## Name-Ishan Kiran Joshi  Div-D15C  Roll No- 21  A.Y.-2024-25

## AIDS Assingment 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**

**1. Mean (Average)**

$$\text{Mean} = \frac{\text{Sum of all values}}{\text{Number of values}}$$

Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = **1611**
Count = 20

Mean=1611/20 =80.55

**2. Median**

Step 1: Sort the data:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Step 2: Since there are 20 values (even), the median is the average of the 10th and 11th values:
10th = 81, 11th = 82

Median=(81+82)/2=81.5

**3. Mode**

Find the value(s) that appear most frequently:

- 76 appears **3 times**
- All other numbers appear less frequently

Mode=76

**4. IQR**

Q1 = 1st quartile (25th percentile) → median of the first 10 values:
First half = 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Q1 = average of 5th and 6th values = (76 + 76) ÷ 2 = 76
Q3 = 3rd quartile (75th percentile) → median of the last 10 values:
Second half = 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Q3 = average of 5th and 6th values = (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**
**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 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**

**1. Tool Evaluation**

Machine Learning for Kids is designed for school students, typically between the ages of 8 to 16, and for educators who want to introduce basic concepts of artificial intelligence in classrooms. Students use this tool to create simple machine learning models through an easy drag-and-drop interface. It works well with Scratch and Python, allowing them to

experiment with projects involving image or text classification. Educators use it as a teaching aid to make machine learning concepts more accessible and interactive.

The main benefits of Machine Learning for Kids include its beginner-friendly design, integration with kid-friendly platforms, and the fact that no coding knowledge is required. However, one drawback is that it only offers limited functionality, which may not be enough for more advanced or in-depth exploration of machine learning. Also, it can sometimes oversimplify complex ideas, which might leave gaps in understanding for older or more advanced learners.

Teachable Machine, developed by Google, is aimed at beginners, hobbyists, students, and educators with little to no programming experience. It allows users to train models for recognizing images, sounds, or body poses using a webcam or microphone. This interactive tool helps users build and test machine learning models in real time and even export them for use in web or mobile applications.

Teachable Machine is beneficial because it requires no coding, supports live input, and allows for exporting models to real projects. Its visual and interactive nature makes it an effective tool for quick demonstrations and learning. However, its major drawbacks are that it lacks deeper customization and is limited to simple classification tasks. It's not suitable for handling complex problems or large datasets.

**2. Predictive vs. Descriptive Analytics**

Both Machine Learning for Kids and Teachable Machine are examples of predictive analytics. They are used to train a model using input-output examples so that the system can make future predictions on unseen data. Machine Learning for Kids teaches students how to build a model that can predict outcomes based on previous training, like whether a sentence is positive or negative. Similarly, Teachable Machine is used to train models to recognize things like a dog or cat based on labeled images. Since both tools generate predictions from the data provided, they clearly fall under the predictive analytic category rather than descriptive analytics.

**3. Type of Machine Learning**

Both tools are examples of supervised learning. This is because the user provides labeled data during training. For instance, in Machine Learning for Kids, users label training examples (like labeling images as "cat" or "dog") and the system learns from these labeled inputs. The same goes for Teachable Machine—users upload or record data and

assign labels (such as “happy” or “sad”) before training the model. The tools then learn to associate the input with the given label and use that to make future predictions. Since both rely on labeled data provided during training, they are categorized under 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.**

Answer- The article by Arthur Kakande titled “What’s in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization” emphasizes how easily data can be manipulated through poor design choices. It explains how misleading charts often distort scale, omit context, or use confusing visuals that create false impressions. Kakande stresses the importance of being a critical reader of data visualizations, suggesting that readers should always ask: what’s missing, what’s exaggerated, and who benefits from the way data is shown?

Similarly, Katherine Ellen Foley’s article “How bad Covid-19 data visualizations mislead the public” highlights real-world consequences of bad visual design during the pandemic. She discusses how some graphs used incorrect axes, distorted proportions, or lacked context, causing the public to misunderstand the severity or spread of COVID-19. These misleading visuals contributed to confusion, fear, or a false sense of safety, depending on how the data was displayed.

