = ["category", "business_id"]
```

now we have 2 dataframes that have the same index

```
In [22]: categories_df.index == business.index
```

```
Out[22]: array([ True,  True,  True, ...,  True,  True,  True])
```

- now can see the unique categories and extract it from the data

```
In [23]: categories_df
```

```
Out[23]:
```

	category	business_id
0	Doctors	vcNAVMLM4dR7D2nwwJ7nCA
1	Restaurants	JwUE5GmEO-sH1FuWJgKBIQ
2	American (Traditional)	uGykseHzyS5xAMVoN6YUqA
3	Food	LRKJF43s9-3jG9Lgx4zODg
4	Chinese	RgDg-k9S5YD_BaxMckifkg
...
41809	Food	uUstfpN81JCMKyH6c0D0bTg
41810	Active Life	tsvWVY4o64xiv7K0VA89R8A
41811	Yelp Events	nYer89hXYAoddMEKTxw7kA
41812	Kosher	BMjgglgOghBMEXPo8q7q3w
41813	Food	BVxlrYVWgmi-8TPGMe6CTpg

41814 rows × 2 columns

- Initialize a dictionary to store categories and business IDs.
- Iterate through the dataset, extracting the first category and business ID.
- Convert the dictionary to a DataFrame and transpose it.
- Rename the columns to `category` and `business_id`.

We identify unique categories.

```
In [25]: # we have now all unique categories
```

```
Out[25]: {'Accessories',
          'Accountants',
          'Active Life',
          'Acupuncture',
          'Adult Education',
          'Adult Entertainment',
          'Afghan',
          'African',
          'Airports',
          'Allergists',
          'American (New)',
          'American (Traditional)',
          'Animal Shelters',
          'Antiques',
          'Appliances',
          'Argentine',
          'Art Galleries',
          'Art Schools',
          'Art Supplies',
```

- Convert the 'categories' list to a set to remove duplicates and get unique categories.

We filter businesses related to food.

```
In [27]: filt = categories_df.loc[:, "category"] == "Food"
          business[filt]
```

	business_id	full_address	hours	open	categories	city	review_count	name	neighborhoods	longitude
3	LRLKJF43s9-3jG9Lgx4zODg	4910 County Rd V\nDe Forest, WI 53532	{\nMonday:\n{\n'close':\n'22:00',\n'open':\n'10:30'}\n}	True	[Food, Ice Cream & Frozen Yogurt, Fast Food, R...	New York	7	Culver's	[]	-89.37498
27	1-EldEHWtEVcpoZm38N8A	1901 Cayuga St\nMiddleton, WI 53562	{\nMonday:\n{\n'close':\n'21:00',\n'open':\n'06:00'}\n}	True	[Food, Beer, Wine & Spirits, Coffee & Tea]	New York	22	Barriques Wine & Spirits	[]	-89.51373
36	3jpKGdkFrCHqjEYVEgcG0A	8310 Greenway Blvd\nMiddleton, WI 53562	{}	True	[Food, Coffee & Tea]	New York	6	Starbucks	[]	-89.52568
39	3XVUkPFvWwKjKFE	1650 Deming Way\nMiddleton,		True	[Food, Ice Cream &	New		Cold Stone		-89.50000

- Create a filter to identify businesses in the "Food" category.

- Apply the filter to the DataFrame and calculate the percentage of food-related businesses.

****11. Dropping Unnecessary Columns****

- We found that there are many columns that are not necessary in the data
- Like (city : always New York , state : always NYC , type : business , open)

We drop columns that are not needed for further analysis.

- Check the unique values of columns to determine redundancy.

```
In [29]: business.columns
```

```
Out[29]: Index(['business_id', 'full_address', 'hours', 'open', 'categories', 'city',
              'review_count', 'name', 'neighborhoods', 'longitude', 'state', 'stars',
              'latitude', 'attributes', 'type'],
              dtype='object')
```

```
In [30]: business.loc[:, "type"].unique()
```

```
Out[30]: array(['business'], dtype=object)
```

```
In [31]: business.loc[:, "open"].value_counts()
```

```
Out[31]: open
         True    36824
         False   4990
         Name: count, dtype: int64
```

```
In [32]: filt1 = business.loc[:, "open"] == False
         business[filt1]
```

