

**INF2008-Machine Learning**

Diamond Cut Classifier

User Manual

| **LAB-P3 - 2** | |
| --- | --- |
| GABRIEL LEE JUN RONG | 2301906 |
| QUAH YONG YAO | 2302009 |
| SUDIPTA KANTI BISWAS | 2303435 |

[**Project Overview**](#_heading=h.72byc685os3j) **3**

[**Project Structure**](#_heading=h.8hhkz3uvi7zz) **3**

[**Environment Setup & Required Packages  
Python Version and Dependencies**](#_heading=h.hwop8meyfker) **4**

[**Diamonds.csv Dataset**](#_heading=h.ko6uqavx457x) **4**

[**DiamondTrainer Notebook**](#_heading=h.hh0ipvbyuk9p) **5**

[Initial Setup](#_heading=h.x8rh7oxs78s9) 5

[Feature Engineering](#_heading=h.k1olg39nb49a) 6

[Model Selection and Training](#_heading=h.5x9j7alyegkv) 9

[Reason for chosen models:](#_heading=h.94oy9g7pb1ka) 9

[Training parameter setup:](#_heading=h.6zjm7hqv0izu) 9

[Model 1: RandomForestClassifier](#_heading=h.6hqys3m581ur) 10

[Model 2: KNeighborsClassifier](#_heading=h.o7j779id8poq) 10

[Model 3: XGBoost](#_heading=h.1mcmchaw41lx) 11

[Model 4: Gradient Boosting Machines](#_heading=h.wc0r5n9geh40) 11

[Model 5: Logistic Regression](#_heading=h.x3jf4sg61lxd) 12

[Evaluations](#_heading=h.b3bqq6kylsfz) 13

[Predictions of new data](#_heading=h.kwh74lifrn1) 13

[Fine tuning](#_heading=h.wdtyhbbyzgrf) 13

[Saving the Best Model](#_heading=h.y44fdh6oqhht) 13

[**DiamondClassifier Notebook**](#_heading=h.xnqa9kzgc8la) **13**

[Loading and running the saved model](#_heading=h.xmm68d8rpxqu) 13

# 

# Project Overview

The Diamond Cut Classifier is a machine learning project designed to predict the cut quality of a diamond based on its physical characteristics. The classifier categorizes diamonds into five quality levels: Fair, Good, Very Good, Premium, and Ideal.

This project uses a stacked ensemble model that combines multiple classification algorithms to provide accurate predictions. The trained model is saved as a pickle file **(DiamondCutClassifier v1-6.pkl**) which can be loaded and used for making predictions on new diamond data.

**System Requirements**

* Minimum 4GB of RAM
* Internet connection needed for initial setup to install packages

# Project Structure

* **diamonds.csv**Dataset with diamond attributes used for training and testing.
* **DiamondTrainer.ipynb**Notebook covering data preprocessing, training, and evaluating the machine learning model.
* **V2 DiamondTrainer.ipynb**An updated version of the training notebook that may include additional improvements or techniques.
* **DiamondClassifier.ipynb**Notebook that demonstrates loading the pre-trained model and making predictions.
* **DiamondCutClassifier v1-6.pkl**The serialized (pickled) machine learning model trained on the dataset, used for making predictions.

# 

# Environment Setup & Required Packages Python Version and Dependencies

* **Python Version:**  
  It is recommended to use **Python 3.9 and higher**.
* **Required Python Packages and Suggested Versions:**
  + numpy (>=1.18)
  + pandas (>=1.0.0)
  + scikit-learn (>=0.24)
  + matplotlib (>=3.3)
  + jupyter (for running notebooks)
  + pickle (standard library module)

# Diamonds.csv Dataset

The dataset (diamonds.csv) contains the following features:

* Carat: Weight of the diamond.
* Cut: Quality of the diamond cut (target variable for classification).
* Color: Color grade of the diamond.
* Clarity: Clarity grade.
* Price: Price of the diamond.
* x, y, z values: Size/dimensions of the diamond

The dataset is used to train the classifier to predict the quality of a diamond's cut based on these features.

