Diamond Prices Prediction with Deep learning

Rangsarid Pringwanid ID 6210422038

Agenda

I. Business Purpose

- Business Purpose

II. Data Collections

- Data Collections
- Data Pre-processing

III. Exploratory Analysis

- Pair plot
- Correlation Graph

IV. Price Prediction

- Modeling
- Model Tuning
- Result Performance

Business Purpose

Can analyze diamonds by their cut, color, clarity, price, and other attributes .And prediction the diamond price from there attributed by Deep learning model.

Data Collection

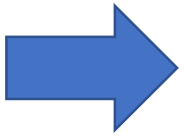
Attributed

#	Attribute
1.	**Price (Predicted)**
2.	Carat weight
3.	Quality of the cut (Fair, Good, Very Good, Premium, Ideal)
4.	Color diamond
5.	Clarity a measurement
6.	x length in mm (0--10.74)
7.	y width in mm (0--58.9)
8.	z depth in mm (0--31.8
9	depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)
10	table width of top of diamond relative to widest point (43--95)

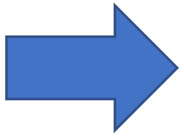
Data PipeLine

Almost 54,000 Diamonds

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39



ANN
Activation = Linear



Price [USD]

Data Pre-Processing

Data Pre-Processing

1. Remove Duplicated

Check Duplicated

```
[7]
## Check duplicated
df_diamond_price.duplicated().sum()
```

```
146
```

Removed Dupplicated Valued

```
[8]
## Remove duplicated
df_diamond_price = df_diamond_price.loc[~df_diamond_price.duplicated()]
```

```
df_diamond_price.duplicated().sum()
```

```
0
```

No Duplicated

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

Data Pre-Processing (2)

2. Remove Dimension X=0 or Y=0 or Z=0

Impossible

Check Zero Dimension

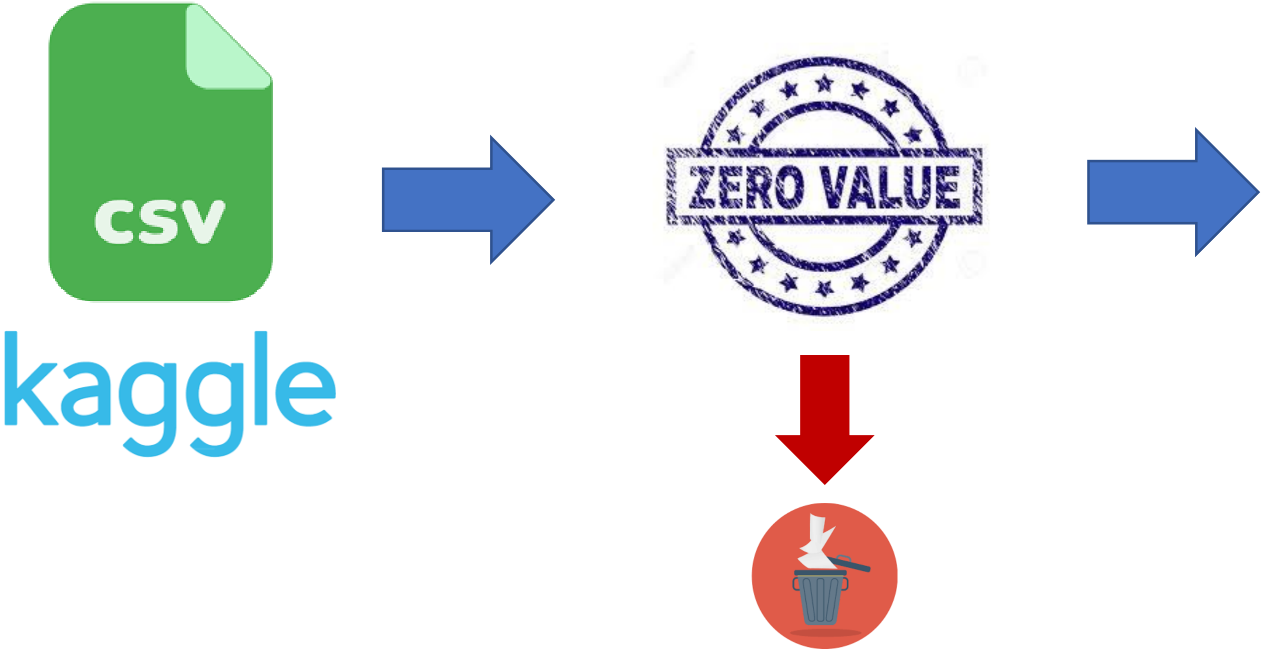
```
[14]
## check x,y,z = 0  (high,wide,lengh) is not possible with  actually dimension
check_zero =  ((df_diamond_price.x == 0 )| (df_diamond_price.y == 0) | (df_diamond_price.z == 0))
check_zero.sum()
```

```
19
```

Remove Zero Dimension

```
### Remove zero dimension
df_diamond_price =  df_diamond_price.loc[~check_zero]
df_diamond_price.shape
```

```
(53775, 10)
```



