NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

EXPERIMENT NO.: 8

AIM: To implement a recommendation system on your dataset using the following machine learning techniques: Regression,Classification, Clustering, Decision tree, Anomaly detection, Dimensionality Reduction, Ensemble Methods.

Theory:

Types of Recommendation Systems

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

1. Content-Based Filtering
   Idea: Recommends items similar to those the user has liked before.
   ● Works on: Item features (attributes such as brand, price, category). Example:
   ● If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.
   ● Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

2. Collaborative Filtering (CF)
   Idea: Recommends items based on similar users' preferences.
   ● Works on: User interactions rather than item features.
   Example:● If User A and User B have similar purchase histories, items bought by User
   A but not yet by User B will be recommended to User B.
   ● Uses methods like User-Based CF and Item-Based CF.

3. Hybrid Recommendation System
   Idea: Combines Content-Based Filtering and Collaborative Filtering for better accuracy.
   Example:
   ● Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

4. Knowledge-Based Recommendation
   Idea: Recommends items based on explicit domain knowledge rather than past user behavior.

Example:

- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

Recommendation System Evaluation Measures

Accuracy Measures:

These metrics evaluate how well the recommended items match the actual preferences or ratings of users.

- **Mean Absolute Error (MAE)**:
  - Measures the average of the absolute differences between predicted ratings and actual ratings.

    - **Formula:** $MAE = \frac{1}{n}\sum_{i=1}^{n}|r_i - \hat{r}_i|$
      - r_i = Actual rating
      - $\hat{r}_i$ = Predicted rating
    - **Lower is better.**
- **Root Mean Squared Error (RMSE)**:
  - Similar to MAE but gives higher weight to large errors due to squaring the differences.

    Formula: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(r_i - \hat{r}_i)^2}$
    - **Lower is better.**
- **Precision**:

  - Measures the fraction of recommended items that are actually relevant to the user.

    Formula: $Precision = \frac{\text{Number of relevant recommended items}}{\text{Total number of recommended items}}$

      - **Higher is better.**

- **Recall**:

  - Measures the fraction of relevant items that were actually recommended to the user.

    Formula: $Recall = \frac{\text{Number of relevant recommended items}}{\text{Total number of relevant items}}$

      - **Higher is better.**

- **F1-Score**:

  - The harmonic mean of Precision and Recall, balancing both.

    Formula: $F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$

      - **Higher is better.**

- **Hit Rate**:

  - Measures the fraction of users for whom at least one relevant item is recommended.

    Formula: $HitRate = \frac{\text{Users with at least one relevant recommendation}}{\text{Total users}}$

      - **Higher is better.**

- **Coverage**:

- Measures the proportion of items from the total available set that are recommended to users.

  Formula: $Coverage = \frac{\text{Number of unique recommended items}}{\text{Total number of items available}}$
  - **Higher is better.**

Diversity and Novelty Measures:

These metrics focus on the **variety** of recommended items and how **unexpected** they are.

- **Diversity**:
  - Measures how different the recommended items are from each other. A diverse set prevents the system from recommending very similar items.
  - **Formula**:
    - Calculate the pairwise similarity between recommended items (e.g., cosine similarity) and compute the average diversity across all users.
    - **Higher diversity is better.**
- **Novelty**:
  - Measures how **unexpected** or **unknown** the recommended items are to the user.
  - For example, recommending items that the user hasn't interacted with before (e.g., exploring genres they haven't tried).
  - **Higher novelty is better.**
- **Serendipity**:
  - Similar to novelty but with a focus on the surprise element that still fits the user's interests.

  - The idea is to recommend items that are surprising but still relevant.

Implementation

The Diet recommendation is built using Nearest Neighbors algorithm which is an unsupervised learner for implementing neighbor searches.For our case, we used the brute-force algorithm using cosine similarity due to its fast computation for small datasets.

