In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import *
from sklearn.feature_selection import *
from sklearn.model_selection import *
from sklearn.linear_model import *
from sklearn.metrics import *
import joblib
import tensorflow as tf
import os
import warnings
warnings.filterwarnings("ignore")
```

```
WARNING:tensorflow:From C:\Users\amith\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\losses.py:
2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cr
oss_entropy instead.
```

Reading the Dataset

In [2]:
```
df = pd.read_csv("diamonds.csv")
df.head()
```

Out[2]:

	Unnamed: 0	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	...	meas_depth	girdle_min
0	0	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	...	1.79	M
1	1	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Very Good	unknown	...	1.78	STK
2	2	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	...	1.77	TN
3	3	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	...	1.78	M
4	4	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Excellent	unknown	...	1.82	STK

5 rows × 26 columns

Data Preprocessing

In [3]:
```
df_1 = df.drop(columns="Unnamed: 0")
df_1.head()
```

Out[3]:

	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	culet_size	...	meas_depth	girdle_min
0	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	N	...	1.79	M
1	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Very Good	unknown	N	...	1.78	STK
2	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	unknown	...	1.77	TN
3	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	unknown	...	1.78	M
4	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Excellent	unknown	N	...	1.82	STK

5 rows × 25 columns

In [4]:
```
(df_1.isna().sum() * 100 / len(df_1)).round(2)
```

Out[4]:
```
cut                             0.00
color                           0.00
clarity                         0.00
carat_weight                    0.00
cut_quality                     0.00
lab                             0.00
symmetry                        0.00
polish                          0.00
eye_clean                       0.00
culet_size                      0.00
culet_condition                 0.00
depth_percent                   0.00
table_percent                   0.00
meas_length                     0.00
meas_width                      0.00
meas_depth                      0.00
girdle_min                      0.00
girdle_max                      0.00
fluor_color                     0.00
fluor_intensity                65.31
fancy_color_dominant_color      0.00
fancy_color_secondary_color     0.00
fancy_color_overtone            0.75
fancy_color_intensity           0.00
total_sales_price               0.00
dtype: float64
```

In [5]:
```
df_1.duplicated().sum()
```

Out[5]:
```
3880
```

In [6]:
```
df_2 = df_1.drop_duplicates()
df_2.head()
```

Out[6]:

	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	culet_size	...	meas_depth	girdle_min
0	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	N	...	1.79	M
1	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Very Good	unknown	N	...	1.78	STK
2	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	unknown	...	1.77	TN
3	Round	E	VVS2	0.09	Excellent	IGI	Very Good	Very Good	unknown	unknown	...	1.78	M
4	Round	E	VVS2	0.09	Very Good	IGI	Very Good	Excellent	unknown	N	...	1.82	STK

5 rows × 25 columns

In [7]:
```
pp = df_2.dtypes.reset_index()
data_types_dict = {pp.loc[i,"index"]:str(pp.loc[i,0]) for i in range(len(pp.index))}
data_types_dict
```

Out[7]:
```
{'cut': 'object',
 'color': 'object',
 'clarity': 'object',
 'carat_weight': 'float64',
 'cut_quality': 'object',
 'lab': 'object',
 'symmetry': 'object',
 'polish': 'object',
 'eye_clean': 'object',
 'culet_size': 'object',
 'culet_condition': 'object',
 'depth_percent': 'float64',
 'table_percent': 'float64',
 'meas_length': 'float64',
 'meas_width': 'float64',
 'meas_depth': 'float64',
 'girdle_min': 'object',
 'girdle_max': 'object',
 'fluor_color': 'object',
 'fluor_intensity': 'object',
 'fancy_color_dominant_color': 'object',
 'fancy_color_secondary_color': 'object',
 'fancy_color_overtone': 'object',
 'fancy_color_intensity': 'object',
 'total_sales_price': 'int64'}
```

In [8]:
```
non_numeric_types = []
numeric_types = []
list_of_features_1 = ""
list_of_features_2 = ""

for i in data_types_dict:
    list_of_features_1 += (i + ", ")
    list_of_features_2 += (data_types_dict[i] + ", ")
    if data_types_dict[i] == "object":
        non_numeric_types.append(i)
    else:
        numeric_types.append(i)
numeric_types_float = numeric_types[0:len(numeric_types)-1]
numeric_types_int = numeric_types[len(numeric_types)-1]