A current example of misinformation in data visualization can be seen in the coverage of inflation rates in the U.S. during 2022–2023. A widely shared chart from a political tweet used a broken Y-axis that exaggerated the impact of inflation by starting the axis at 7% instead of 0%. This design made small differences appear dramatic and led readers to believe inflation was spiking uncontrollably, even though the actual data showed it had started to stabilize. The visual misled the public by manipulating perception through scale rather than content. This example was discussed in an article from The Washington Post titled “This chart about inflation is going viral. But it’s misleading.” (Published April

2023). The article analyzed how political messaging used the chart to stir panic, even though the real situation was improving.

This example shows that even truthful data can be misrepresented through poor or biased visual design. It underscores the importanc]e of transparency, scale integrity, and context when creating or interpreting data visualizations.

**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

SVM

```
[1]  # Step 1: Install necessary libraries
     !pip install imbalanced-learn scikit-learn

     # Step 2: Import libraries
     import pandas as pd
     import numpy as np
     from sklearn.model_selection import train_test_split, GridSearchCV
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
     from imblearn.over_sampling import SMOTE
     import matplotlib.pyplot as plt
     import seaborn as sns
```

**Step 1: Install Libraries**

Install required Python libraries (imbalanced-learn, scikit-learn) that help in handling imbalance and building ML models.

This ensures your Colab environment has everything ready.

**Step 2: Import Libraries**

Import all necessary modules like pandas, numpy, sklearn, matplotlib, and seaborn.

These will be used for data processing, modeling, and evaluation.

```
[2]  # Step 3: Load the dataset
     df = pd.read_csv('/content/diabetes.csv')  # update path as needed
     X = df.iloc[:, :-1]
     y = df.iloc[:, -1]

     # Step 4: Data Preprocessing
     # Replace zero values in columns that shouldn't have zero
     cols_with_zero = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
     X[cols_with_zero] = X[cols_with_zero].replace(0, np.nan)
     X.fillna(X.mean(), inplace=True)

     # Feature Scaling
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
```

**Step 3: Load the Dataset**

Read the CSV file into a pandas DataFrame.

Separate the feature set X and the target label y (last column Outcome).

**Step 4: Data Preprocessing**

Replace invalid zero values in selected columns with NaN, then fill missing values with column means.

Normalize all features using StandardScaler to bring them to a common scale.

```
[3]  # Step 5: Train-Validation-Test Split (70/20/10)
     X_temp, X_test, y_temp, y_test = train_test_split(X_scaled, y, test_size=0.1, random_state=42, stratify=y)
     X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=2/9, random_state=42, stratify=y_temp)

     # Step 6: Handle Class Imbalance
     smote = SMOTE(random_state=42)
     X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

**Step 5: Data Splitting**

Split the data randomly into training (70%), validation (20%), and test (10%) sets.

Use stratified splitting to maintain the class distribution.

**Step 6: Handle Class Imbalance**

Use SMOTE (Synthetic Minority Over-sampling Technique) to generate synthetic

samples of the minority class.
This balances the training data for better model learning.

```
[4] # Step 7: Model Training & Hyperparameter Tuning (SVM example)
    param_grid = {
        'C': [0.1, 1, 10],
        'gamma': ['scale', 'auto'],
        'kernel': ['rbf', 'linear']
    }
    grid = GridSearchCV(SVC(), param_grid, refit=True, cv=5, verbose=0)
    grid.fit(X_train_res, y_train_res)

    print("Best Parameters:", grid.best_params_)