No Dimension = 0

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
6	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39

Data Pre-Processing (3)

Crated dummpy with category variable

```
## crated dummpy with  category variable
df_diamond_price = pd.get_dummies(data = df_diamond_price,drop_first = True)
df_diamond_price.head(10)
```

3. **Created dummy variable with category feature**

	carat	depth	table	x	y	z	price	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	color_E	color_F	color_G	color_H	color_I	color_J	clarity_IF	clarity_SI1	clarity_SI2	clarity_VS1	clarity_VS2	clarity_VVS1	clarity_VVS2
1	0.23	61.5	55.0	3.95	3.98	2.43	326	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
2	0.21	59.8	61.0	3.89	3.84	2.31	326	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
3	0.23	56.9	65.0	4.05	4.07	2.31	327	1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
4	0.29	62.4	58.0	4.20	4.23	2.63	334	0	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0
5	0.31	63.3	58.0	4.34	4.35	2.75	335	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
6	0.24	62.8	57.0	3.94	3.96	2.48	336	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1
7	0.24	62.3	57.0	3.95	3.98	2.47	336	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0
8	0.26	61.9	55.0	4.07	4.11	2.53	337	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0
9	0.22	65.1	61.0	3.87	3.78	2.49	337	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0
10	0.23	59.4	61.0	4.00	4.05	2.39	338	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	0

Data Pre-Processing (4)

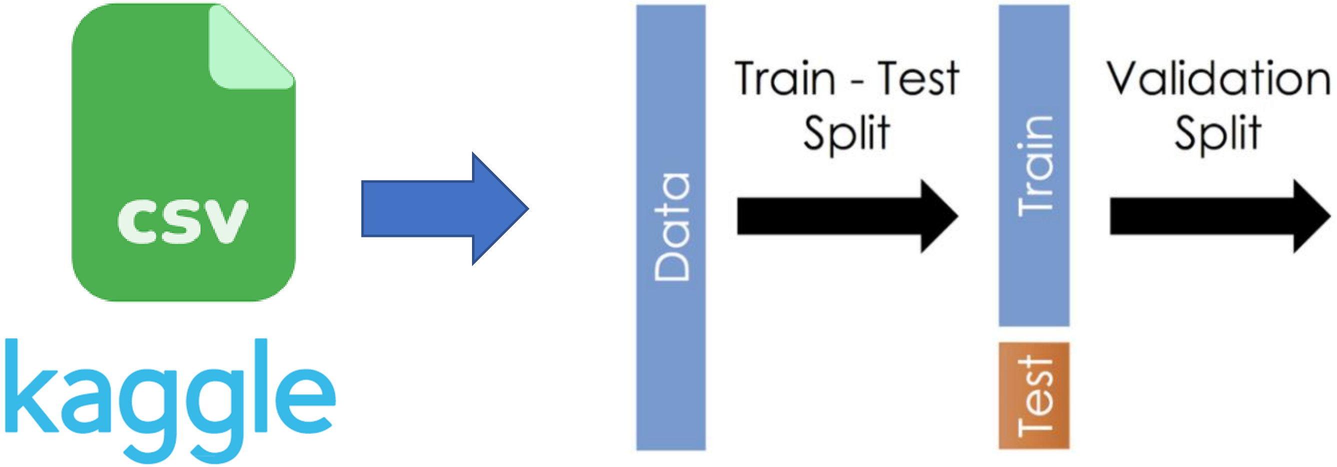
4. **Data Train Test Split**
At Test set = 0.2

Train Test split data

```
## Train test split
x= df_diamond_price.drop('price',axis = 1)
y = df_diamond_price.loc[:,'price']

```

```
[17]  X_train,X_test, y_train ,y_test = train_test_split(x,y,test_size = 0.2, random_state = 42)
```




```
print('Train_Data :',X_train.shape[0])
print('Test_Data:',X_test.shape[0])
```

```
Train_Data : 43020
Test_Data: 10755
```

Data Pre-Processing (5)

5. Data Standardize

Standardize and normalize

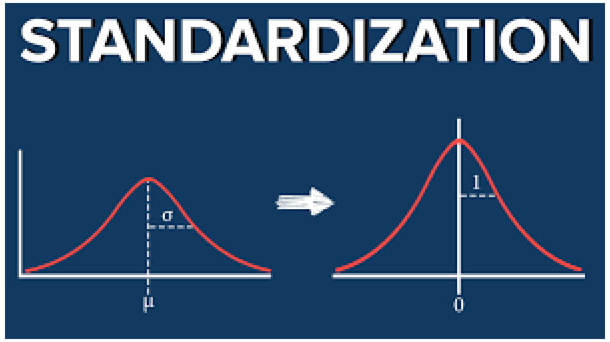
```
sc = StandardScaler()
#X_train = sc.fit_transform(X_train)
X_train_scaled =  sc.fit_transform(X_train.loc[:,['carat','depth','table','x','y','z']])
X_train_scaled = pd.DataFrame(X_train_scaled,columns=['carat','depth','table','x','y','z'],index=X_train.index)

X_test_scaled =  sc.transform(X_test.loc[:,['carat','depth','table','x','y','z']])
X_test_scaled = pd.DataFrame(X_test_scaled,columns=['carat','depth','table','x','y','z'],index=X_test.index)


X_train_scale_final = X_train.copy()
X_test_scale_final = X_test.copy()


X_train_scale_final.loc[:,['carat','depth','table','x','y','z']] = X_train_scaled.loc[:,['carat','depth','table','x','y','z']]
X_test_scale_final.loc[:,['carat','depth','table','x','y','z']] = X_test_scaled.loc[:,['carat','depth','table','x','y','z']]

