

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

Ans)

1. **Mean**:-

Mean = (Sum of all values) ÷ (Number of values)

- Sum =  $82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611$
- Number of values = 20

**Mean =  $1611 / 20 = 80.55$**

2. **Median**:-

First, sort the data into ascending order: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

There are 20 values (even number), so 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**:-

Mode = Most frequently occurring number(s)

From the sorted list:

76 appears **3 times**, more than any other value.

**Mode = 76**

#### 4. Interquartile Range (IQR):-

**Q1** = 25th percentile of the given values

= Median of the first half (first 10 numbers):

59, 64, 66, 70, 76, 76, 76, 78, 79, 81

$$Q1 = (5\text{th} + 6\text{th}) / 2 = (76 + 76) / 2 = \mathbf{76}$$

**Q3** = 75th percentile of the given values

= Median of the second half (last 10 numbers):

82, 82, 84, 85, 88, 90, 90, 91, 95, 99

$$Q3 = (5\text{th} + 6\text{th}) / 2 = (88 + 90) / 2 = \mathbf{89}$$

$$\mathbf{IQR = Q3 - Q1 = 89 - 76 = \underline{13}}$$

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

2. From the two choices listed below, how would you describe each tool listed above?

Why did you choose the answer?

a. Predictive analytic

b. Descriptive analytic

3. From the three choices listed below, how would you describe each tool listed above?

Why did you choose the answer?

a. Supervised learning

b. Unsupervised learning

c. Reinforcement learning

Ans)

1) **Tool Comparison:-**

1. Machine Learning for Kids

a. Target Audience:

Primarily designed for young students (ages 8–16) and educators introducing machine learning concepts in schools.

b. Use by Target Audience:

- Students use visual interfaces to create models by training them on labeled data (e.g., text, images, numbers).
- Integrates with Scratch and Python to build interactive ML-based projects (e.g., chatbots, games).
- Teachers use it in classrooms to explain AI and machine learning fundamentals.

c. Benefits:

- User-friendly and tailored for education.
- Visual and coding integration helps in real application.
- No programming background needed for basic use.

d. Drawbacks:

- Limited complexity — not suitable for advanced ML tasks.
- Mainly focuses on classification — fewer regression or clustering tasks.

## 2. Teachable Machine

### a. Target Audience:

Aimed at students, educators, artists, hobbyists, and beginners in machine learning.

### b. Use by Target Audience:

- Users upload or record images, audio, or pose data.
- Train models directly in the browser without writing code.
- Export models to use in websites or projects (e.g., with TensorFlow.js).

### c. Benefits:

- Extremely easy to use, fast model training.
- Supports multiple input types (image, audio, pose).
- No sign-in or installation required.

### d. Drawbacks:

- Limited customization or control over model parameters.
- Models may lack robustness for real-world deployment.

## 2) Predictive Analytic vs Descriptive Analytic:-

- Machine Learning for Kids: Predictive Analytic  
This is because it focuses on training models to predict outcomes (e.g., classifying emotions, sorting objects based on input), which is a form of predictive analysis.
- Teachable Machine: Predictive Analytic  
This is because it trains models to make real-time predictions based on user inputs (e.g., classifying audio, images, or poses).

## 3) Type of Learning:-

- Machine Learning for Kids: Supervised Learning  
This is because students provide labeled data (e.g., “happy” vs. “sad” texts) for training which is a clear example of supervised learning.
- Teachable Machine: Supervised Learning  
This is because users label data themselves (e.g., Image A = "Cat", Image B = "Dog"), which the model uses to learn which is a clear example of supervised learning.

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) A significant example of misleading data visualization occurred in May 2020 when the Georgia Department of Public Health released a graph showing COVID-19 cases across five counties. The graph misrepresented the data in several critical ways:

1. **Non-Sequential Ordering of Dates:** The x-axis dates were presented out of sequence, which created an artificial sense of a consistent decline in cases. This misrepresentation led viewers to believe the situation was improving steadily when, in reality, the data was distorted by improper sequencing, making it appear as if the decline in cases was more consistent than it was. This error in chronological ordering significantly skewed the interpretation of the trends.
2. **Reordering of Counties:** The counties were reordered daily, making it difficult to track trends for individual counties over time. By constantly changing the order, the graph obscured meaningful comparisons and created a false narrative of a uniform decline across all counties, even though each county had its own unique trends. This also made it harder for viewers to understand the performance of each county separately.
3. **Lack of Contextual Information:** The graph lacked key details such as the dates of significant changes or explanations for data fluctuations. This omission gave the misleading impression that cases were declining significantly, downplaying the ongoing risks and complexities of the pandemic. Without these critical details, viewers were left with an incomplete picture.

