

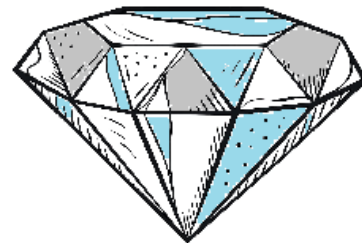


100% ML: Predict DIAMOND prices. Machine Learning basics for beginners



Amongst a variety of items not quantitatively or statistically valued by buyers, Diamonds are possibly the most common. The purchase is far less from rational with a heavy bent on emotional ties. The jewelers would entice every man (and woman) by marketing it as a necessity for the occasion and as a status symbol, and by calling this pricey and unaffordable item as priceless. But what if you get the power to predict diamond prices before negotiating? Won't that be a really cool edge!

The Engagement Ring Story: How De Beers Created a Multi-Billion Dollar Industry From the Ground Up



Written by Lindsay Kolowich Cox [@lkolow](#)

However, the actual value of a diamond is determined by a gemologist after inspecting its various "features" (using proper machine learning basics terminology now since this article is on ML models for prediction) and applying a relative valuation principle of "compare and price".

CONSUMER'S GUIDE to Buying Diamond

Its holiday season! You can use this algorithm to predict the price of the diamond you really desire — a little help in negotiating price with local retailers.

This article is a no-brainer, easy-to-follow article to grasp machine learning basics in 15 mins for a Linear regression problem. The machine learning examples here show how to predict a number using minimal dataset at a fairly good accuracy.



Table of Contents

| | |
|---|-------------------------------------|
| 100% ML: Predict DIAMOND prices. Machine Learning basics for beginners | 1 |
| Get the data | 3 |
| Data preprocessing | 4 |
| Convert the features to log scale..... | 6 |
| 1. Check for any ZERO value | 6 |
| 2. View and remove outliers using z-score..... | 7 |
| 3. Convert to log scale | 9 |
| Convert categorical column to numerical column using labelencoder | 10 |
| Divide the data into independent variable features (X) and dependent variable (y) | 11 |
| Train Test Split..... | 12 |
| Machine Learning algorithms | 13 |
| Linear regression..... | 14 |
| K-nearest neighbors (KNN) | 15 |
| Support vector machines (SVM) | 17 |
| Regression Evaluation Metrics | 17 |
| Random Forest..... | 18 |
| The big daddy — Artificial neural networks..... | 19 |
| How to create an Artificial neural network (ANN) using Keras | 20 |
| References | 22 |
| Frequently Asked Questions (FAQ) | 23 |
| What are different types of Machine Learning | 23 |
| What is the process followed to create ML models for prediction? | 23 |
| What is bias and variance | 23 |
| I am so confused on a confusion matrix. Wait! What's it after all?..... | 24 |
| Did I miss Semi-supervised Machine Learning? What's that? | 24 |
| Precision and Recall formula in 10 seconds! | 24 |
| What's k-fold cross validation? | 24 |
| Leave a Comment..... | Error! Bookmark not defined. |
| Other resources - www.fivestepguide.com | 25 |

In last 2 decades, the valuation and pricing has become more or less quantitative i.e. calculations based on values of many properties not just limiting to 4Cs (carat, cut, colour, clarity).

Properties like culet, pavilion, crown, girdle, girdle thickness polish, symmetry, fluorescence, table, depth and so on are the most easily identifiable and recordable features while the diamond is actually cut.



[Click to Tweet](#)

Let me begin showing you how to predict DIAMOND prices now. In this article, I will show very simple steps in machine learning process for ML models for prediction and how to approach a complex problem with relatively good accuracy.

Get the data

The first steps in Machine learning process are to **Import all basic libraries**.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from sklearn import preprocessing
```

Let's begin with training to predict diamond prices based on dataset from Kaggle—it has basic data on more than 54000 diamonds. In a follow-up article, we will train the same model and [predict prices based on real data from pricescope.com](#).

```
df = pd.read_csv("diamonds.csv")
df.drop('Unnamed: 0', axis=1, inplace=True)
display(df.head(3))
```

The output is something like the following:

| | carat | cut | color | clarity | depth | table | price | l | w | d |
|---|-------|---------|-------|---------|-------|-------|-------|------|------|------|
| 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 |



Data preprocessing

The next steps in Machine learning process begin with basic data preprocessing to predict diamond prices. We first look if any null values or unexpected datatype are present.

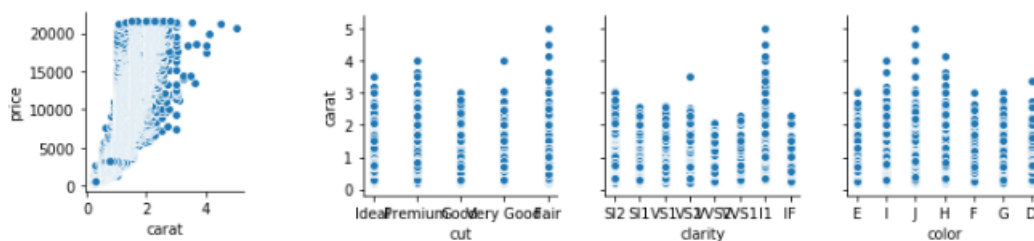
```
df.isnull().sum()df.info()
```

Now we plot a Pair-plot of Price vs. 4 Cs (Carat, Cut, Color, Clarity) — the most popular and marketed properties of a diamond.

```
# plot price vs. carat
sns.pairplot(df, x_vars=['carat'], y_vars = ['price'])

# plot carat vs other Cs
sns.pairplot(df, x_vars=['cut', 'clarity', 'color'], y_vars = ['carat'])
plt.show()
```

The output is something like the following:





We can see that the properties charts reveal a lot about how and where bulk of diamonds fall under each property value e.g. bigger diamonds (higher carat) fall in **Fair** cut, **I1** clarity and **H-I** color. These are poor (commercial grade) diamonds generally sold by retail jewelry shops across the world to attract consumers with advertisements like '**Diamonds at 50% off**'

The price vs. carat chart also show that there are some outliers in the dataset i.e. few diamonds that are really *over priced!*

Now we need to see the distribution of the dataset used to predict diamond prices. We will create a histogram plot for this. Being able to plot and visually interpret your ML models for prediction is a very vital part in completely grasping Machine Learning basics in 15 mins. Therefore, those steps in machine learning process needed for plotting and visually interpreting the data will be shown and explained everywhere in this article.

First we define the `histplot()` function.