# DiamondTrainer Notebook

- This is the notebook covering data preprocessing, training, and evaluating various models. It eventually shows weighted XGBoost as the best model and saves it as “DiamondCutClassifier v1-6.pkl”, which can be used for future predictions

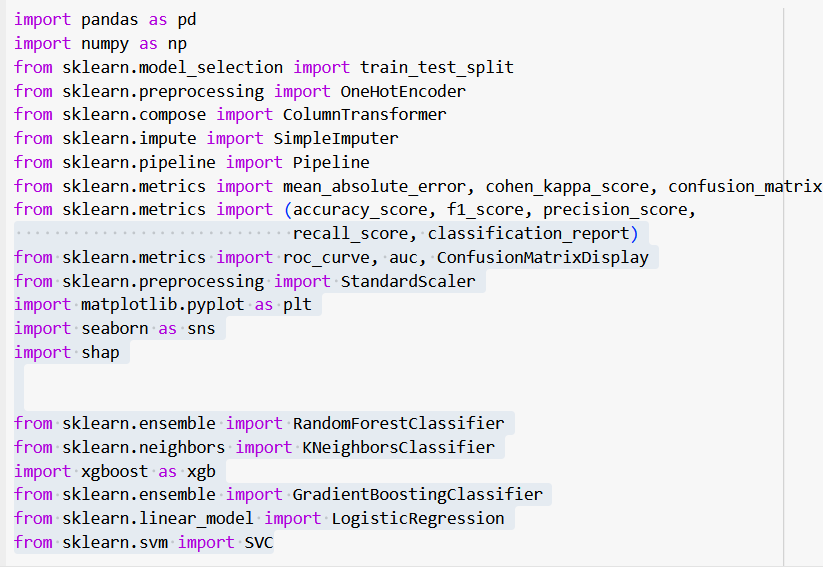
## Initial Setup

1. Install the dependencies mentioned above using pip

Example command, dependency versions may vary:

!pip install pandas==2.2.2 numpy==1.26.4 scikit-learn==1.2.2 xgboost==2.0.3 matplotlib==3.7.5 seaborn==0.12.2 shap==0.44.1 scikit-optimize==0.10.2

2. Import the libraries



3. Load your dataset and clean

dataset\_path = 'diamonds.csv'

df = pd.read\_csv(dataset\_path)

# Data cleaning

df = df.drop('Unnamed: 0', axis=1)

# Handle zero values in dimensions

df[['x', 'y', 'z']] = df[['x', 'y', 'z']].replace(0, np.nan)

df = df.dropna()

## Feature Engineering

The diamond dataset contains a categorical feature 'cut' with different quality levels. To prepare this feature, we need to convert these categorical labels into a numerical format. (0-4). We define a hierarchical order of cut quality from lowest to highest:

0 = Fair (lowest quality)

1 = Good

2 = Very Good

3 = Premium

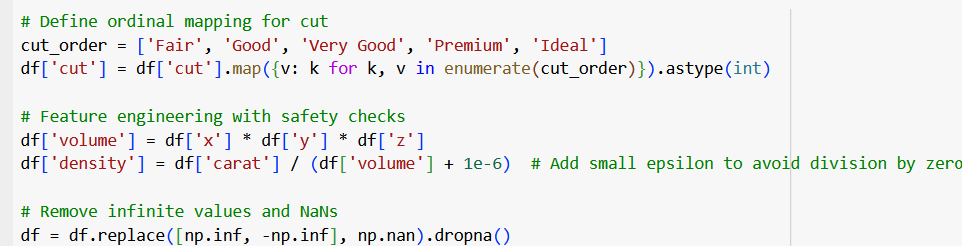
4 = Ideal (highest quality)