```
Out[32]:
```

	business_id	full_address	hours	open	categories	city	review_count	name	neighborhoods	longitude
5	oLctHIA1AxmsgOuu4dM6Vw	4156 County Rd B\nMcFarland, VM 53558	{}	False	[Television Stations, Mass Media]	New York	10	Charter Communications	[]	-89.32292
15	HxPpZSY6G1eARuiahhra6A	6401 University Ave\nMiddleton, VM 53562	{}	False	[Event Planning & Services, Party & Event Plan...]	New York	5	Crandalls Carryout & Catering	[]	-89.49180
17	77ESrCo7hQ96VpCWWdvoXg	6230 University Ave\nMiddleton, VM 53562	{'Monday': {'close': '21:00', 'open': '06:00'}, ...}	False	[Mexican, Restaurants]	New York	17	Mi Cocina	[]	-89.48748
55	lEmqrFe96NOhU07TA0rZdw	6625 Century Ave\nMiddleton, VM 53562	{}	False	[American (Traditional), Restaurants]	New York	7	Stamm House At Pheasant Branch	[]	-89.49442
			{'Tuesday': ...}		[Skin Care]					

- Drop the `city`, `state`, `type`, and `open` columns as they are not useful for our analysis.

1. city and state --> they are all NYC
2. type --> they are all business
3. open --> the closed one is due to the day the data is collected

```
business.drop(columns = {"city" , "state" , "type" , "open"})
```

	business_id	full_address	hours	categories	review_count	name	neighborhoods	longitude	stars	latitude
0	vcNAVWLM4dR7D2nwwJ7nC	4840 E Indian School Rd Ste 101 Phoenix, AZ ...	{ 'Tuesday': 'close': '17:00', 'open': '08:00'...	[Doctors, Health & Medical]	7	Eric Goldberg, MD	[]	-111.983758	3.5	33.499313
1	JwUE5GmEO-sH1FuwwJgKBIG	6162 US Highway 51 De Forest, WI 53532	{}	[Restaurants]	26	Pine Cone Restaurant	[]	-89.335844	4.0	43.238893
2	uGykeHzYSSxAMVoN6YUqA	505 W North St De Forest, WI 53532	{ 'Monday': 'close': '22:00', 'open': '06:00'...	[American (Traditional), Restaurants]	16	Deforest Family Restaurant	[]	-89.353437	4.0	43.252267
3	LRKJF43s9-3jG9Lgx4zODg	4910 County Rd V De Forest, WI 53532	{ 'Monday': 'close': '22:00', 'open': '10:30'...	[Food, Ice Cream & Frozen Yogurt, Fast Food, R...	7	Culver's	[]	-89.374983	4.5	43.251045
4	RgDg-k9S5DY_BaxMckifkg	631 S Main St De Forest, WI 53532	{ 'Monday': 'close': '22:00', 'open': '11:00'...	[Chinese, Restaurants]	3	Chang Jiang Chinese Kitchen	[]	-89.343722	4.0	43.240875

- Summary

****Title: Yelp Business Data Analysis****

****Introduction:****

- A brief introduction to the dataset and the goals of the analysis.

Data Loading and Initial Exploration:

- Description of the dataset loading process.

- Summary of the dataset structure using `info`.

****Category Extraction:****

- Explanation of the process to extract and handle categories.

- Handling errors and removing businesses with no categories.

****Data Cleaning:****

- Steps to remove unnecessary rows and reset the DataFrame index.

****Category DataFrame Creation:****

- Process of creating a new DataFrame to store categories and business IDs.

****Unique Categories and Filtering:****

- Identification of unique categories.
- Filtering businesses related to the "Food" category and calculating their percentage.

****Dropping Columns:****

- Criteria for dropping columns and the actual process.

****Conclusion:****

- Summary of the data cleaning and preprocessing steps.
- Future steps or analysis that could be performed on the cleaned data.

Chapter 2

Shallow and Deep Copy in Python

Shallow Copy

A shallow copy creates a new object but does not create copies of the objects that the original object references. Instead, it only copies the references to those objects. This means that the new object is a copy of the original object, but the elements within the new object are references to the same elements within the original object.

shallow copy

A shallow copy creates a new object, but does not create copies of the objects that the original object references. Instead, it only copies the references to those objects. This means that the new object is a copy of the original object, but the elements within the new object are references to the same elements within the original object.

