
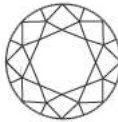
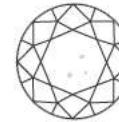
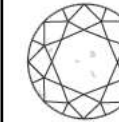
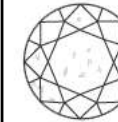



















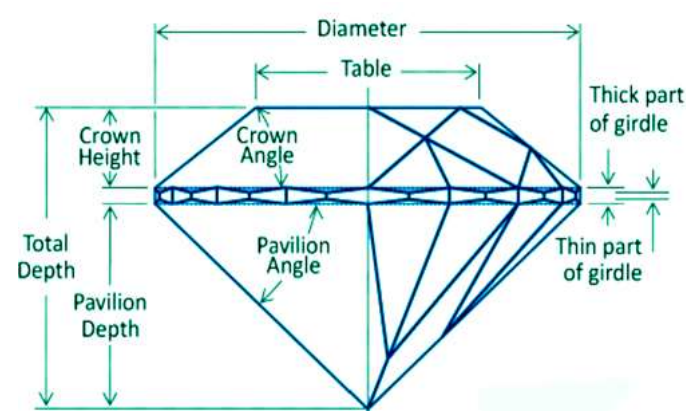


## For the beginner diamonds are given a value by these principle factors

- Carat: Carat is the weight of the diamond. Bigger the diamond, rare to find them.
- Clarity: Clarity of the diamond is intrusions in the diamond, which could be internal flaws, external defects.
- Color: Color of the diamond is self explanatory, and on scale of AtoZ, A is best and colorless, where as Z is worst and yellowish color. The colorless diamonds are rare and costly.
- Cut: Cut of the diamond is about physical dimensions of the diamond. It determine the light amount enter into the diamond. Anatomy of the diamond is an important parameter.

Clarity & Color chart, value diamonds described by viewing with naked eye only										
IF	Vvs1	Vvs2	Vs1	Vs2	SI1	SI2	SI3	I1	I2	I3
										
No flaws Perfect diamond (Rare & very very very expensive)	Nearly Perfect almost no visible flaws under 10x & none visible (Extremely expensive & rare)	Visible crystals under 10x magnification & possible to see if you know location (Very expensive, near rare)	Possible visible crystal inclusions to visible crystal inclusions to eye without magnification (Best buy value)	Visible natural crystal characteristic inclusions without magnification (Great value)						
            										
D E F	G H I J	K L M	N O P Q R	S T U V X Y Z						
colorless	near colorless	taint yellow	very light yellow	light yellow						

Below is the anatomy of a cut diamond. The table length, table width, diamond depth, depth%, and table% are also important features to determine the price of a diamond.



This Machine Learning Model is used to predict the Price of Diamonds based on the following parameters:

- price: The price of the Diamond
- carat: The carat value of the Diamond
- cut: The cut type of the Diamond, it determines the shine
- color: The color value of the Diamond
- clarity: The clarity type of the Diamond
- depth: The depth value of the Diamond
- table: Flat facet on its surface — the large, flat surface facet that you can see when you look at the diamond from above.
  - x: Width of the diamond

- y: Length of the diamond
- z: Height of the diamond

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 #bring in all of the machine learning and modeling techniques...lots of algorithms
5 #measuers to test the accuracy and compute the error score
6 from sklearn.linear_model import LinearRegression
7 from sklearn.tree import DecisionTreeRegressor
8 from sklearn.ensemble import RandomForestRegressor
9 from xgboost import XGBRegressor
10 from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, accuracy_score
11 from math import sqrt
12 from sklearn.model_selection import train_test_split
13 from sklearn import metrics
14 #pickle what does it do?
15 import pickle
```

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```


```
1 #importing the diamond_df and also eliminating the white space and null values
2 diamond_df = pd.read_csv('/content/drive/MyDrive/PredictiveAnalytics/Projects/Project1/data/diamonds.csv')
3 # to ad later possibly after a little eda :, na_values = ['NA', '?'], names = columns, delim_whitespace=True
```

## EDA