**Volume Calculation:**

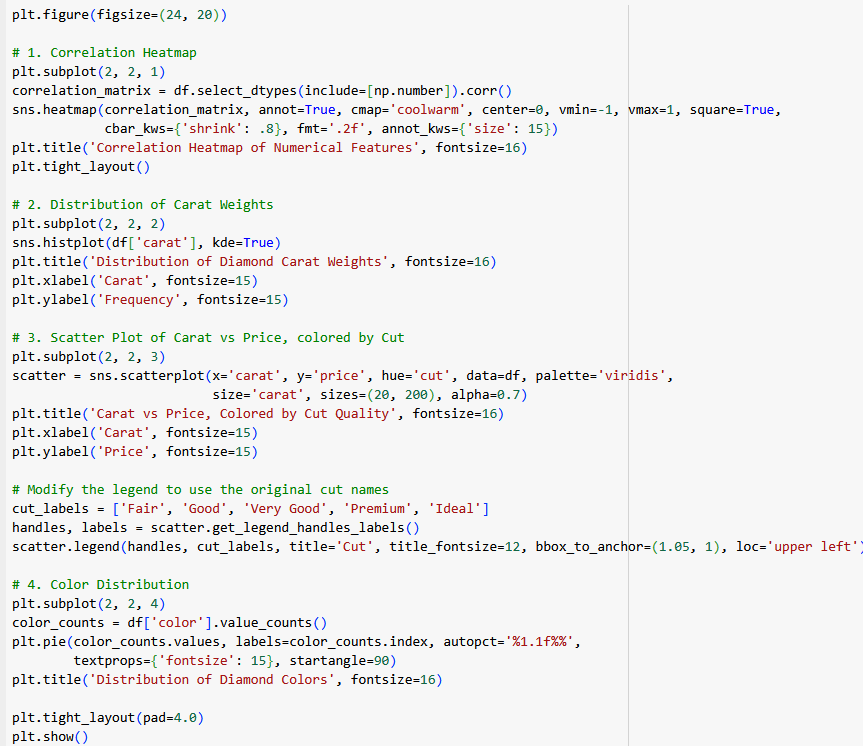
- Compute volume by multiplying the diamond's x, y, and z dimensions. This provides insights into the diamond's overall size, which can be a meaningful predictor of price.

**Density Calculation:**

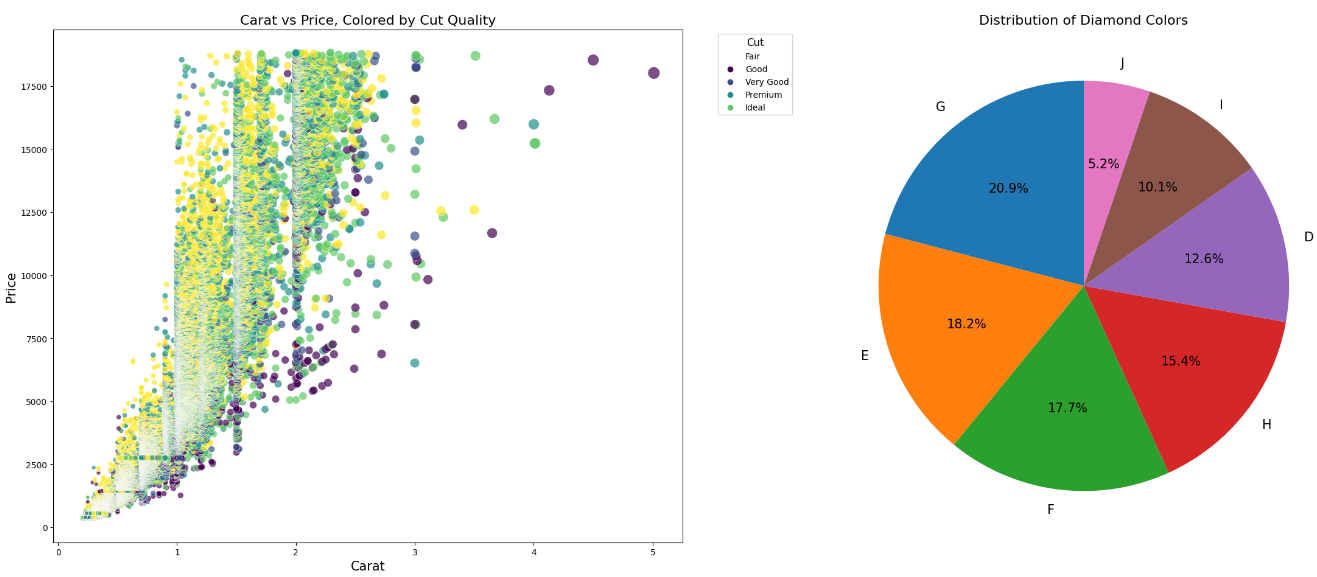
- Compute diamond density by dividing carat weight by volume. This new feature can reveal interesting characteristics about the diamond's composition.