```
def histplot(df, listvar):
    fig, axes = plt.subplots(nrows=1, ncols=len(listvar), figsize=(20, 3))
    counter=0
    for ax in axes:
        df.hist(column=listvar[counter], bins=20, ax=axes[counter])
        plt.ylabel('Price')
        plt.xlabel(listvar[counter])
        counter = counter+1
    plt.show()
```

Now we list the continuous variables and leave out the categorical variable. A continuous variable e.g. **carat** is one which has numerical values whereas a categorical variable is the one with alphanumeric values as categories e.g. **clarity**

```
linear_vars = df.select_dtypes(include=[np.number]).columns
display(list(linear_vars))
```

Output is `['carat', 'depth', 'table', 'price', 'l', 'w', 'd']`

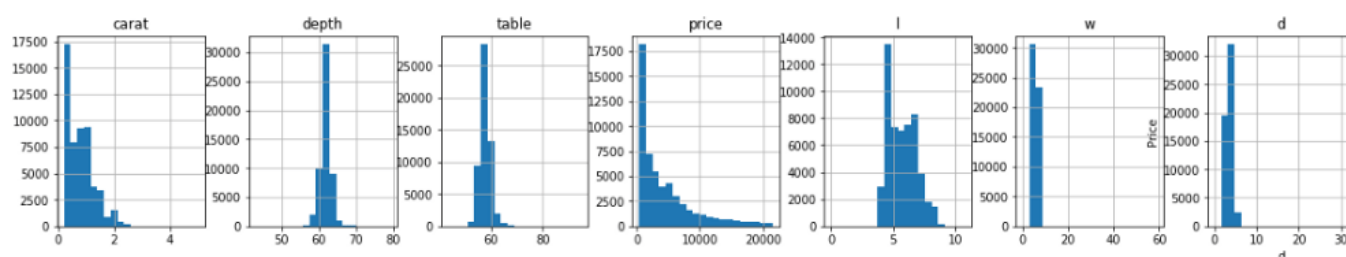
Now we plot the histogram



```
histplot(df,linear_vars)
```

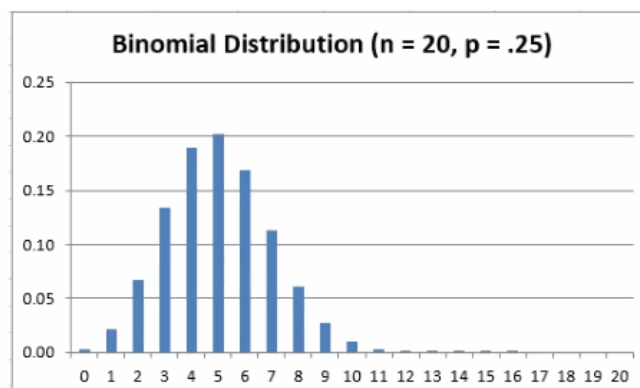
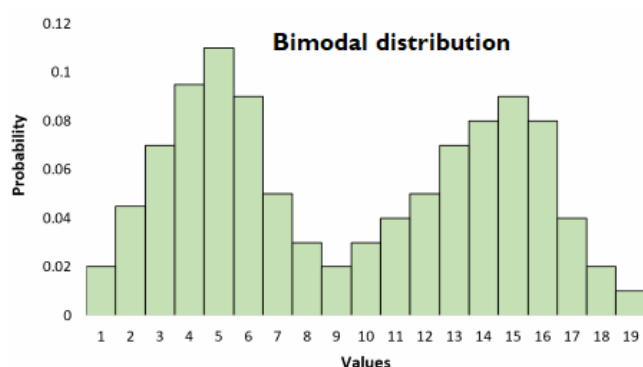
This reveals the distribution of each property. As expected, we see that the data is not normally distributed. After all, how can you expect a 1 carat diamond to be priced just at twice the price of a half-carat given all properties remain the same, while a 1 carat diamond looks much bigger to the eye when in a ring or earrings for that matter than a half carat one.

[Diamond database](#)



Convert the features to log scale

One of the key steps in Machine learning process is to convert features having bimodal distribution to ones with binomial distribution.



Bimodal distribution vs Binomial distribution

1. Check for any ZERO value

Amongst features namely **table**, **depth**, **l**, **w**, **d**, check if any continuous variable has zero value. This results in a *division by zero* error when converted to log. Add a tiny number 0.01 to any zero value.



```
print('0 values ->', 0 in df.values)
df[linear_vars] = df[linear_vars] + 0.01

print('Filled all 0 values with 0.01. Now any 0 values? ->', 0 in df.values)
```

The output is:

```
0 values --> True
Filled all 0 values with 0.01. Now any 0 values? --> False
```

2. View and remove outliers using z-score

Since we could briefly sense some outliers in the pairplot charts, let's dwell deeper and see whether there genuinely are any outliers. Let's first begin by printing top X values of each diamond

```
def sorteddf(df, listvar):
    for var in listvar:
        display('sorted by ' + var + ' --> ' + str(list(df[listvar].sort_values(by=var, ascending=False, topX=10))))
```

100% ML: Predict DIAMOND prices. Machine Learning basics for beginners

Output is like:

```
sorted by carat --> [5.02, 4.51, 4.14, 4.02, 4.02]'
sorted by depth --> [79.01, 79.01, 78.21000000000001, 73.61, 72.91000000000001]'
sorted by table --> [95.01, 79.01, 76.01, 73.01, 73.01]'
sorted by price --> [21646.459999999995, 21640.709999999995, 21626.909999999996, 21624.609999999997, 21624.609999999997]'

sorted by l --> [10.75, 10.24, 10.15, 10.03, 10.02]'
sorted by w --> [58.91, 31.810000000000002, 10.549999999999999, 10.17, 10.11]'
sorted by d --> [31.810000000000002, 8.07, 6.99, 6.7299999999999995, 6.4399999999999995]'
```

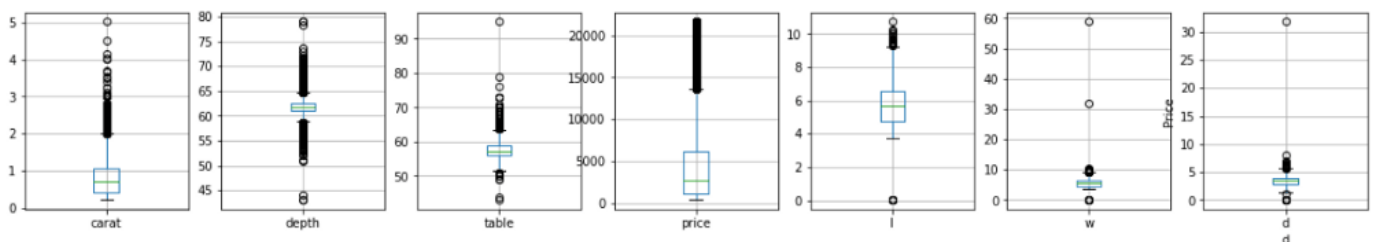
From this list itself, we see that there are some outliers for w, d . Let's visualize those using boxplots. Create a boxplot function



```
def dfboxplot(df, listvar):  
    fig, axes = plt.subplots(nrows=1, ncols=len(listvar), figsize=(20, 3))  
    counter=0  
    for ax in axes:  
        df.boxplot(column=listvar[counter], ax=axes[counter])  
        plt.ylabel('Price')  
        plt.xlabel(listvar[counter])  
        counter = counter+1  
    plt.show()
```

Call dfboxplot to view outliers for all properties


```
dfboxplot(df, linear_vars)
```



We can now clearly visualize that there are outliers for *table*, *w* and *d* properties. Lets call `removeoutliers()` function to remove the outliers based on z-score. There are several methods to remove outliers but I am going to follow the z-score process here since its the easiest to implement and delivers optimal results—after all, this is a no-brainer quickie article for newbie users.

For more on all other alogs to remove outliers, check these out:

[Stackoverflow: Outliers in Pandas dataframe](#)



We've Got Awesome News For You!

Get the latest TECH bytes delivered direct to your mailbox as soon as released. Several useful articles on FiveStepGuide.

Subscribe now

Name:*

Email:*

☒ I agree to be added to the mailing list

Get hooked

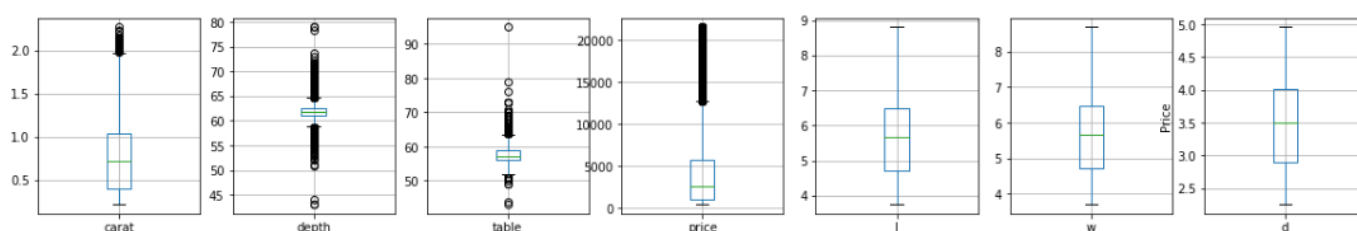
To dismiss, click anywhere out of this box

If you liked this article and would like to read many more such articles, please subscribe to our newsletter by clicking here on the image on your left



```
def removeoutliers(df, listvars, z):  
    from scipy import stats  
    for var in listvars:  
        df1 = df[np.abs(stats.zscore(df[var])) < z]  
    return df1  
  
df = removeoutliers(df, linear_vars,2)
```

Now, after calling dfboxbplot again to view outliers for all properties, we see the output as:



3. Convert to log scale

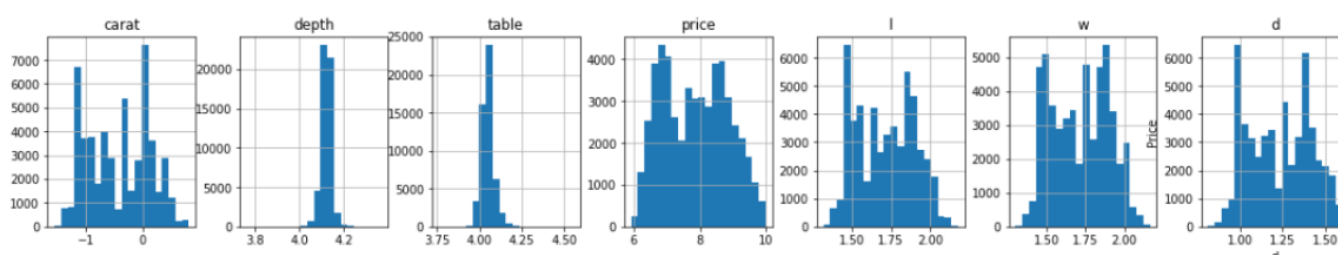
Since we saw earlier that most features (or properties of a diamond) are not normally distributed, and one of the most favored approach, if not a prerequisite, is to use Gaussian distributed (another name for normally distributed) data.

[Quora: Gaussian vs. normal distribution](#) and [Medium: Gaussian-distribution-in-data-science](#)

So in order to predict diamond prices correctly, we would need to *logarithmize* the data for use in ML models for prediction.