```
In [1]: import copy
```

```
In [2]: org = [1,[2,3],4]
```

```
In [3]: shallow = copy.copy(org)
```

```
In [4]: shallow
```

```
Out[4]: [1, [2, 3], 4]
```

```
In [5]: shallow[1] = 2
```

```
In [6]: print(org)
        print(shallow)
```

```
[1, [2, 3], 4]
[1, 2, 4]
```

Deep Copy

A deep copy creates a new object and recursively copies all objects found in the original, creating fully independent duplicates. This means that all objects referenced by the original are copied, and their references are also copied, resulting in a completely independent object structure.

Deep copy

A deep copy creates a new object and recursively copies all objects found in the original, creating fully independent duplicates. This means that all objects referenced by the original are copied, and their references are also copied, resulting in a completely independent object structure.

```
In [7]: original = [1,[2,3,4,5],6,7,8]
        deep = copy.deepcopy(original)
```

```
In [8]: deep[1] = 5
        deep
```

```
Out[8]: [1, 5, 6, 7, 8]
```

```
In [9]: print(original)
        print(deep)
```

```
[1, [2, 3, 4, 5], 6, 7, 8]
[1, 5, 6, 7, 8]
```

Copy of the Data

Copy of the data

```
In [10]: import pandas as pd
import numpy as np

In [11]: business = pd.read_json(r"yelp_academic_dataset_business.json", lines = True)

In [12]: data = copy.deepcopy(business)

In [13]: type(data)
Out[13]: pandas.core.frame.DataFrame

In [14]: data.head()
Out[14]:
```

	business_id	full_address	hours	open	categories	city	review_count	name	neighborhoods	longitude	state	stars	lati
0	vcNAWILM4dR7D2nwwJ7nCA	4840 E Indian School Rd Phoenix, AZ ...	{ 'Tuesday': {'close': '17:00', 'open': '08:00'}, }	True	[Doctors, Health & Medical]	Phoenix	7	Eric Goldberg, MD	[]	-111.963758	AZ	3.5	33.49
1	JwUE5GmEO-sh1FuWJgKBIQ	6162 US Highway 51 De Forest, WI 53532	{ }	True	[Restaurants]	De Forest	26	Pine Cone Restaurant	[]	-89.335844	WI	4.0	43.23
2	uGykseHzysSxAM/VoN6YUqA	505 W North St De Forest, WI 53532	{ 'Monday': {'close': '22:00', 'open': '06:00'}, }	True	[American (Traditional), Restaurants]	De Forest	16	Deforest Family Restaurant	[]	-89.353437	WI	4.0	43.25
3	LRKJF43s9-3jO9Lgx4zODg	4910 County Rd De Forest, WI 53532	{ 'Monday': {'close': '22:00', 'open': '10:30'}, }	True	[Food, Ice Cream & Frozen Yogurt, Fast Food, R...	De Forest	7	Culver's	[]	-89.374983	WI	4.5	43.25
4	RgDg-k9S5YD_BaxMckifkg	631 S Main St De Forest, WI 53532	{ 'Monday': {'close': '22:00', 'open': '11:00'}, }	True	[Chinese, Restaurants]	De Forest	3	Chang Jiang Chinese Kitchen	[]	-89.343722	WI	4.0	43.24

Modified String Data

```
In [15]: # needs
## modified string data
print("modified string data is named (data)")

modified string data is named (data)
```

```
In [16]: # copy of the data

bus1 = copy.deepcopy(business)
print("copy of original data is named (bus1)")

copy of original data is named (bus1)
```

- the aim of this notebook is to take a copy of the data and modifies it
- we will convert the column ("Categories") into string and manipulate it to separate the data apart

- The aim of this notebook is to take a copy of the data and modify it.
- We will convert the "Categories" column into a string and manipulate it to separate the data apart.
- First, we need to remove the businesses with zero categories from the data.

Removing Zero Categories Businesses from the Data

```
In [17]: # removing the zero categories busienss from the data

error_index_zero = dict()

for i in range(len(data)):
    category = data.loc[i,"categories"]
    if (len(category) >= 1):
        pass
    else:
        error_index_zero[i] = data.loc[i,["business_id" , "name"]]
```

Removing the Zero Category Data and Resetting the Index

now remove the ZERO category data and reset the index

```
In [20]: %store -r index_rm

In [21]: data = data.drop(index_rm)

In [22]: data.shape
Out[22]: (41814, 15)

In [23]: # resetting index after deleting rows
data = data.reset_index(drop=True)

In [24]: bus1.shape
Out[24]: (42153, 15)

In [25]: bus1 = bus1.drop(error_index_zero.index)

In [26]: bus1 = bus1.reset_index(drop=True)

