Q.1: Write Application of Recommender System.

The technology that helps guide individuals towards products is a machine learning algorithm called a "recommender system." From the way we shop, to how we get our news, and even how we meet people, recommender systems are practically ubiquitous in our lives.

"We live in an attention economy, where there's an overwhelming number of things, and recommender systems help us make decisions," says Nitya Mandyam, Senior Curriculum Developer at Codecademy. "It's impossible to think of buying a shoe or a dress without having some kind of recommender system on the side giving you suggestions."

Recommender systems are algorithms that make recommendations to users about the best option to choose from a set of options. Of course, the "best" option is going to vary from person to person, which is why recommender systems turn to data about products and users' preferences to generate individualized suggestions. Recommender systems are ideal for situations where users have a lot of options to choose from — like deciding which show to stream on Netflix, or wading through the sea of products on Amazon. Online dating apps use recommender systems to match people with a potential romantic partner based on similar factors, like their location and hobbies. Even social media platforms use recommender systems to determine what shows up in your feed.

Recommender systems are so ingrained in our lives that we've come to expect these tailored suggestions and recommendations from the technology we use. As a developer, it's important to realize that users want systems that capture their tastes or interests — otherwise they might stop using it.

Tons of businesses rely on recommender systems to keep customers engaged with their product and earn more money. In ecommerce, for example, recommender systems can point customers to products that they're more inclined to buy based on their past behaviors and purchases. Businesses can also learn a lot about their customers based on this data, and use it to inform other decisions.

Types: There are several types of recommendation systems, including:

1. Content-based filtering: This type of system uses the characteristics of items that a user has liked in the past to recommend similar items.
2. Collaborative filtering: This type of system uses the past behaviour of users to recommend items that similar users have liked.
3. Hybrid: To generate suggestions, this kind of system combines content-based filtering and collaborative filtering techniques.
4. Matrix Factorization: Using this method, the user-item matrix is divided into two lower-dimension matrices that are then utilized to generate predictions.
5. Deep Learning: To train the user and item representations that are subsequently utilized to generate recommendations, these models make use of neural networks.

The choice of which type of recommendation system to use depends on the specific application and the type of data available. It's worth noting that recommendation systems are widely used and can have a significant impact on businesses and users. However, it's important to consider ethical considerations and biases that may be introduced to the system.

Recommender systems in machine learning have a wide array of applications across various industries. Here are some key applications:

E-commerce: Recommender systems suggest products to users based on their browsing history, purchase history, and preferences. This personalization enhances the shopping experience and can increase sales.

Entertainment: Streaming services like Netflix and Spotify use recommender systems to suggest movies, TV shows, and music based on users' past viewing or listening habits.

Social Media: Platforms like Facebook and Twitter use recommender systems to personalize users' feeds, showing content that is more likely to be of interest based on past interactions1.

Online Dating: Dating apps use recommender systems to suggest potential matches by comparing user profiles and preferences1.

Content Platforms: Websites that offer articles, blogs, or news use recommender systems to suggest relevant content to keep users engaged.

Retail: Recommender systems help in upselling and cross-selling by suggesting related products or accessories at the point of sale.

These systems significantly enhance user experience by providing personalized content and suggestions, leading to increased user engagement and satisfaction.

Q.2: What are Data Collection Method in Recommender System.

Recommender systems have become ubiquitous in our digital lives, influencing everything from the movies we watch to the products we buy. These powerful tools leverage machine learning to analyze user data and suggest items that align with their interests. But before these algorithms can work their magic, they need a foundation – a rich tapestry of data collected about users and the items they interact with. This article delves into the various methods employed to gather this crucial information, the lifeblood of recommender systems.

Unveiling User Preferences: Two Main Approaches Data collection in recommender systems can be broadly categorized into two main approaches: explicit and implicit.

Explicit Data Collection: This method involves directly asking users for their preferences. This can be achieved through various techniques:

1. Ratings and Reviews: Encouraging users to rate items (e.g., with stars or thumbs up/down) or write reviews provides valuable insights into their preferences and opinions.
2. Surveys and Questionnaires: Targeted surveys can gather user demographics, interests, and past experiences, helping build a user profile.
3. Wishlists and Watchlists: Allowing users to curate lists of desired items explicitly reveals their preferences.

