AIDS-I Assignment No: 2

Q.1: Use the following data set for question 1

82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans:

Sort the Data

Sorted Data:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81,
82, 82, 84, 85, 88, 90, 90, 91, 95, 99

1. Mean

Mean is calculated using the formula:

Mean = Sum of all values ÷ Number of values
Mean = 1621 ÷ 20 = **81.05**

2. Median

Since there are 20 values (even number),
the median is the average of the 10th and 11th values.

10th value = 81
11th value = 82

Median = (81 + 82) ÷ 2 = **81.5**

3. Mode

- 76 appears 3 times
- 82 and 90 appear 2 times
- All other values appear only once

Mode = 76

4. Interquartile Range (IQR)

Q1 (Lower Quartile): Median of the first 10 values
First 10 values: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Q1 = (76 + 76) ÷ 2 = **76**

Q3 (Upper Quartile): Median of the last 10 values
Last 10 values: 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Q3 = (88 + 90) ÷ 2 = **89**

Interquartile Range = Q3 - Q1 = 89 - 76 = **13**

Q.2 1) Machine Learning for Kids 2) Teachable Machine

- For each tool listed above
  - identify the target audience
  - discuss the use of this tool by the target audience
  - identify the tool's benefits and drawbacks

Ans:

1. Machine Learning for Kids

Target Audience:

- School-aged children (typically ages 8–16)
- Teachers and educators introducing AI and ML concepts to students
- Beginners with little or no coding experience

Use by Target Audience:

- Kids train ML models by labeling examples (text, images, numbers, or sounds).
- These models are then used in **Scratch**, **Python**, or **App Inventor** projects.
- Teachers use it in classrooms to **teach basic AI/ML concepts through interactive, hands-on learning**.

Benefits:

- **Kid-friendly interface** that simplifies complex ML tasks.
- **Integrates with Scratch and Python**, allowing creative applications.
- Encourages **experiential learning** with real ML models.

- **Free to use** with cloud-based options.

Drawbacks:

- **Limited to educational use**; not suitable for advanced ML tasks.
- **Not robust for complex or large datasets**.
- **Requires internet access** and sometimes a bit of setup (especially for Python/App Inventor).

2. Teachable Machine

Target Audience:

- Students, hobbyists, educators, and beginner-level developers
- Creatives and designers exploring ML without coding

Use by Target Audience:

- Users create ML models by training them with **images, audio, or poses**.
- The trained models can be **exported** for use in websites, apps, or other creative projects.
- Used in classrooms, workshops, or personal experimentation to demonstrate ML in action.

Benefits:

- **No coding required** — fully visual, drag-and-drop interface.
- Quick, **real-time model training and feedback**.
- Supports **exporting models to TensorFlow.js, TensorFlow Lite**, and more.
- Great for **creative applications**, demos, and education.

Drawbacks:

- **Limited model types** (mostly classification).
- Not suitable for complex or large-scale ML applications.
- **Lacks deeper customization** or fine-tuning options.

- From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?

    - Predictive analytic
    - Descriptive analytic

  Ans:Both **Machine Learning for Kids** and **Teachable Machine** are best described as **predictive analytic** tools. This is because they are designed to train machine learning models that can make predictions or classifications based on input data. In Machine Learning for Kids, users—primarily students—label training data such as images, text, or sounds, and the tool creates models that can predict the correct label for new, unseen inputs. Similarly, Teachable Machine allows users to train models using images, audio, or poses, and these models can then predict or classify future inputs in real time. These tools are not focused on analyzing or summarizing past data, which is characteristic of descriptive analytics. Instead, their main function is to use labeled data to learn patterns and apply those patterns to make predictions, which is the core purpose of predictive analytics.

- From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?