```

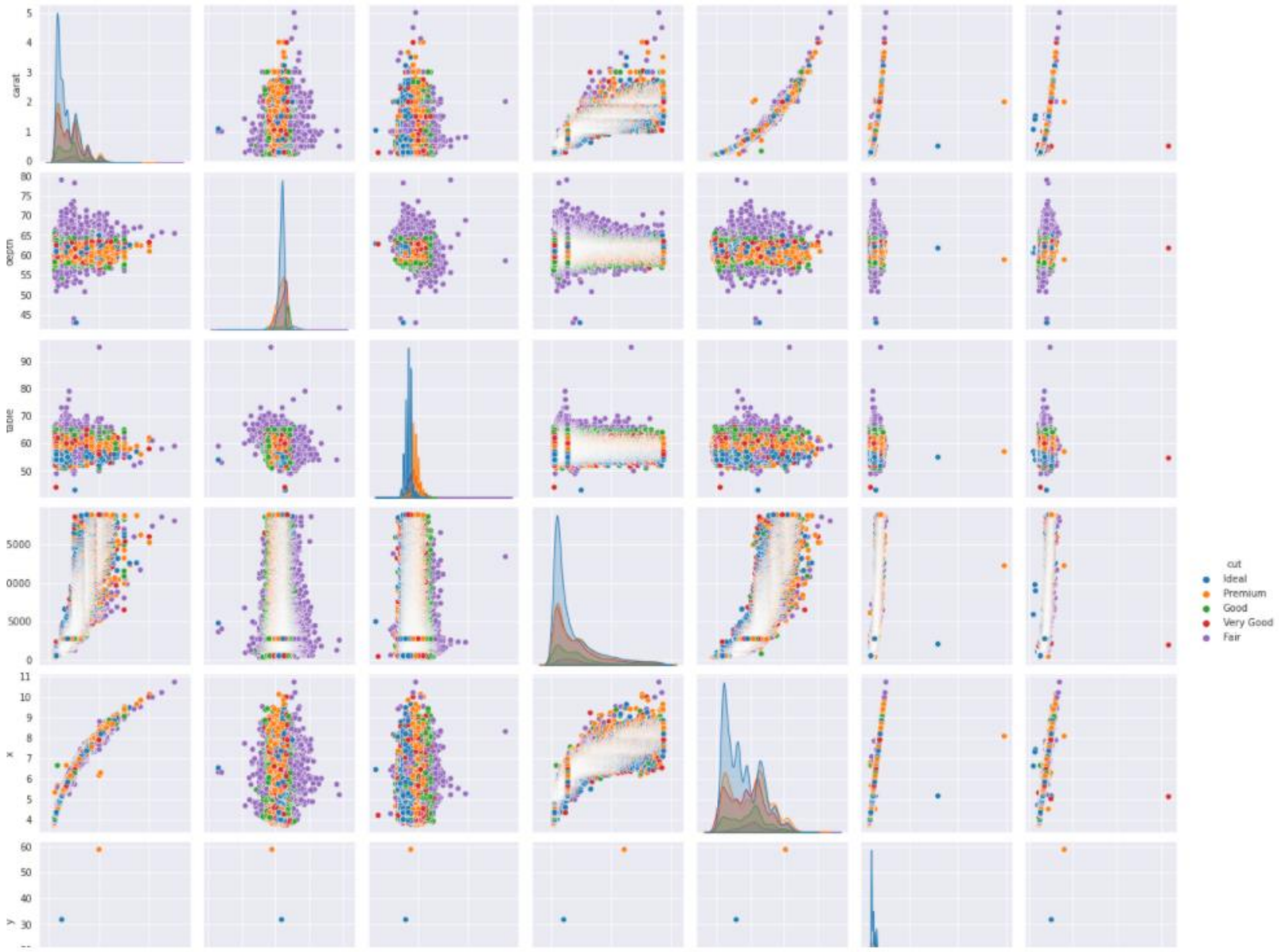


	carat	depth	table	x	y	z	cut_Good	cut_Ideal	cut_Premium	cut_Very Good	color_E	color_F
32547	-1.113335	0.105580	-0.203777	-1.385122	-1.380692	-1.360812	0	1	0	0	0	0
2677	-0.205132	-0.174777	-1.097919	-0.027046	0.005198	-0.027459	0	0	0	1	0	0
22910	1.505669	-1.085937	1.137436	1.581202	1.434670	1.334263	0	0	1	0	0	0
12024	0.660829	0.876562	-0.650848	0.786013	0.728650	0.866171	0	1	0	0	0	0
16371	-0.986609	-0.945758	1.137436	-1.179624	-1.110487	-1.218966	0	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
17217	1.167733	-0.034598	0.690365	1.179140	1.181897	1.164048	0	0	0	1	0	0
52924	-0.458584	-0.174777	-0.650848	-0.321892	-0.291155	-0.325336	0	1	0	0	0	0
35279	-0.627552	0.385937	-1.097919	-0.589933	-0.543928	-0.523920	0	1	0	0	0	0
31970	-1.134456	0.385937	-0.650848	-1.474469	-1.424273	-1.403366	0	1	0	0	0	1
29612	-1.113335	-0.174777	-0.203777	-1.385122	-1.319678	-1.360812	0	0	0	1	0	0

Exploratory Data Analysis (EDA)

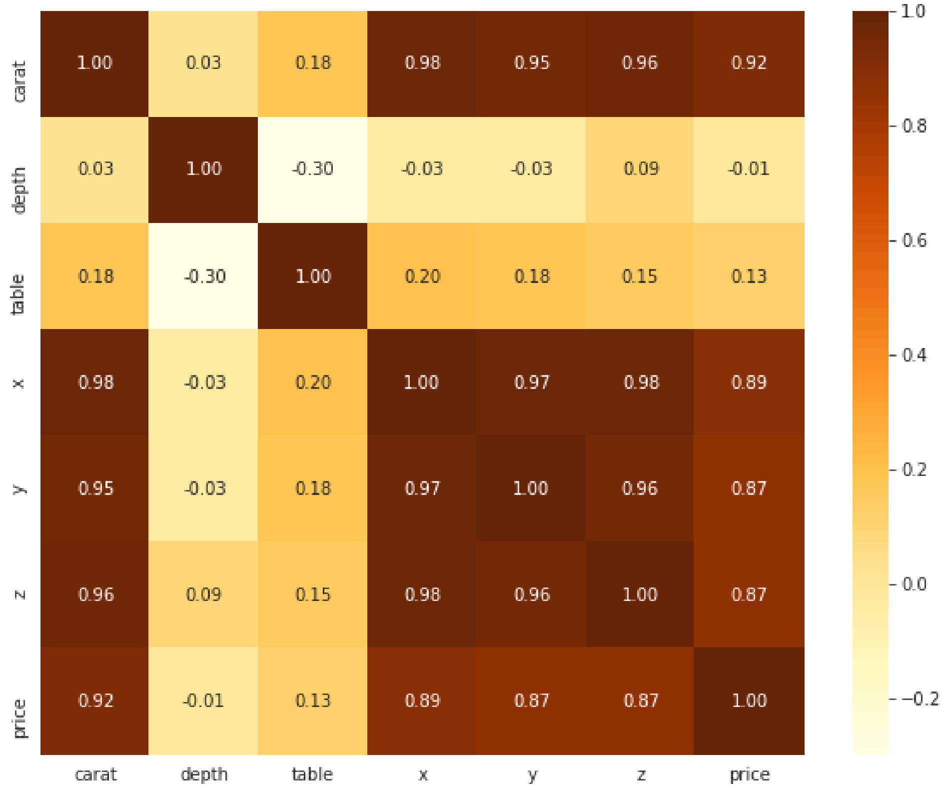
Seaborn Plot

ดูการกระจายตัวของความสัมพันธ์ระหว่าง feature ต่างๆๆ ที่จะใช้ใน model ในการ prediction



Exploratory Data Analysis (EDA)

ค่าความสัมพันธ์ของ feature
(Correlation plot)



Feature Statistics

	carat	depth	table	price	x	y	z
count	53794.00000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000	53794.000000
mean	0.79778	61.748080	57.458109	3933.065082	5.731214	5.734653	3.538714
std	0.47339	1.429909	2.233679	3988.114460	1.120695	1.141209	0.705037
min	0.20000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.40000	61.000000	56.000000	951.000000	4.710000	4.720000	2.910000
50%	0.70000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.04000	62.500000	59.000000	5326.750000	6.540000	6.540000	4.030000
max	5.01000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

Modeling

```
## Model building
model = Sequential()
model.add(Dense(512,activation='relu',input_dim= X_train_scale_final.shape[1]))
model.add(Dense(256,activation='relu'))
model.add(Dense(128,activation='relu'))
model.add(Dense(64,activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(1,activation='linear'))
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 512)               12288
_________________________________________________________________
dense_16 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_17 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_18 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_19 (Dense)             (None, 1)                 65
=================================================================
Total params: 184,833
Trainable params: 184,833
Non-trainable params: 0
_________________________________________________________________
```

Parameter set :

Input layer = 23 Nodes
Hidden layer = 4 layers
Output layers = 1 layer by activation = linear