```
1 diamond_df.shape
```

```
(53940, 10)
```

```
1 diamond_df.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z	
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43	
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31	
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31	
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63	
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75	

```
1 diamond_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
#   Column      Non-Null Count  Dtype
---  -
0   carat       53940 non-null  float64
1   cut         53940 non-null  object
2   color       53940 non-null  object
3   clarity     53940 non-null  object
4   depth       53940 non-null  float64
5   table       53940 non-null  float64
6   price       53940 non-null  int64
7   x           53940 non-null  float64
8   y           53940 non-null  float64
9   z           53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

```
1 diamond_df.describe()
```

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699

```
1 diamond_df.isnull().sum()
```

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

```
1 diamond_df.isna().sum()
```

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

```
1 diamond_df.columns
```

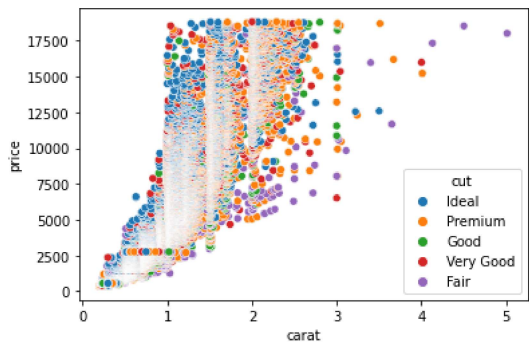
```
Index(['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'price', 'x', 'y',
      'z'],
      dtype='object')
```

```
1 diamond_df.cut.unique()
```

```
array(['Ideal', 'Premium', 'Good', 'Very Good', 'Fair'], dtype=object)
```

```
1 sns.scatterplot(x=diamond_df.carat , y=diamond_df.price, hue=diamond_df.cut)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb5a54f9d0>
```

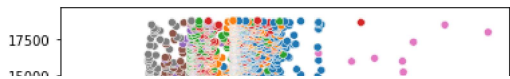


```
1 diamond_df.clarity.unique()
```

```
array(['SI2', 'SI1', 'VS1', 'VS2', 'VVS2', 'VVS1', 'I1', 'IF'],
      dtype=object)
```

```
1 sns.scatterplot(x=diamond_df.carat , y=diamond_df.price, hue=diamond_df.clarity)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb5a6164c0>
```



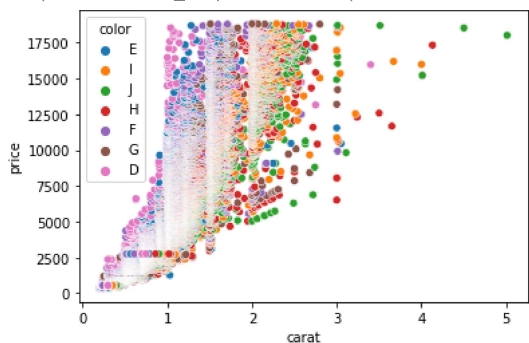
```
1 diamond_df.color.unique()
```