    - Supervised learning
    - Unsupervised learning
    - Reinforcement learning

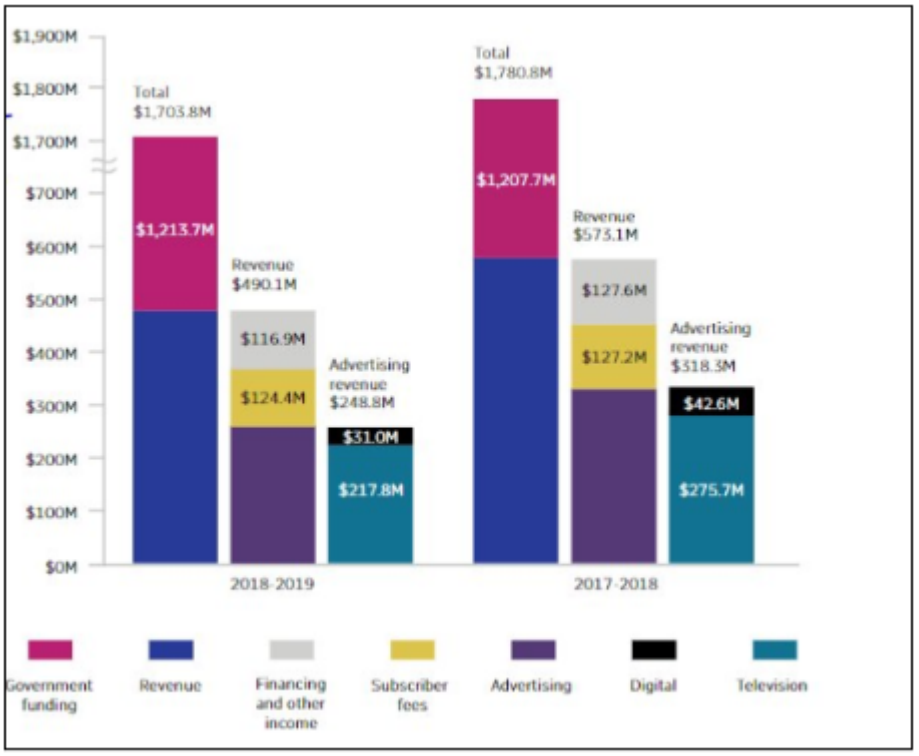
  Ans:Both **Machine Learning for Kids** and **Teachable Machine** use **supervised learning**. This means users give the tool examples that are already labeled—for example, pictures labeled as "cat" or "dog," or sounds labeled as "clap" or "whistle." The tool learns from these examples and then tries to guess the correct label for new data. In both tools, the user teaches the model by showing it what each example means. They don't find patterns on their own like in unsupervised learning, and they don't learn by trying things and getting rewards like in reinforcement learning. That's why **supervised learning** is the best way to describe how both tools work.

  Q.3 Data Visualization: Read the following two short articles:

- Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization." *Medium*
- Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." *Quartz*
- Research a current event which highlights the results of misinformation based on data visualization. Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

  Ans:In April 2023, a bar chart from the Canadian Broadcasting Corporation's (CBC) 2018–2019 Annual Report reappeared online and faced heavy criticism for being misleading. The chart aimed to display CBC's sources of funding, such as government support and advertising income, but its design raised serious concerns about how accurately the data was represented.One major issue was the manipulation of the y-axis, which featured an abrupt jump from $700 million to $1.7 billion. This distorted scale caused the $490 million advertising revenue bar to appear larger than the $1.2 billion bar for government funding, despite the actual values showing otherwise. This visual trick created the false impression that CBC earned more from television advertising than it received from government support. Furthermore, the chart separated advertising and other revenue into different bars instead of

grouping them as parts of the total funding, adding to the confusion. Overall, the flawed visualization misrepresented the true financial picture, misleading viewers and prompting criticism for its lack of clarity and accuracy.



Q. 4 Train Classification Model and visualize the prediction performance of trained model required information

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

Pima Indians Diabetes Database

Ans. Model used: Naive Bayes

Step 1. Importing required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from imblearn.over_sampling import SMOTE
```

Step 2. Loading the dataset

```
from google.colab import files
uploaded = files.upload()
```

Step 3. Performing data preprocessing

```
df.isnull().sum()
```

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

Since here there are no null values we can skip imputation

Step 4. Splitting the dataset

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Second split: Validation (20%) and Test (10%) from Temp
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, stratify=y_temp, random_state=42)
```

In this step we are splitting the dataset into:
70% Train
20% Validation
10% Test

But since train_test_split only lets us split into two parts at a time, we do it in two stages:

Step 1:

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

Here we are separating the features (X) and target label (y).

Step 2: First Split — 70% Train, 30% Temp

```
# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
```

This gives:

- X_train, y_train → 70% of the data (for training)
- X_temp, y_temp → remaining 30% (to be further split into validation and test)

Stratify is used to ensure that the proportion of class labels (0s and 1s) is the same in all splits.

Step 3: Second Split — 20% Validation, 10% Test

```
# Second split: Validation (20%) and Test (10%) from Temp
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, stratify=y_temp, random_state=42)
```

Here we are splitting X_temp (30%) into:

- X_val, y_val → 20% of original data
- X_test, y_test → 10% of original data

Step 5. Using SMOTE for class imbalance

```
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

SMOTE (Synthetic Minority Over-sampling Technique) is used to fix class imbalance by creating synthetic samples for the minority class

In the above given line of code,

fit_resample() applies SMOTE to the training set which balances the class distribution.
X_train_res, y_train_res now contain:
Original majority class samples and Synthetic minority class samples

Step 6. Feature scaling

```
scaler = StandardScaler()
X_train_res = scaler.fit_transform(X_train_res)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Feature scaling standardizes the values of your features so they all have: Mean = 0 and Standard Deviation = 1

In the above code,

We fit the scaler on the training data, then transform the training data using those values.Finally we transform validation and test data using the same scaler.

