**Project Title: Cloud Application**

**Topic**: Big data analysis with IBM cloud data base

Big data analytics describes the process of uncovering trends, patterns, and correlations in large amounts of raw data to help make data-informed decisions. These processes use familiar statistical analysis techniques—like clustering and regression—and apply them to more extensive datasets with the help of newer tools.

Big data has been a buzz word since the early 2000s, when software and hardware capabilities made it possible for organizations to handle large amounts of unstructured data. Since then, new technologies—from Amazon to smartphones—have contributed even more to the substantial amounts of data available to organizations. With the explosion of data, early innovation projects like Hadoop, Spark, and NoSQL databases were created for the storage and processing of big data.

This field continues to evolve as data engineers look for ways to integrate the vast amounts of complex information created by sensors, networks, transactions, smart devices, web usage, and more. Even now, big data analytics methods are being used with emerging technologies, like machine learning, to discover and scale more complex insights.

Works:

Big data analytics refers to collecting, processing, cleaning, and analyzing large datasets to help organizations operationalize their big data.

1. Collect Data

Data collection looks different for every organization. With today’s technology, organizations can gather both structured and unstructured data from a variety of sources — from cloud storage to mobile applications to in-store IoT sensors and beyond. Some data will be stored in data warehouses where business intelligence tools and solutions can access it easily. Raw or unstructured data that is too diverse or complex for a warehouse may be assigned metadata and stored in a data lake.

1. Creating a Dataset from Lists:

Program:

import pandas as pd

# Sample data as lists

names = ['Alice', 'Bob', 'Charlie', 'David']

ages = [25, 30, 35, 28]

# Creating a dictionary from the lists

data = {'Name': names, 'Age': ages}

# Creating a pandas DataFrame (dataset) from the dictionary

dataset = pd.DataFrame(data)

print(dataset)

O/p:

Name Age

0 Alice 25

1 Bob 30

2 Charlie 35

3 David 28

2. Creating a Dataset from a Dictionary:

Program:

import pandas as pd

# Sample data as a dictionary

data = {'Name': ['Alice', 'Bob', 'Charlie', 'David'],'Age': [25, 30, 35, 28]}

# Creating a pandas DataFrame (dataset) from the dictionary

dataset = pd.DataFrame(data)

print(dataset)

Output:

Name Age

0 Alice 25

1 Bob 30

2 Charlie 35

3 David 28

2. Process Data:

Once data is collected and stored, it must be organized properly to get accurate results on analytical queries, especially when it’s large and unstructured. Available data is growing exponentially, making data processing a challenge for organizations. One processing option is batch processing, which looks at large data blocks over time. Batch processing is useful when there is a longer turnaround time between collecting and analyzing data. Stream processing looks at small batches of data at once, shortening the delay time between collection and analysis for quicker decision-making. Stream processing is more complex and often more expensive.

1. Viewing Data:

print(dataset.head())

1. Selecting Columns:

Program:

selected\_column = dataset['ColumnName']

# or for multiple columns

selected\_columns = dataset[['Column1', 'Column2']]

1. Filtering Data:

Program:

filtered\_data = dataset[dataset['Column'] > 30]

1. Descriptive Statistics:

Program:

mean\_value = dataset['Column'].mean()

sum\_value = dataset['Column'].sum()

count\_value = dataset['Column'].count()

1. Sorting Data:

Program:

sorted\_dataset = dataset.sort\_values(by='Column', ascending=False)

1. Grouping Data:

Program

grouped\_data = dataset.groupby('Column').mean()

1. Handling Missing Data:

Program:

cleaned\_dataset = dataset.dropna() # Drops rows with NaN values

# Or fill NaN values with a specific value

filled\_dataset = dataset.fillna(value=0)

1. Adding New Columns:

Program:

dataset['NewColumn'] = dataset['Column1'] + dataset['Column2']

1. Saving Processed Data:

Program:

processed\_data.to\_csv('processed\_data.csv', index=False)

3. Clean Data:

Data big or small requires scrubbing to improve data quality and get stronger results; all data must be formatted correctly, and any duplicative or irrelevant data must be eliminated or accounted for. Dirty data can obscure and mislead, creating flawed insights.

1. Handling Missing Values:

dataset.dropna(inplace=True)

Missing values

dataset.fillna(value=0, inplace=True) # Fill NaN values with 0

1. Removing Duplicates:

dataset.drop\_duplicates(inplace=True)

1. Handling Outliers:

# Define a function to identify outliers using IQR (Interquartile Range)

def identify\_outliers(column):

Q1 = column.quantile(0.25)

Q3 = column.quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = column[(column < lower\_bound) | (column > upper\_bound)]

return outliers

outliers = identify\_outliers(dataset['Column'])

print("Outliers:", outliers)

Removing outliers:

# Remove outliers from a specific column

dataset = dataset[(dataset['Column'] > lower\_bound) & (dataset['Column'] < upper\_bound)]

1. Analyze Data:

Getting big data into a usable state takes time. Once it’s ready, advanced analytics processes can turn big data into big insights. Some of these big data analysis methods

1. Descriptive Statistics:

Program:

# Summary statistics for numerical columns

print(dataset.describe())

# Mean of a specific column

print(dataset['Column'].mean())

# Median of a specific column

print(dataset['Column'].median())

1. Correlation:

Program:

# Correlation matrix for all numerical columns

print(dataset.corr())

# Correlation between two specific columns

print(dataset['Column1'].corr(dataset['Column2']))

1. Data Distribution:

Program:

# Histogram for a specific column

import matplotlib.pyplot as plt

dataset['Column'].hist()

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.title('Histogram')

plt.show()

1. Grouping and Aggregation:

Program:

# Group by a column and calculate mean for another column

grouped\_data = dataset.groupby('Column1')['Column2'].mean()

print(grouped\_data)

1. Visualization:

Program:

# Scatter plot for two columns

plt.scatter(dataset['Column1'], dataset['Column2'])

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.title('Scatter Plot')

plt.show()

# Bar chart for a categorical column

dataset['Category'].value\_counts().plot(kind='bar')

plt.xlabel('Categories')

plt.ylabel('Counts')

plt.title('Bar Chart')

plt.show()

1. Hypothesis Testing:

Program:

from scipy.stats import ttest\_ind

result = ttest\_ind(dataset['Column1'], dataset['Column2'])

print("T-statistic:", result.statistic)

print("P-value:", result.pvalue)

Big data analytics benefits:

The benefits of using big data analytics include:

Quickly analyzing large amounts of data from different sources, in many different formats and types.

Rapidly making better-informed decisions for effective strategizing, which can benefit and improve the supply chain, operations and other areas of strategic decision-making.

Cost savings, which can result from new business process efficiencies and optimizations.

A better understanding of customer needs, behavior and sentiment, which can lead to better marketing insights, as well as provide information for product development.

Improved, better informed risk management strategies that draw from large sample sizes of data.