list_of_features_1 = list_of_features_1[:list_of_features_1.rindex(", ")]
```

```
list_of_features_2 = list_of_features_2[:list_of_features_2.rindex(", ")]

with open(file="model/list_of_features.txt",mode="w+") as fl:
    fl.writelines(list_of_features_1+"\n")
    fl.writelines(list_of_features_2)
```

In [9]:
```
df_3 = df_2.copy()
for i in non_numeric_types:
    le = LabelEncoder()
    df_3[i] = le.fit_transform(df_2[i])
    joblib.dump(value=le,filename="model/"+i+".joblib")
```

In [9]:

In [10]:
```
df_3.head()
```

Out[10]:

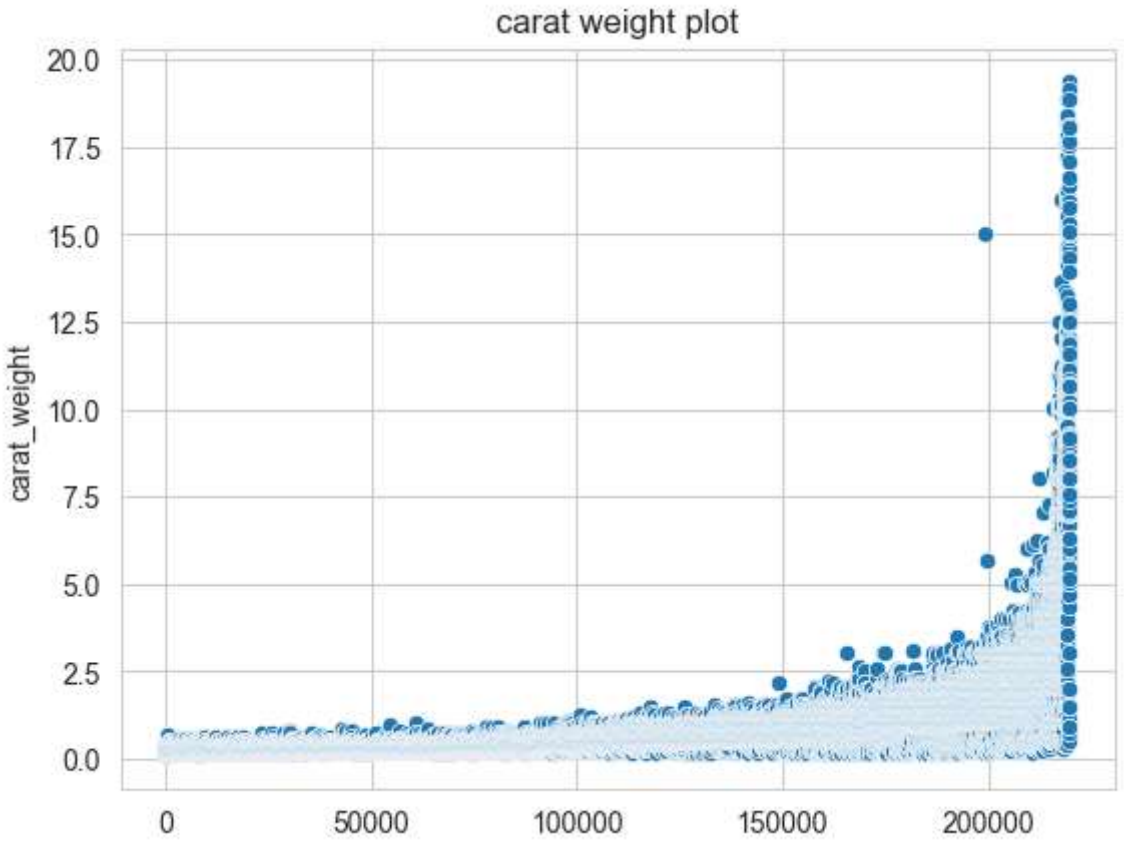
	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	culet_size	...	meas_depth	girdle_min	girdl
0	10	1	10	0.09	0	2	4	4	4	3	...	1.79	0	