    # Step 8: Validation Performance
    y_val_pred = grid.predict(X_val)
    print("\nValidation Results:")
    print(confusion_matrix(y_val, y_val_pred))
    print(classification_report(y_val, y_val_pred))
    print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
```

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

Validation Results:
[[74 26]
 [23 31]]
              precision    recall  f1-score   support

           0       0.76      0.74      0.75       100
           1       0.54      0.57      0.56        54

    accuracy                           0.68       154
   macro avg       0.65      0.66      0.65       154
weighted avg       0.69      0.68      0.68       154

Validation Accuracy: 0.6818181818181818
```

**Step 7: Train SVM with Hyperparameter Tuning**

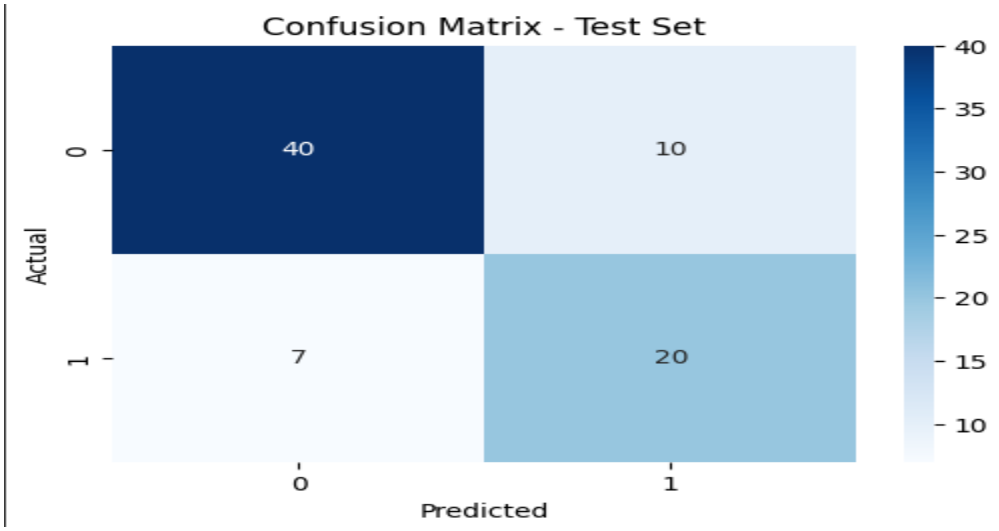
Use GridSearchCV to search for the best combination of SVM hyperparameters (C, gamma, kernel).
This improves model performance on unseen data.

**Step 8: Validate the Model**

Evaluate the trained model on the validation set using metrics like accuracy, precision, recall, and F1-score.
Also print the confusion matrix for deeper insights.

```
# Step 9: Final Testing
y_test_pred = grid.predict(X_test)
print("\nTest Results:")
print(confusion_matrix(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))

# Step 10: Visualize Confusion Matrix
plt.figure(figsize=(6,4))
sns.heatmap(confusion_matrix(y_test, y_test_pred), annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Test Set")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Test Results:
[[40 10]
 [ 7 20]]
              precision    recall  f1-score   support

           0       0.85      0.80      0.82        50
           1       0.67      0.74      0.70        27

    accuracy                           0.78        77
   macro avg       0.76      0.77      0.76        77
weighted avg       0.79      0.78      0.78        77

Test Accuracy: 0.7792207792207793
```




**Step 9: Test the Model**

Use the test set to check final model performance after tuning and validation.
This gives an unbiased estimate of real-world accuracy.

**Step 10: Visualize Results**

Plot the confusion matrix for the test set using Seaborn heatmap.
It visually shows true vs. predicted classifications.

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

- Data File: Regression data.csv

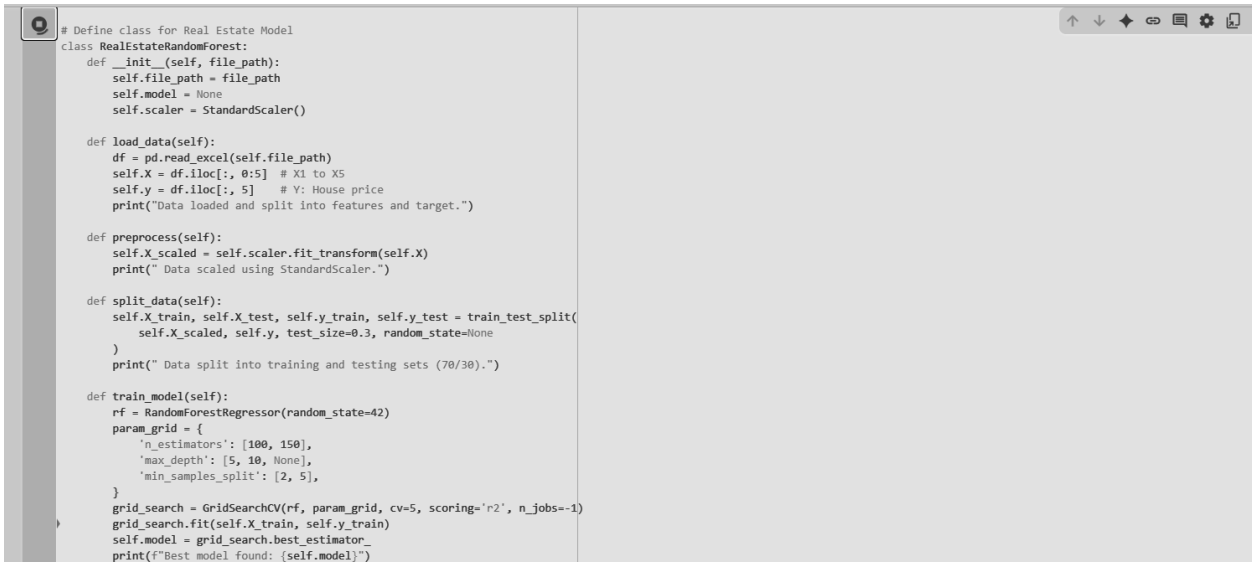
- Independent Variable: 1st Column
- Dependent variables: Column 2 to 5

  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

```
# Install dependencies
!pip install openpyxl scikit-learn -q

# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
```

This block  sets up the environment for running a machine learning regression task. It begins by installing essential libraries like openpyxl for reading Excel files and scikit-learn for machine learning tools. It then imports commonly used libraries for data handling (pandas, numpy), data visualization (matplotlib), preprocessing (StandardScaler), model training (RandomForestRegressor), data splitting (train_test_split), hyperparameter tuning (GridSearchCV), and evaluation metrics like $R^2$, RMSE, and MAE.



```python
# Define class for Real Estate Model
class RealEstateRandomForest:
    def __init__(self, file_path):
        self.file_path = file_path
        self.model = None
        self.scaler = StandardScaler()

    def load_data(self):
        df = pd.read_excel(self.file_path)
        self.X = df.iloc[:, 0:5]  # X1 to X5
        self.y = df.iloc[:, 5]    # Y: House price
        print("Data loaded and split into features and target.")

    def preprocess(self):
        self.X_scaled = self.scaler.fit_transform(self.X)
        print(" Data scaled using StandardScaler.")

    def split_data(self):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
            self.X_scaled, self.y, test_size=0.3, random_state=None
        )
        print(" Data split into training and testing sets (70/30).")

    def train_model(self):
        rf = RandomForestRegressor(random_state=42)
        param_grid = {
            'n_estimators': [100, 150],
            'max_depth': [5, 10, None],
            'min_samples_split': [2, 5],
        }
        grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2', n_jobs=-1)
        grid_search.fit(self.X_train, self.y_train)
        self.model = grid_search.best_estimator_
        print(f"Best model found: {self.model}")
```

This code defines a Python class RealEstateRandomForest that encapsulates the entire workflow of a regression model using the Random Forest algorithm. The __init__ method initializes the model with a file path and sets up a standard scaler for preprocessing. The load_data method reads the Excel file and separates it into features (X1 to X5) and the target (house price). The preprocess method standardizes the features using StandardScaler to ensure consistent scaling. The split_data method randomly divides the data into training and testing sets using a 70/30 ratio. Finally, the train_model method uses GridSearchCV to tune the Random Forest hyperparameters and select the best-performing model based on cross-validated $R^2$ score.

