Data 71200 Final Paper on The Prediction with Machine Learning Model of Red Wine Quality Dataset.

- **Kelechi Iwuagwu**

Abstract

This paper explores the application of machine learning techniques on a dataset of red variants of the Portuguese "Vinho Verde" wine. The dataset contains various chemical properties of the wine, including "fixed_acidity", "volatile_acidity", "citric_acid", "residual_sugar", "chlorides", "free_sulfur_dioxide", "total_sulfur_dioxide", "density", "pH", and "sulphates". The goal is to predict the quality of the wine using both supervised and unsupervised learning methods. We employed K-Nearest Neighbors (KNN) and Random Forest for supervised learning and evaluated the effectiveness of PCA for feature selection. We applied K-Means, Hierarchical Clustering, and DBSCAN for unsupervised learning, both with and without PCA. The findings indicate that Random Forest outperforms KNN in predicting wine quality, and while PCA helps in reducing dimensionality, its impact on clustering performance varies.

1. Introduction

Dataset Description

The dataset comprises red variants of the Portuguese "Vinho Verde" wine, characterized by eleven chemical properties such as ***"fixed_acidity", "volatile_acidity", "citric_acid", "residual_sugar", "chlorides", "free_sulfur_dioxide", "total_sulfur_dioxide", "density", "pH", and "sulphates"***. The objective is to predict the quality rating of the wine, which ranges from 0 to 10.

Goals

The primary goal is to predict wine quality using supervised learning techniques and to explore the structure of the data through unsupervised learning. This involves:

1. Preprocessing and cleaning the dataset.
2. Visualizing the data to understand its distribution and relationships.
3. Applying and comparing the performance of K-Nearest Neighbors and Random Forest for supervised learning.
4. Using PCA for feature selection to enhance model performance.
5. Evaluating clustering algorithms like K-Means, Hierarchical Clustering, and DBSCAN with and without PCA.

2. Data Preprocessing

Data Cleaning

The initial dataset contained ***NO*** missing values *(Nan)* or potential outliers. There was no need for values to be imputed using the median of each feature.

Challenges

One challenge was dealing with the imbalanced distribution of the target variable (wine quality). This was addressed by ensuring balanced training and test splits *(80/20 principle)* and using appropriate performance metrics to evaluate the models.

3. Data Visualization

Exploratory Data Analysis (EDA)

Various visualizations were created to explore the data, including histograms, scatter plots, and correlation matrices. For instance, scatter plots of "alcohol" versus "quality" revealed a positive correlation, suggesting that higher alcohol content is often associated with higher quality ratings.

Insights Obtained.

EDA helped identify significant correlations between certain features and the target variable. It also highlighted the importance of scaling features and the potential benefits of dimensionality reduction techniques like PCA.

4. Supervised Learning Experiments

Algorithms Chosen

1. **K-Nearest Neighbors (KNN)**: Classifies samples based on the majority class of their nearest neighbors.
2. **Random Forest**: An ensemble method that builds multiple decision trees and merges them to obtain a more accurate and stable prediction.

Algorithm Descriptions

- **K-Nearest Neighbors (KNN)**: This algorithm classifies data points based on the class of their nearest neighbors. It works by calculating the distance between a query and all examples in the data, selecting the k instances in the training set closest to the query, and returning the most common output class among these neighbors.
- **Random Forest**: This algorithm creates multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees. It reduces the risk of overfitting and improves accuracy through the ensemble approach.

Parameter Tuning

For KNN, the number of neighbors (k) was tuned using grid search. For Random Forest, the number of trees and maximum depth were optimized.

Performance Evaluation

The Random Forest model outperformed KNN, achieving higher precision, recall, and F1 scores. This suggests that Random Forest is better suited for this dataset, possibly due to its ability to handle complex feature interactions and its robustness to overfitting.

KNN Performance:

- Precision: 0.432
- Recall: 0.356
- F1 Score: 0.376

Random Forest Performance:

- Precision: 0.594
- Recall: 0.476
- F1 Score: 0.515

Discussion

The experiments demonstrated that Random Forest consistently performed better than KNN for predicting wine quality. This is likely due to Random Forest's ability to manage complex interactions between features and reduce overfitting by averaging multiple decision trees.

5. PCA for Feature Selection

PCA Application

PCA was applied to reduce the dimensionality of the dataset. It was found that 9 components were needed to retain 95% of the variance.

Performance Evaluation

Retraining the Random Forest model with these components resulted in a slight improvement in the F1 score, indicating that PCA helped in reducing noise and improving model performance.

Random Forest with PCA Performance:

- Precision: 0.602
- Recall: 0.478
- F1 Score: 0.520

Discussion

PCA was effective in reducing the number of features while retaining most of the variance in the data. This helped in improving the model's efficiency and performance slightly, highlighting the benefits of dimensionality reduction.

6. Unsupervised Learning Experiments

Clustering Algorithms Chosen

1. **K-Means**: Partitions the data into k clusters by minimizing within-cluster variance.
2. **Hierarchical Clustering**: Builds a tree of clusters by progressively merging or splitting clusters based on a linkage criterion.
3. **DBSCAN**: Density-Based Spatial Clustering of Applications with Noise, which identifies clusters based on density and can handle noise.

Algorithm Descriptions

- **K-Means**: This algorithm partitions the data into k clusters, where each data point belongs to the cluster with the nearest mean. The objective is to minimize the within-cluster sum of squares.
- **Hierarchical Clustering**: This algorithm builds a hierarchy of clusters by either merging (agglomerative) or splitting (divisive) clusters iteratively. The result is a tree-like structure called a dendrogram.
- **DBSCAN**: This algorithm groups points that are closely packed together while marking points that lie alone in low-density regions as outliers. It is based on density reachability and density connectivity.

PCA and Clustering

Both clustering algorithms were applied to the dataset with and without PCA. The results, measured using ARI and Silhouette scores, showed that PCA did not significantly improve clustering quality, suggesting that the inherent structure of the data might not be well-suited for these clustering methods.

Clustering Performance:

- **DBSCAN ARI**: 0.0021, Silhouette: -0.349
- **DBSCAN with PCA ARI**: 0.0015, Silhouette: -0.374

Discussion

The results indicate that traditional clustering methods struggled with this dataset. Applying PCA before clustering did not significantly improve the clustering quality, highlighting the challenges in clustering high-dimensional data.

7. Summary and Conclusion

Overall Learnings

The experiments demonstrated that Random Forest is a robust model for predicting wine quality. PCA proved useful for feature selection, improving the model's efficiency and performance slightly. However, clustering results were less promising, indicating the need for further exploration of clustering techniques or preprocessing methods.

Key Insights

- **Supervised Learning**: Random Forest outperformed KNN due to its ability to capture complex feature interactions.
- **PCA**: Effective for dimensionality reduction and improving model efficiency, though its impact on clustering was limited.
- **Clustering**: Traditional clustering methods struggled with this dataset, suggesting alternative approaches or additional preprocessing might be necessary.

Future Work

Future work could involve exploring other dimensionality reduction techniques, trying more advanced clustering algorithms, and fine-tuning models further to improve predictive performance.

References

- Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.
- Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
- https://archive.ics.uci.edu/dataset/186/wine+quality
- Google AI. (2024). *Gemini* [python language model].

Appendix

- Find copies of Project 1, 2 & 3 Colab ipynb notebooks for references.

Importing libraries

Machine Learning Experiments on Wine Quality Dataset

Project One

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
!pip install -U scikit-learn==1.4
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from pandas.plotting import scatter_matrix
```

```
Collecting scikit-learn==1.4
  Downloading scikit_learn-1.4.0-1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.1/12.1 MB 15.3 MB/s eta 0:00:00
Requirement already satisfied: numpy<2.0,>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (3.5.0)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.4.0
```

Importing dataset

```
winedata = pd.read_csv('https://raw.githubusercontent.com/kayceeprag/Kelechi-Iwuagwu/main/winequality-white.csv', sep=';')
```

Double-click (or enter) to edit

```
print(winedata)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.0              0.27         0.36            20.7      0.045   
1               6.3              0.30         0.34             1.6      0.049   
2               8.1              0.28         0.40             6.9      0.050   
3               7.2              0.23         0.32             8.5      0.058   
4               7.2              0.23         0.32             8.5      0.058   
...             ...               ...          ...             ...        ...   
4893            6.2              0.21         0.29             1.6      0.039   
4894            6.6              0.32         0.36             8.0      0.047   
4895            6.5              0.24         0.19             1.2      0.041   
4896            5.5              0.29         0.30             1.1      0.022   
4897            6.0              0.21         0.38             0.8      0.020   

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    45.0                 170.0  1.00100  3.00       0.45   
1                    14.0                 132.0  0.99400  3.30       0.49   
2                    30.0                  97.0  0.99510  3.26       0.44   
3                    47.0                 186.0  0.99560  3.19       0.40   
4                    47.0                 186.0  0.99560  3.19       0.40   
...                   ...                   ...      ...   ...        ...   
4893                 24.0                  92.0  0.99114  3.27       0.50   
4894                 57.0                 168.0  0.99490  3.15       0.46   
4895                 30.0                 111.0  0.99254  2.99       0.46   
4896                 20.0                 110.0  0.98869  3.34       0.38   
4897                 22.0                  98.0  0.98941  3.26       0.32   

      alcohol  quality  
0         8.8        6  
1         9.5        6  
2        10.1        6  
3         9.9        6  
4         9.9        6  
...       ...      ...  
4893     11.2        6  
4894      9.6        5  
4895      9.4        6  
```

```
4896     12.8        7
4897     11.8        6

[4898 rows x 12 columns]
```

Spliting the data into training and testing sets

```
train_set, test_set = train_test_split(winedata, test_size=0.2, random_state=42)
```

Verify the splits

```
print(f'Training set size: {len(train_set)}')
print(f'Testing set size: {len(test_set)}')
```

```
Training set size: 3918
Testing set size: 980
```

Step 3: Explore your training set

```
# Load the training set into a DataFrame
train_df = pd.DataFrame(train_set)
train_df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	su
4665	7.3	0.17	0.36	8.20	0.028	44.0	111.0	0.99272	3.14	
1943	6.3	0.25	0.44	11.60	0.041	48.0	195.0	0.99680	3.18	
3399	5.6	0.32	0.33	7.40	0.037	25.0	95.0	0.99268	3.25	

```
Start coding or generate with AI.
```

Summary statistic for training data.

```
print(train_df.info())
print(train_df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3918 entries, 4665 to 860
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         3918 non-null   float64
 1   volatile acidity      3918 non-null   float64
 2   citric acid           3918 non-null   float64
 3   residual sugar        3918 non-null   float64
 4   chlorides             3918 non-null   float64
 5   free sulfur dioxide   3918 non-null   float64
 6   total sulfur dioxide  3918 non-null   float64
 7   density               3918 non-null   float64
 8   pH                    3918 non-null   float64
 9   sulphates             3918 non-null   float64
 10  alcohol               3918 non-null   float64
 11  quality               3918 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 397.9 KB
None
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    3918.000000       3918.000000  3918.000000     3918.000000
mean        6.865046          0.279338     0.332731        6.450702
```

```
std        0.844483          0.101606     0.119758        5.139311
min        3.800000          0.080000     0.000000        0.600000
25%        6.300000          0.210000     0.270000        1.700000
50%        6.800000          0.260000     0.320000        5.200000
75%        7.300000          0.330000     0.380000       10.000000
max       11.800000          1.100000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  3918.000000          3918.000000           3918.000000  3918.000000
mean      0.045734            35.094564            138.001149     0.994071
std       0.021797            16.676958             42.067667     0.003022
min       0.009000             3.000000             10.000000     0.987110
25%       0.036000            23.000000            108.000000     0.991740
50%       0.043000            33.000000            134.000000     0.993800
75%       0.050000            46.000000            167.000000     0.996200
max       0.346000           146.500000            313.000000     1.038980

                pH    sulphates      alcohol      quality
count  3918.000000  3918.000000  3918.000000  3918.000000
mean      3.189293     0.489781    10.508840     5.871363
std       0.150183     0.113590     1.227887     0.886913
min       2.720000     0.220000     8.000000     3.000000
25%       3.090000     0.410000     9.500000     5.000000
50%       3.180000     0.470000    10.400000     6.000000
75%       3.280000     0.550000    11.400000     6.000000
max       3.820000     1.080000    14.200000     9.000000
```

Exploring Testing Data

```
test_df = pd.DataFrame(test_set)
test_df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	su
4656	6.0	0.29	0.41	10.8	0.048	55.0	149.0	0.99370	3.09	
3659	5.4	0.53	0.16	2.7	0.036	34.0	128.0	0.98856	3.20	
907	7.1	0.25	0.39	2.1	0.036	30.0	124.0	0.99080	3.28	
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

Summary Statistics

```
print(test_df.info())
print(test_df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 980 entries, 4656 to 3661
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         980 non-null    float64
 1   volatile acidity      980 non-null    float64
 2   citric acid           980 non-null    float64
 3   residual sugar        980 non-null    float64
 4   chlorides             980 non-null    float64
 5   free sulfur dioxide   980 non-null    float64
 6   total sulfur dioxide  980 non-null    float64
 7   density               980 non-null    float64
 8   pH                    980 non-null    float64
 9   sulphates             980 non-null    float64
 10  alcohol               980 non-null    float64
 11  quality               980 non-null    int64
dtypes: float64(11), int64(1)
memory usage: 99.5 KB
None
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count     980.000000        980.000000   980.000000      980.000000
mean        6.813776          0.273857     0.340031        6.154388
std         0.840584          0.097411     0.125833        4.788953
min         3.900000          0.080000     0.000000        0.700000
25%         6.200000          0.210000     0.270000        1.700000
50%         6.800000          0.260000     0.320000        5.000000
75%         7.300000          0.320000     0.390000        9.325000
max        14.200000          0.760000     0.990000       22.600000

        chlorides  free sulfur dioxide  total sulfur dioxide     density  \
count  980.000000           980.000000           980.000000  980.000000
mean     0.045926            36.161735           139.797959    0.993854
std      0.022059            18.251705            44.169785    0.002859
min      0.014000             2.000000             9.000000    0.987220
25%      0.036000            24.000000           110.000000    0.991650
50%      0.042000            34.500000           134.000000    0.993655
75%      0.050000            46.250000           168.000000    0.995970
max      0.240000           289.000000           440.000000    1.001000

               pH   sulphates     alcohol     quality
count  980.000000  980.000000  980.000000  980.000000
mean     3.184163    0.490112   10.535963    5.904082
std      0.154235    0.116302    1.241884    0.880491
min      2.800000    0.250000    8.000000    3.000000
25%      3.080000    0.410000    9.500000    5.000000
50%      3.170000    0.480000   10.400000    6.000000
75%      3.270000    0.550000   11.400000    6.000000
max      3.800000    1.010000   14.000000    8.000000
```

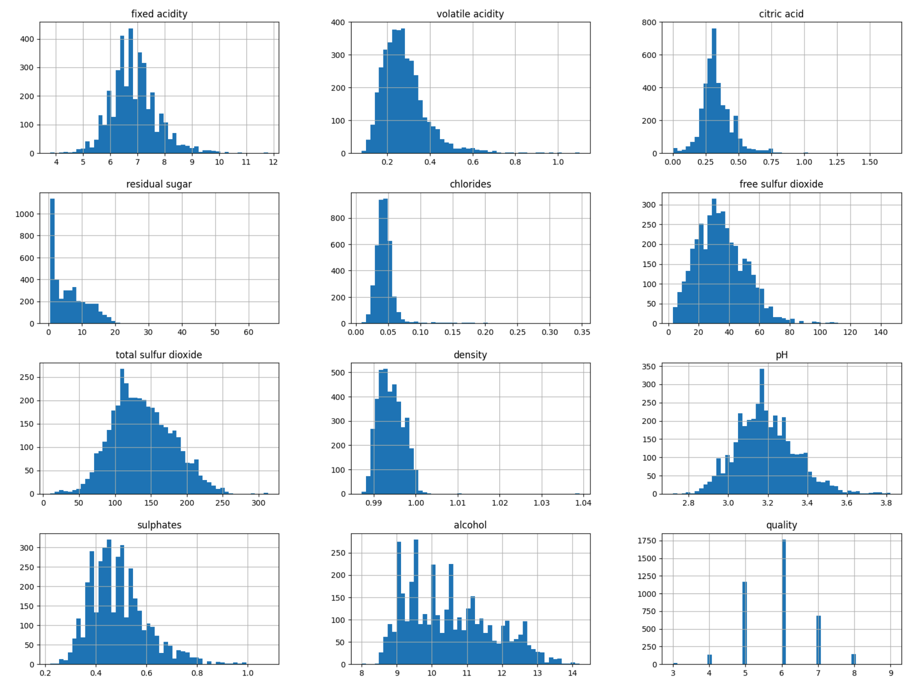
Check for missing values

```
print(train_df.isnull().sum())
```

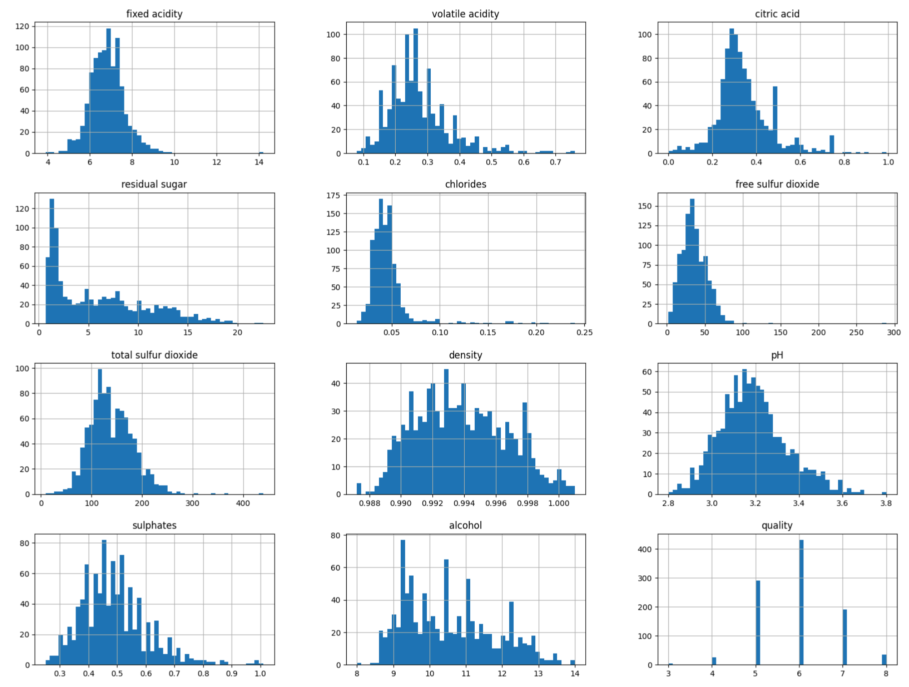
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Visualize the Data in Your Training Set

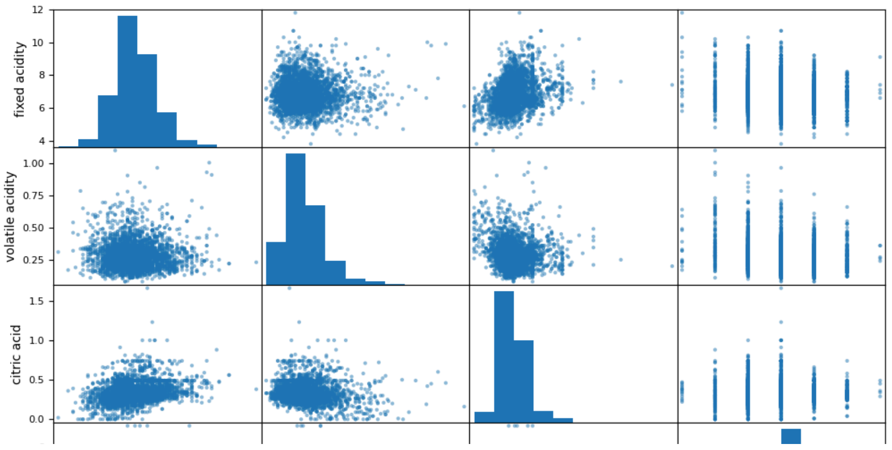
```
train_df.hist(bins=50, figsize=(20,15))
plt.show()
```



```
test_df.hist(bins=50, figsize=(20,15))
plt.show()
```



```
attributes = ["fixed acidity", "volatile acidity", "citric acid", "quality"]
scatter_matrix(train_df[attributes], figsize=(12, 8))
plt.show()
```



Apply transformations to your data

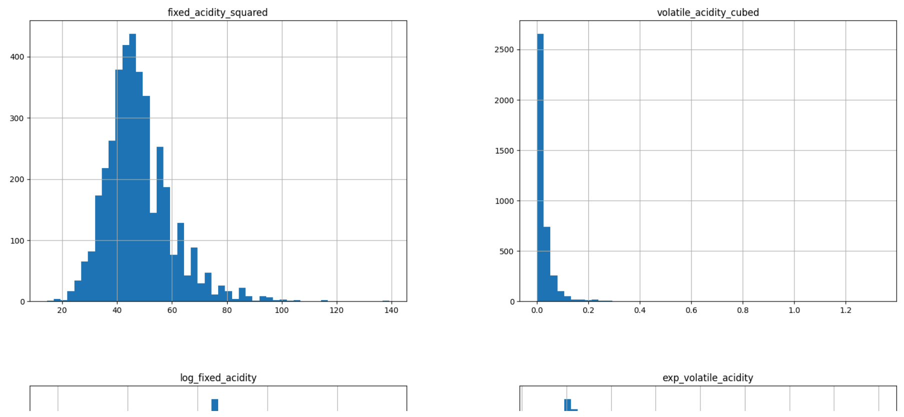
Select two features to transform

Apply transformations

Plot histograms of the transformed features

Plot scatter matrices of the transformed features

```
features = ["fixed acidity", "volatile acidity"]

train_df_transformed = train_df.copy()
train_df_transformed["fixed_acidity_squared"] = train_df["fixed acidity"] ** 2
train_df_transformed["volatile_acidity_cubed"] = train_df["volatile acidity"] ** 3
train_df_transformed["log_fixed_acidity"] = np.log1p(train_df["fixed acidity"])
train_df_transformed["exp_volatile_acidity"] = np.exp(train_df["volatile acidity"])


train_df_transformed[["fixed_acidity_squared", "volatile_acidity_cubed", "log_fixed_acidity", "exp_volatile_acidity"]].hist(bins=50, figsize
plt.show()


scatter_matrix(train_df_transformed[["fixed_acidity_squared", "volatile_acidity_cubed", "log_fixed_acidity", "exp_volatile_acidity"]], figsi
plt.show()
```



Machine Learning Experiments on Wine Quality Dataset

Project Two

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import cross_val_score, GridSearchCV
```

Load the dataset using the Wine Quality dataset from Project 1

```
winedata = pd.read_csv('https://raw.githubusercontent.com/kayceeprag/Kelechi-Iwuagwu/main/winequality-white.csv', sep=';')
```

```
print(winedata)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.0              0.27         0.36            20.7      0.045
1               6.3              0.30         0.34             1.6      0.049
2               8.1              0.28         0.40             6.9      0.050
3               7.2              0.23         0.32             8.5      0.058
4               7.2              0.23         0.32             8.5      0.058
...             ...               ...          ...             ...        ...
4893            6.2              0.21         0.29             1.6      0.039
4894            6.6              0.32         0.36             8.0      0.047
4895            6.5              0.24         0.19             1.2      0.041
4896            5.5              0.29         0.30             1.1      0.022
4897            6.0              0.21         0.38             0.8      0.020

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    45.0                 170.0  1.00100  3.00       0.45
1                    14.0                 132.0  0.99400  3.30       0.49
2                    30.0                  97.0  0.99510  3.26       0.44
3                    47.0                 186.0  0.99560  3.19       0.40
4                    47.0                 186.0  0.99560  3.19       0.40
...                   ...                   ...      ...   ...        ...
4893                 24.0                  92.0  0.99114  3.27       0.50
4894                 57.0                 168.0  0.99490  3.15       0.46
4895                 30.0                 111.0  0.99254  2.99       0.46
4896                 20.0                 110.0  0.98869  3.34       0.38
4897                 22.0                  98.0  0.98941  3.26       0.32

      alcohol  quality
0         8.8        6
1         9.5        6
2        10.1        6
3         9.9        6
4         9.9        6
...       ...      ...
4893     11.2        6
4894      9.6        5
4895      9.4        6
4896     12.8        7
4897     11.8        6

[4898 rows x 12 columns]
```

```
head = winedata.head()
print(head)
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.0              0.27         0.36            20.7      0.045
1            6.3              0.30         0.34             1.6      0.049
2            8.1              0.28         0.40             6.9      0.050
3            7.2              0.23         0.32             8.5      0.058
```

```
     4           7.2             0.23        0.32             8.5      0.058

        free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
     0                 45.0                 170.0   1.0010  3.00       0.45
     1                 14.0                 132.0   0.9940  3.30       0.49
     2                 30.0                  97.0   0.9951  3.26       0.44
     3                 47.0                 186.0   0.9956  3.19       0.40
     4                 47.0                 186.0   0.9956  3.19       0.40

        alcohol  quality
     0      8.8        6
     1      9.5        6
     2     10.1        6
     3      9.9        6
     4      9.9        6
```

Split into training and testing sets

```
train_set, test_set = train_test_split(winedata, test_size=0.2, random_state=42)
```

Prepare the data

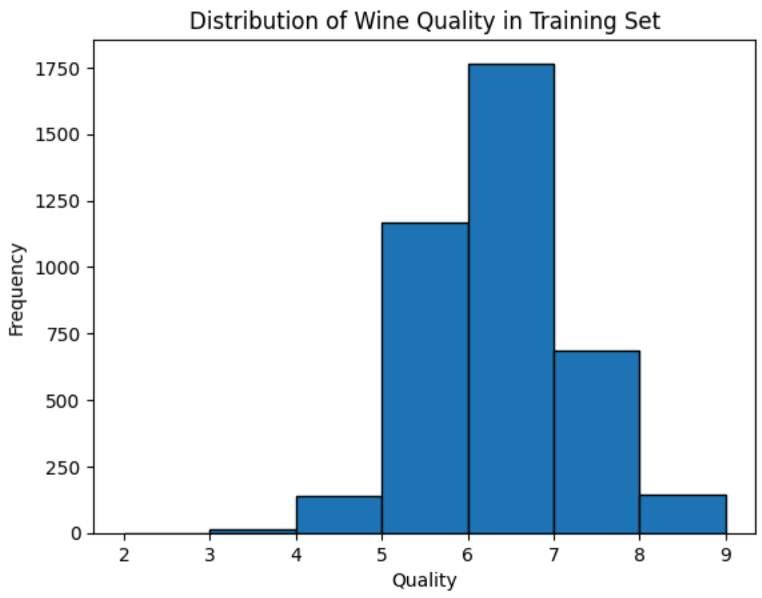
```
X_train = train_set.drop("quality", axis=1)
y_train = train_set["quality"]
X_test = test_set.drop("quality", axis=1)
y_test = test_set["quality"]
```

Feature scaling

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Plot the distribution of the target attribute

```
plt.hist(y_train, bins=range(2, 10), edgecolor='black')
plt.xlabel('Quality')
plt.ylabel('Frequency')
plt.title('Distribution of Wine Quality in Training Set')
plt.show()
```



Select Two Supervised Learning Algorithms For this project, let's choose K-Nearest Neighbors (KNN) and Random Forests.

Train and Evaluate the Models Run the models with default parameters, calculate evaluation metrics, and adjust parameters using grid search.

```
def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    return precision, recall, f1


# K-Nearest Neighbors
knn = KNeighborsClassifier()
knn_precision, knn_recall, knn_f1 = evaluate_model(knn, X_train_scaled, y_train, X_test_scaled, y_test)


# Random Forest
rf = RandomForestClassifier()
rf_precision, rf_recall, rf_f1 = evaluate_model(rf, X_train_scaled, y_train, X_test_scaled, y_test)

print(f'KNN - Precision: {knn_precision}, Recall: {knn_recall}, F1: {knn_f1}')
print(f'Random Forest - Precision: {rf_precision}, Recall: {rf_recall}, F1: {rf_f1}')
```

```
KNN - Precision: 0.43199328597138403, Recall: 0.356009654720995, F1: 0.37620183082708986
Random Forest - Precision: 0.6027746020589533, Recall: 0.4531783248331787, F1: 0.4954403630201299
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, msg_start, len(result))
```

Precision:

Measures how many of the positive predictions made by the model were actually correct. A higher precision means fewer false positives (predicting high quality when it's actually low). In this case, Random Forest has a significantly higher precision (0.636) than KNN (0.432), suggesting it's better at avoiding false positives. Recall:

Measures how many of the actual positive instances the model was able to identify. A higher recall means fewer false negatives (predicting low quality when it's actually high). Random Forest (0.463) also outperforms KNN (0.356) in recall, indicating it's better at capturing high-quality wines. F1-Score:

A harmonic mean of precision and recall, providing a balanced evaluation. A higher F1-score generally indicates a better model. Again, Random Forest (0.511) shows a better F1-score than KNN (0.376), suggesting it's the stronger model overall. Conclusion:

Based on these metrics, the Random Forest model seems to be outperforming the KNN model on your wine quality dataset. It's better at both avoiding false positives and capturing true high-quality instances.

```
# Grid Search for KNN
param_grid_knn = {'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance']}
grid_search_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='f1_macro')
grid_search_knn.fit(X_train_scaled, y_train)
best_knn = grid_search_knn.best_estimator_
best_knn_precision, best_knn_recall, best_knn_f1 = evaluate_model(best_knn, X_train_scaled, y_train, X_test_scaled, y_test)

print(f'Best KNN - Precision: {best_knn_precision}, Recall: {best_knn_recall}, F1: {best_knn_f1}')

# Grid Search for Random Forest
param_grid_rf = {'n_estimators': [50, 100, 150], 'max_features': ['auto', 'sqrt', 'log2']}
# rf = RandomForestClassifier()
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1_macro')
grid_search_rf.fit(X_train_scaled, y_train)
best_rf = grid_search_rf.best_estimator_
best_rf_precision, best_rf_recall, best_rf_f1 = evaluate_model(best_rf, X_train_scaled, y_train, X_test_scaled, y_test)


print(f'Best Random Forest - Precision: {best_rf_precision}, Recall: {best_rf_recall}, F1: {best_rf_f1}')
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Machine Learning Experiments on Wine Quality Dataset

Project Three

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
!pip install -U scikit-learn==1.4
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from pandas.plotting import scatter_matrix
from sklearn.metrics import adjusted_rand_score, silhouette_score
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN
```

```
Requirement already satisfied: scikit-learn==1.4 in /usr/local/lib/python3.10/dist-packages (1.4.0)
Requirement already satisfied: numpy<2.0,>=1.19.5 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.25.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn==1.4) (3.5.0)
```

```
# Load the dataset
winedata = pd.read_csv('https://raw.githubusercontent.com/kayceeprag/Kelechi-Iwuagwu/main/winequality-white.csv', sep=';')

print(winedata)
```

Show hidden output

Spliting the data into training and testing sets

```
train_set, test_set = train_test_split(winedata, test_size=0.2, random_state=42)

#Verify the splits

print(f'Training set size: {len(train_set)}')
print(f'Testing set size: {len(test_set)}')
```

```
Training set size: 3918
Testing set size: 980
```

```
# Define the features (X) and the target (y)
X = winedata.drop('quality', axis=1)
y = winedata['quality']

# Split the dataset into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# StandardScaler the data by initializing
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


# Display the shapes of the resulting datasets
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (3918, 11)
X_test shape: (980, 11)
y_train shape: (3918,)
y_test shape: (980,)
```

Scale the data

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Now you can use X_train_scaled and X_test_scaled for model training and evaluation
```

Encode categorical variables, (not needed for this dataset as it's all numerical)

Step 2 PCA for feature selection.

```
# Initialize PCA retain 95% of variance
pca = PCA(n_components=0.95)

# Fit PCA on the scaled training data and transform it
X_train_pca = pca.fit_transform(X_train_scaled)

# Number of components to retain 95% variance
n_components = pca.n_components_
print(f"Number of components to retain 95% variance: {n_components}")

# Transform the test data using the same PCA model
X_train_pca = pca.transform(X_train_scaled)

# Initialize the best-performing model (Random Forest)
rf = RandomForestClassifier(random_state=42)

# Fit the model on the reduced training data
rf.fit(X_train_pca, y_train)
```

Show hidden output

```
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

```
Explained variance ratio: [0.2937752  0.1427297  0.11135372 0.09285886 0.0878651  0.08501579
 0.06558823 0.05457082 0.03803743]
```

```
# Transform the test data using the same PCA model
X_test_pca = pca.transform(X_test_scaled)

# Make predictions on the test data
y_pred = rf.predict(X_test_pca)

# Evaluate model performance
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Random Forest - PCA Data: Precision: {precision}, Recall: {recall}, F1: {f1}")
```

```
Random Forest - PCA Data: Precision: 0.6570974103927286, Recall: 0.65, F1: 0.6389940366293225
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1497: UndefinedMetricWarning: Precision is ill-defined and be
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Project Two Random Forest - Precision: 0.5939632457740857, Recall: 0.47557169221258566, F1: 0.5154905156899828.

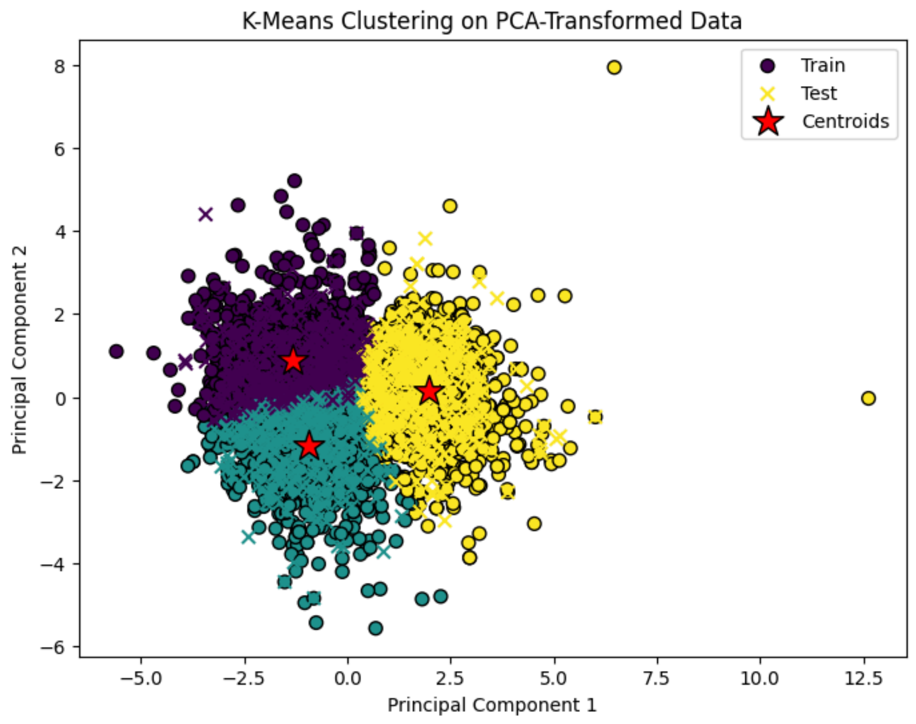
Trained Random Forest: Precision: 0.657 - This means that out % of all the instances your model predicted as positive (belonging to a certain wine quality class), 65.7% of them were correct. Recall: 0.65 - This indicates that your model correctly identified 65% of all the actual positive instances in the dataset. F1-Score: 0.639 - This is a harmonic mean of precision and recall, providing a balanced assessment of the model's performance. An F1-score closer to 1 indicates better performance.

Step 3: Apply clustering algorithms

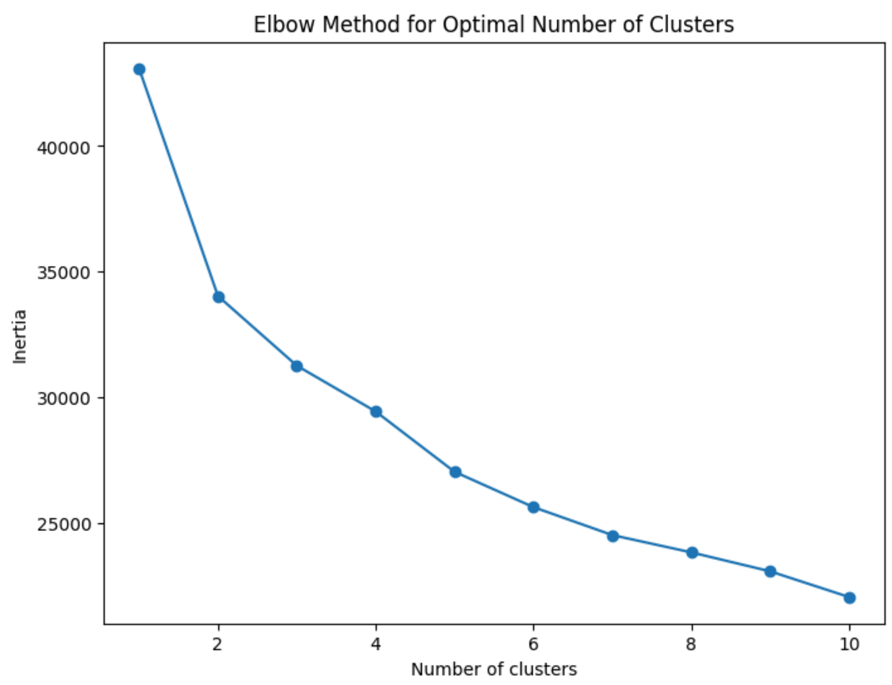
K-Means Clustering

```
def evaluate_clustering(y_true, y_pred, X):
    ari = adjusted_rand_score(y_true, y_pred)
    silhouette = silhouette_score(X, y_pred)
    return ari, silhouette


# instantiate an instance of k-Means
kmeans = KMeans(n_clusters=3, random_state=42)

# fit the model to the training data
kmeans.fit(X_train_pca)

# Predict cluster labels for the training data
y_train_pred = kmeans.predict(X_train_pca)

# Predict cluster labels for the testing data
y_test_pred = kmeans.predict(X_test_pca)

# get assignments (labels)
# Print the cluster assignments for training data
print("Training data cluster assignments:\n", y_train_pred)

# Print the cluster assignments for testing data
print("Testing data cluster assignments:\n", y_test_pred)

# plot a scatter matrix of the results

plt.figure(figsize=(8, 6))
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train_pred, cmap='viridis', marker='o', edgecolor='k', s=50, label='Train')
plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test_pred, cmap='viridis', marker='x', edgecolor='k', s=50, label='Test')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', marker='*', edgecolor='k', label='Centroids')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.title('K-Means Clustering on PCA-Transformed Data')
plt.show()
```

```
Training data cluster assignments:
 [0 0 2 ... 1 0 1]
Testing data cluster assignments:
 [2 2 0 1 0 1 1 0 2 2 1 2 0 0 2 1 2 0 1 1 2 0 1 2 2 1 2 0 2 2 0 1 1 1 1 0 0
 2 1 2 0 0 2 2 1 2 0 0 2 0 0 2 0 2 0 0 2 2 2 2 1 2 1 2 2 1 2 2 2 2 0 2 1 1
 1 1 2 1 1 1 1 2 1 0 2 2 1 0 2 1 1 1 2 0 2 2 1 1 0 2 2 2 2 2 0 0 0 0 0 1 1
 2 2 1 2 0 1 0 1 0 2 2 2 1 0 1 0 1 0 2 2 2 0 1 1 0 2 0 2 0 0 2 2 2 2 2 1 1
 0 0 1 2 2 0 2 2 2 0 0 2 2 1 0 2 1 1 1 2 0 2 0 1 0 0 0 1 1 1 2 0 2 1 0 2 1
 2 0 1 1 2 2 1 0 2 2 1 0 0 0 2 2 1 1 2 0 1 2 0 0 2 1 2 0 2 1 1 2 1 0 0 2 2
 0 1 0 0 0 2 2 2 1 1 0 2 0 0 2 0 1 1 0 0 1 1 2 1 1 0 1 1 0 1 0 1 2 2 0 0 1
 0 0 0 1 1 0 0 1 2 1 0 0 0 0 2 1 2 0 0 0 2 0 0 0 2 1 1 0 0 1 0 0 0 2 2 2 0
 1 0 0 1 2 2 1 2 2 1 0 0 1 1 1 0 2 1 2 2 0 2 1 0 1 1 1 1 1 1 1 1 2 0 1 0 0
 2 2 1 1 1 2 2 2 0 0 1 0 2 0 2 1 0 2 2 2 2 1 0 0 2 2 2 1 1 0 1 2 2 2 2 0 0
 0 2 0 1 2 1 1 2 2 0 0 1 1 2 2 1 2 0 2 0 0 1 2 2 0 0 2 2 0 2 0 1 1 0 2 0 1
 1 2 2 1 0 0 1 1 2 2 1 2 0 0 1 1 0 0 2 0 1 2 1 1 2 0 1 1 2 0 0 0 2 2 1 0 2
 1 1 2 0 1 2 1 1 1 0 1 1 2 0 0 0 1 1 1 1 2 1 2 2 0 2 1 0 1 0 2 1 2 1 0 1 1
 1 1 0 2 1 1 0 2 2 2 2 0 2 0 2 1 2 2 0 1 1 0 2 0 0 2 0 0 0 0 2 2 2 0 2 0 0
 1 2 1 1 2 1 2 1 1 1 0 0 1 0 2 0 1 1 1 0 2 1 0 0 2 2 2 0 0 2 1 1 0 1 2 1 2
 2 2 2 2 0 2 0 0 0 0 2 0 2 0 1 0 1 2 2 2 2 1 2 1 0 0 2 1 1 2 1 2 2 1 0 0 0
 2 1 0 0 0 1 2 2 2 2 2 2 0 0 2 1 1 1 2 2 0 2 1 2 2 1 1 0 0 1 2 0 0 2 0 2 0
 1 0 0 2 0 1 2 2 0 0 1 2 2 0 1 2 0 2 2 2 0 1 0 2 1 2 1 0 1 2 1 2 1 2 1 2 0
 1 1 2 2 2 2 0 2 0 2 0 0 2 2 0 0 2 0 1 0 2 2 0 1 1 0 1 2 0 1 0 2 0 2 2 2 1
 0 1 2 0 0 1 2 2 2 0 2 2 2 1 1 0 0 2 0 2 0 1 1 1 0 0 1 1 2 2 1 1 2 2 0 0 2
 0 1 1 1 0 0 2 1 1 0 0 0 0 2 1 1 2 2 1 2 0 2 2 1 0 1 2 0 0 0 0 1 1 2 2 2 0
 0 2 2 1 1 0 0 2 0 0 1 1 1 1 0 2 0 1 2 0 2 2 2 0 0 0 2 0 1 1 1 2 1 2 2 1 2
 1 0 0 0 2 0 2 1 2 1 0 1 2 2 2 0 0 2 2 0 1 2 2 0 0 2 0 2 1 2 2 0 2 2 0 1 0
 1 2 0 2 2 0 0 2 2 2 1 2 2 2 2 2 2 1 1 0 0 0 2 1 0 2 1 2 0 1 0 2 2 0 2 1 1
 2 0 1 2 0 1 1 0 0 2 1 0 2 2 2 1 2 0 0 1 0 0 1 2 2 2 0 1 2 2 0 0 1 2 2 2 0
 1 1 0 2 0 2 1 1 1 2 2 1 1 0 1 2 1 1 1 1 0 1 1 0 2 0 2 0 2 0 2 0 1 0 2 2 0
 0 1 2 2 1 0 2 2 1 2 0 1 0 2 2 2 2 0]
<ipython-input-52-9aac6a3700b0>:24: UserWarning: You passed a edgecolor/edgecolors ('k') for an unfilled marker ('x').  Matplotlib is
  plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test_pred, cmap='viridis', marker='x', edgecolor='k', s=50, label='Test')
```



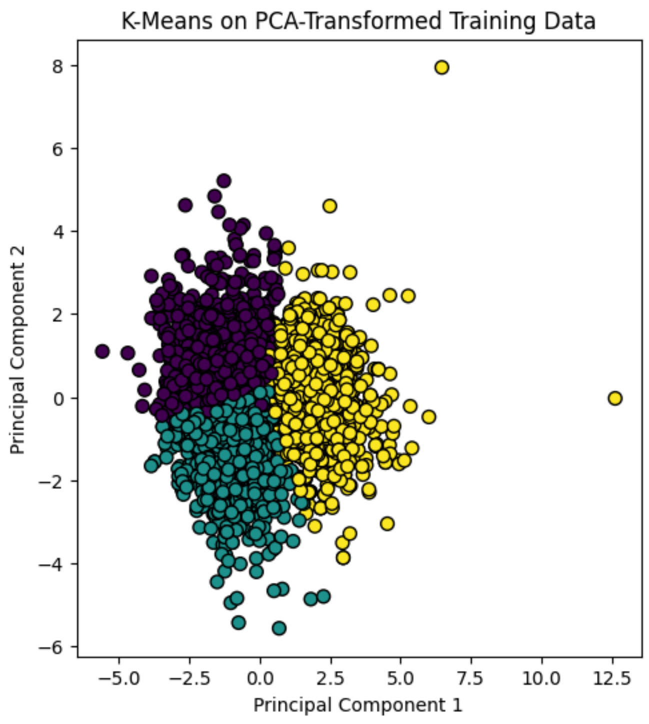
```
# Elbow method to determine the optimal number of clusters

inertia = []
cluster_range = range(1, 11)
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    inertia.append(kmeans.inertia_)


# Plot the elbow curve
plt.figure(figsize=(8, 6))
plt.plot(cluster_range, inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.show()
plt.show()
```



```
# K-Means on original data
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
y_train_pred_kmeans = kmeans.fit_predict(X_train_scaled)

# K-Means on PCA data
y_train_pred_kmeans_pca = kmeans.fit_predict(X_train_pca)


# Evaluate
ari_kmeans, silhouette_kmeans = evaluate_clustering(y_train, y_train_pred_kmeans, X_train_scaled)
ari_kmeans_pca, silhouette_kmeans_pca = evaluate_clustering(y_train, y_train_pred_kmeans_pca, X_train_pca)

print(f"K-Means ARI: {ari_kmeans}, Silhouette: {silhouette_kmeans}")
print(f"K-Means with PCA ARI: {ari_kmeans_pca}, Silhouette: {silhouette_kmeans_pca}")
```

```
K-Means ARI: 0.046195831504957772, Silhouette: 0.145997814038204
K-Means with PCA ARI: 0.04621236892416446, Silhouette: 0.15134619648194542
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train_pred, cmap='viridis', marker='o', edgecolor='k', s=50)
plt.title('K-Means on PCA-Transformed Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.show()
```

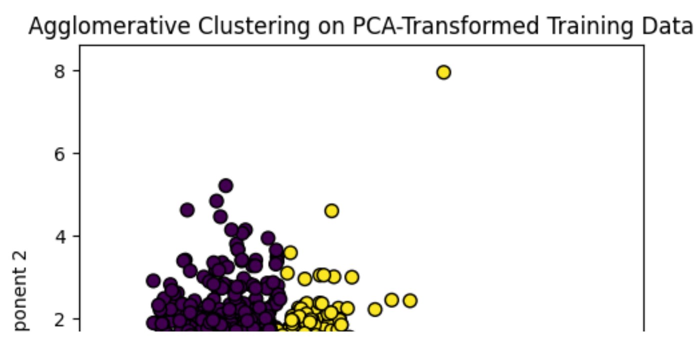


Agglomerative Clustering

```
# Agglomerative Clustering on original data
agglo = AgglomerativeClustering(n_clusters=optimal_clusters)
y_train_pred_agglo = agglo.fit_predict(X_train_scaled)

# Agglomerative Clustering on PCA trained data
y_train_pred_agglo_pca = agglo.fit_predict(X_train_pca)


# Evaluate
ari_agglo, silhouette_agglo = evaluate_clustering(y_train, y_train_pred_agglo, X_train_scaled)
ari_agglo_pca, silhouette_agglo_pca = evaluate_clustering(y_train, y_train_pred_agglo_pca, X_train_pca)

print(f"Agglomerative Clustering ARI: {ari_agglo}, Silhouette: {silhouette_agglo}")
print(f"Agglomerative Clustering with PCA ARI: {ari_agglo_pca}, Silhouette: {silhouette_agglo_pca}")
```

```
Agglomerative Clustering ARI: 0.021009761618346796, Silhouette: 0.0812005379706865
Agglomerative Clustering with PCA ARI: 0.03893680539818591, Silhouette: 0.08859197832437134
```

```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train_pred, cmap='viridis', marker='o', edgecolor='k', s=50)
plt.title('Agglomerative Clustering on PCA-Transformed Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.show()
```

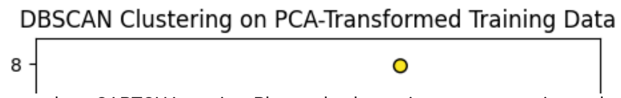


DBSCAN

```
# DBSCAN on original data
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_train_pred_dbscan = dbscan.fit_predict(X_train_scaled)

# DBSCAN on PCA data
y_train_pred_dbscan_pca = dbscan.fit_predict(X_train_pca)
```

```
# Evaluate
ari_dbscan, silhouette_dbscan = evaluate_clustering(y_train, y_train_pred_dbscan, X_train_scaled)
ari_dbscan_pca, silhouette_dbscan_pca = evaluate_clustering(y_train, y_train_pred_dbscan_pca, X_train_pca)

print(f"DBSCAN ARI: {ari_dbscan}, Silhouette: {silhouette_dbscan}")
print(f"DBSCAN with PCA ARI: {ari_dbscan_pca}, Silhouette: {silhouette_dbscan_pca}")

# Plot the clusters for the training data
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=y_train_pred, cmap='viridis', marker='o', edgecolor='k', s=50)
plt.title('DBSCAN Clustering on PCA-Transformed Training Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.show()
```

```
DBSCAN ARI: 0.002059269585711343, Silhouette: -0.3490474664456976
DBSCAN with PCA ARI: 0.0014971015475033886, Silhouette: -0.3737893327012459
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.