**Aim -** Implement and explore performance evaluation metrics for Data Models (Supervised/Unsupervised Learning)

**Theory -**
Evaluation metrics are quantitative measures that assess the performance of a model or system. They are used to compare models, select models, and improve model development.

1. **Supervised Learning**
   Supervised learning involves training a model using labeled data, where the goal is to predict an output based on input features. It is broadly classified into:

   a. **Regression**
      Predicts continuous values (e.g., price, temperature).

      - **$R^2$ Score (Coefficient of Determination)**
        Measures how well the model explains the variability of the dependent variable.

        $$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

        $SS_{res}$ = Sum of squared residuals (errors)

        $SS_{tot}$ = Total sum of squares (variation in data)

        **1 (100%)** → Perfect model
        **0** → Model is no better than a simple mean-based prediction
        **Negative** → Model is worse than a random guess

      - **Root Mean Squared Error (RMSE)**
        Measures the average error in predictions.

        $$RMSE = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$$

        $y_i$ = Actual values

        $\hat{y}_i$ = Predicted values

        $n$ = Number of observations

        **Lower RMSE =** Better predictions.

   b. **Classification**
      Predicts discrete labels (e.g., spam or not spam, disease detection).

      - **Accuracy**
        Measures the percentage of correctly classified instances.

        $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

        TP (True Positives): Correctly predicted positive cases.
        TN (True Negatives): Correctly predicted negative cases.
        FP (False Positives): Incorrectly predicted positive cases.

FN (False Negatives): Incorrectly predicted negative cases.

- **Precision**
  Measures the proportion of correctly predicted positive cases out of all predicted positives.

  $$Precision = \frac{TP}{TP + FP}$$

  High precision = Few false positives.

- **Recall (Sensitivity or True Positive Rate)**
  Measures the proportion of actual positive cases correctly predicted.

  $$Recall = \frac{TP}{TP + FN}$$

  High recall = Few false negatives.

- **F1 Score**
  Harmonic mean of Precision and Recall, balancing both.

  $$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

  High F1 Score = Balanced precision & recall.
  Best for imbalanced datasets.

- **Classification Report**
  Summarizes Precision, Recall, F1-score, and Support for each class.
  Support → Number of true instances per class.

2. **Unsupervised Learning**
   Unsupervised learning involves clustering, dimensionality reduction, or anomaly detection, where there are no labeled outputs.
   a. **K - Means Clustering**
   - **Silhouette Score**
     Measures how well-separated the clusters are.

     $$S(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

     $a(i)$ = Mean distance between point **i** and other points in **same cluster**.

     $b(i)$ = Mean distance between point **i** and nearest cluster.

     **1** → Perfect clustering
     **0** → Overlapping clusters
     **Negative** → Incorrect clustering

- **Inertia (Sum of Squared Distances - SSD)**
  Measures how tightly the clusters are formed.

$$Inertia = \sum_{i=1}^{k} \sum_{x \in C_i} (x - \mu_i)^2$$

$k$ = Number of clusters

$C_i$ = Cluster center

$\mu_i$ = Mean of points in cluster

  Lower inertia = Compact clusters = Better model.
  Should decrease as clusters increase but not too much, to avoid overfitting.

**Implementation -**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score,
precision_score, recall_score, f1_score, classification_report
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import silhouette_score
# Load dataset
file_path = "/content/Cleaned_Accumulative_distribution.csv"
data = pd.read_csv(file_path)
# Display dataset info
print("Dataset Preview:")
print(data.head())
```

```
Dataset Preview:
         ID          Type  Movie_number  Fly_number  Other_number  \
0 -1.731908  Dmelanogaster     -1.540178   -1.474087     -1.647509
1 -1.731623  Dmelanogaster     -1.540178   -1.300665     -1.647509
2 -1.731339  Dmelanogaster     -1.540178   -1.127243     -1.647509
3 -1.731054  Dmelanogaster     -1.540178   -0.953821     -1.647509
4 -1.730769  Dmelanogaster     -1.540178   -0.780399     -1.647509

   Difference_x  Difference_y  Distance
0     -0.291126      0.136903 -0.968219
1     -0.714968     -0.445300 -0.413533
2     -1.780003      1.731326  1.374617
```

```
3     -1.871965     -1.020045  0.942211
4     -0.289079      0.247088 -0.897781
# Encode categorical column (if "Type" exists)
if "Type" in data.columns:
    encoder = LabelEncoder()
    data["Type_encoded"] = encoder.fit_transform(data["Type"])
    data_numeric = data.drop(columns=["Type"])  # Drop original 'Type'
else:
    data_numeric = data.copy()
```

**SUPERVISED LEARNING**

1. **REGRESSION (PREDICTING DISTANCE)**

```
X_reg = data_numeric.drop(columns=["Distance"])
y_reg = data_numeric["Distance"]
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg,
y_reg, test_size=0.25, random_state=42)
reg_model = LinearRegression()
reg_model.fit(X_train_reg, y_train_reg)
y_pred_reg = reg_model.predict(X_test_reg)
print("\n🔹 Regression Model Performance:")
print("R² Score:", r2_score(y_test_reg, y_pred_reg))
print("RMSE:", np.sqrt(mean_squared_error(y_test_reg, y_pred_reg)))
🔹 Regression Model Performance:
R² Score: 0.19509373088751325
RMSE: 0.8960527863178356
```

2. **CLASSIFICATION (BINARY CLASSIFICATION USING TYPE)**

```
if "Type_encoded" in data.columns:
    X_clf = data_numeric.drop(columns=["Type_encoded"])
    y_clf = data_numeric["Type_encoded"]

