Nayaab Jindani D15C Roll No.20

# AIDS-I Assignment No: 2

05/05

**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
2. Find the Median
3. Find the Mode
4. Find the Interquartile range

**Ans.**

1. Find the mean:

To find the mean we use the following formula:

$$\text{Mean} = \frac{\sum x}{n}$$

=1611/20
=80.55

2. Find the median:

To find median, first we will sort the data in ascending order
Sorted data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Since n here is even we will find the average of the 2 numbers at middle position (10th and 11th position)

Median=(81 + 82) / 2 = 81.5

3. Find the mode:

mode is the value that appears most frequently in a dataset.
In the given dataset 76 appears 3 times,which is the most frequent and so mode is 76.

4. Find the interquartile range:

Q1 = median of first 10 = (76 + 76) / 2 = 76
Q3 = median of last 10 = (88 + 90) / 2 = 89
IQR = Q3 - Q1 = 89 - 76 = 13

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

1. 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.**

**Target audience:**

Machine learning for kids:
School students, teachers and educators introducing machine learning concepts

Teachable machine:
General users, developers, students with little/no coding background.

**Use by target audience:**

Machine learning for kids:
Students can build machine learning models (text, image, or number-based) through a drag-and-drop interface and also use pretrained models. Teachers can use it in classroom activities to teach AI/ML fundamentals in a simple, visual way.

Teachable machine:
Users can train ML models using webcam, microphone, or file uploads (e.g., image, sound, pose). It's often used in interactive projects, demos, and classroom activities.

**Tool's benefits and drawbacks:**

Machine learning for kids:
Benefits:

- Easy to use, no prior coding experience needed
- Educational and aligned with school curriculum
- Supports various data types: text, images, numbers
- Encourages creativity through Scratch integration

Drawbacks:

- Limited scalability and complexity
- Not suitable for advanced ML concepts
- Mostly focused on learning rather than production use

Teachable machine:
Benefits:

- No coding required
- Fast, intuitive, web-based interface
- Supports real-time model training with live data
- Easy export to TensorFlow.js and other formats

Drawbacks:

- Not suitable for complex datasets or large-scale projects
- Performance depends on browser and hardware

2. 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 examples of predictive analytics tools. They are designed to train models on labeled data and use those models to make predictions on new inputs. In Machine Learning for Kids, users provide examples of data with labels and the tool learns to predict similar outcomes. Similarly, Teachable Machine allows users to train models using images, sounds, or poses, and predicts the class of new inputs in real time. These tools are not used to summarize or analyze past data (which would be descriptive analytics), but rather to make forward-looking predictions based on training data. Hence, both are best categorized under predictive analytic.

3. 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 are examples of supervised learning. In both tools, users provide labeled training data, which allows the model to learn the relationship between inputs and outputs. For example, a student might label text as "positive" or "negative" in Machine Learning for Kids, or label images as different classes in Teachable Machine. The model then learns to classify new inputs based on this training. Since both tools rely on labeled datasets, they fall under the supervised learning category.

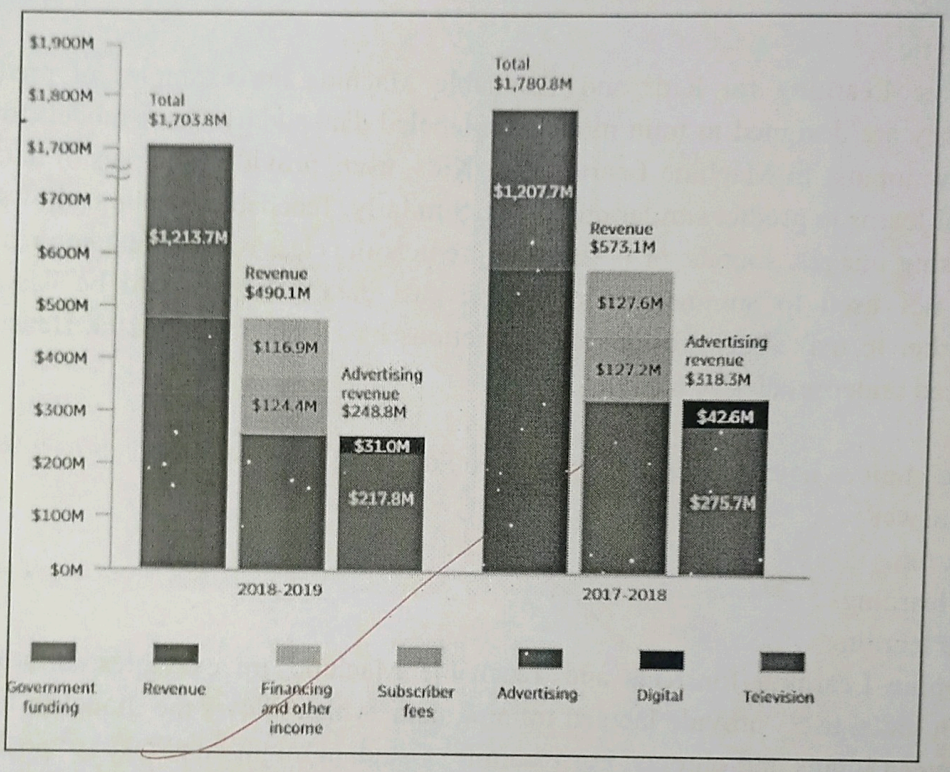
**Q.3** Data Visualization: Read the following two short articles:

- 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.**

Misleading CBC(Canadian national broadcast company) Funding Bar Chart:

In April 2023, a bar chart from the Canadian Broadcasting Corporation's (CBC) 2018–2019 Annual Report resurfaced online and quickly drew widespread criticism for its misleading design. The chart was intended to show CBC's funding sources, including government appropriations and advertising revenue. However, it sparked controversy after viewers noticed serious issues with how the data was presented.