This misleading visualization created a false perception of the situation, leading many viewers to believe that the pandemic was under control, which could have contributed to misguided confidence. This, in turn, may have influenced public decisions regarding safety measures like social distancing and mask-wearing. Misleading visualizations like this can have serious consequences for public health policy, as they may encourage premature relaxation of restrictions, which could lead to further outbreaks and unnecessary health risks.

This case raises important ethical concerns regarding data visualization, particularly in public health contexts. While data visualization is a powerful tool for communication, when used improperly, it can distort the interpretation of critical data, mislead the public, and undermine trust in the authorities. Ethical data visualization practices are essential

to ensure clarity, accuracy, and transparency. Such practices help the public and policymakers make well-informed decisions based on a truthful interpretation of the data, which ultimately leads to better public health outcomes.

In conclusion, the Georgia Department of Public Health's misleading graph highlights the dangers of poorly constructed visualizations and their potential to skew perceptions and influence public behavior. It underscores the need for accurate, transparent, and ethically designed data visualizations, particularly in public health crises, to avoid harmful consequences. Misleading visualizations undermine public trust and hinder informed decision-making in critical situations.

**Source:** Anderson Review. (2020). Graphic Presentation of COVID-19 Data Can Skew Perceptions of Risk. Retrieved from Anderson Review.

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:

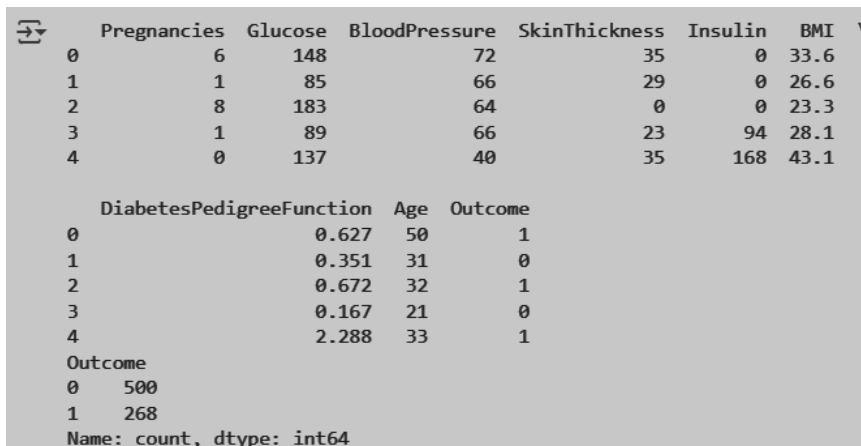
- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python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

Dataset: [Pima Indians Diabetes Database](#)

Ans) Steps to carry out the above mentioned operations:

Step 1: Import required libraries and load the dataset:

```
Code: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")
data = pd.read_csv("diabetes.csv")
print(data.head())
print(data['Outcome'].value_counts())
```



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

	DiabetesPedigreeFunction	Age	Outcome
0	0.627	50	1
1	0.351	31	0
2	0.672	32	1
3	0.167	21	0
4	2.288	33	1

Outcome	
0	500
1	268

Name: count, dtype: int64

## Step 2: Data Preprocessing:

```
Code: print(data.isnull().sum())
z_scores = np.abs(stats.zscore(data.drop('Outcome', axis=1)))
data = data[(z_scores < 3).all(axis=1)]
print("Data shape after outlier removal:", data.shape)
X = data.drop('Outcome', axis=1)
y = data['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
➡ Pregnancies      0
   Glucose          0
   BloodPressure    0
   SkinThickness    0
   Insulin          0
   BMI             0
   DiabetesPedigreeFunction  0
   Age             0
   Outcome          0
   dtype: int64
   Data shape after outlier removal: (688, 9)
```

## Step 3: Train/Val/Test Split (70/20/10) - Randomized:

```
Code: X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3,
random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42,
stratify=y_temp)
print("Train size:", len(X_train), "Validation size:", len(X_val), "Test size:", len(X_test))
```

```
➡ Train size: 481 Validation size: 138 Test size: 69
```