1	10	1	10	0.09	4	2	4	4	4	3	...	1.78	1	
2	10	1	10	0.09	0	2	4	4	4	8	...	1.77	4	
3	10	1	10	0.09	0	2	4	4	4	8	...	1.78	0	
4	10	1	10	0.09	4	2	4	0	4	3	...	1.82	1	

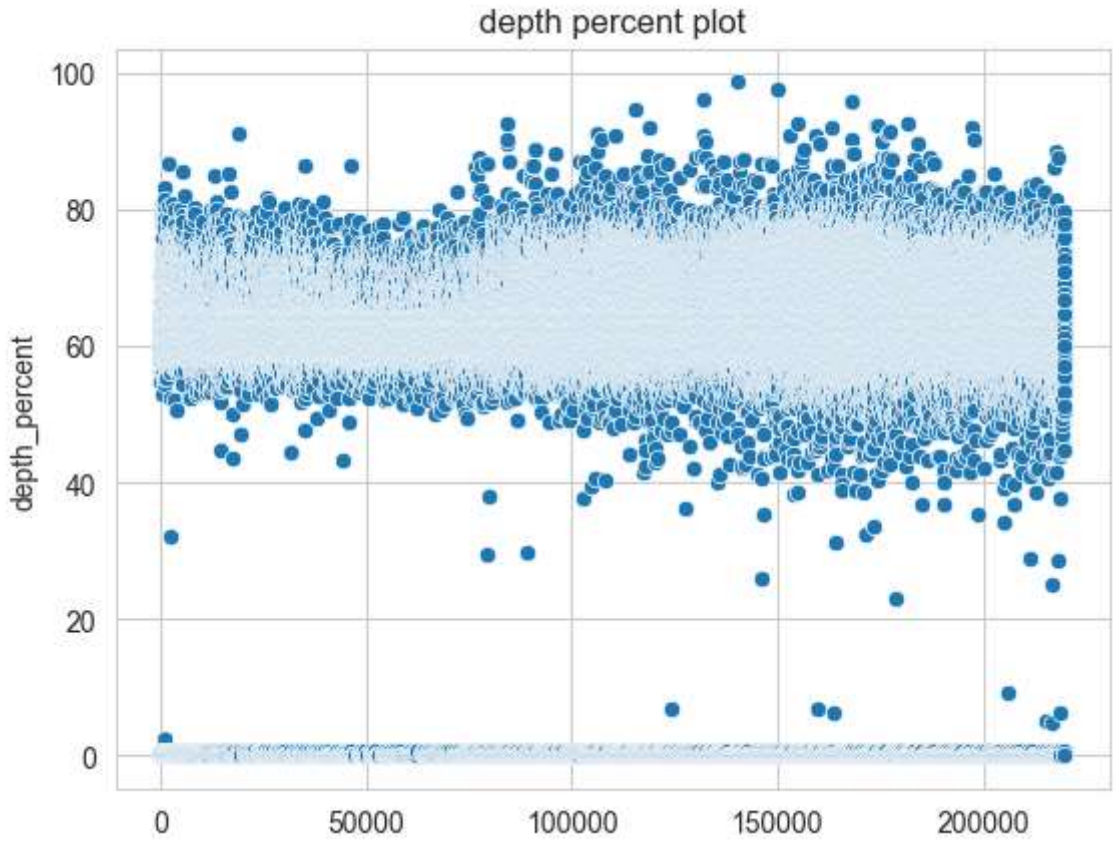
5 rows × 25 columns

Outlier Detection

In [10]:

In [11]:
```
for i in numeric_types_float:
    sns.scatterplot(data=df_3[i])
    plt.title(" ".join(i.split("_"))+" plot")
