EX.NO:1 DEVELOP A PREDICTIVE MODEL FOR SOCIAL MEDIA ENGAGEMENT

DATE:

AIM

To develop a predictive model for social media engagement using Apache Spark, ML lib andPySpark

ALGORITHM / PROCEDURE

1. **Environment Setup:**
   - i. **Install Apache Spark:** Ensure Spark is installed and configured on your system.
   - ii. **Install PySpark**: Install PySpark to interface with Spark through Python.
   - iii. **Data Acquisition:** Gather a dataset from a social media platform that includes user engagement metrics like likes, shares, and comments, alongside other features that may influence engagement (user demographics, post time, content type, etc.).
2. **Data Preprocessing:**
   - iv. **Data Cleaning:** Remove or impute missing values, and remove irrelevant features.
   - v. **Feature Engineering:** Create new features from existing data that might be relevant for predicting engagement, such as time of day, day of the week, or length of the post.
   - vi. **Data Normalization:** Apply normalization or standardization to scale numerical features
3. **Feature Selection:**
   - vii. **Correlation Analysis:** Perform correlation analysis to identify and eliminate highly correlated features.
   - viii. **Feature Importance:** Use Spark MLlib's feature selection tools to identify the most important features for predicting engagement.
4. **Model Selection:**
   - ix. Choose initial machine learning models to test. Common choices for regression tasks include Linear Regression, Decision Trees, Random Forest, and Gradient-Boosted Trees.
5. **Model Training:**
   - x. **Split the Data:** Divide your dataset into training and test sets (e.g., 80% training, 20% testing).
   - xi. **Train Models:** Use PySpark to train models on the training set. Leverage MLlib's machine learning algorithms for this purpose.
6. **Hyperparameter Tuning and Cross-Validation:**
   - xii. **Cross-Validation:** Implement cross-validation to assess model performance across different subsets of the data, ensuring generalizability.
   - xiii. **Hyperparameter Tuning:** Utilize MLlib's ParamGridBuilder and CrossValidator to explore various hyperparameters for each model, finding the optimal configuration.
7. **Model Evaluation:**
   - xiv. **Evaluate Models:** Assess the models' performance on the test set using appropriate metrics, such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or R-squared for regression tasks.
   - xv. **Select Best Model:** Choose the model with the best performance metrics.
8. **Deployment:**
   - xvi. Deploy the model for real-time predictions or batch processing of engagement metrics on new social media posts.

PROGRAM:

Step 1: Initialize Spark Session

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Social Media Engagement Prediction") \
    .getOrCreate()
```

Step 2: Load and Inspect Data

```
# Load data into a DataFrame
data_path = "path/to/your/dataset.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
# Show the DataFrame structure and some initial rows to understand your data
df.printSchema()
df.show(5)
```

Step 3: Data Preprocessing

```
from pyspark.ml.feature import VectorAssembler, StandardScaler
# Assuming 'features' is a list of your feature column names and 'label' is your target column
assembler = VectorAssembler(inputCols=features, outputCol="features_vector")
df = assembler.transform(df)
# Normalize features
scaler = StandardScaler(inputCol="features_vector", outputCol="scaled_features", withStd=True, withMean=False)
scalerModel = scaler.fit(df)
df = scalerModel.transform(df)
```

Step 4: Train-Test Split

```
(train_data, test_data) = df.randomSplit([0.8, 0.2])
```

Step 5: Model Training

```
from pyspark.ml.regression import LinearRegression
# Initialize and train the linear regression model
lr = LinearRegression(featuresCol="scaled_features", labelCol="label")
lrModel = lr.fit(train_data)
# You can replace LinearRegression with a different algorithm from pyspark.ml
```

Step 6: Model Evaluation

```
predictions = lrModel.transform(test_data)
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"Root Mean Squared Error (RMSE) on test data = {rmse}")
```

Output:

```
Root Mean Squared Error (RMSE) on test data = 81.14189497969642
```

RESULT:

Thus, the above program Develop a Predictive Model for Social Media Engagement was successfully completed.

EX.NO:2 INTEGRATE STATISTICAL MEASURES AND PROBABILITY ANALYSIS

DATE:

AIM

To Perform statistical analysis and apply probability theories to a dataset, covering mean, median, mode, standard deviation, variance, PDF, percentiles, moments, correlation, covariance, and Bayes' Theorem application.

ALGORITHM / PROCEDURE

1. **Setup and Data Preparation:**

   1.1 Initialize Spark Session: Start a Spark session in your environment.

   1.2 Load Data: Load your dataset into a Spark DataFrame from a CSV file or another data source.

2. **Descriptive Statistics:**

   2.1 Calculate Basic Statistics: Use Spark SQL functions to compute mean, median (approximated), mode (custom UDF may be needed), standard deviation, and variance for your data columns.

   2.2 Display Summary Statistics: Leverage the .describe() method on the DataFrame for a summary of statistics.

3. **Advanced Measures:**

   3.1 Correlation and Covariance: Utilize DataFrame's .stat.corr() and .stat.cov() for calculating correlation and covariance between pairs of columns.

   3.2 Probability Density Function (PDF): While Spark doesn't directly compute PDF, you can sample your data and use Python libraries like SciPy or NumPy for PDF calculations.

4. **Percentiles and Moments:**

   4.1 Calculate Percentiles: Use the .approxQuantile() method for approximate percentiles.

   4.2 Moments: Calculate moments (skewness and kurtosis) using .select() with .skewness() and .kurtosis() functions applied to the columns.

PROGRAM:

Step 1: Initialize Spark Session

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Statistical Measures and Probability Analysis") \
    .getOrCreate()
```

Step 2: Load Data

```
# Load your dataset
data_path = "path/to/your/dataset.csv"
df = spark.read.csv(data_path, header=True, inferSchema=True)
df.show(5)
```

Step 3: Descriptive Statistics

```
from pyspark.sql.functions import mean, stddev, col, max, min
# Assuming 'your_column' is the name of your column of interest
df.select(mean(col("your_column")).alias("Mean"),
        stddev(col("your_column")).alias("Standard Deviation"),
        max(col("your_column")).alias("Max"),
        min(col("your_column")).alias("Min")).show()
```

Step 4: Correlation and Covariance

```
from pyspark.sql.functions import corr
# Calculate correlation between two columns
df.select(corr("column1", "column2").alias("Correlation")).show()
# For covariance, use the .cov function on a StatFunctions object (not directly available as a DataFrame method in PySpark)
covariance = df.stat.cov("column1", "column2")
print(f"Covariance: {covariance}")
```

Step 5: Applying Probability Theories (e.g., Bayes' Theorem)

```
# Assuming you have a DataFrame 'df' with columns 'event' and 'condition' for P(Event|Condition)
# Calculate P(Condition|Event) using Bayes' Theorem components
# First, calculate the necessary probabilities: P(Event), P(Condition), and P(Event|Condition)
p_event = df.filter(df['event'] == 1).count() / df.count()
p_condition_given_event = df.filter((df['event'] == 1) & (df['condition'] == 1)).count() / df.filter(df['event'] == 1).count()
p_condition = df.filter(df['condition'] == 1).count() / df.count()
# Apply Bayes' Theorem: P(Condition|Event) = (P(Event|Condition) * P(Condition)) / P(Event)
p_condition_given_event = (p_condition_given_event * p_condition) / p_event
print(f"P(Condition|Event): {p_condition_given_event}")
```

Output:

```
 Sno          Name  Quiz1  Quiz2  AVG1  Assig1  Assig2  AVG2  Total 75
0   1   Ashir Mehfooz   14.0   14.0  14.0    13.0    13.0  13.0     68.00
1    2      Atif Raftad    4.0  10.0   7.0     4.0     5.0  4.5     41.50
2  3       Saiqa Aziz  15.0   11.0  13.0    14.0    13.0  13.5     60.50
3  8      Ozair Minhas    6.0    5.0  5.5     4.0     6.0  5.0     22.50
4  9   Naveera Subhani    5.0   11.0   8.0     4.0     5.0   4.5     46.50

   Quiz1       Quiz2        AVG1       Assig1      Assig2       AVG2
 count  214.000000  211.000000  216.000000  211.000000  213.000000  216.000000
 mean     8.163551    8.142180    8.020833    7.853081    8.014085    7.787037
 std      4.574514    4.687234    3.699878    4.663151    5.240876    3.814113
 min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
 25%      4.000000    4.000000    5.000000    4.000000    4.000000    4.500000
 50%      7.000000    7.000000    7.500000    6.000000    6.000000    7.500000
 75%     13.000000   13.000000   11.000000  13.000000   13.000000   10.000000
 max     15.000000   15.000000   15.000000  15.000000   44.000000   23.000000,
 0.2506696476361742)

+------------------+--------------------------+----------+----------+
|        Mean Quiz1|Standard Deviation Quiz1|Max Quiz1|Min Quiz1|
+------------------+--------------------------+----------+----------+
|8.208955223880597|      4.575600441798329|    15.0|      0.0|
+-----------------+-----------------------+---------+---------+

+-----------------+-----------------------+---------+---------+
|       Mean Quiz2|Standard Deviation Quiz2|Max Quiz2|Min Quiz2|
+-----------------+-----------------------+---------+---------+
|8.109452736318408|      4.657033411841166|    15.0|      0.0|
+-----------------+-----------------------+---------+---------+

+-----------------+----------------------+--------+--------+
|        Mean Avg1|Standard Deviation Avg1|Max Avg1|Min Avg1|
+-----------------+----------------------+--------+--------+
|8.159203980099502|    3.6471944509696876|   15.0|     0.0|
+-----------------+----------------------+--------+--------+

+------------------+-----------------------+----------+----------+
|       Mean Assig1|Standard Deviation Assig1|Max Assig1|Min Assig1|
+------------------+-----------------------+----------+----------+
|7.8606965174129355|       4.687269729004061|      15.0|       0.0|
+------------------+-----------------------+----------+----------+
+-----------------+-----------------------+----------+----------+
|      Mean Assig2|Standard Deviation Assig2|Max Assig2|Min Assig2|
+-----------------+-----------------------+----------+----------+
|8.059701492537313|        5.31473592104516|      44.0|       0.0|
+-----------------+-----------------------+----------+----------+

+-----------------+----------------------+--------+--------+
|        Mean Avg2|Standard Deviation Avg2|Max Avg2|Min Avg2|
+-----------------+----------------------+--------+--------+
|7.960199004975125|    3.8004483893613137|   23.0|     0.0|
+-----------------+----------------------+--------+--------+

+----------------------+
|Correlation Quiz1 Quiz2|
+----------------------+
|   0.24834968491901308|
+----------------------+

Covariance Quiz1 Quiz2: 5.292014925373135
```

RESULT:

Thus, the above program Integrate Statistical Measures and Probability Analysis was successfully completed.