```
# this log converts dataframe's features inplace  
def convertfeatures2log(df, listvars):  
    for var in listvars:  
        df[var] = np.log(df[var])  
    convertfeatures2log(df, linear_vars)  
histplot(df, linear_vars)
```

The output now is:





Convert categorical column to numerical column using labelencoder

We now have to convert all categorical columns to numerical columns using labelencoder. You may read more about labelencoder in the following webpage—quick and easy to understand description there.

[Towards Data Science: encoding categorical features](#)

First we define the `convert_catg()` function to convert categorical columns to numerical columns

```
def convert_catg(df1):
    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()

    # Find the columns of object type along with their column index
    object_cols = list(df1.select_dtypes(exclude=[np.number]).columns)
    object_cols_ind = []
    for col in object_cols:
        object_cols_ind.append(df1.columns.get_loc(col))

    # Encode the categorical columns with numbers
    for i in object_cols_ind:
        df1.iloc[:,i] = le.fit_transform(df1.iloc[:,i])
```

Next, we run the function and see the head of the dataframe.

```
convert_catg(df).head(3)
```



| | carat | cut | color | clarity | depth | table | price | l | w | d |
|---|-----------|-----|-------|---------|----------|----------|----------|----------|----------|----------|
| 0 | -1.427116 | 2 | 1 | 3 | 4.119200 | 4.007515 | 5.926686 | 1.376244 | 1.383791 | 0.891998 |
| 1 | -1.514128 | 3 | 1 | 2 | 4.091173 | 4.111038 | 5.926686 | 1.360977 | 1.348073 | 0.841567 |
| 2 | -1.427116 | 1 | 1 | 4 | 4.041471 | 4.174541 | 5.929749 | 1.401183 | 1.406097 | 0.841567 |

Let's now prepare our data further to start building the **Machine learning model**.

Other articles on www.fivestepguide.com that you may like:

5-step All-inclusive ETF portfolio builder with examples. A Step-by-step guide.

Cooking Food the Right way. Better Cooking methods

How to get fit with healthy cookware? Utensils for different foods.

Divide the data into independent variable features (X) and dependent variable (y)

The next step in our journey to predict diamond prices is to set independent variable (X) and dependent variable (y).

X is the matrix (or DataFrame) for all the properties (independent features) and y is the vector for output (dependent variables) i.e. diamond price

```
X_df = df.drop(['price', 'l', 'w', 'd'], axis=1)
y_df = df[['price']] # two [] to create a DF
```

Now, we determine correlations between price and all other attributes.

- I will be combining both X (already converted categorical to numerical) and y to form a new dataframe for correlation



```
df_le = X_df.copy()

# add a new column in dataframe - join 2 dataframe columns-wise
df_le['price'] = y_df['price'].values
df_le.corr()
```

The statement `df_le = X_df` means that `df_le` will be like a pointer to `X_df`. Any change made to `df_le` will actually be a change to `X_df`. Therefore `df_le = X_df.copy()` is better.

- It seems price is highly correlated with carat and fairly with table, color and clarity, not much with cut

| | carat | cut | color | clarity | depth | table | price |
|---------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| carat | 1.000000 | 0.017165 | 0.215271 | -0.214794 | 0.002246 | 0.194943 | 0.963616 |
| cut | 0.017165 | 1.000000 | 0.001026 | 0.023756 | -0.172718 | 0.158586 | 0.025828 |
| color | 0.215271 | 0.001026 | 1.000000 | -0.014120 | 0.038490 | 0.019935 | 0.106469 |
| clarity | -0.214794 | 0.023756 | -0.014120 | 1.000000 | -0.039886 | -0.087248 | -0.096520 |
| depth | 0.002246 | -0.172718 | 0.038490 | -0.039886 | 1.000000 | -0.305286 | -0.022676 |
| table | 0.194943 | 0.158586 | 0.019935 | -0.087248 | -0.305286 | 1.000000 | 0.158129 |
| price | 0.963616 | 0.025828 | 0.106469 | -0.096520 | -0.022676 | 0.158129 | 1.000000 |

Note on Feature scaling—it seems it's not needed here since we have already log the properties. Nevertheless, if I had feature scaled, then the code would have been as written below:

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()

X_df = sc_X.fit_transform(X_df)
X_df[0:3]
```

Train Test Split

Data scientists generally split the data for machine learning into either two or three subsets: 2 subsets for training and testing, while 3 for training, validation and testing. I will talk about it in detail a bit later. This



data split prevents an algorithm from overfitting and underfitting. I have explained overfitting and underfitting briefly in my other post here:

[Machine-learning-interview-notes-part-3-short-bullet-points](#)

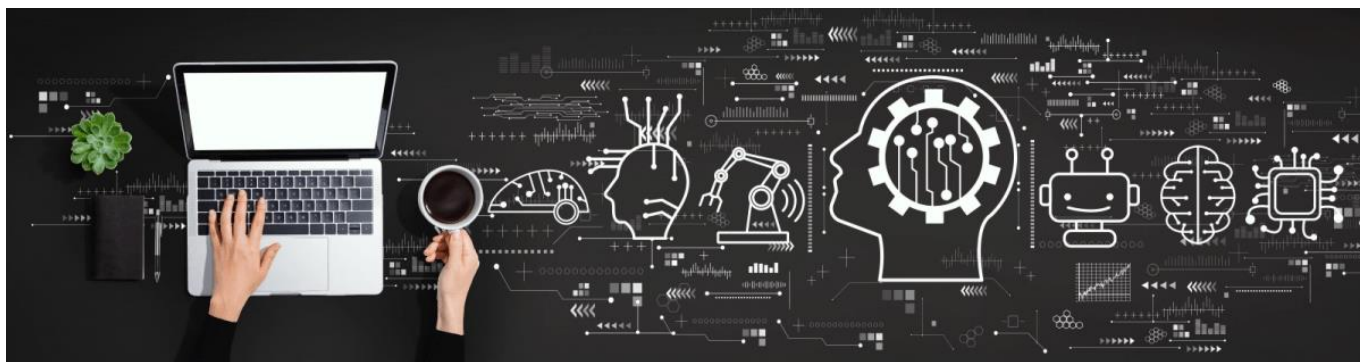
The code for train_test_split is as follows:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_df, y_df, test_size=0.3, random_state=42)
```

Now we will run different algorithms. Once you are really ready with your data, coding a simple Machine Learning algorithm is a cakewalk. Let me demonstrate a few with code, output and charts.

- split the data into training set and test set.
- Train the algorithm on training set data
- Use the trained algorithm (or trained ML model) to predict prices from diamond properties in test data.
- Verify / visualize / measure the differences between predicted prices and actual prices using scatterplots, histograms, accuracy metrics etc.

Machine learning algorithms



Now is the time to start coding machine learning algorithms on this data. After completing all necessary data pre-processing steps, let's move ahead and see some Machine learning algorithms in action.



Linear regression

Lets start with the simplest of all, the ubiquitous linear regression model.

1. Import `LinearRegression` class from Sci-kit learn
2. Create an object of `LinearRegression` model
3. Fit the model to `X_train` and `y_train`
4. Make predictions

```
# Import the class from Sci-kit learn
from sklearn.linear_model import LinearRegression

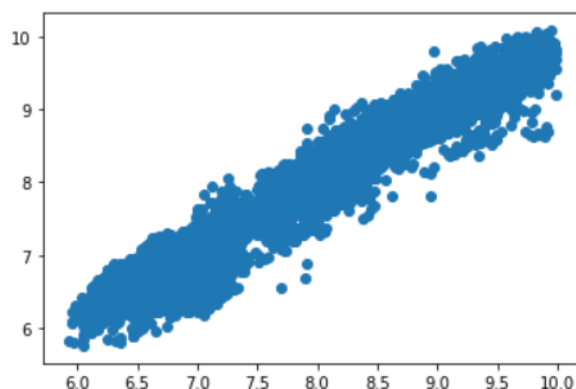
# Create an object of LinearRegression model
reg_all = LinearRegression()

# Fit the model to X_train and y_train
reg_all.fit(X_train,y_train)

# Make predictions
y_pred=reg_all.predict(X_test)
```

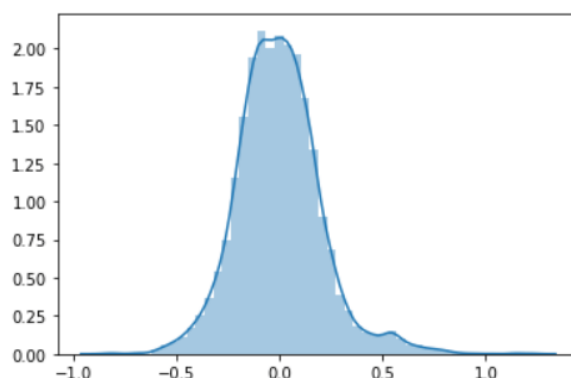
Now visualize the discrepancy of predictions vs. actual prices using scatterplot and histogram

```
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
```





```
import seaborn as sns
sns.distplot((y_test-y_pred),bins=50);
```

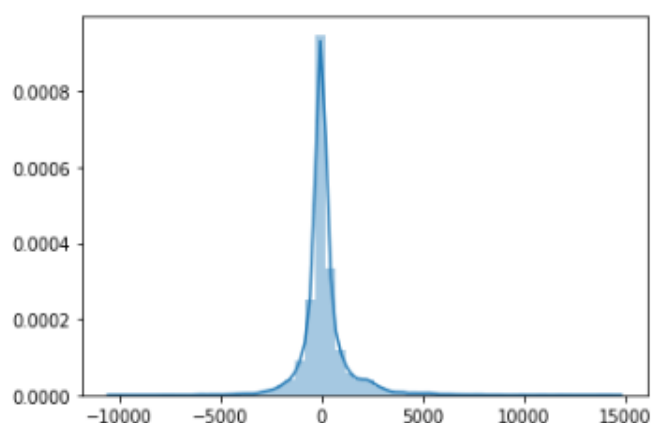
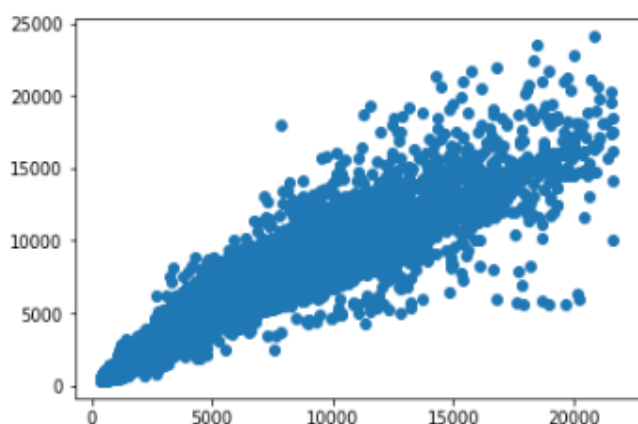


Note that this is a comparison between logarithm of prices and predictions → See the code again. It is written as `plt.scatter(y_test,y_pred)` after all features were converted to logarithm using `convertfeatures2log()`

This means to see actual discrepancies, we should un-logarithm it i.e. find the exponent of every price and prediction and then plot. The following code does this:

```
# convert prices and predictions back to exp
y_pred2 = np.exp(y_pred)
y_test2 = np.exp(y_test)
```

Now see the scatterplot and histogram again:



K-nearest neighbors (KNN)

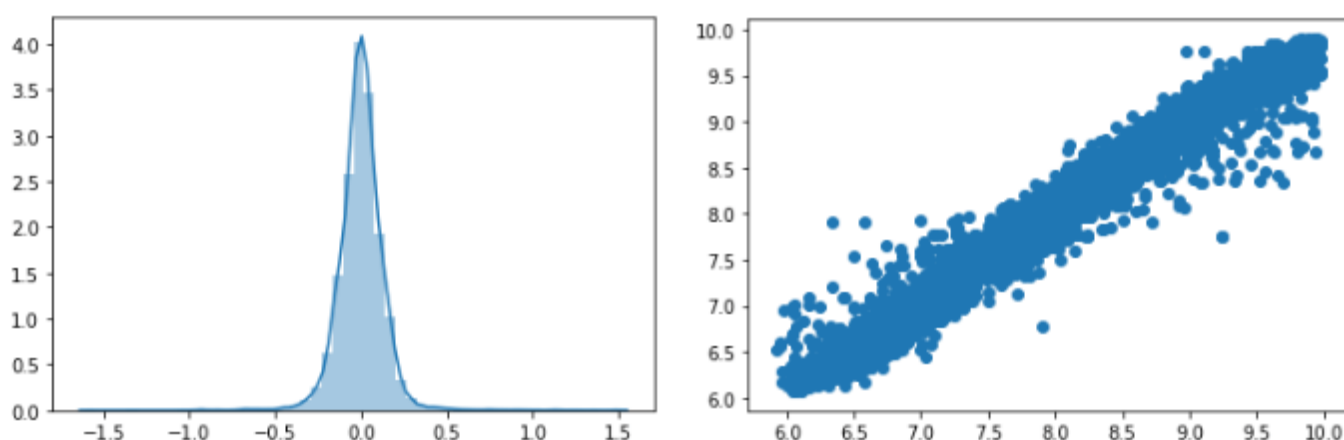
Because so many API libraries exist for several machine learning algorithms, the code for all simple machine learning algorithms is straightforward.



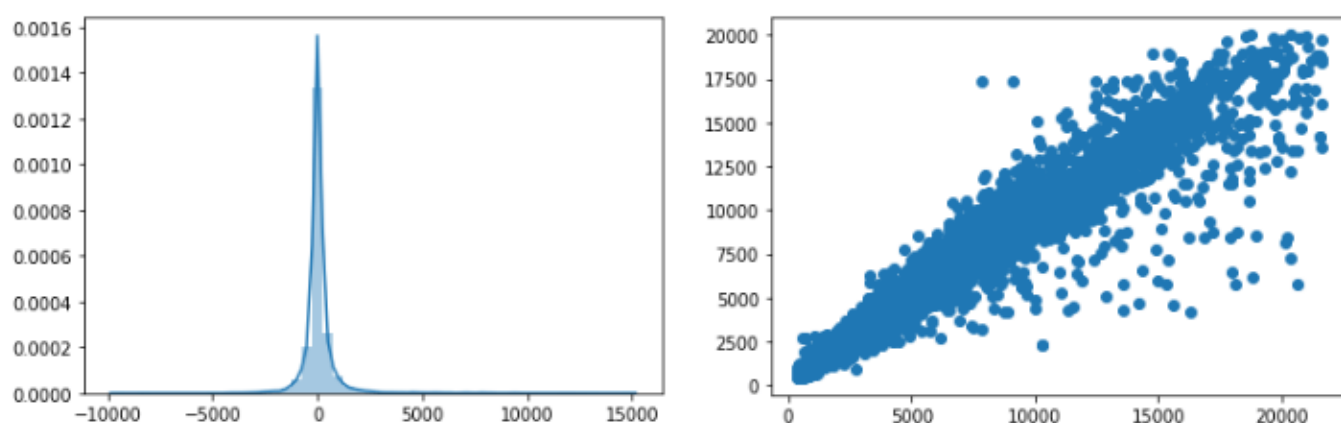
On the lines of linear regression code, we use sklearn library for KNN algo as well.

```
from sklearn.neighbors import KNeighborsRegressor
reg_all = KNeighborsRegressor(n_neighbors = 8, metric = 'minkowski', p = 2)
reg_all.fit(X_train,y_train)
y_pred=reg_all.predict(X_test)
```

The distplot and scatterplot for **logarithmic** features are as follows. You can confirm from values in x-axis and y-axis that y_test and y_pred are log here.



The distplot and scatterplot for **absolute values** of features are as follows. You can confirm from values in x-axis and y-axis that y_test and y_pred are **NOT** log here.





Get AWESOME articles directly in your email

[SIGN ME UP!](#)

Powered by 

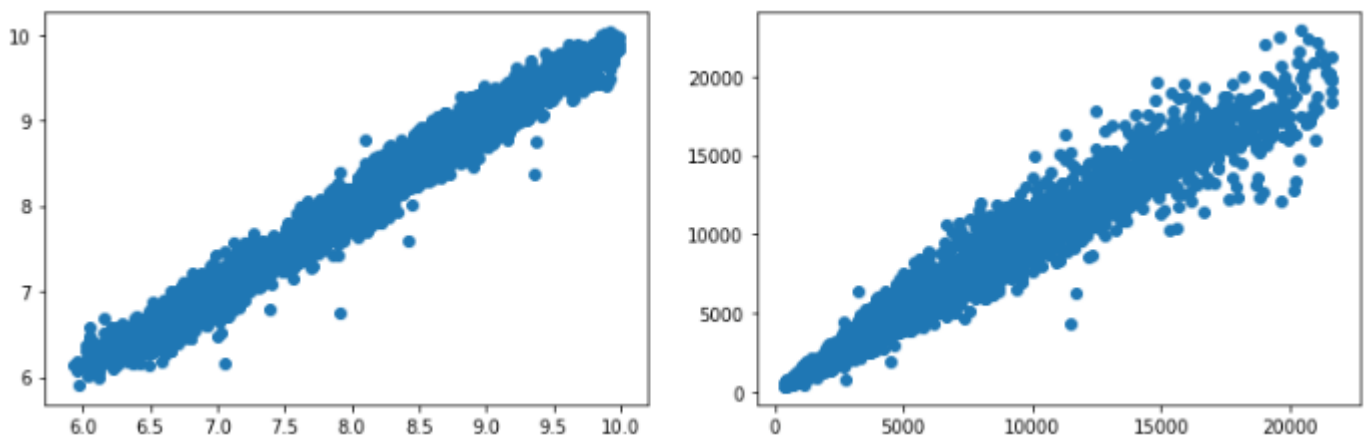


Support vector machines (SVM)

```
from sklearn.svm import SVR

regressor = SVR(kernel='rbf')
regressor.fit(X_train,y_train)
y_pred = regressor.predict(X_test)
```

Below is a comparison of scatterplots of log values of features (left plot) and absolute values (right plot) of features.



As of now, we can deduce that SVM is a better option because it gives better metrics score and a better scatterplot than Linear Regression and KNN

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error formula

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Mean Squared Error formula

$$\frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|^2$$



Root Mean Squared Error formula

$$\sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

The code to call functions related to all 3 evaluation metrics is mentioned below.

