Customer Segmentation Using data Science :

Introduction :

Customer segmentation in AI refers to the process of categorizing a company's customers into distinct groups based on various characteristics and behaviors using artificial intelligence techniques. These characteristics can include demographics, purchasing habits, online behavior, and more. The goal of customer segmentation is to better understand and target different customer groups with tailored marketing strategies, product recommendations, and services to enhance customer satisfaction and ultimately drive business growth. AI helps automate and refine this process by analyzing large amounts of data to identify meaningful customer segments.

Types of AI in Costomer Segmentation :

Artificial Intelligence (AI) can be applied to customer segmentation in various ways to enhance the accuracy and effectiveness of the process. Here are some common types of AI techniques used for customer segmentation:

1. Clustering Algorithms : AI algorithms like K-Means, hierarchical clustering, or DBSCAN can group customers based on their similarities in terms of purchasing behavior, demographics, or other features.

2. Classification Algorithms : Algorithms such as decision trees, random forests, or support vector machines can be used for predictive customer segmentation. These algorithms categorize customers into predefined segments based on historical data.

3. Neural Networks : Deep learning models, particularly neural networks, can analyze complex data and uncover hidden patterns in customer behavior. Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) can be used for sequential and image-based data, respectively.

4. Natural Language Processing (NLP) :

NLP techniques are employed to analyze customer reviews, feedback, or social media interactions, allowing for sentiment-based segmentation and understanding customer opinions.

5. Recommendation Systems :

Collaborative filtering and content-based recommendation systems use AI to segment customers based on their preferences and past interactions, enabling personalized recommendations.

6. **Anomaly Detection :** AI can identify unusual behavior or fraud by detecting anomalies in customer transactions, which

can be a form of segmentation for risk management.

7. Dimensionality Reduction : Techniques like Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) can reduce the dimensionality of data while retaining its structure, which can aid in customer segmentation.

8. Time-Series Analysis : For businesses with temporal data, AI techniques can segment customers based on their historical behaviors and changes over time.

9. Reinforcement Learning : In some cases, AI models can learn to segment customers by interacting with them and optimizing their segmentation based on business goals.

10. Hybrid Models : Many companies use a combination of these AI techniques to perform customer segmentation, as different approaches may be suitable for different aspects of the business.

1. <u>Clustering Algorithms :</u>

Example Input :

```
   import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
 # Sample data
   data = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8],
 [1, 0.6], [9, 11]])
 # Number of clusters
 k = 2
 # Create a K-Means model
 kmeans = KMeans(n_clusters=k)
 # Fit the model to the data
   kmeans.fit(data)
```

```
  # Get cluster centers and labelscentroids =
kmeans.cluster_centers_labels =
kmeans.labels_
# Visualize the data points and cluster centers
  colors = ["g.", "r."]
for i in range(len(data)):
    plt.plot(data[i][0], data[i][1],
colors[labels[i]], markersize=10)
      plt.scatter(centroids[:, 0], centroids[:, 1],
   marker="x", s=150, linewidths=5,
   zorder=10)
   plt.show()
```

<u>Example Output:</u>

```
   You should see a graphical window displaying a
scatter  plot with two clusters and their centroids. The
data points are colored based on their assigned
clusters, and the centroids are marked with 'x'
symbols.
```

In this example, the K-Means algorithm has clustered the data points into two distinct groups based on their proximity to the cluster centers. The actual output will depend on the data you use and the random initialization of the K-Means algorithm, so your specific plot might look slightly different.

2. Classification Algorithms :

Example Input :

```
# Import necessary libraries
import numpy as np
    from sklearn.tree import
    DecisionTreeClassifier
# Sample input data (features)
# Replace this with your own input data
  input_data = np.array([
 [5.1, 3.5],
 [4.9, 3.0],
 [6.7, 3.1],
[6.0, 3.4]])

# Sample labels (classes) for the input data

# Replace this with your own labels

labels = np.array(['Iris-setosa', 'Iris-setosa',

'Iris-versicolor', 'Iris-versicolor'])

# Create a Decision Tree classifier

classifier = DecisionTreeClassifier()

  # Train the classifier on the input data and

labels

classifier.fit(input_data, labels)

# User-provided input data for classification

user_input = np.array([[5.5, 3.5]])

# Predict the class for the user-provided input

    predicted_class =

classifier.predict(user_input)

# Output the predicted class to the user

  print("Predicted class for input data:",

predicted_class[0])
```