Remark : Prediction with linear Regression

Modeling Parameter Tuning

Define Parameter

```
model.compile(optimizer='adam', loss = 'mse')
```

```
model_fit = model.fit(x = X_train_scale_final,y = y_train,validation_data=(X_test_scale_final,y_test),epochs = 200,batch_size=512)
```

```
85/85 [==============================] - 2s 19ms/step - loss: 196552.8281 - val_loss: 300338.8750
Epoch 172/200
85/85 [==============================] - 2s 19ms/step - loss: 199845.0469 - val_loss: 288701.1875
Epoch 173/200
85/85 [==============================] - 2s 19ms/step - loss: 194220.9688 - val_loss: 309358.5000
Epoch 174/200
85/85 [==============================] - 2s 19ms/step - loss: 200583.7188 - val_loss: 336822.7188
Epoch 175/200
85/85 [==============================] - 2s 19ms/step - loss: 205837.3125 - val_loss: 316562.1875
Epoch 176/200
85/85 [==============================] - 2s 19ms/step - loss: 208423.4219 - val_loss: 336710.8750
Epoch 177/200
85/85 [==============================] - 2s 18ms/step - loss: 210855.0469 - val_loss: 304388.3750
Epoch 178/200
85/85 [==============================] - 2s 18ms/step - loss: 205068.1562 - val_loss: 297402.2812
Epoch 179/200
85/85 [==============================] - 2s 18ms/step - loss: 202149.4531 - val_loss: 346206.9375
Epoch 180/200
85/85 [==============================] - 2s 19ms/step - loss: 199464.4844 - val_loss: 294194.0000
Epoch 181/200
85/85 [==============================] - 2s 18ms/step - loss: 199140.5625 - val_loss: 412933.4375
Epoch 182/200
85/85 [==============================] - 2s 18ms/step - loss: 249469.8281 - val_loss: 337596.1875
Epoch 183/200
85/85 [==============================] - 2s 19ms/step - loss: 224355.4688 - val_loss: 288413.7188
Epoch 184/200
85/85 [==============================] - 2s 19ms/step - loss: 194559.2344 - val_loss: 289059.8750
Epoch 185/200
85/85 [==============================] - 2s 18ms/step - loss: 193231.3594 - val_loss: 296385.7500
Epoch 186/200
85/85 [==============================] - 2s 19ms/step - loss: 197051.0938 - val_loss: 298700.3750
Epoch 187/200
85/85 [==============================] - 2s 18ms/step - loss: 197122.6406 - val_loss: 335064.9062
Epoch 188/200
85/85 [==============================] - 2s 20ms/step - loss: 205347.7812 - val_loss: 402392.3438
Epoch 189/200
85/85 [==============================] - 2s 19ms/step - loss: 225989.1094 - val_loss: 307359.4688
Epoch 190/200
```

Parameter

Epochs : 200

Batch_size : 512

Optimizer : Adam

Loss : MSE

Grid_Search Parameter

```
model_fit2 = model.fit(x = X_train_scale_final,y = y_train,validation_data=(X_test_scale_final,y_test),epochs = 50,batch_size=32)
```

```
1345/1345 [==============================] - 7s 5ms/step - loss: 303965.5179 - val_loss: 315619.9688
Epoch 22/50
1345/1345 [==============================] - 7s 5ms/step - loss: 298835.6172 - val_loss: 329825.1250
Epoch 23/50
1345/1345 [==============================] - 7s 5ms/step - loss: 293561.8969 - val_loss: 349734.2500
Epoch 24/50
1345/1345 [==============================] - 7s 5ms/step - loss: 304414.0411 - val_loss: 351488.1562
Epoch 25/50
1345/1345 [==============================] - 7s 5ms/step - loss: 296151.3997 - val_loss: 303639.4375
Epoch 26/50
1345/1345 [==============================] - 7s 5ms/step - loss: 298540.3267 - val_loss: 295214.9688
Epoch 27/50
1345/1345 [==============================] - 7s 5ms/step - loss: 293356.9518 - val_loss: 302574.0625
Epoch 28/50
1345/1345 [==============================] - 7s 5ms/step - loss: 301724.2036 - val_loss: 278042.7500
Epoch 29/50
1345/1345 [==============================] - 7s 5ms/step - loss: 287963.4724 - val_loss: 295011.0000
Epoch 30/50
1345/1345 [==============================] - 7s 5ms/step - loss: 292811.1778 - val_loss: 296739.2188
Epoch 31/50
1345/1345 [==============================] - 7s 5ms/step - loss: 280933.5151 - val_loss: 281776.9375
Epoch 32/50
1345/1345 [==============================] - 7s 5ms/step - loss: 281066.6895 - val_loss: 349147.0938
Epoch 33/50
1345/1345 [==============================] - 7s 5ms/step - loss: 282244.8484 - val_loss: 290895.5625
Epoch 34/50
1345/1345 [==============================] - 7s 5ms/step - loss: 289366.2306 - val_loss: 299167.9062
Epoch 35/50
1345/1345 [==============================] - 7s 5ms/step - loss: 278462.6897 - val_loss: 289669.8125
Epoch 36/50
1345/1345 [==============================] - 7s 5ms/step - loss: 266161.8125 - val_loss: 321050.7500
Epoch 37/50
1345/1345 [==============================] - 7s 5ms/step - loss: 270079.1213 - val_loss: 332894.5000
Epoch 38/50
1345/1345 [==============================] - 7s 5ms/step - loss: 271290.2992 - val_loss: 295081.2812
Epoch 39/50
1345/1345 [==============================] - 7s 5ms/step - loss: 261238.4909 - val_loss: 298384.3438
Epoch 40/50
1345/1345 [==============================] - 7s 5ms/step - loss: 278551.7964 - val_loss: 339508.7500
Epoch 41/50
1345/1345 [==============================] - 7s 5ms/step - loss: 266720.0340 - val_loss: 282967.0312
Epoch 42/50
1345/1345 [==============================] - 7s 5ms/step - loss: 272602.2274 - val_loss: 285479.7500
```