EX.NO:3 IMPLEMENT AND COMPARE DIMENSIONAL REDUCTION METHODS
DATE:

AIM

To Implement and Compare Dimensionality Reduction Methods: focus on the necessity of dimensionality reduction,executing PCA and LDA to reduce dataset dimensions, enhance class separability, and conduct a comparitive analysis and multivariate analysis post-reduction.

ALGORITHM / PROCEDURE

1. **Input:**
   - Dataset X with n samples and m features.
   - Corresponding target labels y.
2. **Preprocessing:**
   - Standardize the features of X using a standard scaler.
   - Optionally, split the dataset into training and testing sets for post-reduction analysis.
3. **Apply PCA:**
   - Initialize a PCA object with the desired number of components k.
   - Fit the PCA model to the standardized data X.
   - Transform the data to k-dimensional space using the fitted PCA model.
4. **Apply LDA:**
   - Initialize an LDA object with the desired number of components k.
   - Fit the LDA model to the standardized data X along with corresponding labels y.
   - Transform the data to k-dimensional space using the fitted LDA model.

5 **Visualize the Results:**
   - Plot the transformed data for both PCA and LDA to observe the distribution and class separability.

6 **Post-reduction Analysis:**
   - Optionally, perform multivariate analysis techniques such as classification or clustering on the reduced datasets.
   - Split the transformed data into training and testing sets.
   - Train classifiers or clustering algorithms on the reduced data.
   - Evaluate the performance of the trained models using appropriate metrics.

7 **Comparison**
   - Compare the performance of PCA and LDA in terms of dimensionality reduction and post-reduction analysis.
   - Optionally, visualize the performance metrics or results obtained from post-reduction analysis.

PROGRAM:

1. Load the Dateset

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris
```

# Load the Iris dataset

```
iris = load_iris()
X = iris.data
y = iris.target
```

2. Preprocess the Data

```
from sklearn.preprocessing import StandardScaler
# Standardize the features
scaler =  StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3. Apply PCA and LDA for Dimensionality Reduction

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X_scaled, y)
```

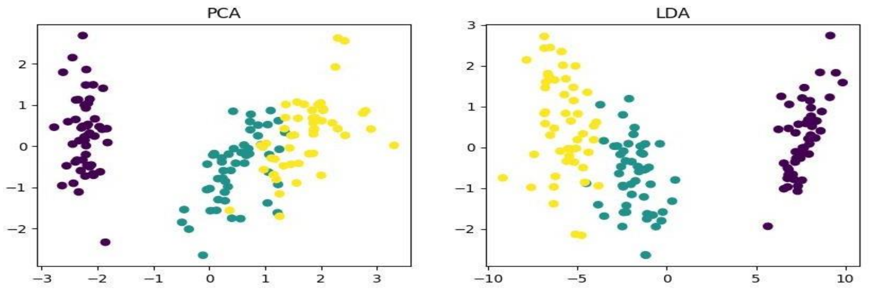
4. Analyze the Results and Compare the Performance

```
import matplotlib.pyplot as plt
# Plot PCA
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.title('PCA')
# Plot LDA
plt.subplot(1, 2, 2)
plt.scatter(X_lda[:, 0], X_lda[:, 1], c=y, cmap='viridis')
plt.title('LDA')
plt.show()
```

5. Perform Multivariate Analysis Post-Reduction

```
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.linear_model import LogisticRegression

# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Train logistic regression on PCA-reduced data

model_pca = LogisticRegression()

model_pca.fit(X_train, y_train)

y_pred_pca = model_pca.predict(X_test)

accuracy_pca = accuracy_score(y_test, y_pred_pca)

print("Accuracy on PCA-reduced data:", accuracy_pca)

# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)

# Train logistic regression on LDA-reduced data

model_lda = LogisticRegression()

model_lda.fit(X_train, y_train)

y_pred_lda = model_lda.predict(X_test)

accuracy_lda = accuracy_score(y_test, y_pred_lda)

print("Accuracy on LDA-reduced data:", accuracy_lda)
```

OUTPUT:



```
Accuracy on PCA-reduced data: 0.9

Accuracy on LDA-reduced data: 1.0
```

RESULT:

Thus, the above program Implement and Compare Dimensional Reduction Metho**ds** was successfully completed

EX.NO:4 IMPLEMENT REGRESSION TECHNIQUES AND DATA PER-PROCESSING

DATE:

AIM

To incorporates data preprocessing (cleaning, normalization, outlier detection) and applies linear, polynomial, and multivariate regression models. We'll use Python with libraries such as Pandas for data manipulation, Scikit-learn for preprocessing and regression models, and statsmodels for a detailed statistical analysis where applicable.

ALGORITHM / PROCEDURE

Step 1: Setup and Data Preparation

Step 2: Data Cleaning -Handling Missing Values,Removing Duplicates

Step 3: Normalization use standardization, which scales features to have a mean of 0 and a standard deviation of 1.

Step 4:Outlier Detection Methods

1. Interquartile Range (IQR) Method
2. Isolation Forest

PROGRAM:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Enhanced mock dataset with missing values and duplicates
data = {
    'SquareFeet': [650, 850, np.nan, 1100, 950, 1170, 980, np.nan, 700, 850, 1000, 960, 850],
    'Bedrooms': [2, 3, 2, 3, np.nan, 4, 3, 4, 2, 3, 3, 2, 3],
    'Age': [5, 7, 3, 10, 2, 12, 8, 4, 5, 6, 9, 11, 7],
    'Price': [300000, 350000, 320000, 500000, 330000, 600000, 370000, 620000, 310000,
340000, 400000, 360000, 350000]
}
df = pd.DataFrame(data)
# Fill missing values with the median
for column in ['SquareFeet', 'Bedrooms', 'Age']:
    df[column].fillna(df[column].median(), inplace=True)

# Remove duplicate rows
df.drop_duplicates(inplace=True)
# Standardization of features
scaler = StandardScaler()
df[['SquareFeet', 'Bedrooms', 'Age']] = scaler.fit_transform(df[['SquareFeet', 'Bedrooms',
'Age']])

def detect_outliers_iqr(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
     outliers = df[(df[feature] < lower_bound) | (df[feature] > upper_bound)]
    return outliers

# Example: Detect outliers in the 'SquareFeet' feature
outliers_squarefeet = detect_outliers_iqr(df, 'SquareFeet')
print("Outliers detected using IQR in 'SquareFeet':\n", outliers_squarefeet)

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Assume 'df' is our DataFrame and it's already preprocessed

# Separating the features and target variable
X = df[['SquareFeet', 'Bedrooms', 'Age']] # Independent variables
y = df['Price']                           # Dependent variable
# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Fitting the linear regression model
linear_reg = LinearRegression()
linear_reg.fit(X_train, y_train)
# Making predictions and evaluating the model
y_pred = linear_reg.predict(X_test)
print("Linear Regression RMSE:", np.sqrt(mean_squared_error(y_test, y_pred)))

from sklearn.preprocessing import PolynomialFeatures
# Transforming the features into polynomial features
poly = PolynomialFeatures(degree=2) # You can adjust the degree based on your analysis
X_poly = poly.fit_transform(X)

X_train_poly, X_test_poly, y_train, y_test = train_test_split(X_poly, y, test_size=0.2,
random_state=42)

# Fitting the linear regression model on polynomial features
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train)
# Making predictions and evaluating the model
y_pred_poly = poly_reg.predict(X_test_poly)
print("Polynomial Regression RMSE:", np.sqrt(mean_squared_error(y_test, y_pred_poly)))
```

OUTPUT:

index	SquareFeet	Bedrooms	Age	Price	outliers
0	-1.950876556	-1.212678125	-0.595683397	300000	1
1	-0.536343705	0.242535625	0.054153036	350000	1
2	0.170922719	-1.212678125	-1.245519830	320000	1
3	1.231822357	0.242535625	1.028907686	500000	1
4	0.170922719	0.242535625	-1.570438047	330000	1
5	1.726908855	1.697749375	1.678744119	600000	-1
6	0.383102647	0.242535625	0.379071252	370000	1
7	0.170922719	1.697749375	-0.920601613	620000	-1
8	-1.597243343	-1.212678125	-0.595683397	310000	1
9	-0.536343705	0.242535625	-0.270765180	340000	1
10	0.524555932	0.242535625	0.703989469	400000	1
11	0.241649361	-1.212678125	1.353825902	360000	1

```
Outliers detected using Isolation Forest:
   SquareFeet  Bedrooms       Age  Price  outliers
5    1.726909  1.697749  1.678744  600000       -1
7    0.170923  1.697749 -0.920602  620000       -1


Linear Regression RMSE: 53952.69437438349

Polynomial Regression RMSE: 37736.92624527236
```

RESULT:

Thus, the above program Implement Regression Techniques and Data Pre-processingwas successfully completed

EX.NO:5 **Explore machine learning techniques with real-world data**

DATE:

AIM

To understand and apply machine learning paradigms (supervised, unsupervised, reinforcement learning) for real-world data challenges, including K-fold cross-validation and clustering techniques.