3. Neural Networks :

Example Input :

```
# Import necessary libraries
  import numpy as np
  from keras.models import Sequential
  from keras.layers import Dense
# Sample input data (exam scores)
# Replace this with your own data
  input_data = np.array([
    [90, 80],
    [85, 75],
    [70, 65],
   [60, 50]])
# Sample labels (pass/fail)
# Replace this with your own labels
```

```
  labels = np.array([1, 1, 0, 0])
# Create a Sequential model
  model = Sequential()
  # Add input layer with 2 input features and a
hidden layer with 4 neurons
      model.add(Dense(4, input_dim=2,
    activation='relu'))
    # Add output layer with 1 neuron (binary
classification)
  model.add(Dense(1, activation='sigmoid'))
# Compile the model
      model.compile(loss='binary_crossentropy
    ', optimizer='adam', metrics=['accuracy'])
  # Train the model on the input data and
labels
    model.fit(input_data, labels, epochs=1000,
verbose=0)
```

```
   # User-provided input data for prediction
     user_input = np.array([[75, 70]])
   # Predict the class (pass/fail) for the user-
 provided input
   predicted_class = model.predict(user_input)
 # Output the predicted class to the user
   If predicted_class > 0.5:
   print("Predicted class: Pass")
 else:
    print("Predicted class: Fail")
```

4. Natural Language Processing (NLP) :

Example Input :

```
 import nltk
     # Download NLTK data for tokenization,
sentence segmentation, POS tagging, and NER
```

```
  nltk.download('punkt')
  nltk.download('averaged_perceptron_tagge
r')
  nltk.download('maxent_ne_chunker')
  nltk.download('words')
 # Sample text for NLP processing
    text = "Albert Einstein was a German-born
   theoretical physicist who developed the
   theory of relativity, one of the two pillars
   of modern physics. He received the Nobel
   Prize in Physics in 1921 for his work on the
   photoelectric effect."
 # Tokenization
  tokens = nltk.word_tokenize(text)
 # Sentence segmentation
  sentences = nltk.sent_tokenize(text)
```

```
# Part-of-speech tagging
  pos_tags = nltk.pos_tag(tokens)
# Named Entity Recognition (NER)
  ner_tags = nltk.ne_chunk(pos_tags)
# Display the results
  print("Tokenization:")
  print(tokens)
  print("\nSentence Segmentation:")
  print(sentences)
  print("\nPart-of-Speech Tagging:")
  print(pos_tags)
  print("\nNamed Entity Recognition:")
  print(ner_tags)
```

5. Recommendation Systems :

Example Input :

```
  import pandas as pd
    from surprise import Dataset, Reader,
   KNNBasic
    from surprise.model_selection import
   train_test_split
  from surprise import accuracy
 # Load the MovieLens dataset (change the
  path to your dataset file)
   file_path =
 'path_to_movie_lens_dataset.csv'
 # Define the Reader object
    reader = Reader(line_format='user item
   rating timestamp', sep=',')
 # Load the dataset
```

```
    data    =    Dataset.load_from_file(file_path,
    reader=reader)
# Split the data into training and testing sets
     trainset,   testset   =   train_test_split(data,
    test_size=0.2)
  # Create a KNNBasic collaborative filtering
model
    sim_options = {
      'name': 'cosine',
        'user_based':    False       #    Item-based
    recommendation
  }
      model                                          =
    KNNBasic(sim_options=sim_options)
# Fit the model on the training data
    model.fit(trainset)
  # Make predictions on the test set
    predictions = model.test(testset)
    # Calculate and print the Root Mean
        Squared Error (RMSE) of the model
      rmse = accuracy.rmse(predictions)
      print(f'RMSE: {rmse:.4f}')
  # Recommend items for a specific user
      user_id = '1'  # Change to the user ID you
want to recommend movies for
  # Get a list of items the user has not rated
        items_to_predict = [item for item in
    trainset.all_items() if item not in
    trainset.ur[int(user_id)]]
  # Predict ratings for these items
        user_ratings = [model.predict(user_id,
    item) for item in items_to_predict]
    # Sort the predicted ratings and
recommend  the top N items
```

```
    top_n = 10  # Number of
    recommendations to    provide
user_ratings.sort(key=lambda       x:       x.est,
    reverse=True)
  top_items = user_ratings[:top_n]
# Print the top N recommended items
print(f"Top {top_n} Recommendations for
    User {user_id}:")
      for item in top_items:
      print(f"Item     ID:     {item.iid},     Estimated
Rating: {item.est}")
```