In [27]: bus1.shape
Out[27]: (41814, 15)
```

Saving Data Across Notebooks

```
In [28]: var = [1,2,3,4,5,6]
%store var

Stored 'var' (list)
```

Dividing the Data According to Categories Column

Convert the Column into String

now we will divide the data according to categories column

step 1 :convert the column into string

```
In [29]: data.columns
Out[29]: Index(['business_id', 'full_address', 'hours', 'open', 'categories', 'city',
              'review_count', 'name', 'neighborhoods', 'longitude', 'state', 'stars',
              'latitude', 'attributes', 'type'],
              dtype='object')

In [30]: data["categories"] = data["categories"].astype(str)

In [31]: sorted(data["categories"].unique())

Out[31]: ["Children's Clothing", "Men's Clothing", "Women's Clothing", '\\Fashion\\', '\\Shopping\\',
          '\\Children's Clothing', '\\Men's Clothing', '\\Fashion\\', '\\Shopping\\',
          '\\Children's Clothing', '\\Women's Clothing', '\\Fashion\\', '\\Shopping\\', "Men's Clothing"],
          ["Children's Clothing", "Women's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Accessories\\'],
          ["Children's Clothing", "Women's Clothing", '\\Fashion\\', '\\Shopping\\'],
          ["Children's Clothing", '\\Baby Gear & Furniture\\', '\\Fashion\\', '\\Shopping\\', '\\Toy Stores\\'],
          ["Children's Clothing", '\\Department Stores\\', '\\Fashion\\', '\\Shopping\\', "Men's Clothing", "Women's Clothing"],
          ["Children's Clothing", '\\Department Stores\\', '\\Fashion\\', '\\Shopping\\'],
          ["Children's Clothing", '\\Department Stores\\', '\\Thrift Stores\\', '\\Shopping\\', '\\Fashion\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Accessories\\', '\\Shoe Stores\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Accessories\\', '\\Toy Stores\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Baby Gear & Furniture\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Gyms\\', '\\Fitness & Instruction\\', '\\Active Life\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Health & Medical\\', '\\Baby Gear & Furniture\\', '\\Lactation Services\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Home Services\\', '\\Interior Design\\', "Women's Clothing"],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Jewelry\\', '\\Baby Gear & Furniture\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Shoe Stores\\', '\\Sporting Goods\\', "Men's Clothing", "Women's Clothing",
          '\\Sports Wear\\'],
          ["Children's Clothing", '\\Fashion\\', '\\Shopping\\', '\\Sporting Goods\\', "Women's Clothing", '\\Sports Wear\\'],
          ...]
```

Lets see , the Plan is :-

1. Identify the unique keywords of every section (by developer's eye).
2. Iterate through original data (business) --> deep_copy and identify the data we want according to the keywords.
3. Get the index of these data.
4. Separate the data into new data.
5. Remove from both original data and copy data.
6. Iterate till we finish the data.

Requirements

1. Copy of the data --> original.
2. Copy of the modified string data.

Function to Extract Data Index According to Keywords

```
In [34]: # function that return index of specific data with keywords

def get_category(dataset , column_name , keywords):
    index_list = list()
    for i in range(len(dataset)):
        check_list = dataset.loc[i, str(column_name)]
        for j in range(len(check_list)):
            if check_list[j] in keywords:
                index_list.append(i)
                break
    return index_list

In [35]: key_word_list = [ "Children's Clothing" , "Men's Clothing" , "Women's Clothing"]
men_women_clothes = get_category(bus1 , "categories" , key_word_list)

In [36]: len(men_women_clothes)

Out[36]: 795

In [37]: (len(men_women_clothes)/len(data))*100

Out[37]: 1.9012770842301623
```

Function to Separate Data According to Index and Reset the Index

```
function that delete data according to index and reset the index

In [38]: '''
- function that separate the data according to index
- update the index of the original and copy data
- allocate the index of new data in a dictionary
- calculate the percentage of the data from total data
'''

Out[38]: '\n - function that separate the data according to index \n - update the index of the original and copy data\n - allocate the index of new data in a dictionary\n - calculate the percentage of the data from total data\n\n'

In [39]: def separate_category(dataset, datacopy , index_list , new_data_name):
    data_log = dict()
    dataset_org = dataset
    dataset_copy = datacopy
    # dropping the index from both dataset and copy data
    dataset_org = dataset_org.drop(index_list , inplace = True)
    dataset_copy = dataset_copy.drop(index_list , inplace = True)