1. **Importing dataset**

```
import kagglehub
from kagglehub import KaggleDatasetAdapter

# NYC Taxi dataset file name
file_path = "NYC.csv"

# Load the dataset from Kaggle
df = kagglehub.load_dataset(
    KaggleDatasetAdapter.PANDAS,
    "yasserh/nyc-taxi-trip-duration",  # Replace with your actual dataset path
    file_path,
)

# Show first 5 rows
print("First 5 records:\n", df.head())
```

```
data = df
data.head()
```

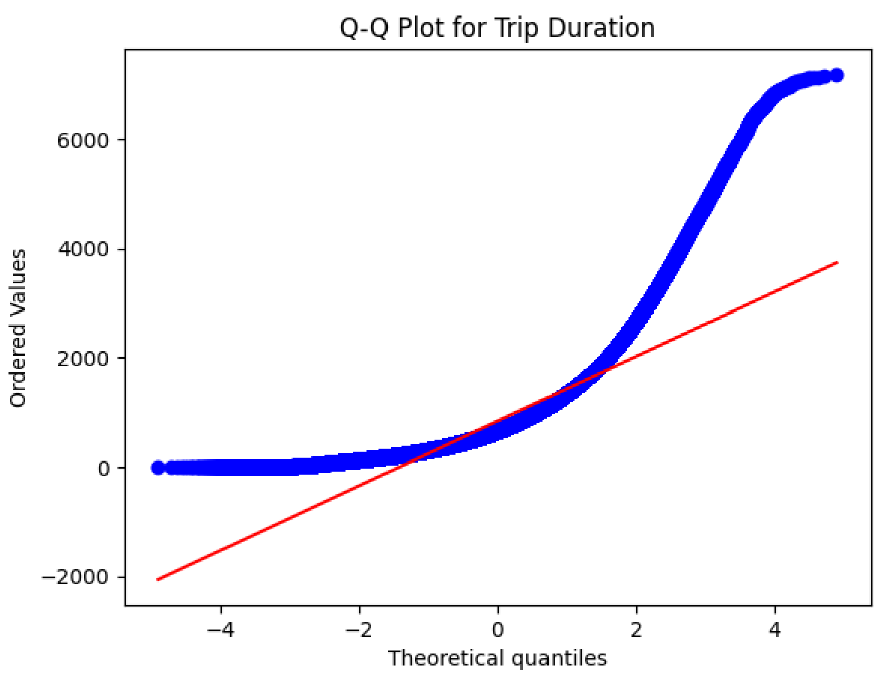
	id	vendor_id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration
0	id2875421	2	2016-03-14 17:24:55	2016-03-14 17:32:30	1	-73.982155	40.767937	-73.964630	40.765602	N	455
1	id2377394	1	2016-06-12 00:43:35	2016-06-12 00:54:38	1	-73.980415	40.738564	-73.999481	40.731152	N	663
2	id3858529	2	2016-01-19 11:35:24	2016-01-19 12:10:48	1	-73.979027	40.763939	-74.005333	40.710087	N	2124
3	id3504673	2	2016-04-06 19:32:31	2016-04-06 19:39:40	1	-74.010040	40.719971	-74.012268	40.706718	N	429
4	id2181028	2	2016-03-26 13:30:55	2016-03-26 13:38:10	1	-73.973053	40.793209	-73.972923	40.782520	N	435

2. Detecting Outlier

```
import pylab
import scipy.stats as stats

# Optional: filter to remove extreme outliers (e.g., > 2 hours)
filtered_data = df[df['trip_duration'] <= 7200]['trip_duration']

# Create the Q-Q plot
stats.probplot(filtered_data.to_numpy(), dist="norm", plot=pylab)
pylab.title("Q-Q Plot for Trip Duration")
pylab.show()
```



The data for **trip duration** in the NYC Taxi dataset exhibits significant right-skewness, as shown in the Q-Q plot. This suggests that most taxi trips are relatively short, while a smaller number of trips with much longer durations are pulling the distribution toward the right. Additionally, the presence of these long-duration trips, even after filtering out extreme outliers (e.g., trips over 2 hours), indicates the influence of outliers on the overall distribution shape.