6. Anomaly Detection :

Example Input :

```
import numpy as np
```

```
from sklearn.ensemble import
IsolationForest
import matplotlib.pyplot as plt
# Generate synthetic data with anomalies
np.random.seed(42)
data = 0.2 * np.random.randn(1000, 2)
anomalies = np.random.uniform(low=-6,
high=6, size=(20, 2))
data = np.vstack([data, anomalies])
# Create an Isolation Forest model model
IsolationForest(contamination=0.02,
random_state=42)
# Fit the model to the data
model.fit(data)
# Predict the anomaly scores for the
data points
anomaly_scores =
model.decision_function(data)
```

```
# Visualize the data and highlight anomalies
plt.scatter(data[:, 0], data[:, 1],
c=anomaly_scores, cmap='viridis')
plt.colorbar(label='Anomaly Score')
plt.title('Anomaly Detection with Isolation
Forest')
plt.show()
# Find and display the anomalies
anomalies_indices =
np.where(anomaly_scores < 0)
print("Detected Anomalies:")
for index in anomalies_indices[0]:
print(data[index])
```

7. Dimensionality Reduction :

Example Input :

```
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.decomposition import PCA
    from sklearn.datasets import load_iris
# Load the Iris dataset as an example
  data = load_iris()
  X = data.data
  y = data.target
# Perform PCA for dimensionality reduction
  pca = PCA(n_components=2)
  X_reduced = pca.fit_transform(X)
# Variance explained by the selected
  components
```

```
      explained_variance
    pca.explained_variance_ratio_
# Scatter plot of the reduced data
    plt.figure(figsize=(8, 6))
      plt.scatter(X_reduced[:,   0],   X_reduced[:,
1],  c=y, cmap=plt.cm.Set1)
    plt.title('PCA of Iris Dataset')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()


#   Output   the   variance   explained   by   the
    selected components
print("Variance explained by each
      component:", explained_variance)
```

8. Time-Series Analysis :

Example Input :

```
import pandas as pd
import matplotlib.pyplot as plt
 # Sample time series data (replace with
 your own data)
data = {
'Date': ['2023-01-01', '2023-01-02', '2023-01-
 03', '2023-01-04', '2023-01-05'],
  'Value': [10, 12, 15, 11, 13]
}
# Create a pandas DataFrame
 df = pd.DataFrame(data)
 # Convert the 'Date' column to a datetime
 object
 df['Date'] = pd.to_datetime(df['Date'])
```

```
  # Set the 'Date' column as the index
  df.set_index('Date', inplace=True)


  # Plot the time series data
  plt.figure(figsize=(10, 6))
    plt.plot(df.index,    df['Value'],    marker='o',
linestyle='-', color='b')
  plt.title('Time Series Data')
  plt.xlabel('Date')
  plt.ylabel('Value')
  plt.grid()
  plt.show()
```

9. Reinforcement Learning :

Example Input :

```
import numpy as np

import gym

# Create the FrozenLake environment

 env = gym.make('FrozenLake-v1')

# Q-learning parameters

 learning_rate = 0.8

 discount_factor = 0.95

epsilon = 0.2

 num_episodes = 1000


# Initialize Q-table with zeros

 num_states = env.observation_space.n

num_actions = env.action_space.n

 Q = np.zeros((num_states, num_actions))
```

```
# Q-learning algorithm
  for episode in range(num_episodes):
 state = env.reset()
  done = False
while not done:
   # Exploration vs. exploitation
   if np.random.uniform(0, 1) < epsilon:
       action   =   env.action_space.sample()     #
Explore
    else:
        action = np.argmax(Q[state, :])  # Exploit
  # Take the action
    next_state,      reward,      done,      _      =
    env.step(action)
  # Q-value update
```

```
   Q[state,    action]    =    Q[state,    action]    +
  learning_rate * (reward + discount_factor *
  np.max(Q[next_state, :]) - Q[state, action])
 state = next_state
  # Evaluate the trained Q-table
  num_episodes = 100
  total_rewards = 0
  for episode in range(num_episodes):
 state = env.reset()
 done = False
while not done:
   action = np.argmax(Q[state, :)
   next_state,      reward,      done,      _      =
    env.step(action)
   total_rewards += reward
   state = next_state
     average_reward     =     total_rewards     /
num_episodes
```

```
    print(f'Average              reward              over
  {num_episodes}  episodes: {average_reward}')
  # Test the trained agent
    state = env.reset()
    done = False
    env.render()
  while not done:
   action = np.argmax(Q[state, :])
   state, reward, done, _ = env.step(action)
   env.render()
```

