BIG DATA ANALYTICS WITH IBM CLOUD

DEVELOPMENT 1:

To develop queries or scripts to explore and analyze a dataset in a database, you'll need to adapt the code to the specific database service you're using (e.g., Db2 or MongoDB) and the dataset you're working with. Below, I'll provide examples for both structured data (Db2) and semi-structured data (MongoDB) and demonstrate basic data cleaning and transformation operations.

- Structured Data (e.g., Db2):

Suppose you have a structured dataset in a Db2 database. You can use SQL queries for exploration and analysis:

- Select Data:

Retrieve a sample of data to get an overview of what's in the dataset. SELECT *
FROM your_table
FETCH FIRST 10 ROWS ONLY;

- Data Cleaning:

Remove
duplicates
DELETE
FROM
your_table
WHERE
ROWID NOT
IN (SELECT
MAX(ROWI
D)
FROM your_table
GROUP BY column1, column2
);

- Data Transformation:

  Calculate the sum or average of a

  numerical column SELECT

  AVG(numeric_column) AS avg_value

  FROM your_table;

➢ Semi-Structured Data (e.g., MongoDB):

If you are working with semi-structured data in a MongoDB database, you can use MongoDB query operations for exploration and analysis:

- Select Data:

  Retrieve documents that meet specific

  criteria. db.collection.find({ field: value

  }).limit(10);

- Data Cleaning:

  Remove documents with missing or null values in a

  specific field. db.collection.deleteMany({ field: null });

- Data Transformation:

  Perform an aggregation to calculate the average of a numeric field.

```
db.collection.aggregate([
  {
    $group: {
      _id: null,
      avg_value: { $avg: "$numeric_field" }}}]);
```

EXAMPLE 1:

```
import pandas as pd
import numpy as np

# Replace 'your_dataset.csv' with the actual file path or URL to your dataset
df = pd.read_csv('your_dataset.csv')

print(df.head())
print(df.describe())
print(df.dtypes)
print(df.isnull().sum())

# Drop rows with missing values
df.dropna(inplace=True)

# Fill missing values with a specific value
df['column_name'].fillna(value, inplace=True)

df = df.drop_duplicates()
df['new_column'] = df['column1'] + df['column2']
df['numeric_column'] = df['numeric_column'].astype(float)
df['column'] = df['column'].apply(lambda x: your_function(x))

# Example: Filter rows where a condition is met
filtered_data = df[df['column_name'] > 50]

# Example: Group by a column and calculate the mean of another column
grouped_data = df.groupby('grouping_column')['numeric_column'].mean()
```

EXAMPIE 2

```
import pandas as pd
import numpy as np

data = pd.read_csv('your_data.csv')
# Check for missing values
missing_values = data.isnull().sum()

# If you have missing values, you can handle them in various ways:
#  Remove rows with missing values
data.dropna(inplace=True)

#  Fill missing values with a specific value
data.fillna(value, inplace=True)

#  Interpolate missing values
data.interpolate(inplace=True)
data.drop_duplicates(inplace=True)

#  Convert a column to a different data type (e.g., from object to datetime)
data['date_column'] = pd.to_datetime(data['date_column'])
data.rename(columns={'old_column_name': 'new_column_name'}, inplace=True)

#  Normalize a numeric column
data['numeric_column'] = (data['numeric_column'] - data['numeric_column'].mean()) /
data['numeric_column'].std()
# Create a new column based on existing columns
data['new_column'] = data['column1'] + data['column2']

# Identify and handle outliers, e.g., by replacing them with the mean or median
outliers = data['numeric_column'][np.abs(data['numeric_column'] - data['numeric_column'].mean()) > 3 *
data['numeric_column'].std()]
data['numeric_column'][outliers.index] = data['numeric_column'].median()

data.to_csv('cleaned_data.csv', index=False)
```