Parameter

Epochs : 50

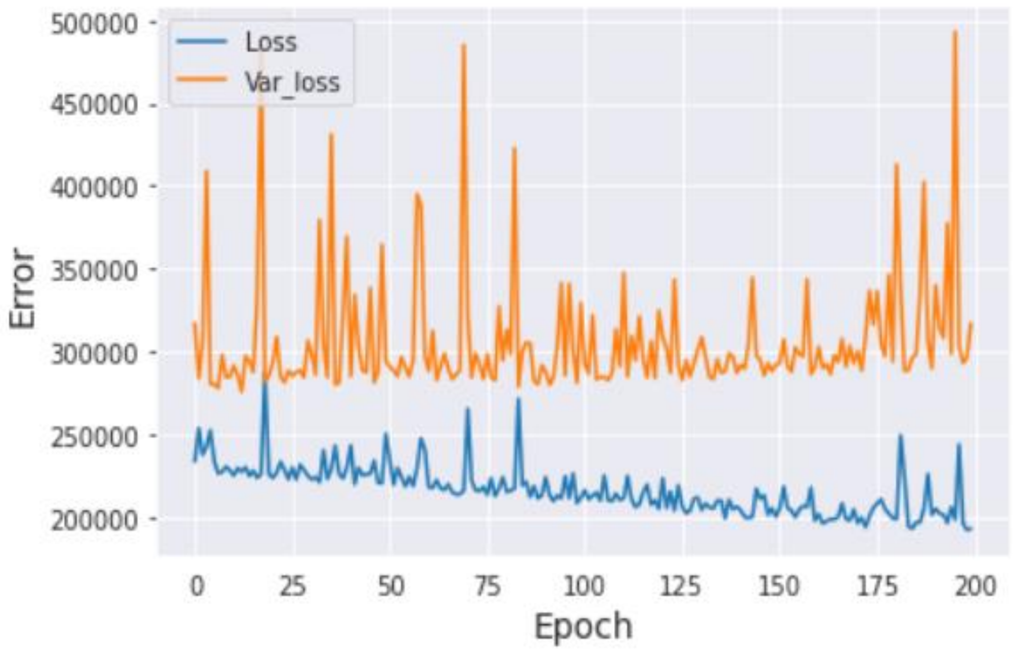
Batch_size : 32

Optimizer : Adam

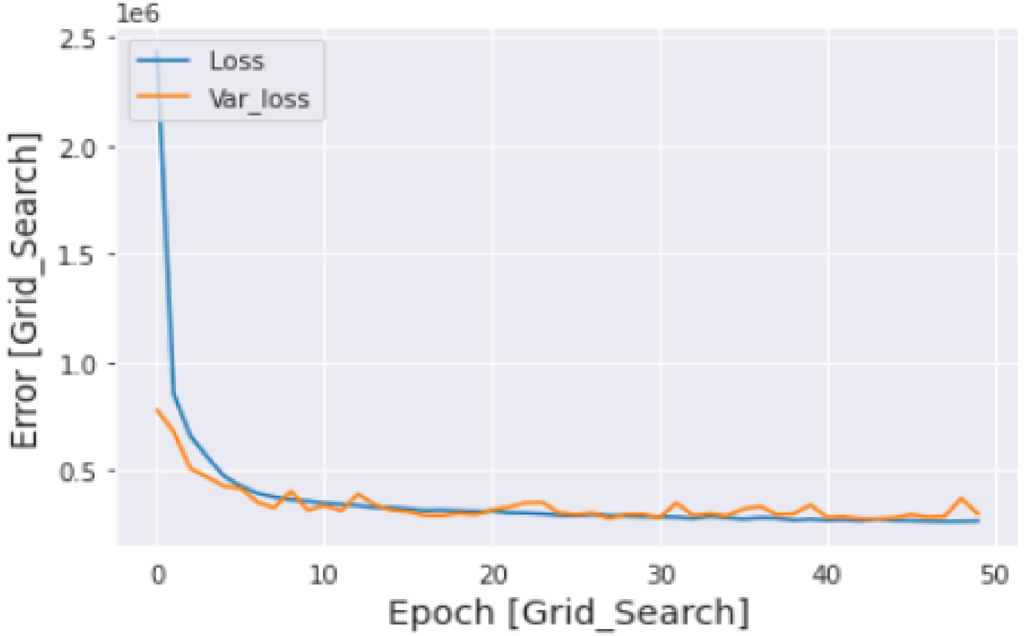
Loss : MSE

Model Tuning (Visual model Performance)