    # reset index in both datasets
    dataset = dataset.reset_index(drop=True , inplace = True)
    datacopy = datacopy.reset_index(drop=True , inplace = True)

    # saving the data to a dictionary to retrieve it
    data_log[new_data_name] = index_list

    return f"the data is created and added" , data_log

In [40]: # the index list check
men_women_clothes

Out[40]: [26,
64,
75,
112,
129,
728,
759,
828,
874,
931,
978,
979,
```

Summary

- We have two deep copies of the data, one for the original data copy and the other for extracting the unique values in the category data.

Functions

1. `'get_category'`: Retrieves categories according to the index.
2. `'seperate_category'`: Adds the index to the dictionary of datasets, deletes the detected columns, and updates the two other datasets.

Extracting Keyword Data from the Original Data

```
In [51]: # extracting keyword data from the original data
men_women_data = business.loc[men_women_clothes , :]
```

```
In [53]: # saving data as csv
men_women_data.to_csv("MenWomenCloth.csv")
```

```
In [54]: # extracting the specified data from the review data
review = pd.read_json(r"yelp_academic_dataset_review.json" , lines = True)
```

```
In [56]: review.head(3)
```

```
Out[56]:
```

	votes	user_id	review_id	stars	date	text	type	business_id
0	{'funny': 0, 'useful': 2, 'cool': 1}	Xq0DzH4yRqVH3VRG7hgz	1SSdjuK7DmYqUAj6rjGowg	5	2007-05-17	dr. goldberg offers everything i look for in a...	review	vcNAWILM4dR7D2nwwJ7nCA
1	{'funny': 0, 'useful': 2, 'cool': 0}	H1kH5GZV7Le4zqTRNxoZow	RF6UnRTIG7WVMcrO2GEoAg	2	2010-03-22	Unfortunately, the frustration of being Dr. Go...	review	vcNAWILM4dR7D2nwwJ7nCA
2	{'funny': 0, 'useful': 1, 'cool': 1}	zvJCorpm2yOZrxKffwGGLA	-TsVN230RCKLYKBeLsuz7A	4	2012-02-14	Dr. Goldberg has been my doctor for years and ...	review	vcNAWILM4dR7D2nwwJ7nCA

```
In [57]: men_women_data.head(3)
```

```
Out[57]:
```

	business_id	full_address	hours	open	categories	city	review_count	name	neighborhoods	longitude	state	stars
26	x7pStpT09Tdc8d7oRU0Tmw	1620 Deming Way\nMiddleton, WI 53562	{\"Monday\": {\"close\": \"20:00\", \"open\": \"10:00\"}...	True	[Women's Clothing, Shopping Centers, Fashion, ...	Middleton	7	Greenway Station Shopping Center		-89.528450	WI	3.5
64	xm1RU63pxYsm-7dzGaZA	32 W Towne Mall\nMadison, WI 53719	{}	True	[Women's Clothing, Men's Clothing, Fashion, Sh...	Madison	6	Banana Republic		-89.509317	WI	3.0
75	pwHZ5UUB3cWls9O-L_hdA	Greenway Station\n1621 Demingway\nMiddleton, W...	{\"Monday\": {\"close\": \"20:00\", \"open\": \"10:00\"}...	True	[Women's Clothing, Fashion, Shopping, Accessor...	Middleton	4	LOFT		-89.525338	WI	2.0

Extracting the Specified Data from the Review Data

```
In [58]: list_of_bus_id = men_women_data.business_id
```

```
In [60]: list_of_bus_id = list(list_of_bus_id)
```

```
Out[60]:
```

```
[\"x7pStpT09Tdc8d7oRU0Tmw\", \"xm1RU63pxYsm-7dzGaZA\", \"pwHZ5UUB3cWls9O-L_hdA\", \"RF6UnRTIG7WVMcrO2GEoAg\", \"zvJCorpm2yOZrxKffwGGLA\", \"1SSdjuK7DmYqUAj6rjGowg\", \"H1kH5GZV7Le4zqTRNxoZow\", \"-TsVN230RCKLYKBeLsuz7A\", \"C0ep2B8uJ2ETf6uS85ye\", \"u-5ia-8d3m3CJ_dhM\", \"C4qRNM1m6D0u8mW_-j8tNg\", \"r1apR0C0u8d4H2p0dG\", \"0r4t02p-a3z8u_-8u0t8e\", \"ed3cywffQ2_-uNPO2b\", \"39a2386_v311TtTtFg-qC\", \"C15ag8Z9M1Acg3nLEdRag\", \"L4u75uFLu0g8YK8Dm-A\", \"d088ygl172yRauuH3-Q\", \"d08L0950p238p0WwCCUg\", \"3W6T7G3H6FAw82u4kIavQ\", \"...\"],
```