```
array(['E', 'I', 'J', 'H', 'F', 'G', 'D'], dtype=object)
```

```
7500 | | vs1 |
```

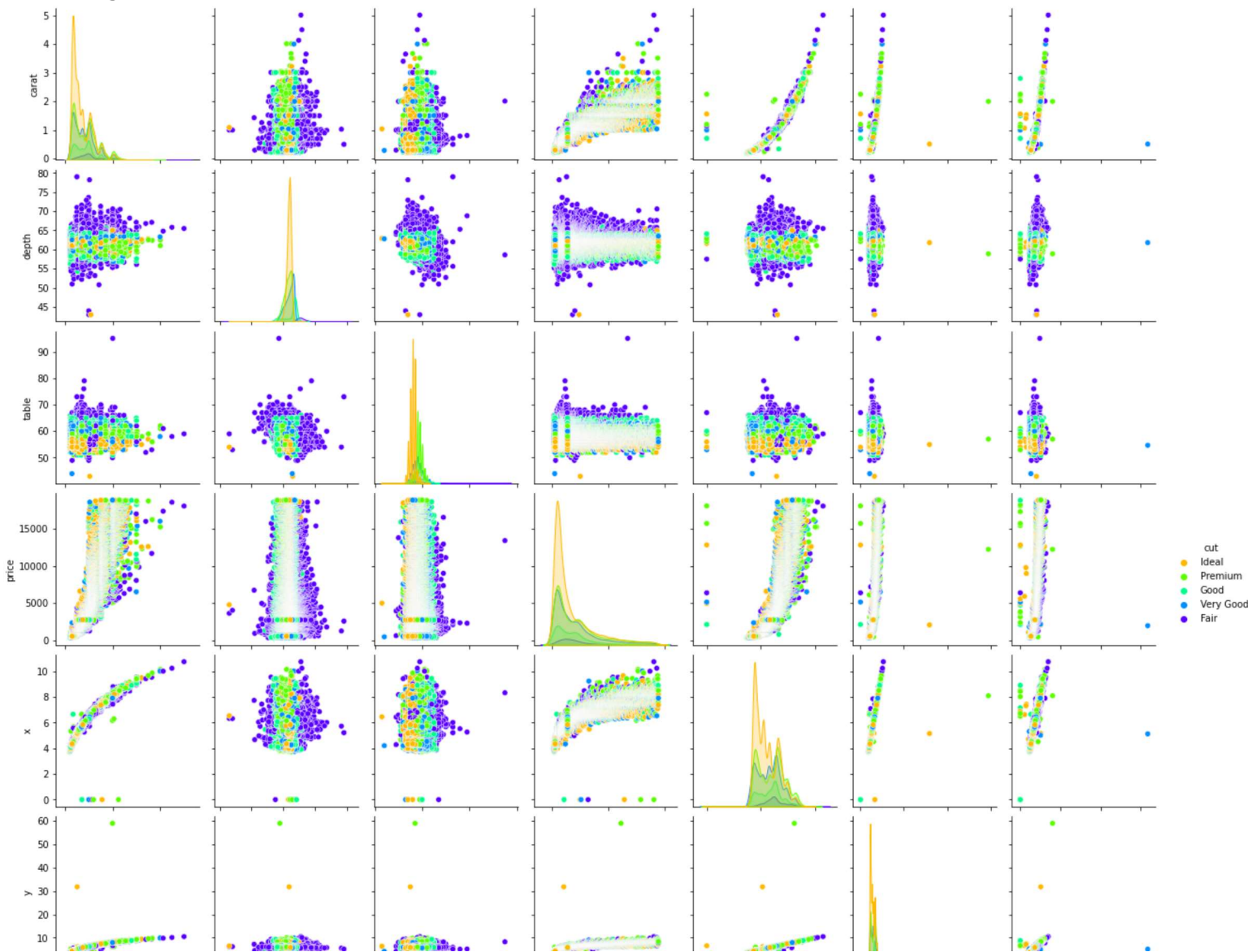
```
1 sns.scatterplot(x=diamond_df.carat , y=diamond_df.price, hue=diamond_df.color)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcb59e7f550>
```



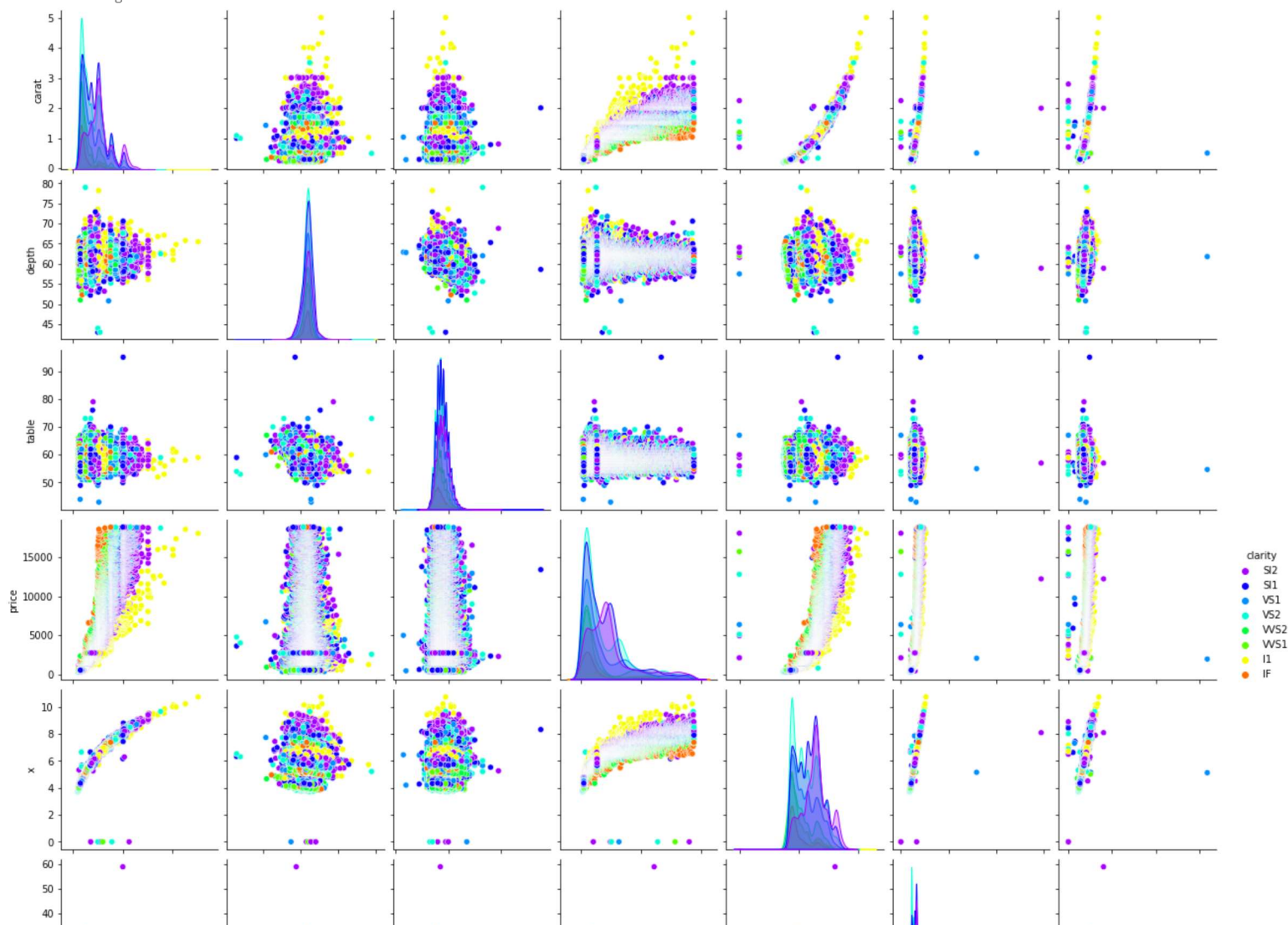
```
1 sns.pairplot(diamond_df, hue='cut', palette='gist_rainbow')
```