10. Hybrid Models :

Example Input :

```
    import numpy as np
    import pandas as pd
```

```
from sklearn.metrics.pairwise import
cosine_similarity
from sklearn.feature_extraction.text
import TfidfVectorizer
from sklearn.metrics import
pairwise_distances
# Sample movie data (you can replace this
with your own data)
movies_data = pd.DataFrame({
'movie_id': [1, 2, 3, 4, 5],
' movie_title': ['Movie A', 'Movie B', 'Movie C',
'Movie D', 'Movie E'],
'genre': ['Action', 'Comedy', 'Drama', 'Action',
'Comedy'],})
# Sample user data
user_data = pd.DataFrame({
'movie_id': [1, 2],
'rating': [5, 4],})
```

```
  # Collaborative filtering recommendation
    collab_filtered_movie_id = 3   # Movie for
    which we want to make recommendations
  collab_filtered_recommendations =
movies_data[~movies_data['movie_id'].isin(u
ser_data['movie_id'])]
      collab_filtered_recommendations['collab
_similarity'] = np.nan
  for index, row in
collab_filtered_recommendations.iterrows():
  similarity = 0  # Calculate similarity with user
  data (e.g., using user-item matrix)
  collab_filtered_recommendations.at[index,
  'collab_similarity'] = similarity
      collab_filtered_recommendations =
    collab_filtered_recommendations.sort_val
    ues(by='collab_similarity', ascending=False)
    collab_filtered_top_recommendation        =
    collab_filtered_recommendations.iloc[0]
# Content-based recommendation
  tfidf_vectorizer = TfidfVectorizer()
       genre_matrix                           =
    tfidf_vectorizer.fit_transform(movies_data
    ['genre'])
        content_based_recommendations      =
    movies_data[~movies_data['movie_id'].isin
    (user_data['movie_id'])]
      content_based_recommendations['conte
n t_similarity'] = np.nan


    for          index,          row          in
    content_based_recommendations.iterrow
    s():
movie_vector                                  =
  tfidf_vectorizer.transform([row['genre']])
```

```
 similarity   =   cosine_similarity(movie_vector,
   genre_matrix)
 content_based_recommendations.at[index,  '
     content_similarity'] = similarity
   content_based_recommendations              =
content_based_recommendations.sort_value
s(by='content_similarity', ascending=False)
     content_based_top_recommendation     =
   content_based_recommendations.iloc[0]
     #   Hybrid   recommendation   (weighted
   combination)
   alpha  =  0.7    #  Weight  for  collaborative
   filtering
   beta = 0.3  # Weight for content-based
      hybrid_recommendations                    =
   movies_data[~movies_data['movie_id'].isin
   (user_data['movie_id'])]
```

```
hybrid_recommendations['hybrid_score']          =
 np.nan
    for            index,            row            in
  hybrid_recommendations.iterrows():
   collab_score = row['collab_similarity'] if not
    np.isnan(row['collab_similarity']) else 0
  content_score  =  row['content_similarity']  if
    not  np.isnan(row['content_similarity'])  else
    0
  hybrid_score = alpha * collab_score + beta *
    content_score
  hybrid_recommendations.at[index,
    'hybrid_score'] = hybrid_score
      hybrid_recommendations                     =
     hybrid_recommendations.sort_values(by='
     hybrid_score', ascending=False)
    hybrid_top_recommendation                   =
  hybrid_recommendations.iloc[0]
  print("Collaborative Filtering Recommendation:", collab_filtered_top_recommendation['movie_title'])
  print("Content-Based Recommendation:", content_based_top_recommendation['movie_title'])
  print("Hybrid Recommendation:", hybrid_top_recommendation['movie_title'])
```

<u>Thankyou</u>