Advance Data Science

Step by step guide for Pycret Model

Step1: Package installation

```
[ ]  pip install pycaret # package pycaret installation

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: pycaret in /usr/local/lib/python3.8/dist-packages (2.3.10)
     Requirement already satisfied: kmodes>=0.10.1 in /usr/local/lib/python3.8/dist-packages (from pycaret) (0.12.2)
     Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist-packages (from pycaret) (3.2.2)
     Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from pycaret) (1.2.0)
     Requirement already satisfied: yellowbrick>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from pycaret) (1.3.post1)
     Requirement already satisfied: scipy<=1.5.4 in /usr/local/lib/python3.8/dist-packages (from pycaret) (1.5.4)
     Requirement already satisfied: imbalanced-learn==0.7.0 in /usr/local/lib/python3.8/dist-packages (from pycaret) (0.7.0)
     Requirement already satisfied: textblob in /usr/local/lib/python3.8/dist-packages (from pycaret) (0.15.3)
     Requirement already satisfied: mlflow in /usr/local/lib/python3.8/dist-packages (from pycaret) (2.0.1)
     Requirement already satisfied: pyod in /usr/local/lib/python3.8/dist-packages (from pycaret) (1.0.6)
     Requirement already satisfied: pyyaml<6.0.0 in /usr/local/lib/python3.8/dist-packages (from pycaret) (5.4.1)
     Requirement already satisfied: scikit-learn==0.23.2 in /usr/local/lib/python3.8/dist-packages (from pycaret) (0.23.2)
```

Fig1: Pycrat package installation on colab

The above line of code install the package in google colab.

Similarly install following 2 package as well.

```
[ ]  pip install markupsafe==2.0.1

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: markupsafe==2.0.1 in /usr/local/lib/python3.8/dist-packages (2.0.1)

[ ]  pip install jinja2

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: jinja2 in /usr/local/lib/python3.8/dist-packages (2.11.3)
     Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dist-packages (from jinja2) (2.0.1)
```

Fig2: Installing Package for Pycret Model

Install 3 package using pip package manager.

Pycret is low code ML library for EDA, Pre-processing, Modelling , Training and MLOPS.

Markupsafe is for dealing with text and special character to wrap in markup.

Jinja2 is fast templating engine.

Step2: Import Libraries

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
import jinja2
from pycaret.regression import *

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Fig3: Required library for Pycret Model

Step 3: Upload dataset.

```
from google.colab import files
files.upload()
```

Fig 4: To upload the data on colab

Step 4:Data pre-processing

```
df = pd.read_csv('coin_Bitcoin.csv', parse_dates=['Date'])
```

```
df = df.sort_values('Date')
```

Fig4 : Reading the data and sorting it according to data .

```
coinbit.isnull().sum()# checking null values
```

```
SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

Fig5: Checking for null values in the data set

```
[ ] df.duplicated().sum()

0
```

Fig 6: Checking for Duplicate row

```
coinbit=coinbit.dropna()
coinbit=coinbit.drop(columns=['New_Price'])
```

Fig 7: Dropping the unused column

```
[ ] coinbit.isnull().sum()# checking null values

SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

Fig 8: Checking Null values

<u>Step5:EDA</u>

All the EDA is same as LSTM Model.

Run the following cell and analyze the output.

```
df.head(10)
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.603769e+09
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.542813e+09
2	3	Bitcoin	BTC	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.0	1.298955e+09
3	4	Bitcoin	BTC	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.0	1.168517e+09
4	5	Bitcoin	BTC	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.0	1.085995e+09
5	6	Bitcoin	BTC	2013-05-04 23:59:59	115.000000	92.500000	98.099998	112.500000	0.0	1.250317e+09
6	7	Bitcoin	BTC	2013-05-05 23:59:59	118.800003	107.142998	112.900002	115.910004	0.0	1.288693e+09
7	8	Bitcoin	BTC	2013-05-06 23:59:59	124.663002	106.639999	115.980003	112.300003	0.0	1.249023e+09
8	9	Bitcoin	BTC	2013-05-07 23:59:59	113.444000	97.699997	112.250000	111.500000	0.0	1.240594e+09
9	10	Bitcoin	BTC	2013-05-08 23:59:59	115.779999	109.599998	109.599998	113.566002	0.0	1.264049e+09

Fig 10: Reading 10 rows from head

```
df.tail(10)
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
2981	2982	Bitcoin	BTC	2021-06-27 23:59:59	34656.127356	32071.757148	32287.523211	34649.644588	3.551164e+10	6.494617e+11
2982	2983	Bitcoin	BTC	2021-06-28 23:59:59	35219.891791	33902.075892	34679.122222	34434.335314	3.389252e+10	6.454428e+11
2983	2984	Bitcoin	BTC	2021-06-29 23:59:59	36542.111018	34252.484892	34475.559697	35867.777735	3.790146e+10	6.723334e+11
2984	2985	Bitcoin	BTC	2021-06-30 23:59:59	36074.759757	34086.151878	35908.388054	35040.837249	3.405904e+10	6.568525e+11
2985	2986	Bitcoin	BTC	2021-07-01 23:59:59	35035.982712	32883.781226	35035.982712	33572.117653	3.783896e+10	6.293393e+11
2986	2987	Bitcoin	BTC	2021-07-02 23:59:59	33939.588699	32770.680780	33549.600177	33897.048590	3.872897e+10	6.354508e+11
2987	2988	Bitcoin	BTC	2021-07-03 23:59:59	34909.259899	33402.696536	33854.421362	34668.548402	2.438396e+10	6.499397e+11
2988	2989	Bitcoin	BTC	2021-07-04 23:59:59	35937.567147	34396.477458	34665.564866	35287.779766	2.492431e+10	6.615748e+11
2989	2990	Bitcoin	BTC	2021-07-05 23:59:59	35284.344430	33213.661034	35284.344430	33746.002456	2.672155e+10	6.326962e+11

Fig 11: Reading 10 rows from tail of data set

```
[ ]  df.shape

(2991, 10)
```

Fig12: Counting rows and column of data set

```
[ ]  df.describe

<bound method NDFrame.describe of        SNo     Name Symbol                Date         High          Low  \
0        1  Bitcoin    BTC 2013-04-29 23:59:59    147.488007    134.000000
1        2  Bitcoin    BTC 2013-04-30 23:59:59    146.929993    134.050003
2        3  Bitcoin    BTC 2013-05-01 23:59:59    139.889999    107.720001
3        4  Bitcoin    BTC 2013-05-02 23:59:59    125.599998     92.281898
4        5  Bitcoin    BTC 2013-05-03 23:59:59    108.127998     79.099998
...    ...      ...    ...                 ...           ...          ...
2986  2987  Bitcoin    BTC 2021-07-02 23:59:59  33939.588699  32770.680780
2987  2988  Bitcoin    BTC 2021-07-03 23:59:59  34909.259899  33402.696536
2988  2989  Bitcoin    BTC 2021-07-04 23:59:59  35937.567147  34396.477458
2989  2990  Bitcoin    BTC 2021-07-05 23:59:59  35284.344430  33213.661034
2990  2991  Bitcoin    BTC 2021-07-06 23:59:59  35038.536363  33599.916169

              Open         Close        Volume     Marketcap
0       134.444000    144.539993  0.000000e+00  1.603769e+09
1       144.000000    139.000000  0.000000e+00  1.542813e+09
2       139.000000    116.989998  0.000000e+00  1.298955e+09
3       116.379997    105.209999  0.000000e+00  1.168517e+09
4       106.250000     97.750000  0.000000e+00  1.085995e+09
...            ...           ...           ...           ...
2986  33549.600177  33897.048590  3.872897e+10  6.354508e+11
```