```
In [61]: review = review.set_index(\"business_id\")
```

```
In [64]: review.head(3)
```

```
Out[64]:
```

	votes	user_id	review_id	stars	date	text	type
vcNAWILM4dR7D2nwwJ7nCA	{'funny': 0, 'useful': 2, 'cool': 1}	Xq0DzH4yRqVH3VRG7hgz	1SSdjuK7DmYqUAj6rjGowg	5	2007-05-17	dr. goldberg offers everything i look for in a...	review
vcNAWILM4dR7D2nwwJ7nCA	{'funny': 0, 'useful': 2, 'cool': 0}	H1kH5GZV7Le4zqTRNxoZow	RF6UnRTIG7WVMcrO2GEoAg	2	2010-03-22	Unfortunately, the frustration of being Dr. Go...	review
vcNAWILM4dR7D2nwwJ7nCA	{'funny': 0, 'useful': 1, 'cool': 1}	zvJCorpm2yOZrxKffwGGLA	-TsVN230RCKLYKBeLsuz7A	4	2012-02-14	Dr. Goldberg has been my doctor for years and ...	review

```
In [ ]: review.loc[list_of_bus_id,:]
```

```
In [66]: len(list_of_bus_id)
```


Step 4: Data Pre-processing

We are going to use a technique called collaborative filtering to generate recommendations for users. This technique is based on the premise that similar people like similar things. The first step is to transform our data into a user-item matrix, also known as a "utility" matrix. In this matrix, rows represent users and columns represent businesses. The beauty of collaborative filtering is that it doesn't require any information about the users or the businesses to generate recommendations.

The `create_X()` function outputs a sparse matrix (X) with four mapper dictionaries:

- **user_mapper**: maps user id to user index
- **business_mapper**: maps business id to business index
- **user_inv_mapper**: maps user index to user id
- **business_inv_mapper**: maps business index to business id

We need these dictionaries because they map which row/column of the utility matrix corresponds to which user/business id.

Our (X) (user-item) matrix is a [scipy.sparse.csr matrix](#) which stores the data sparsely.

```
from scipy.sparse import csr_matrix

def create_X(df):
    """
    Generates a sparse matrix from ratings dataframe.

    Args:
        df: pandas dataframe containing 3 columns (userId, businessId, rating)

    Returns:
        X: sparse matrix
        user_mapper: dict that maps user id's to user indices
        user_inv_mapper: dict that maps user indices to user id's
        business_mapper: dict that maps business id's to business indices
        business_inv_mapper: dict that maps business indices to business id's
    """
    M = df['userId'].nunique()
    N = df['businessId'].nunique()

    user_mapper = dict(zip(np.unique(df["userId"]), list(range(M))))
    business_mapper = dict(zip(np.unique(df["businessId"]), list(range(N))))

    user_inv_mapper = dict(zip(list(range(M)), np.unique(df["userId"])))
    business_inv_mapper = dict(zip(list(range(N)), np.unique(df["businessId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    item_index = [business_mapper[i] for i in df['businessId']]

    X = csr_matrix((df["rating"], (user_index, item_index)), shape=(M, N))

    return X, user_mapper, business_mapper, user_inv_mapper, business_inv_mapper

X, user_mapper, business_mapper, user_inv_mapper, business_inv_mapper = create_X(ratings)
X.shape
```

Our X matrix contains 610 users and 9724 businesses.

Evaluating sparsity

Here, we calculate sparsity by dividing the number of stored elements by the total number of elements. The number of stored (non-empty) elements in our matrix ([nnz](#)) is equivalent to the number of ratings in our dataset.