Implicit Data Collection: This method gathers data from user interactions without directly asking for their preferences. It paints a picture of user behavior by capturing their digital footprints. Here are some common techniques:

1. Purchase History: Tracking what users buy reveals their past preferences and can be used to suggest similar items.
2. Browsing Behavior: Monitoring which items users view, click on, or add to their carts provides insights into their current interests.
3. Search Queries: Analyzing search terms used by users can unveil their needs and buying intent.
4. Clickstream Data: Recording the sequence of pages a user visits on a website or app helps understand their navigation patterns and areas of interest.
5. Engagement Metrics: Tracking metrics like time spent on a product page, adding items to a wishlist, or reading reviews indicates user interest.
6. Social Media Interactions: Analyzing likes, shares, and comments on social media platforms can reveal user preferences and affinities with certain brands or products.
7. Beyond Basic Interactions: Expanding the Data Net

While basic interaction data forms the core, recommender systems can leverage additional data sources to create a more comprehensive picture of users. Here are some examples:

Demographic Data: Age, location, gender, income, and other demographic information can be used to personalize recommendations based on user segments.

Content-Based Data: Information about the items themselves, such as genre, brand, technical specifications, or reviews, can be used to recommend similar items.

Contextual Data: Factors like time of day, location, and browsing device can be used to tailor recommendations to the user's current situation.

Social Network Data: Information about a user's friends and their preferences can be used to suggest items popular within their social circle (with appropriate privacy considerations).

Striking a Balance: Privacy and Data Collection The vast amount of data collected by recommender systems raises concerns about user privacy. It's crucial to strike a balance between gathering valuable data and protecting user information. Here are some best practices:

Transparency and User Consent: Be transparent about the data collected and how it's used. Obtain explicit user consent for data collection and provide clear opt-out options.

Data Anonymization and Aggregation: Data can be anonymized or aggregated before being used for training models, reducing the risk of identifying individual users.

Data Security Measures: Implement robust security measures to protect user data from unauthorized access or breaches. Regulation and Compliance: Adhere to relevant data privacy regulations like GDPR and CCPA.

Conclusion: Data – The Fuel for Effective Recommendations Data collection methods are the foundation for building effective recommender systems. By employing a combination of explicit and implicit techniques, along with additional data sources, these systems can create a rich user profile that fuels personalized recommendations. However, it's equally important to prioritize user privacy by adopting ethical data collection practices and ensuring compliance with regulations. As recommender systems continue to evolve, striking this balance will be key to creating a user experience that is both personalized and trustworthy.

Q.3: Build a Basic Recommender system

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

ratings = pd.read_csv('RC_Ratings.csv')
ratings.head()
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100836 non-null  int64  
 1   movieId    100836 non-null  int64  
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

```
ratings.size
```

```
403344
```

```
ratings.shape
```

```
(100836, 4)
```

```
ratings.columns
```

```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

```
ratings.describe()
```

```
              userId        movieId         rating     timestamp
count  100836.000000  100836.000000  100836.000000  1.008360e+05
mean      326.127564   19435.295718       3.501557  1.205946e+09
std       182.618491   35530.987199       1.042529  2.162610e+08
min         1.000000       1.000000       0.500000  8.281246e+08
25%       177.000000    1199.000000       3.000000  1.019124e+09
50%       325.000000    2991.000000       3.500000  1.186087e+09
75%       477.000000    8122.000000       4.000000  1.435994e+09
max       610.000000  193609.000000       5.000000  1.537799e+09
```

```
movies = pd.read_csv('RC_Movies.csv')
movies.head()
```

```
   movieId                                title  \
0        1                    Toy Story (1995)
1        2                      Jumanji (1995)
2        3             Grumpier Old Men (1995)
3        4            Waiting to Exhale (1995)
4        5  Father of the Bride Part II (1995)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                              Comedy|Romance
3                        Comedy|Drama|Romance
4                                      Comedy
```

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   title    9742 non-null   object
```

```
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

```
movies.size
```

```
29226
```

```
movies.shape
```

```
(9742, 3)
```

```
movies.columns
```

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

```
movies.describe()
```

```
             movieId
count    9742.000000
mean    42200.353623
std     52160.494854
min         1.000000
25%      3248.250000
50%      7300.000000
75%     76232.000000
max    193609.000000
```

Statistical Analysis of Ratings

```
n_ratings = len(ratings)
n_movies = len(ratings['movieId'].unique())
n_users = len(ratings['userId'].unique())

print(f"Number of Ratings: {n_ratings}")
print(f"Number of Unique MovieId's: {n_movies}")
print(f"Number of Unique Users: {n_users}")
print(f"Average Ratings Per User: {round(n_ratings/n_users, 2)}")
print(f"Average Ratings Per Movie: {round(n_ratings/n_movies, 2)}")
```