Define parameter

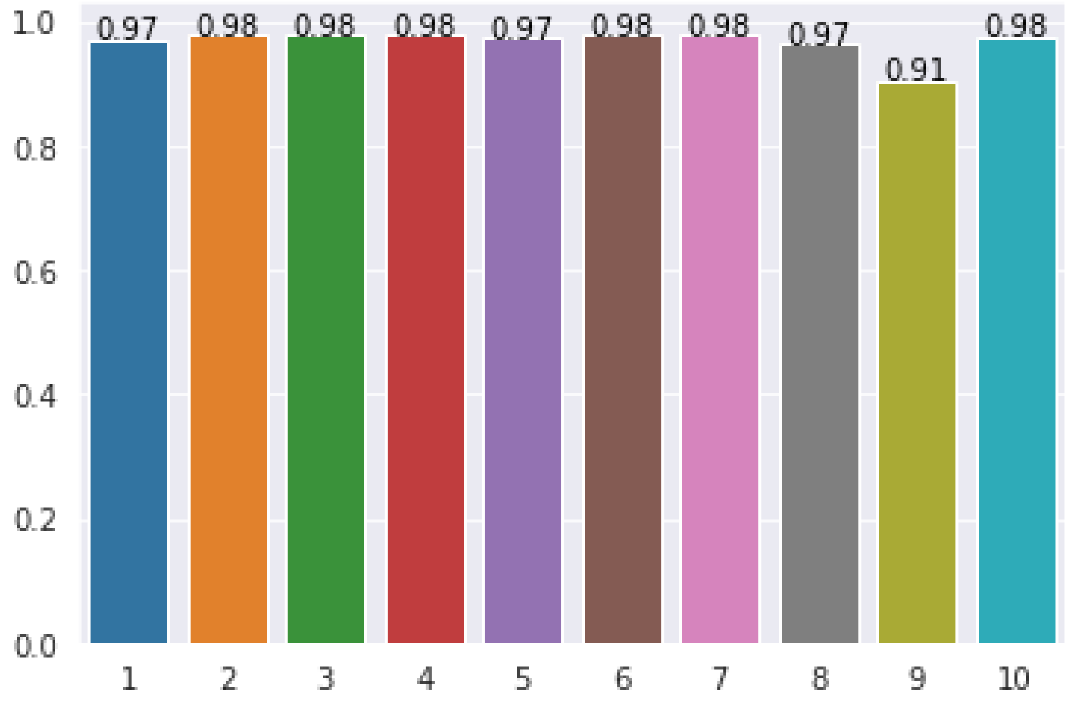


Grid Search Parameter



Model Result Cross validation

K = 10




```
from keras.wrappers.scikit_learn import KerasRegressor
from sklearn.model_selection import train_test_split,cross_val_score
print((X_test_scale_final.shape)[1])
def build_linear_cross() :
  clf = Sequential()
  clf.add(Dense(512,activation='relu',input_dim= list(X_train_scale_final.shape)[1]))
  clf.add(Dense(256,activation='relu'))
  clf.add(Dense(128,activation='relu'))
  clf.add(Dense(64,activation='relu'))
  clf.add(Dense(1,activation='linear'))
  clf.compile(optimizer = 'adam', loss='mse', metrics=['mean_squared_error'])
  return clf
clf =  KerasRegressor(build_fn=build_linear_cross, batch_size=32, epochs=50)
accuracies = cross_val_score(estimator = clf, X = X_train, y = y_train, scoring='r2', cv = 10, n_jobs = 1)
```

```
mean = accuracies.mean()
std = accuracies.std()
print(f'Mean: {mean}')
print(f'Variance: {std*std}')
```

```
Mean: 0.968542231632286
Variance: 0.00045592895427757236
```

Parameter set

- Optimizer: Adam
- Epochs : 50
- Batch_size : 32
- Loss : MSE

Mean : 0.968
Variance: 0.00045

Model Performance (MSE & R2 Score)

- Grid Search Approach

```
print('*'*15,'Grid Search Parameter','*'*15)
print(f"Mean Square Error Deep_learning : {mean_squared_error(y_test,y_pred_GSD_F) ** 0.5}")
print(f'R2 Score :{r2_score(y_test, y_pred_GSD_F)}' )
```

```
*************** Grid Search Parameter ***************
Mean Square Error Deep_learning : 545.5787177481428
R2 Score :0.98116485769992
```

MSE : 545.578 (USD)
R2 Score: 0.981

- Define Parameter Approach

```
print('*'*15,'Specify Parameter DNN Model','*'*15)
print(f"Mean Square Error Deep_learning : {mean_squared_error(y_test,y_pred) ** 0.5}")
print(f'R2 Score :{r2_score(y_test, y_pred)}' )
```

```
*************** Specify Parameter DNN Model ***************
Mean Square Error Deep_learning : 563.0169549703736
R2 Score :0.9799415665761724
```