## Step 4: Resolve Class Imbalance Using SMOTE:

```
Code: sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
print("Balanced classes in training set:", np.bincount(y_train_res))
```

```
➡ Balanced classes in training set: [322 322]
```

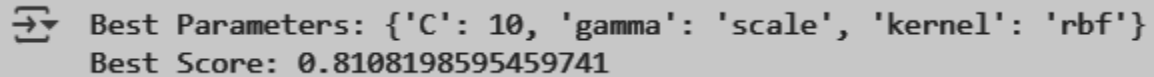
The code uses SMOTE to fix class imbalance by creating fake examples of the smaller class in the training data. It adjusts the training set to have an equal number of examples for each class and then shows the new class distribution to confirm the balance.

## Step 5: Hyperparameter Tuning with GridSearchCV (SVM):

```
Code:
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```



```
grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(np.vstack((X_train_res, X_val)), np.hstack((y_train_res, y_val)))
print("Best Parameters:", grid.best_params_)
print("Best Score:", grid.best_score_)
```



```
➤ Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best Score: 0.8108198595459741
```

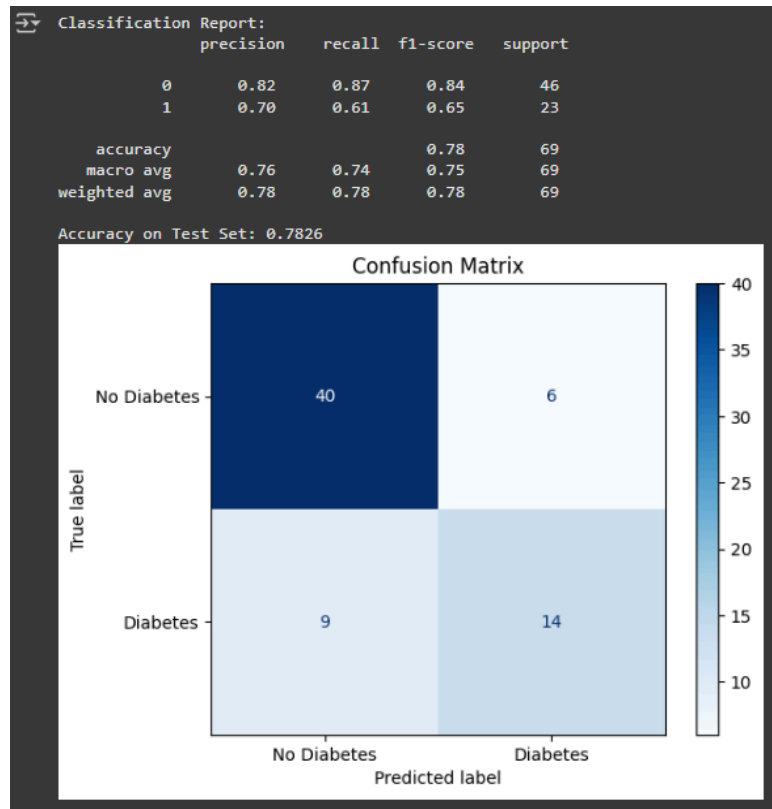
The code performs hyperparameter tuning for an SVM model using GridSearchCV. It tests different values for the C, kernel, and gamma parameters, using 5-fold cross-validation to find the best combination that maximizes accuracy. The best parameters and score are then printed.

#### Step 6: Train Final Model with Best Parameters:

```
Code: final_model = grid.best_estimator_
final_model.fit(np.vstack((X_train_res, X_val)), np.hstack((y_train_res, y_val)))
```

#### Step 7: Evaluate on Test Set:

```
Code: from sklearn.metrics import accuracy_score
y_pred = final_model.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on Test Set: {accuracy:.4f}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Diabetes",
"Diabetes"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()
```



The classification model achieves an overall accuracy of 78.26% on the test set. It performs better at identifying non-diabetic cases (class 0) with a precision of 0.82 and recall of 0.87, compared to diabetic cases (class 1) where precision drops to 0.70 and recall to 0.61. The confusion matrix shows the model correctly classified 40 non-diabetic and 14 diabetic cases, but it misclassified 6 non-diabetics as diabetic and 9 diabetics as non-diabetic, indicating a need for improvement in detecting diabetes cases.

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 the 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

Dataset: <https://github.com/Sutanoy/Public-Regression-Datasets/blob/main/Heart.csv>

Ans) Steps to carry out the above mentioned operations:

Step 1: Import libraries and load the dataset:

```
Code: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("Heart.csv") # replace with path if needed
df.head()
```

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Step 2: OOP Approach – Define a Regression Pipeline Class:

Code: class HeartRegressionModel:

```
def __init__(self, data):
    self.df = data.copy()
    self.model = None
    self.X_train = None
    self.X_test = None
```

```

        self.y_train = None
        self.y_test = None
        self.y_pred = None
    def preprocess(self):
        if 'Unnamed: 0' in self.df.columns:
            self.df.drop(columns=['Unnamed: 0'], inplace=True)
        self.df = pd.get_dummies(self.df, drop_first=True)
        self.X = self.df.drop(columns=['Age']) # Predicting 'age'
        self.y = self.df['Age']
        scaler = StandardScaler()
        self.X = scaler.fit_transform(self.X)
    def split_data(self):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            self.X, self.y, test_size=0.3, random_state=None
        )
    def tune_model(self):
        model = XGBRegressor(objective='reg:squarederror', random_state=42)
        params = {
            'n_estimators': [100, 200],
            'max_depth': [3, 5, 7],
            'learning_rate': [0.01, 0.1, 0.2],
            'subsample': [0.8, 1],
            'colsample_bytree': [0.8, 1]
        }
        grid = GridSearchCV(model, params, cv=5, scoring='r2', n_jobs=-1)
        grid.fit(self.X_train, self.y_train)
        self.model = grid.best_estimator_
        print(" Best Parameters:", grid.best_params_)
    def train_model(self):
        self.model.fit(self.X_train, self.y_train)
    def evaluate(self):
        self.y_pred = self.model.predict(self.X_test)
        r2 = r2_score(self.y_test, self.y_pred)
        n = len(self.y_test)
        p = self.X_test.shape[1]
        adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
        print("\nModel Evaluation:")
        print(f"R² Score: {r2:.5f}")
        print(f"Adjusted R² Score: {adjusted_r2:.5f}")
        print(f"Mean Squared Error: {mean_squared_error(self.y_test, self.y_pred):.5f}")
        print(f"Mean Absolute Error: {mean_absolute_error(self.y_test, self.y_pred):.5f}")
        return adjusted_r2
    def visualize_predictions(self):
        plt.figure(figsize=(8, 6))
        sns.scatterplot(x=self.y_test, y=self.y_pred)
        plt.xlabel("Actual Age")
        plt.ylabel("Predicted Age")
        plt.title("Actual vs Predicted Age")