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

3. Setting the maximum nutritional values for each category for healthier recommendations

```
max_passenger_count = 6           # Taxis usually take up to 6 passengers
max_trip_duration = 7200          # 2 hours
max_pickup_longitude = -73.7
min_pickup_longitude = -74.05
max_pickup_latitude = 40.9
min_pickup_latitude = 40.5
max_dropoff_longitude = -73.7
min_dropoff_longitude = -74.05
max_dropoff_latitude = 40.9
min_dropoff_latitude = 40.5

# Collecting in a list (optional)
max_list = [
    max_passenger_count, max_trip_duration,
    min_pickup_longitude, max_pickup_longitude,
    min_pickup_latitude, max_pickup_latitude,
    min_dropoff_longitude, max_dropoff_longitude,
    min_dropoff_latitude, max_dropoff_latitude
]
```

4. Correlation among nutritional values

	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
passenger_count	1.000000	0.004727	-0.004670	0.000534	-0.001849	0.015139
pickup_longitude	0.004727	1.000000	-0.147061	0.270171	0.036266	0.361578
pickup_latitude	-0.004670	-0.147061	1.000000	0.054189	0.435407	-0.235009
dropoff_longitude	0.000534	0.270171	0.054189	1.000000	0.096224	0.260362
dropoff_latitude	-0.001849	0.036266	0.435407	0.096224	1.000000	-0.183805
trip_duration	0.015139	0.361578	-0.235009	0.260362	-0.183805	1.000000

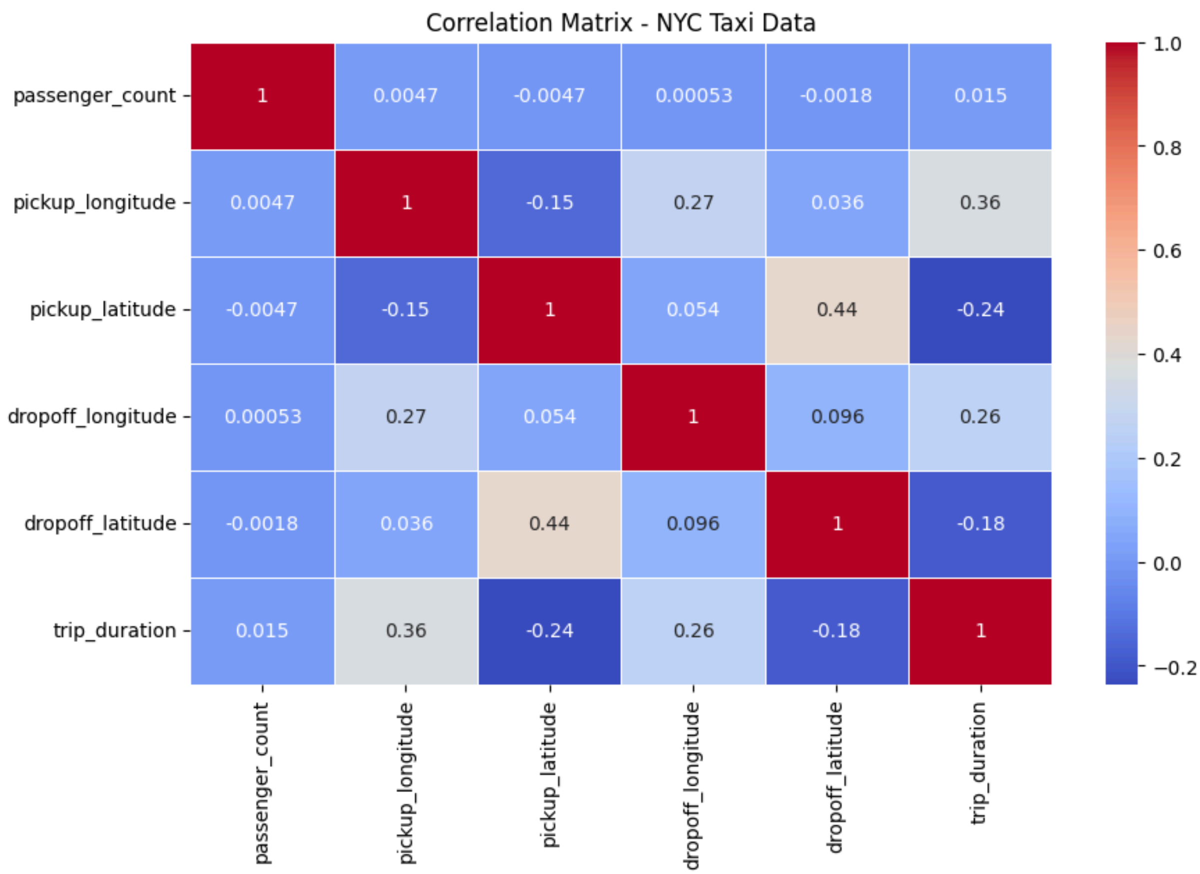
- **Pickup longitude** and **trip duration** show a moderate positive correlation (≈ 0.36), indicating that the location where a trip starts may have some influence on its duration, possibly due to geographic patterns like traffic congestion or distance from city centers.