MSE : 563.017 (USD)
R2 Score : 0.979

- Linear Regression model Approach

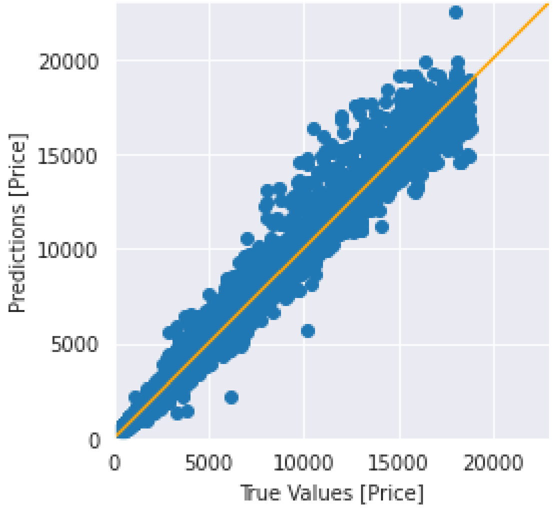
```
print('*'*15,'Linear Regression Model','*'*15)
print(f"Mean Square Error LinearRegression : {rmse}")
print(f'R2 Score : {r2_score(y_test, y_pred_1)}' )
```

```
*************** Linear Regression Model ***************
Mean Square Error LinearRegression : 1128.9110798751835
R2 Score : 0.9193557262563594
```

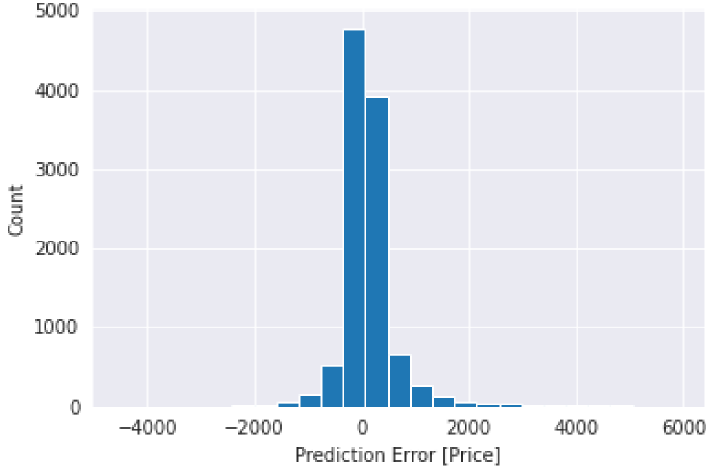
MSE : 1128.911 (USD)
R2 Score : 0.919

Model Result Actual VS Prediction Price

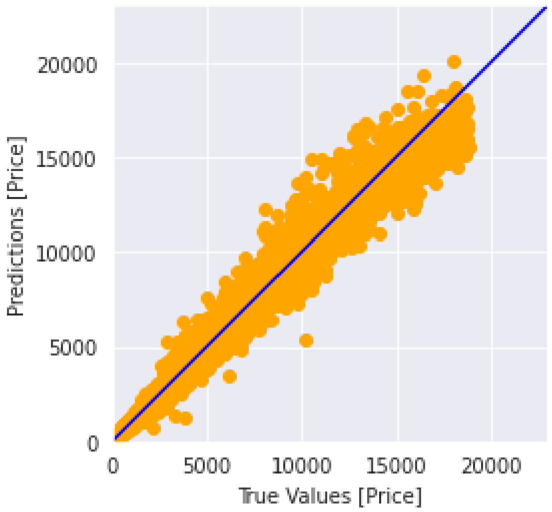
Define Parameter



Error Distribution



Grid Search



Error Distribution

Result Grid_Seach VS Define Parameter

Define Parameter

Specify parameter base approch

	Actual(USD)	Predict(USD)	Difference(USD)
5	335	-3421.744962	3756.744962
7	336	-343.741918	679.741918
8	337	-1003.403383	1340.403383
32	402	387.078570	14.921430
35	402	635.394403	233.394403
39	403	471.698352	68.698352
41	403	-2017.108534	2420.108534
42	403	-1945.199633	2348.199633
50	404	-1971.912825	2375.912825
53	404	-1076.405874	1480.405874

Grid_Seach

Grid Search approch

	Actual(USD)	Predict(USD)	Difference(USD)
5	335	336.074219	1.074219
7	336	438.114868	102.114868
8	337	393.332733	56.332733
32	402	409.774780	7.774780
35	402	343.094269	58.905731
39	403	400.464264	2.535736
41	403	496.129517	93.129517
42	403	494.555634	91.555634
50	404	373.603577	30.396423
53	404	508.685883	104.685883

Summary

- จากการทำนายราคาของเพชรจากข้อมูล feature ต่างๆ พบว่า Model (DNN) ที่ได้จากการทำ Grid Search Parameter tuning ให้ค่า MSE และ R2_Score

ที่ดีที่สุดคือ MSE = 545.578 (USD) และ ค่า R2_Score = 0.981 (98.1 %) โดยได้

Hyper Parameter คือ Optimizer = Adam, Epochs = 50, Batchs Size = 32

THANK YOU