✓ 1s completed at 11:01 PM

Fig 13: Describing the data and its property

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2991 entries, 0 to 2990
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   SNo        2991 non-null   int64
 1   Name       2991 non-null   object
 2   Symbol     2991 non-null   object
 3   Date       2991 non-null   datetime64[ns]
 4   High       2991 non-null   float64
 5   Low        2991 non-null   float64
 6   Open       2991 non-null   float64
 7   Close      2991 non-null   float64
 8   Volume     2991 non-null   float64
 9   Marketcap  2991 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(1), object(2)
memory usage: 257.0+ KB
```

Fig 14: looking into the info of data for data type and null values

```
[ ]  df.dtypes
```

```
SNo                    int64
Name                  object
Symbol                object
Date          datetime64[ns]
High                 float64
Low                  float64
Open                 float64
Close                float64
Volume               float64
Marketcap            float64
dtype: object
```

Fig 15: Looking into the datatype of dataset

```
[ ]  df.isna()
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
2986	False	False	False	False	False	False	False	False	False	False
2987	False	False	False	False	False	False	False	False	False	False
2988	False	False	False	False	False	False	False	False	False	False
2989	False	False	False	False	False	False	False	False	False	False
2990	False	False	False	False	False	False	False	False	False	False

✓ 1s completed at 11:01 PM

Fig 16: Checking for NaN values

```
[ ]  df.duplicated().sum()
```

```
0
```

Fig 17: Checking for duplicate values

```
df.value_counts()
```

```
SNo   Name     Symbol  Date                 High          Low           Open          Close         Volume        Marketcap
1     Bitcoin  BTC     2013-04-29 23:59:59  147.488007    134.000000    134.444000    144.539993    0.000000e+00  1.603769e+09    1
1998  Bitcoin  BTC     2018-10-17 23:59:59  6601.210000   6517.450000   6590.520000   6544.430000   4.088420e+09  1.133993e+11    1
1989  Bitcoin  BTC     2018-10-08 23:59:59  6675.060000   6576.040000   6600.190000   6652.230000   3.979460e+09  1.151629e+11    1
1990  Bitcoin  BTC     2018-10-09 23:59:59  6661.410000   6606.940000   6653.080000   6642.640000   3.580810e+09  1.150078e+11    1
1991  Bitcoin  BTC     2018-10-10 23:59:59  6640.290000   6538.960000   6640.290000   6585.530000   3.787650e+09  1.140308e+11    1
                                                                                                                               ..
1000  Bitcoin  BTC     2016-01-23 23:59:59  394.542999    381.980988    382.433990    387.490997    5.624740e+07  5.858060e+09    1
1001  Bitcoin  BTC     2016-01-24 23:59:59  405.484985    387.510010    388.101990    402.971008    5.482480e+07  6.093788e+09    1
1002  Bitcoin  BTC     2016-01-25 23:59:59  402.316986    388.553986    402.316986    391.726013    5.906240e+07  5.925345e+09    1
1003  Bitcoin  BTC     2016-01-26 23:59:59  397.765991    390.575012    392.002014    392.153015    5.814700e+07  5.933373e+09    1
2991  Bitcoin  BTC     2021-07-06 23:59:59  35038.536363  33599.916169  33723.509655  34235.193451  2.650126e+10  6.418992e+11    1
Length: 2991, dtype: int64
```

Fig18: Counting for the entire data

```
df.max()
```

```
SNo                            2991
Name                        Bitcoin
Symbol                          BTC
Date            2021-07-06 23:59:59
High                   64863.098908
Low                    62208.964366
Open                   63523.754869
Close                   63503.45793
Volume        350967941479.059998
Marketcap    1186364044140.27002
dtype: object
```

```
[ ] print("All Time High Price:",max(coinbit['Close']))
    print("Highest Number of Bitcoin units traded during the minute:",max(coinbit['Volume']))
```

```
All Time High Price: 63503.45793019
Highest Number of Bitcoin units traded during the minute: 350967941479.06
```