```
<seaborn.axisgrid.PairGrid at 0x7fcb6464bc40>
```



```
1 sns.pairplot(diamond_df, hue='clarity', palette='gist_rainbow_r')
```

<seaborn.axisgrid.PairGrid at 0x7fcb5bbfab20>



## Data Cleaning

```
1 diamond_df.shape
```

```
(53940, 10)
```

Duplicates are an extreme case of nonrandom sampling, and they bias your fitted model. Including them will essentially lead to the model overfitting this subset of points.

Dummy variables for categorical variables- The dataset includes 3 categorical variables (cut, color, and clarity). I chose to create dummy variables for those categorical variables using a "replace" function.

```
1 #looking at all of the features need to assign a numeric value to 'cut','clarity', 'color'
2 diamond_df.head()
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.500000	55.0	326.0	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.799999	61.0	326.0	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.900002	65.0	327.0	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.400002	58.0	334.0	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.299999	58.0	335.0	4.34	4.35	2.75

First we need to understand if there singular correlations between cut, color, clarity and the price of the diamond\_df. Is there a correlation between the size and the price?

What is the distribution of the samples according to the features of the diamond\_dfs.

```

1 diamond_df.cut.replace({'Ideal':5, 'Premium':4, 'Good':2, 'Very Good':3, 'Fair':1}, inplace=True)
2 diamond_df.color.replace({'E':2, 'I':6, 'J':7, 'H':5, 'F':3, 'G':4, 'D':1}, inplace=True)
3 diamond_df.clarity.replace({'SI2':1, 'SI1':2, 'VS1':3, 'VS2':4, 'VVS2':5, 'VVS1':6, 'I1':7, 'IF':8}, inplace=True)

```

## Splitting

```

1 #creating X and y so you need to turn something around... has a regular flow...
2 #creating a numeric list of the diamond_df dataframe
3 #turning it into an np.array so we can work with it in a more stream lined computation
4 #also into np.floats with 32 bit which is standard for computing
5 numeric_list = diamond_df.select_dtypes(include=[np.number]).columns
6 diamond_df[numeric_list] = diamond_df[numeric_list].astype(np.float32)

1 #X and Y values... x = all columns except mpg because it is the Y
2 X = diamond_df[['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y',
3               'z']].values
4 y = diamond_df['price'].values

1 type(y)

        numpy.ndarray

1 type(X)

        numpy.ndarray

1 #Splitting data set for training/testing
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state = 101)

1 #Splitting data set for training/testing
2 #REspecify that the train diamond_df is a float of 32bytes
3 X_train = X_train.astype(np.float32)

1 #looking at the diamond_df shape
2 #printing with f-string \train and test diamond_df\seeing the shape of all of the diamond_df
3 print('The shape of the split data is: \n X_train: \t{X_train.shape} \n X_test: \t{X_test.shape} \n y_train: \t{y_train.shape} \n y_test: \t{y_test.shap

The shape of the split data is:
X_train:      (43152, 9)
X_test:       (10788, 9)
y_train:      (43152,)
y_test:       (10788,)

```

f is string formatting..in the print

```
name = input('enter name')
```

```
print(f'hello world {name}')
```

<https://realpython.com/python-string-formatting/>

## Machine Learning

```

1 #bring in a variable
2 LinearRegression_model = LinearRegression()
3 DecisionTree_model = DecisionTreeRegressor()
4 RandomForest_model = RandomForestRegressor()
5 XGBRegressor_model = XGBRegressor()

1 LR = LinearRegression_model.fit(X_train, y_train)

1 print('R squared of the Linear Regression on training set: {:.2%}'.format(LR.score(X_train, y_train)))
2 print('R squared of the Linear Regression on test set: {:.2%}'.format(LR.score(X_test, y_test)))

R squared of the Linear Regression on training set: 88.54%
R squared of the Linear Regression on test set: 88.61%

1 #creating a list of models to save time. 4 models trained in one line
2 models = [LinearRegression_model, DecisionTree_model, RandomForest_model, XGBRegressor_model]
3 for model in models:
4     model.fit(X_train, y_train)
5     print(f'{model} is trained!')

```



```

LinearRegression() is trained!
DecisionTreeRegressor() is trained!
RandomForestRegressor() is trained!
[00:58:33] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
XGBRegressor() is trained!

```

```

1 #Machine Learning
2 #now we want to get the accuracy. creating the score
3 accuracy_LinearRegression = LinearRegression_model.score(X_test, y_test)
4 accuracy_DecisionTree = DecisionTree_model.score(X_test, y_test)
5 accuracy_RandomForest = RandomForest_model.score(X_test, y_test)
6 accuracy_XGBoost = XGBRegressor_model.score(X_test, y_test)

```

```

1 #Machine Learning
2 #creating a dictionary now can choose which model works best
3 models = {'LinearRegression_model': accuracy_LinearRegression, 'DecisionTree_model': accuracy_DecisionTree, 'RandomForest_model': accuracy_RandomForest, 'XGBRegressor_model': accuracy_XGBoost}
4
5 for model, score in models.items():
6     print(f'The accuracy score for the {model} is {round(score, 4)}')

```

```

The accuracy score for the LinearRegression_model is 0.8861
The accuracy score for the DecisionTree_model is 0.9655
The accuracy score for the RandomForest_model is 0.9813
The accuracy score for the XGBRegressor_model is 0.9717

```

```

1 diamond_df.head()

```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	5.0	2.0	1.0	61.500000	55.0	326.0	3.95	3.98	2.43
1	0.21	4.0	2.0	2.0	59.799999	61.0	326.0	3.89	3.84	2.31
2	0.23	2.0	2.0	3.0	56.900002	65.0	327.0	4.05	4.07	2.31
3	0.29	4.0	6.0	4.0	62.400002	58.0	334.0	4.20	4.23	2.63
4	0.31	2.0	7.0	1.0	63.299999	58.0	335.0	4.34	4.35	2.75

## TEST

```

1 #based on the 7 columns making a feature called testing vector
2 test_x = np.zeros((1, 9))
3 test_x

```

```

array([[0., 0., 0., 0., 0., 0., 0., 0., 0.]])

```

```

1 #now assigning values to the test value x
2 test_x[0, 0] = .29
3 test_x[0, 1] = 4.00
4 test_x[0, 2] = 6.00
5 test_x[0, 3] = 4.00
6 test_x[0, 4] = 62.00
7 test_x[0, 5] = 58.00
8 test_x[0, 6] = 4.20
9 test_x[0, 7] = 4.23
10 test_x[0, 8] = 2.63
11 test_x

```

```

array([[ 0.29,  4. ,  6. ,  4. , 62. , 58. ,  4.2 ,  4.23,  2.63]])

```

???what is up with linear Rgression having a negative value??

```

1 prediction = LinearRegression_model.predict(test_x) #??????
2 round(float(prediction[0]), 2)
3 #at two precission intervals

```

```

-519.92

```

```

1 #now a list of all the models no quotes
2 models = [LinearRegression_model, DecisionTree_model, RandomForest_model, XGBRegressor_model]

```

```

1 #running a loop for models in models predict and print with fstring
2 #linear predicted the best
3 for model in models:
4     prediction = model.predict(test_x)
5     print(f'The model {model} predicts an price of {round(float(prediction[0]), 2)}')

```

```
5 print('The model {model} predicts an price of {round(float(prediction[0]), 2)}')  
The model LinearRegression() predicts an price of -519.92  
The model DecisionTreeRegressor() predicts an price of 334.0  
The model RandomForestRegressor() predicts an price of 381.96  
The model XGBRegressor() predicts an price of 252.22
```

1st test of numbers predicts

\$326.00 DecisionTree prediction was exact based on features inputed into test variables.

2nd test of numbers

\$334.00 DecisionTree prediction was exact based on features inputed into test variables.

Which Machine Learning Model works best linear or desicion for testing and prediction purposes? LinearRegression or DecicionTree... chosing the model to save

```
1 #calling it model file holding the folder  
2 model_file = '/content/drive/MyDrive/PredictiveAnalytics/Projects/Project1'
```

```
1 #Desicion Tree  
2 with open('dtm_file', 'wb') as file:#saving the DTfile  
3     model = pickle.dump(DecisionTree_model, file)
```

```
1 #linear  
2 #now saving the file with the picle library...  
3 #with it open you dump the linear regression file into it.  
4 #make sure it saves to google drive in the right folder.#  
5 with open('pkl_file', 'wb') as file:  
6     model = pickle.dump(LinearRegression_model, file)
```