Step 7. Training the Naive Bayes model

```
model = GaussianNB()
model.fit(X_train_res, y_train_res)
```

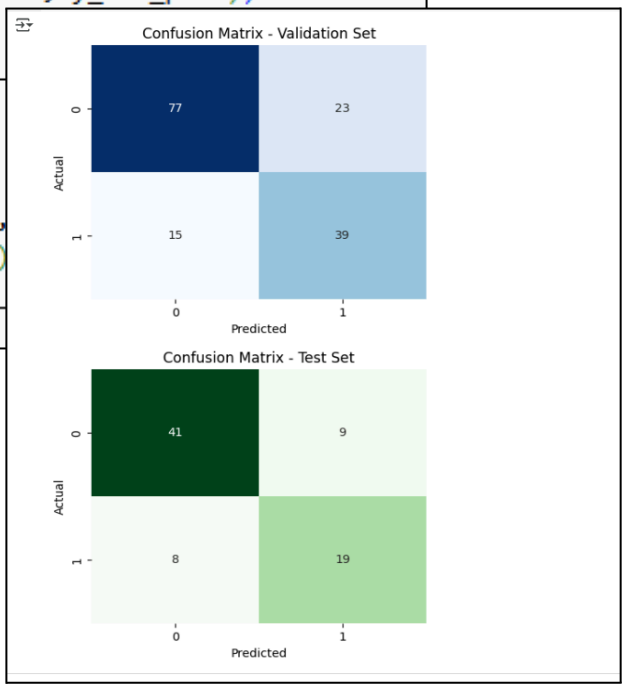
```
GaussianNB
GaussianNB()
```

This creates a Naive Bayes model using the GaussianNB class.
GaussianNB is used when the features are continuous and assumed to follow a normal (Gaussian) distribution — which is common for numeric data like BMI, glucose, etc.

Step 8. Evaluating the model

```
# On Validation Set
y_val_pred = model.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_

# On Test Set
y_test_pred = model.predict(X_test)
print("\nTest Accuracy:", accuracy_score(y_test,
print(classification_report(y_test, y_test_pred)
print("Confusion Matrix:\n", confusion_matrix(y_
```




```
Validation Accuracy: 0.7532467532467533
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       100
           1       0.63      0.72      0.67        54

    accuracy                           0.75       154
   macro avg       0.73      0.75      0.74       154
weighted avg       0.76      0.75      0.76       154

Confusion Matrix:
 [[77 23]
 [15 39]]

Test Accuracy: 0.7792207792207793
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        50
           1       0.68      0.70      0.69        27

    accuracy                           0.78        77
   macro avg       0.76      0.76      0.76        77
weighted avg       0.78      0.78      0.78        77

Confusion Matrix:
 [[41  9]
 [ 8 19]]
```

Validation Accuracy: 75.3%
Test Accuracy: 77.9%
This shows good generalization and consistent performance.

Recall for Class 1 (Diabetic):
72% (Validation), 70% (Test)

Model is effectively identifying most actual diabetic cases.

Precision for Class 1 (Diabetic):
63% (Validation), 68% (Test)
Around two-thirds of diabetic predictions are correct.

Confusion Matrix indicates:

- High true positive rate for both classes.
- Moderate number of false positives and false negatives.

Q.5 Train Regression Model and visualize the prediction performance of trained model

- Data File: Regression data.csv
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5

Use any Regression model to predict the values of all Dependent variables using values of 1st column.
Requirements to satisfy:

- Programming Language: Python
- OOP approach must be followed
- Hyper parameter tuning must be used
- Train and Test Split should be 70/30
- Train and Test split must be randomly done
- Adjusted R2 score should more than 0.99
- Use any Python library to present the accuracy measures of trained model

https://github.com/Sutanoy/Public-Regression-Datasets

https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv

- URL: https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx

(Refer any one)

Ans:

1. Importing Required Libraries

```
import pandas as pd, numpy as np
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import r2_score, mean_squared_error
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

We use:

- Pipeline to streamline polynomial features → standardization → ridge regression.
- GridSearchCV for hyperparameter tuning
- r2_score, mean_squared_error for evaluation

2. Defining the RegressionModel Class

```
class RegressionModel:
    def __init__(self, model_pipeline, param_grid):
        ...
```

- Encapsulates model training, evaluation, and hyperparameter tuning.
- Reusable and extensible for other regression problems.

3. Loading the Dataset