```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))

# Output for SVM is :
MAE: 0.08815796872409032
MSE: 0.012811091991743056
RMSE: 0.11318609451581522
```

Random Forest

Random forest is one of the most popular algorithms in most use cases / projects across industries. Its fast, easier to implement, needs lesser data, doesn't require extensive training and produces almost equally good results.

Again, because so many API libraries exist for several machine learning algorithms, the code for all simple machine learning algorithms is straightforward.

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators = 10)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

Now the metrics and their outputs

```
MAE: 0.08413000449351056
MSE: 0.012880789432555585
RMSE: 0.1134935655997977
```



It turns out that Random Forest is similar to the far slower SVM.

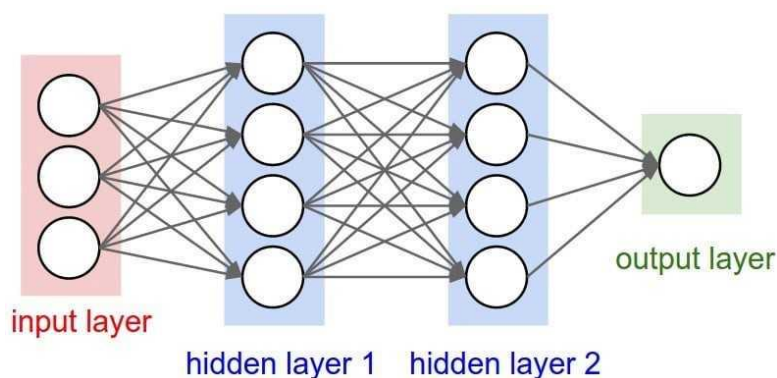
Other articles on www.fivestepguide.com that you may like:

Speed BONUS! Change domain nameservers for 5 Domain Registrars | Set nameservers to Cloudflare

Convert CSV rows To multiple JSON using Python & pandas in Jupyter notebook

\$3/month Best Cheap WordPress blog hosting to Start a blog or FREE with Bluehost.

The big daddy — Artificial neural networks



Generally neural networks or ANNs are more suited for classification problems requiring lots of complex logic / decision making and huge computations. They require larger datasets for their optimization to get the benefit of generalization and nonlinear mapping. But, if there's not enough data, a plain regression model may be better suited despite a few nonlinearities.

However, just for sake of completeness, I will show you how to predict diamond price (a regression problem) using ANNs.

You can read more about whether Neural networks are really needed for regression problems or not in the following webpage.

[Neural Networks for Regression](#)

And for ANN, the best, fastest and easiest to use and code library is Keras. Read more about Keras at their official website:



How to create an artificial neural network (ANN) using Keras

The step-wise process for creating an ANN to predict diamond prices is:

1. Construct a baseline Keras neural network model
2. Compile and return a Keras model using KerasRegressor,
3. Use the K-fold function to get perfect results
4. Fit the estimator to training data
5. Finally, predict the output (diamond prices in this case)
6. Evaluate the model using metrics between y-predicted vs. y-test

Now we go through each step in detail.

Firstly import the libraries for our project:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import RMSprop

from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
```

Next, we construct a `baseline_model()` function to create and return a Keras model

```
# define base model
def baseline_model():
    # create model
    model = Sequential()
    # add 1st layer
    model.add(Dense(output_dim=18, input_dim=11, activation='relu')) # kernel_initializer='normal',
    # add hidden layer
    model.add(Dense(output_dim=12, kernel_initializer='normal', activation='relu'))
    # add output layer
    model.add(Dense(1, kernel_initializer='normal'))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```



Then, we run the `KerasRegressor()` function, which returns a baseline ANN model built in Keras. It takes as input the model, epochs and batch-size. An **epoch** defines the number of times the learning algorithm will work through the entire training dataset to update the weights for a neural network. An epoch that has one batch is called the batch gradient descent learning algorithm. For batch training, all the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated. The **batch size** is a number of samples processed before the model is updated.

More information on `KerasRegressor` can be found at Tensorflow website:



Thereafter, we run the `KFold()` function. K-Fold Cross Validation uses a given data set, splits it into a K number of folds where each fold is used as a testing set while other K-1 are used as training set. For example, for 10-Fold cross validation (K=10), the dataset is split into 10 folds. In the first iteration, the first fold is used for validation while the 9 remaining folds form the training set. In the second iteration, 2nd fold is used as the testing set while the rest serve as the training set. This process is repeated until each fold of the 10 folds have been used as testing sets.

Subsequently, the `cross_val_score` function takes the model, X and y, and kfold's result as inputs and outputs multiple results—a list of regression model metrics scores. The `cross_val_score` function splits the data, using `KFold` as described above, into K pieces, trains on each combination of K-1 folds and gives back the metrics of the model.

```
1 estimator = KerasRegressor(build_fn=baseline_model, epochs=10, batch_size=5)
2 kf = KFold(n_splits=5)
3
4 results = cross_val_score(estimator, X_train, y_train, cv=kf)
5 print("Results: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

Brief information and steps on KFold and cross_val_score can be found in my other post; link below.