```
n_total = X.shape[0] * X.shape[1]
n_ratings = X.nnz
sparsity = n_ratings / n_total
print(f"Matrix sparsity: {round(sparsity*100,2)}%")
```

`csr_matrix.nnz` counts the stored values in our sparse matrix. The rest of our cells are empty.

The **cold start problem** is when there are new users and businesses in our matrix that do not have any ratings. In our business dataset, all users and businesses have at least one rating but in general, it's useful to check which users and businesses have few interactions.

To solve the cold start problem

```
n_ratings_per_user = np.diff(X.indptr)
len(n_ratings_per_user)

print(f"Most active user rated {n_ratings_per_user.max()} businesses.")
print(f"Least active user rated {n_ratings_per_user.min()} businesses.")

n_ratings_per_business = np.diff(X.tocsc().indptr)
len(n_ratings_per_business)

print(f"Most rated business has {n_ratings_per_business.max()} ratings.")
print(f"Least rated business has {n_ratings_per_business.min()} ratings.")

plt.figure(figsize=(16,4))
plt.subplot(1,2,1)
sns.kdeplot(n_ratings_per_user, shade=True)
plt.xlim(0)
plt.title("Number of Ratings Per User", fontsize=14)
plt.xlabel("number of ratings per user")
plt.ylabel("density")
plt.subplot(1,2,2)
sns.kdeplot(n_ratings_per_business, shade=True)
plt.xlim(0)
plt.title("Number of Ratings Per Business", fontsize=14)
plt.xlabel("number of ratings per business")
plt.ylabel("density")
plt.show()
```

Step 5: Item-item Recommendations with k-Nearest Neighbors

We are going to find the (k) businesses that have the most similar user engagement vectors for business (i).

```
from sklearn.neighbors import NearestNeighbors

def find_similar_businesses(business_id, X, business_mapper, business_inv_mapper, k, metric='cosine'):
    """
    Finds k-nearest neighbours for a given business id.

    Args:
        business_id: id of the business of interest
        X: user-item utility matrix
        k: number of similar businesses to retrieve
        metric: distance metric for kNN calculations

    Output: returns list of k similar business ID's
    """
    X = X.T
    neighbour_ids = []

    business_ind = business_mapper[business_id]
    business_vec = X[business_ind]
    if isinstance(business_vec, (np.ndarray)):
        business_vec = business_vec.reshape(1, -1)
    # use k+1 since kNN output includes the businessId of interest
    kNN = NearestNeighbors(n_neighbors=k+1, algorithm="brute", metric=metric)
    kNN.fit(X)
    neighbour = kNN.kneighbors(business_vec, return_distance=False)
    for i in range(1, k+1):
        n = neighbour.item(i)
        neighbour_ids.append(business_inv_mapper[n])
    return neighbour_ids
```

`find_similar_businesses()` takes in a `businessId` and `X` matrix and outputs a list of (k) businesses that are similar to the `businessId` of interest.

Let's see how it works in action. We will first create another mapper that maps `businessId` to `title` so that our results are interpretable.

```
similar_businesses = find_similar_businesses(1, X, business_mapper, business_inv_mapper, k=10)
similar_businesses
```

`find_similar_businesses()` returns a list of `businessId`'s that are most similar to your business of interest. Let's convert these ids to titles so that we can interpret our results. To make things easier, we will create a dictionary that maps `businessId` to `title`.

```
business_titles = dict(zip(businesses['businessId'], businesses['title']))

business_id = 1

similar_businesses = find_similar_businesses(business_id, X, business_mapper, business_inv_mapper, metric='cosine', k=10)
business_title = business_titles[business_id]

print(f"Because you liked {business_title}:")
for i in similar_businesses:
    print(business_titles[i])
```

The results above show the 10 businesses that are most similar. Note that these recommendations are based solely on user-item ratings. Business features such as categories are not used in this approach.

We can also play around with the kNN distance metric and see what results you would get if you use "manhattan" or "euclidean" instead of "cosine".

```
business_id = 1

similar_businesses = find_similar_businesses(business_id, X, business_mapper, business_inv_mapper, metric='euclidean', k=10)
business_title = business_titles[business_id]

print(f"Because you liked {business_title}:")
for i in similar_businesses:
    print(business_titles[i])
```

Step 6: Handling the cold-start problem

Collaborative filtering relies solely on user-item interactions within the utility matrix. The issue with this approach is that brand new users or items with no interactions get excluded from the recommendation system. This is called the **cold start problem**. Content-based filtering is a way to handle this problem by generating recommendations based on user and item features.

First, we need to convert the `categories` column into binary features. Each category will have its own column in the dataframe and will be populated with 0 or 1.