ALGORITHM / PROCEDURE

1. **Importing Libraries**: Import necessary libraries including NumPy, pandas, Matplotlib, Seaborn, and scikit-learn modules.
2. **Loading Dataset**: Load the Iris dataset using load_iris() function from scikit-learn. Split the dataset into features (X) and target labels (y).
3. **Data Splitting**: Split the data into training and testing sets using train_test_split () function from scikit-learn. Here, 80% of the data is used for training and 20% for testing.
4. **Supervised Learning - Linear Regression**: Train a linear regression model (linear_reg_model) using the training data (X_train, y_train).
5. **Supervised Learning - Logistic Regression**: Train a logistic regression model (logreg_model) using the training data (X_train, y_train).
6. **K-fold Cross-Validation**: Perform K-fold cross-validation with a decision tree classifier (tree_model) using cross_val_score () function. Here, K is set to 5.
7. **Unsupervised Learning - K-Means Clustering**: Standardize the features using StandardScaler () from scikit-learn. Fit a K-means clustering model (kmeans_model) with 3 clusters to the standardized data.
8. **Evaluation: Silhouette Score**: Calculate the silhouette score for K-means clustering using silhouette_score() from scikit-learn.
9. **Plotting Results**: Plot the results using Matplotlib. Two subplots are created side by side:
   - The left subplot shows the actual versus predicted values for the linear regression model.
   - The right subplot shows the actual versus predicted values for the logistic regression model.

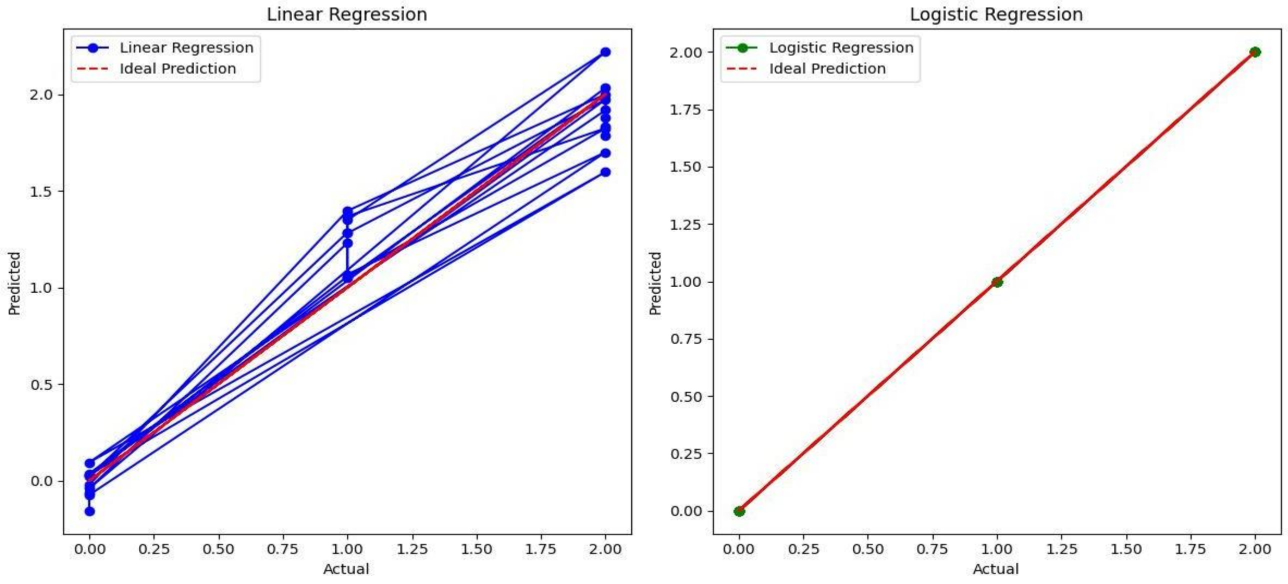
Display Plots: Display the plots using plt.show()

PROGRAM:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import  silhouette_score
from sklearn.preprocessing import StandardScaler

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
```

```
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Supervised Learning: Linear Regression
# Train the model
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)

# Supervised Learning: Logistic Regression
# Train the model
logreg_model = LogisticRegression(max_iter=1000)
logreg_model.fit(X_train, y_train)

# K-fold Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Cross-validation with decision tree
tree_model = DecisionTreeClassifier()
tree_cv_score = cross_val_score(tree_model, X, y, cv=kf)
print ("Cross-Validation Accuracy (Decision Tree):", tree_cv_score.mean())

# Unsupervised Learning: K-Means Clustering
# Standardize the data
scaler  = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Fit K-Means clustering
kmeans_model = KMeans(n_clusters=3, random_state=42)
kmeans_model.fit(X_scaled)

# Evaluation: Silhouette Score
silhouette_avg = silhouette_score(X_scaled, kmeans_model.labels_)
print ("Silhouette Score for K-Means Clustering:", silhouette_avg)

# Plot results
plt.figure(figsize=(12, 6))

# Plot for Linear Regression
plt.subplot(1, 2, 1)
plt.plot(y_test, linear_reg_model.predict(X_test), 'bo-', label='Linear Regression')
plt.plot(y_test, y_test, 'r--', label='Ideal Prediction')
plt.title("Linear Regression")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.legend()

# Plot for Logistic Regression
plt.subplot(1, 2, 2)
plt.plot(y_test, logreg_model.predict(X_test), 'go-', label='Logistic Regression')
plt.plot(y_test, y_test, 'r--', label='Ideal Prediction')
plt.title("Logistic  Regression")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.legend()

plt.tight_layout()
plt.show()
```

OUTPUT:



RESULT:

The results for the above program User exploring machine learning techniques with real-world data was written, verified and executed successfully.

EX.NO:6 INSTALLATION OF HADOOP ENVIRONMENT

DATE:

AIM

The aim of the program is installation of Hadoop framework and study of Hadoop ecosystem

ALGORITHM / PROCEDURE

Installing Hadoop Environment

The installation of a Hadoop environment involves several steps and components. Here's a high-level overview suitable for a single-node setup, which is a good starting point for learning and development purposes. This guide assumes you're using a Unix-like environment (e.g., Linux, macOS).

Prerequisites

Java: Hadoop is written in Java, so you need to have Java installed on your machine.
SSH: For communication between Hadoop's nodes, even in a single-node setup.

Steps

1.Install Java

- ✓ Update your package index: **sudo apt-get** upDATE (Debian/Ubuntu) or **sudo yum update** (Fedora/CentOS).
- ✓ Install Java: s**udo apt-get install default-jdk (**Debian/Ubuntu) or **sudo yum install java-1.8.0-openjdk** (Fedora/CentOS).
- ✓ Verify the installation: **java -version**.

2.Configure SSH

- ✓ Install SSH (if it's not already installed): **sudo apt-get install openssh-server openssh-client** (Debian/Ubuntu) or **sudo yum install openssh-server openssh-clients (Fedora/CentOS).**
- ✓ Generate SSH keys, if you haven't already: **ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa.**
- ✓ Authorize SSH keys: **cat ~/.ssh/id_rsa.pub** >> **~/.ssh/authorized_keys and chmod 0600 ~/.ssh/authorized_keys.**

3.Install Hadoop

- ✓ Download the latest Hadoop release from the official website.
- ✓ Extract the archive: t**ar zxvf hadoop-x.y.z.tar.gz.**
- ✓ Move the Hadoop folder to a convenient location: **sudo mv hadoop-x.y.z /usr/local/hadoop.**

4.Configure Hadoop

- ✓ Set Java path in **hadoop-env.sh:** Edit **/usr/local/hadoop/etc/hadoop/hadoop-env.sh**, find the line **export JAVA_HOME=**, and set it to your Java installation path.
- ✓ Configure **core-site.xml, hdfs-site.xml, mapred-site.xml, and yarn-site.xml** according to your needs. Basic configurations can be found in Hadoop's documentation or tutorials online.

5. **Format the NameNode**: **hadoop namenode -format**.

6. **Start Hadoop**: Start all Hadoop daemons with **/usr/local/hadoop/sbin/start-all.sh.**

7. **Verify Installation:** Access the Hadoop web interface by visiting **http://localhost:50070/** from your browser.

Study of Hadoop Ecosystem

The Hadoop ecosystem includes a variety of tools that extend Hadoop's capabilities in processing, managing, and analyzing big data:

- **HDFS (Hadoop Distributed File System):** The primary storage system used by Hadoop applications.
- **MapReduce:** A programming model for processing large datasets with a parallel, distributed algorithm on a cluster.
- **YARN (Yet Another Resource Negotiator):** Manages resources of the systems storing the data and running the analysis.
- **Hive:** A data warehouse infrastructure that provides data summarization and ad hoc querying.
- **Pig:** A high-level platform for creating MapReduce programs used with Hadoop.
- **HBase:** A non-relational (NoSQL) database that runs on top of HDFS.
- **Sqoop:** A tool designed for efficiently transferring bulk data between Hadoop and structured datastores such as relational databases.
- **Flume:** A service for efficiently collecting, aggregating, and moving large amounts of log data to HDFS.
- **Oozie:** A workflow scheduler system to manage Hadoop jobs.
- **Zookeeper:** A centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

EX.NO:7 IMPLEMENT WORD COUNT USING MAPREDUCE

DATE:

AIM

To implement a simple MapReduce algorithm for word count in Python using concurrent execution with ThreadPoolExecutor.

ALGORITHM / PROCEDURE

1) The mapper function takes a text string as input, splits it into words, and creates a dictionary where the keys are words and the values are their count.
2) The reducer function takes an item, which is a tuple containing a word and a list of its counts, and returns a tuple with the word and the sum of its counts.
3) The map_reduce function orchestrates the MapReduce process. It first maps each input data using the mapper function in parallel, then reduces the mapped data using the reducer function.
4) The if___name___== "_main_": block serves as the entry point of the script. It defines the sample input data and executes the MapReduce process on that data, printing the result.

PROGRAM:

```
from collections import defaultdict
from concurrent.futures import ThreadPoolExecutor
def mapper(text):
    words = text.split()
    word_count = defaultdict(int)
    for word in words:
        word_count[word] += 1
    return word_count.items()
def reducer(item):
    word, counts = item
    return word, sum(counts)
def map_reduce(data):
    # Map phase
    with ThreadPoolExecutor() as executor:
        mapped_data = list(executor.map(mapper, data))
     # Reduce phase
    reduced_data = defaultdict(list)
    for sublist in mapped_data:
        for item in sublist:
            word, count = item
            reduced_data[word].append(count)
       with ThreadPoolExecutor() as executor:
        reduced_data = list(executor.map(reducer, reduced_data.items()))
        return reduced_data
if___name___== "__main__":
    # Sample input data
    data = [        "hello world",        "world hello",        "hello hello world"    ]
    result = map_reduce(data)
    print(result)
```

OUTPUT:

```
[('hello', 4), ('world', 3)]
```

RESULT:

Thus, the above program implement word count using mapreduce was successfully completed

EX.NO:8 HADOOP MAPREDUCE/PYSPARK

DATE:

AIM

To Implementing algorithms like matrix multiplication in MapReduce.

ALGORITHM / PROCEDURE

1. Mapper Input:

The input to the mapper will be the row index, column index, and value from both matrices A and B.

Each input record could be in the format (matrixID, rowIndex, columnIndex, value), where matrixID is 'A' or 'B' indicating the matrix to which the value belongs.

2. Map Phase:

For each element from matrix A, emit a key-value pair for each column index in the result matrix C. The key is a tuple (rowIndex of A, columnIndex of C), and the value is a tuple (matrixID, columnIndex of A, value).

For each element from matrix B, emit a key-value pair for each row index in the result matrix C. The key is a tuple (rowIndex of C, columnIndex of B), and the value is a tuple (matrixID, rowIndex of B, value).

3. Reduce Phase:

The reducer receives all values for each key, which includes all necessary elements to compute a cell in matrix C.

For each key (i, j), compute the sum of the product of corresponding elements from A and B that share the same inner dimension index. This involves multiplying values from A and B where the columnIndex of A matches the rowIndex of B, and summing these products to get the final value for C[i][j].

4. Output:

The reducer outputs the final computed value for each cell in matrix C, along with its row and column indices.

PROGRAM:

```
from pyspark import SparkContext
sc = SparkContext()
# Example matrices A and B
# Represented as ((row, col), value)
matrix_a = [((0, 0), 1), ((0, 1), 2),
            ((1, 0), 3), ((1, 1), 4)]
matrix_b = [((0, 0), 5), ((0, 1), 6),
            ((1, 0), 7), ((1, 1), 8)]
# Parallelize matrices
rdd_a = sc.parallelize(matrix_a)
rdd_b = sc.parallelize(matrix_b)
# Map step
mapped_a = rdd_a.flatMap(lambda x: [((x[0][0], i), ('A', x[0][1], x[1])) for i in range(2)])
mapped_b = rdd_b.flatMap(lambda x: [((i, x[0][1]), ('B', x[0][0], x[1])) for i in range(2)])
```

```
# Reduce step
result_rdd = (mapped_a.union(mapped_b)
            .groupByKey()
            .map(lambda x: (x[0], sum([val[1]*val[2] for val in list(x[1]) if val[0]=='A']) *
sum([val[1]*val[2] for val in list(x[1]) if val[0]=='B']))))
result = result_rdd.collect()
print(result)
```

OUTPUT:

```
[((1, 1), 32), ((0, 1), 16), ((0, 0), 14), ((1, 0), 28)]
```

RESULT:

Thus, the above program Implementing algorithms like matrix multiplication in Map Reduce was successfully completed

EX.NO:9 INSTALL AND CONFIGURE NOSQL DATABASES

DATE:

AIM

To Installing and configuring NoSQL databases like MongoDB, Cassandra, HBase, or Hypertable involves different steps based on the specific database system

PROGRAM:

MongoDB

Installation:

On Ubuntu:

sudo apt-get upDATE

sudo apt-get install -y mongodb

Configuration:

MongoDB runs with default settings suitable for development. For production, you might need to edit the configuration file, typically located at /etc/mongod.conf.

Basic Commands:

- **Start MongoDB:** sudo systemctl start mongodb
- **Verify MongoDB is running:** sudo systemctl status mongodb
- **Connect to MongoDB shell:** mongo
- **Create or switch to a database:** use mydatabase
- **Insert a document:** db.mycollection.insertOne({name: "John Doe", age: 30})
- **Find documents:** db.mycollection.find()

Cassandra

Installation:

On Ubuntu:

echo "deb http://www.apache.org/dist/cassandra/debian 311x main" | sudo tee -a /etc/apt/sources.list.d/cassandra.sources.list

curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-key add -

sudo apt-get update

sudo apt-get install cassandra

Configuration:

Configuration files are located in /etc/cassandra. The main configuration file is cassandra.yaml.

Basic Commands:

- **Start Cassandra:** sudo service cassandra start
- **Verify it's running:** nodetool status
- **Connect to Cassandra shell:** cqlsh
- **Create a keyspace:** CREATE KEYSPACE mykeyspace WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};
- **Use keyspace:** USE mykeyspace;

- **Create a table:** CREATE TABLE mytable (id int PRIMARY KEY, name text, age int);
- **Insert data:** INSERT INTO mytable (id, name, age) VALUES (1, 'John Doe', 30);
- **Query data:** SELECT * FROM mytable;

HBase

Installation:

HBase requires Hadoop as a prerequisite. Once Hadoop is set up:

- Download HBase from HBase's website and extract it.
- Configure HBase in standalone mode by editing conf/hbase-site.xml.

Configuration:

For a basic setup, edit hbase-site.xml to define the file system and directory where HBase will store data.

Basic Commands:

- **Start HBase:** *start-hbase.sh*
- **Access HBase shell:** *hbase shell*
- **List tables:** *list*
- **Create a table:** *create 'mytable', 'mycf'*
- **Put data:** *put 'mytable', 'row1', 'mycf:name', 'John Doe'*
- **Get data:** *get 'mytable', 'row1'*
- **Scan table:** *scan 'mytable'*

MongoDB

MongoDB is a document-oriented database that stores data in JSON-like formats. Here are some basic operations:

1. Connect to MongoDB Shell: Run mongo in your terminal.
2. Create or Switch Database:

   use myDatabase
3. Insert Documents:

   db.myCollection.insertOne({name: "John Doe", age: 30})
4. Find Documents:

   db.myCollection.find({name: "John Doe"})
5. Update Documents:

   db.myCollection.updateOne({name: "John Doe"}, {$set: {age: 31}})
6. Delete Documents:

   db.myCollection.deleteOne({name: "John Doe"})

Cassandra

Cassandra is a wide-column store designed to handle large amounts of data across many commodity servers.

1. Connect to Cassandra: Run cqlsh in your terminal.
2. Create Keyspace (similar to a database in relational databases):

   CREATE KEYSPACE myKeyspace WITH replication = {'class': 'SimpleStrategy', 'replication_factor' : 3};

3. Use Keyspace:
   USE myKeyspace;
4. Create Table:
   CREATE TABLE myTable (id UUID PRIMARY KEY, name TEXT, age INT);
5. Insert Data:
   INSERT INTO myTable (id, name, age) VALUES (uuid(), 'John Doe', 30);
6. Query Data:
   SELECT * FROM myTable WHERE name = 'John Doe';
7. Up Date Data:
   UPDATE myTable SET age = 31 WHERE id = <some-uuid>;
8. Delete Data:

   DELETE FROM myTable WHERE id = <some-uuid>;

HBase

HBase is a distributed column-oriented database built on top of the Hadoop file system.

1. Start HBase Shell: Run hbase shell in your terminal.
2. List Tables:
   list
3. Create Table:
   create 'myTable', 'myFamily'
4. Insert Data:
   put 'myTable', 'row1', 'myFamily:name', 'John Doe'
5. Retrieve Data:
   get 'myTable', 'row1'
6. Scan Table:
   scan 'myTable'
7. Delete Data:
   delete 'myTable', 'row1', 'myFamily:name'
8. Drop Table:
   disable 'myTable'
   drop 'myTable'

RESULT:

Thus, the above program Install and Configure NoSQL Databaseswas successfully completed

EX.NO:10 QUERIES TO SORT AND AGGREGATE THE DATA IN A TABLE USING HIVEQL

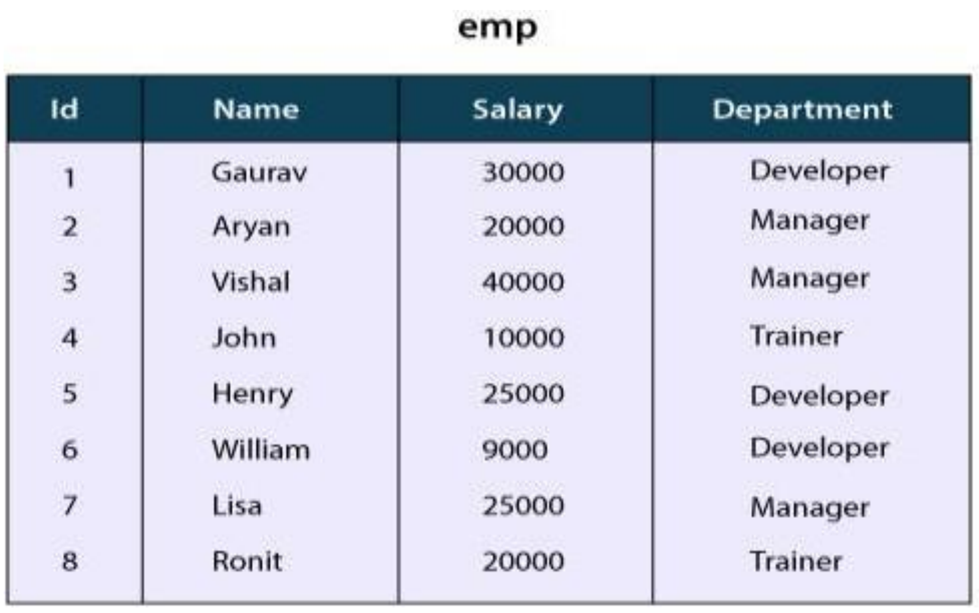
DATE:

AIM

To Write queries to sort and aggregate the data in a table using HiveQL

DESCRIPTION:

Hive is an open-source data warehousing solution built on top of Hadoop. It supports an SQL-like query language called HiveQL. These queries are compiled into MapReduce jobs that are executed on Hadoop. While Hive uses Hadoop for execution of queries, it reduces the effort that goes into writing and maintaining MapReduce jobs. Hive supports database concepts like tables, columns, rows and partitions. Both primitive (integer, float, string) and complex data-types(map, list, struct) are supported. Moreover, these types can be composed to support structures of arbitrary complexity. The tables are serialized/deserialized using default serializers/deserializer. Any new data format and type can be supported by implementing SerDe and ObjectInspector java interface. HiveQL - ORDER BY and SORT BY Clause By using HiveQL ORDER BY and SORT BY clause, we can apply sort on the column. It returns the result set either in ascending or descending order. Here, we are going to execute these clauses on the records of the below table:


emp

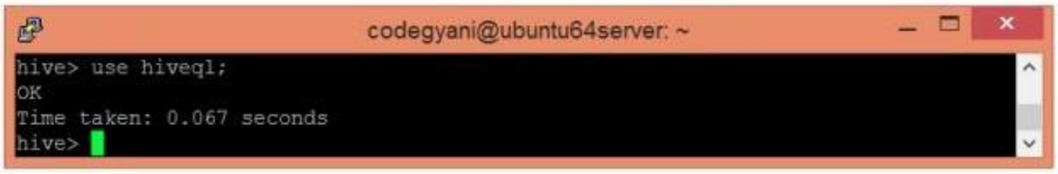
Id	Name	Salary	Department
1	Gaurav	30000	Developer
2	Aryan	20000	Manager
3	Vishal	40000	Manager
4	John	10000	Trainer
5	Henry	25000	Developer
6	William	9000	Developer
7	Lisa	25000	Manager
8	Ronit	20000	Trainer

HiveQL - ORDER BY Clause

In HiveQL, ORDER BY clause performs a complete ordering of the query result set. Hence, the complete data is passed through a single reducer. This may take much time in the execution of large datasets. However, we can use LIMIT to minimize the sorting time.

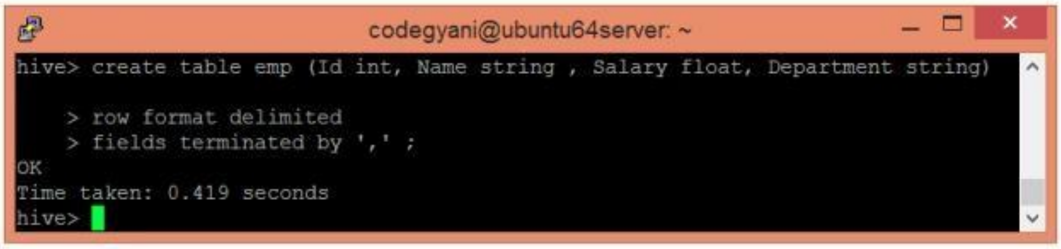
Example: Select the database in which we want to create a table.

hive> use hiveql;



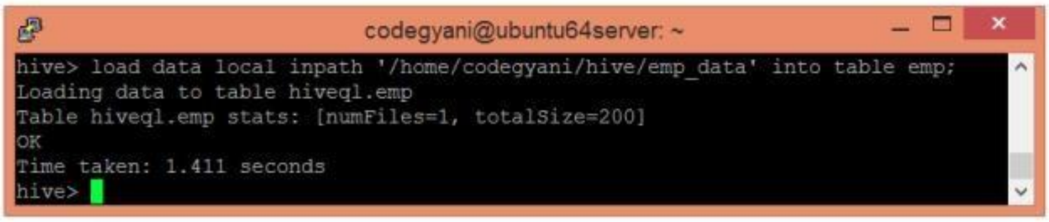
```
hive> use hiveql;
OK
Time taken: 0.067 seconds
hive>
```

Now, create a table by using the following command:
hive> create table emp (Id int, Name string , Salary float, Department string) row format delimited
fields terminated by ',' ;



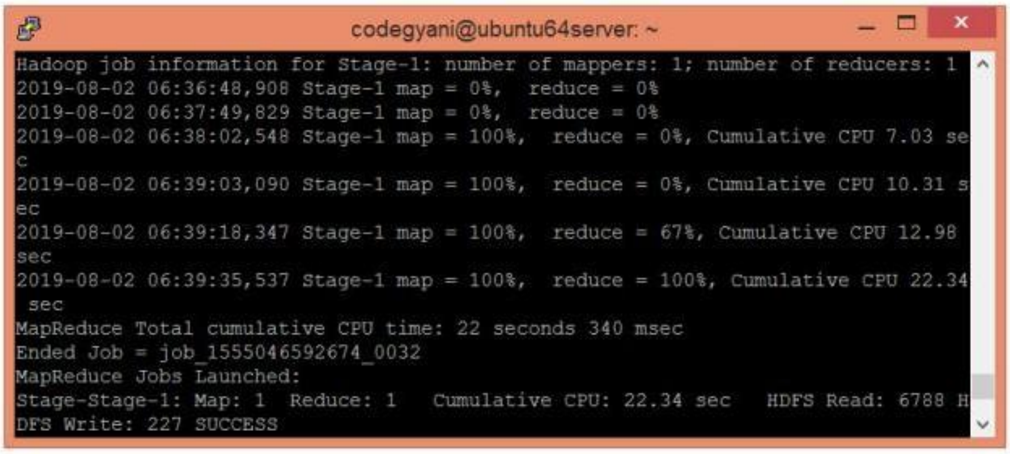
```
hive> create table emp (Id int, Name string , Salary float, Department string)

    > row format delimited
    > fields terminated by ',' ;
OK
Time taken: 0.419 seconds
hive>
```

Load the data into the table
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;

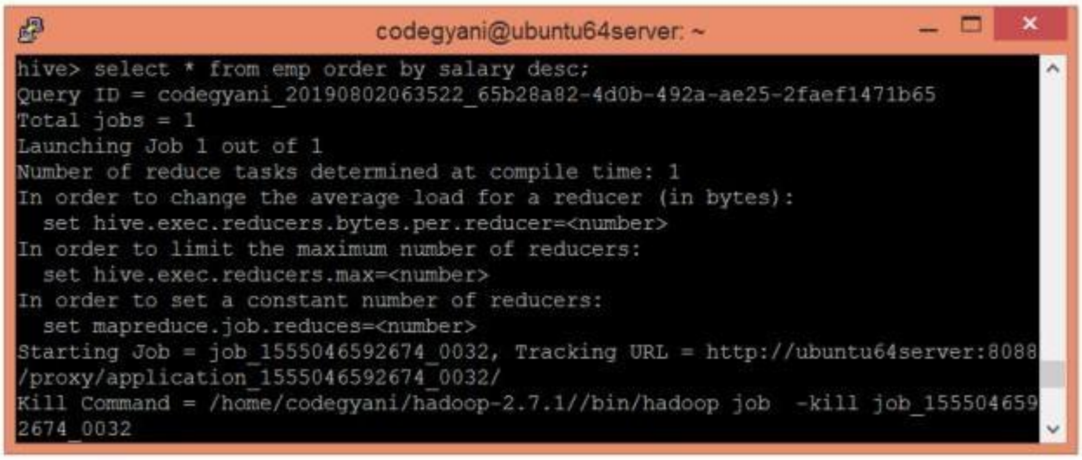


```
hive> load data local inpath '/home/codegyani/hive/emp_data' into table emp;
Loading data to table hiveql.emp
Table hiveql.emp stats: [numFiles=1, totalSize=200]
OK
Time taken: 1.411 seconds
hive>
```

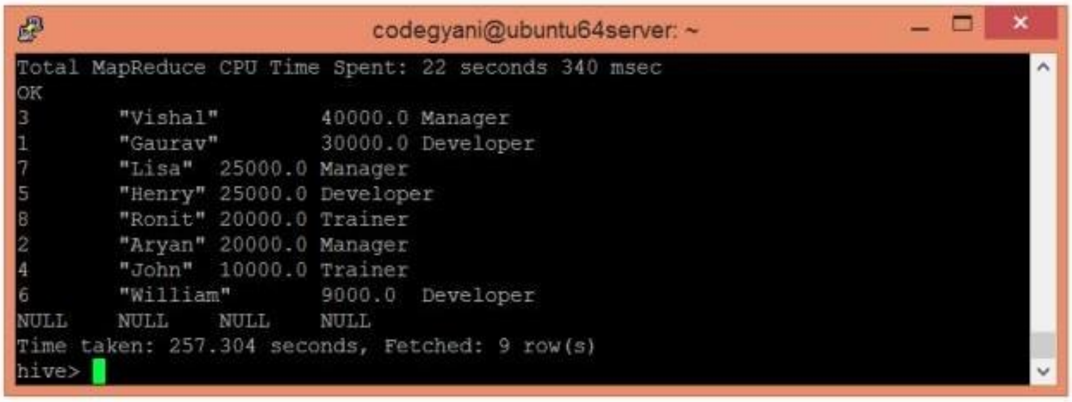
Now, fetch the data in the descending order by using the following command
*hive> select * from emp order by salary desc;*