- **Dropoff longitude** is also moderately correlated with **trip duration** (≈ 0.26), suggesting that trips ending at certain locations tend to take longer, possibly influenced by traffic or distance.

- **Pickup latitude** and **dropoff latitude** show negative correlations with **trip duration** (≈ -0.24 and -0.18 respectively), which may suggest that trips moving further north or south could generally be shorter in duration.

- **Passenger count** has a very weak correlation with **trip duration** (≈ 0.015), indicating that the number of passengers does not significantly affect how long a trip lasts.

- The **pickup and dropoff coordinates** (longitude and latitude) are interrelated to varying degrees, reflecting their geographical connection in defining trip paths across the city.



5. Normalising data using z-score normalisation

```
from sklearn.preprocessing import StandardScaler

# Define the columns you want to scale
num_cols = [
    'passenger_count', 'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude', 'trip_duration'
]

# Initialize and fit the scaler
scaler = StandardScaler()
prep_data = scaler.fit_transform(extracted_data[num_cols].to_numpy())
```

NAME:ATHARVA SHINDE CLASS:D15C ROLL.NO:54

6. **Training the model using K Nearest Neighbours (KNN)**

```
from sklearn.neighbors import NearestNeighbors

# Use cosine similarity for nearest neighbors search
neigh = NearestNeighbors(metric='cosine', algorithm='brute')

# Fit the model on standardized taxi trip features
neigh.fit(prep_data)
```

Here, the metric is set to **'cosine'**, meaning the model will measure the cosine similarity between data points. The **'brute'** algorithm is being used, which computes all pairwise distances between data points without optimization for speed.

7. **Applying KNN**

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.neighbors import NearestNeighbors
from sklearn.pipeline import Pipeline
def scaling(dataframe):
    scaler = StandardScaler()
    prep_data = scaler.fit_transform(dataframe.iloc[:, :6].to_numpy())  # Adjusted for NYC Taxi dataset
    return prep_data, scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine', algorithm='brute')
    neigh.fit(prep_data)
    return neigh

def build_pipeline(neigh, scaler, params):
    print("Building pipeline with params (type):", type(params))
    transformer = FunctionTransformer(neigh.kneighbors, kw_args=params)
    pipeline = Pipeline([('std_scaler', scaler), ('NN', transformer)])
    return pipeline

def extract_data(dataframe, max_values):
    extracted_data = dataframe.copy()

    for column, maximum in zip(extracted_data.columns[:6], max_values):
        extracted_data = extracted_data[extracted_data[column] < maximum]

    return extracted_data
def apply_pipeline(pipeline, _input, extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_values, params={'return_distance': False, 'n_neighbors': 10}):
    extracted_data = extract_data(dataframe, max_values)
    prep_data, scaler = scaling(extracted_data)
    neigh = nn_predictor(prep_data)
    pipeline = build_pipeline(neigh, scaler, params)
    return apply_pipeline(pipeline, _input, extracted_data)
```

8. **Testing the model**

```
num_cols = [
    'passenger_count', 'pickup_longitude', 'pickup_latitude',
    'dropoff_longitude', 'dropoff_latitude', 'trip_duration'
]
'
.