```
n_businesses = businesses['businessId'].nunique()
print(f"There are {n_businesses} unique businesses in our dataset.")

categories = set(cat for sublist in businesses['categories'].apply(lambda x: x.split(',')).tolist() for cat in sublist)

for c in categories:
    businesses[c] = businesses['categories'].apply(lambda x: 1 if c in x else 0)

business_categories = businesses.drop(columns=['businessId', 'title', 'categories'])
business_categories.head()

from sklearn.metrics.pairwise import cosine_similarity

cosine_sim = cosine_similarity(business_categories, business_categories)
print(f"Dimensions of our categories cosine similarity matrix: {cosine_sim.shape}")
```

As expected, after passing the `business_categories` dataframe into the `cosine_similarity()` function, we get a cosine similarity matrix

businesses

This matrix is populated with values between 0 and 1 which represent the degree of similarity between businesses along the x and y axes.

Creating a business finder function

Let's say we want to get recommendations for businesses that are similar to Americano. To get results from our recommender, we need to know the exact title of a business in our dataset.

In our dataset, Americano is actually listed as 'Americano (1995)'. If we misspell Americano or forget to include its year of release, our recommender won't be able to identify which business we're interested in.

To make our recommender more user-friendly, we can use a Python package called [fuzzywuzzy](#) which will find the most similar title to a string that you pass in. Let's create a function called `business_finder()` which takes advantage of fuzzywuzzy's string matching algorithm to get the most similar title to a user-inputted string.

```
from fuzzywuzzy import process

def business_finder(title):
    all_titles = businesses['title'].tolist()
    closest_match = process.extractOne(title, all_titles)
    return closest_match[0]

# Let's test this out with our Jumanji example.
title = business_finder('americano')
title
```

To get relevant recommendations for Americano, we need to find its index in the cosine similarity matrix. To identify which row we should be looking at, we can create a business index mapper which maps a business title to the index that it represents in our matrix.

Let's create a business index dictionary called `business_idx` where the keys are business titles and values are business indices:

```
business_idx = dict(zip(businesses['title'], list(businesses.index)))
idx = business_idx[title]
print(f"Business index for Jumanji: {idx}")
Using this handy business_idx dictionary, we know that Jumanji is represented by index 1 in our matrix. Let's get the top 10 most :
n_recommendations = 10
sim_scores = list(enumerate(cosine_sim[idx]))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:(n_recommendations+1)]

similar_businesses = [i[0] for i in sim_scores]

print(f"Because you liked {title}:")
print(businesses['title'].iloc[similar_businesses])
```

Cool! These recommendations seem pretty relevant and similar to americano. The first 5 businesses are family-friendly

We can test our recommender further with other business titles. For your convenience, let's package the steps into a single function which takes in the business title of interest and number of recommendations.

```
def get_content_based_recommendations(title_string, n_recommendations=10):
    title = business_finder(title_string)
    idx = business_idx[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:(n_recommendations+1)]
    similar_businesses = [i[0] for i in sim_scores]
    print(f"Because you liked {title}:")
    print(businesses['title'].iloc[similar_businesses])
```

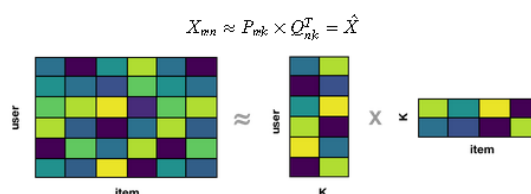
`get_content_based_recommendations('mexican', 5)`

Step 7: Dimensionality Reduction with Matrix Factorization (advanced)

Matrix factorization (MF) is a linear algebra technique that can help us discover latent features underlying the interactions between users and businesses. These latent features give a more compact representation of user tastes and item descriptions. MF is particularly useful for very sparse data and can enhance the quality of recommendations. The algorithm works by factorizing the original user-item matrix into two factor matrices:

- user-factor matrix $((n_{\text{users}}, k))$
- item-factor matrix $((k, n_{\text{items}}))$

We are reducing the dimensions of our original matrix into "taste" dimensions. We cannot interpret what each latent feature (k) represents. However, we could imagine that one latent feature may represent users who like romantic comedies from the 1990s, while another latent feature may represent movies which are independent foreign language .



```

from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=20, n_iter=10)
Q = svd.fit_transform(X)
Q.shape
Now, let's use the reduced dimensions to find similar businesses:
business_id = 1
similar_businesses = find_similar_businesses(business_id, Q, business_mapper, business_inv_mapper, metric='cosine', k=10)
business_title = business_titles[business_id]

print(f"Because you liked {business_title}.")
for i in similar_businesses:
    print(business_titles[i])

```

The results above are the most similar businesses to Mexican using kNN on our “compressed” business-factor matrix. We reduced the dimensions down to ($n_{\text{components}}=20$). We can think of each component representing a latent feature such as business genre.