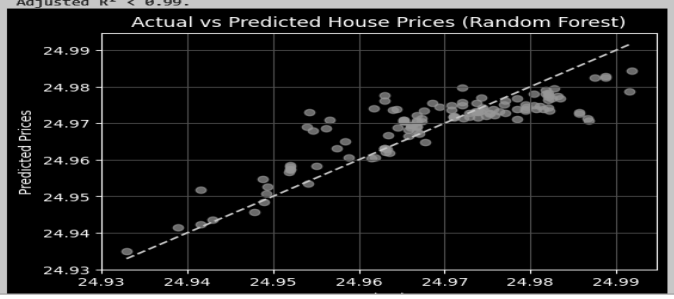
```python
def evaluate(self):
    y_pred = self.model.predict(self.X_test)
    r2 = r2_score(self.y_test, y_pred)
    n = len(self.y_test)
    k = self.X_test.shape[1]
    adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)
    rmse = np.sqrt(mean_squared_error(self.y_test, y_pred))
    mae = mean_absolute_error(self.y_test, y_pred)

    print(f"\n Evaluation Metrics:")
    print(f"R² Score: {r2:.4f}")
    print(f"Adjusted R² Score: {adj_r2:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"MAE: {mae:.4f}")

    if adj_r2 < 0.99:
        print(" Adjusted R² < 0.99. ")

    # Plot actual vs predicted
    plt.figure(figsize=(6, 4))
    plt.scatter(self.y_test, y_pred, alpha=0.7)
    plt.plot([self.y_test.min(), self.y_test.max()],
             [self.y_test.min(), self.y_test.max()], 'r--')
    plt.xlabel("Actual Prices")
    plt.ylabel("Predicted Prices")
    plt.title("Actual vs Predicted House Prices (Random Forest)")
    plt.grid(True)
    plt.show()
```

The evaluate method assesses the performance of the trained Random Forest model. It first makes predictions on the test data and calculates several key metrics: $R^2$ score (which shows how well the model fits), adjusted $R^2$ (which adjusts $R^2$ for the number of predictors), RMSE (root mean squared error), and MAE (mean absolute error), giving a

complete view of prediction accuracy. Finally, it creates a scatter plot comparing actual vs predicted prices, helping visualize how closely the predictions match real values.

```
file_path = "/content/Real estate valuation data set.xlsx"

rf_model = RealEstateRandomForest(file_path)
rf_model.load_data()
rf_model.preprocess()
rf_model.split_data()
rf_model.train_model()
rf_model.evaluate()
```

```
Data loaded and split into features and target.
 Data scaled using StandardScaler.
 Data split into training and testing sets (70/30).
Best model found: RandomForestRegressor(max_depth=10, min_samples_split=5, n_estimators=150,
                        random_state=42)

 Evaluation Metrics:
R² Score: 0.7358
Adjusted R² Score: 0.7247
RMSE: 0.0062
MAE: 0.0047
 Adjusted R² < 0.99.
```




This shows the execution of a Random Forest Regression model on a real estate dataset. The model achieves an $R^2$ score of 0.7358, meaning it explains about 73% of the variance in house prices. The Adjusted $R^2$ is 0.7247, slightly lower. The scatter plot shows that the model's predictions are reasonably close to actual values, but still exhibit some variance, suggesting potential for more feature engineering or a more complex model.

**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).**

The Wine Quality Dataset consists of 12 attributes that describe various physicochemical properties of wine samples and their corresponding quality score. After correctly loading the dataset using delimiter=';', we can explore the key features and understand their influence on wine quality.