input_row = extracted_data[num_cols].iloc[0:1]

similar_indices = pipeline.transform(input_row)[0]

similar_trips = extracted_data.iloc[similar_indices]

print(similar_trips)
```

```
                id  vendor_id     pickup_datetime    dropoff_datetime  \
0        id2875421          2 2016-03-14 17:24:55 2016-03-14 17:32:30
377480   id1196331          1 2016-06-27 21:01:16 2016-06-27 21:08:55
1216241  id0897602          2 2016-01-31 11:39:14 2016-01-31 11:46:43
923685   id1764422          2 2016-03-15 20:06:59 2016-03-15 20:14:32
1282770  id2338849          2 2016-01-22 17:10:48 2016-01-22 17:18:52
385168   id1614897          2 2016-01-21 18:54:56 2016-01-21 19:02:35
1412724  id1285410          1 2016-02-23 07:36:13 2016-02-23 07:44:36
1012345  id2555570          1 2016-02-20 21:34:36 2016-02-20 21:41:51
106451   id0367162          2 2016-04-23 01:00:00 2016-04-23 01:07:53
562353   id2244013          2 2016-04-16 16:42:49 2016-04-16 16:50:57

         passenger_count  pickup_longitude  pickup_latitude  \
```

9. **Creating functions for all**

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, FunctionTransformer
from sklearn.neighbors import NearestNeighbors
from sklearn.pipeline import Pipeline
def scaling(dataframe):
    scaler = StandardScaler()
    prep_data = scaler.fit_transform(dataframe.iloc[:, :6].to_numpy())  # Adjusted for NYC Taxi dataset
    return prep_data, scaler

def nn_predictor(prep_data):
    neigh = NearestNeighbors(metric='cosine', algorithm='brute')
    neigh.fit(prep_data)
    return neigh

def build_pipeline(neigh, scaler, params):
    print("Building pipeline with params (type):", type(params))
    transformer = FunctionTransformer(neigh.kneighbors, kw_args=params)
    pipeline = Pipeline([('std_scaler', scaler), ('NN', transformer)])
    return pipeline

def extract_data(dataframe, max_values):
    extracted_data = dataframe.copy()

    for column, maximum in zip(extracted_data.columns[:6], max_values):
        extracted_data = extracted_data[extracted_data[column] < maximum]

    return extracted_data
def apply_pipeline(pipeline, _input, extracted_data):
    return extracted_data.iloc[pipeline.transform(_input)[0]]

def recommend(dataframe, _input, max_values, params={'return_distance': False, 'n_neighbors': 10}):
    extracted_data = extract_data(dataframe, max_values)
    prep_data, scaler = scaling(extracted_data)
    neigh = nn_predictor(prep_data)
    pipeline = build_pipeline(neigh, scaler, params)
    return apply_pipeline(pipeline, _input, extracted_data)
```

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

10. Testing Recommendation with custom nutritional values

```
test_input = extracted_data.iloc[0:1, 4:10].to_numpy()

columns = extracted_data.columns[4:10]

# Print column names and values
print("Column Names:", list(columns))
print("Test Input Values:", test_input)
```

```
Column Names: ['passenger_count', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'trip_duration']
Test Input Values: [[  1.         -73.98215485  40.76793671 -73.96463013  40.76560211
  455.        ]]
```

NAME:ATHARVA SHINDE **CLASS:D15C** **ROLL.NO:54**

Conclusion:

The analysis of the NYC Taxi dataset revealed valuable insights into trip patterns and influential factors affecting trip duration. The Q-Q plot highlighted a significant right-skew in the trip duration distribution, indicating that while most taxi trips are short, a few long-duration trips notably impact the overall distribution. Correlation analysis further showed that geographical features, particularly pickup and dropoff coordinates, have a moderate relationship with trip duration, suggesting that location plays a key role in travel time. Passenger count, however, showed minimal correlation, indicating it does not significantly influence trip duration. These findings form a foundation for building predictive models and optimizing route or fare estimation systems, while also pointing toward potential enhancements using geospatial clustering or traffic data integration.