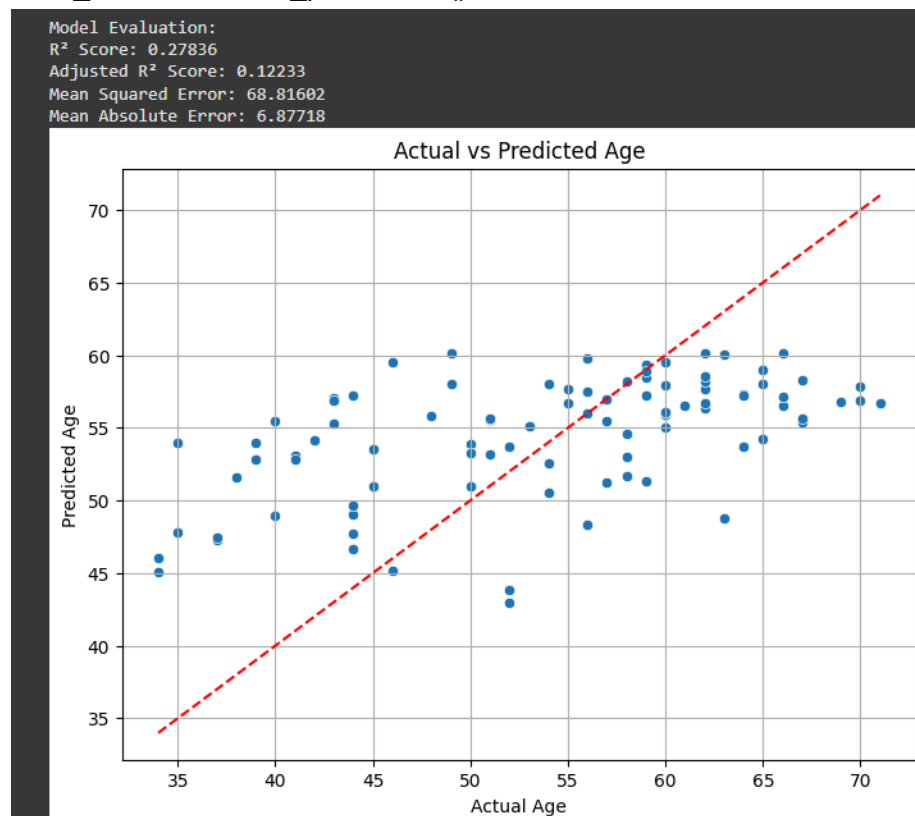
```

```
plt.plot([self.y_test.min(), self.y_test.max()],
         [self.y_test.min(), self.y_test.max()],
         color='red', linestyle='--')
plt.grid(True)
plt.show()
```

The HeartRegressionModel class performs regression on the heart dataset to predict the target variable (Age). It includes methods for preprocessing the data (handling categorical features and scaling), splitting it into training and test sets, tuning hyperparameters using XGBRegressor with GridSearchCV, training the model, and evaluating its performance (including  $R^2$ , adjusted  $R^2$ , MSE, and MAE). Additionally, it visualizes the model's predictions against the actual values using a scatter plot.

### Step 3: Run the Complete Pipeline:

```
Code: heart_model = HeartRegressionModel(df)
heart_model.preprocess()
heart_model.split_data()
heart_model.tune_model()
heart_model.train_model()
adjusted_r2 = heart_model.evaluate()
heart_model.visualize_predictions()
```



The regression model performs poorly, with an  $R^2$  score of 0.28 and an adjusted  $R^2$  of 0.12, suggesting it explains little variance in the data. The mean absolute error of 6.88 indicates significant deviations between actual and predicted ages. The scatter plot shows that predictions are biased toward the mean, often underestimating higher ages and overestimating lower ones.

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)

- **Key Features of the Wine Quality Dataset:**

1. Fixed Acidity – Tartaric acid; affects taste and stability.
2. Volatile Acidity – High levels give vinegar taste; lowers quality.
3. Citric Acid – Adds freshness; improves quality in moderation.
4. Residual Sugar – Impacts sweetness; minimal effect on quality.
5. Chlorides – Salt content; too much is bitter.
6. Free & Total Sulfur Dioxide – Preserves wine; excess affects taste.
7. Density – Related to sugar/alcohol; useful with other features.
8. pH – Acidity level; extremes can reduce quality.
9. Sulphates – Antimicrobial; moderate levels improve quality.
10. Alcohol – Higher alcohol often means better quality.
11. Quality – Target variable (score 0–10) from human tasters.

- **Importance of Features in Predicting Quality:**

- a. Highly Influential: Alcohol, volatile acidity, sulphates
- b. Moderately Influential: Citric acid, density, pH
- c. Low Influence: Residual sugar, chlorides, total sulfur dioxide

Importance is determined using feature correlation, model coefficients (like in logistic regression), or feature importance from tree-based models.

- **Handling Missing Data During Feature Engineering:**

Step 1: Detect Missing Data

- Use `.isnull().sum()` or `.info()` to identify missing values.

Step 2: Strategy Based on Data Type & Distribution:

- For numerical columns:
  - Impute with mean if data is normally distributed.
  - Use median for skewed distributions (robust to outliers).
  - In some models, missing values are imputed using KNN Imputer or IterativeImputer for better estimation.
- For outlier-influenced features:
  - Median is usually preferred to reduce distortion.

Step 3: Advanced Imputation (if needed):

- Use KNN imputation to estimate values based on the nearest neighbors.

- Use regression imputation where a feature is predicted using a model trained on the other features.

- **Advantages and Disadvantages of Imputation Techniques:**

Technique	Advantages	Disadvantages
<b>Mean Imputation</b>	Simple and fast	Affected by outliers; reduces data variance
<b>Median Imputation</b>	Robust to outliers	Can still distort the distribution
<b>Mode Imputation</b>	Good for categorical data	Not suitable for continuous variables
<b>KNN Imputer</b>	Maintains feature relationships	Computationally expensive; sensitive to outliers
<b>Regression Imputer</b>	Captures complex relationships	Assumes linearity; can overfit
<b>Drop Missing Rows</b>	Simplifies dataset	Risk of losing valuable data and reducing model performance