The axis had a sudden break—jumping from $700M to $1.7B—causing a $490M revenue bar to appear visually larger than the $1.2B government funding bar. This misleading scale made it seem like television revenue equaled or exceeded government funding, which is factually incorrect. Additionally, the chart layout was flawed: revenue and advertising revenue were shown as separate bars rather than subdivisions of the total income, creating further confusion.

Sources:

https://thedeepdive.ca/cbc-commits-a-chart-crime-in-representing-its-funding-and-revenue-sources/

https://www.codeconquest.com/blog/12-bad-data-visualization-examples-explained/htoc-bad-data-visualization-example-2

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

**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)
```

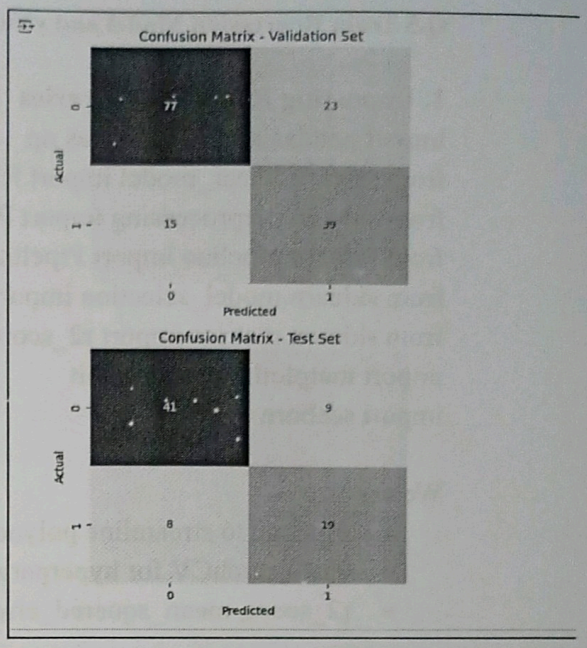
```
GaussianNB
GaussianNB()
```

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_val, y_val_pred))

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

```
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**

**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**

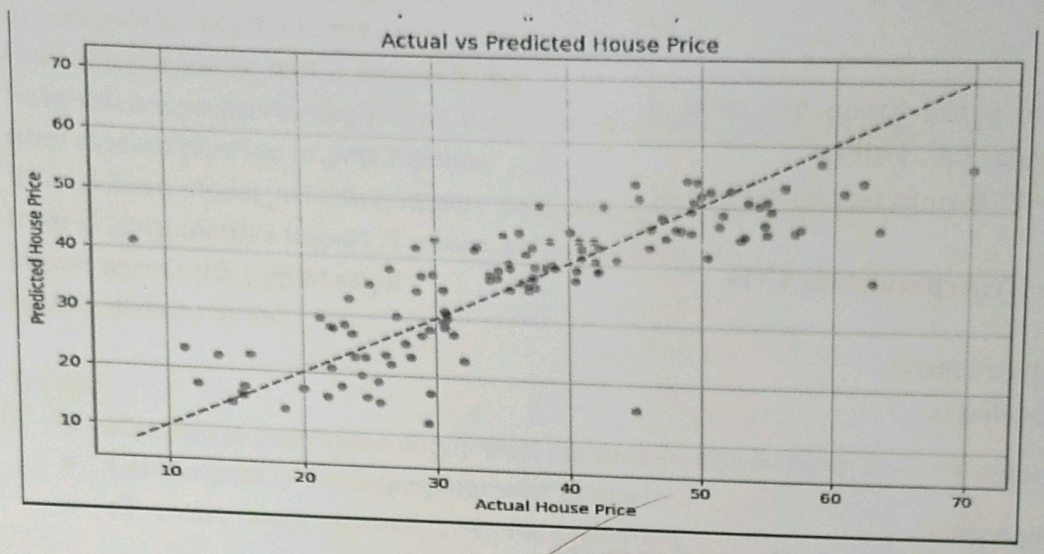
```
best_model = reg_model.train(X_train, y_train)
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)
```

- 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

**Ans.**

The Wine Quality dataset from Kaggle includes several key physicochemical features that influence the quality of wine, such as

fixed acidity,

volatile acidity,
citric acid,
residual sugar,
chlorides,
free and total sulfur dioxide,
density,
pH,
sulphates,
alcohol.

Each of these features plays a role in determining the taste, stability, and preservation of the wine. For instance, high levels of volatile acidity often lower the quality, while alcohol content and sulphates generally show a positive correlation with better wine ratings.

During the feature engineering process, handling missing data is crucial for model performance. Although the original dataset is relatively clean, if missing values are present, the following techniques can be used:

1. Remove Missing Data:
Delete rows or columns with missing values.
Best when only a small portion of data is missing.

2. Fill with Mean:
Replace missing values with the column's average.
Works well for normally distributed numerical data.

3. Fill with Median:
Use the middle value of the column for imputation.
Better than mean when data has outliers or is skewed.

5. Fill with Constant:
Replace missing data with a fixed value like 0 or "unknown."
Helps retain rows while signaling missingness.

Advantages and disadvantages of different imputation techniques

1. <u>Mean Imputation</u>

Advantage: Simple to implement and works well with normally distributed data.
Disadvantage: Sensitive to outliers and may distort the overall variance.

2. <u>Median Imputation</u>

Advantage: More robust than mean; not affected by outliers.
Disadvantage: Does not consider relationships between different features.

3. Mode Imputation

Advantage: Ideal for categorical data; retains the most common category.
Disadvantage: Can lead to overrepresentation of one category.

4. Constant Value Imputation

Advantage: Easy to implement and helps highlight missing data.
Disadvantage: May introduce meaningless or misleading values.