```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx"
df = pd.read_excel(url)
```

The dataset contains real estate records including:

- Distance to MRT station
- Number of nearby convenience stores
- Age of building
- Geographic coordinates

4. Data Preprocessing

```
df = df.drop(columns=['No'])  # Drop irrelevant index
X = df.drop(columns=['Y house price of unit area'])  # Features
y = df['Y house price of unit area']  # Target
```

We define:

- X: all columns except house price
- y: the target variable (house price)

5. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(...)
```

- 70% training data, 30% testing
- random_state=42 ensures reproducibility

6. Model Pipeline & Hyperparameter Grid

```
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])
```

- Pipeline Components:
    - poly: adds non-linearity (degree=2 or more)
    - scaler: standardizes features (important for ridge!)
    - ridge: regularized regression
- Parameter Grid:

```
param_grid = {
    'poly__degree': [2, 3, 4],
    'ridge__alpha': [0.1, 1, 10, 100]
```

}
We let GridSearchCV try different combinations of:

- Polynomial degrees: 2 to 4
- Ridge regularization strengths (alpha): 0.1 to 100

7. Training and Evaluation

best_model = reg_model.train(X_train, y_train)
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)

- The model is trained with 5-fold cross-validation
- Best estimator is used for prediction
- We calculate:
    - R^2: proportion of variance explained
    - Adjusted R^2: penalizes for extra features
    - MSE: average squared prediction error

8. Visualization

sns.scatterplot(x=y_test_actual, y=y_pred)

- Compares actual vs predicted prices
- Red dashed line shows ideal predictions (perfect match)

Final Results:

Best Params: {'poly__degree': 2, 'ridge__alpha': 1}
R^2 Score: 0.6552
Adjusted R^2 Score: 0.6376
Mean Squared Error: 57.6670

- The model captures ~66% of the variance in house prices.
- There's room for improvement, but this is reasonable for real-world data.
- Adjusted R^2 shows performance after accounting for model complexity.

Why we May Not Reach $R^2 > 0.99$

- Real-world datasets include noise, missing features, and non-linear interactions.
- Polynomial features help but too high a degree → overfitting.
- Ridge helps reduce overfitting, but can't add missing signal.

Q.6 What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during

the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

Ans:The Wine Quality dataset from Kaggle contains physicochemical properties of red and white wine samples, along with a quality score (rated from 0 to 10) assigned by wine tasters. This dataset is widely used for regression and classification problems in machine learning. Here's a breakdown of the key features, their relevance, handling of missing data, and imputation methods:

Key Features and Their Importance:

Feature	Description	Importance in Wine Quality Prediction
Fixed acidity	Non-volatile acids in wine (e.g., tartaric acid)	Affects flavor, crispness, and balance
Volatile acidity	Acetic acid content; high levels lead to vinegar taste	Negatively correlated with quality
Citric acid	Adds freshness and flavor	Mild positive impact on taste
Residual sugar	Sugar left after fermentation	Sweetness; important for white wines
Chlorides	Salt content	Too much can spoil taste
Free sulfur dioxide	Prevents microbial growth	Affects preservation and freshness
Total sulfur dioxide	Sum of all SO_2 forms	Excess can cause unpleasant taste
Density	Related to sugar and alcohol content	Helps distinguish wine types
pH	Acidity level	Affects stability and taste
Sulphates	Adds sulfur dioxide; acts as preservative	Somewhat correlated with better quality
Alcohol	Alcohol content in %	Strong positive correlation with quality
Quality	Target variable (score from 0 to 10)	-

Handling Missing Data During Feature Engineering:

The original Kaggle wine quality datasets (red and white) are generally **clean with no missing values**. However, in real-world scenarios or modified datasets, missing values might appear. During feature engineering, we can handle them using different imputation techniques depending on the nature and amount of missing data.

Imputation Techniques: Pros and Cons:

Method	Advantages	Disadvantages
Mean/Median Imputation	Simple and fast; preserves sample size	Can distort variance and distribution
Mode Imputation	Good for categorical values	Not suitable for continuous data
KNN Imputation	Considers similarity with other samples	Computationally expensive for large datasets
Regression Imputation	Predicts missing values using other features	May introduce bias if assumptions are violated
Multiple Imputation	Accounts for uncertainty; statistically rigorous	More complex and time-consuming
Dropping Rows	Easy if missing data is minimal	Risk of losing valuable data if too many rows removed

Conclusion: Understanding each feature's role is essential for building accurate wine quality prediction models. Most features directly affect taste, aroma, or preservation. When working with real-world or expanded datasets, proper imputation techniques help maintain data integrity and model performance. The choice of imputation should balance simplicity, accuracy, and the amount of missing data.