**Visualisation:**



## 



asd

asd

## 

## Model Selection and Training

### Reason for chosen models:

**Logistic Regression:**

* Served as a linear baseline (one-vs-rest for multi-class).
* Assumes a linear decision boundary.

**k-Nearest Neighbors (kNN):**

* Instance-based method.
* Used k=5 neighbors.
* Classifies based on similar diamonds in the feature space.

**Random Forest:**

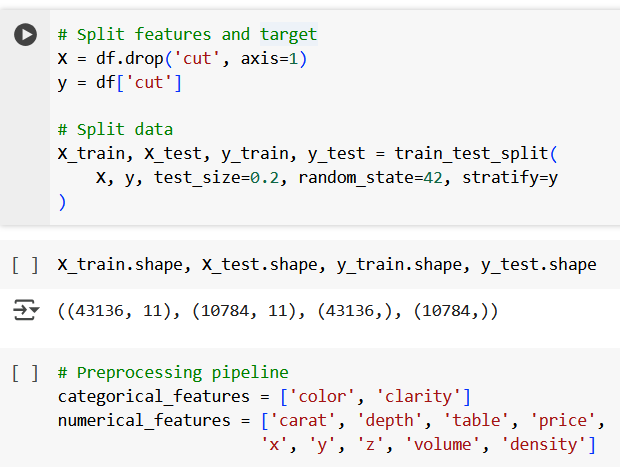
* Ensemble of 100 decision trees.
* Captures non-linear relationships.
* Handles mixed data well after encoding.

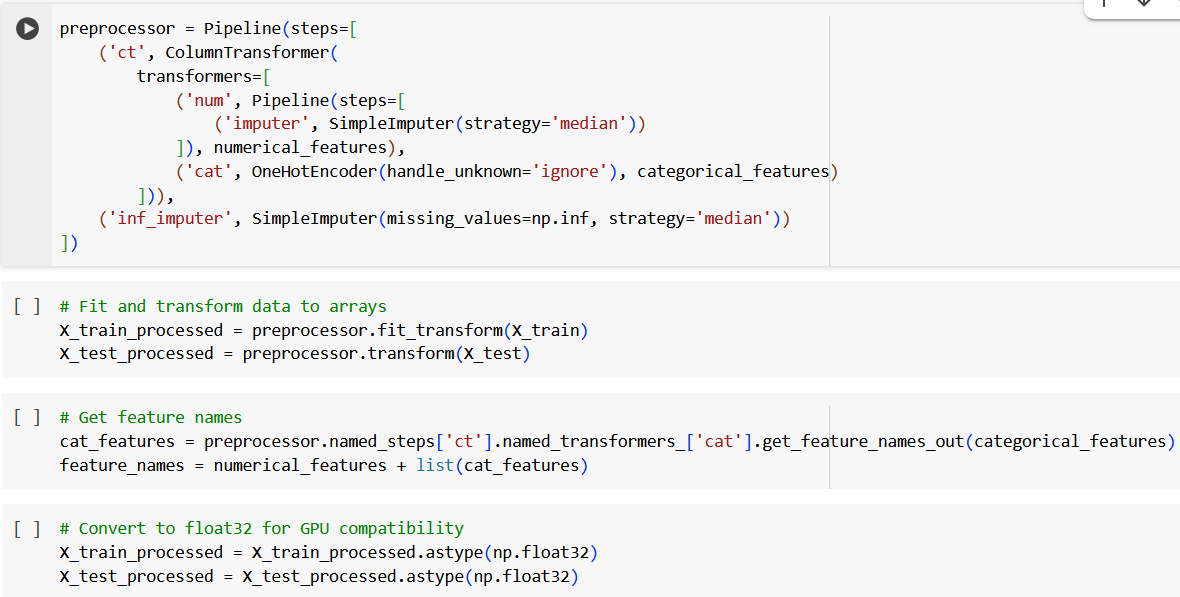
**Gradient Boosting & XGBoost:**

* More powerful tree ensembles.
* Build trees sequentially to minimize classification error.