Key Features and Their Importance:

1. Fixed Acidity – Represents non-volatile acids. It affects the wine’s sharpness and structure.

2. Volatile Acidity – Higher values are undesirable, as they produce vinegary flavors. Negatively correlated with quality.
3. Citric Acid – Enhances freshness and flavor. Moderate levels are seen in higher-quality wines.
4. Residual Sugar – The remaining sugar after fermentation. Its influence on quality is limited in dry wines.
5. Chlorides – Reflects salt content; excessive amounts negatively affect taste and quality.
6. Free Sulfur Dioxide – Prevents microbial growth. Proper balance ensures wine stability.
7. Total Sulfur Dioxide – Total concentration of both bound and free forms. Too much can create an off-putting smell.
8. Density – A measure related to sugar and alcohol content. Lower density is usually better in red wines.
9. pH – Indicates acidity level. Wines with balanced pH are generally rated better.
10. Sulphates – Acts as a preservative and contributes to flavor. Positively correlated with wine quality.
11. Alcohol – Strongly correlated with wine quality. Wines with higher alcohol content tend to have richer flavors.
12. Quality (Target Variable) – Integer score ranging typically from 3 to 8, representing the overall quality based on sensory data.

**Handling Missing Data (Feature Engineering Step):**

1. Mean Imputation:
   Replaces missing values with the average of the column.
   Advantage: Simple and fast.
   Disadvantage: Can distort the distribution, especially if outliers are present.

2. Median Imputation:
   Fills missing values with the median.
   Advantage: Better than mean for skewed data.
   Disadvantage: Still a constant value; doesn't account for relationships between variables.

3. KNN Imputation (K-Nearest Neighbors):
   Predicts missing values based on the similarity to other rows.
   Advantage: Maintains data patterns and relationships.

Disadvantage: Computationally intensive and may be influenced by irrelevant features if not scaled properly.

The SimpleImputer and KNNImputer classes from Scikit-learn to demonstrate these techniques, even though the dataset didn't require actual imputation.

```
import pandas as pd
from sklearn.impute import SimpleImputer, KNNImputer
import matplotlib.pyplot as plt
import seaborn as sns

# Load the wine quality dataset (red wine as an example)
df = pd.read_csv("winequality-red.csv", delimiter=';')


# Display basic info
print("First 5 rows:\n", df.head())
print("\nMissing values:\n", df.isnull().sum())

# If there were missing values, here's how to handle them:

# 1. Mean Imputation
mean_imputer = SimpleImputer(strategy='mean')
df_mean_imputed = pd.DataFrame(mean_imputer.fit_transform(df), columns=df.columns)

# 2. Median Imputation
median_imputer = SimpleImputer(strategy='median')
df_median_imputed = pd.DataFrame(median_imputer.fit_transform(df), columns=df.columns)

# 3. KNN Imputation
knn_imputer = KNNImputer(n_neighbors=3)
df_knn_imputed = pd.DataFrame(knn_imputer.fit_transform(df), columns=df.columns)

# Feature correlation with wine quality
correlation = df.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation, annot=True, cmap='coolwarm')
plt.title("Feature Correlation with Wine Quality")
plt.show()

# Sort features based on correlation with quality
quality_corr = correlation['quality'].drop('quality').sort_values(ascending=False)
print("\nTop features influencing quality:\n", quality_corr)
```

```
Top features influencing quality:
 alcohol                 0.476166
sulphates                0.251397
citric acid              0.226373
fixed acidity            0.124052
residual sugar           0.013732
free sulfur dioxide     -0.050656
pH                      -0.057731
chlorides               -0.128907
density                 -0.174919
total sulfur dioxide    -0.185100
volatile acidity        -0.390558
Name: quality, dtype: float64
```

**Conclusion:**

Each feature plays a role in shaping the final wine quality score. Among them, alcohol, volatile acidity, and sulphates are the most impactful. Proper feature engineering, including handling missing data, ensures a reliable and accurate model. Choosing the right imputation method depends on the data's nature, with trade-offs between simplicity and accuracy.