```
Number of Ratings: 100836
Number of Unique MovieId's: 9724
Number of Unique Users: 610
Average Ratings Per User: 165.3
Average Ratings Per Movie: 10.37
```

User Rating Frequency

```
user_freq = ratings[['userId', 'movieId']].groupby(
     'userId').count().reset_index()
```

```
user_freq.columns = ['userId', 'n_ratings']
print(user_freq.head())
```

```
   userId  n_ratings
0       1        232
1       2         29
2       3         39
3       4        216
4       5         44
```

Movie Rating Analysis

```
# Find Lowest and Highest rated movies:
mean_rating = ratings.groupby('movieId')[['rating']].mean()

# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies.loc[movies['movieId'] == lowest_rated]

# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies.loc[movies['movieId'] == highest_rated]

# show number of people who rated movies rated movie highest
ratings[ratings['movieId']==highest_rated]

# show number of people who rated movies rated movie lowest
ratings[ratings['movieId']==lowest_rated]

## the above movies has very low dataset. We will use bayesian average
movie_stats = ratings.groupby('movieId')[['rating']].agg(['count',
'mean'])
movie_stats.columns = movie_stats.columns.droplevel()
```

User-Item Matrix Creation

```
# Now, we create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix

def create_matrix(df):

    N = len(df['userId'].unique())
    M = len(df['movieId'].unique())

    # Map Ids to indices
    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    # Map indices to IDs
    user_inv_mapper = dict(zip(list(range(N)),
```

```
np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)),
np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

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

    return X, user_mapper, movie_mapper, user_inv_mapper,
movie_inv_mapper

X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper =
create_matrix(ratings)
```

Movie Similarity Analysis

```
"""
Find similar movies using KNN
"""

from sklearn.neighbors import NearestNeighbors

def find_similar_movies(movie_id, X, k, metric='cosine',
show_distance=False):
    neighbour_ids = []

    movie_ind = movie_mapper[movie_id]
    movie_vec = X[movie_ind]
    k += 1
    kNN = NearestNeighbors(n_neighbors=k, algorithm="brute",
metric=metric)
    kNN.fit(X)
    movie_vec = movie_vec.reshape(1, -1)
    neighbour = kNN.kneighbors(movie_vec,
return_distance=show_distance)
    for i in range(0, k):
        n = neighbour.item(i)
        neighbour_ids.append(movie_inv_mapper[n])
    neighbour_ids.pop(0)
    return neighbour_ids

movie_titles = dict(zip(movies['movieId'], movies['title']))

movie_id = 3
similar_ids = find_similar_movies(movie_id, X, k=10)
movie_title = movie_titles[movie_id]

print(f"Since you watched {movie_title}:")
```

```
for i in similar_ids:
    print(movie_titles[i])
```

```
Since you watched Grumpier Old Men (1995):
Grumpy Old Men (1993)
Striptease (1996)
Nutty Professor, The (1996)
Twister (1996)
Father of the Bride Part II (1995)
Broken Arrow (1996)
Bio-Dome (1996)
Truth About Cats & Dogs, The (1996)
Sabrina (1995)
Birdcage, The (1996)
```

Movie Recommendation with respect to Users Preference

```
def recommend_movies_for_user(user_id, X, user_mapper, movie_mapper,
movie_inv_mapper, k=10):
    df1 = ratings[ratings['userId'] == user_id]

    if df1.empty:
        print(f"User with ID {user_id} does not exist.")
        return

    movie_id = df1[df1['rating'] == max(df1['rating'])]
['movieId'].iloc[0]

    movie_titles = dict(zip(movies['movieId'], movies['title']))

    similar_ids = find_similar_movies(movie_id, X, k)
    movie_title = movie_titles.get(movie_id, "Movie not found")

    if movie_title == "Movie not found":
        print(f"Movie with ID {movie_id} not found.")
        return

    print(f"Since you watched {movie_title}, you might also like:")
    for i in similar_ids:
        print(movie_titles.get(i, "Movie not found"))
```

Reccommend The Movies

```
user_id = 150 # Replace with the desired user ID
recommend_movies_for_user(user_id, X, user_mapper, movie_mapper,
movie_inv_mapper, k=10)
```

```
Since you watched Twelve Monkeys (a.k.a. 12 Monkeys) (1995), you might
also like:
Pulp Fiction (1994)
```

```
Terminator 2: Judgment Day (1991)
Independence Day (a.k.a. ID4) (1996)
Seven (a.k.a. Se7en) (1995)
Fargo (1996)
Fugitive, The (1993)
Usual Suspects, The (1995)
Jurassic Park (1993)
Star Wars: Episode IV - A New Hope (1977)
Heat (1995)
```