Fig 19: looking for maximum values for each row

Step 6:Visualization analysis

```
sns.pairplot(df)
```

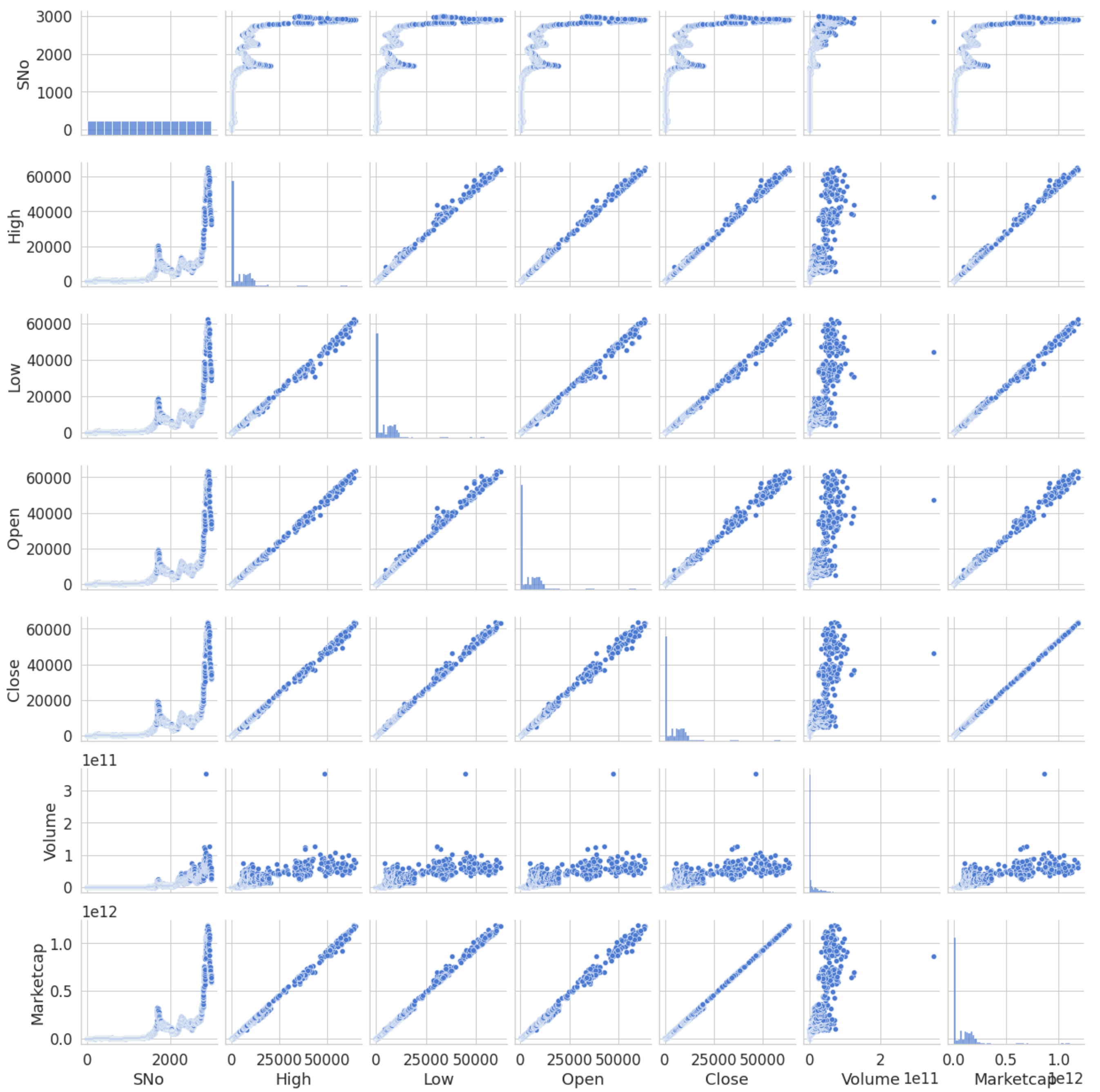


Fig 20 : Pair plot of each and every row

```
sns.boxplot(x=df["Close"])
```

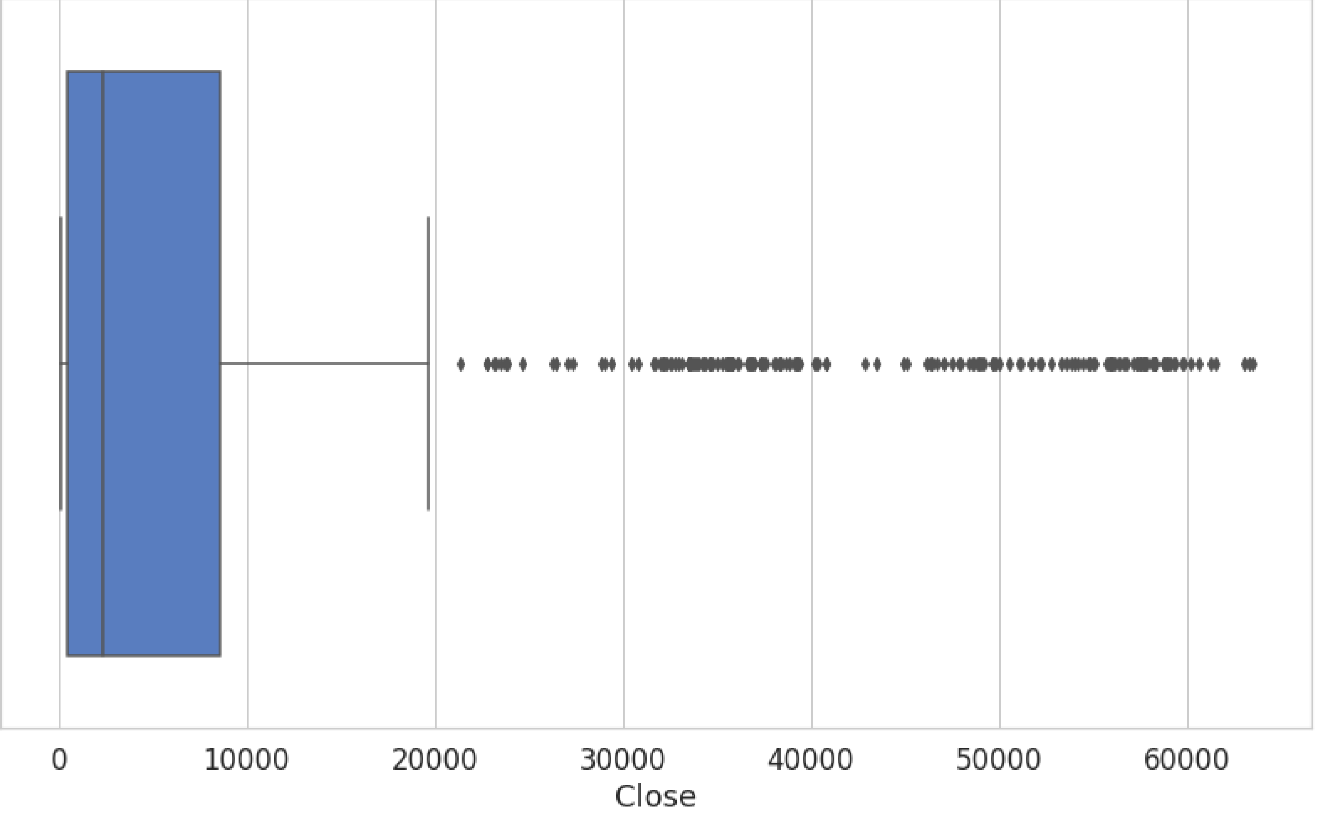


Fig 21: Boxplot for Close price

Draw a single horizontal boxplot, assigning the data directly to the coordinate variable:

```
[ ] boxplot = df.boxplot(column=['Low','High','Open','Close'])
```

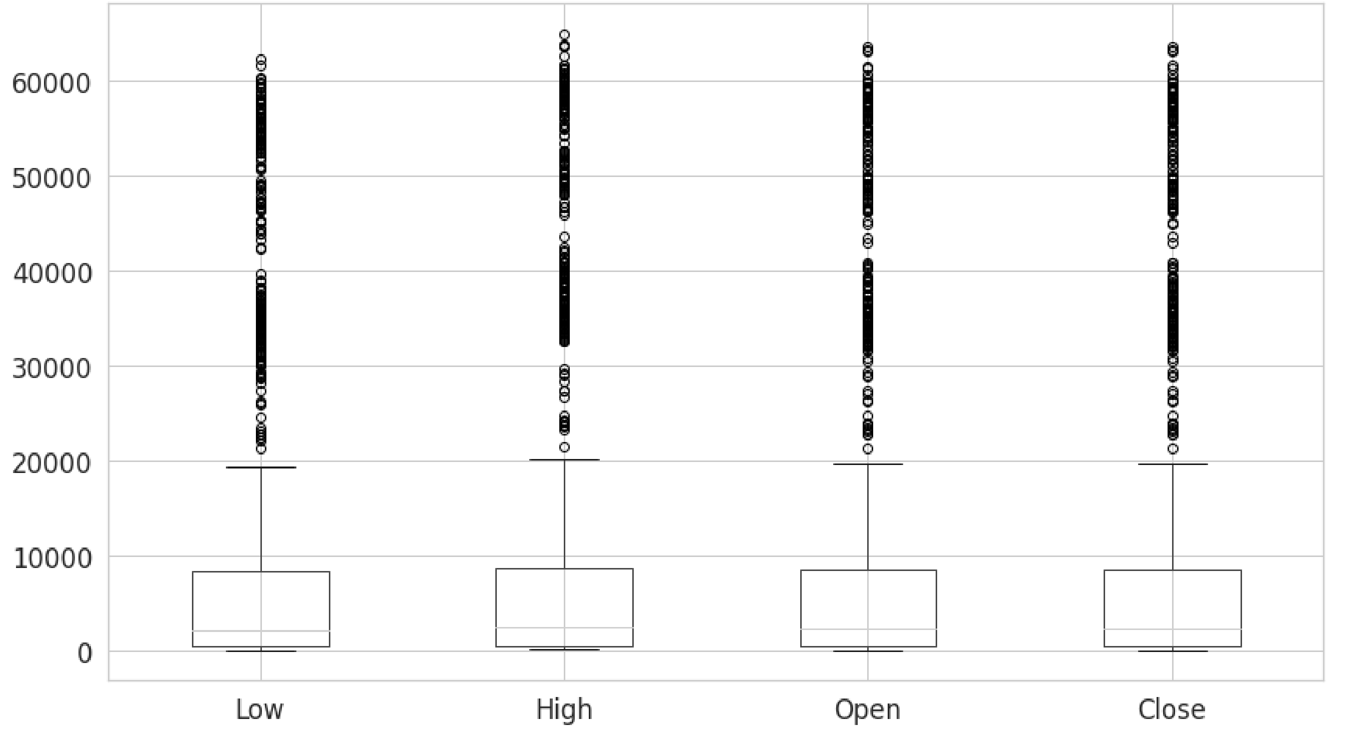


Fig 22: Boxplot for Low, High, Open and Close combined

Group by a categorical variable, referencing columns in a dataframe:

```
sns.boxplot(data=df[["Open", "Close" ,"High" ,"Low"]], orient="h")
```

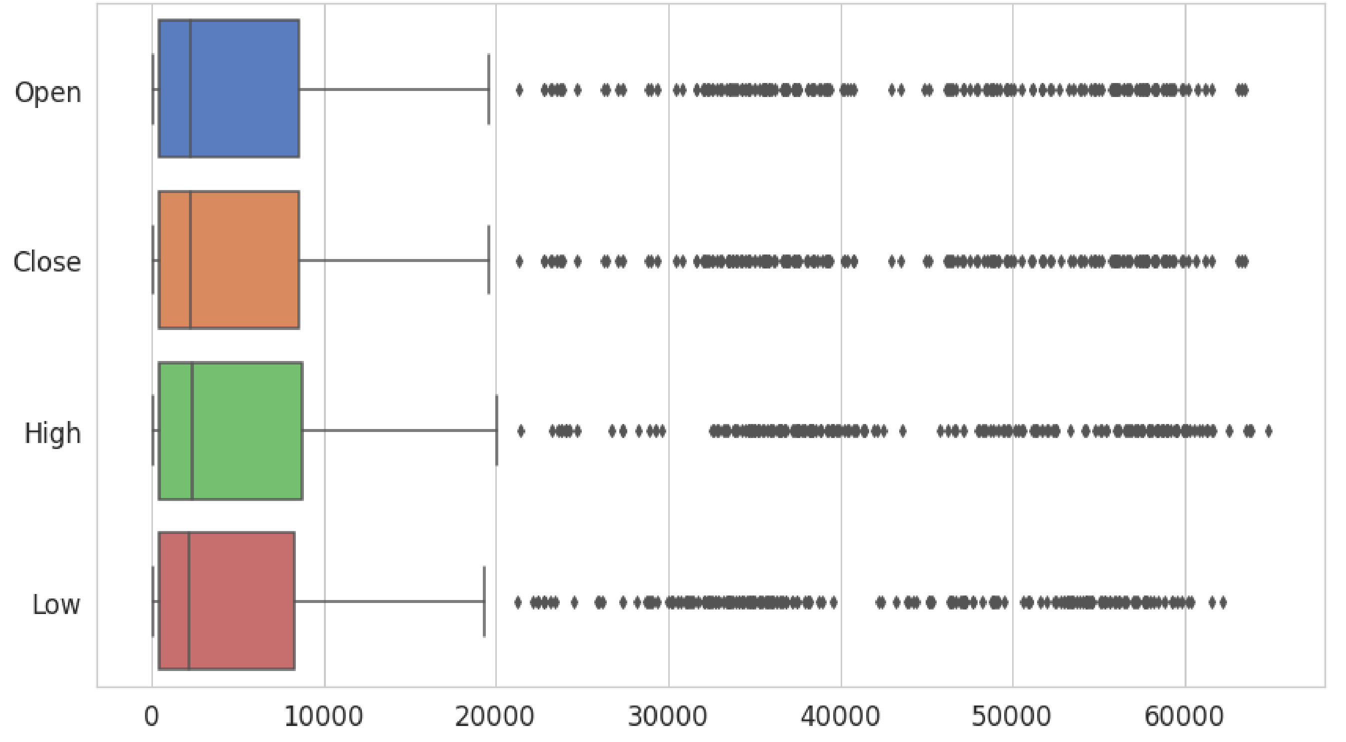


Fig 23: Boxplot for Low, High, Open and Close combined in horizontal orientation

Group by a categorical variable, referencing columns in a dataframe:

```
sns.boxplot(data=df[["Open", "Close" ,"High" ,"Low","Volume", "Marketcap"]], orient="h")
```

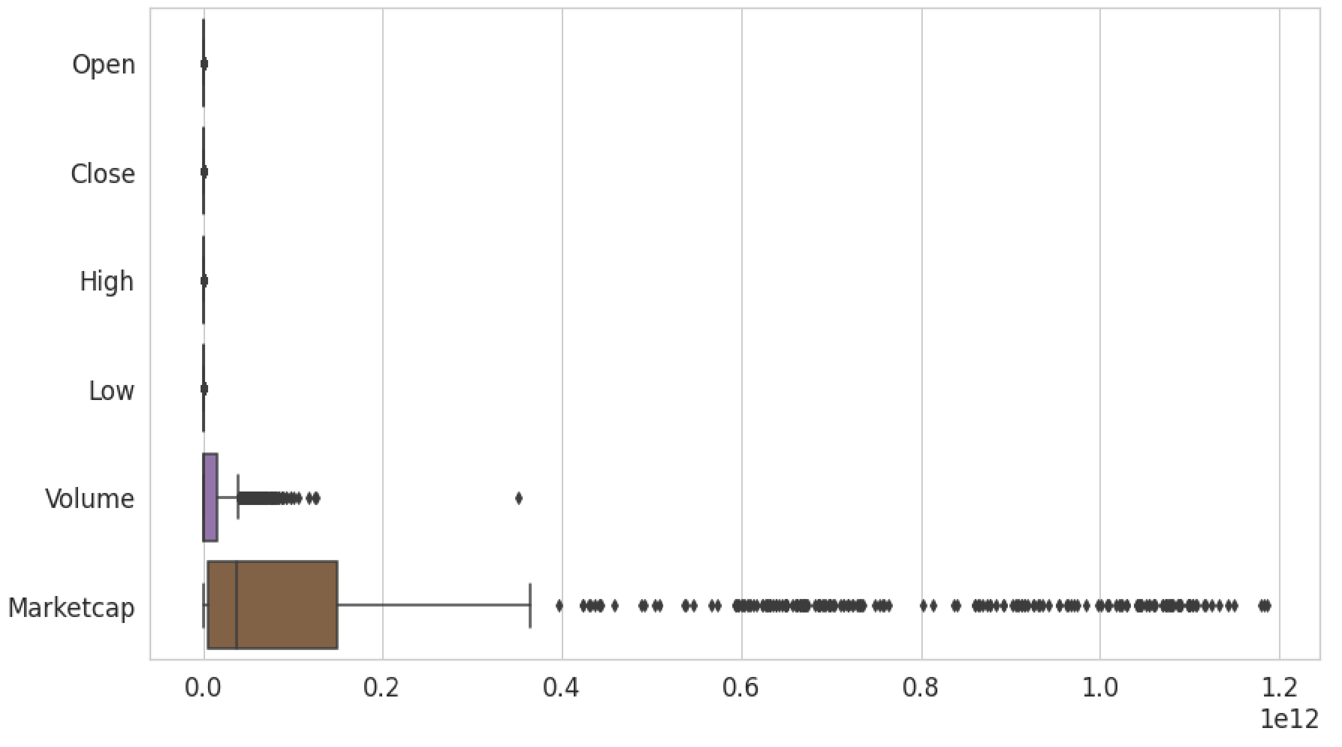


Fig 24: Boxplot for Low, High, Open , Close, Volume and Market Capitalization combined

Group by a categorical variable, referencing columns in a dataframe:

```
[ ] sns.boxplot(
        data=df, x="Close",
        notch=True, showcaps=False,
        flierprops={"marker": "x"},
        boxprops={"facecolor": (.4, .6, .8, .5)},
        medianprops={"color": "coral"},
    )
```

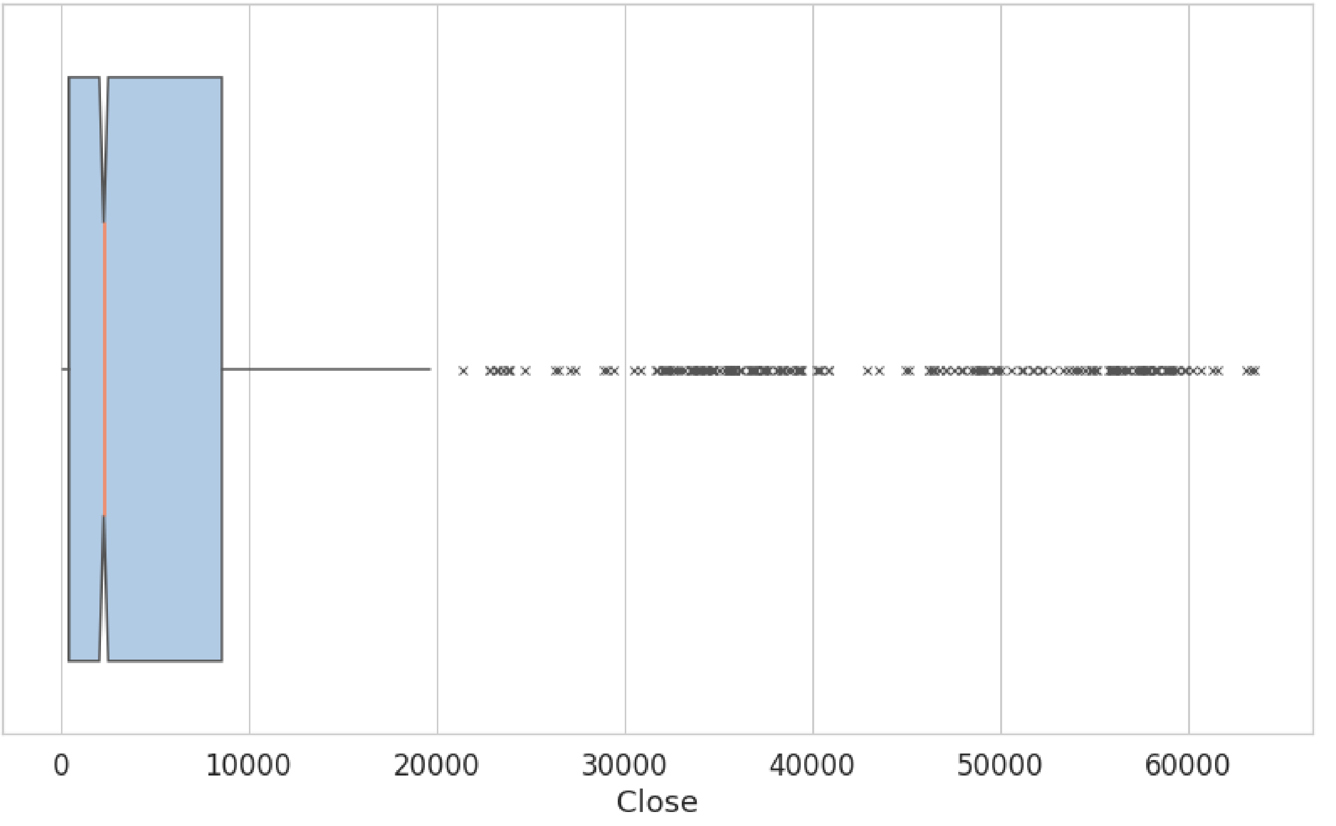


Fig 25: Boxplot Close value with additional information

Step7:Creating dependent variable

```
coming_day = 10 # variable for predicting for coming 10 day
coinbit['New_Price'] = coinbit[['Close']].shift(-coming_day)#creating the new coumns for dependent variable
coinbit = coinbit[['Close' , 'New_Price']] # choose new column
coinbit # displayes close value and new dependent variables data.
```

	Close	New_Price
0	144.539993	112.669998
1	139.000000	117.199997
2	116.989998	115.242996
3	105.209999	115.000000
4	97.750000	117.980003
...	...	...

Fig 40: Creating new dependent variable

Step8: Developing Training and Testing Model

```
df = coinbit.copy()# making a copy of data set as data frame.
x = np.array(df[df.columns])#creating independent data set
x = x [:len(coinbit)-coming_day] # remov last n  row from the data set now n= coming_day=10
y = np.array(df['New_Price']) # creating dependent data set
y = y[:-coming_day] # getting  all y values except last 10 rows
X_train , X_test , y_train , y_test = train_test_split(x , y , test_size=0.2 , random_state = 0 , shuffle = False)#spliting training and testing data set  Tra
```

Fig 41: Splitting dataset for training and testing

```
train_data = pd.DataFrame(X_train , columns = df.columns) # getting train data and transform into data frame.
train_data.head(10)# show first 10 rows of data
```

	Close	New_Price
0	144.539993	112.669998
1	139.000000	117.199997
2	116.989998	115.242996
3	105.209999	115.000000
4	97.750000	117.980003
5	112.500000	111.500000
6	115.910004	114.220001
7	112.300003	118.760002

Fig 42: Training data set

```
test_data = pd.DataFrame(X_train , columns= df.columns) # getting test data and transform into dataframe
test_data.head(10)# Show forst 10 rows of data
```

	Close	New_Price
0	144.539993	112.669998
1	139.000000	117.199997
2	116.989998	115.242996
3	105.209999	115.000000
4	97.750000	117.980003
5	112.500000	111.500000
6	115.910004	114.220001
7	112.300003	118.760002
8	111.500000	123.014999

Fig 43:Testing data set

Step9: Setup initialization

```
regression_setup = setup(data = train_data, target = 'New_Price' , session_id =123 , use_gpu = True)# setup initialization
```

	Description	Value
0	session_id	123
1	Target	New_Price
2	Original Data	(2384, 2)
3	Missing Values	False
4	Numeric Features	1
5	Categorical Features	0
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(1668, 1)
10	Transformed Test Set	(716, 1)

Fig44 : setup initialization

Step10: Finding best Model

```
#Train all the model ad sort it by R -squire matrix(r2) and store the model.
best_model = compare_models(sort = 'r2')
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
lightgbm	Light Gradient Boosting Machine	317.4969	5.284034e+05	713.1998	0.9622	0.1346	0.0964	0.066
llar	Lasso Least Angle Regression	370.5780	5.529972e+05	732.1215	0.9602	0.2919	0.2675	0.009
lasso	Lasso Regression	357.3850	5.514736e+05	731.3831	0.9602	0.2456	0.2173	0.012
br	Bayesian Ridge	357.4169	5.514734e+05	731.3823	0.9602	0.2458	0.2175	0.009
omp	Orthogonal Matching Pursuit	357.3848	5.514735e+05	731.3831	0.9602	0.2456	0.2173	0.008
lr	Linear Regression	357.3848	5.514735e+05	731.3831	0.9602	0.2456	0.2173	0.009
lar	Least Angle Regression	357.3848	5.514736e+05	731.3831	0.9602	0.2456	0.2173	0.012
en	Elastic Net	357.3850	5.514736e+05	731.3831	0.9602	0.2456	0.2173	0.012
ridge	Ridge Regression	357.3849	5.514735e+05	731.3831	0.9602	0.2456	0.2173	0.008
huber	Huber Regressor	332.2659	5.528015e+05	732.3528	0.9601	0.1433	0.1059	0.020
knn	K Neighbors Regressor	328.7909	5.655004e+05	738.4657	0.9594	0.1386	0.0996	0.182
gbr	Gradient Boosting Regressor	329.0785	5.995877e+05	764.1338	0.9561	0.1457	0.1091	0.111

Fig 45 : train all the model and best outcome

```python
#create the model and and disply the production matrix for training data sets
training_model = create_model(best_model)
```

	MAE	MSE	RMSE	R2	RMSLE	MAPE
Fold						
0	235.2947	2.896122e+05	538.1563	0.9720	0.1290	0.0986
1	295.9197	3.212800e+05	566.8157	0.9717	0.1406	0.0998
2	279.8981	3.453254e+05	587.6440	0.9688	0.1247	0.0933
3	306.0329	5.609197e+05	748.9457	0.9628	0.1455	0.0949
4	373.1305	7.282873e+05	853.3975	0.9505	0.1383	0.0975
5	312.7246	5.365766e+05	732.5139	0.9612	0.1247	0.0966
6	333.4934	4.710762e+05	686.3499	0.9722	0.1305	0.0919
7	432.9803	1.080500e+06	1039.4712	0.9430	0.1431	0.1014
8	308.9565	4.816790e+05	694.0310	0.9589	0.1399	0.0964
9	296.5380	4.687770e+05	684.6729	0.9615	0.1300	0.0935
Mean	317.4969	5.284034e+05	713.1998	0.9622	0.1346	0.0964
Std	50.9806	2.209870e+05	140.5326	0.0092	0.0073	0.0029

Fig46; Production matrix after training

Step 11: Model Evaluation

```python
[ ] #model evaluation
    evaluate_model(training_model)
```

```
INFO:logs:Initializing evaluate_model()
INFO:logs:evaluate_model(estimator=LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
              random_state=123, reg_alpha=0.0, reg_lambda=0.0, silent='warn',
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0), fold=None, fit_kwargs=None, plot_kwargs=None, feature_name=None, groups=None, use_tr
```

Plot Type: Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve | Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals

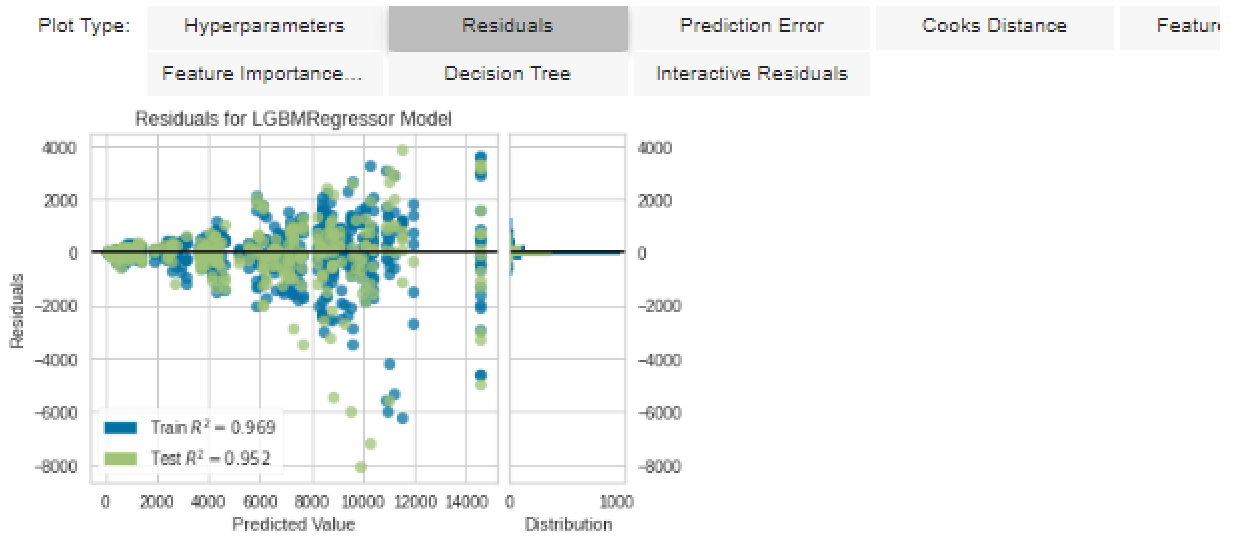
```
INFO:logs:Initializing plot_model()
INFO:logs:plot_model(fold=KFold(n_splits=10, random_state=None, shuffle=False), use_train_data=False, verbose=True, is_in_evaluate=True, display=None,
display_format=None, estimator=LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
```

Plot Type: Hyperparameters | Residuals | Prediction Error | Cooks Distance | Feature Selection | Learning Curve | Manifold Learning | Validation Curve | Feature Importance | Feature Importance... | Decision Tree | Interactive Residuals

	Parameters
boosting_type	gbdt
class_weight	None
colsample_bytree	1.0
importance_type	split
learning_rate	0.1
max_depth	-1
min_child_samples	20
min_child_weight	0.001
min_split_gain	0.0
n_estimators	100
n_jobs	-1
num_leaves	31

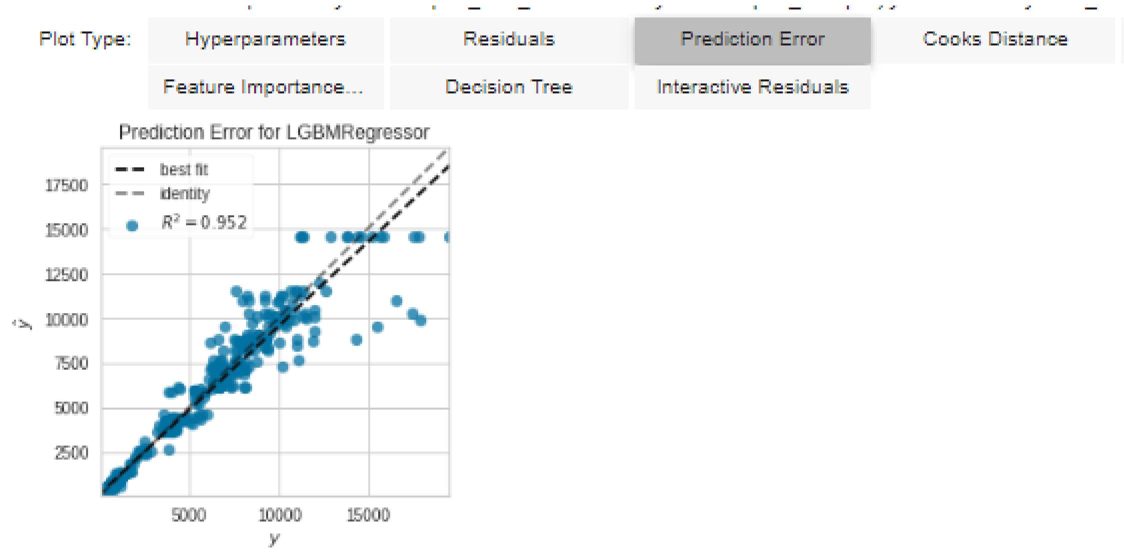
✓ 0s completed at 3:25 AM

Fig 47: Model evaluation



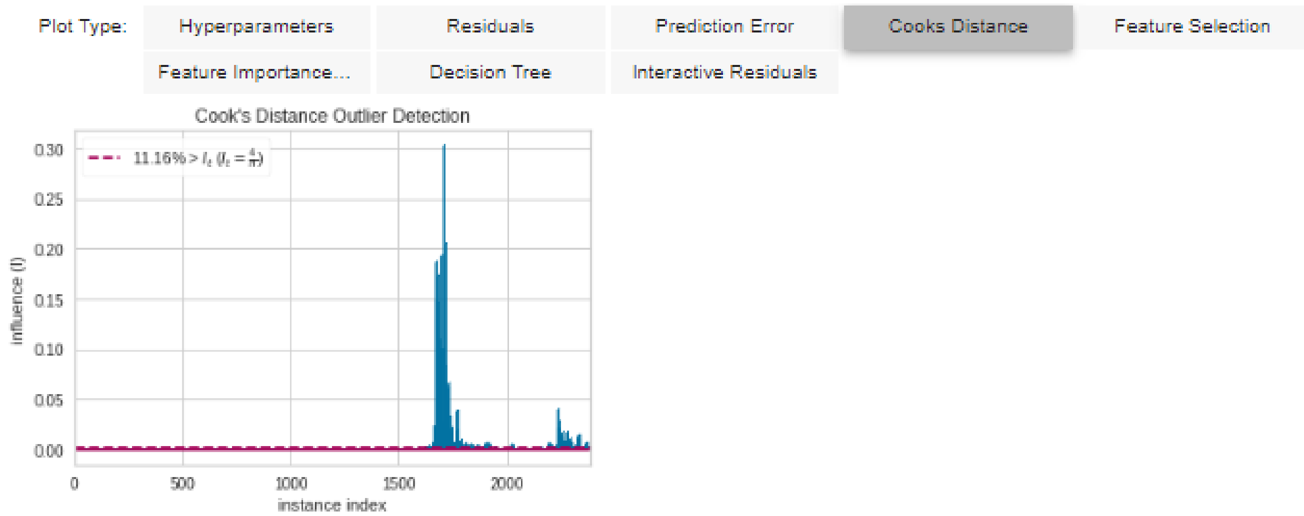

```
INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully completed......................................
```

Fig 48:Plotting Residual



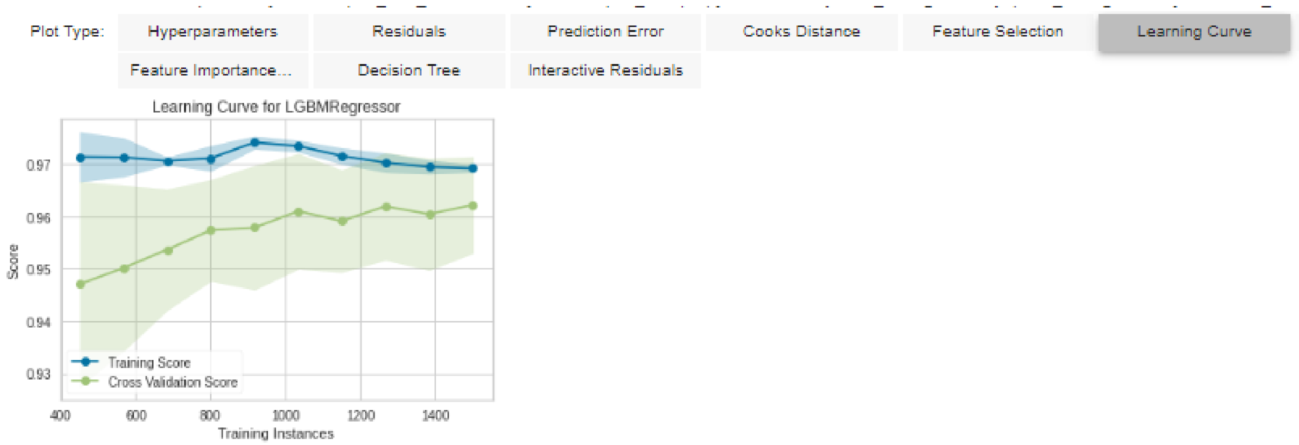

```
INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully completed......................................
```

Fig 49: Plotting prediction error



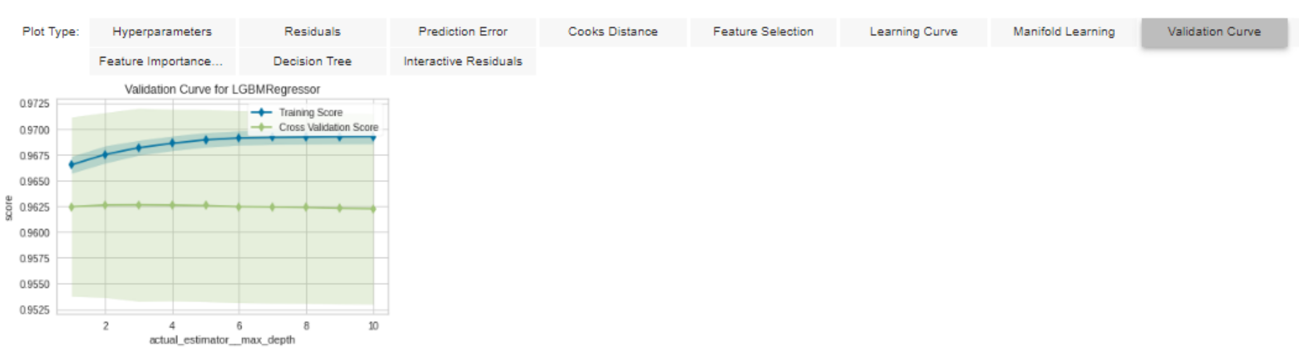

```
INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully completed......................................
```

Fig 50: Plotting Cooks distance




```
INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully completed......................................
```

Fig 51: Leaning curve




```
INFO:logs:Visual Rendered Successfully
INFO:logs:plot_model() succesfully completed......................................
```

Fig 52: Validation Curve

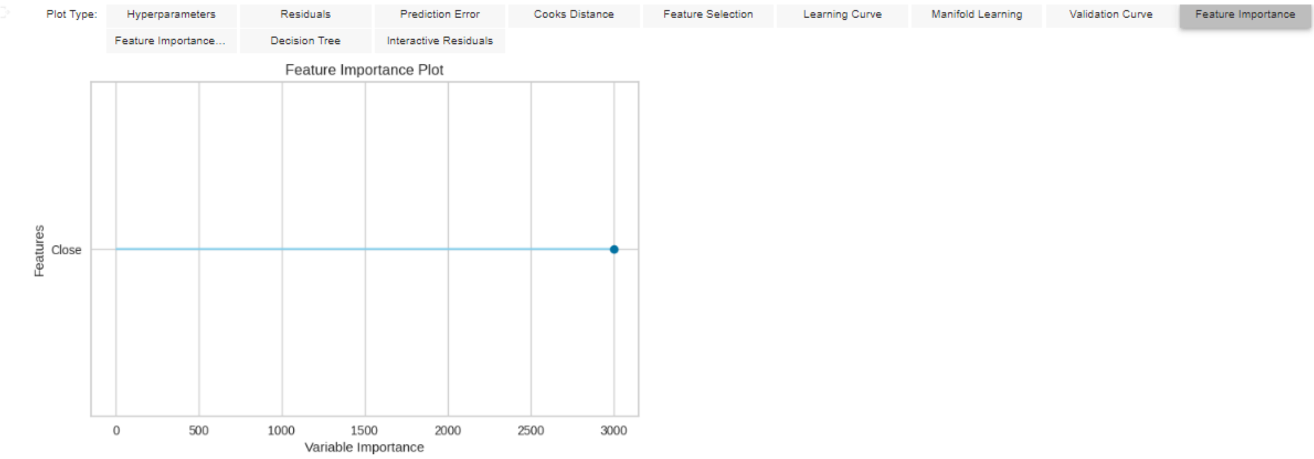


Fig53: Feature importance

Step12: Prediction

```
#prediction
future = predict_model(training_model , data= test_data)

#predicted future price
future
```

```
INFO:logs:Initializing predict_model()
INFO:logs:predict_model(estimator=LGBMRegressor(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
              importance_type='split', learning_rate=0.1, max_depth=-1,
              min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
              n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
              random_state=123, reg_alpha=0.0, reg_lambda=0.0, silent='warn',
              subsample=1.0, subsample_for_bin=200000, subsample_freq=0), probability_threshold=None, encoded_labels=True, drift_report=False, raw_score=False, round=4, verbose=Tru
INFO:logs:Checking exceptions
INFO:logs:Preloading libraries
INFO:logs:Preparing display monitor
```

	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE
0	Light Gradient Boosting Machine	301.3771	522515.8494	722.8526	0.9632	0.1277	0.0902

	Close	New_Price	Label
0	144.539993	112.669998	144.953197
1	139.000000	117.199997	144.953197
2	116.989998	115.242996	123.577929
3	105.209999	115.000000	106.255753
4	97.750000	117.980003	100.305696

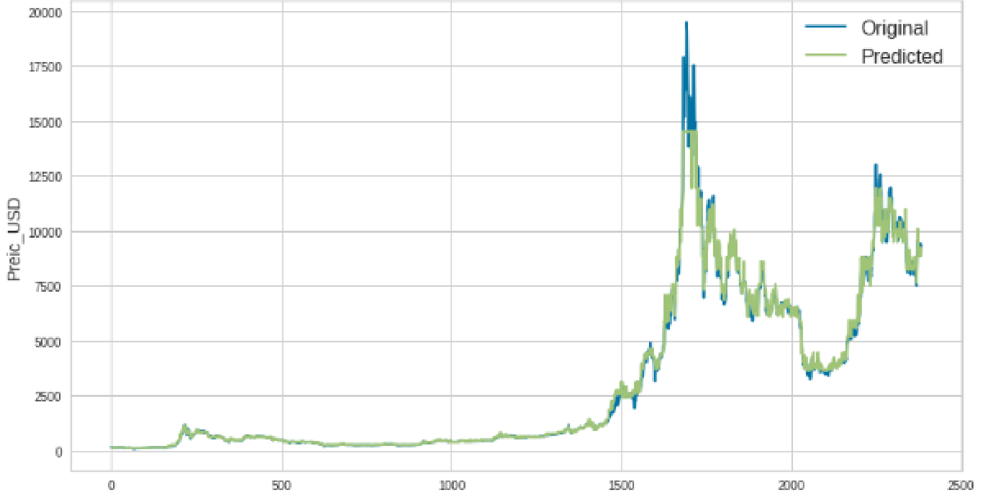
✓ 0s completed at 3:33 AM

Fig 54: Prediction and predicted result

Step 13:Plotting the Result

```
[ ] import matplotlib.pyplot as plt
    fig, ax = plt.subplots(1, figsize=(13, 7))
    ax.plot(future['Close'], label='Original', linewidth=2)
    ax.plot(future['Label'],label = 'Predicted',linewidth=2)
    ax.set_ylabel('Preic_USD' ,fontsize =14)
    ax.set_title('',fontsize =16)
    ax.legend(loc ='best',fontsize =16)
```

```
<matplotlib.legend.Legend at 0x7efdb1a700d0>
```



✓ 0s completed at 3:33 AM

Fig55 : Visualization of Prediction By Pycret Model