```
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2019-08-02 06:36:48,908 Stage-1 map = 0%,  reduce = 0%
2019-08-02 06:37:49,829 Stage-1 map = 0%,  reduce = 0%
2019-08-02 06:38:02,548 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 7.03 se
c
2019-08-02 06:39:03,090 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 10.31 s
ec
2019-08-02 06:39:18,347 Stage-1 map = 100%,  reduce = 67%, Cumulative CPU 12.98
sec
2019-08-02 06:39:35,537 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 22.34
 sec
MapReduce Total cumulative CPU time: 22 seconds 340 msec
Ended Job = job_1555046592674_0032
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 22.34 sec   HDFS Read: 6788 H
DFS Write: 227 SUCCESS
```



```
hive> select * from emp order by salary desc;
Query ID = codegyani_20190802063522_65b28a82-4d0b-492a-ae25-2faef1471b65
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1555046592674_0032, Tracking URL = http://ubuntu64server:8088
/proxy/application_1555046592674_0032/
Kill Command = /home/codegyani/hadoop-2.7.1//bin/hadoop job  -kill job_155504659
2674_0032
```



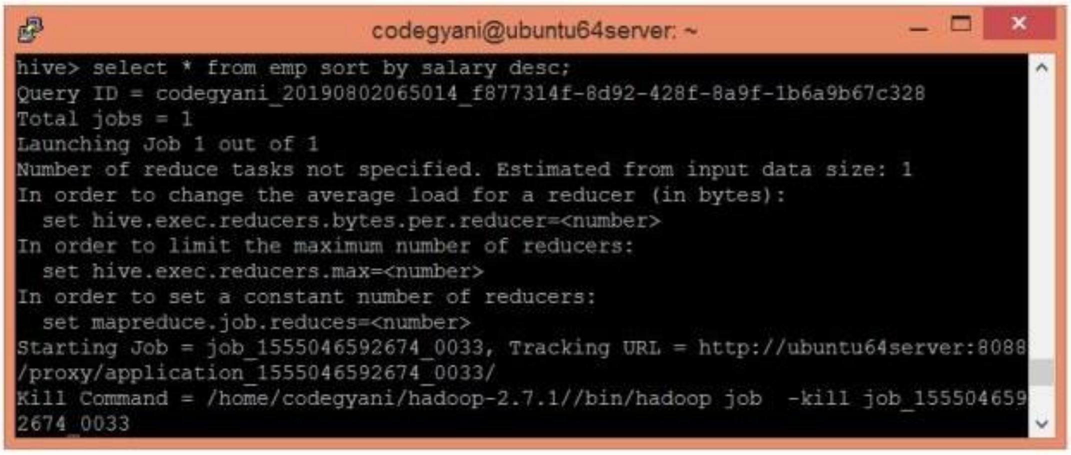
```
Total MapReduce CPU Time Spent: 22 seconds 340 msec
OK
3       "Vishal"        40000.0 Manager
1       "Gaurav"        30000.0 Developer
7       "Lisa"  25000.0 Manager
5       "Henry" 25000.0 Developer
8       "Ronit" 20000.0 Trainer
2       "Aryan" 20000.0 Manager
4       "John"  10000.0 Trainer
6       "William"       9000.0  Developer
NULL    NULL    NULL    NULL
Time taken: 257.304 seconds, Fetched: 9 row(s)
hive>
```

HiveQL - SORT BY Clause

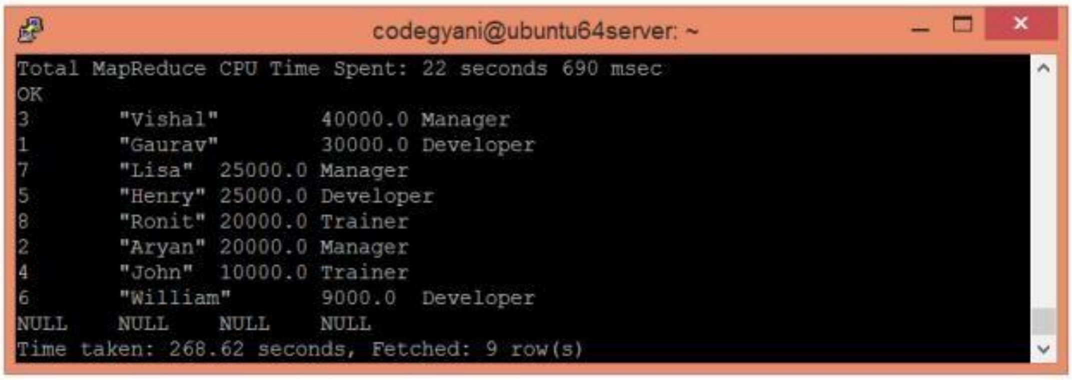
The HiveQL SORT BY clause is an alternative of ORDER BY clause. It orders the data within each reducer. Hence, it performs the local ordering, where each reducer's output is sorted separately. It may also give a partially ordered result. Example:

Let's fetch the data in the descending order by using the following command

*hive> select * from emp sort by salary desc;*



```
hive> select * from emp sort by salary desc;
Query ID = codegyani_20190802065014_f877314f-8d92-428f-8a9f-1b6a9b67c328
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1555046592674_0033, Tracking URL = http://ubuntu64server:8088
/proxy/application_1555046592674_0033/
Kill Command = /home/codegyani/hadoop-2.7.1//bin/hadoop job  -kill job_155504659
2674_0033
```



```
Total MapReduce CPU Time Spent: 22 seconds 690 msec
OK
3       "Vishal"        40000.0 Manager
1       "Gaurav"        30000.0 Developer
7       "Lisa"  25000.0 Manager
5       "Henry" 25000.0 Developer
8       "Ronit" 20000.0 Trainer
2       "Aryan" 20000.0 Manager
4       "John"  10000.0 Trainer
6       "William"       9000.0  Developer
NULL    NULL    NULL    NULL
Time taken: 268.62 seconds, Fetched: 9 row(s)
```

Cluster By:

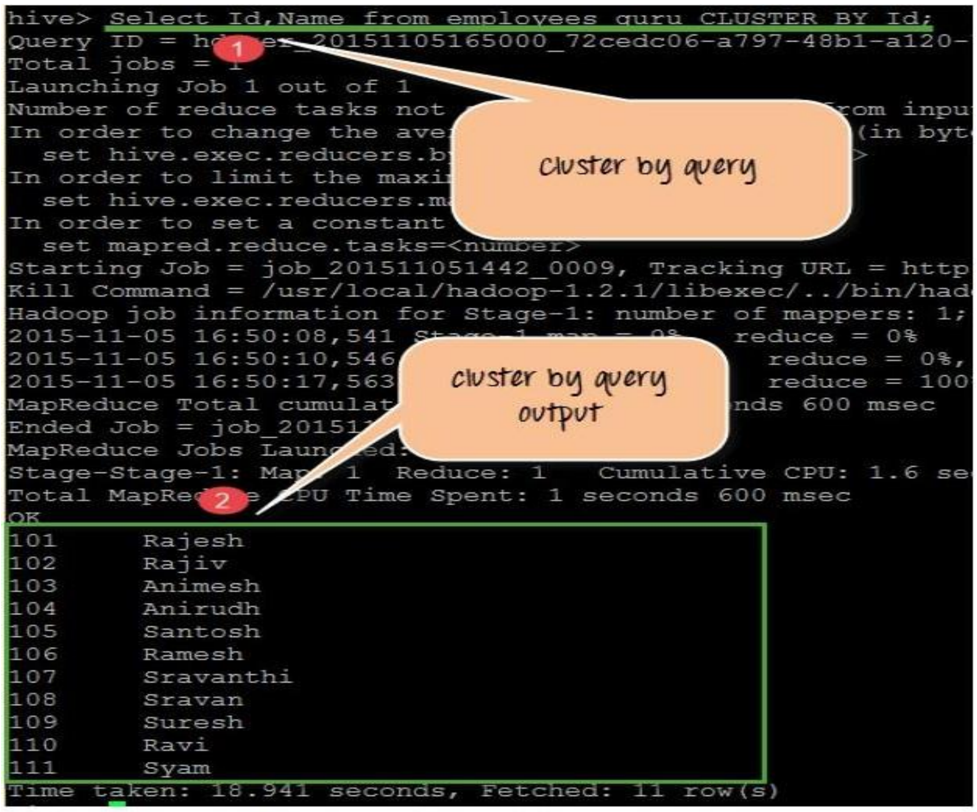
Cluster By used as an alternative for both Distribute BY and Sort BY clauses in Hive-QL. Cluster BY clause used on tables present in Hive. Hive uses the columns in Cluster by to distribute the rows among reducers. Cluster BY columns will go to the multiple reducers.

- It ensures sorting orders of values present in multiple reducers

For example, Cluster By clause mentioned on the Id column name of the table employees_guru table. The output when executing this query will give results to multiple reducers at the back end. But as front end it is an alternative clause for both Sort By and Distribute By.

Example:

SELECT Id, Name from employees_guru CLUSTER BY Id;