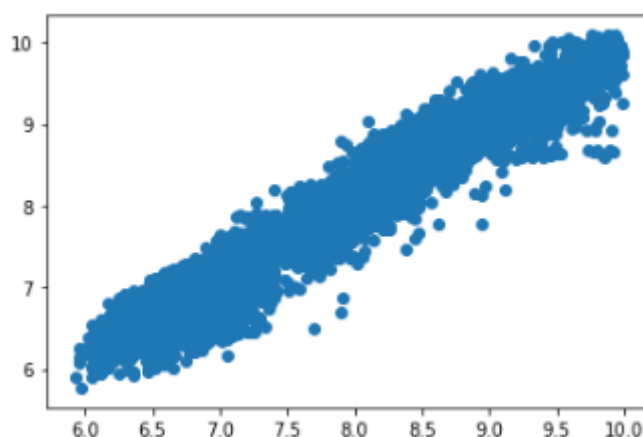
[Machine-learning-interview-notes-part-3-short-bullet-points](#)

Finally, we fit the estimator to our training data to get predictions.



```
estimator.fit(X_train, y_train)
y_pred = estimator.predict(X_test)# Plot a scatter plot like above to see prediction perfection
plt.scatter(y_test,y_pred)
```

The scatterplot output for log(features) is as below. It turns out that it is not as bad as we expected when I said that ANNs are mostly used for classification, not regression. You can yourself decide if you would like to invest resources in ANN deployment simply to predict diamond prices.



In the next page we will look at real-life example of ML models for prediction using data from Pricescope.com.

References

For this particular post, I have referred to several websites including [Beyond4Cs](#), [Machine Learning Mastery](#) and others few posts and websites on internet.

The same post was first written on Medium.com. Please [click here](#) for part 1, and [here](#) for part 2 of my initial two posts on medium.

Thanks for reading this post. I have written several other such Machine Learning related articles on this website. If you liked this post, kindly comment and like using the comment form below.



Frequently Asked Questions (FAQ)

What are different types of Machine Learning?

This is one of the very first questions by readers of any guide on Machine Learning basics. Generally speaking, there are 3 types of machine learning:

1. Supervised Learning, where an ML model makes predictions based on known or tagged or labeled data (data with tags or labels, thereby seemingly more meaningful)
2. Unsupervised Learning, where there's no labeled data. An ML model here identifies patterns, relationships and anomalies to predict the outcome.
3. Reinforcement Learning, where the ML model for prediction learns continually based on the output it predicts and the rewards it receives for its previous actions.

What is the process followed to create ML models for prediction?

One of the first steps in Machine Learning process is to understand the business requirements and then identify the ML model. Then, a very simple 3-step machine learning basic process is followed to create ML models for prediction:

1. Train the model: Split the entire data to be used to predict diamond prices into *train* and *test* data using [train-test-split](#), or any other method. The train data is run on the agreed ML model for prediction. The model is tweaked regularly by editing model parameters unless accuracy, precision and recall and other tests match your expectations.
2. Test the model: Once the output is up to satisfactory levels, you use the test data to check if the model predicts with agreed accuracy. This would help determine if training is effective. On errors, change your ML model for prediction or retrain it with more data.
3. Deploy the model: Finally, upon several rounds of testing and training, once the model reaches your levels of expectations, you deploy the model into production.

What is bias and variance?

Bias is the distance of predicted values (\hat{y}) from the actual values (y). Bias is High when the average predicted values are far from the actual values.

Variance is high when the model performs good on the training set but not on test set. Variance shows how scattered are the predicted values from actual values.



I am so confused on a confusion matrix. Wait! What's it after all?

A Confusion matrix is used for evaluating the performance not for ML models for prediction but for classification. It is an 2×2 matrix which compares the actual values with those predicted by the machine learning model. It is the first step in machine learning basics on performance evaluation. Furthermore, it helps tell you how well the ML classification model is performing and the errors it is making.

True Positive (TP) : The predicted value matches the actual value. E.g. actual value = positive, predicted value = positive

True Negative (TN): The predicted value matches the actual value. E.g. actual value = *negative*, predicted value = *negative*

False Positive (FP), also called Type 1 error: Wrong prediction. E.g. actual value = negative , predicted value = positive

False Negative(FP), also called Type 2 error: Wrong prediction. E.g. actual value = positive, predicted value = negative

Did I miss Semi-supervised Machine Learning? What's that?

Supervised learning uses labeled data while unsupervised learning uses no training data.

The third type, semi-supervised learning uses a small amount of labeled data and a large amount of unlabeled data.

Precision and Recall formula in 10 seconds!

Another important item falling under Machine Learning basics.

Precision = (True Positive) / (True Positive + False **Positive**)

Recall = (True Positive) / (True Positive + False **Negative**)

What's k-fold cross validation?

K-fold cross validation is a procedure used to estimate the skill of the model on unseen data.

K = number of groups your data is split into.

K-fold cross validation procedure:

1. Shuffle the dataset randomly by splitting the dataset into k groups (10 groups)
2. For each unique group: (for i = 1 to 10)
 - a) Take the group as test data set (train_test_split = 10% test & 90% train) + remaining groups as a training data set



- b) Fit a model on the training set and evaluate it on this test set
 - c) Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Other resources - www.fivestepguide.com

Thank you for downloading this article.

You may visit the [Website metrics section](#) where I discuss nuances of WordPress websites, how to make websites faster, how to COPY my formula to speed up your website. You may click below to visit any of the best articles.



Quick and easy 5-steps of Backward Elimination in Machine Learning (with Python code)

📅 January 10, 2021 • 💬 0

This quick 5-step guide will describe Backward Elimination in machine learning, an advanced technique for feature selection. It basically helps you select optimal number of



5-step Machine Learning model to predict car mileage using UCI dataset

📅 January 8, 2021 • 💬 0

The mission of this Five Step Guide article is to teach you how to create a machine learning model to predict car mileage using UCI



Convert CSV rows To multiple JSON using Python & pandas in Jupyter notebook

📅 January 6, 2021 • 💬 0

Have you tried to search the internet for a simple solution to your problem but not found a suitable one and so started exploring and



If you buy any products or subscribe to any services using the links on this page, your gesture will be a huge support in maintenance and running of this website through affiliate commissions to the website owner.

Click the image below to subscribe to my latest articles on <https://fivestepguide.com>



Subscribe to receive latest articles hot off the press

Name:*

Enter your email here:*

☒ I agree to be added to the mailing list

Subscribe Now!

We hate spam. You can unsubscribe anytime.
GDPR data protection – your data will never be sold.

If you liked this article and would like to read many more such articles, **please subscribe to newsletter** by www.fivestepguide.com