### Training parameter setup:

-split the data into an 80% training set and 20% hold-out test set, stratified by cut grade​





### Model 1: RandomForestClassifier

# Initialize and train

model1 = RandomForestClassifier(

n\_estimators=200,

max\_depth=12,

class\_weight='balanced',

random\_state=42,

n\_jobs=-1

)

model1.fit(X\_train\_processed, y\_train)

### Model 2: KNeighborsClassifier

# Initialize and train

model1 = RandomForestClassifier(

n\_estimators=200,

max\_depth=12,

class\_weight='balanced',

random\_state=42,

n\_jobs=-1

)

model1.fit(X\_train\_processed, y\_train)

# Fit on original X\_train

model2.fit(X\_train, y\_train)

### Model 3: XGBoost

# Create DMatrix for XGBoost

dtrain = xgb.DMatrix(X\_train\_processed, label=y\_train, feature\_names=feature\_names)

dtest = xgb.DMatrix(X\_test\_processed, label=y\_test, feature\_names=feature\_names)

# GPU-accelerated parameters

params = {

'objective': 'multi:softprob',

'num\_class': 5,

'eval\_metric': ['mlogloss', 'merror'],

'tree\_method': 'gpu\_hist',

'gpu\_id': 0,

'eta': 0.1,

'max\_depth': 6,

'subsample': 0.8,

'colsample\_bytree': 0.8,

'seed': 42

}

# Train model

model3 = xgb.train(

params,

dtrain,

num\_boost\_round=1000,

evals=[(dtrain, 'train'), (dtest, 'test')],

early\_stopping\_rounds=20,

verbose\_eval=50

)

### Model 4: Gradient Boosting Machines

model4 = Pipeline([

('preprocessor', preprocessor),

('scaler', StandardScaler()), # optional, but often helpful

('gbm', GradientBoostingClassifier(

n\_estimators=100,

learning\_rate=0.1,

max\_depth=6,

random\_state=42

))

])

model4.fit(X\_train, y\_train)

### Model 5: Logistic Regression

model5 = Pipeline([

('preprocessor', preprocessor),

('scaler', StandardScaler()),

('logreg', LogisticRegression(

penalty='l2',

C=1.0,

max\_iter=1000,

random\_state=42

))

])

model5.fit(X\_train, y\_train)

## Evaluations

**Understanding Model Output**

* The Diamond Cut Classifier outputs predictions in the form of cut categories: Fair, Good, Very Good, Premium, and Ideal
* Along with the predicted category, the model provides probability scores for each possible outcome
* Example outputs include both the class label (e.g., ['Ideal']) and probability distribution (e.g., [0.02, 0.08, 0.15, 0.25, 0.50])

**Interpreting Confidence Scores**

* High confidence (>0.7 for one category) indicates strong confidence in the classification
* Moderate confidence (0.4-0.7) represents reasonable evidence for the classification
* Low confidence (<0.4) suggests uncertainty; the diamond may have borderline characteristics
* When probabilities are distributed across multiple categories, the diamond likely has features spanning different cut grades

**Model Performance Analysis**

* The classifier combines gradient boosting and XGBoost algorithms for enhanced accuracy
* The model is most accurate for diamonds with measurements within common ranges
* Extreme or unusual diamond measurements may result in less reliable predictions
* Use confusion matrices and accuracy metrics to quantitatively assess model performance
* Track prediction consistency by evaluating the model against known professionally graded diamonds

**Common Evaluation Scenarios**

* **High Confidence Prediction**: When one class probability dominates (e.g., 0.85 for "Ideal"), the model strongly believes in this classification
* **Split Confidence**: When probabilities are split between neighboring categories (e.g., 0.45 "Very Good", 0.40 "Premium"), the diamond sits at the boundary between categories
* **Distributed Confidence**: When probabilities are spread across multiple categories, the diamond has unusual characteristics or measurements

## Predictions of new data

**Loading the Trained Model**import joblib  
import pandas as pd  
import matplotlib.pyplot as plt

# Load the pre-trained model  
stacking\_clf = joblib.load('DiamondCutClassifier v1-6.pkl')

**Preparing Input Data**# Create a DataFrame with the diamond's characteristics  
input\_data = pd.DataFrame({

'carat': [0.70],

'color': ['D'],

'clarity': ['VVS1'],

'depth': [62.52],

'table': [57],

'price': [6122],

'x': [5.66],

'y': [5.62],

'z': [3.52]