```
hive> Select Id,Name from employees_guru CLUSTER BY Id;
Query ID = hd   er_20151105165000_72cedc06-a797-48b1-a120-
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not                                  from inpu
In order to change the ave                                  (in byt
  set hive.exec.reducers.b                                  >
In order to limit the maxi
  set hive.exec.reducers.m
In order to set a constant
  set mapred.reduce.tasks=<number>
Starting Job = job_201511051442_0009, Tracking URL = http
Kill Command = /usr/local/hadoop-1.2.1/libexec/../bin/had
Hadoop job information for Stage-1: number of mappers: 1;
2015-11-05 16:50:08,541 Stage-1 map = 0%,  reduce = 0%
2015-11-05 16:50:10,546                    reduce = 0%,
2015-11-05 16:50:17,563                    reduce = 100
MapReduce Total cumulat                 nds 600 msec
Ended Job = job_20151
MapReduce Jobs Laun   ed:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.6 se
Total MapReduce CPU Time Spent: 1 seconds 600 msec
OK
101     Rajesh
102     Rajiv
103     Animesh
104     Anirudh
105     Santosh
106     Ramesh
107     Sravanthi
108     Sravan
109     Suresh
110     Ravi
111     Syam
Time taken: 18.941 seconds, Fetched: 11 row(s)
```

RESULT:

Thus, the above program sort and aggregate the data in a table using HiveQL was successfully completed

EX.NO:11 CRAFT A CODEBASE FOR SOCIAL MEDIA DATA VISUALIZATION

DATE:

AIM

The aim of the algorithm is to retrieve tweets from a specific Twitter user's timeline using the Tweepy library in Python. By providing the username of the Twitter account as input, the algorithm authenticates with the Twitter API, retrieves the specified number of tweets from the user's timeline, and then prints out each tweet. This allows users to access and analyze tweets from a particular user programmatically, which can be useful for various purposes such as sentiment analysis, trend analysis, or simply for monitoring specific accounts.

ALGORITHM / PROCEDURE

```
1. Import the tweepy library

2. Define function get_tweets(username):
   a. Initialize consumer_key, consumer_secret, access_key, and access_secret with your Twitter API credentials
   b. Authenticate with Twitter API using tweepy.OAuthHandler and set access token with set_access_token method
   c. Create API object using tweepy.API(auth)
   d. Set number_of_tweets to the desired number of tweets to be extracted
   e. Use api.user_timeline method to fetch tweets for the given username, providing screen_name=username and count=number_of_tweets as parameters
   f. Initialize an empty list tweet_list to store tweets
   g. Loop through each tweet in tweets fetched:
      i. Append the text of each tweet to tweet_list
   h. Print each tweet in tweet_list

3. Define main function:
   a. Call get_tweets function with the desired Twitter handle as argument

4. Execute main function if this script is run as the main program
```

PROGRAM:

```
from mpl toolkits. mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt # plotting
import numpy as np # linear algebra
import os # accessing directory structure
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Distribution graphs (histogram/bar graph) of column data
def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
    nunique = df.nunique()
    df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying
```

```
purposes, pick columns that have between 1 and 50 unique values
    nRow, nCol = df.shape
    columnNames = list(df)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor
= 'w', edgecolor = 'k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = df.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
        plt.ylabel('counts')
        plt.xticks(rotation = 90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
    plt.show()

    # Correlation matrix
    def plotCorrelationMatrix(df, graphWidth):
        filename = df.dataframeName
        df = df.dropna('columns') # drop columns with NaN
        df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more
than 1 unique values
        if df.shape[1] < 2:
            print(f'No correlation plots shown: The number of non-NaN or constant columns
({df.shape[1]}) is less than 2')
            return
        corr = df.corr()
        plt.figure(num=None, figsize=(graphWidth, graphWidth), dpi=80, facecolor='w',
edgecolor='k')
        corrMat = plt.matshow(corr, fignum = 1)
        plt.xticks(range(len(corr.columns)), corr.columns, rotation=90)
        plt.yticks(range(len(corr.columns)), corr.columns)
        plt.gca().xaxis.tick_bottom()
        plt.colorbar(corrMat)
        plt.title(f'Correlation Matrix for {filename}', fontsize=15)
        plt.show()

    # Scatter and density plots
    def plotScatterMatrix(df, plotSize, textSize):
```

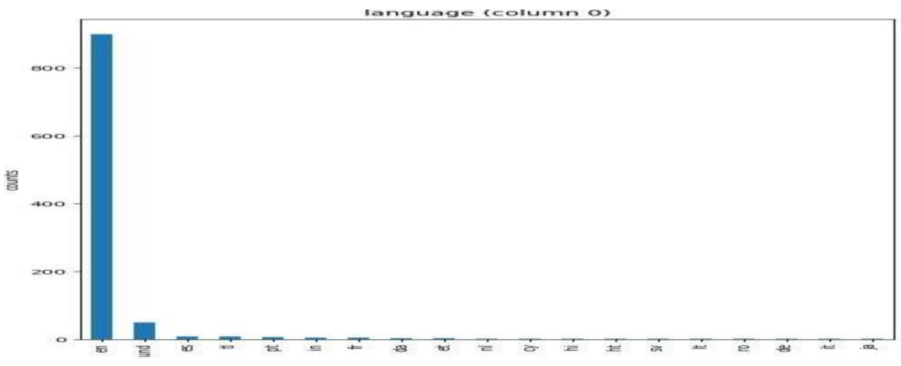
```
    df = df.select_dtypes(include =[np.number]) # keep only numerical columns
    # Remove rows and columns that would lead to df being singular
    df = df.dropna('columns')
    df = df[[col for col in df if df[col].nunique() > 1]] # keep columns where there are more
than 1 unique values
    columnNames = list(df)
    if len(columnNames) > 10: # reduce the number of columns for matrix inversion of
kernel density plots
        columnNames = columnNames[:10]
    df = df[columnNames]
    ax = pd.plotting.scatter_matrix(df, alpha=0.75, figsize=[plotSize, plotSize],
diagonal='kde')
    corrs = df.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k = 1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction',
ha='center', va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()

  nRowsRead = 1000 # specify 'None' if want to read whole file
  # tweets.csv may have more rows in reality, but we are only loading/previewing the first
1000 rows
  df1 = pd.read_csv('extracted_tweets.csv', delimiter=',', nrows = nRowsRead)
  df1.dataframeName = 'tweets.csv'
  nRow, nCol = df1.shape
  print(f'There are {nRow} rows and {nCol} columns')
```

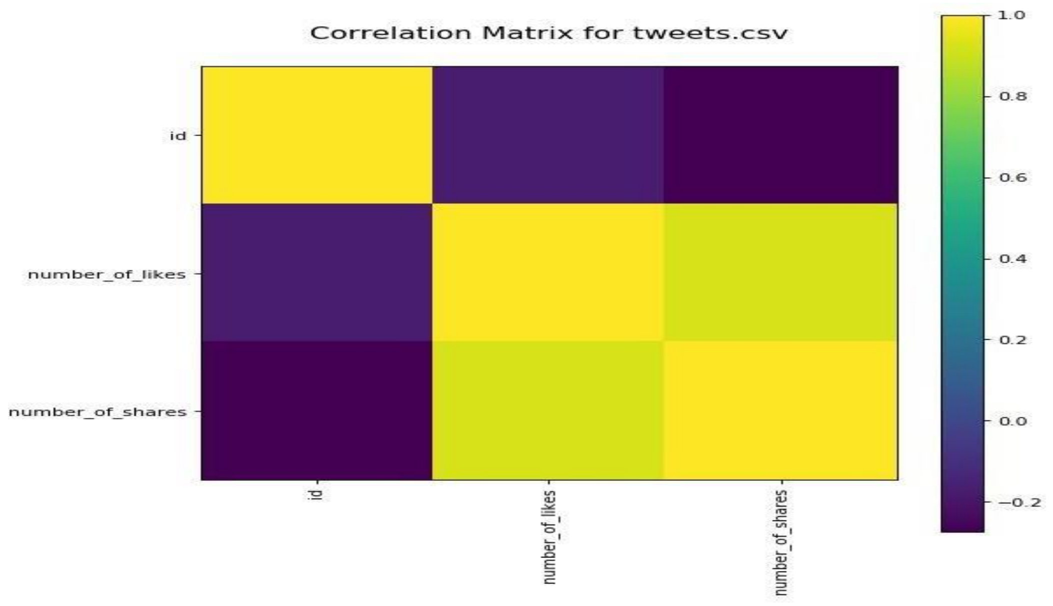
Tweepy :

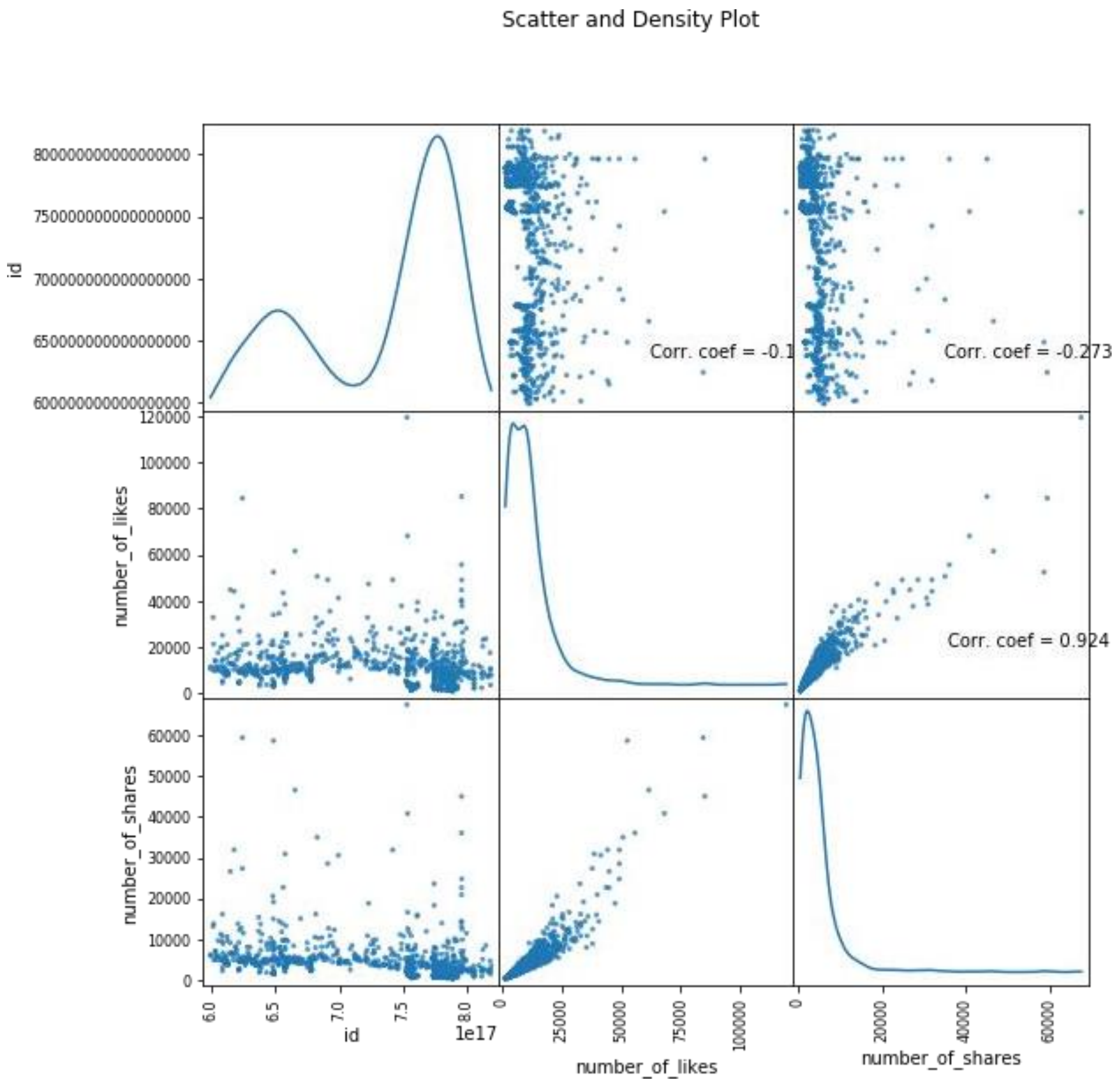
```
import tweepy