    plt.show()
```


carat weight plot
carat_weight
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
50000
100000
150000
200000


depth percent plot
depth_percent
100
80
60
40
20
0
0
50000
100000
150000
200000

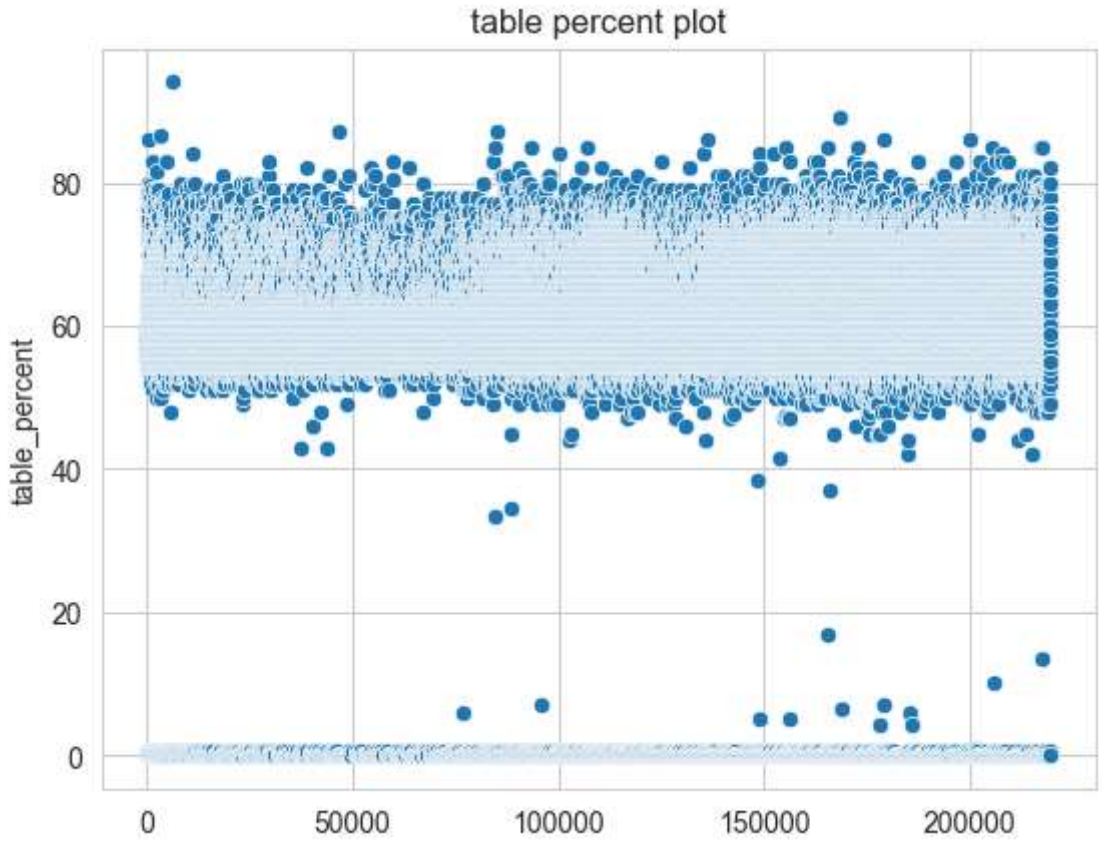
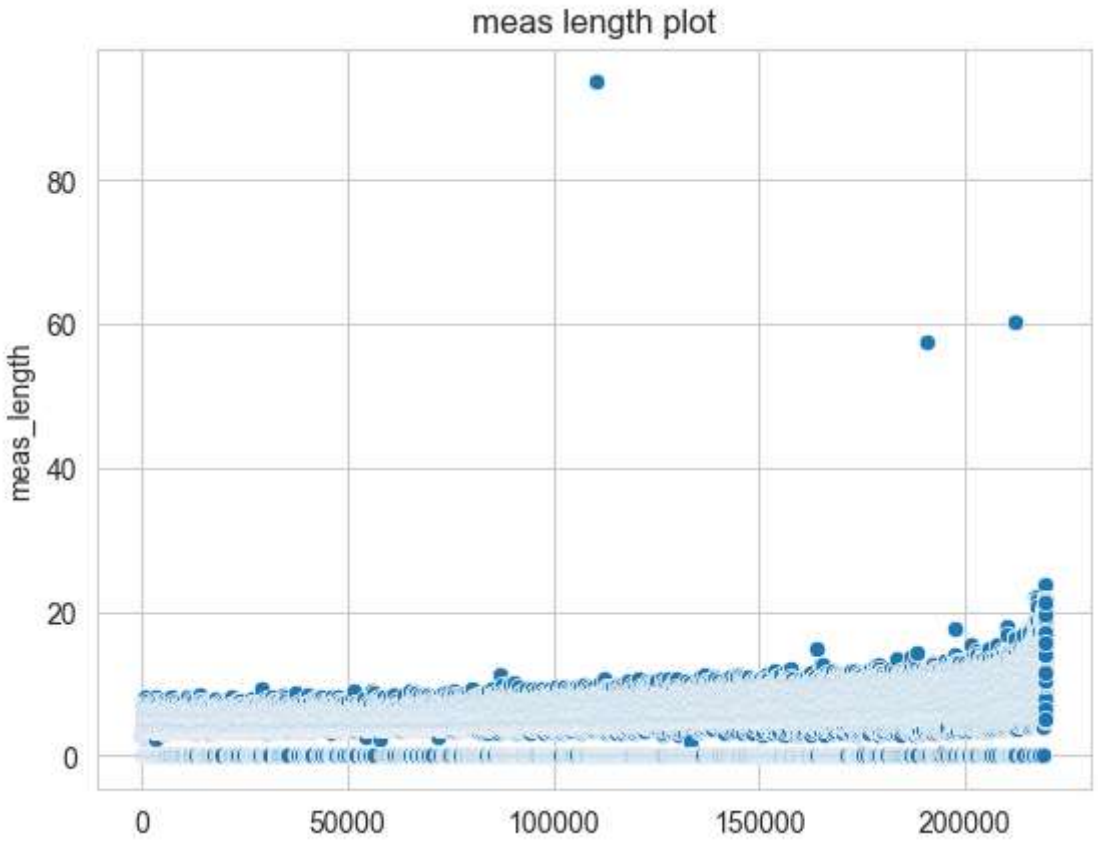
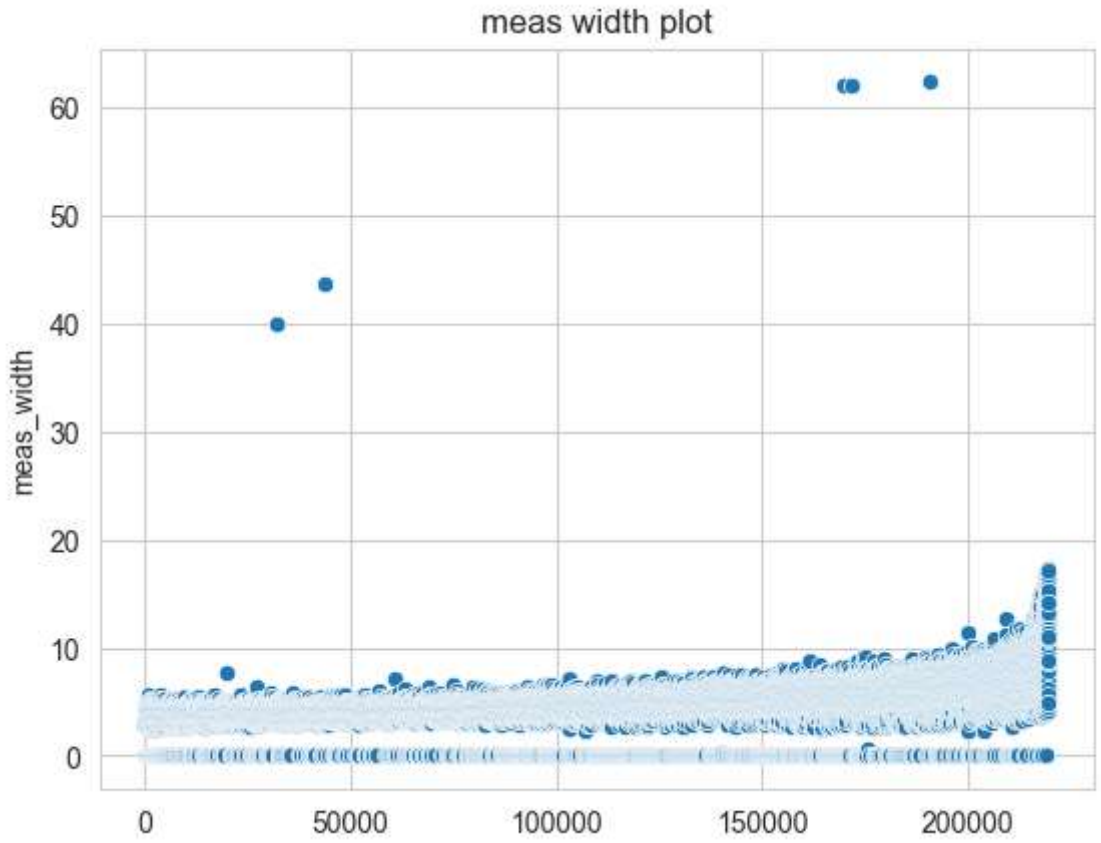
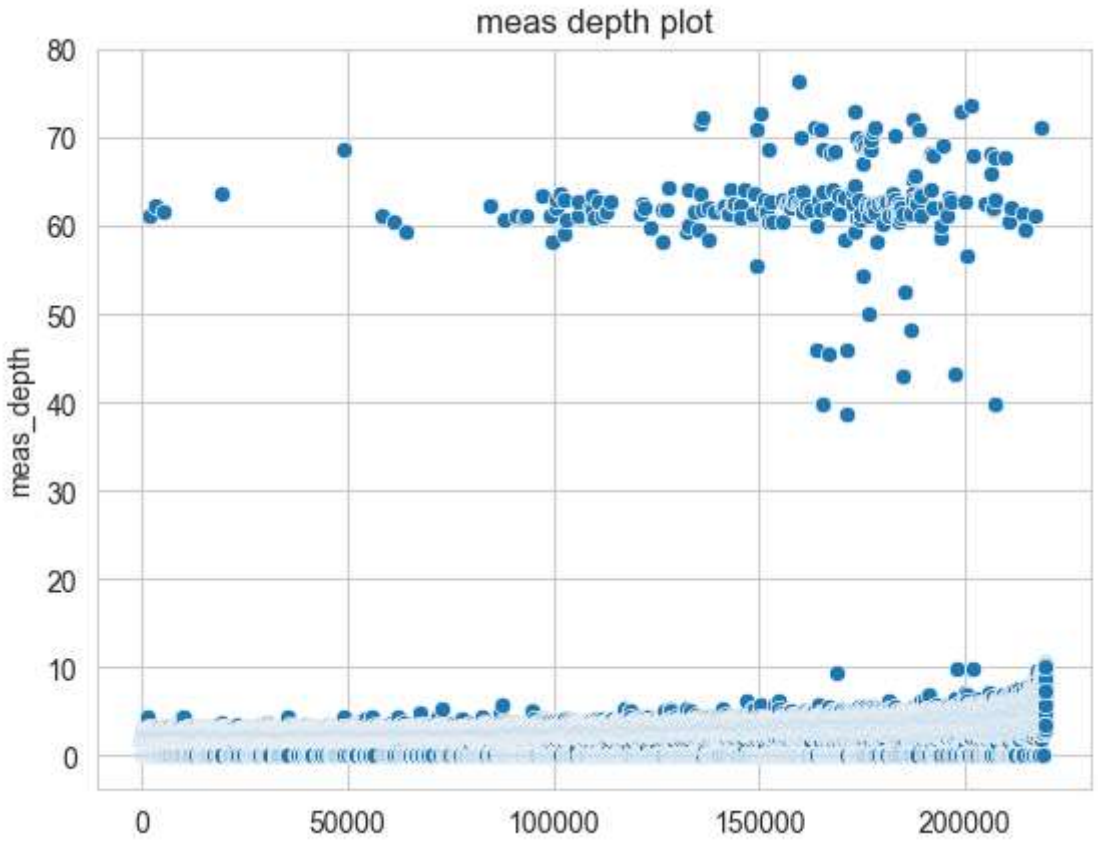


table percent plot
table_percent
80
60
40
20
0
0
50000
100000
150000
200000


meas length plot
meas_length
80
60
40
20
0
0
50000
100000
150000
200000


meas width plot
meas_width
60
50
40
30
20
10
0
0
50000
100000
150000
200000



In [12]:
```
cond_1 = df_3["depth_percent"] >= 40
cond_2 = df_3["depth_percent"] < 85

cond_3 = df_3["table_percent"] >= 50
cond_4 = df_3["table_percent"] < 80

cond_5 = df_3["meas_length"] < 30
cond_6 = df_3["meas_width"] < 20

cond_7 = df_3["meas_depth"] < 10

cond_8 = df_3["depth_percent"] != 0

cond_9 = df_3["table_percent"] != 0
```

```
cond_10 = df_3["meas_length"] != 0
cond_11 = df_3["meas_width"] != 0

cond_12 = df_3["meas_depth"] != 0

cond = cond_1 & cond_2 & cond_3 & cond_4 & cond_5 & cond_6 & cond_7 & cond_8 & cond_9 & cond_10 & cond_11 & cond_12

df_4 = df_3[cond].reset_index(drop=True)
df_4.head()
```

Out[12]:

	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	culet_size	...	meas_depth	girdle_min	girdl
0	10	1	10	0.09	0	2	4	4	4	3	...	1.79	0	
1	10	1	10	0.09	4	2	4	4	4	3	...	1.78	1	
2	10	1	10	0.09	0	2	4	4	4	8	...	1.77	4	
3	10	1	10	0.09	0	2	4	4	4	8	...	1.78	0	
4	10	1	10	0.09	4	2	4	0	4	3	...	1.82	1	

5 rows × 25 columns

Feature selection and Data splitting

In [13]:
```
X = df_4.drop(columns="total_sales_price")
y = df_4["total_sales_price"]
```

In [14]:
```
mms = MinMaxScaler()
X_res = pd.DataFrame(mms.fit_transform(X), columns=X.columns)
X_res.head()
```

Out[14]:

	cut	color	clarity	carat_weight	cut_quality	lab	symmetry	polish	eye_clean	culet_size	...	meas_width	meas_depth	gird
0	1.0	0.1	1.0	0.000519	0.0	1.0	1.0	1.0	1.0	0.375	...	0.158145	0.135394	0
1	1.0	0.1	1.0	0.000519	0.8	1.0	1.0	1.0	1.0	0.375	...	0.159334	0.134328	0
2	1.0	0.1	1.0	0.000519	0.0	1.0	1.0	1.0	1.0	1.000	...	0.159929	0.133262	0
3	1.0	0.1	1.0	0.000519	0.0	1.0	1.0	1.0	1.0	1.000	...	0.158740	0.134328	0
4	1.0	0.1	1.0	0.000519	0.8	1.0	1.0	0.0	1.0	0.375	...	0.155767	0.138593	0

5 rows × 24 columns

In [15]:
```
f_regr = f_regression(X_res,y)
f_res = pd.DataFrame(columns=["columns","f-values","p-values"])
f_res["columns"] = X_res.columns
f_res["f-values"] = f_regr[0]
f_res["p-values"] = f_regr[1].round(2)
features_order = f_res.sort_values(by="f-values",ascending=False).reset_index(drop=True)
best_features = features_order["columns"].values[0:15]
print(best_features)
```

```
['carat_weight' 'meas_depth' 'meas_width' 'meas_length' 'cut'
 'fancy_color_intensity' 'table_percent' 'fancy_color_dominant_color'
 'cut_quality' 'color' 'eye_clean' 'depth_percent' 'fluor_color'
 'fancy_color_secondary_color' 'culet_condition']
```

In [16]:
```
X_res_2 = X_res[best_features]
X_res_2.head()
```

Out[16]:

	carat_weight	meas_depth	meas_width	meas_length	cut	fancy_color_intensity	table_percent	fancy_color_dominant_color	c
0	0.000519	0.135394	0.158145	0.117746	1.0	1.0	0.301003	1.0	
1	0.000519	0.134328	0.159334	0.117325	1.0	1.0	0.301003	1.0	
2	0.000519	0.133262	0.159929	0.119008	1.0	1.0	0.301003	1.0	
3	0.000519	0.134328	0.158740	0.118167	1.0	1.0	0.301003	1.0	
4	0.000519	0.138593	0.155767	0.115223	1.0	1.0	0.284281	1.0	

In [17]:
```
X_train,X_test,y_train,y_test = train_test_split(X_res_2,y,test_size=0.2,random_state=np.random.randint(0,1000))
```

Modelling the Regression

In [18]:
```
linear_regr = LinearRegression()
linear_regr.fit(X_train,y_train)
y_pred = np.abs(linear_regr.predict(X_test))
```

In [19]:
```
intercept = round(linear_regr.intercept_,2)
coefficients = [i.round(2) for i in linear_regr.coef_]
print(intercept)
print(coefficients)
```

```
42851.87
[761091.88, -219499.86, 118712.69, -73610.47, 6423.91, 317.12, -10442.19, -25649.58, 40.05, -8207.59, 1199.31, 23417.
25, 181.32, -1150.87, -3266.84]
```

In [20]:
```
mae = mean_absolute_error(y_test,y_pred)
r2 = r2_score(y_test,y_pred) * 100

print("Mean Absolute Error : ",round(mae,2))
print("R2 score            : ",round(r2,2))
```

```
Mean Absolute Error :  3571.91
R2 score            :  65.93
```

Neural Network

In [21]:
```
X_2 = df_3.drop(columns="total_sales_price")
y_2 = df_3["total_sales_price"]
X_train_2,X_test_2,y_train_2,y_test_2 = train_test_split(X_2,y_2,test_size=0.2,random_state=np.random.randint(0,1000)
```

In [22]:
```
model = tf.keras.models.Sequential(layers=[
    tf.keras.layers.Dense(units=50,activation="relu",input_dim=24),
    tf.keras.layers.Dense(units=100,activation="relu"),
    tf.keras.layers.Dense(units=100,activation="relu"),
    tf.keras.layers.Dense(units=50,activation="relu"),
    tf.keras.layers.Dense(units=1,activation="linear")
])
```