    X_train_clf, X_test_clf, y_train_clf, y_test_clf =
train_test_split(X_clf, y_clf, test_size=0.25, random_state=42)
    clf_model = LogisticRegression()
    clf_model.fit(X_train_clf, y_train_clf)
    y_pred_clf = clf_model.predict(X_test_clf)

    print("\n🔹 Classification Model Performance:")
    print("Accuracy:", accuracy_score(y_test_clf, y_pred_clf))
    print("Precision:", precision_score(y_test_clf, y_pred_clf,
average="weighted"))
```

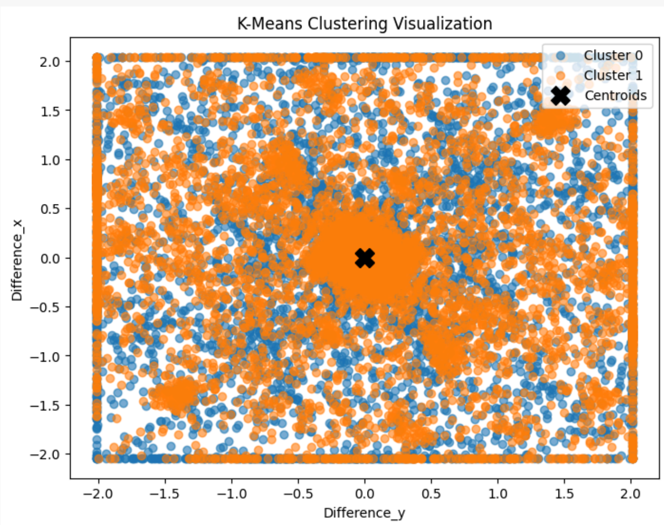
```
    print("Recall:", recall_score(y_test_clf, y_pred_clf,
average="weighted"))
    print("F1 Score:", f1_score(y_test_clf, y_pred_clf,
average="weighted"))
    print("\nClassification Report:\n", classification_report(y_test_clf,
y_pred_clf))
🔹 Classification Model Performance:
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 Score: 1.0

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       895
           1       1.00      1.00      1.00       989
           2       1.00      1.00      1.00      1156

    accuracy                           1.00      3040
   macro avg       1.00      1.00      1.00      3040
weighted avg       1.00      1.00      1.00      3040
```

**UNSUPERVISED LEARNING**

1. **K-MEANS CLUSTERING**

```
import matplotlib.pyplot as plt

# Select two features for clustering visualization
x_feature = "Difference_y"  # Change this if needed
y_feature = "Difference_x"  # Change this if needed

# Perform K-Means clustering
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
clusters = kmeans.fit_predict(data_numeric)

# Add cluster labels to the dataset
data_numeric["Cluster"] = clusters

# Scatter plot of clusters
plt.figure(figsize=(8, 6))
```

```
for cluster in range(2):
    plt.scatter(
        data_numeric[data_numeric["Cluster"] == cluster][x_feature],
        data_numeric[data_numeric["Cluster"] == cluster][y_feature],
        label=f"Cluster {cluster}",
        alpha=0.6
    )

# Plot cluster centroids
plt.scatter(
    kmeans.cluster_centers_[:, data_numeric.columns.get_loc(x_feature)],
    kmeans.cluster_centers_[:, data_numeric.columns.get_loc(y_feature)],
    s=200, c="black", marker="X", label="Centroids"
)

plt.xlabel(x_feature)
plt.ylabel(y_feature)
plt.title("K-Means Clustering Visualization")
plt.legend()
plt.show()

# Print clustering metrics
sil_score = silhouette_score(data_numeric.drop(columns=["Cluster"]),
clusters)
print("\n 🔹 Clustering Metrics:")
print("Silhouette Score:", sil_score)
print("Inertia:", kmeans.inertia_)
```



```
🔹 Clustering Metrics:
Silhouette Score: 0.19087302057057917
Inertia: 77229.78013831074
```

**Inference Drawn -**

1. **Regression Model Performance**
    a. **$R^2$ Score: 0.195**
        - This means that only 19.5% of the variation in the dependent variable is explained by the regression model.
        - Since it's low, the model isn't capturing the patterns well.
    b. **RMSE (Root Mean Squared Error): 0.896**
        - RMSE measures the average error in the predictions.
        - Since it's close to 1, it suggests that predictions deviate by about 0.9 units on average.
        - A lower RMSE means better predictions.
    c. **Overall**
        - The regression model needs improvement, possibly with feature selection, non-linearity handling, or hyperparameter tuning.
2. **Classification Model Performance**
    a. **Accuracy: 1.0 (100%)**
    b. **Precision, Recall, F1 Score: 1.0 (100%) for all classes**
    c. **Classification Report shows perfect scores for all categories**
    d. **Overall**
        - The classification model is perfect on this dataset.
        - All predictions are 100% correct across all categories.
        - This could indicate overfitting (especially if this is on training data).
3. **Clustering Metrics**
    a. **Silhouette Score: 0.190**
        - Measures how well-separated the clusters are.

- Since 0.190 is low, it suggests overlapping clusters → K-Means might not be the best algorithm here.

b. **Inertia: 77,229.78**

- Measures how tight clusters are (lower = better).
- High inertia means clusters are spread out, indicating that K-Means might not be grouping data well.

**Conclusion -** This experiment helps evaluate different machine learning models based on their performance metrics. Regression models are judged by how well they fit the data ($R^2$, RMSE), while classification models are evaluated based on their accuracy, precision, recall, and F1-score. Unsupervised clustering models are measured using the Silhouette Score and Inertia, which indicate the quality of the clusters formed.