# Fill the X's with the credentials obtained by
# following the above mentioned procedure.
consumer_key = "your-generated-key"
consumer_secret = "your-generated-secret"
access_key = "your-developer-key"
access_secret = "your-developer-secret"

# Function to extract tweets
def get_tweets(username):

            # Authorization to consumer key and consumer secret
            auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
```

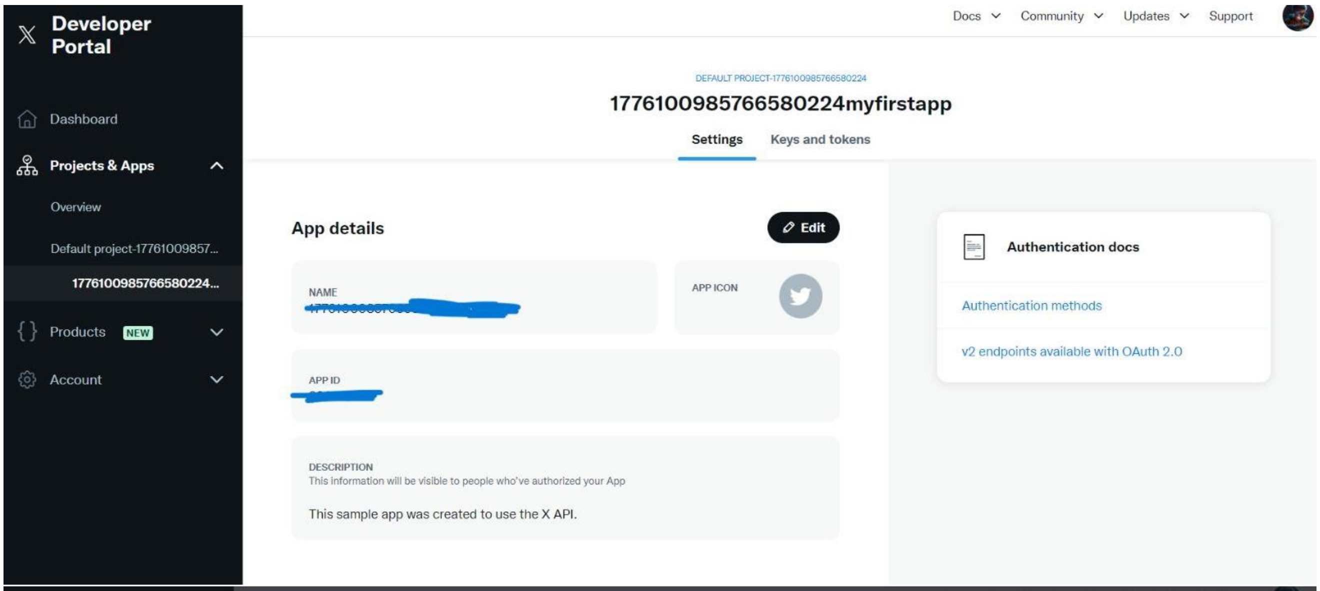
```
                # Access to user's access key and access secret
                auth.set_access_token(access_key, access_secret)

                # Calling api
                api = tweepy.API(auth)

                # 200 tweets to be extracted
                number_of_tweets=200
                tweets = api.user_timeline(screen_name=username)

                # Empty Array
                 tmp=[]

                # create array of tweet information: username,
                # tweet id, date/time, text
                tweets_for_csv = [tweet.text for tweet in tweets] # CSV file created
                for j in tweets_for_csv:

                    # Appending tweets to the empty array tmp
                    tmp.append(j)

                # Printing the tweets
                print(tmp)
# Driver code
if _name_ == '_main_':

    # Here goes the twitter handle for the user
    # whose tweets are to be extracted.
    get_tweets("your-twitter-handle-name")
```

OUTPUT:




Correlation Matrix for tweets.csv
id
number_of_likes
number_of_shares
id
number_of_likes
number_of_shares
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Scatter and Density Plot
Corr. coef = -0.1
Corr. coef = -0.273
Corr. coef = 0.924
id
number_of_likes
number_of_shares
1e17



DEFAULT PROJECT-1776100985766580224

1776100985766580224myfirstapp

Settings Keys and tokens

App details

NAME
[redacted]

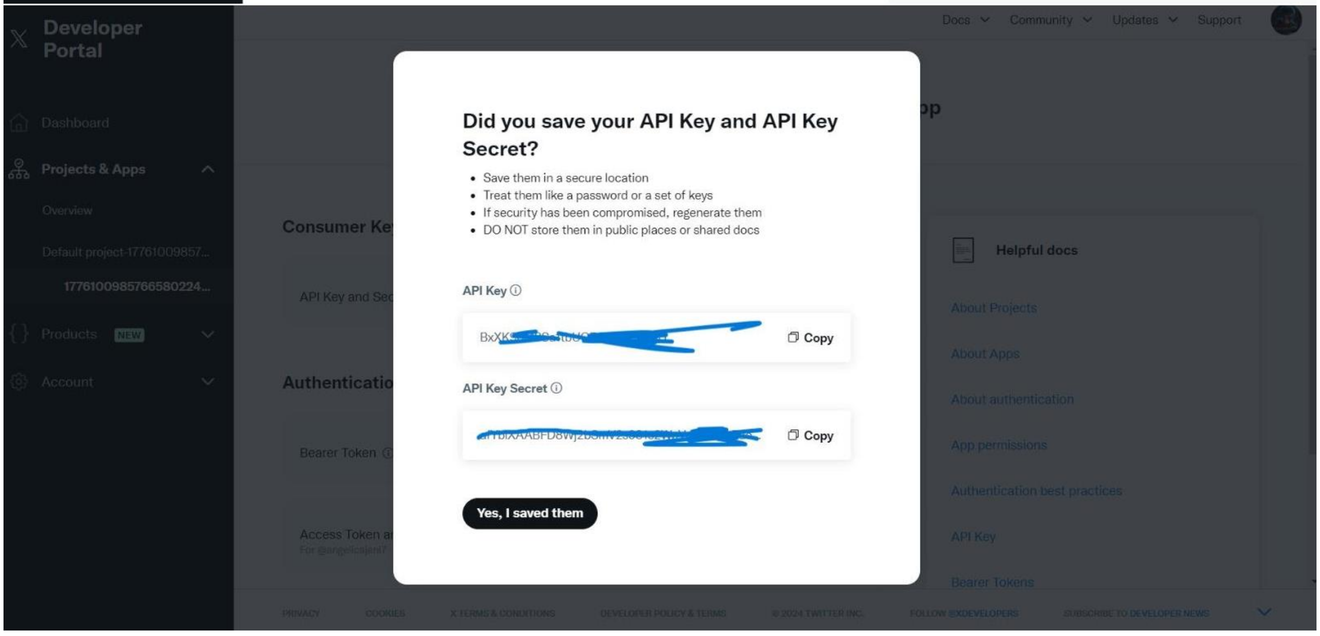
APP ICON

APP ID
[redacted]

DESCRIPTION
This information will be visible to people who've authorized your App

This sample app was created to use the X API.

Authentication docs
Authentication methods
v2 endpoints available with OAuth 2.0



Did you save your API Key and API Key Secret?

- Save them in a secure location
- Treat them like a password or a set of keys
- If security has been compromised, regenerate them
- DO NOT store them in public places or shared docs

API Key
[redacted]

API Key Secret
[redacted]

Yes, I saved them

RESULT:

Thus, the above program craft a codebase for social media data visualization was successfullycompleted.