```
WARNING:tensorflow:From C:\Users\amith\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\backend.p
y:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

In [23]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(),loss=tf.keras.losses.mean_squared_error,metrics=tf.keras.metrics.r
model.fit(X_train_2,y_train_2,validation_split=0.2,batch_size=32,epochs=100,verbose=1,callbacks=tf.keras.callbacks.Ea
```

```
Epoch 1/100
WARNING:tensorflow:From C:\Users\amith\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\utils\tf_u
tils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue ins
tead.
WARNING:tensorflow:From C:\Users\amith\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\engine\bas
e_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executin
g_eagerly_outside_functions instead.
4317/4317 [==============================] - 17s 3ms/step - loss: 331454112.0000 - mean_absolute_error: 4171.4634 - v
al_loss: 269382144.0000 - val_mean_absolute_error: 2848.2710
Epoch 2/100
4317/4317 [==============================] - 17s 4ms/step - loss: 226126704.0000 - mean_absolute_error: 2917.7297 - v
al_loss: 238752064.0000 - val_mean_absolute_error: 2948.0525
Epoch 3/100
4317/4317 [==============================] - 17s 4ms/step - loss: 207618096.0000 - mean_absolute_error: 2736.1934 - v
al_loss: 243626672.0000 - val_mean_absolute_error: 4290.1152
Epoch 4/100
4317/4317 [==============================] - 17s 4ms/step - loss: 184473984.0000 - mean_absolute_error: 2580.3181 - v
al_loss: 224674704.0000 - val_mean_absolute_error: 2610.7742
Epoch 5/100
4317/4317 [==============================] - 16s 4ms/step - loss: 170059904.0000 - mean_absolute_error: 2437.7590 - v
al_loss: 285907296.0000 - val_mean_absolute_error: 4199.1484
Epoch 6/100
4317/4317 [==============================] - 14s 3ms/step - loss: 166363984.0000 - mean_absolute_error: 2463.7437 - v
al_loss: 187367136.0000 - val_mean_absolute_error: 2373.7590
Epoch 7/100
4317/4317 [==============================] - 9s 2ms/step - loss: 158069648.0000 - mean_absolute_error: 2406.0737 - va
l_loss: 173548384.0000 - val_mean_absolute_error: 2058.1091
Epoch 8/100
4317/4317 [==============================] - 9s 2ms/step - loss: 153555504.0000 - mean_absolute_error: 2352.0454 - va
l_loss: 207897952.0000 - val_mean_absolute_error: 4712.5415
Epoch 9/100
4317/4317 [==============================] - 9s 2ms/step - loss: 152034096.0000 - mean_absolute_error: 2341.2429 - va
l_loss: 174461792.0000 - val_mean_absolute_error: 2213.2874
```

Out[23]: <keras.src.callbacks.History at 0x2b1efac9ed0>

In [24]:
```
y_pred_2 = np.abs(model.predict(X_test_2))
```

```
1349/1349 [==============================] - 2s 1ms/step
```

In [25]:
```
y_pred_2
```

Out[25]:
```
array([[14893.539 ],
       [17155.438 ],
       [ 6797.581 ],
       ...,
       [ 4709.224 ],
       [ 1077.9987],
       [ 1432.0774]], dtype=float32)
```

In [26]:
```
print(X_test_2.iloc[0,:].values.tolist())
```

```
[7.0, 3.0, 5.0, 1.51, 5.0, 0.0, 0.0, 0.0, 4.0, 8.0, 3.0, 63.7, 60.0, 10.21, 6.19, 3.94, 9.0, 9.0, 5.0, 3.0, 12.0, 10.
0, 7.0, 9.0]
```

In [27]:
```
mae_2 = mean_absolute_error(y_test_2,y_pred_2)
r2_2 = r2_score(y_test_2,y_pred_2) * 100

print("Mean Absolute Error : ",round(mae_2,2))
print("R2 score            : ",round(r2_2,2))
```

```
Mean Absolute Error :  2114.42
R2 score            :  76.86
```

In [28]:
```
pa = os.getcwd().replace("\\","/")
tf.keras.models.save_model(model=model,filepath=pa+"/model/deep_learning_linear_regression.h5")
```

In [ ]:
```

```