})

# Calculate derived features

input\_data['volume'] = input\_data['x'] \* input\_data['y'] \* input\_data['z']

input\_data['density'] = input\_data['carat'] / (input\_data['volume'] + 1e-6)

Making Prediction  
# Define the diamond cut categories

diamondCuts = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']

# Make the prediction

predictions = stacking\_clf.predict(input\_data)

print(f"Predicted Diamond Cut: {[diamondCuts[p] for p in predictions]}")  
  
# Get the prediction probabilities

probs = stacking\_clf.predict\_proba(input\_data)[0]

**Visualizing Prediction Confidence**# Create a visualization of prediction probabilities

plt.figure(figsize=(10, 5))

bars = plt.bar(diamondCuts, probs, color='skyblue')

plt.ylabel('Probability', fontsize=12)

plt.title('Predicted Probability for Each Cut Category', fontsize=14)

plt.ylim(0, 1)

# Add percentage labels to each bar

for bar in bars:

height = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2., height,

f'{height:.1%}', ha='center', va='bottom')

plt.show()  
  
**Interpreting Prediction Results**

* The highest probability category indicates the model's best guess for the diamond's cut quality
* Consider the confidence distribution across all categories to understand the certainty of prediction
* For high-value diamonds, examine those with uncertainty (distributed probabilities) more carefully
* The model provides objective assessment based on physical measurements, complementing expert evaluation
* Use the prediction as one factor in determining a diamond's overall quality and value

## Fine tuning

asd

## Saving the Best Model

import joblib

joblib.dump(weighted\_xgb, 'DiamondCutClassifier v1-6.pkl')

# DiamondClassifier Notebook

- This notebook is an example to show how to run the model file saved from DiamondTrainer - “DiamondCutClassifier v1-6.pkl”

## Loading and running the saved model

1. Install the following dependencies using pip

pip install joblib xgboost

2. Create a Python script to load the model. Make sure “DiamondCutClassifier v1-6.pkl”

is in the same directory as the script

import joblib

import pandas as pd

import matplotlib.pyplot as plt

stacking\_clf = joblib.load('DiamondCutClassifier v1-6.pkl')

2. (Optional) Code to show values of features needed in dataset  
if hasattr(stacking\_clf, "classes\_"):

print("Classes:", stacking\_clf.classes\_)

if hasattr(stacking\_clf, "feature\_names\_in\_"):

print("Feature Names:", stacking\_clf.feature\_names\_in\_)

if hasattr(stacking\_clf, "n\_features\_in\_"):

print("Number of Features:", stacking\_clf.n\_features\_in\_)

3. Input dataset for prediction

\*Below is just an example. Input your own dataset for prediction if needed

input\_data = pd.DataFrame({ # This is an ideal diamond from MichaelTrio.com

'carat': [0.70],

'color': ['D'],

'clarity': ['VVS1'],

'depth': [62.52],

'table': [57],

'price': [6122],

'x': [5.66],

'y': [5.62],

'z': [3.52]

})

input\_data['volume'] = input\_data['x'] \* input\_data['y'] \* input\_data['z']

input\_data['density'] = input\_data['carat'] / (input\_data['volume'] + 1e-6)

diamondCuts = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']

4. Let the model predict and show its results

predictions = stacking\_clf.predict(finalTestData)

print("Predicted Diamond Cut:", [diamondCuts[p] for p in predictions])

probs = stacking\_clf.predict\_proba(input\_data)[0]

# Plot

plt.figure(figsize=(10, 5))

bars = plt.bar(diamondCuts, probs, color='skyblue')

plt.ylabel('Probability', fontsize=12)

plt.title('Predicted Probability for Each Cut Category', fontsize=14)

plt.ylim(0, 1)

for bar in bars:

height = bar.get\_height()

plt.text(bar.get\_x() + bar.get\_width()/2., height,

f'{height:.1%}', ha='center', va='bottom')

plt.show()

Example output:



