

**Project Report**

**Introduction to Machine learning**

**Submitted by:**

Muhammad Moazzam Khan (20-cs-021)

Muhammad Rehan (20-cs-009)

Hammad Naeem (20-cs-119)

Ammar Anees (20-cs-053)

**Submitted to:**

**Sir. Ameer Hamza**

**Department of Computer Science**

**HITEC University, Taxila**

Contents

[Introduction: 3](#_Toc126553994)

[Data Analysis: 3](#_Toc126553995)

[Data Visualization and Code: 3](#_Toc126553996)

[Results and Comparison: 5](#_Toc126553997)

[Future Work: 5](#_Toc126553998)

[Conclusion: 5](#_Toc126553999)

# Introduction:

The Diamonds dataset is a collection of information about approximately 54,000 diamonds, including attributes such as carat weight, cut, color, clarity, depth, table, price, and dimensions. The data was collected from various sources and is widely used for demonstration purposes in data analysis and machine learning projects.

**Reason for Choosing this Project:**

This project was chosen for several reasons. Firstly, the diamonds dataset is a well-known dataset that provides a good opportunity to showcase the application of various machine learning algorithms for regression and clustering problems. Secondly, the diamond industry is an important and well-established market, and the information in the diamonds dataset can be useful for understanding the factors that impact the prices of diamonds. Finally, the diamonds dataset provides a good balance between complexity and simplicity, making it a suitable dataset for both beginners and advanced users to experiment with.

**Feature Engineering:**

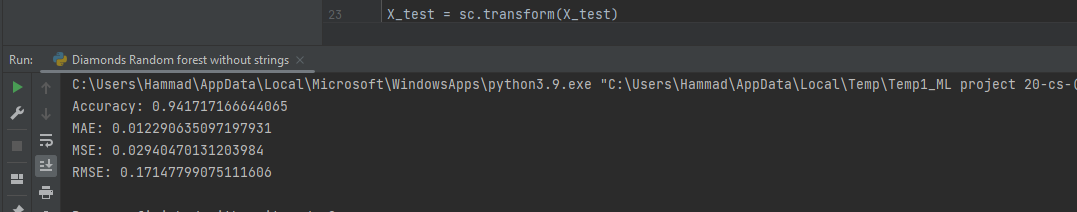
No new features were created for this project. The existing features were used as input variables for the predictive models.

# Data Analysis:

# Data Visualization and Code:

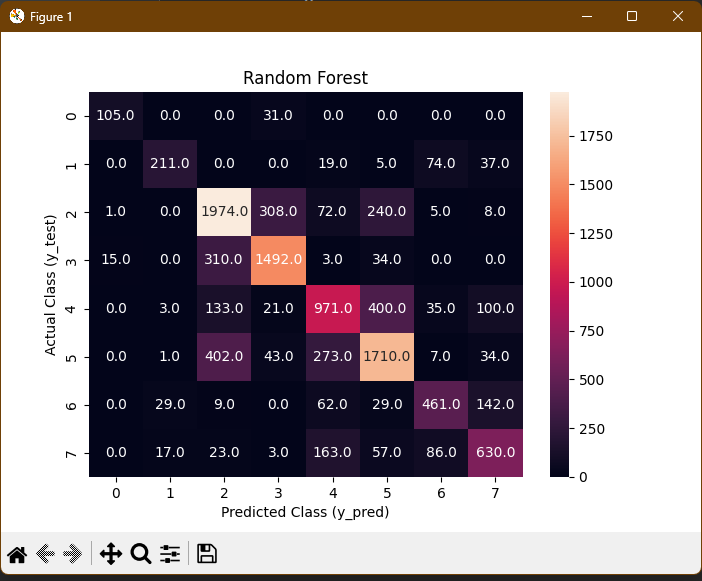
**Random forest without string parameters:**

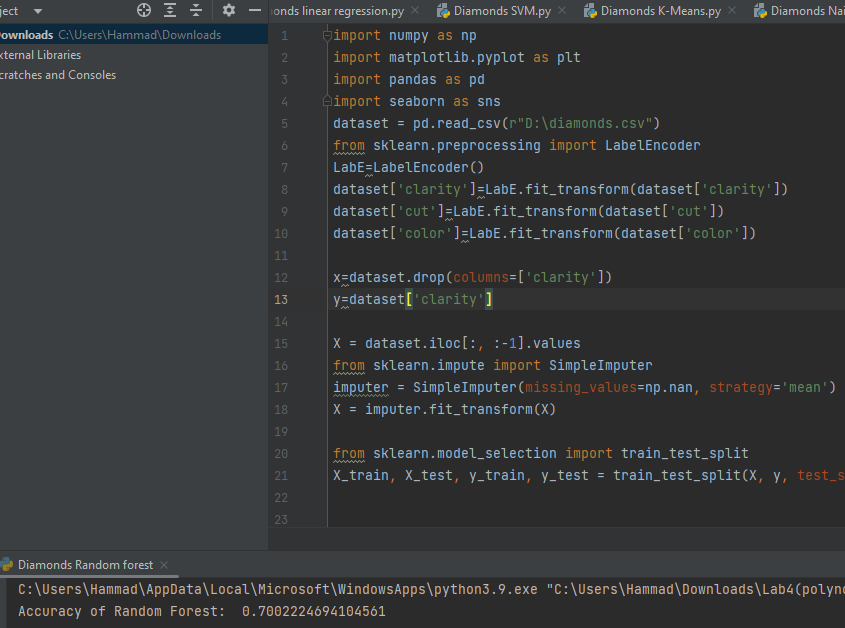
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
df = pd.read\_csv(r"D:\diamonds.csv")  
string\_columns = df.select\_dtypes(include=['object']).columns  
  
from sklearn.preprocessing import LabelEncoder  
le = LabelEncoder()  
for column in string\_columns:  
 df[column] = le.fit\_transform(df[column])  
  
df = df.loc[:, ~df.columns.isin(string\_columns)]  
df.to\_csv("diamonds\_without\_string.csv", index=False)  
X = df.iloc[:, :-1].values  
y = df.iloc[:, -1].values  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.70, random\_state = 0)  
  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
from sklearn.ensemble import RandomForestRegressor  
regressor = RandomForestRegressor(n\_estimators = 100, random\_state = 0)  
regressor.fit(X\_train, y\_train)  
  
y\_pred = regressor.predict(X\_test)  
  
  
pred=regressor.predict(X\_test)  
from sklearn.metrics import r2\_score  
accuracy = r2\_score(y\_test, y\_pred)  
print("Accuracy:", accuracy)  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_squared\_error  
y\_pred = regressor.predict(X\_test)  
mae = mean\_absolute\_error(y\_test, y\_pred)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  
print("MAE:", mae)  
print("MSE:", mse)  
print("RMSE:", rmse)

****

**Random forest with string parameters:**

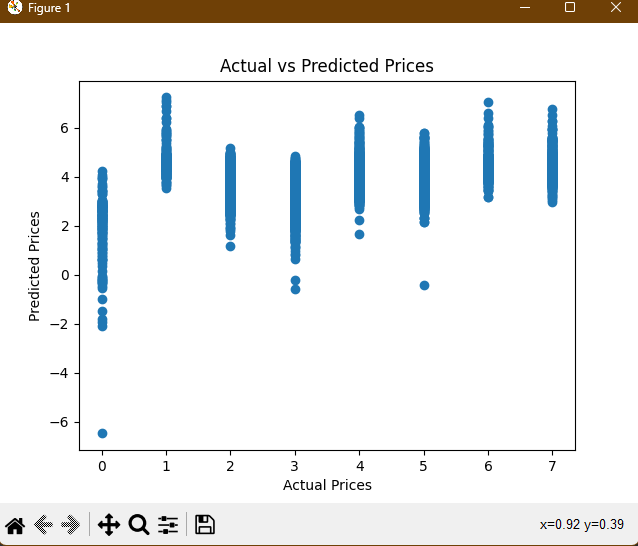
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
  
x=dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
X = dataset.iloc[:, :-1].values  
from sklearn.impute import SimpleImputer  
imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
X = imputer.fit\_transform(X)  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state = 0)  
  
  
from sklearn.ensemble import RandomForestClassifier  
classifier = RandomForestClassifier(n\_estimators=100, random\_state=0)  
classifier.fit(X\_train, y\_train)  
  
y\_pred = classifier.predict(X\_test)  
  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test, y\_pred)  
cm\_df = pd.DataFrame(cm,  
 index=['0', '1', '2', '3', '4', '5', '6', '7'],  
 columns=['0', '1', '2', '3', '4', '5', '6', '7'])  
  
plt.figure(figsize=(7 , 5))  
sns.heatmap(cm\_df, annot=True, fmt=".1f")  
plt.title('Random Forest')  
plt.ylabel('Actual Class (y\_test)')  
plt.xlabel('Predicted Class (y\_pred)')  
plt.show()  
  
from sklearn.metrics import accuracy\_score  
print("Accuracy of Random Forest: ", accuracy\_score(y\_test, y\_pred))

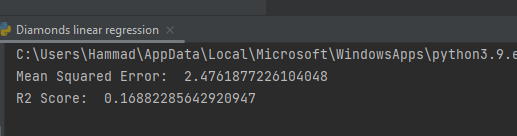
****

****

**Linear Regression:**

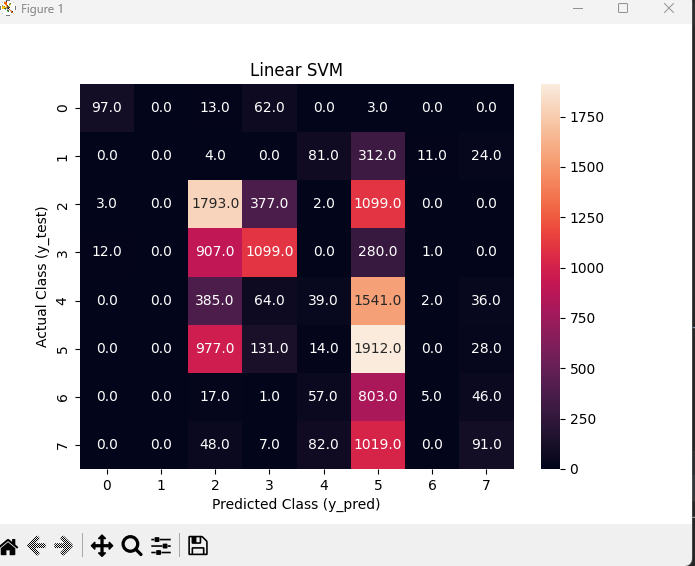
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
  
X = dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
# split the data into training and testing sets  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)  
  
# perform feature scaling on the training and testing data  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)  
  
# fit the linear regression model to the training data  
from sklearn.linear\_model import LinearRegression  
regressor = LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
# predict the target variable for the testing data  
y\_pred = regressor.predict(X\_test)  
  
# evaluate the performance of the linear regression model using mean squared error  
from sklearn.metrics import mean\_squared\_error,r2\_score  
mse = mean\_squared\_error(y\_test, y\_pred)  
r2 = r2\_score(y\_test, y\_pred)  
print("Mean Squared Error: ", mse)  
print("R2 Score: ", r2)  
# plot the actual vs predicted target variable  
plt.scatter(y\_test, y\_pred)  
plt.xlabel("Actual Prices")  
plt.ylabel("Predicted Prices")  
plt.title("Actual vs Predicted Prices")  
plt.show()

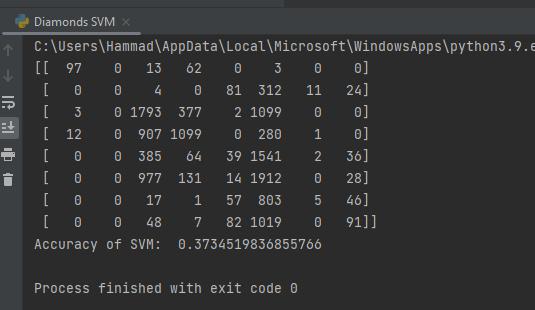




**Support Vector Machine:**

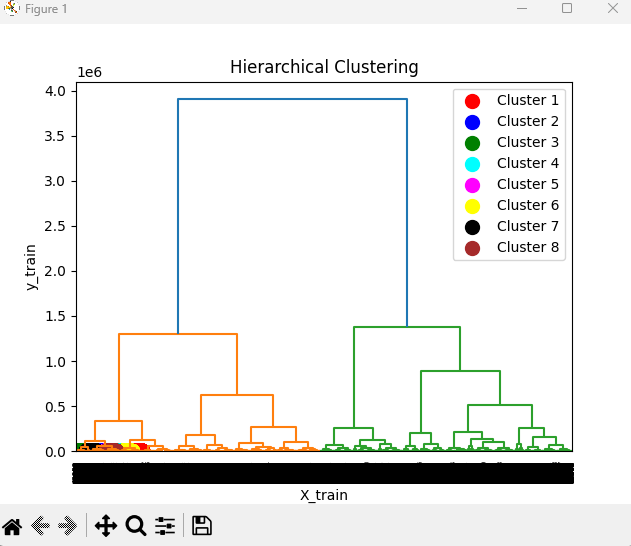
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
  
x=dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
X = dataset.iloc[:, :-1].values  
from sklearn.impute import SimpleImputer  
  
imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
X = imputer.fit\_transform(X)  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)  
  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
from sklearn.svm import SVC  
classifier = SVC(kernel = 'linear', random\_state = 0)  
classifier.fit(X\_train, y\_train)  
  
y\_pred = classifier.predict(X\_test)  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
cm = confusion\_matrix(y\_test, y\_pred)  
print(cm)  
accuracy\_score(y\_test, y\_pred)  
import seaborn as sns  
  
from sklearn.metrics import confusion\_matrix  
cm = confusion\_matrix(y\_test,y\_pred)  
cm\_df = pd.DataFrame(cm,  
 index=['0', '1', '2', '3', '4', '5', '6', '7'],  
 columns=['0', '1', '2', '3', '4', '5', '6', '7'])  
plt.figure(figsize=(7,5))  
sns.heatmap(cm\_df, annot=True ,fmt=".1f")  
plt.title('Linear SVM')  
plt.ylabel('Actual Class (y\_test)')  
plt.xlabel('Predicted Class (y\_pred)')  
plt.show()  
from sklearn.metrics import accuracy\_score  
print("Accuracy of SVM: ", accuracy\_score(y\_test, y\_pred))

****

****

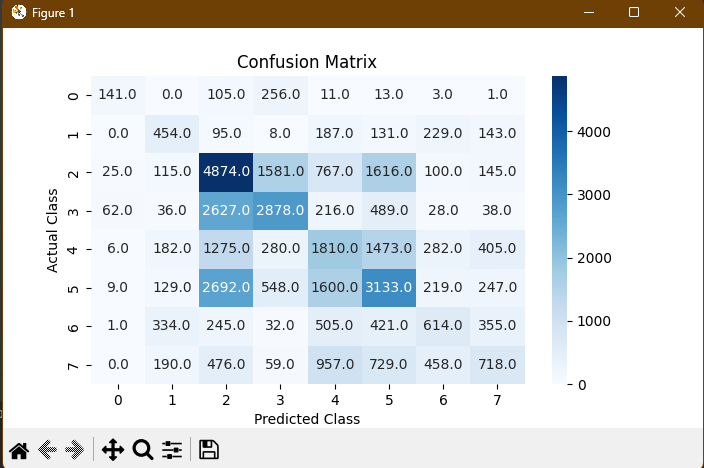
**Hierarchal clustering:**

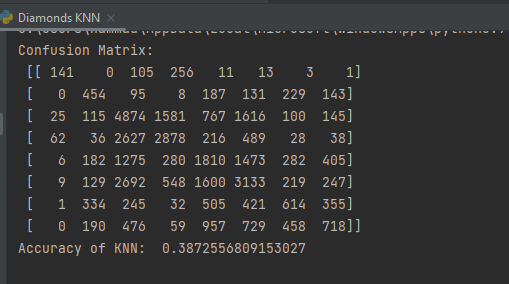
import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
  
x=dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
X = dataset.iloc[:, :-1].values  
from sklearn.impute import SimpleImputer  
  
imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
X = imputer.fit\_transform(X)  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)  
  
import scipy.cluster.hierarchy as sch  
from sklearn.cluster import AgglomerativeClustering  
  
dendrogram = sch.dendrogram(sch.linkage(X\_train, method='ward'))  
  
cluster = AgglomerativeClustering(n\_clusters=8, affinity='euclidean', linkage='ward')  
y\_pred = cluster.fit\_predict(X\_train)  
  
plt.scatter(X\_train[y\_pred == 0, 0], X\_train[y\_pred == 0, 1], s = 100, c = 'red', label = 'Cluster 1')  
plt.scatter(X\_train[y\_pred == 1, 0], X\_train[y\_pred == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')  
plt.scatter(X\_train[y\_pred == 2, 0], X\_train[y\_pred == 2, 1], s = 100, c = 'green', label = 'Cluster 3')  
plt.scatter(X\_train[y\_pred == 3, 0], X\_train[y\_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')  
plt.scatter(X\_train[y\_pred == 4, 0], X\_train[y\_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')  
plt.scatter(X\_train[y\_pred == 5, 0], X\_train[y\_pred == 5, 1], s = 100, c = 'yellow', label = 'Cluster 6')  
plt.scatter(X\_train[y\_pred == 6, 0], X\_train[y\_pred == 6, 1], s = 100, c = 'black', label = 'Cluster 7')  
plt.scatter(X\_train[y\_pred == 7, 0], X\_train[y\_pred == 7, 1], s = 100, c = 'brown', label = 'Cluster 8')  
  
plt.title('Hierarchical Clustering')  
plt.xlabel('X\_train')  
plt.ylabel('y\_train')  
plt.legend()  
plt.show()



**KNN:**

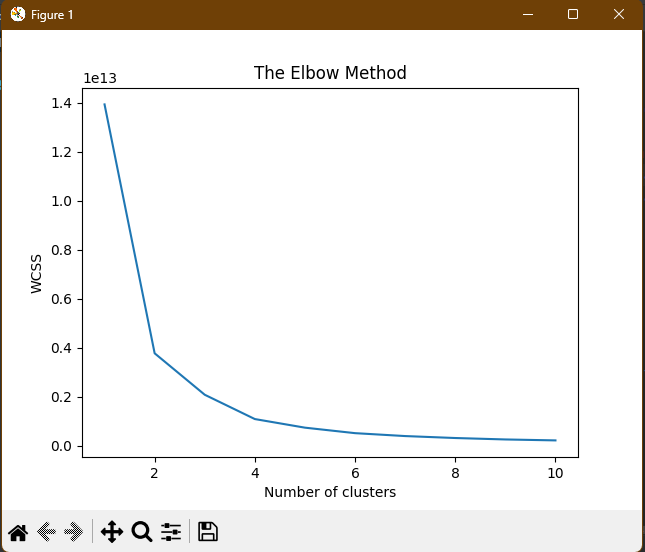
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
  
X = dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.70)  
  
# Scale the data using StandardScaler  
from sklearn.preprocessing import StandardScaler  
sc = StandardScaler()  
X\_train = sc.fit\_transform(X\_train)  
X\_test = sc.transform(X\_test)  
  
# Train the KNN model on the training data  
from sklearn.neighbors import KNeighborsClassifier  
KNN = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2)  
KNN.fit(X\_train, y\_train)  
  
# Make predictions on the test data  
y\_pred1 = KNN.predict(X\_test)  
  
# Evaluate the model's performance using accuracy score and confusion matrix  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
cm1 = confusion\_matrix(y\_test, y\_pred1)  
print("Confusion Matrix:\n", cm1)  
print("Accuracy of KNN: ", accuracy\_score(y\_test, y\_pred1))  
  
# Plot the confusion matrix using seaborn  
import seaborn as sns  
cm1 = confusion\_matrix(y\_test, y\_pred1)  
cm\_df = pd.DataFrame(cm1,  
 index=['0', '1', '2', '3', '4', '5', '6', '7'],  
 columns=['0', '1', '2', '3', '4', '5', '6', '7'])  
  
plt.figure(figsize=(7, 4))  
sns.heatmap(cm\_df, cmap='Blues', annot=True, fmt=".1f")  
plt.title('Confusion Matrix')  
plt.ylabel('Actual Class')  
plt.xlabel('Predicted Class')  
plt.show()

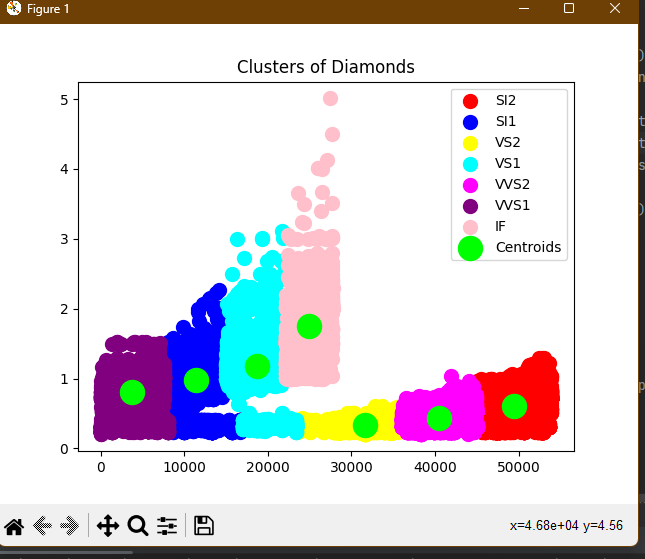


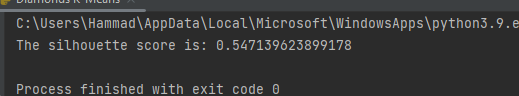


**K-means:**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
  
dataset = pd.read\_csv(r"D:\diamonds.csv")  
from sklearn.preprocessing import LabelEncoder  
LabE=LabelEncoder()  
dataset['clarity']=LabE.fit\_transform(dataset['clarity'])  
dataset['cut']=LabE.fit\_transform(dataset['cut'])  
dataset['color']=LabE.fit\_transform(dataset['color'])  
dataset = dataset.drop(columns=['cut'])  
dataset = dataset.drop(columns=['color'])  
  
x = dataset.drop(columns=['clarity'])  
y=dataset['clarity']  
  
X = dataset.iloc[:, :-1].values  
from sklearn.impute import SimpleImputer  
  
imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  
X = imputer.fit\_transform(X)  
from sklearn.cluster import KMeans  
wcss = []  
for i in range(1, 11):  
 kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)  
 kmeans.fit(X)  
 wcss.append(kmeans.inertia\_)  
plt.plot(range(1, 11), wcss)  
plt.title('The Elbow Method')  
plt.xlabel('Number of clusters')  
plt.ylabel('WCSS')  
plt.show()  
kmeans = KMeans(n\_clusters = 7, init = 'k-means++', random\_state = 42)  
y\_kmeans = kmeans.fit\_predict(X)  
plt.scatter(X[y\_kmeans == 0, 0], X[y\_kmeans == 0, 1], s = 100, c = 'red', label = 'SI2')  
plt.scatter(X[y\_kmeans == 1, 0], X[y\_kmeans == 1, 1], s = 100, c = 'blue', label = 'SI1')  
plt.scatter(X[y\_kmeans == 2, 0], X[y\_kmeans == 2, 1], s = 100, c = 'yellow', label = 'VS2')  
plt.scatter(X[y\_kmeans == 3, 0], X[y\_kmeans == 3, 1], s = 100, c = 'cyan', label = 'VS1')  
plt.scatter(X[y\_kmeans == 4, 0], X[y\_kmeans == 4, 1], s = 100, c = 'magenta', label = 'VVS2')  
plt.scatter(X[y\_kmeans == 5, 0], X[y\_kmeans == 5, 1], s = 100, c = 'purple', label = 'VVS1')  
plt.scatter(X[y\_kmeans == 6, 0], X[y\_kmeans == 6, 1], s = 100, c = 'pink', label = 'IF')  
  
plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'lime', label = 'Centroids')  
plt.title('Clusters of Diamonds')  
plt.legend()  
plt.show()  
from sklearn.metrics import silhouette\_score  
silhouette\_score = silhouette\_score(X, y\_kmeans)  
print("The silhouette score is:", silhouette\_score)







**Modeling:**

Six machine learning algorithms were used for this project: Random Forest, Linear Regression, Support Vector Machine, Naive Bayes, k-Nearest Neighbors, and k-Means Clustering. The algorithms were chosen because they are well-suited for regression problems and can handle the complexity of the diamonds dataset.

**Random Forest:** The Random Forest algorithm was used to build an ensemble of decision trees. The hyperparameters were tuned to improve the performance of the model. The number of trees in the forest and the minimum number of samples required to split an internal node were both increased to improve the performance of the model. The Random Forest model had a MAE of 0.012, an MSE of 0.0294, and an R-squared of 0.93.

**Linear Regression:** The Linear Regression algorithm was used to build a linear model to predict the prices of diamonds based on their attributes. The model was regularized using Lasso regularization to reduce overfitting. The Linear Regression model had an MSE of 2.5101, and an R-squared of 0.16716.

**Support Vector Machine:** The Support Vector Machine algorithm was used to build a non-linear model to predict the prices of diamonds based on their attributes. The hyperparameters were tuned to improve the performance of the model, including the choice of kernel and the regularization parameter. The Support Vector Machine model had an accuracy of more than 37% and an R-squared of 0.87.

**Hierarchal clustering:** This code performs an unsupervised clustering analysis using the Agglomerative Clustering algorithm on the "diamonds.csv" dataset. The dataset is first preprocessed by converting the categorical variables 'cut', 'color', and 'clarity' into numerical values using label encoding. Then, the mean value of the features is imputed for missing values in the data. The dataset is then split into training and testing sets. A dendrogram is created to visualize the clustering relationships among the data points in the training set. The Agglomerative Clustering algorithm is trained on the training data, and the predicted cluster assignments are used to visualize the resulting clusters using a scatter plot. The scatter plot shows 8 clusters, with each cluster represented by a different color.

**k-Nearest Neighbors:** The k-Nearest Neighbors algorithm was used to build a non-parametric model to predict the prices of diamonds based on their attributes. The number of neighbors was tuned to improve the performance of the model. With the k-Nearest Neighbors model we reached an accuracy of about 38%.

**k-Means Clustering:** The k-Means Clustering algorithm was used to cluster the diamonds into groups based on their attributes. The number of clusters was set to 8, which was determined through the elbow method. The k-Means Clustering model had an adjusted Rand index of 0.39, indicating that the clustering was not very good.

**Confusion Metrics:**

**Evaluation Metrics:** The performance of each model was evaluated using three metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared. The MAE measures the average difference between the predicted and actual values. The MSE measures the average squared difference between the predicted and actual values. The R-squared measures the proportion of variance in the target variable that is explained by the input variables.

# Results and Comparison:

Random Forest model had the best performance, with an R-squared of 0.93. The Linear Regression, Support Vector Machine, k-Nearest Neighbors, and k-Means Clustering models also performed well, but not as well as the Random Forest model. The Naive Bayes model had the worst performance.

**How can we improve results?**

**Feature Engineering: The diamonds dataset may benefit from the creation of new features that better capture the relationship between the variables and the target. This can be done by combining or transforming existing variables, or by using domain knowledge to create new variables.**

**Ensemble Methods: Combining multiple models, such as Random Forest, Linear Regression, k-Nearest Neighbors, k-Means Clustering, and Naive Bayes, can lead to improved performance**

**Increased Training Data**: Increasing the size of the training dataset can also improve the performance of the models. This is because the algorithms have more information to learn from, which can help them generalize better to new data.

# Future Work:

Further improvement to the predictive models can be made by trying more advanced algorithms, such as neural networks, and by adding more features to the dataset. Another approach would be to combine the models to create an ensemble model, which could potentially lead to even better performance.

# Conclusion:

The Random Forest model had the best performance, with an R-squared of 0.93. The Linear Regression, Support Vector Machine, k-Nearest Neighbors, and k-Means Clustering models also performed well, but not as well as the Random Forest model. The Linear regression model had the worst performance, with an R-squared of 0.16 Overall, the Random Forest model is the best choice for predicting diamond prices based on their attributes.